Bases de données avancées — **Cours n°1** — GB5 BIMB et M2 BIM, 1^{er} semestre 2018-2019
Cours de PHP — Gilles Bernot

Les notes de cours sont disponibles (avec un peu de retard par rapport au déroulement du cours) à l'adresse web suivante :

`http://www.i3s.unice.fr/~bernot/Enseignement/GB5+BIM_PHP/`

1 Programmation en PHP : le contexte

1.1 Architecture client-serveur

Il est bon de toujours garder en tête l'emplacement et le rôle des différents éléments mis en jeu lorsqu'on veut mettre en place un site web qui permet de gérer une base de données :

ICI UN DESSIN CLIENT-SERVEUR AVEC BD

- Le serveur web, celui sur lequel est installé Apache (i.e. le serveur http à proprement parler), est en général connecté à un serveur SQL (qui gère directement la base de donnée). En fait c'est généralement la même machine qui acceuille les deux.
- En revanche, la machine « client » (i.e. celle de la persone qui veut utiliser le site web) est distante, connectée à internet. On a affaire à une *architecture client-serveur* : le client demande au serveur web de lui envoyer une page écrite en HTML, le serveur calcule cette page, l'envoit à la machine du client et le texte HTML est interprété par le navigateur pour produire un affichage graphique. Le serveur ne garde *a priori* aucune trace de cet échange et le client, d'une page à l'autre, ne garde *a priori* pas trace non plus des pages reçues.
- Les échanges entre client et serveur ne sont donc pas un « dialogue » : chaque envoi ou lecture d'une page HTML est effectué *de novo*. Il faut toujours garder en tête que par défaut le serveur (Apache) ne « suit » pas une session de client : il se contente de réagir en envoyant des pages (HTML/CSS/JavaScript) à la demande de chaque client. Cette architecture client-serveur est cependant assez logique puisque de toute façon le client peut abandonner à tout moment sa session.
- Ce n'est donc en aucun cas HTML qui peut « calculer » ce qui est envoyé au client. On peut donc utiliser un serveur HTML ne contenant que des pages HTML pour concevoir un site web *statique*, où l'enchaînement des pages ne repose que sur les liens. En revanche, si l'on veut adapter les pages envoyées au client en fonction de ses réponses, tout est alors « à (re)calculer » à chaque fois qu'on envoie une page et cela doit se faire au moyen d'un autre langage.
- Enfin c'est le serveur web qui peut interroger la base de donnée, ce n'est pas directement le client. Ceci permet de contrôler l'accès à la base de données et de protéger les données (ne divulguer que ce qui est nécessaire et ne pas permettre de modifications intempestives). Le client peut en effet envoyer des informations erronnées sur l'état d'avancement de sa session. Le serveur doit aussi contrôler cela.

Il y a deux types de gestion de cette architecture client-serveur si l'on veut « calculer dynamiquement » les pages que visualisera le client :

- On peut vouloir déléguer autant que possible *au client* les calculs et la gestion d'une « mémoire » des divers échanges entre le client et le serveur. C'est le choix que l'on fait typiquement en envoyant au navigateur du client non seulement du code HTML mais aussi des *Java-scripts*. Dès lors c'est plutôt le client qui palie l'absence de gestion d'un historique dans l'architecture client-serveur.
- On peut à l'inverse faire faire les calculs et la gestion des sessions par le serveur web. C'est le choix qu'on fait typiquement avec PHP. Ce choix peut cependant être très coûteux en temps de calcul pour le serveur s'il a vraiment beaucoup de clients en même temps.

Dans un contexte où l'on met la priorité au maintien et à la sécurité des données, et où l'on veut éviter d'entrer dans des considérations sécuritaires complexes, on privilégie la seconde solution. C'est généralement le cas en bio-informatique en raison du coût des expériences biologiques qui permettent d'obtenir ces données, des nombreuses autres préoccupations d'un bio-informaticien, et des conséquences graves de travailler sur des données abimées. Il faut en effet être conscient qu'un choix de type Java-script n'offre par défaut

- aucune sécurité au client, parce qu'il ne domine ni les actions faites en son nom sur sa propre machine, ni les informations transmises de la machine client vers le serveur,
- et aucune fiabilité au serveur, parce que le client pourrait tout à fait modifier les fonctionnalités des calculs qu'on lui délègue afin d'obtenir du serveur des informations ou des droits qu'on ne souhaite pas lui donner par défaut.

Le choix de PHP a un coût (temps calcul sur le serveur) car on doit donc programmer au niveau du serveur et en gardant en tête que plusieurs clients peuvent avoir des sessions ouvertes en même temps et préserver malgré cela l'intégrité des données de la base. Il n'en reste pas moins que dans le contexte bio-informatique de ce cours, on adoptera la solution fiable la moins complexe et nous utiliserons donc PHP.

1.2 Mode de fonctionnement d'un serveur PHP

Un serveur PHP fonctionne de la manière suivante :
- Le client demande à voir un fichier du serveur
- Si c'est du HTML/CSS il est envoyé tel quel par le serveur au client. Ces fichiers là se terminent par « `.html` » ou « `.htm` » typiquement.
- Si c'est du PHP, le serveur exécute les commandes PHP avant d'en envoyer le résultat (en HTML/CSS) au client. Les fichiers qui contiennent du PHP se terminent par « `.php` ». PHP sert donc en première approche à calculer du code HTML, que Apache enverra au client.
- Également, il se peut qu'il faille interroger la base de donnée du serveur pour fournir des résultats au client, donc PHP sert aussi à calculer, faire exécuter et récupérer les résultats de requêtes SQL. Il dialogue avec le serveur MySQL de même qu'il dialogue avec le serveur Apache.
- PHP peut également manipuler des fichiers du serveur (comme en Python) mais c'est plus rare car peu sécuritaire.

Le client ne peut donc jamais voir le code PHP mais seulement son résultat lorsque la page HTML calculée est envoyée par le serveur.

Sue le fond, PHP est un langage impératif et l'on retrouvera donc, avec des mots-clefs un peu différents, les mêmes primitives qu'en Python.

1.3 Installation de Apache, MySQL et PHP

A priori l'installation a déjà été effectuée à l'occasion du cours de bases de données (souvent XAMPP). Il faut ensuite regarder dans la documentation où se trouve la racine de l'arborescence Apache dans le système de fichiers du serveur Apache : sous une installation par packages linux, c'est souvent `/var/www/html/` ; sous XAMPP avec Windows, c'est souvent `C:\xampp\htdocs`, qui contient un fichier d'index qui redirige vers `C:\xampp\htdocs\dashboard`. Le plus simple est alors de créer un sous-répertoire de test et de vous en rendre propriétaire. Par exemple si la racine de l'arborescence pour Apache est `/var/www/html/` il suffit de passer root, puis `mkdir /var/www/html/test`, puis `chown moi.mongroupe /var/www/html/test`. Dès lors, votre serveur Apache possèdera l'adresse `http://localhost/test` si vous y accédez en local, et `http://monAdresseExterne/test` si votre ordinateur est connecté en extranet et connu à l'adresse `monAdresseExterne` par les DNS. Sinon son numéro IP peut remplacer `monAdresseExterne`. Une autre possibilité pour gérer en tant que simple utilisateur une arborescence sous Apache est d'avoir votre répertoire sous votre homedirectory et d'y faire un lien symbolique sous la racine de l'arborescence Apache.

> Voir le cours d'*Administration Système et Réseau* pour rediriger tous les `http` en `https` afin d'éviter que les transactions passent en clair sur le réseau...

Par convention, lorsque le client demande à voir un fichier HTML, Apache le fournit au client tel quel. Si c'est un fichier PHP, Apache le fait exécuter par PHP et envoie au client le texte écrit par PHP (voir `print()` plus loin). Si c'est un répertoire, Apache cherche s'il contient un fichier `index.html`, `index.shtml`, `index.php` ou `index.htm` et si oui tout se passe comme si le client avait demandé ce fichier. Attention : si non, Apache montre le listing (cliquable) de *tous* les fils de ce répertoire.

2 Le contrôle en PHP

2.1 Généralités

On rappelle qu'un langage est défini par ses primitives de contrôle et ses structures de données. On connaît déjà le contrôle dans les langages impératifs et il ne diffère que fort peu d'un langage impératif à un autre, donc on va commencer par là.

Tout d'abord, dans un fichier PHP, on peut mettre du texte HTML et il ne sera pas interprété par le langage PHP, donc recopié tel quel vers le serveur Apache. Pour que le texte devienne du code PHP, il faut l'inclure entre « `<?php` » et « `?>` ». Entre ces deux balises, il s'agit d'un programme PHP.

Python est le seul langage largement utilisé où les indentations permettent de délimiter la portée des commandes. Dans tous les autres langages impératifs « classiques », les blocs sont délimités par deux mots-clefs qui en donnent le début et la fin. En PHP se sont les accolades « `{` » et « `}` ». Un bloc d'instructions est donc soit une instruction seule, soit une suite d'instructions entre accolades. Cela ne doit pas faire perdre pour autant les bonnes habitudes d'indentation de marges, qui facilitent grandement la lecture d'un programme !

Enfin, le changement de ligne ne suffit pas à délimiter deux instructions. Il faut *toujours* terminer chaque instruction par un « `;` » . Ce point-virgule est un *terminateur* d'instruction, *pas un séparateur* d'instructions...

Une difficulté majeure en PHP est que si un programme échoue, le serveur Apache se contente par défaut d'afficher une page blanche sans message d'erreur...

2.2 Instructions utiles

Comme PHP doit essentiellement fournir du code HTML au serveur, l'instruction la plus utile est certainement celle qui transmet au serveur une chaîne de caractère pour qu'il la mette dans la page HTML calculée. Il s'agit de `print`. On l'utilise comme suit :

```
print("une chaîne de caractères");
```

Si l'on veut inclure un guillemet, il faut alors le « backslasher ». Pour inclure un backslash il faut donc le doubler.

```
print("Ceci \"\\\" est un unique backslash entre guillemets.");
```

Comme en Python, on peut aussi délimiter les chaînes de caractères avec des simples quotes (`'ceci est une chaîne'`), et les caractères entre les simples quotes autres que le backslash (\) ne sont pas interprétés (voir le type `string` plus loin).

```
print('Il a dit "Bonjour à tous" ce matin.');
```

Contrairement à Python, `print()` ne fait *pas de retour à la ligne*. On peut donc mettre bout à bout des morceaux de texte HTML sur une même ligne avec des `print` successifs. Pour ajouter un retour à la ligne, il faut utiliser le caractère « `\n` », comme pour l'écriture dans un fichier en Python.

L'instruction `echo` fait à peu près la même chose que `print` mais sans parenthèses. Cependant on préférera `print` dans ce cours car `echo` n'est ni une fonction ni une procédure (mauvais style de programmation donc).

Enfin tous les noms de variable doivent nécessairement commencer par « `$` » : `$toto`, `$UneVariableARalonge`, *etc.* Par exemple :

```
<?php
    $monTitre="Conclusion et perspectives";
    print("<h1>$monTitre</h1>\n");
?>
```

et cela affichera le titre de premier niveau en HTML.

Les instructions conditionnelles s'écrivent :

```
    if(..ici un booléen..)
      {
      ..ici des instructions terminées par ;
      }
    elseif(..ici un booléen..)
      {
      ..ici des instructions terminées par ;
      }
    .
    .
```

```
.
else
  {
  ..ici des instructions terminées par ;
  }
```

Les boucles « tant que » s'écrivent :

```
while(..ici un booléen..)
  {
  ..ici des instructions terminées par ;
  }
```

Une primitive bien pratique pour faire un automate en PHP est le `switch` (instruction malheureusement manquante en Python) :

```
switch($etatAutomate)
  {
  case "valeur1":
    ..ici des instructions terminées par ;
    break;
  case "valeur2":
    ..ici des instructions terminées par ;
    break;
  .
  .
  .
  default:
    ..ici des instructions terminées par ;
  }
```

Par défaut, si l'on ne met pas le « `break;` », les instructions continuent en exécutant celles du cas d'après. C'est parfois pratique. On trouve souvent des formes du genre :

```
  .
  .
  case "une premiere valeur":
  case "une seconde valeur":
    ..ici des instructions terminées par ;
    break;
  .
  .
```

qui permettent de partager le même ensemble d'instructions pour deux valeurs différentes dans le `switch`.

Un raccourci « antique » de boucle `while` est la boucle « `for` », qui, ici, n'est pas l'énumération de structure de donnée linéaire comme dans les langages récents, mais seulement un contrôle beaucoup moins élégant directement lié à l'indice d'une boucle. C'est un style typique des « vieux » langages de programmation :

```
for(initialiseIndice; condition; miseAjour_Indice)
  {
  ..ici des instructions terminées par ;
  }
```

est simplement un raccourci pour :

```
initialiseIndice
while(condition)
  {
```

```
  ..ici des instructions terminées par ;
  miseAjour_Indice;
  }
```

L'usage de `for` est souvent plus court qu'un `while` mais pas vraiment plus lisible...

La définition des fonctions utilise le mot clef « `function` » au lieu du « `def` » de Python :

```
function maFonction($x1,$x2,..,$xn)
  {
  ..ici le corps de la fonction..
  }
```

Comme en Python, « `return(valeur);` » permet de renvoyer la valeur résultat de la fonction et sort de la fonction. Une fonction qui ne « `return` » rien est une procédure. Les procédures sont fréquentes en PHP pour produire le code HTML.

PHP offre de plus la possibilité d'avoir des valeurs par défaut pour les derniers arguments d'une fonction, de sorte qu'un utilisateur de la fonction n'est pas obligé de donner tous les arguments. Par exemple avec :

```
function affiche($info1,$info2,$info3='',$montrerNom=TRUE,$montrerAge=FALSE)
  {
  ..corps de la fonction..
  }
```

on peut utiliser la fonction avec au choix 2, 3, 4 ou 5 arguments. Les arguments manquants prendront la valeur par défaut déclarée dans la définition de fonction. Par exemple

```
affiche("un truc","un machin",'',FALSE);
```

exécutera la fonction comme si elle avait été appelée sous la forme

```
affiche("un truc","un machin",'',FALSE,FALSE);
```

On remarque que l'ordre des arguments doit être respecté : on a donc été obligé d'expliciter que le troisième argument est `''` alors que c'est sa valeur par défaut, de façon à donner la valeur du quatrième argument, mais le cinquième a bénéficié du raccourci...

D'un point de vue pratique, il peut arriver qu'on doive définir vraiment beaucoup de fonctions dans un projet PHP. Il est donc utile de placer ces fonctions dans des fichiers PHP différents. On peut par exemple avoir un ou plusieurs fichiers pour le noyau du logiciel, pour la production de pages HTML (Interface Homme-Machine), pour la production de requêtes SQL, *etc.* La page principale (souvent `index.php`) doit alors importer ces fichiers de fonctions pour pouvoir les utiliser (un peu comme des modules en Python). Le début de la page principale contient typiquement :

```
<?php
require("fichierDeFonctions-1.php");
.
.
require("fichierDeFonctions-n.php");
?>
```

et on peut dans la suite du fichier utiliser les fonctions définies dans ces fichiers PHP annexes (le client ne les voit absolument pas, naturellement).

Nota : on peut aussi inclure des fichiers HTML dans un fichier PHP :

```
<?php
include("fichierTruc.html");
?>
```

et même inclure un fichier PHP : dans ce cas, il arrive souvent que l'on ne veuille pas que l'arborescence des `include()` puisse conduire à exécuter deux fois le code du fichier (à cause d'inclusions multiples contenant elles-mêmes un include de ce fichier). On utilise alors `include_once()` :

```
<?php
include("fichierTruc.html");
?>
```

Dans les trois cas, `require()`, `include()` ou `include_once()`, vous pouvez tout à fait calculer dans une variable le nom du fichier que vous allez importer *mais attention*, pour des raisons évidentes de sécurité, ne laissez jamais la possibilité au client d'influer sur le nom des fichiers importés !

Enfin un commentaire peut s'écrire de différentes façons :
- Avec un « `//` » tout ce qui est à sa droite jusqu'à la fin de la ligne est un commentaire. Il en est de même avec « ♯ » mais c'est beaucoup moins utilisé.
- Tout ce qui se trouve entre « `/*` » et « `*/` » est un commentaire.

3 Premières structures de données en PHP

Rappel : une structure de données est définie par le nom de type, l'ensemble des valeurs qu'elle contient et les opérations qui travaillent dessus.

3.1 Les booléens

- Le nom du type est `boolean`.
- Ses valeurs sont `TRUE` et `FALSE` (en majuscules, et bien sûr sans guillemets).
- Les opérations habituelles sur les booléens sont disponibles : `and` s'écrit aussi `&&` ; `or` s'écrit aussi `||` ; la négation s'écrit `!` et on dispose aussi du `xor` (ou exclusif).

En réalité, PHP considère que `TRUE`=`1` et `FALSE`=`0`, rendant ainsi son typage laxiste... assez pour ne pas se rendre compte de ses erreurs de programmation...

3.2 Les nombres entiers

- Le nom du type est `integer`.
- Il représente l'ensemble des entiers relatifs ($\mathbb{Z}$) mais en fait les entiers sont seulement codés sur 32 bits, donc attention aux bornes le cas échéant.
- Les opérations habituelles sur les entiers `+`, `-`, `*` et `%` sont disponibles mais attention `/` est la *division exacte* et le résultat est donc un nombre réel. Les comparaisons $<$, $>$, `<=`, `>=`, `==`, *etc*, comme en Python, renvoient des booléens.

Attention de ne pas commencer un nombre entier non nul par un `0` car il est alors interprété en base 8, et en base 16 s'il commence par `0x`...

Enfin lorsqu'une variable est de type integer, on peut écrire « `$i++` » au lieu de « `$i=$i+1` » et « `$i--` » au lieu de « `$i=$i-1` ».
Plus subtil encore : `$i++` possède une valeur que l'ont peut utiliser dans une expression, celle de `$i` *avant* l'incrémentation. On peut écrire par exemple « `$oldi=$i++` » et `$oldi` vaudra l'ancienne valeur de `$i` alors que `$i` aura été augmenté de 1. Si l'on veut la valeur de `$i` *après* l'incrémentation, il suffit d'écrire « `++$i` ». Par exemple, après l'affectation « `$newi=++$i` », les deux variables `$i` et `$newi` valent 1 de plus que l'ancienne valeur de `$i`. Il en est de même avec la décrémentation : `--$i` au lieu de `$i--`.

3.3 Les nombres réels

- Le nom du type est `double`.
- Il représente l'ensemble des nombres réels ($\mathbb{R}$) en réalité approximés sur 64 bits.
- Les opérations habituelles sur les nombres réels sont disponibles : `+`, `-`, `*`, `/`, les fonctions trigonométriques, *etc*, ainsi que les comparaisons comme en Python.

3.4 Les chaînes de caractères

— Le nom du type est `string`.
— L'ensemble des valeurs est constitué des séquences de caractères comme en Python. On met une chaîne de caractères entre simples quotes : `'toto'` ou entre doubles quotes (`"toto"`) et dans dernier ce cas, un certain nombre d'expressions sont évaluées. Par exemple `"\n"` dénote un passage à la ligne comme en Python alors que ce n'est pas le cas avec `'\n'` qui est considéré littéralement comme ces deux caractères. Plus précisément seuls « \' » pour une apostrophe et « \\ » pour un backslash sont interprétés dans une chaîne entourée de simples quotes (les autres caractères sont laissés tels quels, ce qui est bien pratique lorsque l'on veut écrire une expression régulière).
— L'opération de concaténation s'écrit avec un point « . », la longueur `strlen()`, l'extraction de sous-chaîne `substr()` prend en arguments la chaîne de référence, l'indice de début (qui commence à 0) et la longueur de la sous-chaîne voulue... et il existe de *très nombreuses* fonctions PHP sur les chaînes de caractères puisque PHP est essentiellement utilisé pour la manipulation de chaînes de caractères : impossible à énumérer ici. Parmi les plus utiles : `empty()` dit si la chaîne est vide, `strtolower()` retourne la chaîne toute en minuscules, `strtoupper()` en majuscules, `ucwords()` fournit une majuscule à tous les débuts de mot, `ucfirst()` seulement la première lettre, *etc.* Voir le manuel PHP lorsque le besoin d'une fonction sur les chaînes arrive : elle existe probablement ! Les comparaisons sont la plupart du temps comme en Python *sauf* que si les chaînes comparées ne contiennent que des chiffres, alors c'est la comparaison des nombres qu'ils représentent qui est donnée.

Les *expressions régulières* donnent lieu à plusieurs fonctions en PHP : `ereg()` [`preg_match()` pour les nouvelles versions de PHP] prend en arguments le motif (qui doit être encadré par exemple par des "/") et la chaîne et retourne `TRUE` si la chaîne matche le motif : `preg_match('/motif/',"chaîne");`. La fonction `ereg_replace()` [`preg_replace()` pour les nouvelles versions de PHP] prend en arguments le motif (qui doit là aussi être encadré par exemple par des "/"), une chaîne de remplacement et la chaîne de référence, et retourne la chaîne obtenue à partir de la chaîne de référence en remplaçant la partie qui matche le motif par la chaîne de remplacement : `preg_replace('/motif/',"remplacement","référence");`.