HTML, JavaScript, PHP et MySQL :

Découverte et initiation par l'exemple

Préface

Ce petit ouvrage, composé de cinq documents, à été initialement écrit pour servir de support à une courte formation sous forme de « travaux pratiques à la console ».
Pour sa rédaction, l'auteur a bénéficié de l'expérience acquise, durant six ans, au cours d'une formation du même type nommée « initiation à LaTeX » qui lui avait été confiée à l'IUP Statistique et Informatique Décisionnelle de l'Université Toulouse III.

Le lecteur constatera, tout au long de ces documents, que le langage de cet ouvrage est celui de l'enseignant qui passe et repasse derrière les étudiants travaillant à la console. Tout en reprenant les explications contenues dans les documents, il s'assure que chacun d'eux teste chaque balise et refait chaque exemple. Le lecteur est donc encouragé à travailler de la même manière : lire les documents en traitant des exemples au fur et à mesure de la progression.

Cet ouvrage est écrit pour ceux qui veulent apprendre et comprendre afin de concevoir eux-mêmes des applications de premier niveau. Bien entendu, ces documents ne peuvent traiter que d'une première partie du sujet proposé mais ils permettent d'arriver à la création d'applications simples utilisables (cf. l'exemple du document n°5 qui peut être facilement adapté à des situations plus complexes).

On remarquera que l'on a évité de donner d'emblée des listes d'éléments des langages : l'apprenant progresse en partant d'exemples simples et découvre, par nécessité, les plus élémentaires de ces éléments.

Cet ouvrage peut être abordé par de vrais débutants sur le sujet (il est fait pour eux !) ; une culture scientifique minimale est cependant souhaitable mais il est évident que ceux qui hésitent devant un ouvrage avec un tel titre la possèdent forcément.

Le lecteur qui aura suivi le programme proposé sera bien préparé pour aborder les ouvrages « professionnels » : à chaque étape, il retrouvera souvent un aspect qu'il a déjà abordé et cela l'aidera à absorber et à classer les nouvelles connaissances qu'il acquerra (cf. fin du dernier document).

Yves Soulet

Table des matières

Découverte

HTML : qu'est ce que c'est ? Document I
Introduction I-1
Balises et structure d'un fichier HTML I-1
§ Balises et attributs : première approche I-1
§ Structure d'un fichier HTML I-2
§ Balises "texte" et attributs I-3
§ § Titres I-3
§ § Paragaphes, passages à la ligne, italique et gras I-4
§ § Alignements horizontaux I-4
§ § Taille, police et couleur des caractères I-4
§ § Texte cité ou mis en valeur, marges I-4
§ § Listes numérotées et listes à puces I-5
§ § Séparateurs ou filets horizontaux I-5
§ § Tableaux I-6
§ § Texte préformaté ou verbatim I-6
Insertion d'images I-7
Hypertexte : liens et cibles I-8
§ § Liens locaux I-8
§ § Liens extérieurs I-9
Résumé des balises I-9
Balises div et span I-10
Feuilles de style et propriétés I-11
Résumé des propriétés I-13
Doctype et validation I-16
Remarques I-16

HTML et Java-Script : les formulaires ! Document II
Introduction II-1
Formulaires (présentation sur un exemple) II-2
§ § Balise form II-2
§ § Balise input II-2
Vérification des formulaires II-3
Autres possibilités pour envoyer des données II-7
§ § Menus déroulants II-7
§ § Cases à cocher II-7
§ § Bouton radio II-8
§ § Vérifications pour les menus déroulants, cases à cocher et boutons radio II-8
§ § Mot de passe (remarque) II-11
§ § Texte de plusieurs lignes II-11
§ § Ouverture de fenêtres d'information II-11
Remarques II-12
Processeurs de test des formulaires des exemples choisis II-13

PHP et MySQL : va-et-vient des données ! Document III
Introduction III-1
Création d'une base de données élémentaire III-1
§ Création d'une table III-2
§ Insertion des données dans une table III-2
§ Table de l'exemple choisi III-3
Création du formulaire III-3
Création du processeur III-3
§ Généralités sur les processeurs III-3
§ Récupération des données envoyées par le formulaire III-4
§ Connexion à la base de données III-5
§ Lecture de la table et affichage des données demandées III-6
Remarques III-7
Fichiers complets de l'exemple choisi : formulaire et processeur III-8

Initiation

Travailler avec une base de données Document IV
Introduction IV-1
Domaine choisi : études d'un groupe d'étudiants IV-1
§ Structure d'une base de données IV-1
§ Exemple très élémentaire d'une "vraie" base de données IV-1
Requêtes sur la base de données choisie IV-3
§ Requête (1) IV-3
§ § Formulaire IV-3
§ § Processeur IV-3
§ Requête (2) IV-6
§ § Formulaire IV-6
§ § Processeur IV-6
Remarques IV-8

Gérer la participation à une activité Document V
Introduction V-1
Exemple choisi : gestion des inscriptions à un repas V-1
Création de la table des inscriptions V-1
Outils à créer et leur conception entièrement détaillée V-2
§ Page d'accueil V-2
§ Processeurs d'affichage des inscriptions V-3
§ Formulaire d'inscription V-5
§ Processeur d'inscription V-6
§ Formulaire de recherche d'une inscription V-8
§ Processeur de recherche d'une inscription V-9
§ Formulaire de modification d'une inscription V-11
§ Processeur de modification d'une inscription V-12
§ Traitement de l'apostrophe V-14
§ Autre construction des formulaires V-16
Remarques V-17

HTML : qu'est ce que c'est ?

Introduction

L'arrivée de l'ère de la mondialisation au sens le plus large a nécessité le transfert d'informations scientifiques, juridiques, commerciales, etc. avec la rigueur la plus stricte tout en respectant certaines traditions typographiques, culturelles, etc.

L'information a du être structurée d'une façon universellement acceptée : elle est transmise sous la forme de documents constitués d'un "contenant" et d'un "contenu" étroitement imbriqués l'un dans l'autre. Le contenant décrit la structure du document en éléments d'information ; il est constitué de "balises" qui assurent le formatage ; le contenu est constitué par ces éléments d'information eux-mêmes.

C'est ainsi qu'on a été logiquement amené aux langages dits langages balisés (*markup languages*). Le premier de ces langages a été SGML (*Standard Generalized Markup Language*) : c'est un langage limité à la description de la structure des documents. Mais sa complexité et ses limitations ont rapidement donné lieu à un nouveau langage appelé HTML (*Hyper Text Markup Language*) : il permet par exemple d'écrire des pages d'information très complexes par l'utilisation d'une centaine de balises (pour la version 4.0, version utilisée pour une grande majorité de pages Web) ; en plus il contient toute la machinerie nécessaire pour faire de l'hypertexte. On va commencer par préciser la notion de balise et le lecteur comprendra mieux les quelques mots de cette introduction.

Balises et structure d'un fichier HTML

Balises et attributs : première approche

On donne deux codages du titre de ce document suivis des explications correspondantes. Le premier est :

```
<h1 align="center">HTML : qu'est ce que c'est ?</h1>
```

On remarque que les balises `<h1>` et `</h1>` forment un couple ; la balise "début" : `<h1>` et la balise "fin" : `</h1>` (on dit aussi ouvrante et fermante).
Ces balises peuvent être saisies en capitales ou en minuscules

La balise ouvrante `<h1>` (avec sa balise fermante `</h1>`) crée un titre de niveau 1 (niveau le plus grand ; les possibilités vont de 1 à 6). Ce titre est centré grâce à l'attribut `align` affecté de la valeur `center`.

Le second fait l'identique avec la couleur en plus :

```
<br><br><center>
<b><font size="+3" color="#FF6633">HTML : qu'est ce que c'est ?
</font> </b></center><br><br>
```

Ces exemples illustrent le fait que ces balises peuvent avoir des "attributs" ; les balises `<br>` (retour à la ligne), `<b>` (mise en gras) et `<center>` (centrage) n'ont pas d'attribut ; la balise `<br>` n'a pas de balise fermante correspondante ; ici, elle génère les espaces avant et après le titre qui étaient ajoutés par la balise `<h1>`. Par contre les balises `<h1...>` et `<font...>` ont des attributs, chacun d'eux affecté d'une valeur. Ces attributs concernent :

1. le positionnement horizontal du texte, attribut `align`, valeur `center left right` ;
2. la taille des caractères, attribut `size`, valeur : nombre variant de 1 à 7 ou de -3 à +3 par pas de 1 ;
3. le choix de la fonte, attribut `face`, valeur `Arial`, mais il y en a beaucoup d'autres : `Times ...` ; si on en choisit plusieurs (séparées par des virgules), le navigateur client choisit suivant l'ordre donné et en fonction de ses disponibilités ;
4. la couleur, attribut `color`, valeur codée en codage RGB : chaque couleur est définie par 3 nombres (de 0 à 255) écrits en hexadécimal (il y a d'autres possibilités, en particulier des couleurs prédéfinies, mais il faut qu'elles soient également définies dans le navigateur client !). *Il faut toujours bien se rappeler que l'on écrit du code HTML qui sera interprété par le navigateur client*. On donnera d'autres explications sur les couleurs plus loin.

Ce qui précède est très général. Par exemple, voilà des attributs de couleur pour la balise <body> (balise fondamentale comme on va le voir dans l'étude de la structure d'un fichier HTML) :

- `bgcolor=...` pour colorer le fond,
- `text=...` pour colorer le texte,
- `link=...` pour colorer les liens hypertexte,
- `alink` et `vlink` pour colorer les liens actifs et les liens déjà visités.

On peut tout faire avec ces balises et leurs attributs : il suffit d'avoir leur liste avec les explications correspondantes.

Certaines balises de fin peuvent être omises mais ce n'est pas recommandé par le W3C (Consortium W3C, on en reparlera plus loin) dont les directives gèrent rigoureusement le côté syntaxe du langage du Web. Par contre la balise `</p>`, balise fermante de la balise `<p>` (balise débutant un paragraphe), peut être omise si elle est suivie de la balise `<p>` et ceci en toute sécurité. Quelques balises n'ont pas de balise fin, par exemple `<br>` `<input...>` (qui sera introduite plus loin), etc. Il y a nécessairement une balise de fin lorsqu'il faut délimiter le champ d'action de la balise de début.

Une remarque s'impose avant de continuer : il est bien évident que si l'on veut écrire les caractères spécifiques utilisés pour les balises, il faut utiliser la convention établie par les créateurs du langage :

< > & s'obtiennent avec `< > &`

Avec un clavier d'un pays européen, on peut ainsi écrire tous les caractères du codage Wondows-1252. Par exemple pour écrire acò avec un clavier français, on écrira `acò` ou `ac&#242` (où `ograve` est le nom informatique du caractère ò et où `242` est son code décimal Windows-1252 ; il faut pour cela disposer d'une table des caractères).

Structure d'un document HTML

Avant de continuer sur les balises, on donne la structure type d'un document HTML :

`<!DOCTYPE...>` C'est un entête indispensable pour un site Web. Il contient des informations sur le document, notamment la version de HTLM utilisée (cf. fin du doc.). Il est inutile pour un document interne (exercices, etc.) ainsi que pour les pages internes d'un site.

`<html>` Début du fichier HTML.

`<head>` Début de l'entête.

`<meta http-equiv="Content-Type" content="text-html;charset=windows-1252">`
contenant le codage utilisé localement (important pour l'ordinateur client).
`<meta http-equiv="Content-Language" content="fr">`
annonce la langue du document (également important pour le navigateur client).
`<meta name="Description" content="`Description du contenu`">`
`<meta name="Keywords" content="`Mots clés séparés par des virgules`">`
contenant les mots-clés pour le référencement par les moteurs de recherche.
Il y a d'autres balises `meta` : auteur, Copyright, etc.

`<title>`Titre`</title>` Cet élément peut être considédéré comme un titre logique ; Il s'affiche dans la barre supérieure du navigateur.

On y place aussi les définitions des fonctions JavaScript, les redéfinitions des balises et les définitions de variantes de balises ainsi que les appels aux feuilles de style.

`</head>` Fin de l'entête.

`<body>` Début du corps du document.

Ici le contenu du document !

`</body>` Fin du corps du document.

`</html>` Fin du fichier HTML.

A partir d'ici, le lecteur est invité à faire un fichier HTML, d1.htm par exemple (extension .htm obligatoire), pour tester tout ce qui va être exposé. Pour cela, on utilisera un éditeur de texte (Bloc-notes, Technic-Center, etc.).

Pour afficher le résultat produit par le fichier d1.htm, on utilisera le navigateur habituel dans sa configuration "Travail hors connexion".
Commentaires et parties de code non utilisées sont à placer entre les balises `<!--` et `-->`.

Balises "Texte" et attributs

Il faut noter que certains attributs sont utilisables avec plusieurs balises comme on le verra dans la liste suivante. Dans la suite, sauf exception pour raisons pédagogiques, on citera les balises sans les crochets ouvrants et fermants et la balise fermante ne sera pas mentionnée ; au contraire, c'est l'absence de balise fermante qui sera signalée.

Titres

On dispose des balises `h1` à `h6` : (1 pour les gros titres, jusqu'à 6 pour les plus petits) ; les titres sont à gauche par défaut ; on peut les centrer (cf. le premier exemple de codage) ou les placer à droite avec la valeur `right` affectée à l'attribut `align`. On peut toujours faire des titres en utilisant une combinaison de balises et d'attributs bien choisis (cf. deuxième codage du titre avec la balise obsolète `<center>`).

Paragraphes

La balise `p` est à placer en début de paragraphe. On a vu que la balise fermante correspondante n'est pas nécessaire si elle est suivie d'une nouvelle balise `p`.

Passages à la ligne

La balise `br` provoque un passage à la ligne ; elle peut être uilisée pour générer des espaces verticaux (cf. le deuxième exemple de codage). Il n'y a pas de balise fermante correspondante.

Italique et gras

On a les balises `i` pour l'italique, `b` et pour le gras ; le gras italique s'obtient avec ces deux couples "proprement entrelacés" : `<b><i> ... </i></b>`.

Alignements horizontaux

On dispose de l'attribut `align="left" center right` dans les balises `h1` à `h6` mais aussi dans la balise `p` et dans les balises pour les tableaux (balise `table` et toutes les balises de ligne et de cellule (cf. paragraphe consacré aux tableaux).

Taille, police et couleur des caractères

On dispose de la balise font qui a les attributs :

- `size="1" ... "7"` (7 donnant la plus grande taille) ou `="-3" ... "+3"` : les valeurs négatives donnant des tailles plus petites que la taille courante qui correspond à la valeur 0.
- `face="nom-informatique-de-la-fonte"`. On peut en mettre plusieurs séparées par des virgules : le navigateur client les prend dans l'ordre en fonction de ses possibilités. Les fontes "baton" sont plus lisibles sur les écrans que les fontes avec "pleins" et "déliés" ; `Arial` (utilisée pour ce document) est sur tous les navigateurs clients et est bien lisible car elle a un bel "œil".
- `color="couleur-codée-en-RGB"` ou `="couleur-prédéfinie"` ; le codage RGB (*Red,Green,Blue*) est assuré par trois nombres hexadécimaux entre 00 et FF (soit entre 0 et 255 en décimal) ; exemple : `color="#FF0000"` (donne le rouge).
  Bien entendu, il faut disposer d'une table de couleur ou utiliser la liste des 16 couleurs prédéfinies : `black maroon green olive navy purple teal silver gray red lime yellow blue fuchia aqua white` ; exemple `color="red"`

Texte cité ou mis en valeur

> On dispose de la balise `blockquote` qui va afficher du texte avec un retrait à gauche et à droite ; par exemple, la saisie du présent paragraphe est :

```
<blockquote>On dispose...suivante.</blockquote>
```

Marges

La balise `body` accepte les attributs suivants où les dimensions sont exprimées en pixels : `leftmargin="10"` ainsi que `rightmargin topmargin bottomargin`.

Cela permet une lecture plus agréable (on n'imagine pas un livre sans marges !).

Listes numérotées et listes à puces

Le plus simple est d'écrire un exemple de liste numérotée ; le couple `<ol> </ol>` marque le début et la fin de liste et les couples `<li> </li>` marque le début et la fin des éléments de la liste :

A. premier élément,
    a. premier sous-élément,
    b. deuxième sous-élément,
B. deuxième élément.

Il provient de la saisie :

```
<ol type="A">
  <li>premier élément</li>
    <ol type="a">
      <li>premier sous-élément</li>
      <li>deuxième sous-élément</li>
    </ol>
  </li>
  <li>deuxième élément</li>
</ol>
```

Les blancs en début de ligne n'interviennent pas sur le résultat, *mais il est vivement conseillé de les introduire* ; ils servent à aligner verticalement les balises de même rang sur le fichier de saisie : cela facilite la saisie et surtout la relecture pour correction.

L'attribut `type` de la balise `ol` permet de choisir la numérotation :
- `A` pour la numérotation alphabétique en majuscules et `a` en minuscules,
- `I` pour la numérotation en chiffres romain majuscules et `i` en minuscules,
- `1` pour la numérotation en chiffres arabes (défaut).

On remarque sur l'exemple que l'on peut mettre les listes en cascade en choisissant des valeurs différentes pour l'attribut `type`. Il faut dans ce cas veiller à l'entrelacement correct des différents niveaux.

Les listes à puces se saisissent de la même manière, il suffit de changer la balise `ol` par la balise `ul`. On redonne l'exemple précédent sur lequel on remarquera que, par défaut, les dessins des puces changent à chaque niveau :

- premier élément,
    - premier sous-élément,
    - deuxième sous-élément,
- deuxième élément.

Séparateurs ou filets horizontaux

La balise `hr`, qui n'a pas de balise fermante correspondante, trace des "traits" horizontaux. Elle accepte les attributs suivants :
- `align color` déja vus,
- `size="5"` donne l'épaisseur du trait en pixels,
- `width="100" 60%` où la première valeur est exprimée en pixels et la deuxième en pourcentage de la largeur de la fenêtre du navigateur ; c'est cette possibilité qui permet d'avoir la meilleure reproduction de la mise en page sur l'ordinateur client.

Tableaux

Comme pour les listes, on donne d'abord un exemple que l'on expliquera ensuite :

Cellule 1	Cellule 2	Cellule 3
Texte sur 3 colonnes		
Cellule 4	Texte en deux lignes	Texte en haut
	Cellule 5	Cellule 6

Ce tableau provient de la saisie suivante :

```
<table align="center" border="5" width="50%">
   <tr>
      <>Cellule 1</th><th>Cellule 2</th><th align="right">Cellule 3</th>
   </tr><tr>
      <td colspan=3 align="center"><i>Texte sur 3 colonnes</i></td>
   </tr><tr>
      <td rowspan=2>Cellule 4</td><td>Texte en deux<br>lignes</td>
      <td valign="top">Texte en haut</td>
   </tr><tr>
      <td>Cellule 5</td> <td bgcolor="yellow" bordercolor="red">
      <font color="green">Cellule6</font></td>
   </tr>
</table>
```

Les balises et attibuts couramment utilisés sont (cf. tableau ci-dessus et tableau suivant) :
- la balise principale `table`
- l'attribut `align` pour placer horizontalement le tableau : valeur `left center right`
- l'attribut `border` pour faire un cadre (autour du tableau) d'épaisseur exprimée en pixels,
- l'attribut `bordercolor` pour en choisir la couleur,
- les attributs `width` ou `height` pour donner une largeur ou une hauteur du tableau (par exemple `width="60%"` où la largeur est exprimée en pourcentage de la largeur de la fenêtre du navigateur ; on peut aussi exprimer ces dimensions en pixels sous la forme `width="300"`),
- les balises de lignes `tr`
- les balises de cellules ordinaires `td`
- les balises de cellules de titre `th` (texte centré et en gras par défaut),
- l'attribut `align` pour la position horizontale du texte dans les cellules,
- l'attribut `colspand=3` pour écrire sur plusieurs colonnes,
- l'attribut `rowspand=2` pour écrire sur plusieurs lignes,
- l'attribut `valign="top" middle ...` pour la position verticale du texte dans les cellules,
- l'attribut `bgcolor` pour colorer le fond du tableau, des lignes ou des cellules.

On remarque qu'un attribut affecté d'une valeur dans la balise de ligne est actif dans toutes les cellules de la ligne ; cependant, on peut faire une autre affectation dans une balise de cellule de cette ligne. Si l'on souhaite qu'une cellule vide ait quand même une bordure, il faut y placer le caractère espace obtenu avec ` ` ou ` `. Il reste encore beaucoup de possibilités pour affiner la présentation, en particulier le positionnement du texte dans les cellules et les dimensions des cellules ... mais on en reste là.

Texte préformaté ou verbatim

La balise `pre` à la propriété d'afficher les caractères positionnés exactement comme dans

la saisie, avec une fonte à espacement constant ; les espaces et les aller-à-la-ligne sont respectés ; les caractères spécifiques restent actifs et doivent être affichés comme on l'a déjà vu. Cela peut être utile dans certains cas ; par exemple, la reproduction de code informatique (voir ci-dessus le codage des liste numérotées et du tableau). Cette balise réserve une surprise qui n'est visible que sur l'aperçu avant impression : si on saisit des lignes trop longues, le navigateur réduit la taille de *toutes* les fontes *automatiquement*.

Insertion d'images

Il est clair que cette fonction du langage HTML est très importante dans la mesure où l'information par l'image prend de plus en plus d'importance. On ne va donner ici que quelques éléments permettant toutefois d'obtenir de bons résultats quoique très modestes.

L'intégration d'une image (formats jpg (jpeg) et gif) se fait grâce à la balise `img`, qui n'a pas de balise fermante ; la syntaxe correspondante est :

```
<img src="nom-du-fichier" attribut="valeur"...>
```

dont les attributs les plus utilisés sont :
- `src="nom-du-fichier-image"` ici nom-du-fichier désigne le chemin complet, écrit avec des slash comme sous Unix ou Linux (même si l'on travaille sous Windows) ;
- `alt="Texte de remplacement"` est utilisé par le navigateur client s'il ne peut pas afficher l'image ou s'il est configuré pour ne pas l'afficher ;
- `width="80"` ou `height="50"` pour fixer les dimensions de l'image en pixels ;
- `hspace="20"` permet de placer un espace horizontal en pixels avant et après l'image ;
- `align="top" middle bottom baseline` pour positionner verticalement l'image par rapport au texte qui se trouve avant ;
- `vspace="20"` permet de placer un espace vertical en pixels avant et après l'image.

Voici un exemple :

La culture du bégonia est très difficile et demande beaucoup de travail.

Il faut d'abord penser que cette plante n'aime pas être en plein soleil toute la journée. Il faut aussi lui réserver un endroit frais, facile à arroser,...

Il s'agit d'une présentation simple où l'image est placée dans une cellule d'un tableau et le commentaire dans la cellule suivante (présentation valable dans le cas d'un commentaire long, sinon on peut mettre le commentaire dans la cellule située sous l'image) ; le tout est centré dans la page. La justification du texte n'est pas ici très heureuse mais la balise `p` a

été redéfinie en y incluant cette propriété (on verra plus loin comment gérer ces problèmes). Ce tableau résulte de la saisie suivante :

```
<table align="center" border="10" bordercolor=#238E23 width="80%"
cellspacing="5" cellpadding="20"><tr>
<td bgcolor=#238E23><img src="begonia.jpg"
  height="200" alt="Image"></td>
<td bgcolor=#F6F694><p>La...</p> <p>Il...</p</td>
</tr></table>
```

On remarque que l'on a utilisé ci-dessus deux attributs de la balise `table` concernant les cellules:
- `cellspacing` pour fixer l'espace entre les cellules (en pixels),
- `cellpadding` pour fixer l'espace autour des éléments dans les cellules (en pixels).

Le rôle des valeurs de l'attribut `align` dans la balise `img` sont :
- `top` : alignement du haut de l'image avec le haut de la ligne courante,
- `middle` : alignement de la mi-hauteur de l'image avec la ligne de base courante,
- `baseline` : alignement du bas de l'image avec la ligne de base courante,
- `bottom` : alignement du bas de l'image avec la bas de la ligne courante.

On ne peut utiliser valablement que les cas où il n'y a qu'une de ces valeurs par ligne ; voilà les deux exemples les plus intéressants, `middle` et `baseline` ; en outre, la deuxième balise `img` de la première ligne de l'exemple ci-dessous montre l'espace supplémentaire introduit des deux côtés de l'image par `hspace="20"`. L'effet de `vspace="20"` serait exactement le même mais dans le sens vertical.

AA BB CC DD

AA BB CC DD

A noter ici que l'attribut `src` peut aller chercher l'image sur un quelconque site en écrivant :

```
... src=http://addresse-du-site/chemin-sur-le-serveur/nom-du-fichier ...
```

Hypertexte : liens et cibles

On arrive maintenant aux liens qui permettent la navigation hypertexte dans toutes les URL, *Uniform Ressource Locator*, existant dans le monde. La syntaxe des liens utilise la balise `a` de la manière suivante :

```
<a href="cible">texte du lien</a>
```

où `cible` représente ce que l'on veut atteindre et où "texte du lien" est le mot (ou l'élément du document) sur lequel il faut cliquer pour afficher la cible ; cela peut être aussi une image et même une partie d'une image, mais le but n'est pas ici de recopier les livres .

Liens locaux

Pour les liens locaux, la cible peut être :

- le nom du fichier à atteindre : `nom-de-fichier`
- le nom de l'ancre à atteindre dans le même fichier écrite sous la forme : `#nom-de-l'ancre`
- l'adresse complète de l'ancre dans un autre fichier écrite sous la forme : `nom-de-fichier#nom-de-l'ancre`

Le `nom-de-fichier` comprend, comme pour les images, le chemin écrit avec des slash (et non des backslash). La syntaxe de pose de l'ancre dans un fichier est :

```
<a name="nom-de-l'ancre">texte de l'ancre</a>
```

où "texte de l'ancre" est le mot (ou l'élément de document) qui sera atteint par le clic sur le "texte du lien". En général, cet élément sera affiché à la première ligne de l'écran.

Voilà un exemple de lien interne au présent fichier ; le "texte de l'ancre" est la chaîne de caractères "listes à puces".

Liens extérieurs

Pour un lien vers un site, la cible est l'adresse de la page ; il est créé grâce à la syntaxe :

```
<a href="http://yves.soulet.free.fr">site</a>
```

Mais on peut aussi faire un lien direct vers une page d'un site en écrivant :

```
<a href="http://yves.soulet.free.fr
        /pub/petit_manuel_pour_tikz_2012.pdf">page</a>
```

où l'on a précisé le chemin du fichier de la page sur le serveur. On peut même faire un lien vers une ancre située dans un fichier d'un serveur : il suffit de rajouter `#nom-de-l'ancre` après `nom-du-fichier` si on connaît ce nom bien entendu !

Pour un lien vers une adresse courriel, la cible est l'adresse courriel ; le lien est créé grâce à la syntaxe :

```
<a href="mailto:yves.soulet@free.fr">courriel</a>
```

Dans le premier cas le clic sur "site" affiche le portail du site.
Dans le deuxième cas le clic sur "page" affiche la page voulue du site.
Dans le troisième cas le clic sur "courriel" ouvre l'utilitaire de courrier.
Dans les trois cas le retour se fait en cliquant sur 'Page précédente'.

Résumé des balises

On reprend les différentes balises en ne rappelant que certains détails, les plus importants ou ceux sommairement introduits pour ne pas alourdir le texte et dérouter le lecteur.

- Balises globales :
  `html`
  `head` (avec balises secondaires)
  - `ttitle` pour le titre logique
  - `meta Description Keywords ...` (cf. début de doc.)
  - `script language="JavaScript" ...` fonctions JavaScript (cf. doc. suivant)

    - `style type="text/css" ...` redéfinitions et définitions des variantes de balises (cf. fin de doc.)
    - `link type="text/css" ...` feuilles de style externes (cf. fin de doc.)

  `body bgcolor=... background="nom-du-fichier-image"` pour le fond de page, ainsi que `topmargin=...` pour les marges et idem avec `bottom left right`
  `basefont size=... color=... face=...` fonte par défaut dans tout le document.

- Balises de bloc de document :
  `h1 ... H6 align=...` pour les titres
  `blockquote` pour paragraphe avec retrait
  `hr align=... width=... size=...` pour les tirets horizontaux
  `p align=...` pour les paragraphes
  `pre` pour du texte verbatim.
- Balises de lignes :
  `br` retour à la ligne
  `font size=... color=... face=...` pour le choix des fontes
  `sup sub` mise en indice supérieur et inférieur
  `b i` pour mettre en gras et en talique
  `small big` pour taille un peu plus petite et un peu plus grande
  `tt t` pour fonte machine à écrire et pour soulignement.
- Balise de listes :
  `ol type=...` pour énumérations
  `ul` pour listes à puces
  `li` pour éléments des listes ci-dessus
  `dl` pour listes de définitions (peu utilisé, même syntaxe que pour les listes)
  `dt` pour terme à définir
  `dd` pour texte de la définition.
- Balises de tableaux :
  `table` avec les attributs `align width height bgcolor bordercolor cellpading cellspacing` pour les tableaux
  `tr` pour les lignes
  `td` avec les attributs `colspan rowspan align valign bgcolor bodercolor` pour les cellules (certains pouvant être affectés aux balises de ligne)
  `th` pour les cellules en tête des colonnes ou des lignes
  `caption align="top" "bottom"` ajoute une légende en dessus ou en dessous du tableau (saisie comme pour une ligne : `<caption>`Légende`</caption>`).
- img scr="nom-fichier-image" et les attributs `alt width height hspace align border` pour l'insertion d'images
  `a href=...` pour les liens intérieurs et extérieurs ou vers une adresse courriel
  `a name=...` pour définir une ancre dans un fichier.

Comme on l'a déjà précisé, il n'y a ci-dessus que les balises les plus courantes et les attributs les plus utilisés : il n'est pas question de recopier les livres de référence, mais de proposer au lecteur un premier ensemble d'éléments pour commencer l'écriture de pages HTML.

Balises div et span

A partir de la version 4.0 de HTML, on dispose de deux nouvelles balises très puissantes dans le sens qu'elles permettent d'enrichir aisément la présentation des documents. Tout au début du présent document, on a présenté deux possibilités pour coder le titre du document. Il est évident que la deuxième est conçue à partir de balises de niveau plus

profond que le niveau de celles utilisées dans la première ; dans cet esprit, on a encore une troisième possibilité de codage utilisant les nouvelles balises :

- `<div>` est une balise qui crée un élément du document du type bloc ; cette balise prend un attribut nommé style dont la valeur est formée de couples du type propriété:valeur ; cet attribut style définit le formatage de l'élément du type bloc ainsi créé ;
- `<span>` est une balise semblable à la précédente excepté qu'elle crée un élément en-ligne au lieu d'un élément bloc.

Alors que la syntaxe d'utilisation des balises déjà citées est de la forme :

```
<balise attribut="valeur" attribut="valeur" ... > ... </balise>
```

celle de ces deux nouvelles balises est de la forme (cas de la balise `div`) :

```
<div style="propriété:valeur;propriété:valeur; ... "> ... </div>
```

Cette troisième possibilité, annoncée plus haut, pour coder le titre du document avec la balise `div` s'écrit alors :

```
<div style="text-align:center; font-family:Arial; font-size:20pt
   font-weight:bold;color:green;padding-top:18pt;
   padding-bottom:5pt">HTML : qu'est ce que c'est ?</div>
```

et donne :

HTML : qu'est ce que c'est ?

où l'on remarque que l'on a introduit un espace avant le titre et un espace après avec les propriétés `padding-top` et `padding-bottom` exprimés en points.

On donne un exemple d'utilisation de la balise `span` pour afficher quelques mots en petites capitales sur un font gris; on écrit :

```
Quelques <span style="background-color:#CCCCCC;
font-variant:small-caps">mot sur fond gris et en petites
capitales</span> par exemple.
```

et on obtient :

Quelques mot sur fond gris et en petites capitales par exemple.

Les propriétés disponibles sont en grand nombre ; on en donnera une liste suffisante pour pouvoir produire des pages Web satisfaisantes à la fin de la section suivante.

Feuilles de style et propriétés

On vient de voir à la section précédente consacrée aux balises `div` et `span`, une autre forme de syntaxe pour le formatage de parties de document ; ce formatage est assuré par un attribut `style` défini par :

```
... style="propriété:valeur;propriété:valeur; ..." ...
```

Cette nouvelle façon de définir les outils de formatage constitue la base de ce qu'on appelle les feuilles de style (CSS : *Cascading Style Sheets*) ; c'est cette méthode qui est maintenant généralement utilisée pour écrire les pages Web. Elle repose sur la possibilité de redéfinir et de créer des variantes, nommées *classes*, des balises déjà définies.

Pour illustrer tout cela, on considère la balise `em` qui permet de mettre en évidence (c'est-à-dire en italique) un mot ou un groupe de mots ; par exemple :

```
<em>Mots en italique</em>
```

donne : *Mots en italique*.

En reprenant la syntaxe utilisée pour l'exemple concernant la balise `div`, on trouve que :

```
<em style="font-weight:bold">Mots en italique gras</em>
```

donne : ***Mots en italique gras*** et que :

```
<em style="font-style:normal;font-weight:bold">Mots en gras droit</em>
```

donne : **Mots en gras droit**.

Ces opérations correspondent à une sorte de "redéfinition locale" de la balise `em`.
A la première ligne, l'effet de l'italique initial reste et on a ajouté l'effet de gras.
A la deuxième ligne, l'effet de l'italique initial est annihilé par le choix du droit, `normal`, et le gras est ajouté comme précédemment.
Ceci montre bien que, dans ce type de redéfinitions, les propriétés ajoutés prévalent sur les propriétés initiales.

Ce type de redéfinition locale tel qu'il est présenté ci-dessus est un peu lourd à utiliser ; mais il a une version globale qui se place *une fois pour toutes* dans l'entête du fichier du document. Voilà la syntaxe complète des deux redéfinition globales correspondant aux deux redéfinitions locales de la balise `em` (à placer dans l'entête (entre `<head>` et `</head>`) :

```
<style type="text/css">
em{font-weight:bold} ou em{font-style:normal;font-weight:bold}
</style>
```

On peut donc ainsi redéfinir toutes les balises ! Par exemple, une redéfinition possible de h1 peut être faite avec le code suivant (placé dans l'entête entre `<style type="text/css">` et `</style>` comme expliqué ci-dessus) :

```
h1{text-align:center;font-family:Arial; font-size:20pt;font-weight:bold;
color:green;padding-top:18pt;padding-bottom:5pt}
```

Ces redéfinitions effacent complètement les définitions initiales. Pour éviter cela, on a la deuxième possibilité annoncée : la définition de variantes d'une balise tout en conservant la définition initiale : on dit que ces variantes sont des *classes*. Les classes sont nommées par `nom-de-la-balise.nom-de-la-classe`. Les définitions suivantes (toujours placées dans l'entête comme expliqué ci-dessus) sont des variantes des balises `em`, `h1` et `a` :

```
em.a{font-style:normal;text-decoration:line-trough}
h1.a {text-align:center;color:green;font-family:times}
```

```
a.l1 {font-weight:bold}
a.l2 {font-style:italic}
```

permettent d'obtenir :

~~Mots en droit barrés~~ (*Mots en italique pour comparaison*)

HTML : qu'est ce que c'est ? (en Times)

Lien en gras *Lien en italique* (Lien normal pour comparaison)

avec la saisie :

```
<em class="a">Mots en droit barrés</em>
(<em>Mots en italique normal pour comparaison</em>)
<h1 class="a">HTML : qu'es acò ? (en Times)</h1>
<a class="l1" href="#motcible">Lien en gras</a>
<a class="l2" href="#motcible">Lien en italique</a>
(<a href="#motcible">Lien normal pour comparaison</a>)
```

Les deux dernières variantes peuvent être utilisées pour distinguer les liens vers des URL de différentes provenances.

Ces quelques exemples de redéfinitions de balises et de définitions de variantes de balises sont suffisants pour être en mesure d'en réaliser d'autres afin de produire aisément des pages Web de bonne qualité. Ensuite, pour aller plus loin, il faudra se plonger dans les ouvrages avancés, ce sera plus lourd mais le lecteur y sera bien préparé. On rappelle les deux syntaxes correspondantes (`xx` désigne une balise):

- Ou bien on écrit dans l'entête la redéfinition d'une balise ou la définition d'une variante d'une balise sous la forme :

  ```
  <style type="text/css">
  xx ou xx.cl {suite de couples propriété:valeur
    séparés par un point-virgule}
  </style>
  ```

  et on écrit dans le corps du document :

  ```
  <xx>texte...</xx> ou <xx classe="cl">texte...</xx>
  ```

- Ou bien on écrit directement dans le corps du document :

  ```
  <xx style="suite de couples propriété:valeur
    séparés par un point-virgule">...</xx>
  ```

Résumé des propriétés

Comme déjà annoncé, on donne une liste suffisante de propriétés pour pouvoir produire aisément des pages Web satisfaisantes (`nn` désigne un nombre).

- Fontes :
  `font-family:Arial Times Helvetica ...` choix de la fonte

`font-size:nn` taille en pt, cm, px ou % ; 200% : taille de base doublée
`font-style:normal italic`
`font-variant:normal small-caps`
`font-weight:lighter normal bold bolder`
Il y a une abréviation pour les suites de choix : `font-aa:rr; font-bb:ss; ...` peut s'écrit `font:rr ss ...` ; exemple : `font:italic bold 12pt Arial`

- Texte :
  `text-align:left right center justify` disposition des lignes
  `text-indent:nn` (en cm ou %) retrait au début de première ligne d'un bloc
  `text-decoration:underline overline line-through`
  `line-height:nn` (en pt, cm, px ou % ; 200% : interligne doublée)
  `letter-spacing:nn` (en pt, cm, px ou %) pour espacer les lettres à l'intérieur d'un mot, *à n'utliser que très exceptionnellement*
  `color:#nnnnnn` couleur en hexadécimal ou couleur prédéfinie
  `width:nn height:nn` (en pt, cm, px ou %) ; pour fixer les dimensions d'un élément créé par les balises `p h1 ... h6 div span ...`
- Arrière-plan (balise `body`) :
  `background-color:#nnnnnnnn` (en hexadécimal ou couleur prédéfinie)
  `background-image:url(nom-fichier-image)` fond avec une image
  `background-repeat:repeat no-repeat` pour répéter ou ne pes répéter l'image
  `background-attachment:scroll fixed` l'image suit le déroulement de la page ou reste fixe.
- Marges, bordures, enrobage et positionnement :
  `margin-top:nn` (en pt, cm, px ou %) pour l'épaisseur des marges et idem avec `bottom left right`
  `border-top-width:nn` et idem avec `bottom left right` (en pt, cm, px ou %) marges d'éléments créés par des balises
  `border-width:nn` (en pt, cm, px ou %) idem pour les quatre côtés
  `border-color:#nnnnnnnn` (en hexadécimal ou couleur prédéfinie)
  `border-style:none solid double groove ...` (*groove* : strié)
  `padding-top:nn` (en pt, cm, px ou %) et idem avec `bottom left right` (en pt, cm, px ou %) pour faire des espacements entre texte et bordure de tableau ou avant et après les titres
  `padding-top:nn` (en pt, cm, px ou %) idem pour les quatre côtés avec la balise `div`
- Autres propriétés
  `list-style-type:disc circle square` puces des listes à puces
  `list-style-image:url(image.gif)` pour remplacer les puces par une image
  `list-style-position:inside outside` position des puces par rapport au texte
  `page-break-after:always auto` provoque une coupure de page après le bloc, toujours ou automatiquement (utile pour l'impression)
  `page-break-before` comme ci-dessus mais avec la coupure avant le bloc
  `position:relative absolute` place un élément à un endroit défini par `left:xx top:yy` relativement à la position qu'il occuperait normalement ou relativement par rapport une position donnée indépendante de l'environnement (origine : coin haut gauche de la page) ; il y a aussi `z-index:nn` qui définit l'ordre des recouvrements des éléments les uns par rapport aux autres (l'élément d'indice `nn` recouvre l'élément d'indice avec `nn-1`).

Pour faciliter le travail, il est possible de placer toutes les redéfinitions et définitions de variantes de balises déjà définies dans un fichier spécifique nommé, par exemple, stlsup.css (extension .css obligatoire). Si ce fichier contient la redéfinition suivante :

```
samp{font-family:times}
```

et si on place l'appel suivant dans l'entête :

```
<link rel=stylesheet type="text/css" href="stlsup.css">
```

alors la saisie `<samp>Texte</samp>` donne Texte en Times et non pas en Courrier comme cela se serait produit sans la redéfinition de la balise `samp`.

On comprend bien maintenant la dénomination "feuille de style". Cette possibilité est intéressante du point de vue pratique (saisie une fois pour toutes) mais elle a surtout l'avantage de permettre une rigoureuse homogénéité dans une série de documents.

La dénomination "cascading style sheet" provient du fait que l'on peut utiliser une feuille de style externe au document (cf. le fichier stlsup.css), en modifier certaines parties dans l'entête du document, (cf. redéfinition de la balise `samp` dans l'entête) et enfin modifier encore localement l'effet de ces parties (cf. ci-dessus la modification locale de la balise `h1` qui écrit le titre en Times). On rappelle que l'on a déjà trouvé au début de la section un effet de prévalence de la modification locale sur la définition initiale ; c'est cet aspect de "prévalence en cascade" qui a suggéré la dénomination adoptée.

On propose au lecteur un exercice pour découvrir les choix disponibles pour les tableaux en reprenant le code du tableau donné dans la sous-section Tableaux du présent document. On a déjà fait la remarque qu'un attribut affecté d'une valeur donnée est respectivement actif dans le tableau si l'affectation se fait dans la balise `<table ...>`, dans les cellules d'une ligne si l'affectation se fait dans une balise `<tr ...>` ou `<th ...>` et dans une cellule si l'affectation se fait dans une balise `<td ...>`. Cela est vrai pour la couleur du fond, la couleur de la bordure (du tableau ou des cellules), la position horizontale (à gauche, au centre ou à droite), etc. Les choix de l'italique et de la couleur du texte dans une cellule ont été respectivement choisis avec le code
`<i>` ... `</i>` et `<font color="red">` ... `</font>`
ce qui peut devenir fastidieux si l'on veut changer la famille de la fonte. La méthode des feuilles de style facilite et uniformise tous ces choix.

On prend l'exemple du changement de famille : on écrit
`style="font-family:Helvetica"`
dans la balise `<table ...>`, une balise de ligne `<tr ...>` ou `<th ...>` ou dans une balise de cellule `<td ...>` et l'effet sera visible dans tout le tableau, dans les cellules d'une ligne ou dans une cellule. En outre, on pourra définir des classes telles que :
`table.H{font-family:Helvetica}`, `tr.H{...}`, `td.H{...}` et `td.H{...}`,
ce qui facilitera une présentation homogène d'un ensemble de tableaux.

On termine par une application de la dernière propriété citée : un exemple très simple de recouvrement d'éléments (le code et les explications suivent l'affichage de l'exemple) :

Son code est (utilisation des propriétés `position top left`) :

```
<div style="text-align:center;height:45pt;font-weight:bold;
font-family:Times">
<span style="font-size:45pt;color:red">STYLES</span><br>
<span style="position:relative;font-size:15pt;font-weight:bold;
color:green;top:-36pt">Voici un exemple de superposition de
deux éléments</span></div>
```

Voilà les explications correspondantes : on place d'abord l'ensemble dans un bloc créé avec la balise `div`. Les deux lignes sont placés dans des balises `span`. Les propriétés de taille, couleur et famille de fonte sont placés normalement comme déjà vu. Ce qui est nouveau, c'est la propriété `position:relative` qui permet de placer où l'on veut la deuxième ligne, l'origine des coordonnées étant la position qu'aurait normalement occupée cette deuxième ligne ; ici, elle aurait été normalement centrée donc on ne modifie pas la coordonnées horizontale (`left:0pt` peut être omis) ; il suffit donc de remonter cette ligne de 36 pt, `top:-36pt`. Enfin, on va avoir à l'affichage un grand espacement à la place qu'aurait occupé la ligne, d'où la nécessité de limiter la hauteur de l'élément enveloppant, `height:45pt`.

Pour terminer, il faut signaler que l'affichage de ces constructions peut dépendre de l'écran de l'ordinateur client : il ne faut utiliser que des unités fixes (pt et cm) et non les pixels. L'ensemble de la machinerie permettant les superpositions les plus complexes n'est pas du niveau de ce document, on n'en dira pas davantage !

Doctype et validation

Comme on l'a dit lors de l'étude de la structure d'un document HTML, le DOCTYPE est un entête indispensable : il permet d'obtenir une validation officielle pour faciliter le référencement par les moteurs de recherche. Voilà un exemple :

```
<!DOCTYPE HTML PUBLIC
"-//W3C//DTD HTML 4.0 Transitional//EN"
"http://www.w3.org/TR/1998/REC-html40-19980424/loose.dtd">
```

Cet exemple déclare que le document est écrit en HTML version 4.0 mais peut contenir quelques petites variantes relativement à la norme de la version 4.0 ; il y a d'autres exemples sans "transitional" et "loose" remplacé par "strict" ; cela pour les cas où la version 4.0 est rigoureusement respectée.

Avec le DOCTYPE, on peut demander la validation du document. Le validateur apprécié par nombre de programmeurs français est le validateur du Quebec (w3qc.org/validator/). Il faut fournir l'adresse Web de l'URL à valider (si le document est déjà installé sur un serveur) ou le chemin d'accès local (si le document est encore sur l'ordinateur). Le validateur vérifie si le document est bien conforme à la déclaration contenue dans le DOCTYPE ; il signale aussi toutes les erreurs et affiche une aide (parfois difficile à interpréter) pour les corriger. Une fois la validation obtenue, on peut placer un avis de validation dans le document. Les moteurs de recherche ne référencent que les sites validés et c'est assez logique ! La validation assure que les navigateurs clients pourront afficher les documents aussi conformément que possible par rapport au désir du concepteur.

Hélas, contrairement à ce que peuvent laisser penser les lignes précdentes, la validation est difficile à obtenir ; elle est, de l'avis de l'auteur, à la limite du niveau débutant. Le lecteur pourra commencer par faire valider (avec ce qui est donné dans ce document) une page d'accueil simple et consulter les pages proposées par le validateur dans les explications des erreurs détectées (choisir l'option *verbose output* pour la validation). Ensuite, progressivement, il ajoutera des éléments plus complexes : tableaux, images, etc.

Remarques sur ce premier document

On n'a pratiquement pas parlé de couleurs mais ce qui a été dit et une table de couleurs (en codage RGB) permet de travailler sur des documents courants.

On a rapidement abordé les feuilles de style mais on a donné le nécessaire pour redéfinir et créer des variantes de balises. Ce qui a été exposé suffit pour commencer à produire facilement des documents peu complexes mais de qualité.

Cependant, il semble profitable pour le débutant de commencer, dès son envolée, d'utiliser les feuilles de style : il peut, par exemple, redéfinir les balises de titre et définir des classes de ces balises pour obtenir différents blocs de texte avec des fontes et des fonds colorés mettant en valeurr le texte ... en fonction de ses compétences artistiques. Tout cela est en dehors du présent document mais il peut trouver, sur les milliers de sites existants, des idées pour guider ses débuts.

Rien n'a été dit sur les cadres (*frame*) qui permettent d'afficher simultanément plusieurs pages avec possibilité de navigation d'une page à l'autre. Cette technique est très puissante, mais sa mise en pratique est très exigeante et elle est en dehors du domaine du présent document dont le propos correspond à un premier niveau permettant de produire des pages Web utilisables bien que relativement élémentaires.

Fichier(s) : f1.htm, begonia.jpg (image insérée), stlsup.css (exemple élémentaire de css contenant une seule redéfinition).

HTML et JavaScript : les formulaires !

Introduction

Le langage HTML était initialement destiné à l'affichage de documents ; sous sa forme "stricte" (cf. le premier document) il ne permet pas d'interaction entre le navigateur client et le serveur. Pour palier à ce manque de communication, il a fallu adjoindre à HTML des "extensions" pour assurer le transfert de données du navigateur client vers le serveur et vice versa ; cela a consisté en :

- la création de balises spécifiques pour assurer ce transfert de données : l'ensemble de ces balises constituant un "bloc" de fichier HTML que l'on appelle un formulaire ;
- l'utilisation de scripts en langage Java pour créer des éléments dynamiques : cela comprend la vérification des données saisies sur l'ordinateur client avant leur transfert vers le serveur, la création de fenêtres d'information pour guider cette saisie et d'autres possibilités non abordées dans ce document ; ces scripts Java constituent en quelque sorte une "extension programmable" de HTML.

Par la suite, on verra que le traitement des données reçues par le serveur et le renvoi des résultats vers le navigateur client se font à l'aide d'un autre langage nommé PHP, que l'on peut considérer comme une "suite programmable" de HTML.

Il est conseillé au lecteur de créer un fichier d2.htm pour faire quelques tests et les deux formulaires f2a.htm et f2b.htm pour tester les deux exemples d'envoi de données. Ces tests peuvent être faits en utilisant les processeurs élémentaires p2a.php et p2b.php donnés en fin du document. Il devra aussi créer le fichier f2c.htm de la fenêtre d'information contenant un peu de texte (un titre et deux ou trois petits paragraphes).

Formulaires (présentation sur un exemple)

Pour comprendre le principe des formulaires, on va considérer un cas très simple consistant à l'envoi d'une chaîne de caractères au serveur, par exemple le nom d'un étudiant dont on veut connaître le diplôme. Il s'agit d'un fichier HTML ; on va donc se concentrer sur la partie "formulaire" (à placer entre les balises `<body>` et `</body>`) et laisser de côté le titre et les éventuels textes d'information. Voici le code de l'exemple choisi (la parenthèse (1) ne fait pas partie du code : elle n'est qu'un repère utilisé pour les explications données par la suite) :

```
<form enctype="application/x-www-form-urlencoded"
action="p2a.php" method="post" onSubmit="return verification()">
Entrer le nom de l'étudiant
<input type="text" name="nom" size=20 (1)>
Cliquer pour envoyer
<input type="submit" value="Envoyer">
Cliquer pour effacer
<input type="reset" value="Effacer">;
</form>
```

Les lignes 3 à 8 de ce code donne l'affichage suivant :

Entrer le nom de l'étudiant

Cliquer pour envoyer Envoyer Cliquer pour effacer Effacer

Le lecteur peut le tester tout de suite en saisissant une chaîne de caractères dans le cadre prévu à cet effet. Sans les trois autres lignes, les deux boutons n'ont pas d'effet : il n'y a que l'affichage (que l'on améliorera en le présentant sous forme d'un tableau).

Balise form

C'est la balise principale des formulaires destinés à saisir les données sur le navigateur client puis à les envoyer au serveur. Elle accepte plusieurs attributs :

- `encaptype=...` et `method=...` sont deux attributs concernant le type de l'application et le type de transfert choisis (transferts de données) ; les valeurs qui leur sont données sont adaptées au type d'applications courantes envisagées dans ce document ; les autres valeurs possibles sont utilisées pour d'autres applications.
- `onSubmit=...` est un attribut dont la valeur affectée a la signification suivante : avant de déclencher l'envoi des données saisies, le bouton d'envoi déclenche une opération de vérification de ces données et ne les envoie que si la vérification est satisfaisante, c'est-à-dire que si la fonction de vérification retourne `true` ; cette opération de vérification fait l'objet de la section suivante : construction de la fonction JavaScipt nommée verification.
- `action=...` est un attribut prenant pour valeur le nom du processeur que l'on doit écrire et déposer sur le serveur ; ce processeur recevra le nom de l'étudiant envoyé par le formulaire, cherchera ce nom dans la base de données, prendra son diplôme et l'affichera sur l'écran du navigateur client ; c'est ce processeur qui sera écrit en PHP comme signalé plus haut (on verra tout cela dans les documents suivants).
  Dans cet exemple, pour tester ce premier formulaire, on utilise un processeur élémentaire p2a.php qui ne fait que donner un avis de fonctionnement du formulaire (il affiche à l'écran de l'ordinateur client les données que le formulaire a envoyées).

Balise input

C'est la balise servant à saisir les données et à les envoyer ; elle n'a pas de balise fermante. Elle a trois utilisations possibles. On va détailler la syntaxe et ses attributs en fonction de son utilisation :

- `type` précise le type de données entrées ; dans l'exemple et pour la première utilisation de la balise `input`, le type est `text` ; il en serait de même s'il fallait entrer d'autres données (prénom, adresse, téléphone, etc.).
  Les deux autres utilisations de la balise `input` sont très spécifiques : le type est respectivement `submit` (le clic sur le bouton correspondant déclenche l'envoi des données) et `reset` (le clic sur le bouton correspondant efface toutes les données saisies, ce qui peut être très utile dans le cas de saisies complexes).
- `name` définit une variable dans la première utilisation de la balise `input`, `xxx` par exemple ; cette variable prendra pour valeur la chaîne qui sera saisie dans le cadre prévu à cet effet. Dans le processeur recevant les données, il va y avoir création d'une variable PHP `$xxx` qui sera affectée de la valeur de la variable `xxx` (la chaîne saisie) ... mais on n'en est pas encore là !
- `value` est un attribut à placer au repère (1) qui permet, dans la première utilisation de la balise `input`, de donner à la variable définie par `name` une valeur par défaut qui s'affiche dans le cadre de saisie, ce qui n'a aucun sens pour l'exemple choisi.
  Dans les deuxième et troisième utilisations (utilisations particulièrement spécifiques

comme on vient de le voir), elle prend le texte à écrire sur le bouton correspondant.
- `size` prend le nombre de caractères que peut afficher le cadre prévu pour saisir la donnée correspondante (qui peut cependant avoir davantage de caractères, auquel cas ils ne sont pas tous visibles simultanément).

On développera plus loin toutes les possibilités disponibles (en plus de la balise `input`) pour envoyer des données. L'exemple proposé ci-dessus est simplement destiné à illustrer la "machinerie" des formulaires. Après avoir fait la saisie des données il faut passer à leur vérification avant leur envoi.

Vérifications des formulaires

On envisage tous les problèmes qui peuvent se poser pour l'exemple choisi : demander le renvoi du diplôme de l'étudiant dont on a envoyé le nom.

1. Si par inattention on presse le bouton d'envoi avant d'avoir saisi le nom, le processeur chargé de la recherche dans la base de données (écrit en PHP) va donner une erreur telle que : "La variable `$nom` n'est pas définie !" (`$nom` étant la variable PHP correspondant à la variable `nom` du formulaire).
2. Il est possible de faire une faute à la saisie du nom ; il est donc utile de demander à la personne qui fait la saisie de vérifier, avant l'envoi, si elle n'a pas fait d'erreur.
3. Il faut aussi envisager le cas où, ne voyant pas de réponse immédiate de la part du serveur surchargé, on clique de nouveau sur le bouton d'envoi ; on recevra successivement (au bout d'un certain délai) deux réponses identiques, ce qui n'est pas trop gave. Mais ce clic supplémentaire aurait des conséquences désagréables dans les cas d'un paiement d'impôt ou d'un achat en ligne !
4. On peut imaginer encore le cas où on demande le diplôme obtenu avant une date donnée dont la saisie est demandée sous la forme jj/mm/aaaa. Les dates telles que 23/13/1998 et 29/02/2001 ainsi que les dates futures, ne sont pas acceptables et donc doivent être corrigées ; on laisse le lecteur imaginer que, pour chaque type de requête, il faut prévoir de nombreuses vérifications de ce type. Bien que cela semble difficile, il y a de nombreuses possibilités de vérification utilisant les expressions régulières ... mais cela devient vite une affaire de spécialiste (exemple élémentaire : détecter qu'une adresse courriel ne contient pas le caractère @).

Il faut faire le possible pour empêcher l'envoi de données dont on est certain qu'elles sont inacceptables : c'est ce que l'on appelle la vérification des formulaires. Pour l'exemple choisi, il reste les cas où le nom est mal connu ou mal saisi : dans ce cas, c'est le serveur qui renverra un message du type "Xxxxx ne figure pas dans la liste !" ... mais ce cas est une autre affaire qui ne se traite pas à ce stade.

Pour l'exemple choisi, on va faire les vérifications concernant les points 1) et 2). Relativement au point 3), on va seulement aborder les difficultés rencontrées pour empêcher le double envoi. Pour cela on va écrire, en JavaScript, une procédure de vérification qui sera déclenchée par le *clic sur le bouton Envoyer* (cf. l'attribut `onSubmit` de la balise `form`). On testera cette procédure et l'on constatera qu'elle ne fait pas tout ce qu'il faudrait. On découvrira la difficulté de ce problème, bien cachée pour le débutant et on verra ce qu'il reste à faire pour la surmonter ... et qui n'est pas du tout du niveau de ce document ; on se contentera d'un pis aller. Voilà l'algorithme de cette procédure :

```
fois=0
définition de la fonction annexe verifnom :
 si (le nom est vide){message(Saisir le nom);retour mauvais;}
 retour bon;
```

définition de la fonction annexe veriftout :
si (fois=0) {fois=fois+1;message(Vérifier la saisie);retour mauvais;}
retour bon;
définition de la fonction verification :
si ((verifnom retourne bon) et (veriftout retourne bon))
{retour bon (donc envoi)} sinon {retour mauvais (donc blocage)}

Pour suivre le fonctionnement pas à pas de cet algorithme, il faut savoir que le premier retour mauvais trouvé dans une fonction arrête l'exécution de cette fonction et retourne mauvais indépendamment de tout le reste de la fonction. Ainsi le processus décrit par cet algorithme est le suivant :
La fonction annexe verifnom retourne mauvais (et affiche la demande de saisir un nom) si un nom n'est pas saisi, sinon elle retourne bon.
La fonction annexe veriftout retourne mauvais au premier clic (et affiche la demande de vérification de la saisie) et elle retourne bon au clic suivant.
La fontion verification retourne bon (c'est-à-dire déclenche l'envoi) si les deux fonctions annexes retournent bon toutes les deux.

On insère alors dans l'entête (entre les balises `<head>` et `</head>`) le code suivant définissant les trois fonctions verifnom, veriftout et verification correspondant à l'algorithme ci-dessus :

```
<SCRIPT language="JavaScript">
var fois=0;
function verifnom(){
   if (document.forms[0].nom.value.length<1)
   {alert("Saisir un nom puis cliquer.");
   document.forms[0].nom.focus();return false;}
<!--éventuellement autres triplets de lignes pour les autres champs-->
   return true;}
function veriftout(){
   if (fois==0){fois=fois+1;alert("Vérifier la saisie puis cliquer.");
   return false;}
   return true;}
function verification(){
   if ((verifnom()==true) && (veriftout()==true))
   {return true;}else{return false;}}
</SCRIPT>
```

Ce code nécessite les explications suivantes :

- on rappelle que le premier `return false` trouvé dans une fonction arrête l'exécution de cette fonction qui donne alors le résultat `return false` indépendamment de tout le reste de la fonction ;
- une variable `xxx` définie dans les balises `input` (cas de la variable définie avec l'attribut `name` dans l'exemple choisi) est, pour être utilisée dans les scripts Java, automatiquement renommée `document.forms[0].xxx` : l'indice `0` entre crochets est important si l'on doit avoir plusieurs formulaires sur le même fichier utilisant le même nom de variable `xxx` : c'est le cas pour le présent document (cet indice sera alors `1`) ;
- la notation `zzzz.value.length` désigne le nombre de caractères de la chaîne affectée à la variable `zzzz`, d'où le test pour déterminer si un nom a été saisi ;
- les fenêtres des messages, sont affichées avec la fonction JavaScript à un paramètre `alert("texte-du-message")` ;
- la fonction `focus()` est une fonction JavScript qui place le curseur dans le cadre où l'on doit saisir la variable dont le nom précède : c'est évidemment bien commode !

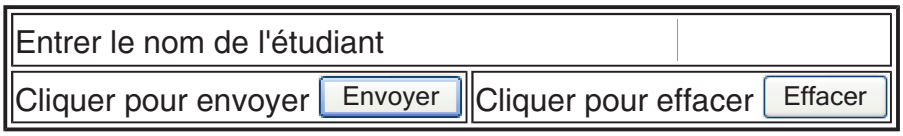
Le code du tableau pour la saisie des données (une seule donnée dans ce cas) est :

```
<form enctype="application/x-www-form-urlencoded"
action="p2a.php" method="post" onSubmit="return verification()">
<table align="center" border="2">
 <tr>
   <td colspan=2>Entrer le nom de l'étudiant
   <input type="text" name="nom" size=20></td>
 </tr>
 <tr>
   <td>Cliquer pour envoyer
       <input type="submit" value="Envoyer"></td>
   <td>Cliquer pour effacer
       <input type="reset" value="Effacer"></td>
 </tr>
</table>
</form>
```

Finalement, le formulaire complet f2a.htm est :

```
<HTML>
<HEAD>
<TITLE>Formulaire d2a.htm</TITLE>
<SCRIPT language="JavaScript">
...
</SCRIPT>
</HEAD>
<BODY>
<form enctype="application/...
...
</form>
</BODY>
</HTML>
```

et son affichage est :



On encourage le lecteur à le tester en utilisant le processeur élémentaire p2a.php dont on a déjà conseillé la saisie. Ce processeur ne fait qu'afficher les valeurs reçues : il constitue seulement un test du processus d'envoi des données par le formulaire. Le formulaire f2a.htm sera de nouveau utilisé sous le nom formua.htm après avoir remplacé p2a.php par le nom du "vrai" processeur qui renverra le diplôme de l'étudiant.

On constate que, après avoir effectué l'envoi, chaque autre clic sur le bouton d'envoi (utiliser Page précédente pour refaire apparaître le formulaire) provoque (après la pause demandant la verification de la saisie) un nouvel envoi : il faut donc éviter de cliquer une deuxième fois sur ce bouton d'envoi. Cette prescription est triviale à suivre s'il y a renvoi par le processeur d'une page qui recouvre la page du formulaire (cas de l'exemple). On verra que l'on peut apporter, au niveau débutant, une solution loin d'être satisfaisante, en créant un autre bouton qui va désactiver le bouton d'envoi.

On propose ici de découvrir la difficulté pour concevoir un code rigoureusement fiable évitant le double envoi. Pour cela, on prend la variante suivante de la fonction veriftout :

```
var fois=0;
function veriftout(){alert(fois);
if (fois==0){fois=fois+1;alert("Vérifier ...");return false;}
if (fois==1){fois=fois+1;return xxx;}
if (fois>1){alert("Patienter : les données
            ont été envoyées !");return false;}}
```

Premier test avec `xxx=false` :
La fonction `alert(fois)` affiche la valeur de la variable `fois` en début d'exécution de la fonction : après avoir saisi le nom, le premier clic sur le bouton d'envoi affiche la valeur 0 suivi de la demande de vérification, le deuxième affiche 1 et le troisième affiche 2 suivie de la demande de patienter.

Deuxième test `xxx=true` :
On reprend le processus : toujours après avoir saisi le nom. Le premier clic affiche 0 suivi de la demande de vérification, le deuxième affiche 1 et est suivi par l'envoi des données et leur affichage par le processeur p2a.php ; un clic sur "Page précédente" refait apparaître le formulaire et le troisième clic sur le bouton d'envoi affiche 0 : `return true` a envoyé les données par l'intermédiaire de la fonction `verification()` et *a terminé* le processus. Le 0 affiché montre qu'un nouveau processus a démarré (risque de double envoi) et que valeur 2, atteinte juste avant `return true`, est perdue. Il faudrait donc, d'une façon ou d'une autre, sauvegarder cette valeur 2 puis la reprendre et l'utiliser afin que le prochain clic sur le bouton d'envoi ne puisse plus faire l'envoi et affiche le conseil de patience ; c'est pour cela que la plupart des méthodes utilisent un aller-retour d'information avec le processeur (cf. le Web sur ce sujet, mots clés : bouton valider javascript et/ou post redirect get).

On repart sur le chemin des débutants en reprenant la fonction veriftout initiale. Par contre, on introduit un nouveau bouton ; cela se fait en remplaçant la partie du code du tableau située entre la première balise `<tr>` et la dernière balise `</tr>` par :

```
<tr>
   <td colspan=3>Saisir le nom de l'étudiant
   <input type="text" name="nom" size=30></td>
</tr>
<tr>
   <td>Pour envoyer <input type="submit" name="envoi"
       value="Envoyer"></td>
   <td>Pour bloquer <input type="button"
       onClick="document.forms[0].envoi.disabled=true" value="Bloquer"></td>
   <td>Pour effacer <input type="reset" value="Effacer"></td>
</tr>
```

On a donné le nom `envoi` au bouton d'envoi à l'aide de l'attribut `name` encore inutilisé et on a introduit un bouton de blocage ; son type est le type général `button` qui permet d'introduire des boutons avec l'attribut `value` prenant le texte à placer sur le bouton, l'attribut `name`, inutilisé ici, et l'attribut `onClick` prenant l'événement déclanché par les clics reçus. On notera que les types `submit` et `reset` sont des cas particuliers du type `button`. La propriété `document.forms[0].envoi.disabled` commande l'état du bouton d'envoi et donc l'événement déclanché par le clic sur le bouton de blocage désactive le bouton d'envoi (*disabled* : désactivé). Cela conduit à l'affichage suivant :

Le lecteur est de nouveau encouragé à tester cette nouvelle version du formulaire et, en particulier, la possibilité de bloquer le bouton d'envoi (on utlisera encore Page précédente pour refaire apparaître le formulaire et le processeur p2a.php pour tester l'envoi).

Dans la suite on abandonne les deux variantes proposées, l'une dans un but d'explication (fonction veriftout modifiée) et l'autre pour éviter tant que ce peut un deuxième envoi par un clic accidentel sur le bouton d'envoi (ajout d'un bouton de désactivation du bouton d'envoi) : c'est la première version du formulaire f2a.htm que l'on reprendra.

Autres possibilités pour envoyer des données

Si on demande à choisir deux villes entre quatre villes, cela va être laborieux et risque de conduire à des fautes de saisie dans les noms des villes choisies. Pour éviter toute difficulté, il suffit de présenter les noms des villes proposées et de demander de cocher en face des deux villes choisies. Autre exemple, pour choisir une commune dans un formulaire administratif, on présente la liste des communes par un "menu déroulant" : on évite ainsi toute erreur. Pour cela, il y a de nombreux moyens que l'on va examiner.

Menus déroulants

C'est la méthode idéale pour faire choisir entre plusieurs plusieurs mots ou groupes de mots avec choix unique : c'est ce qu'on appelle le "menu déroulant". Le code utilise la balise `select` (au lieu de la balise `input`), car on ne fait que choisir en cliquant sur le mot ou le groupe de mots choisis ; le code correspondant et l'affichage qu'il produit sont :

```
Villes visitées (a) :
<select name="aville">
    <option value="astg">Santiago</option>
    <option value="aslmc">Salamanca</option>
    <option value="ahc">Huesca</option>
    <option value="avldl">Valladolid</option>
</select>
```

Villes visitées (a) :
Santiago

Une fois Huesca choisie par un clic sur Huesca dans le menu déroulant, la variable `aville` a la valeur `ahc`. On peut afficher plus d'un élément avec l'attribut `size` de la balise `select`.

Cases à cocher

Cette méthode a un but semblable à la prédente mais permet des choix multiples ; elle utilise la balise `input` et le type `checkbox` ; le code et l'affichage correspondants sont :

```
Villes visitées (b) :
Santiago<input type="checkbox" name="bstg" value="oui"><br>
Salamanca<input type="checkbox" name="bslmc" value="oui"><br>
Huesca<input type="checkbox" name="bhc" value="oui"><br>
Valladolid<input type="checkbox" name="bvldl" value="oui"><br>
```

Villes visitées (b) :
Santiago
Salamanca
Huesca
Valladolid

Après le choix de Salamanca en cochant la case Salamanca, la variable `bslmc` a la valeur `oui` ; après le choix de Valladolid, la variable `bvldl` a la valeur `oui`, etc. ; on peut choisir autant de villes que l'on veut ; les variables attachées aux villes non choisies ne sont pas affectées (peu importe leur valeur).

Boutons radio

Cette méthode ressemble à la précédente du point de vue du code mais ne permet qu'un choix unique ; le code correspondant et l'affichage qu'il produit sont :

```
Villes visitées (c) :
Santiago<input type="radio" name="cville" value="cstg"><br>
Salamanca<input type="radio" name="cville" value="cslmc"><br>
Huesca<input type="radio" name="cville" value="chc"><br>
Valladolid<input type="radio" name="cville" value="cvldl"><br>
```

Villes visitées (c) :
Santiago ○
Salamanca ○
Huesca ○
Valladolid ○

Elle utilise la balise `input` et le type `radio` ; le choix unique permet l'introduction d'une seule variable qui prend des valeurs différentes suivant le bouton pressé. Une fois Huesca choisie par l'allumage du bouton Huesca, la variable `cville` a la valeur `chc`.

Vérifications pour les menus déroulants, cases à cocher et boutons radio

Voici un exemple de formulaire pour lequel on construit la vérification des données pour les trois méthodes présentées ci-dessus. On imagine que l'on demande la ville du meilleur accueil (a), la(les) ville(s) où l'on retournera l'an prochain (b) et la ville où l'on a dépensé le moins d'argent (c). Il faut donc respectivement une réponse, une ou plusieurs réponses et une réponse.

La présentation du formulaire est la plus simple possible. On peut l'imaginer sous forme de tableau avec des photos des villes ... mais ça c'est le travail du designer !

On propose d'abord l'affichage du formulaire puis son code :

Ville du meilleur accueil (a) : Liste des villes
Ville(s) où l'on retournera (b) : Santiago ☐ Salamanca ☐ Huesca ☐ Valladolid ☐
Ville la moins chère (c) : Santiago ○ Salamanca ○ Huesca ○ Valladolid ○
Cliquer pour envoyer Envoyer Cliquer pour effacer Effacer

```
<form enctype="application/x-www-form-urlencoded"
action="p2a.php" method="post" onSubmit="return verifvilles()">

Ville du meilleur accueil (a) :
<select name="aville">
    <option value="o">Liste des villes</option>
    <option value="astg">Santiago</option>
    <option value="aslmc">Salamanca</option>
    <option value="ahc">Huesca</option>
    <option value="avldl">Valladolid</option>
</select>
```

```
Villes où l'on retournera (b) :
Santiago <input type="checkbox" name="bstg" value="oui">
Salamanca <input type="checkbox" name="bslmc" value="oui">
Huesca <input type="checkbox" name="bhc" value="oui">
Valladolid <input type="checkbox" name="bvldl" value="oui"
onClick="return chaleur()">
Ville la moins chère (c) :
Santiago <input type="radio" name="cville" value="cstg">
Salamanca <input type="radio" name="cville" value="cslmc">
Huesca <input type="radio" name="cville" value="chc">
Valladolid <input type="radio" name="cville" value="cvldl">

Cliquer pour envoyer <input type="submit" value="Envoyer">
Cliquer pour effacer <input type="reset" value="Effacer">

</form>
```

On note que, pour le choix de la (des) ville(s) où l'on désire retourner, on a utilisé l'attribut `onClik` de la balise `input` en lui affectant la valeur `return chaleur()` où `chaleur()` est une fonction Java à placer dans l'entête avec les fonctions de vérification ; cette fonction fait que le clic sur la case Valladolid fait apparaître une fenêtre portant un message d'information. On laisse au lecteur le soin d'imaginer les nombreuses utilisations de cette possibilité.

Il reste maintenant à construire la fonction de vérification :

- Cas du menu déroulant (a) :
  Le menu déroulant est très utile lorsque l'on a à choisir un élément parmi un grand nombre d'éléments : il évite la perte de beaucoup d'espace. Cependant, on peut afficher plusieurs villes : on a déjà vu que l'on dispose pour cela de l'attribut `size`. On dispose de la variable `document.forms[1].aville.selectedIndex` dont la valeur est le numéro d'ordre de la ville choisie dans la liste et 0 par défaut. (on rappelle qu'en informatique les numérotations commencent à 0 par défaut). On veut que l'internaute fasse un choix : pour cela, on introduit une "fausse ville" nommée "Liste des villes" et on la place en première position : ainsi son indice vaut 0 ; pour obliger l'internaute à choisir une des quatre "vraies villes", il suffit de refuser de continuer tant que l'indice n'aura pas pris une valeur plus grande que 0 (soit 1, 2, 3 ou 4), d'où le premier test de la fonction `verifvilles()`.
- Cas des cases à cocher (b) :
  Pour les cases à cocher, à chaque case est associée une propriété nommée à partir du nom de la variable correspondant à la case : pour l'exemple, ces variables sont : `document.forms[1].bstg.checked -bsmlc.checked -bhc.checked -bvldl.checked` ; ces propriétés sont vraies si la case correspondante est cochée et fausses dans le cas contraire. Pour obliger l'internaute à cocher au moins une des cases, il suffit de refuser de continuer si au moins une des quatre propriétés n'est pas vraie, ce qui revient à dire, si les quatre propriétés sont fausses, d'où le deuxième test de la fonction `verifvilles()`. On remarque le ! qui a la signification de `not`, ce qui signifie que !(proposition-X) équivaut à (proposition-contraire-de-X). On note aussi que le "et" logique s'écrit `&&`.
- Cas des boutons radio (c) :
  A chaque bouton est associé une propriété nommée à partir du nom de la variable, soit pour l'exemple `document.forms[1].cville[i].checked`, i=0, 1, 2 et 3 : ces propriétés sont vraies si le bouton correspondant est illuminé et fausses dans le cas contraire. Pour obliger l'internaute à presser un bouton il suffit de refuser de continuer si une des quatre propriétés n'est pas vraie ou si les quatre propriétés sont fausses. On est donc ramené au problème déjà traité pour les cases à cocher. Mais il est aussi possible d'écrire directement le test qui assure qu'une de ces propriétés est

vérifiée en utilisant une boucle dont le nombre de pas est le nombre de boutons radio, d'où le troisième test de la fonction `verifvilles()`. Ce test est surtout intéressant pour les cas comportant de nombreux boutons ; il peut s'adapter au cas des cases à cocher à condition que les propriétés des cases soit définies comme les éléments d'un tableau (pour l'exemple, `bstg` serait redéfinie `bville[0]`, `bslmc` serait `bville[1]`, etc.).

L'ensemble des explications ci-dessus conduisent à la fonction `verifvilles()` suivante :

```
function verifvilles(){
if (document.forms[1].aville.selectedIndex<1)
   {alert("Faire un choix dans la liste déroulante puis cliquer.");
   return false;}
if ((!document.forms[1].bstg.checked) &&
   (!document.forms[1].bslmc.checked) &&
   (!document.forms[1].bhc.checked) &&
   (!document.forms[1].bvldl.checked))
   {alert("Choisir au moins une case puis cliquer.");return false;}
var nombre=0;
for(var i=0; i<4; i++)
   {if(document.forms[1].cville[i].checked){nombre=nombre+1;}}
if (nombre==0)
   {alert("Choisir un bouton radio puis cliquer.");return false;}
return true;}
```

Il faut alors définir une fonction `verificationbis()` en remplacement de la fonction `verification()` de l'exemple précédent :

```
function verificationbis(){
if ((verifvilles()==true) && (veriftout()==true))
   {return true;}
else
   {return false;}}
```

On remarque ci-dessus la syntaxe de la boucle : la variable de la boucle (`i` dans cet exemple), l'initialisation de cette variable, la condition sur la variable qui arrête l'exécution de la boucle et l'incrémentation de la variable (`i++` est une abréviation de `i=i+1`).

On remarque aussi que la présentation ci-dessus ne privilégie aucune ville : dans l'ordre de présentation aucune ville n'est affichée, aucune case n'est cochée et aucun bouton radio n'est illuminé. On peut, pour des raisons extérieures à l'informatique, vouloir proposer des choix. Pour cela il faut ajouter respectivement l'attribut `checked` :
- à la balise `option` correspondant à la ville proposée,
- à la(aux) balise(s) `input` correspondant à la(aux) ville(s) proposée(s),
- à la balise `input` correspondant à la ville proposée.

On aura alors, à l'ouverture du formulaire, respectivement :
- la ville proposée visible et sélectionnée,
- la(les) case(s) de(s) ville(s) proposée(s) cochée(s),
- le bouton de la ville proposée illuminé.

A ce stade le lecteur peut construire le formulaire complet f2b.htm comme on a construit précédemment le formulaire f2a.htm. La fonction `verifnom()` sera remplacée par la fonction `verifvilles()` dans laquelle l'indice [1] sera remplacé par l'indice [0] puisque le fichier f2b.htm ne contiendra qu'un seul formulaire de saisie ; enfin, la fonction `verification()` sera remplacée par la fonction `verificationbis()` : il faudra aussi adapter le titre et peut-être quelques éléments de présentation.
On pourra le tester grâce au processeur élémentaire p2b.php.

Mots de passe (remarque)

Pour la saisie du mot de passe, on introduit la ligne suivante dans le formulaire :

```
<input type="password" name="pass" size=8>
```

où le type `password` est semblable au type `text` sauf que, à la saisie, les caractères affichés sont remplacés par les traditionnelles puces noires :

Mot de passe :

La vérification du mot de passe ne peut se faire par la méthode du type exposée ci-dessus. En effet, au lieu de tester si la chaîne a été saisie, il faut tester si la chaîne saisie est identique à la chaîne `mot-de-passe` ; pour cela, on incorporerait dans la fonction de vérification deux lignes telles que :

```
if (document.forms[0].pass.value=="mot-de-passe")
     {alert("Mot de passe incorrect, recommencer !");return false;}
```

qui empêcherait l'envoi des données saisies au processeur tant que le bon mot de passe ne serait pas saisi. Mais alors, dès qu'un internaute appellerait le fichier formulaire, il pourrait l'enregistrer et s'approprier la chaîne `mot-de-passe`.
La conclusion est que la vérification des mots de passe ne peut être faite au niveau des formulaires HTML vus dans ce document. On verra plus loin une première solution de ce problème avec les formulaires en PHP.

Texte de plusieurs lignes

Il est souvent intéressant de prévoir la possibilité pour l'internaute de donner un avis, de faire une suggestion, de fournir des informations, de formuler des demandes non prévues dans le formulaire. Cela est possible avec la balise `textarea` utilisée comme suit :

```
<textarea name="avis" rows="4" cols="70">Ici votre suggestion</textarea>
```

dont l'affichage produit est :

```
Ici votre suggestion
```

Ici, le nom de la variable est `avis` ; elle prend la chaîne de caractères saisie. L'attribut `rows` prend le nombre de lignes visibles de la chaîne saisie et l'attribut `cols` prend le nombre maximum de caractères par ligne ; enfin l'attribut `wrap` (pas toujours reconnu) affecté de la valeur `virtual` automatise le retour à la ligne lors de la saisie (il reste toujours la possibilité d'aller à la ligne avec la touche habituelle). On peut afficher dans le cadre quelques mots pour attirer l'attention de la personne qui fait la saisie en plaçant ces mots entre les balises `<textarea ...>` et `</textarea>` (comme cela est montré sur l'exemple).

Ouverture de fenêtres d'information

Voici encore une utilisation de l'attribut `onClick`. Il permet l'ouverture de fenêtres d'information très simplement. Le fichier contenant ces informations doit être un fichier HTLM, f2c.htm par exemple (avec un titre dans l'entête, comme expliqué plus loin). La

syntaxe pour l'ouverture de la fenêtre affichant le fichier f2c.htm est la suivante :

```
<a href="?" onClick="window.open('f2c.htm','','width=120,height=50,
scrollbars=yes,resizable=yes,left=100,top=300')">fenêtre</a>
```

où l'argument de `onClick` doit être écrit *impérativement* sans espace et sans aller à la ligne ; les couples `attribut=valeur` donnés dans cet exemple ont un rôle facile à deviner grâce à leur nom (dimensions de la fenêtre, défilement permis, dimensions modifiables à l'affichage avec la souris et positionnement par rapport à l'écran). Il est recommandé de placer un bouton de fermeture de la fenêtre dans son propre code ; la syntaxe correspondante est :

```
<form>
<input type="button" value="Fermer" onClick="self.close()">
<form>
```

L'utilisation de ces fenêtres permet d'afficher et d'effacer successivement des informations assez volumineuses. A noter que, si le formulaire que l'on utilise se trouve dans un document s'affichant sur plus d'un écran, le clic sur le lien fait apparaître la fenêtre au début du document ; il peut être alors nécessaire de cliquer sur "Page précédente" pour se retrouver au niveau du document où l'on a ouvert la fenêtre ; pour refaire apparaître la fenêtre, il faut alors cliquer sur son titre dans la barre des tâches (d'où l'intérêt de lui donner un titre facilement repérable). Par contre, une fois qu'elle a été ouverte, on peut la réduire et la restaurer autant de fois que l'on veut.

On peut aussi ouvrir une fenêtre à l'aide d'un bouton défini par une balise `input` à l'intérieur d'un formulaire : voilà l'affichage du bouton (que l'on peut inclure dans un tableau) suivi du code correspondant :

```
<form ...>
...
<input type="button" value="Plus de détails ?"
  onClick="window.open('f2c.htm','','width=220,height=150')">
....
</form>
```

L'ouverture d'une fenêtre à l'aide d'un bouton évite la complication signalée ci-dessus : la page principale (qui contient le formulaire) ne bouge pas à l'ouverture, ce qui est appréciable pour donner des informations au fur et à mesure que l'on renseigne un long formulaire.

Remarques sur ce deuxième document

Encore un fois, il faut dire que l'on n'a considéré que des cas simples ; mais cela suffit pour commencer à traiter des cas pratiques plus complexes. Puisqu'on a souvent conclu que certains formulaires ne pouvaient se faire qu'en langage PHP, il ne faut pas croire que l'on a perdu son temps : les formulaires en PHP sont en fait des programmes en PHP, situés sur le serveur, qui écrivent des formulaires en HTML sur l'écran de l'ordinateur client ; par abus de langage, on dit que ce sont des formulaires ... patience, on y arrivera !

Fichier(s) : f2.htm, f2a.htm et f2b.htm (formulaires à reconstituer avec les parties de code données dans le texte), p2a.php et p2b.php (processeurs pour tester la vérification des formulaires précédents), f2c.htm (fichier d'une fenêtre d'information).

Processeurs élémentaires de test des formulaires

Pour placer ces deux processeurs, ainsi que tous les autres, il est proposé au lecteur de faire un saut de quelques pages pour aller au début du document suivant où l'on distingue les deux situations possibles de travail : il est conseillé de faire une première lecture des explications correspondantes et l'on en retiendra, pour l'instant, que ce qui suit.

(Cas 1) : On travaille avec un serveur extérieur, par exemple celui de son FAI (Fournisseur d'Accès à Internet). C'est dans l'espace disque alloué par le FAI (espace où l'on place généralement les pages Web personnelles) qu'il faut déposer les formulaires et les processeurs quand ils sont au point.

(Cax 2) : On travaille localement après avoir installé le "package" EasyPHP (ou équivalent). Tous les formulaires et tous les processeurs sont alors à placer dans le sous-répertoire "www local" créé automatiquement par la procédure d'installation du package EasyPHP (méthode fortement conseillée pour les mises au point).

Bien entendu, une fois que les formulaires et les processeurs sont au point, c'est dans cet espace disque alloué qu'il faut les placer pour qu'ils soient utilisables par les internautes (Cas 1). Par contre, s'il s'agit d'une gestion d'une activité personnelle ou locale, formulaires et processeurs peuvent rester dans le sous-répertoire "www local" (Cas 2).

Ces deux processeurs sont donnés sans aucune explication pour tester les formulaires ci-dessus. Ce n'est qu'à partir du document suivant que l'on verra la construction des processeurs PHP. Cependant on explique la raison de la présence d'une commande de suppression d'un warning : par défaut, PHP renvoie un warning si une variable utilisée n'est pas definie ; or ici, le nom des villes s'affiche ou ne s'affiche pas suivant qu'une variable est définie ou non ; il est donc souhaitable d'éviter l'affichage de ce warning qui ne correspond pas à une faute mais qui est un élément de code parmi les autres. Par contre, il est en d'autres cas particulièrement utile pour trouver les bogues lors des mises au point.

Fichier p2a.php

```
<HTML>
<HEAD><TITLE>Processeur p2a.htm</TITRE></HEAD>
<BODY>
<?php
error_reporting(E_ALL-E_NOTICE); //pour éviter un warning
echo "<br>";echo "Valeur reçue :<br><br>";
$nom=$_POST['nom'];$definie_nom=isset($nom);
if($definie_nom){echo $nom;}
?>
</BODY>
</HTML>
```

Fichier p2b.php

```
<HTML>
<HEAD><TITLE>Processeur p2b.htm</TITRE></HEAD>
<BODY>
<?php
error_reporting(E_ALL-E_NOTICE); //pour éviter des warnings
echo "<br>";
echo "Noms des villes choisies :<br><br>";
$aville=$_POST['aville'];$definie_aville=isset($aville);
if($definie_aville){
if ($aville=="astg"){echo "Santiago";}
```

```
if ($aville=="aslmc"){echo "Salamanca";}
if ($aville=="ahc"){echo "Huesca";}
if ($aville=="avldl"){echo "Valladolid";}}
echo "<br>"
$bstg=$_POST['bstg'];
$definie_bstg=isset($bstg);if($definie_bstg){echo "Santiago ";}
$bslmc=$_POST['bslmc'];
$definie_bslmc=isset($bslmc);if($definie_bslmc){echo "Salamanca ";}
$bhc=$_POST['bhc'];
$definie_bhc=isset($bhc);if($definie_bhc){echo "Huesca ";}
$bvldl=$_POST['bvldl'];
$definie_bvldl=isset($bvldl);if($definie_bvldl){echo "Valladolid ";}
echo "<br>"
$cville=$_POST['cville'];$definie_cville=isset($cville);
if($definie_cville){
if ($cville=="cstg"){echo "Santiago";}
if ($cville=="cslmc"){echo "Salamanca";}
if ($cville=="chc"){echo "Huesca";}
if ($cville=="cvldl"){echo "Valladolid";}}
?>
</BODY>
</HTML>
```

PHP et MySQL : va-et-vient des données !

Introduction

Dans les deux premiers documents, on a abordé ce qui se passe du côté du *navigateur client* : la conception, le vérification et le fonctionnement des formulaires (avec les langages HTML et JavaScript) destinés à la demande d'envoi de données ; ces demandes sont traditionnellement appelées des requêtes.

Dans la grande majorité des applications, les données sont stockées dans des bases de données situées sur des ordinateurs appelés *serveurs*. Par le biais des formulaires, on peut créer et alimenter ces bases de données ou, au contraire, en extraire des éléments à l'aide des requêtes. Dans le premier exemple du deuxième document concernant les formulaires, exemple dont on va continuer le développement, on envoie le nom d'un étudiant et on demande le renvoi de son diplôme.

Recevoir les données et les requêtes envoyées par l'ordinateur client, créer, remplir, modifier et lire les bases de données, envoyer les données demandées et les réponses aux requêtes à l'ordinateur client sont les tâches du *processeur* déposé sur le serveur. Dans le présent document, qui n'est qu'une petite découverte de la machinerie du Web, on va se limiter à concevoir un processeur très élémentaire pour lire une base de données également très élémentaire et renvoyer les données souhaitées au navigateur client. Il n'empêche que l'on va rencontrer (sous la forme la plus simple) certaines des principales tâches exécutées par les processeurs PHP.

Création d'une base de données élémentaire

Il n'est pas question d'aller très loin dans le monde des bases de données : c'est un domaine complexe et difficile. Pour débuter, on va se restreindre dans ce document au cas le plus simple où l'on utilise une base de données constituée d'une seule table comparable à un tableau tel que ceux que l'on utilise avec les classeurs disponibles sur les PC.

Il faut d'abord séparer les deux situations suivantes : on travaille avec un serveur et une base de données extérieurs ou, au contraire, on travaille avec un serveur et une base de données locaux ; ces deux cas seront par la suite référencés par (Cas 1) et (Cas 2) :

- (Cas 1) : On travaille avec un serveur extérieur, par exemple celui de son FAI. Prenons le cas de FREE ; ce dernier alloue un espace disque pour héberger les pages Web personnelles et les documents mis à disposition des autres internautes. C'est là que l'on va déposer les formulaires et les processeurs.

  Dans le cas du FAI cité, après être entré dans le compte (avec identifiant et mot de passe, supposés être respectivement dacos et diguns dans la suite), on peut, par un simple clic sur "Activer votre base de données MySQL", créer une base de données qui deviendra disponible après un certain délai (de l'ordre d'une demi-journée). Pour cela, on doit donner un identifiant et un mot de passe (pour simplifier on peut prendre les mêmes que pour le compte). Cette base de données sera nommée automatiquement "dacos" (l'identifiant) et pourra contenir de nombreuses tables.

  Après le délai imparti, on entre à nouveau dans le compte. On clique sur "Accéder et gérer votre base de données MySQL" et, après une nouvelle identification (identifiant et mot de passe choisis ci-dessus), la page d'accueil de l'utilitaire de gestion de

bases de données phpMyAdmin s'affiche. Cliquer alors sur "Créer une nouvelle table".

- (Cas 2) : On a installé le "package" EasyPHP (ou équivalent) qui, outre PHP, contient le serveur Appache et la base de données MySQL : on peut tout mettre au point en local et, quand tout est parfait, formulaire et processeur, on dépose ces fichiers sur le serveur du FAI : le temps gagné est ainsi très grand (surtout aux heures surchargées). L'installation de EasyPHP est simple et sans intervention (il est simplement demandé de choisir le répertoire d'installation).
  Quant on lance EasyPHP (en cliquant sur l'exécutable ou sur l'icone associée sur le bureau), il apparaît dans la barre des tâches une petite icône EasyPHP. En y cliquant dessus avec le bouton droit, on obtient un petit menu comportant Administration, Web local, etc.

  - Le clic sur "Administration" mène à la page d'accueil de phpMyAdmin. On est alors invité a créer une base de données ; on la nomme man_php pour l'exemple choisi. Après exécution, le nom de la base apparaît à gauche de l'écran, on y clique dessus et la fenêtre de gestion de la base apparaît. On clique alors sur "Créer une nouvelle table" et on se retrouve exactement au stade où l'on à abandonné le (Cas 1).
  - Le clic sur "Web local" ouvre le répertoire www (crée par l'installation). *C'est là où seront placés les formulaires et les processeurs* qui seront créés et corrigés avec un éditeur de textes (Bloc-Notes, etc.). On en profite pour dire que, pour faire les tests, il suffira de cliquer sur le nom du formulaire concerné.

Création de la table

On mène maintenant les deux cas ensemble. On crée la table en l'appelant liste_diplomes.

Après avoir cliqué sur "Exécuter", phpMyAdmin propose des tableaux où il faut donner les noms des champs (c'est-à-dire des colonnes), leur type, leur taille et de nombreuses autres propriétés parmi lesquelles on n'en considérera qu'une seule pour débuter. Cette table liste_diplomes sera référencée par dacos.liste_diplomes ou man_php.liste_diplomes respectivement dans le (Cas 1) ou le (Cas 2). On va terminer sa création comme suit :

- Pour le champ nommé `Id` (le numéro d'ordre), on choisit le type ENT (entier) et la taille 11 (c'est le nombre d'octets) ; exceptionnellement on clique sur la propriété AUTO-INCREMENT : ce champ se numérotera automatiquement à partir de la première valeur rentrée (affectée du numéro 1);

- Pour le champ nommé `Nom` (le nom de l'étudiant), on choisit le type VARCHAR (chaîne de caractères) et la taille 30 (nombre maximum de caractères) ;

- Pour le champ `Diplome` on fait le même choix que pour le champ précédent sauf que nous donnons une taille 40.

Bien entendu, on termine en sauvegardant.

Insérer les données dans la table

On reprend le processus ci-dessus, (Cas 1) ou (Cas 2), pour se retrouver à la page d'accueil de phpMyAdmin. Dans le (Cas 1), on se retrouve directement dans la base de données dacos. Dans le (Cas 2), il faudra d'abord ouvrir la base de données man_php.

Dans les deux cas on trouve, à gauche, la liste des tables créées : en cliquant sur un nom, on ouvrira la table correspondante :

- S'il n'y a pas de données enregistrées, l'utilitaire affiche les champs et leurs propriétés. On peut alors cliquer sur INSERER pour renseigner des lignes ; l'utilitaire affiche alors des tableaux qu'il suffit de renseigner ; on n'est pas obligé de remplir tous les champs de toutes les lignes en une seule fois (on pourra compléter par la suite) ; on pourra aussi ne pas remplir certains champs. A la fin, il ne faut pas oublier de cliquer sur le bouton Sauvegarder.
- S'il y a des données déjà enregistrées, l'utilitaire les affiche et on dispose alors de nombreuses possibilités :
  - entrer de nouvelles données avec INSERER comme expliqué ci-dessus,
  - corriger des données dans n'importe quelle ligne de n'importe quelle colonne (en cliquant sur le crayon après avoir coché la ligne à corriger),
  - afficher de nouveau la table avec AFFICHER,
  - vider la table (mais la structure reste) avec VIDER et effacer complètement la table (et la structure) avec EFFACER,
  - exporter la table sous différents formats (notamment au format Exel) avec EXPORTER.

Bien entendu, si l'on a fait des modifications, il faut sauvegarder avant de quitter.

Table de l'exemple choisi

A partir de maintenant, on considère que l'on dispose de la table liste_diplomes constituée avec suffisamment de données pour pouvoir faire des requêtes non triviales.

Id	Nom	Diplome
1	Valette	master de physique
2	Broussous	master de lettres modernes
3	Valette	master de chimie
4	Bardou	master de catalan
5	Durand	doctorat de physique

Création du formulaire

Comme signalé dans le document précédent, le formulaire utilisé, formua.htm, est le formulaire f2a.htm où l'on va changer la valeur p2a.php de l'attribut `action` dans la balise `form` par procea.php, le nom du processeur que l'on va construire. On devra aussi changer le titre logique et faire quelques ajouts (voir en fin du présent document).

Création du processeur

On a déjà survolé le rôle du processeur : il faut maintenant entrer dans les détails.

Généralités sur les processeurs

La structure du processeur est celle d'un fichier HTML incluant des commandes en langage PHP qui seront encadrées par les balises HTML `<?php` et `?>`.
On ne va pas faire un exposé du langage PHP qui est un vrai langage informatique avec

une syntaxe propre, des variables, des opérateurs et des structures de contrôle : on découvrira les éléments de base de ce langage au fur et à mesure de la construction du processeur. Comme le processeur a la structure d'un fichier HTML, on ne va parler que du code qui est inclus dans le corps du document entre les balises `<?php` et `?>`.

Le travail du processeur comprend :

- la récupération des données envoyées par le formulaire,
- la connexion aux bases de données,
- la lecture dans les bases de données des données demandées (via le formulaire) par le navigateur client et leur envoi au navigateur client,
- l'écriture dans les bases de données des données envoyées par le navigateur client, toujours via le formulaire (cette tâche et la suivante ne concernent pas l'exemple choisi),
- des processus beaucoup plus complexes où, à partir de données reçues de l'ordinateur client et de données lues dans les bases de données, il exécute une succession de tâches et de calculs intermédiaires se terminant par l'envoi des données résultantes.

Tout au long de ces actions, le processeur fait des :

- affichages sur l'écran de l'ordinateur client : il s'agit d'affichages d'éventuels messages d'information concernant l'avancement (ou l'échec) des requêtes et de l'affichage des données demandées, il peut aussi y avoir des messages demandant des données supplémentaires dans le cas de requêtes complexes,
- calculs de tous types (un exemple élémentaire est le calcul du total à payer en fonction des objets commandés), mais aussi des calculs sur des chaînes de caractères (un exemple simple de ce cas est donné plus loin : on forme le nom de la table à utiliser en lecture ou écriture par une concaténation de chaînes de caractères).

Pour l'exemple choisi, on va donner successivement le code des tâches énumérées ci-dessus suivi par les explications correspondantes et parfois par des remarques concernant des éventuelles généralisations.

Récupération des données envoyées par le formulaire

Le code du processeur commence par les lignes :

```
<?php
echo "<BASEFONT face='Arial'>";
echo "<br>";
echo "<h2 align='center'>Recherche du diplôme d'un étudiant</h2>";
echo "<br><br>";
echo "<h3 align='center'>Base de données : liste_diplomes</h3>";
echo "<br><br>";
$nom=$_POST['nom'];
```

La première ligne signale que le code qui suit doit être interprété en tant que code PHP, balise `<?php` (la fin du code PHP sera signalée par la balise `?>`).
La commande PHP `echo` envoie à l'ordinateur client pour affichage la chaîne de caractères entre doubles-quotes qui suit. Ainsi les lignes 2 à 7 lignes affichent sur l'écran de l'ordinateur client une entête : les six chaînes de caractères envoyées sont bien des lignes de code HTML comme on l'a vu dans le premier document. Après l'affichage de cet entête, le processeur affichera les données demandées .. mais il faut d'abord qu'il les trouve !

La commande suivante définit la variable PHP `$nom` en lui donnant la valeur de la variable `nom` envoyée par le formulaire avec la méthode `POST`, valeur de l'attribut `method` de la balise `form`
Doivent suivre ensuite les commandes de récupération de toutes les autres données envoyées par le formulaire (il n'y en a qu'une dans le cas de l'exemple choisi). Ainsi, à chaque variable `xxx` du formulaire à laquelle la valeur `aaa` est affectée, le processeur associe la variable PHP `$xxx` affectée encore de la valeur `aaa`.

Connexion à la base de données

Pour cette action et les suivantes, il faut disposer des quatre paramètres suivants :

(Cas 1)

```
$host='dacos.sql.free.fr';
$user='dacos';
$password='diguns' ;
$dbname='dacos.liste_diplomes';
```

(Cas 2)

```
$host='127.0.0.1';
$user='root';
$password='';
$dbname='man_php.liste_diplomes';
```

(Cas 1) `$host` est l'adresse où se trouve les bases de données sur le serveur ; pour le FAI cité, c'est l'identifiant suivi de .sql.free.fr ; ensuite viennent le nom d'utilisateur et le mot de passe (choisis pour simplification identiques à ceux du compte).

(Cas 2) Pour le travail en local avec EasyPHP, les trois premiers paramètres résultent de la configuration par défaut de l'installation.

(Cas 1 et 2) Le quatrième paramètre est formé dans les deux cas de la même manière par la concaténation : non_de_la_base.nom_de_la_table.

Mais il se peut que la table à consulter ou à alimenter doive être choisie en fonction des données transmises par le formulaire.
Pour la gestion des inscriptions à des activités nautiques, il se peut que l'on ait une table d'inscription pour les hommes et une pour les femmes ; dans ce cas on aurait dans le formulaire d'inscription une variable `$sexe` prenant uniquement les valeurs `H` ou `F`") et on écrirait alors dans le (Cas 1) :

```
$sexe==$_POST['sexe']; if ($sexe=="H")
{$dbname="dacos.Brasse_H";} else {$dbname="dacos.Brasse_F";}.
```

Mais on pourrait aussi écrire directement :

```
$dbname="dacos.Brasse_".$sexe;
```

qui, par concaténation des chaînes (l'opérateur de concaténation est le point), donnerait les noms dacos.Brasse_H ou dacos.Brasse_F suivant la valeur de la variable `$sexe`

On en arrive à la connexion à la base avec le code :

```
$id_connect=@mysql_connect($host,$user,$password);
if ($id_connect==false){echo "La connexion a échoué !";exit;}
   else {echo "La connexion a réussi.<br>";}
```

Dans cette partie de code, on remarque d'abord l'ajout de `@` en préfixe de la commande `mysql_connect()` pour éviter l'affichage, en cas d'échec, de messages difficiles à interpréter par le non spécialiste ; cependant, pour déboguer, il peut être utile de retirer cet `@`. Cette possibilité est valable pour toutes les autres commandes MySQL qui vont suivre.

Ensuite on détaille le fonctionnement : la fonction `mysql_connect()` renvoie l'identifiant de connexion `$id_connect` s'il y a succès ou la valeur `false` s'il y a échec. En cas d'échec, l'identifiant de connexion `$id_connect` vaut donc `false` et la première condition entre parenthèses est vraie : c'est le premier code entre accolades qui est exécuté : un message est envoyé et le programme se termine ; dans le cas contraire, c'est le deuxième code entre accolades qui est exécuté et le programme envoie un message et continue.

La déconnexion est automatique quand le processeur se termine. Pour les spécialistes, il existe une fonction spécifique pour cette opération : `mysql_close($id-connect)`.

Lecture de la table et affichage des données demandées

On arrive à la suite et fin du code de l'exemple traité; les explications, ligne par ligne (1, 2,...,11) sont données immédiatement à la suite :

```
$sql="SELECT Diplome FROM $dbname WHERE Nom='$nom'";
$requete=mysql_query($sql,$id_connect);
if ($requete==false){echo "La requête a échoué !";exit;}
   else {echo "La requête a réussi.<br>";}
$row=mysql_fetch_array($requete);
if ($row==NULL) {echo "Il n'y a pas d'étudiant nommé ".$nom." !";exit;}
   else {echo $nom." est titulaire du dipôme ".$row['Diplome']."<br>";}
$row=mysql_fetch_array($requete);
if ($row==NULL) {exit;}
   else {echo "Attention : ".$nom." est aussi titulaire du
   diplôme ".$row['Diplome']." !";exit;}
?>
```

1) définit la requête : prendre la valeur du champ `Diplome` dans les lignes de la table `$dbname` où le champ `Nom` a la valeur `$nom`

2) exécute la requête (on remarque que l'identifiant de connexion `$id_connect` est donné en paramètre) ; le résultat de la requête se retrouve dans la variable `$requete`

3) et 4) font un test pour savoir si la requête a réussi ; ce test utilise le fait que si la requête a échoué, elle retourne `false` et, dans ce cas, le programme envoie un message et se termine ; dans le cas contraire, le programme envoie un message et continue ; à ce stade, il faut bien noter la différence entre une requête qui échoue et une requête qui ne donne pas de résultat (pas de résultat est un résultat comme on va le voir quelques lignes plus loin !).

Pour pouvoir continuer les explications, il faut préciser le contenu de la variable `$requete` ; quand la requête à réussi : c'est, dans le cas général, une variable tableau à double entrée ; dans le cas traité, ce tableau n'a qu'une colonne (le diplôme de l'étudiant) ; il peut n'avoir aucune ligne (cas où il n'y a pas d'étudiant nommé `$nom` dans la liste) ; il peut avoir une ligne (cas où il n'y a qu'un étudiant nommé `$nom` dans la liste) ; il peut aussi avoir deux lignes si l'étudiant nommé `$nom` apparaît deux fois dans la liste parce qu'il a deux

diplômes ... ou parce qu'il a été inscrit deux fois (erreur dans la création de la table), etc.

Ici la requête ne demande qu'une seule donnée : le diplôme ; ce cas est tellement particulier que l'on va considérer, pour compléter l'explication, le cas où la table aurait cinq colonnes : `Id Nom Prenom Diplome Universite` et où on demanderait la liste des étudiants qui ont un diplôme donné, la liste demandée devant contenir le nom, le prénom et l'université où l'étudiant a obtenu le titre ; le code de cette requête serait le suivant :

```
$sql="SELECT Nom,Prenom,Universite FROM $dbname WHERE Diplome='$diplome'";
```

Le résultat serait alors un tableau de trois colonnes (Nom, Prénom et Université) et de zéro, une ou plusieurs lignes.

5) lit la première ligne du tableau de résultat contenu dans la variable `$requete` ; si une ligne est trouvée le résultat est placé dans la variable tableau `$row` (qui n'a ici qu'un seul élément :`$row['Diplome']` ; dans le cas contraire la commande renvoie l'élément `NULL`

6) et 7) font un test pour savoir si le tableau n'est pas vide ; si aucune ligne n'est trouvée, il y a envoi d'un message signalant qu'il n'y a pas d'étudiant nommé `$nom` dans la liste et le programme se termine ; dans le cas contraire le diplôme est alors contenu dans la variable `$row['Diplome']` et un message affiche le résultat de la requête.

Dans le cas cité ci-dessus d'un tableau à trois colonnes, le nom serait dans `$row['Nom']`, le prénom serait dans `$row['Prenom']` et l'université serait dans `$row['Universite']`

8) fait la lecture de la ligne suivante du tableau de résultat ;

9), 10) et 11) font un nouveau test : s'il n'y pas de ligne suivante, le programme se termine, sinon il affiche un message pour signaler que l'étudiant nommé `$nom` a un autre diplôme, puis se termine.

12) signale la fin du code PHP, balise `?>`

Tant que l'on débute, ces nombreux messages permettent de localiser les erreurs. Il est recommandé au lecteur de faire des essais en introduisant des erreurs : faute dans le nom d'une table ou d'une variable ou bien encore faute de syntaxe.

A titre d'exercice, le lecteur peut essayer de modifier la fin du processeur pour que, dans le cas où l'étudiant `Xxxx` apparaît deux fois dans la liste parce qu'il a obtenu deux diplômes, cas de Valette, ou parce qu'il y a un doublon dans la liste, le processeur affiche respectivement le résultat sous la forme :
"Valette est titulaire des diplômes master de physique et master de chimie."
ou
"Xxxx figure deux fois dans la liste avec le même diplôme Yyyy."
Conseil : on devra d'abord faire la lecture de toutes les lignes pour décider du type d'affichage.

Remarques sur ce troisième document

On va donner un exemple de calcul intermédiaire (il n'y en a pas dans l'exemple traité). C'est le cas d'un achat de deux objets A et B dont les prix sont respectivement 100 et 50 euros avec une réduction de 50% pour B si A est acheté. Les décisions d'achat sont représentées par des booléennes mises à true par sélection de cases à cocher ; le calcul de la somme à payer est :

```
$total=0;
if ($choixA==true) {$total=$total+100;}
if ($choixB==true) {if ($choixA==true) {$total=$total+25;}
                    else {$total=$total+50;}}
```

On notera quelques éléments de syntaxe du langage PHP rencontrés :
- les noms des variables `$xxxx`
- les syntaxes de l'affectation `=` et de la comparaison `==`
- l'élément `NULL` qui permet de savoir si la requête contient ou ne contient pas de ligne,
- la syntaxe spéciale de la structure de contrôle `if` pour laquelle il n'est pas nécessaire de donner des instructions à exécuter pour le cas où la condition est fausse,
- le double slash `//` qui annule le code qui suit sur la ligne ; il ne peut se placer que sur une ligne suivant la ligne de la balise `<?php` ou sur une ligne précédent la ligne de la balise `?>`

Il ne faut pas se faire des illusions : l'exemple traité est tellement simple qu'il est difficile de lui trouver une application dans des problèmes réels. Cependant, le lecteur est invité à le manipuler ; il utilise les fichiers suivants :
- formua.htm : formulaire,
- procea.php : processeur correspondant.

Le processeur devra être adapté suivant que l'on travaille sur le serveur d'un FAI ou en local avec une distribution de EasyPHP par exemple. Après la mise au point du processeur en local et son transfert sur le serveur du FAI, il ne faut pas oublier de *créer la base de données sur le serveur*. Si le formulaire reste en local, il faut *donner le nom du processeur précédé du chemin d'accès* à l'attribut `action` de la balise `form` ; dans le cas de l'exemple, ce serait : dacos.free.fr/procea.php.

Fichier(s) : f3.htm, formulaire et processeur : formua.htm et procea.php, table : man_php.liste_diplomes.

Fichiers complets de l'exemple

Formulaire formua.html

```
<HTML>
<HEAD>
<TITLE>Formulaire A - formua.htm</TITLE>
<SCRIPT language="JavaScript">
function verifnom(){
if (document.forms[0].nom.value.length<1)
   {alert("Saisir un nom puis cliquer.");
   document.forms[0].nom.focus();return false;}
return true;}
var fois=0;
function veriftout(){
if (fois==0)
   {fois=fois+1;alert("Vérifier les données saisies puis cliquer.");
   return false;}
return true;}
function verification(){
if ((verifnom()==true) && (veriftout()==true))
   {return true;}
else
   {return false;}}
</SCRIPT>
</HEAD>
<BODY leftmargin="40" rightmargin="40">
```

```
<BASEFONT face="Arial">
<br>
<h2 align="center">Recherche du diplôme d'un étudiant</h2>
<br>
<h3 align="center">Table : liste_diplomes</h3>
<br><br>
<p><form enctype="application/x-www-form-urlencoded"
action="procea.php" method="post" onSubmit="return verification()">
<table align="center" border="2">
 <tr>
   <td>Entrer le nom de l'étudiant</td>
   <td><input type="text" name="nom" size=20></td>
 </tr>
 <tr>
   <td>Cliquer pour envoyer
       <input type="submit" value="Envoyer"></td>
   <td>Cliquer pour effacer
       <input type="reset" value="Effacer"></td>
 </tr>
</table>
</form>
</BODY>
</HTML>
```

Processeur procea.php

```
<HTML>
<HEAD>
<TITLE>Processeur A -- Fichier procea.php</TITLE>
</HEAD>
<BODY leftmargin="40" rightmargin="40">
<?php
echo "<BASEFONT face='Arial'>";
echo "<br>";
echo "<h2 align='center'>Recherche du diplome d'un étudiant</h2>";
echo "<br><br>";
echo "<h3 align='center'>Table : liste_diplomes</h3>";
echo "<br><br>";
$nom=$_POST['nom'];
//$host='dacos.sql.free.fr';
//$user='dacos';
//$password='diguns' ;
//$dbname='dacos.liste_diplomes';
$host='127.0.0.1';
$user='root';
$password='';
$dbname='man_php.liste_diplomes';
$id_connect=@mysql_connect($host,$user,$password);
if ($id_connect==false){echo "La connexion a échoué !";exit;}
   else {echo "La connexion a réussi.<br>";"}
$sql="SELECT Diplome FROM $dbname WHERE Nom='$nom'";
$requete=mysql_query($sql,$id_connect);
if ($requete==false){echo "La requête a échoué !";exit;}
   else {echo "La requête a réussi.<br>";}
$row=mysql_fetch_array($requete);
if ($row==NULL) {echo "Il n'y a pas d'étudiant nommé ".$nom." !";exit;}
   else {echo $nom." est titulaire du diplôme ".$row['Diplome'].".<br>";}
$row=mysql_fetch_row($requete);
if ($row==NULL) {exit;}
   else {echo "Attention : ".$nom." est aussi titulaire
         du diplôme ".$row['Diplome']." !";exit;}
?>
</BODY>
</HTML>
```

Travailler avec une base de données

Introduction

On va faire un petit pas dans la gestion des bases de données à l'aide de PHP et de MySQL. Dans le document précédent, on s'était limité à des bases de données composées d'une seule table, c'est-à-dire à un tableau de données du type de ceux manipulés avec les tableurs sur les PC. On va maintenant considérer une vraie base de données qui rassemble dans différentes tables l'ensemble des données concernant un domaine de gestion, un domaine d'activité, un domaine de distribution commerciale, etc.

Domaine choisi : études d'un groupe d'étudiants

Ici, ce domaine sera la formation acquise par un certain nombre d'étudiants de familles toulousaines ayant fait leurs études dans des établissements d'enseignement situés dans d'autres villes universitaires.

Structure d'une base de données

Le principe fondamental pour la création d'une base de données est d'éviter toute redondance ; la structure la plus répendue actuellement est la suivante : les données sont stockées dans des tables, les tables de données, qui sont mises en relation par d'autres tables, les tables de relations, dont on va découvrir le rôle ci-dessous.

Exemple très élémentaire de "vraie" base de données

Pour le domaine choisi ci-dessus, on considère la base de données constituée par les quatre tables suivantes dont on va préciser la structure à la suite :

etu

Id	Nom	Prenom	Age
1	Delprat	Robert	27
2	Revellat	Marcel	23
3	Laplace	Michel	28
4	Delprat	Antoine	28
5	Andrieu	Maurice	22

dip

Id	Diplome	Duree
1	Master de Mathématiques	4
2	Master de Physique	3
3	Doctorat de Physique	4
4	Ingéniorat en Math. Appli.	3
5	Doctorat en Informatique	4

vil

Id	Ville	Distance
1	Bordeaux	294
2	Marseille	389
3	Grenoble	526
4	Paris	686

etu_dip_vil

Id_etu	Id_dip	Id_vil
1	2	1
1	3	1
2	2	4
3	1	3
3	5	2
4	2	2
4	4	1
5	1	4

On va créer directement ces quatre tables avec l'utilitaire de gestion de bases de données phpMyAdmin comme cela a été décrit au début du document précédent.

On remarque *deux* catégories de tables (cela appelle un certain nombre de remarques) :

- Les tables de données, etu, dip et vil comportent :
  - un numéro d'identification `Id`, type INT et taille (11), soit de type(11) en abrégé, possédant la propriété AUTO-INCREMENT ;
  - une donnée principale ou plusieurs données principales *identifiant la ligne de façon biunivoque* : `Non` plus `Prenom`, `Diplome` et `Ville` respectivement, avec le type VARCHAR et les tailles respectives 30 et 30, 40, et 40 ;
  - une (ou éventuellement plusieurs) donnée(s) associée(s) : `Age`, `Duree` et `Distance` respectivement avec le type INT(11), ne prenant qu'une seule valeur possible pour une valeur de la donnée principale (ou pour un ensemble de valeurs des données principales) et *ne pouvant identifier une ligne.*
- La table de relations etu_dip_vil dont les colonnes `Id_etu`, `Id_dip` et `Id_vil` sont du type INT(11). Ses lignes sont composées de trois numéros provenant respectivement des colonnes `Id` des trois tables prises dans l'ordre etu, dip et vil.
- Dans cet exemple, on ne peut pas faire un simple tableau puisque un étudiant peut avoir obtenu un ou plusieurs diplômes, chacun de ces diplômes pouvant être obtenus dans des villes différentes. Ces tables de relations sont le seul moyen de stocker toutes les données du domaine choisi sans redondance (on remarque en effet que chaque donnée n'est écrite qu'une fois).
- En général, il y a plusieurs tables de relations et aucune de ces tables ne recouvre toutes les tables de données ; c'est parce que l'exemple choisi est très simple que l'on a une table de relations recouvrant toutes les tables de données. Si l'on voulait déterminer la durée des études de chacun des étudiants, on se rend compte qu'il suffirait d'avoir une table de relations recouvrant les tables etu et dip ; en effet, la durée de préparation des diplomes est indépendante du lieu où elle est effectuée et la table de relations serait constituée des deux premières colonnes de la table etu_dip_vil.
- Le lecteur peut se demander comment sont constituées ces tables. Pour le cas de l'exemple choisi, on procède comme suit : avec les dossiers des étudiants, on construit d'abord les trois tables etu, dip et vil bien que les dossiers comportent beaucoup d'autres informations dont on pense qu'elles ne peuvent pas entrer en ligne de compte pour les réponses aux requêtes que l'on veut lancer. Ensuite, on construit la table de relations ligne par ligne : l'étudiant 1 a le diplôme 2 obtenu à l'université 1 et le diplôme 3 obtenu à l'université 2, l'étudiant 2 a le diplome 2 obtenu à l'université 4, etc. (on reprenda cet aspect en fin de document).
- Il est clair que seuls les spécialistes d'un domaine de gestion particulier sont à même de définir les tables de données et les tables de relations à utiliser pour la gestion de ce domaine ; le présent document n'a pour but que de montrer comment, sur un exemple particulièrement simple, on utilise ces tables pour en tirer des informations.

Voilà des définitions concernant les colonnes ; elles sont données à titre d'information ; en réalité, ces définitions correspondent aux propriétés déja définies ci-dessus.

- Une colonne ou un groupe de colonnes qui identifie biunivoquement une ligne est appelée une "clé candidate" ; pour la table etu, la colonne `Id` et le groupe de colonnes `Nom` et `Prenom` constituent des clés candidates (mais pas la colonne `Age`).
- La clé dite "primaire" d'une table est une clé candidate choisie pour abréger le plus possible le code des requêtes ; nous choisissons les colonnes `Id`.
- Les autres clés candidates restantes, s'il y en a, sont appellées clés "uniques".
- Une colonne ou groupe de colonnes d'une table qui référence une clé candidate d'une autre table est appelée clé "étrangère" ; ici la table etu_dip_vil a trois clés étrangères qui référencent respectivement les clés primaires des tables etu, dip et vil.

Requêtes sur la base de données choisie

Comme précédemment, on va illustrer la réalisation de requêtes par des exemples simples mais suffisamment élaborés pour apréhender les principales étapes.

Requête (1)

On se propose d'évaluer le nombre d'années d'études que l'ensemble des étudiants toulousains ont faites dans une ville autre que Toulouse.

Formulaire formub.htm

Le formulaire formub.htm correspondant est exactement celui de l'exemple du document précédent (formua.htm) dans lequel on remplace :
- le titre entre balises par Formulaire B - formub.htm
- les dénominations `verifnom()` par `verifville()` et `nom` par `ville`
- le processeur procea.php par le processeur proceb.php.

Enfin on adapte les titre, sous-titre et messages.

Processeur proceb.php

Voilà les étapes du raisonnement et du calcul que l'on ferait *à la main* dans le cas où !'on choisirait la ville de Bordeaux (numéro 1 dans la table vil) :

- Soient toutes les lignes de la table etu_dip_vil dont la colonne `Id_vil` contient 1 ;
- La première de ces lignes contient 2 en colonne `Id_dip`, ce qui, d'après la table dip, donne une durée d'études de 3 ans (master de Physique) ;
- La deuxième de ces lignes contient 3 en colonne `Id_dip`, ce qui, d'après la table dip, donne une durée d'études de 4 ans (doctorat de Physique) ;
- Enfin, la troisième et dernière de ces lignes contient 4 en colonne `Id_dip`, ce qui, d'après la table dip, donne une durée d'études de 3 ans (ingéniorat en Math. Appli.).

Au total, il y a eu 3+4+3=10 années d'études faites à Bordeaux. On remarque qu'il n'y a eu aucune référence à la table etu (on n'a utilisé que deux tables de données).

Voilà le code qui réalise cette tâche avec les explications bloc par bloc et ligne par ligne :

```
<HTML>
<HEAD>
<TITLE>Processeur B -- Fichier proceb.php</TITLE>
</HEAD>
<BODY>
<?php
echo "<br><BASEFONT face='Arial'><br>";
echo "<h2 align='center'>Durée des études dans une
     ville</h2><br>";                    // Bloc A)
echo "<h3 align='center'>Base : man-php - Tables : etu, dip, vil
      et etu_dip_vil</h3><br><br>";
$nomville=$_POST['ville'];
$duree_etude=0;
//$host='dacos.sql.free.fr';
//$user='dacos';
//$password='diguns';
$host='127.0.0.1';
$utilisateur='root';
$motdepasse='';
$id_connect=mysql_connect($host,$utilisateur,$motdepasse);
if ($id_connect==false){echo "La connexion a échoué !<br>";exit;}
```

Bloc A) (18 lignes) Il contient :

- lignes 1) à 6) entête incluant un titre (comme tous les fichiers HTML), la balise BODY et la balise de début de code PHP,
- lignes 7) à 11) pour choisir la fonte utilisée et pour afficher les titre et sous-titre sur la page réponse,
- ligne 12) pour définir la variable `$nomville` en lui affectant le nom de la ville choisie envoyé par le formulaire formub.htm,
- ligne 13) pour initialiser une variable,
- lignes 14) à 16) pour définir les paramètres nécessaires pour la connection à la base de données,
- lignes 17) et 18) pour effectuer cette connection avec un test renvoyant un message en cas d'échec de connection.

Toutes ces lignes se retrouvent dans le processeur procea.php de l'exemple du document précédent. Il faut signaler que les deux dernières tâches ci-dessus se retrouvent dans tous les processeurs qui vont lire et/ou écrire dans une base de données.

```
\\$dbname="dacos.vil";
$dbname="man_php.vil"; // Bloc B)
$sql="SELECT Id FROM $dbname WHERE Ville='$nomville'";
$requete=mysql_query($sql,$id_connect);
if ($requete==false){echo "La lecture 1 a échoué !";exit;}
$row=mysql_fetch_array($requete);
if ($row==NULL) {echo "Il n'y a pas de ville nommée ".$nomville." !";exit;}
$numville=$row['Id'];
```

Bloc B) (7 lignes). On y recherche le numéro `$numville` de la ville `$nomville` par la lecture de la table vil ; cela comprend :

- lignes 1) à 4) pour définir et effectuer la requête de lecture de la table vil avec un test renvoyant un message si la lecture a échoué,
- lignes 5) à 7) pour rechercher le numéro `$numville` avec un test renvoyant un message si la ville `$nomville` n'est pas dans la table vil.

On remarque que les lignes 3) à 6) de ce bloc constituent l'équivalant des lignes 1) à 8) de la partie de code "Lecture de la base et affichage du résultat" dans le processeur procea.php (cf. doc. précédent ; revoir éventuellement les explications qui suivent ces lignes).
On remarque aussi que, comme il y a lecture de plusieurs tables, l'on a fait la connexion à la base de données une fois pour toute avec les trois premiers paramètres de connection et l'on adapte le quatrième paramètre juste avant chaque lecture.

```
\\$dbname="dacos.etu_dip_vil";
$dbname="man_php.etu_dip_vil"; // Bloc C)
$sql="SELECT Id_dip FROM $dbname WHERE Id_vil='$numville'";
$requete=mysql_query($sql,$id_connect);
if ($requete==false){echo "La lecture 2 a échoué !";exit;}
$row=mysql_fetch_array($requete);
if ($row==NULL) {echo "Il n'y a pas eu de diplôme obtenu à ".$nomville." !";exit;}
```

Bloc C) (6 lignes) Dans ce bloc, on recherche dans la table etu_dip_vil toutes les lignes où `Id_vil=$numville` ; cela comprend :

- lignes 1) à 4) pour la définition de la requête et son exécution avec un test renvoyant un message si la lecture a échoué,
- lignes 5) et 6) pour tester si aucune ligne n'est trouvée auquel cas un message est envoyé signalant qu'aucun diplôme n'a été obtenu dans la ville nommée `$nomville`

Dans la dernière tâche, on retrouve l'utilisation de la proposition `($row==NULL)` qui, si elle est vraie, signifie que, bien que la lecture ait été réussie, aucune ligne n'a répondu à la condition exigée par la requête. Cet élément de code a déjà été utilisé à la fin du processeur procea.php et se retrouve très souvent après une lecture de table.

```
while ($row=mysql_fetch_array($requete)){ // Bloc D)
$numdiplome=$row['Id_dip'];
//$dbname="dacos.dip";
$dbname="man_php.dip";
$sql="SELECT Duree FROM $dbname WHERE Id='$numdiplome'";
$requetee=mysql_query($sql,$id_connect);
if ($requetee==false){echo "La lecture 3 a échoué !";exit;}
$row=mysql_fetch_array($requetee);
$duree=$row['Duree'];
$duree_etude=$duree_etude + $duree;}
```

Bloc D) (9 lignes) 0n y fait le calcul final à partir des lignes trouvées dans la variable $requete (cf. bloc précédent) en prenant la durée d'étude des diplômes dans la table dip.

- ligne 1) pour exécuter tout le code nécessaire pour chacune de ces lignes, les unes après les autres, on utilise l'instruction :

  ```
  while(condition){action}
  ```

  qui répète l'action tant que la condition est vraie. La condition utilisée ici mérite des éclaircissements : ils sont reportés à la suite des explications de l'action. La commande `mysql_fetch_array()` lit la première de ces lignes disponible (il y en a au mois une, cf. la fin du bloc précédent) ; l'action se terminera quand il n'y aura plus de ligne à lire ;
- ligne 2) après cette lecture, on sait qu'un diplôme numéro `$numdiplome` a été préparé dans la ville numéro `$numville`
- lignes 3) à 7) on recherche dans la table dip le champ `Duree` correspondant au diplôme numéro `$numdiplome` ; si cette lecture échoue, un message est envoyé et le programme se termine, si non cette lecture donne la durée de préparation `$duree` du diplôme numéro `$numdiplome`
- ligne 9) on calcule la durée totale des études préparées dans la ville numéro `$numville` par additions successives des valeurs de la variable `$duree` trouvées après chaque lecture de ligne de la ligne 1) du bloc.

La condition d'exécution de la boucle est :

```
($row=mysql_fetch_array($requete))
```

Explication : une fois la commande `mysql_fetch_array()` exécutée, la parenthèse `($row= ... quete)` est "une chaîne de caractères non nulle" ou `NULL`, comme on l'a déjà vu dans le processeur procea.php. Lors de l'exécution de la condition, la parenthèse est alors traduite par PHP en `true` ou `false`. Il n'est pas prévu d'aborder ce tableau de conversion dans ce document d'autant plus que l'on peut présenter cette partie de code d'une autre façon évitant l'appel à cette conversion d'une expression en une booléene. Voici le code de remplacement suivi par l'explication de son fonctionnement.

```
while (!$row==NULL){ // Bloc D) variante
$numdiplome=$row['Id_dip'];
lignes 3, 4, 5, 6, 7, 8 inchangées
$duree_etude=$duree_etude + $duree;
$row=mysql_fetch_array($requete);}
```

Cette variante fonctionne comme suit : au début du bloc D) la condition `($row==NULL)` est

fausse d'après la dernière ligne du bloc C) : la boucle s'exécute une première fois et termine sur une nouvelle lecture de `$requete`. Si la condition `($row==NULL)` est encore fausse (ce qui signife qu'il y a un deuxième diplôme obtenu dans la ville `$nomville`) la boucle s'exécute encore une fois et ceci jusqu'à ce que la condition `($row==NULL)` soit vraie.

```
echo "Durée d'études à ".$nomville." : ".$duree_etude." années<BR>"; // Bloc E)
?>
</BODY>
</HTML>
```

Bloc E) Il affiche le résultat et contient les balises fermantes nécessaires.

Requête (2)

On veut calculer les kilomètres parcourus par un étudiant en suposant qu'il y a 30 semaines de cours et que, si la distance Toulouse-ville d'étude est plus petite que 300 km, l'étudiant fait 30 A-R par année, et que, dans le cas contraire, il se limite à 15 A-R.

Formulaire formubb.htm

Le formulaire formubb.htm est exactement celui du premier exemple (formua.htm) à quelques détails et compléments près. D'abord on rajoute :

```
if (document.forms[0].prenom.value.length<1)
   {alert("Saisir un prenom puis cliquer.");
   document.forms[0].prenom.focus();return false;}
```

dans la fonction `verifnom()`, juste avant la dernière ligne `return true;}` ; on rajoute aussi :

```
<tr>
   <td>Entrer le prénom de l'étudiant</td>
   <td>input type="text" name="prenom" size=20</td>
</tr>
```

dans le tableau, juste avant la ligne `<td>Cliquer pour ....`

Ensuite on remplace :
- le titre entre balises par Formulaire BB - formubb.htm,
- le nom du processeur procea.php par procebb.php,
et l'on adapte les titre, sous-titre et messages.

Processeur procebb.php

Pour cet exemple, le processeur procebb.php comporte les mêmes éléments que celui de l'exemple précédent mais il lit les quatre tables ; voilà les étapes du calcul que l'on ferait *à la main* dans le cas où l'on choisirait l'étudiant Laplace Michel :

- On considère toutes les lignes de la table etu_dip_vil dont la colonne `Id_etu` contient 3 (numéro de Laplace Michel) ;
- La première de ces lignes contient 1 en colonne `Id_dip`, et 4 en colonne `Id_vil` ce qui, d'après les tables dip et vil, correspond à une durée d'études de 4 ans à 686 km (Master de Mathématiques à Paris) ; d'où le kilométrage 686*2*15*4=82320 km.
- La deuxième et dernière de ces lignes contient 5 en colonne `Id_dip`, et 2 en colonne `Id_vil` ce qui, d'après les tables dip et vil, correspond à une durée d'études de 4 ans à 389 km (Doctorat en Info. à Marseille) ; d'où le kilométrage 389*2*30*4=93360 km.

Au total, Laplace Michel a parcouru 82320+93360=175680 km. Voilà le code qui réalise ce travail avec les explications blocs par blocs et lignes par lignes :

```
<HTML>
<HEAD>
<TITLE>Processeur BB -- Fichier procebb.php</TITLE>
</HEAD>
<BODY>
<?php
echo "<br><BASEFONT face='Arial'><br>";
echo "<h2 align='center'>Kilométrage parcouru par un
   étudiant pour ses études</h2><br>"; // Bloc A)
echo "<h3 align='center'>Base : man-php - Tables :
   etu, dip, vil et etu_dip_vil</h3><br><br>";
$nom=$_POST['nom'];$prenom=$_POST['prenom'];
define('nombreARvillepres',30);define('nombreARvilleloin',15);
define('distancelimite',500);
$distance_etudiant=0;
//$host='dacos.sql.free.fr';
//$user='dacos';
//$password='diguns';
$host='127.0.0.1';
$utilisateur='root';
$motdepasse='';
$id_connect=mysql_connect($host,$utilisateur,$motdepasse);
if ($id_connect==false){echo "La connexion a échoué !<br>";exit;}
```

Bloc A) Il est semblable au bloc A) du processeur proceb.php de l'exemple précédent :

- lignes 1) à 6) entête ;
- lignes 7) à 11) pour la présentation des données (titre, etc.) ;
- ligne 12) pour récupérer le nom et le prénom envoyés par le formulaire ;
- lignes 13) et 14) pour définir trois données sous forme de constantes ;
- ligne 15) pour initialiser une variable ;
- lignes 16) à 20) pour la connection à la base de données : paramètres nécessaires et connexion avec envoi de message en cas d'échec de connection.

Les définitions des trois constantes ci-dessus ont des noms suffisamment explicites pour que le lecteur puisse interpréter le calcul final du Bloc D) ; ces distances sont des données spécifiques au domaine considéré. On peut imaginer une version de la requête où la distance limite (fixée à 300 km) serait une donnée fournie par le formulaire formubb.htm.

```
//$dbname="dacos.etu";
$dbname="man_php.etu"; // Bloc B)
$sql="SELECT Id FROM $dbname WHERE Nom='$nom' AND Prenom='$prenom'";
$requete=mysql_query($sql,$id_connect);
if ($requete==false){echo "La requête 1 aéchoué !";exit;}
$row=mysql_fetch_array($requete);
if ($row==NULL)
 {echo "Il n'y a pas d'étudiant nommé ".$nom." ".$prenom." !";exit;}
$numetudiant=$row['Id'];
```

Bloc B) Il est identique au bloc B) du processeur proceb.php ; il permet de trouver le numéro `$numetudiant` de l'étudiant nommé `$nom` et prénommé `$prenom` et renvoie un message si cet étudiant ne figure pas sur la table etu.

```
$dbname="dacos.etu_dip_vil";
$dbname="man_php.etu_dip_vil"; // Bloc C)
$sql="SELECT Id_dip, Id_vil FROM $dbname WHERE Id_etu='$numetudiant'";
$requete=mysql_query($sql,$id_connect);
if ($requete==false){echo "La requête 2 a échoué !";exit;}
```

Bloc C) On recherche dans la table etu_dip_vil toutes les lignes où `Id_etu=$numetudiant` ; il correspond seulement à la première partie du bloc C) du processeur proceb.php : il est raisonnable de penser que, compte tenu de la manière dont les tables ont été constituées, tout étudiant figurant dans la table etu a son numéro `$numetudiant` figurant au moins une fois dans la colonne `Id_etu` de la table etu_dip_vil et donc l'équivalent de la deuxième partie du bloc C) du processeur proceb.php serait ici sans objet.

```
while ($row=mysql_fetch_array($requete)){          // Bloc D)
$numdiplome=$row['Id_dip'];$numville=$row['Id_vil'];
//$dbname="dacos.dip";
$dbname="man_php.dip";
$sql="SELECT Duree FROM $dbname WHERE Id='$numdiplome'";
$requetee=mysql_query($sql,$id_connect);
if ($requetee==false){echo "La requête 3 a échoué !";exit;}
$row=mysql_fetch_array($requetee);
$duree=$row['Duree'];
//$dbname="dacos.vil";
$dbname="man_php.vil";
$sql="SELECT Distance FROM $dbname WHERE Id='$numville'";
$requetee=mysql_query($sql,$id_connect);
if ($requetee==false){echo "La requête 4 a échoué !";exit;}
$row=mysql_fetch_array($requetee);
$distance=$row['Distance'];
if ($distance<=$distancelimite)
   {$distancepardiplome=$duree*$nombreARvillepres*2*$distance;}
   else{$distancepardiplome=$duree*$nombreARvilleloin*2*$distance;}
$distance_etudiant=$distance_etudiant + $distancepardiplome;}
```

Bloc D) Il est l'équivalent du bloc D) de proceb.php excepté que l'on doit cette fois rechercher, pour effectuer les calculs, les valeurs des deux variables `$duree` et `$distance` (au lieu d'une seule), respectivement dans les tables dip et vil. Il y a aussi une nouvelle tâche : un calcul élémentaire incluant les trois constantes introduites dans le Bloc A) :

- ligne 1) déclenchement de la boucle qui commence par la lecture des lignes contenues dans la variable `$requete` (cf. fin du bloc précédent) ;
- ligne 2) après une lecture, on sait que l'étudiant numéro `$numetudiant` a obtenu le diplôme numéro `$numdiplome` dans la ville numéro `$numville`
- lignes 3) à 8) pour rechercher la durée `$duree` de préparation du diplome numéro `$numdiplôme`
- lignes 9) à 14) pour rechercher la distance `$distance` entre Toulouse et la ville numéro `$numville`
- lignes 15 à 18) calcul final du kilomètrage parcouru en tenant compte du fait que le nombre d'A-R dépend de la distance `$distance`

```
echo "Distance parcourue par
   ".$nom." ".$prenom." : ".$distance_etudiant."kms<BR>"; Bloc E)
?>
</BODY>
</HTML>
```

Bloc E) Le bloc final est semblable au bloc E) de proceb.php : il affiche le résultat de la requête et place les balises fermantes nécessaires.

Remarques finales sur ce document

D'abord, le lecteur est engagé à écrire la variante du bloc D) du processeur procebb.php parallèlement à la variante du bloc D) du processeur proceb.php (cf. pages précédentes).

Le présent document a été écrit sans prétention : les étudiants de nombreuses formations en informatique ont des cours sur les bases de données répartis sur plusieurs années, tout ça pour dire au lecteur que presque tout reste à apprendre ! Il est seulement une modeste initiation à cette partie l'informatique de gestion.

Jusqu'à ce document compris, on a créé et rempli les tables de données directement sur le serveur (qu'il soit local ou extérieur). Mais on verra plus loin que l'on peut aussi créer et remplir des tables par des requêtes écrites en PHP (dans les exemples précédents les requêtes choisies n'ont nécessité que des lectures de tables).
Il faut aussi souligner que l'on n'a pas présenté les catégories d'éléments du langage PHP : les variables, les opérateurs, les tableaux, les structures de contrôle, etc. ; *on n'en a simplement utilisé que quelques éléments pour traiter les exemples choisis dans ces documents.*

Pour terminer ce document, on peut dissiper partiellement le "nuage" qui enveloppe le choix des tables et de leur contenu. On a déjà noté que seuls les spécialistes du domaine d'activité concerné pouvait construire une base de données appropriée à la gestion de ce domaine.

Cependant, pour l'exemple du présent document, il est assez intuitif de proposer un ensemble de tables adhéquat : d'abord, il est évident qu'il faut une table pour les diplômes et une table pour les villes d'études.
Si l'on veut s'intéresser à des aspects liés à la durée d'études, il est logique d'introduire la durée de préparation des dplômes (d'où la table dip).
Si l'on veut s'intéresser à des questions pratiques (coût suplémentaire, durée de fatigue supplémentaire, etc.) il est logique d'introduire les distances de Toulouse aux différentes villes d'études (d'où la table vil).

Puisque l'exemple concerne l'activité d'un groupe d'étudiants, il est encore logique de faire une table pour les étudiants. L'ajout de l'âge des étudiants est inutile pour les requêtes prises en exemple : le champ âge n'a pas été utilisé.
Les tables etu et etu_dip_vil peuvent être construites logiquement de la façon suivante : par exemple, lorsque l'on reçoit l'information concernant Delprat Robert,
-- on le place dans la table etu où il prend le n° 1.
Puisqu'il a le master de Physique (dip. n° 2) à Bordeaux (ville n° 1) et le doctorat de Physique (dip. n° 3) à Bordeaux (ville n° 1),
-- on écrit dans la table etu_dip_vil les lignes 1, 2, 1 et 1, 3, 1.
Et ainsi de suite ... et, bien entendu, on peut faire un formulaire pour créer ces tables et les renseigner. Le lecteur trouvera dans les livres appropriés, des exemples de bases de données pour gérer des activtés commerciales, industrielles, etc. Ces bases comprennent des dizaines de tables.

Fichier(s) : f4.htm, les quatre tables de l'exemple : etu, dip, vil, et etu_dip_vil, formub.htm, formubb.htm, proceb.php et procebb.php (les formulaires et processeurs des deux requêtes ; le lecteur est invité à les reconstituer à l'aide des explications et/ou des blocs de code donnés).

Gérer les inscriptions à une activité

Introduction

On monte encore d'un petit pas dans l'exploitation de la machinerie du Web. On va créer des formulaires en langage PHP dont on découvrira les possibilités et les avantages au fur et à mesure de la progression dans le présent document. Pour l'instant, on se limite à dire que les formulaires en PHP ont un rôle identique à celui des formulaires en HTML tout en ayant une différence fondamentale avec ces derniers : ils sont nécessairement placés sur le serveur et ils affichent les formulaires proprement dits (en d'autres termes : les parties formulaires à renseigner) sur l'écran de l'ordinateur client.

On peut donc, dès maintenant, faire une remarque concernant un problème déjà abordé pour apprécier un exemple de ce que cette différence permet : la gestion d'un mot de passe. Voilà son traitement dans le cas des formulaires en PHP :

- le formulaire en PHP, placé sur le serveur, affiche sur l'écran de l'ordinateur client la demande d'envoi du mot de passe ;
- le processeur, également placé sur le serveur, récupère le mot de passe envoyé et le compare avec le vrai mot de passe ; si le mot de passe reçu n'est pas correct, le processeur relance le formulaire qui affiche sur l'écran de l'ordinateur client une nouvelle demande d'envoi du mot de passe ;
- le processus continue en boucle jusqu'à ce que le mot de passe soit correct (en général, on limite le processus à trois ou quatre boucles).

Ainsi la comparaison entre le mot de passe envoyé par l'ordinateur client et le vrai mot de passe se fait sur le serveur, ce qui assure une bien meilleure sécurité.

Exemple choisi : gestion des inscriptions à un repas

On propose pour débuter un exemple simplifié au maximun : la gestion des inscriptions à un repas de quartier. Pour faciliter le plus possible la compréhension du fonctionnement des outils créés pour ce type de gestion, on va éviter d'enrober les parties de code importantes par du code consacré à l'embellissement de la présentation. Cet exemple, bien que très simple, contient le nécessaire pour construire des ensembles d'outils de gestion plus importants.

L'inscription se fait avec seulement quatre champs : nom, prénom, mot de passe et plat apporté ; on aurait pu y adjoindre d'autres champs obligatoires et des champs facultatifs, par exemple le numéro de téléphone et/ou le numéro dans la rue. Le mot de passe permet à chaque inscrit de modifier et/ou d'annuler, son inscription en toute sécurité.

Création de la table des inscriptions

Il faut se reporter au troisième document pour créer la table des inscriptions nommée "dacos.repas13" ou "man_php.repas13" suivant que l'on travaille dans le (Cas 1) ou le (Cas 2) : il faut suivre exactement ce qui a été fait pour créer la table liste_diplomes.

La table repas13 compte cinq colonnes : `Id` (type ENT, taille 11, propriété AUTO-INCREMENT), `Nom` (type VARCHAR, taille 30), `Prenom` (type VARCHAR, taille 20), `Passe` (type VARCHAR, taille 6) et `Apporte` (type VARCHAR, taille 60).

Outils à créer et leur conception entièrement détaillée

On commence par établir la liste des outils à créer : on choisit délibérément de construire des petits outils indépendants les uns des autres pour chacune des tâches prévues.

Page d'accueil
repacc.htm : cette page donne l'accès à :
- la visualisation de la liste des inscrits en temps réel (en plus de savoir quels sont les voisins déjà inscrits, cette visualisation permet à l'internaute qui va s'inscrire de choisir judicieusement le plat apporté en fonction des plats déjà proposés),
- l'inscription au repas,
- la modification ou l'annulation de l'inscription.

Visualisation des inscriptions, version visiteur
repaff.php : cette liste affiche l'identité des voisins déjà inscrits et le plat qu'ils apporteront, sans afficher les mots de passe bien entendu !

Visualisation des inscriptions, version organisateur
repU430P37Baff.php : copie de la précédente visualisation montrant les mots de passe.

Formulaire d'inscription
repins.php : c'est le formulaire d'inscription à renseigner avec nom, prénom, mot de passe choisi et plat apporté.

Processeur d'inscription
repprins.php

Formulaire de recherche d'une inscription
repmod.php : c'est le formulaire à renseigner avec nom, prénom et mot de passe pour modifier ou annuler l'inscription.

Processeur de recherche d'une inscription
repprmod.php

Formulaire de modification ou d'annulation d'une inscription
repmodd.php : ce formulaire est affiché par le processeur précédent si les données reçues correspondent bien à une inscription déjà enregistrée ; il suffit de renseigner les champs que l'on veut modifier (un seul dans le présent exemple) ou de déclancher l'annulation simplement en remplaçant le nom par la chaîne Annulation.

Processeur de modification ou d'annulation
repprmodd.php

Page d'accueil

Cette page, fichier HTML, propose les trois possibilités suivantes :
- visualisation des inscriptions,
- inscriptions,
- modification ou annulation d'une inscription.

On laisse au lecteur le loisir d'imaginer une page d'accueil avec quelques unes des meilleures photos du repas de l'année précédente et toute une série de recommandations pour une réussite assurée.

Voilà le code du formulaire correspondant dont les lignes (déjà rencontrées dans des documents précédents) sont expliquées à la suite.

```
<html>
<head>
<title>Page accueil repas quartier 2013 - Fichier repacc.htm</title>
</head>
<body>

<basefont face='Arial'><br><br><br><br>
<h1 align="center">Repas de quartier des Genêts -- 30 juin 2013 à 20 h</h1>
<h3 align="center">Sous le grand saule de la rue J. Berto</h3><br><br>
<h2 align="center">Page d'accueil</h3>

<table align="center" border="6" bgcolor="aqua"  bordercolor="blue"
    cellpadding="10">
<tr><td align="center">Vous diposez de trois choix possibles</td></tr>
<tr><td align="center"><a href="http:repaff.php">Pour voir la liste
    des inscrits</a></td></tr>
<tr><td align="center"><a href="http:repins.php">Pour s'inscrire</a>
    </td></tr>
<tr><td align="center"><a href="http:repmod.php">Pour modifier ou
    annuler son inscription</a></td></tr>
</table>

</body>
</html>
```

Ce code est divisé en quatre parties : les premier et dernier blocs figurent dans le code de tous les autres fichiers de l'exemple traité ; ils ne seront donc plus affiché dorénavant : seul, le titre de l'en-tête sera mentionné.
Le deuxième bloc affiche les titre et sous-titres nécessaires (il pourrait aussi contenir des informations pour aider à renseigner le formulaire, des précisions sur le type de repas souhaité par l'organisateur, etc.). Le troisième bloc est l'ensemble des liens présentés sous forme d'un tableau ; ces liens pointent vers les formulaires correspondant aux tâches proposées.

Processeurs d'affichage des inscriptions

La liste des participants est affichée en temps réel sous la forme d'un tableau par le processeur dont le code suit et est expliqué bloc par bloc. La minime différence entre la version visiteur et la version organisateur est signalée à la ligne voulue :

```
...tle>Processeurs affichage inscriptions
                        repas quartier 2013 - Fichier repaff.php</tit...

<?php

$dateheure=date("d/m/Y à H:i:s");             //B
echo "<FONT face='arial'><br><br><br><br>
<h1 align='center'>Repas de quartier des Genêts -- 30
       juin 2013 à 20 h</h1>
<h3 align='center'>Sous le grand saule de la rue J. Berto</h3>
       <br><br>
<h2 align='center'>Etat des inscriptions le $dateheure</h2>";

//$host='dacos.sql.free.fr';$utilisateur='dacos';
//$pass='diguns';$dbname='dacos.repas13';
$host='127.0.0.1';$utilisateur='root';       //C
$motdepasse='';$dbname='man_php.repas13';
```

```
$id_connect=@mysql_connect($host,$utilisateur,$motdepasse);
  if (!$id_connect==false){echo "<br><br><br>
       <center>Connexion réussie !</center><br><br>";}
  else {echo "<br><br><br>
       <center>Echec de connection !</center><br><br>";exit;}

$sql="SELECT * FROM $dbname"; //D
$result=@mysql_query($sql,$id_connect);

echo "<TABLE align='center' bgcolor='aqua' bordercolor='blue'
       border='6'>";                          //E
echo "<TR><TH>N°</TH><TH>Nom</TH><TH>Prénom</TH>
      <TH>Passe</TH><TH>Apporte</TH></TR>";
while($row=mysql_fetch_array($result))
 {echo "<TR><TD align='right'>".$row['Id']."</TD>";
  echo"<TD>".$row['Nom']."</TD><TD>".$row['Prenom']."</TD>";
  echo"<TD>.........</TD>";  // version visiteur
  echo"<TD>".$row['Passe']."</TD>";  // version organisateur
  echo"<TD>".$row['Apporte']."</TD></TR>";}
  echo "</TABLE>";

exit;
?>
```

Le bloc B) commence par la définition d'une variable PHP contenant la date et l'heure. Ensuite, avec la commande PHP `echo "..."` qui écrit son argument sur l'écran de l'ordinateur client, il affiche les titre et sous-titres dont le dernier contient la date et l'heure.

Le bloc C), déjà rencontré, assure la connection à la base de données et renvoie un message infirmant ou confirmant la connection.

Dans le bloc D) on note d'abord que, compte tenu de ce qui a été vu précédemment, on devrait se trouver dans des situations où des couples de double-quote sont à l'intérieur des couples de double-quote encadrant l'argument des commandes `echo` ; on doit alors *protéger* les double-quote des couples intérieurs par un *caractère d'échapement*, ici le backslash (cette méthode se retrouve dans d'autres langages pour traiter les caractères ayant une fonction spécifique). Mais on peut aussi remplacer les couples de double-quote intérieurs par des couples d'apostrophes qui ont aussi la fonction de limitateur de chaîne, c'est ce qui a été fait ci-dessus. Dans la suite, on verra que ce remplacement n'est pas toujours acceptable (cf. test en fin de doc.).
Ce bloc fait la lecture de la table entière. La syntaxe générale de lecture est :

```
$sql="SELECT * FROM non-de-la-table
WHERE liste-de-conditionds-séparées-par-AND";
$result=@mysql_query($sql,$id_connect);
```

mais ici la partie `WHERE` n'est pas utilisée. On sait déjà que le résultat se retrouve dans `$result` ; si la lecture échoue, `$result` vaut `false`, d'ou la possibilité du test de réussite.

Les quatre premières lignes du bloc E) contiennent le code de construction du tableau. Dans les six lignes suivantes, l'une d'elles est à éliminer suivant la version concernée ; les cinq restantes constituent la boucle : tant que la condition :

```
$row=mysql_fetch_array($result)
```

est vraie (cf. doc. IV, processeur proceb.php bloc D), on ajoute une ligne au tableau avec les valeurs des champs :

```
$row=['Nom'], $row=['Prenom'], ......... et $row['Apporte'] // pour vis.
$row=['Nom'], $row=['Prenom'], $row['Passe'] et $row['Apporte'] // pour org.
```

Enfin; on trouve la balise fermante du tableau, la commande `exit` suivie de la commande de fin de code PHP.

Formulaire d'inscription

C'est le premier formulaire écrit en PHP ; il doit être placé sur le serveur du FAI (Cas 1) ou sur le serveur local (Cas 2). Voilà le code et les explications, bloc par bloc, en suivant.

```
...tle>Formulaire inscriptions repas
                    quartier 2013 - Fichier repins.php</tit...

<basefont face='Arial'><br><br><br><br>  //B
<h1 align='center'>Repas de quartier des Genêts -- 30 juin 2013 à 20 h</h1>
<h3 align='center'>Sous le grand saule de la rue J. Berto</h3><br><br>
<h2 align='center'>Formulaire d'inscription</h2>

<?php

$definie_fois=isset($fois);if (!$definie_fois){$fois=0;}    //C
$definie_nom=isset($nom);if (!$definie_nom){$nom="";}
$definie_prenom=isset($prenom);if (!$definie_prenom){$prenom="";}
$definie_passe=isset($passe);if (!$definie_passe){$passe="";}
$definie_apporte=isset($apporte);if (!$definie_apporte){$apporte="";}
//$apporte=stripslashes($apporte) // C6
$definie_message=isset($message);
if (!$definie_message){$message="Renseignez les champs ci-dessus :
  :<br> Le nom, le prénom, le mot de passe et le plat
  apporté sont indispensables.";}

echo "<form action='repprins.php' method='POST'>";     //D
echo "<input type='hidden' name='fois' value='$fois'>"; // D2
echo "<table align='center' border='6' bgcolor='aqua'
      bordercolor='blue' cellpadding='6'>";
echo "<tr><td align='right'>Nom</td>
  <td><input type='text' name='nom' size='25' value='$nom'></td>
  <td align='right'>Prénom</td> <td><input type='text' name='prenom'
      value='$prenom' size='16'</td></tr>";
echo "<tr><td align='right'>Mot de passe à choisir
      <br>(6 chiffres au plus)</td>
  <td><input type='password' name='passe'
      size='6' value='$passe'></td>
  <td align='right'>Plat(s)<br>apporté(s)</td>
  <td><input type='text' name='apporte'
      value=\"$apporte\" size='50'</td></tr>";
echo "<tr><td align='right'>Pour vous inscrire</td>
  <td><input type='submit' value='Envoyer'></td>
  <td align='right'>Pour effacer<br>la saisie</td>
  <td><input type='reset' value='Effacer'</td></tr>";
echo "</table>";
echo "</form>";
echo "<p align='center'>$message</p>";

exit;
?>
```

Bloc B) : ce bloc affiche les titre et sous-titres nécessaires pour la page d'inscription : ce n'est que du code HTML et donc on ne place la balise `<?php` qu'après ce bloc.

Bloc C) : il initialise les six variables PHP : `$fois $nom $prenom $passe $apporte $message` si elles ne sont pas déja définies. La fonction PHP `isset($xxx)` renvoie `true` si la variable `$xxx` est définie et `false` dans le cas contraire : c'est ainsi que les variables déjà définies conservent leur valeur et que celles qui ne sont pas définies sont initialisées. La ligne C6) et la ligne Eb1) du processeur repprins.php seront expliquées plus loin.

On ne pourra bien comprendre le rôle de cette partie de code que lorsque l'on détaillera le code du processeur correspondant repprins.php.

On donne seulement quelques explications concernant d'abord les quatre variables `$nom $prenom $passe $apporte`. Elles sont affectées d'une chaîne vide ; en conséquence, les attributs `value` (destinés à prendre les valeurs par défaut de ces variables dans les balises `input` correspondantes, bloc D) sont aussi affectés de cette valeur ; en conséquence encore, c'est cette valeur (chaîne vide) qui sera affichée pour les quatre champs à la première apparition du formulaire à l'écran.

Si on oublie par exemple de donner le prénom, le processeur va recevoir la valeur chaîne vide pour la variable `$prenom` ; il va donc retourner le formulaire avec les quatre variables affectées des trois chaînes saisies et d'une chaîne vide. Ainsi le formulaire va réapparaître avec les attributs `value` affectés des trois chaînes saisies et d'une chaîne vide. Les trois chaînes déjà saisies vont ainsi réapparaître à leur place, la place du prénom restant vide ; il faut alors saisir le prénom sinon le clic sur le bouton d'envoi refera apparaître encore une fois le formulaire.

Les variables `$fois` et `$message` ont la propriété d'être gérées par le processeur (elles sont inconnues de l'utilisateur). On approfondit le cas de la variable `$fois` : à la relance du formulaire, le processeur retourne des valeurs de `$fois` (et de `$message`) éventuellement modifiées. Au clic suivant sur le bouton d'envoi, le formulaire renvoie la valeur nouvellement reçue de `$fois` au processeur si bien que le processeur va pouvoir s'exécuter avec une valeur de `$fois` qu'il a lui-même éventuellement modifiée lors de sa précédente exécution.
Comme annoncé plus haut, tout cela sera repris lors de l'examen du code du processeur.

Bloc D) : si l'on omet la ligne D2) et si l'on prend en considération l'effet de la commande PHP `echo`, on se rend compte que le bloc D) écrit, sur l'écran de l'ordinateur client, le formulaire proprement dit de l'exemple traité suivi du message contenu dans la variable `$message` et cela comme on l'aurait écrit dans un formulaire en HTML au cours des documents précédents. Cependant, on n'a pas écrit `value='$apporte'` pour pouvoir accepter des apostrophes dans le champ Plat(s) apporté(s) (pain d'épice, etc.) ; on a gardé `value=\"$apporte\"` et ceci également dans le formulaire de modification d'inscription (cf. les tests proposés en fin de doc.). On remarquera que la variable `$fois` (dont la valeur est envoyée par la ligne D2) lors du clic sur le bouton d'envoi) a le type spécifique *hidden* (caché) ; la variable `$message` est aussi de ce type mais n'est jamais envoyée au processeur par le formulaire.

Bloc E) : il termine avec la commande `exit`. On passe au processeur associé.

Processeur d'inscription

On donne le code ; il sera expliqué, bloc par bloc, à la suite.

```
...tle>Processeur inscriptions repas
                quartier 2013 - Fichier repprins.php</tit...
```

```
<?php

$manque=0;                                          //B
$fois=$_POST['fois'];
$nom=$_POST['nom'];if($nom==""){$manque=1;}
$prenom=$_POST['prenom'];if($prenom==""){$manque=1;}
$passe=$_POST['passe'];if($passe==""){$manque=1;}
$apporte=$_POST['apporte'];if($apporte==""){$manque=1;}

if($manque==1)                                              //C
   {$message="Rappel : le nom, le prénom, le mot de passe et
   le plat apporté sont indispensables !";
   $fois=0; include"repins.php"; exit;}
else
   {if ($fois==0)
       {$message="Vérifiez avec attention votre saisie !"; //D
       $fois=1; include"repins.php"; exit;}
   else
       {//$host='dacos.sql.free.fr';$utilisateur='dacos';
       //$pass='diguns';$dbname='yves_soulet.repas13';   //E
       $host='127.0.0.1';$utilisateur='root';
       $motdepasse='';$dbname='man_php.repas13';

   $id_connect=@mysql_connect($host,$utilisateur,$motdepasse); //Ea
   if (!$id_connect==false){echo "<br><br><br>
       <center>Connexion réussie !</center><br><br>";}
       else {echo "<br><br><br>
       <center>Echec de connection !</center><br><br>";exit;}

       //$apporte=addslashes($apporte);              //Eb1 //Eb
       $sql="INSERT INTO $dbname (Nom,Prenom,Passe,Apporte)
            VALUES ('$nom','$prenom','$passe','$apporte')";
       $requete=@mysql_query($sql,$id_connect);
       if (!$requete==false)
          {echo "<br><center>Inscription réussie !</center><br>";exit;}
       else
          {echo "<br><center>Echec de l'inscription !</center>"; exit;}
       exit;}}

?>
```

Bloc B) : il initialise la variable `$manque` à 0 et récupère la valeur de la variable `fois` du formulaire pour l'affecter à la variable PHP `$fois`. Ensuite, il récupère les valeurs des variables `nom prenom passe apporte` et les place dans les variables PHP `$nom $prenom $passe $apporte`. Si une (ou plus d'une) de ces varables est affectée de la chaîne nulle, la variable `$manque` prend la valeur 1 sinon elle garde la valeur 0.

Si on a `$manque==1`, alors le bloc C) du processeur renvoie le formulaire avec :

- la variable `$message` pour rappeler que les quatre champs sont obligatoires,
- la variable `$fois` mise à 0 ; cette affectation est utille dans le cas où, au cours de la dernière vérification, une variable obligatoire serait annulée par mégarde : l'exécution du bloc B) qui en résulterait reprendrait avec `$fois=0`, sa valeur initiale, et tout le processus de vérification recommencerait,
- les variables `$fois $nom $prenom $passe $apporte` en les affectant respectivement aux attributs `value` (valeurs par défaut) des variables `fois nom prenom apporte` ; ainsi les valeurs convenables déjà envoyées sont affichées et seront automatiquement renvoyées par le clic sur le bouton d'envoi.

Cette boucle se répète tant que les variables obligatoires ne sont pas toutes affectées.

Si on a `$manque==0` (les champs obligatoires sont remplis) :

- Si `$fois==0`, alors le bloc D) du processeur relance le formulaire avec :
  - la variable `$message` contenant un message invitant cette fois à faire une dernière vérification,
  - la variable `$fois` mise à 1, ce qui a pour conséquence que le clic sur le bouton d'envoi va renvoyer au processeur cette variable mise à 1 et donc que le processeur s'exécutera avec `$fois=1` au lieu de `$fois=0`.
- Si `$fois==1`, c'est-à-dire si le bloc D) a déjà été exécuté, le bloc E) commence le processus d'enregistrement de l'inscription.

Le bloc Ea) assure la connection à la base de données avec renvoi d'un message informant de la réussite ou de l'échec de la connection (cela a dèjà été vu dans le troisième document, processeur procea.php).

Enfin le bloc Eb), ligne Eb1) omise, assure l'insertion des quatre chaînes représentant les quatre champs indispensables avec envoi d'un message infirmant ou confirmant l'inscription. On remarquera la syntaxe de l'insertion ; on verra plus loin le cas où l'on n'insère qu'une partie des champs. La syntaxe de la définition de l'insertion d'une ligne est :

```
$sql="INSERT INTO nom-de-la-table (liste-des-champs-séparés-par-virgule)
      VALUE (liste-des-valeurs-affectées-aux-champs-séparées-par-virgule)";
```

où la valeur de la variable `$xxx` se note '`$xxx`'. L'insertion est exécutée par la ligne suivante : si l'insertion échoue, alors `$requete` retoune `false`, ce qui permet de faire le test de réussite.

Les lignes C6 du formulaire associé repins.php et Eb1 du présent processeur jouent un rôle lorsque l'on saisit des champs de type VARCHAR avec des apostrophes, cas de pain d'épices pour le champ Plat(s) apporté(s). On terminera l'ensemble des formulaires et processeurs de ce document en évitant l'usage de l'apostrophe ; à la fin, on verra comment le problème est résolu.

Formulaire de recherche d'une inscription

La modification d'une inscription commence par la recherche de cette inscription : le formulaire correspondant est une réplique du formulaire d'inscription à la différence près que l'on demande seulement le nom, le prénom et le mot de passe. Voilà son code expliqué, bloc par bloc, en suivant :

```
...tle>Formulaire recherche inscription
             repas de quartier 2013 - Fichier repmod.php</tit...

<basefont face='Arial'><br><br><br><br> //B
<h1 align='center'>Repas de quartier des
      Genêts -- 30 juin 2013 à 20 h</h1>
  <h3 align='center'>Sous le grand saule
       de la rue J. Berto</h3><br><br>
  <h2 align='center'>Formulaire de modification ou d'annulation
     d'une inscription<br>Partie I - Recherche de l'inscription</h2>

<?php

$definie_fois=isset($fois);if (!$definie_fois){$fois=0;}     //C
$definie_reste=isset($reste);if (!$definie_reste){$reste=4;}
```

```
$definie_nom=isset($nom);if (!$definie_nom){$nom="";}
$definie_prenom=isset($prenom);if (!$definie_prenom){$prenom="";}
$definie_passe=isset($passe);if (!$definie_passe){$passe="";}
$definie_message=isset($message);if (!$definie_message)
 {$message="Renseignez  les champs ci-dessus :<br>Le nom,
  le prénom et le mot de passe sont indispensables.";}

echo "<form action='repprmod.php' method='POST'>"; //D
echo "<input type='hidden' name='fois' value='$fois'>"; //D2
echo "<input type='hidden' name='reste' value='$reste'>";//D3
echo "<table align='center' border='6' bgcolor='aqua'
  bordercolor='blue' cellpadding='6'>";
echo "<tr><td align='right'>Nom</td>
  <td><input type='text' name='nom' size='25' value='$nom'></td>
  <td align='right'>Prénom</td><td><input type='text' name='prenom'
      value='$prenom' size='16'</td></tr>";
echo "<tr>
  <td align='right'>Mot de passe<br>(6 chiffres au plus)</td>
  <td><input type='password' name='passe'
      size='6' value='$passe'></td>";
echo "<td> </td> <td> </td></tr>";
echo "<tr><td align='right'>Pour modifier<br>l'inscription</td>
  <td><input type='submit' value='Envoyer'></td>
  <td align='right'>Pour effacer<br>la saisie</td>
  <td><input type='reset' value='Effacer'</td></tr>";
echo "</table>";
echo "</form>";
echo "<p align='center'>$message</p>";

exit; //E
?>
```

On retrouve l'affichage des titre et sous-titres, bloc B), et l'initialisation des variables, Bloc C) : encore six variables : il n'y a plus `$apporte` mais il y a une nouvelle variable cachée `$reste` qui sert à limiter le nombre d'essais de saisie du mot de passe en cas d'oubli.

Enfin, le bloc D), excepté la ligne D2) (dont on a déjà expliqué la rôle) et la ligne D3) (assurant l'envoi de la nouvelle variable cachée) assure l'affichage du formulaire proprement dit suivi du message contenant les instrutions.

On précise le rôle de cette nouvelle variable cachée `$reste` : le processeur de recherche de l'inscription retourne le formulaire de recherche repmod.php tant qu'il n'a pas trouvé la ligne contenant les champs `Nom Prenom Passe` correspondant aux variables `$nom $prenom $passe` qui ont été saisis et il diminue `$reste` de 1 ; cette boucle s'arrête quand `$reste`, initialisée à 4, devient nulle (voir plus bas le code du processeur de recherche).

Processeur de recherche d'une inscription

La première partie du processeur de recherche est semblable à la pemière partie du processeur d'inscription, sauf qu'il n'y a que trois champs obligatoires au lieu de quatre. Voilà son code suivi des explications, bloc par bloc.

```
...tle>Processeur recherche inscription
               repas quartier 2013 à 20 h - Fichier repprmod.php></tit...
<?php

$manque=0; //B
$fois=$_POST['fois'];
$reste=$_POST['reste'];
```

```
$nom=$_POST['nom'];if($nom==""){$manque=1;}
$prenom=$_POST['prenom'];if($prenom==""){$manque=1;}
$passe=$_POST['passe'];if($passe==""){$manque=1;}

if($manque==1)
   {$message="Rappel : le nom, le prénom et le mot de passe
    sont indispensables !";$fois=0; include"repmod.php"; exit;}//C
else
  {if ($fois==0)
   {$message="Vérifiez avec attention votre saisie !"; //D
   $fois=1;include"repmod.php"; exit;}
  else
   {$fois=$fois+1; //E
   //$host='dacos.sql.free.fr';//$utilisateur='dacos';
   //$pass='diguns';//$dbname='dacos.repas13';
   $host='127.0.0.1';$utilisateur='root';
   $motdepasse='';$dbname='man_php.repas13';
   $id_connect=@mysql_connect($host,$utilisateur,$motdepasse); //Ea
    if (!$id_connect==false) {echo "<br><br><br>
       <center>Connexion réussie.
       </center><br><br>";}
   else {echo "<br><br><br>
       <center>Echec de connexion !.
       </center><br><br>";exit;}
   $sql="SELECT * FROM $dbname WHERE Nom='$nom'       //Eb
       AND Prenom='$prenom' AND Passe='$passe'";
   $requete=mysql_query($sql,$id_connect);
   if (!$requete==false)
    {echo "<br><br><br><center>La lecture
    a réussi !</center><br>";}
   else
    {echo "<br><br><br><center>La lecture
    a échoué !</center><br>"; exit;}

   $row=mysql_fetch_array($requete);                    //Ec
   if ($row==NULL){
     if($reste>0{$message="Il n&#146y a pas
       d&#146inscrit ayant ces nom, prénom et mot de passe
       !<br>Vérifiez votre saisie ! Si nom et prénom sont corrects,<br>
       c&#146est votre mot de passe qui est incorrect.<br>
       vous avez droit à $reste essais ; si le dernier est encore
       mauvais, il vous conseillera pour continuer.";
       $reste=$reste-1;include"repmod.php";exit;}
     else{echo "<br><br><br><center>
       Impossible de continuer.<br>Contacter
       l'organisateur pour qu'il vous communique
       votre mot de passe.</center>";exit;}}
   else{$num=$row['Id'];$nom=$row['Nom'];$apporte=$row['Apporte'];
     include "repmodd.php";exit;}}}
?>
```

Le bloc B) initialise `$manque` à 0 et récupère les valeurs des cinq variables.

Si on a $manque==1, le bloc C) met `$fois` à 0, place un nouveau message de rappel dans la variable `$message` pour signaler que les trois champs doivent être renseignés et retourne le formulaire de recherche avec les valeurs déjà saisies. La boucle se répète jusqu'à ce que les trois champs soient remplis.

Si on a $manque==0 (les trois champs sont remplis) :

- Si `$fois==0`, le bloc D) met `$fois` à 1 et retourne le formulaire avec un nouveau message invitant à faire une dernière vérification.

- Si `$fois==1`, le bloc E) commence le processus de recherche de l'inscription.
    - Le bloc Ea) assure la connexion à la base de données (avec renvoi d'un message infirmant ou confirmant la connexion).
    - Le bloc Eb) lit la table des inscriptions jusqu'à la ligne où les champs `Nom Prenom Passe` ont les valeurs de `$nom $prenomm $passe` avec envoi d'un message infirmant ou confirmant la validité de la lecture ; on rappelle que si la lecture demandée est réussie, cela ne signifie pas que l'inscription est trouvée. C'est le bloc Ec) qui va donner la réponse à cette question.
        - Si `$row==NULL`, ce qui signifie que la lecture, qui a réussi (bloc précédent), n'a pas trouvé d'inscription répondant à la requête, il y a deux possibilités :
            1. Si `$reste>0`, `$reste` est diminué de 1 et il y a retour du formulaire de recherche accompagné d'un message proposant de vérifier les champs demandés et donnant le nombre d'essais encore possibles.
            2. Sinon, le processeur affiche un message invitant à s'adresser à l'organisateur pour obtenir le mot de passe oublié puis termine.
        - Si `$row` n'est pas égal à `NULL`, on fait les définitions `$num=$row['Id'] $apporte=$row[Apporte]` et le formulaire de modification est envoyé.

Formulaire de modification d'une inscription

La modification d'une inscription s'achève par la modification proprement dite. Le formulaire correspondant est une réplique du formulaire d'inscription à deux différences près :

- le processeur de recherche, après avoir trouvé l'inscription dans la table, lance ce formulaire de modification en lui donnant les valeurs des cinq variables `num nom prenom passe apporte` définies et affectées des valeurs correspondant à l'inscription à modifier ;
- dans ce formulaire, on ne peut intervenir que sur les champs autres que nom, prénom et mot de passe sauf, pour annuler l'inscription ; ce détail est précisée dans le message contenu dans la variable `$messagee` et affiché sous le tableau.

Voilà le code et les explications, bloc par bloc, en suivant.

```
...tle>Formulaire de modification inscription
              repas de quartier 2013 - Fichier repmodd.php</tit...

<basefont face='Arial'>
<br><br><br><br>
<h1 align='center'>Repas de quartier des
      Genêts -- 30 juin 2013 à 20 h</h1>
      <h3 align='center'>Sous le grand saule de
      la rue Juliet Berto</h3><br><br>
      <h2 align='center'>Formulaire de modification ou
      d'annulation d'une inscription<br>Partie II - Saisie
      des éléments à modifier</h2>

<?php

$definie_foiss=isset($foiss);if (!$definie_foiss){$foiss=0;}  // C
$definie_foisss=isset($foisss);if (!$definie_foisss){$foisss=0;}
//$apporte=stripslashes($apporte);                              //C3
```

```
$definie_messagee=isset($messagee);
if (!$definie_messagee){$messagee="La modification des
      champs en rouge est sans effet (sauf exception ci-dessous).
      Pour annuler l'inscription saisir le nom Annulation
      à la place de votre nom.<br>
      Pour modifier l'inscription, modifier le champ Plat(s) apporté(s).";}

echo "<form action='repprmodd.php' method='POST'>
  <input type='hidden' name='foiss' value='$foiss'>
  <input type='hidden' name='foisss' value='$foisss'>
  <input type='hidden' name='num' value='$num'>";       //D
echo "<table align='center' border='6' bgcolor='aqua'
  bordercolor='blue' cellpadding='6'>";
echo "<tr>
  <td align='right'>Nom</td>
  <td><input type='text' name='nom' style='color:red'
      size='25' value='$nom'></td>
  <td align='right'>Prénom</td>
  <td><input type='text' name='prenom' style='color:red'
      value='$prenom' size='16'</td></tr>";
echo "<tr>
  <td align='right'>Mot de passe à choisir>
      <br>(6 chiffres au plus)</td>
  <td><input type='text' name='passe' style='color:red'
      size='6' value='$passe'></td>
  <td align='right'>Plat(s)<br>apporté(s)</td>
  <td><input type='text' name='apporte'
      value=\"$apporte\" size='50'</td></tr>";
echo "<tr>
  <td align='right'>Pour modifier<br>l'inscription</td>
  <td><input type='submit' value='Envoyer'></td>
  <td align='right'>Pour effacer<br>la saisie</td>
  <td><input type='reset' value='Effacer'</td></tr>";
echo "</table>";
echo "</form>";

echo "<p align='center'>$messagee</p>";
exit;

?>
```

Le bloc B) assure l'affichage des titre et sous-titres.

On trouve dans le bloc C) l'initialisation des variables qui ne sont pas déja définies : il y a deux variables cachées `$foiss` et `$foisss` et la variable `$messagee` qui vont jouer les rôles de `$fois`, pour les deux premières, et de `$message`, pour la dernière, jouent dans les formulaires et processeurs précédents. Le contenu de la variable `$messagee` décrit la marche à suivre.

Le bloc D) contient les envois des variables cachées `$foiss` et `$foisss` ainsi que l'envoi de la variable `$num` traitée en variable cachée pour que le processeur de modification puisse l'utiliser pour repérer dans la table la ligne à modifier. La ligne C3) et la ligne Eb221) du processeur repprmodd.php seront expliquée plus loin.

Enfin, le bloc D) assure l'affichage du formulaire proprement dit qui apparaît entièrement renseigné avec les champs non modifiables colorés en rouge et le message précisant la marche à suivre affiché en dessous. A noter que l'attribut `type` de la variable `passe` est affecté de la valeur `password` : le mot de passe s'affiche avec les puces ; avec la valeur `text`, il ne serait pas caché.

Processeur de modification d'une inscription

On arrive finalement au dernier élement de cet ensemble de gestion : le processeur de modification d'inscription proprement dit. Comme précédemment, on donne le code et on l'explique, bloc par bloc, en suivant.

```
..tle>Processeur modification inscription
        repas quartier 2013 - Fichier repprmodd.php</tit...
<?php

$manque=0;                                          // B
$foiss=$_POST['foiss'];$foisss=$_POST['foisss'];
$num=$_POST['num'];$nom=$_POST['nom'];
$prenom=$_POST['prenom'];$passe=$_POST['passe'];
$apporte=$_POST['apporte'];if($apporte==""){$manque=1;}

//$host='dacos.sql.free.fr';
//$utilisateur='dacos';
//$pass='diguns';
//$dbname='dacos.repas13';
 $host='127.0.0.1'; $utilisateur='root';        //C
 $motdepasse='';
 $dbname='man_php.repas13';
  $id_connect=@mysql_connect($host,$utilisateur,$motdepasse);
   if (!$id_connect==false){echo "<br><br><br>
     <center>Connexion réussie !</center><br><br>";}
    else{echo "<br><br><br>
     <center>Echec de connection !</center> <br><br>";exit;}

if($nom=='Annulation'){$foisss=0;                           // Da
   if($foiss==0)
      {$foiss=$foiss+1;$messagee="Soit vous confirmez l'annulation<br>
      soit vous passez à l'autre choix en replaçant votre nom";  // Da1
      include "repmodd.php";exit;}
   else
   {$sql="UPDATE $dbname SET Nom='', Prenom='', Passe='',
     Apporte='' WHERE Id='$num'";                           // Da2
     $requete=@mysql_query($sql,$id_connect);
     if (!$requete==false){echo "<br><br><br><center>
       Annulation réussie !</center><br>"; exit;}
    else
       {echo "<br><br><br><center>Echec de
       l'annulation!</center>"; exit;}}}

if($nom<>'Annulation'){$foiss=0;                        // Db
  if($manque==1){$messagee="Rappel : le plat apporté est indispensable !";
                 $foisss=0; include "repmodd.php"; exit;}   // Db1
  else
    {if($foisss==0)                                         // Db2
      {$foisss=1;$messagee="Soit vous confirmez le changement<br>
      soit vous passez à l'autre choix en saisissant le nom Annulation";
      include "repmodd.php";exit;}                          // Db21
    else
      {$apporte=addslashes($apporte);  // Db221
      $sql="UPDATE $dbname SET Apporte='$apporte' WHERE Id='$num'";
      $requete=@mysql_query($sql,$id_connect);              // Db22
      if (!$requete==false){echo "<br><br><br><center>
       Modification réussie !</center><br>"; exit;}
      else
       {echo "<br><br><br><center>Echec
       de la modification !</center>"; exit;}}}}

?>
```

Le bloc B) initialise `$manque` à 0 et assure la récupération du contenu des variables parmi lesquelles `$prenom` et `$passe` qui ne vont que servir que pour l'affichage puisqu'elles ne peuvent pas être modifiées.

La connection à la base de données est faite par le bloc C) avec message de réussite ou d'échec.

Ensuite le bloc D) comprend deux blocs :

Si on a `$nom=="Annulation"`, bloc Da), d'abord `$foisss` est mise à 0, ensuite :

- Si `$foiss==0`, bloc Da1), `$foiss` est mise à 1 et le formulaire de modification est relancé avec un message demandant
  -- soit la confirmation de l'annulation,
  -- soit le passage à l'autre choix en saisissant la modification du champ Plat(s) apporté(s), puis en remettant le nom en place et enfin en terminant par la confimation.
- Si `$foiss<>=0`, bloc Da2), il y a effacement des champs `Nom Prenom Passe Apporte`, effacement désignant ici la modification particulière consistant à remplacer la chaîne initiale par la chaîne vide. Enfin il y a un message de réussite ou d'échec d'effacement de l'inscription.

La syntaxe générale pour la modification des champs dans les tables est :

```
$sql="UPDATE nom-de-la-table
SET couples-champ:valeur-séparés-par-virgule
WHERE condition-identifiant-la-ligne";
```

où la condition identifiant la ligne peut être beaucoup plus complexe (ici, la ligne est simplement identifiée par son numéro, la variable `$num`, récupérée par le formulaire de recherche d'inscription).

Si on a `$nom<>"Annulation"`, bloc Db) ; d'abord `$foiss` est mise à 0, ensuite :

- Si `$manque==1`, bloc Db1) ; le formulaire de modification est relancé avec un message rappelant que le champ Plat(s) apporté(s) est indispensable.
- Si `$manque==0`, bloc Db2), il y a les deux possibilités :

  - Si `$foisss==0`, bloc Db21) : `$foisss` est mise à 1 et le formulaire de modification est relancé avec un message proposant :
    -- soit de faire la modification puis de la comfirmer,
    -- soit de passer à l'autre choix en saisissant le nom Annulation.

  - Si `$foisss==1`, bloc Db22) : il y a enregistrement de la modification, ici limité au seul champ `Apporte`, suivi d'un message annonçant la réussite ou l'échec de la modification.
    La ligne C3) du formulaire de modification repmodd.php et la ligne Db221) sont encore liées à l'utilisation de l'apostrophe.

Traitement de l'apostrophe

Pour aborder ce problème, déjà mentionné, on propose de donner la « recette » (qui dépend de la configuration PHP utilisée) suivie de quelques tests pour comprendre la difficulté résulatnt du choix de l'apostrophe pour limiteur de chaîne de caractère.

Toutes les versions de PHP disposent des fonctions `addslashes(.)` et `stripslashes(.)` qui ajoutent et enlèvent repspectivement un backslash devant les caractères ayant une fonction spécifique dans le langage PHP (apostrophe, double-quote et backslash). Mais certaines installations ont la variable `magic_quotes_gpc` initialisée à `off` (cas d'EasyPHP avec installation par défaut) ; dans ce cas, on doit utiliser la fonction `addslashes(.)` pour pouvoir utiliser l'apostrophe sans problème. Par contre avec d'autres serveurs, celui du FAI FREE par exemple, la variable `magic_quotes_gpc` est initialisée à `on` ; on peut ainsi enregistrer directement : il faut alors désactiver `addslashes(.)` dans les processeurs d'inscription et modification d'inscription ; mais la difficulté se retrouve à l'affichage et il faut activer la fonction `stripslashes(.)` dans les formulaires d'inscription et modification d'inscription où elle se trouve désactivée par des `//` ; voir les tests suivants.
On peut modifier cette variable dans le fichier d'initialisation du programme PHP : c'est possible dans le (Cas 2), avec un serveur local, mais on ne peut pas aller tripoter dans le serveur du FAI (Cas 1) ! Le petit programme nommé valmag.php :

```
<html><head><title>val_mag.php</title></head><body>
<?php $valmaj=get_magic_quotes_gpc();echo "valmaj : ".$valmaj; ?>
</body></htlm>
```

affiche `0` si la variable est à `off` et `1` si elle est à `on`. Il se place localement (Cas 2) avec les formulaires et pocesseurs. Dans le cas du serveur extérieur (Cas 1), il faut le déposer sur le serveur avec les autres formulaires et proceseurs puis le lancer avec la commande :
`http://dacos.free.fr/valmag.php`

On commence par le cas où `magic-quotes_gpc` à `off` (cas du présent ouvrage) :

- Faire une inscription avec :
  `value='$apporte'` au lieu de `value=\"$apporte\"`
  dans le formulaire d'inscription, en oubliant volontairement le prénom et en prenant « pain d'épice » pour le champ Plat(s) apporté(s) ; résultat : l'apostrophe est considérée comme une fin de chaîne et la partie « 'épice » disparaît dès le début.
- Faire une inscription avec :
  `//` devant `$apporte=addslashes($apporte);` (commande désactivée)
  dans le processeur d'inscription et toujours avec « pain d'épice » ; résultat : l'apostrophe est encore interprétée comme une fin de chaîne et les caractères restant génèrent une erreur à l'insertion ; il n'y a pas d'inscription.
- Faire le même test, avec cette fois « pain d\'épice » ; résultat : l'inscription est réussie (vérifier en affichant la liste des inscription) ; cette méthode a un intérêt pédagogique mais n'est pas très pratique !
- Faire le test normal, toujours avec « pain d'épice » ; résultat : la commande `addslashes(.)` a ajouté le backslash devant l'apostrophe et l'inscription est réussie.

On fait trois tests avec `magic_quotes_gpc` à `on` (fichier php.ini), ce qui permert l'insertion de l'apostrophe directement :

- En conséquence de cette possibilité, on va faire une inscription avec :
  `//` devant `$apporte=addslashes($apporte);` (commande désactivée)
  dans le processeur d'inscription et avec « pain d'épice » ; résultat : l'inscription s'est faite mais un backslash s'est placé devant l'apostrophe !
- Le résultat précédent ci-dessus suggère de faire une une inscription toujours avec :
  `//` devant `$apporte=addslashes($apporte);` (command désactivée)
  dans le processeur d'inscription, mais sans :
  `//` devant `$apporte=stripslashes($apporte);` (commande activée)
  dans le formulaire d'inscription et avec « pain d'épice » ; résultat : le backslash

n'apparaît plus, mais si on tente ensuite une modification d'inscription le backslash réapparaît. Il faut alors faire aussi les deux modifications ci-dessus respectivement dans les formulaire et processeur de modification d'inscription : c'est l'objet du troisième test laissé à l'initiative du lecteur. Cette dernière configuration est celle que le lecteur devra utiliser si le serveur de son FAI a la variable `magic_quotes_gpc` à `on`.

Autre construction des formulaires PHP

Dans ce document, on peut dire que l'on a opté pour une construction des formulaires PHP un peut brutale. En effet, il n'y a que le code correspondant aux titre et sous-titres qui sont en dehors de la zone d'activation de PHP, zone délimitée par les balises `<?php` et `?>` elles mêmes incluses. Mais on peut faire le contraire en activant PHP uniquement lorsque c'est indispensablbe. Le choix initial était possible car la commande PHP `echo` écrit, par construction, les parties de code HTML contenues dans son argument.

On propose une nouvelle version, repins00.php du formulaire repins.php écrite comme cela est suggéré ci-dessus ; les zones d'activation de PHP sont typographiées en gras.

```
...tle>Forlulaire inscriptions repas
                    quartier 2013 - Fichier repins00.php</tit...

<basefont face='Arial'><br><br><br><br>
<h1 align='center'>Repas de quartier des Genêts -- 30 juin 2013 à 20 h</h1>
  <h3 align='center'>Sous le grand saule de la rue J. Berto</h3><br><br>
  <h2 align='center'>Formulaire d'inscription</h2>
<?php
$definie_fois=isset($fois);if (!$definie_fois){$fois=0;}
$definie_nom=isset($nom);if (!$definie_nom){$nom="";}
$definie_prenom=isset($prenom);if (!$definie_prenom){$prenom="";}
$definie_passe=isset($passe);if (!$definie_passe){$passe="";}
$definie_apporte=isset($apporte);if (!$definie_apporte){$apporte="";}
//$apporte=stripslashes($apporte)
$definie_message=isset($message);
if (!$definie_message){$message="Renseignez les champs ci-dessus :
  <br> Le nom, le prénom, le mot de passe et le plat
  apporté sont indispensables.";}
?>
<form action='repprins.php' method='POST'>
<input type='hidden' name='fois' value='<?php echo $fois ?>'>
<table align='center' border='6' bgcolor='aqua'
      bordercolor='blue' cellpadding='6'>
<tr><td align='right'>Nom</td>
  <td><input type='text' name='nom' size='25' value='<?php echo $nom ?>'</td>
  <td align='right'>Prénom</td> <td><input type='text' name='prenom'
      value='<?php echo $prenom ?>' size='16'</td></tr>
<tr><td align='right'>Mot de passe à choisir
      <br>(6 chiffres au plus)</td>
  <td><input type='password' name='passe'
      size='6' value='<?php echo $passe ?>'></td>
  <td align='right'>Plat(s)<br>apporté(s)</td>
  <td><input type='text' name='apporte'
      value=\"<?php echo $apporte ?>\" size='50'</td></tr>
<tr><td align='right'>Pour vous inscrire</td>
  <td><input type='submit' value='Envoyer'></td>
  <td align='right'>Pour effacer<br>la saisie</td>
  <td><input type='reset' value='Effacer'</td></tr>
</table>
</form>
<p align='center'><?php echo"$message"; ?></p>";
```

Remarques finales sur ce document

D'abord ce que l'on aurait pu faire ... et que l'on n'a pas fait, mais il y a aussi tout ce que le lecteur peut entreprendre de sa propre initiative :

- Prévoir la possibilité d'apostrophes dans les noms et prénoms : un bon exercice pour le lecteur.
- Ecrire les versions des deux derniers formulaires comme on l'a fait pour le premier.
- Des champs en plus grand nombres avec des champs facultatifs ; à l'aide d'une variable du type de `$manque`, on peut faire retourner le formulaire avec un message demandant si les champs facultatifs non renseignés le sont après réflexion.
- Des groupes de champs dont certains sont obligatoires ; exemple : adresse, fixe et portable dont deux obligatoires ; avec une variable du type de `$manque` qui s'incrémente de 1 à chaque champ vide, on peut faire retouner le formulaire si la variable est inférieure à 2 et cela avec un message approprié.
- Tester chaque champ obligatoire (avec un ensemble de variables du type mentionné ci-dessus) et retouner le formulaire si c'est nécessaire, accompagné d'un message approprié.
- Des champs avec des menus déroulants, des cases à cocher et des boutons radios : encore un bon exercice pour le lecteur (exemple proposé : faire choisir le plat apporté dans une liste de six plats).
- Vérifier que le candidat à l'inscription n'est pas déjà inscrit ; cela revient à adapter et inclure une partie du processeur de recherche d'une inscription (portant uniquement sur le nom et le prénom) dans le processeur d'inscription : encore un exercice pour le lecteur. Cette manière de faire est à la base d'une méthode de prévention du double envoi : l'ensemble de données d'un envoi est stocké par le processeur à sa réception ; si l'ensemble des données suivant est identique à l'ensemble stocké, on peu déduire qu'il s'agit d'un deuxième envoi accidentel : si cette méthode est efficace pour éviter les doubles inscriptions (résultant d'un oubli), elle n'est pas parfaite pour éviter le double envoi (cf. commentaires sur l'empêchement du double envoi, document II).
- Le lecteur peut ensuite amplifier les exemples proposés dans ces documents et en concevoir d'autres du même niveau mais de plus en plus complexes.
- Et puis ... il faut qu'il « mette les pieds dans le plat ». Ces documents lui ont fait connaître une petite partie des bases des langages concernés : il est donc prêt pour « entrer à fond » dans les ouvrages « professionnels » où il retrouvera ce qu'il a appris et beaucoup plus, le tout classé d'une manière logique, rigoureuse, structurée, ... que beaucoup trouvent trop aride pour débuter. Puisque ce présent ouvrage est limité à la version 4 de HTML, il pourra passer à la version 5 avec, par exemple le livre *HTML 5, le guide complet*, L. Guitier et J.-P. Imbert, Micro Application, 2011 ; cette migration serait d'autant plus judicieuse que le Consortium W3C la recommande maintenant.

  Fichier(s) : f5.htm, repacc.htm (page d'accueil), repaff.php et repU430P37Baff.php (processeurs d'affichage des inscriptions), repins.php (et sa variante repins00.php), repmod.php et repmodd.php (formulaires d'incription, de recherche d'inscription et de modification d'inscription), repprins.php repprmod.php et repprmodd.php (processeurs correspondants), valmag.php (pour connaître la valeur de la variable `magic_quote_gpc`) du serveur utilisé.