Programmation web

J'assume Pas

30 octobre 2014

Chapitre 1

HTML

Sachez que ce cours ne fait que reprendre ce qui a été vu et s'inspire grandement de ce cours : *http ://j-willette.developpez.com/tutoriels/html/les-bases-du-html/*.

1.1 Posons les bases

A la base, une page web, ben c'est du texte. Juste du texte structuré.
Ainsi, on utilise des balises pour structurer le texte ; ces balises, c'est le *HTML*.
Eh oui, le *HTML* est figé, dans le sens où dés que le navigateur web a lu une page HTML, il ne se passe plus rien.
C'est pour ça que ce **n'est pas** un language de programmation !

1.1.1 La base de la page

Avant de faire du html et de mettre des jolies balises, il faut d'abord avoir une sorte de squelette obligatoire à chaque page html.
Pas besoin d'en faire des tonnes, le voici :

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">

        <head>
                <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
                <title> Titre de la page </title>
        </head>

        <body>
                <!-- corps de la page -->
        </body>

</html>
```

Et voilà ! Un beau squelette de page html tout prêt à recevoir de belles balises !

1.1.2 De Bonnes Balises

Vous l'aurez certainement compris : le *HTML* repose entièrement sur ce système de balises. Alors, comment ça marche ? Regardons une petite balise random :

```
<html lang="fr">
```

Hop, là on a une balise de nom **html** avec un attribut **lang** prenant une valeur **fr**. On peut enchainer les attributs à la suite, bien_entendu. On ferme une balise à l'aide d'un / avant la fin de la balise. Vous avez compris cet exemple ? Vous avez compris les balises.
Ah ? On me dit que c'est pas fini ? Bon.

Balises-mania Il existe deux niveaux de balises : des balises **simples** et des balises **doubles**.
La différence entre les deux est simple : une balise simple s'ouvre et se ferme dans une même balise alors qu'une balise double a besoin d'une balise **ouvrante** et d'une balise **fermante**.

```
<!-- une balise simple -->
<img src="../images/youpi.jpg" />

<!-- une balise double -->
<p>
        blablabla
</p>
```

Bien sûr, on peut imbriquer des balises, mais dans ce cas il faut faire attention à l'ordre dans lequel on **ouvre/ferme les balises**. On ferme en premier la balise la plus récemment ouverte, logique.

```
<body>
        <p>
                blablabla
        </p>
</body>
```

1.2 Posons les briques : le body

Maintenant qu'on a posé les échafaudages, il est temps de remplir notre page *HTML* !

1.2.1 "Des titres et des paragraphes, what else ?" - Jorge Ben Genre

Un peu comme ce document, c'est plutôt *pas mal* de structurer sa page avec des titres et des paragraphes. Ainsi, il existe en HTML des balises permettant de définir ce qui est titre, sous-titre, sous-sous-titre, paragraphe, etc..
Commençons par les titres.

Des titres On utilise la balise *<hx>* avec x étant un nombre allant de 1 à 6. Plus le chiffre est grand, moins le titre est important.

```
<h1> Un titre très important </h1>
<h6> Un titre dont l'existence pose question </h6>
```

Des paragraphes Pour un paragraphe, c'est très simple : on utilise la balise *<p>*.

```
<p>
        un paragraphe random
</p>
<p>
        un AUTRE paragraphe
</p>
```

Maintenant qu'on a vu comment avoir un texte à peu près structuré, regardons un peu les balises qui nous aideront à structurer davantage la page.

1.2.2 Listes

— Quelque fois, on a envie de mettre des listes sur sa page html
— Des listes numérotées
— Ou encore des listes non numerotés
— Par conséquent, on utilise la balise <ul> ou encore

Des listes non ordonnées Une liste non ordonnée se créé avec les balises *<ul>* - pour *Unordered List* - et se remplit avec des *<li>* - *Listed Item* -.
Imaginons que je veux faire une liste de courses, on a donc :

```
<ul>
        <li> Du chocolol </li>
        <li> Du Super-Timor </li>
        <li> Le dernier CD de miko black </li>
        <li> des brocolis </li>
<ul>
```

Et voilà, on a une jolie liste non ordonnée.

Des listes numérotées Le principe est le même que pour une liste non structurée, sauf qu'on utilise *<ol>* - pour *Ordered List* -.
Maintenant, je fais une liste de tâches selon l'importance des tâches :

```
<ol>
        <li> Regarder des vidéos de chat </li>
        <li> Taunter bruno </li>
        <li> Finir Persona 3 </li>
        <li> Finir la fiche de système </li>
<ol>
```

ET C TOU.

Bien sur, on peut imbriquer les listes, et faire des listes dans l'item d'une liste, par exemple :

```
<ol>
        <li>Regarder des vidéos de chat</li>
        <li>Taunter bruno <!-- La balise </li> n'est pas encore placée ! -->
                <ul>
                        <li>Faire des paints rigolols</li>
                        <li>Les poster sur big_bisous</li>
                        <li>Spammer son mur</li>
                </ul>
         </li>                  <!-- Voici la balise </li> ! -->
         <li>Finir Persona 3</li>
         <li>Finir la fiche de système</li>
</ol>
```

Et hop, maintenant j'ai une liste non ordonnée dans ma liste ordonnée. *"Étonnant, non ?"*
Mais notre site est un peu fade sans images non ? Ajoutons quelques images.

1.2.3 Des images

Une image vaut mieux que mille mots...

un Putain de Random

C'est pas compliqué pour une image : il faut utiliser la balise *<img>*. Attention cependant, cette balise - simple, d'ailleurs - prend **obligatoirement** deux attributs : *src* et *alt*.

src désigne le chemin vers l'image et *alt* ce qui s'affichera si l'image ne se charge pas. En pratique, voilà comment ça s'utilise :

```
<img src="chemin/vers/image" alt="description de l'image" />
```

Bon c'est bien joli tout ça, mais notre page mène nul part ; alors rajoutons quelques liens !

1.2.4 Des liens

On utilise ici la balise *<a>* avec l'attribut *href*, attribut qui désigne le lien vers lequel on veut aller.

```
<a href="autrePage.html"> un lien vers mon autrePage </a>
```

On peut imbriquer la balise *<a>* avec une image, pour faire une image cliquable :

```
<a href="encoreUneAutrePage.html"> <img src="image_cool.jpg" alt="une image cool" /> </a>
```

1.2.5 Un peu d'interaction : les formulaires

L'internaute qui surfera sur votre page sera peut être un peu frustré de ne pas avoir son mot à dire. Ainsi, pour satisfaire ses besoins d'expression, il existe des **formulaires**.
Tous les champs de formulaires devront se trouver dans une balise *<form>*. *<form>* prend deux attributs, *<action>*, qui définit la page de destination du formulaire, et *<method>*, qui définit comment l'infomation est envoyée.
On peut envoyer soit par **GET** - les informations sont dans l'url - , soit par **POST** - les informations envoyées n'apparaissent pas dans l'url -. Par souci de sécurité, on utilisera plutôt **POST**. Pour résumer, on met nos formulaires dans ce squelette :

```
<form action="envoi-formulaire.php" method="post">
    <!-- Contenu du formulaire -->
</form>
```

Il existe plusieurs types de formulaires :
— les champs de texte sur 1 ligne
— les champs de texte sur plusieurs lignes
— les boutons radio
— les cases à cocher
— des listes déroulantes
— des boutons tout court / le bouton envoyer

Il faut savoir que les champs de texte monoligne, les boutons radio, les cases à cocher et les boutons tout court utilisent la même balise, *<input>*. La différence tient en fait dans l'attribut *type*.

Aussi, sachez que qu'il vaut mieux donner un nom - attribut *name* - et un id - attribut *id* - à chaque formulaire. Cela permettra de récupérer par la suite la valeur et de faire des modifications.

De plus, n'hésitez pas à mettre des **labels** à vos formulaires. Un label s'utilise de la manière suivante :

```
<!-- On a ici un champ de texte pour récuperer le nom -->
<label for="nom">Nom :</label>
<input type="text" name="nom" id="nom" >
```

Maintenant, regardons les formulaires que nous avons à notre disposition.

Champs de texte monoligne : Ce sont les *<input type="text">*. Y a pas grand chose à savoir sur ces trucs. AH SI, y a pas de "/" à la fin de la balise.

```
<input type="text" name="nom" id="nom >
```

Champs de texte sur plusieurs lignes : Ce sont les *<textarea>*. On peut spécifier le nombre de colonnes - cols - et le nombre de lignes - rows -.

```
<!-- on a ici un bloc de champ de texte de 20 lignes
        et 50 colonnes -->
<textarea name="demande" rows="20" cols="50"> </textarea>
```

Les boutons radio : Ce sont les *<input type="radio">*. Il faut donner le même nom à chaque groupe de bouton radio et une valeur différente à chaque bouton radio. (attributs *name* et value respectivement).

```
<!-- on ne peut faire qu'un seul choix -->
    <input type="radio" name="civilité" value="mlle"> Mademoiselle </input>
    <input type="radio" name="civilité" value="mme"> Madame </input>
    <input type="radio" name="civilité" value="mr"> Monsieur </input>
```

Les checkbox (cases à cocher) : Ce sont les *<input type="checkbox">*. Rien de spécial.

```
<input type="checkbox" name="documents"> Demande de documents </input>
```

Les listes déroulantes : Ce sont les *<select>*. Les *<option>* désignent les choix possibles.

```
<label for="departement">Dans quel département habitez-vous? </label>
<select name="departement" id="departement">
        <option value="essonne">Essonne</option>
        <option value="paris">Paris</option>
        <option value="yvelines">Yvelines</option>
        <option value="autre">Autre</option>
</select>
```

Les boutons : On s'intéressera ici au bouton d'envoi. C'est *<input type="submit">*. C'est tout.

```
<input type="submit" name="envoyer"> </input>
```

1.2.6 Ranger des informations de manière jolie : les tableaux

Quelques fois, les tableaux c'est pas mal pour trier et ranger des informations, mais **attention**, il ne faut surtout pas utiliser les tableaux pour faire du **formatage de page** ! Si vous voulez mettre en forme une page, attendez le *CSS*.
Bon. Un tableau est délimité par des balises *<table>*. En *HTML*, on gère le contenu d'un tableau ligne par ligne, il faut donc utiliser la balise *<tr>* pour délimiter une ligne ; ensuite, on utilise *<td>* pour délimiter une colonne. C'est seulement entre les balises *<td>* que l'on peut écrire du contenu. Résumons :

```
<table border="1">
        <!-- ma premiere ligne -->
        <tr>
                <td> Colonne 1, Ligne 1 </td>
                <td> Colonne 2, Ligne 1 </td>
        </tr>
        <!--- ma 2e ligne -->
```

```
        <tr>
                <td> Colonne 1, Ligne 2 </td>
                <td> Colonne 2, Ligne 2 </td>
        </tr>
</table>
```

Je pense que c'est tout pour les tableaux.

1.2.7 Diviser pour mieux régner : les blocs/div

Pour mieux organiser le code mais aussi afin de rendre l'application de styles CSS plus facile [1], on peut diviser la page html en divers blocs : c'est ce qu'on appelle des **div**.
Ainsi, on peut diviser sa page html en plusieurs parties ; par exemple, dans le TD1, on faisait la différence entre le menu -* *-, la bannière - * * - et tout ce qui était saisie - * * et j'en passe -. Voilà comment on utilise * * :

```
<div class="nomDuBloc" >
        <!-- le contenu du bloc -->
</div>
```

Pour l'instant, ça n'a pas grand interêt, mais passons au CSS.

1. Vous inquietez pas, ça arrive bientôt

Chapitre 2

CSS : "Mon Style"

"Regarde Mon Style, écoute Mon Style"

- un trètrè bon rappeur

CSS signifie *Cascade StyleSheet*, soit *Feuille de Style en Cascade*. Cela veut dire que la feuille est appliquée au fur et à mesure ; ce qui est en dernier est donc appliqué en dernier.
Mais avant de commencer à faire du *CSS*, il faut savoir qu'il faut l'inclure dans la page html où l'on souhaite faire de la mise en forme.
On l'inclut de la manière suivante, dans l'entete du fichier html (c'est à dire entre <head> </head>) :

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

2.1 Éléments stylisables

Un fichier CSS se présente sous la forme suivante :

```
/* désigne le body */
body {
        font-family : Arial, Helvetica, sans-serif;
        font-size :10px;
}

/* le . désigne une <div> */
.bloc_page {
    width : 950px;
    margin : auto;
    border : solid;
}

/* désigne les listes non ordonnées en général */
ul {
    display : inline;
    margin-right : 15px;
}

/* désigne les listes ordonnées dans la div bloc_page */
.bloc_page ol{
   display : inline;
}
```

Bon, j'avoue que c'est pas très interessant tout ça. Regardez la cheatsheet et le tutoriel de developpez.com, ça suffira.

Chapitre 3

PHP Done Quick

On ne résumera ici que ce qui a été vu en cours & td. *soz* si je zappe des trucs.

3.1 La syntaxe

Le but du PHP est d'apporter du dynamisme à une page HTML. Ainsi, le PHP est un véritable langage de programmation.
Comme tout langage de programmation, le PHP possède des variables, des expressions conditionnelles, etc. Vous êtes tous des pros de la programmation, alors faisons juste un point sur la syntaxe PHP.

3.1.1 Où est-ce que j'écris du PHP ?

Dans une page html, entre des balises *<php>*. Regardez l'exemple :

```
<html>
    <head>
        <title>Ma page d'accueil </title>
    </head>
    <body>
        <h1>Bienvenue sur le site de toto</h1>
        <p> Quelle est la date?</p>
                <?php
                        echo '<p>On est le ' .date("l").'</p>';
                ?>
    </body>
</html>
```

3.1.2 Les variables

Une variable est désignée par **$**. *$moi* est donc une variable avec comme nom *moi*.
Contrairement aux langages comme C ou java, on ne type pas la variable lors de la déclaration. De plus, on doit mettre **$** à **chaque fois** que l'on souhaite utiliser une variable.

```
<?php
        $moi='ouam';
        $deux=2;
        $quatre=4;
        $res= $deux + $quatre;
?>
```

3.1.3 *echo* et concaténation de variables

Avec du PHP, et c'est bien pratique, on peut faire de l'HTML directement dans du PHP. Par exemple, dans l'exemple un peu plus en haut, lorsque on fait :

```
<?php
        echo '<p> On est le ' . date("1") .'</p>';
?>
```

On a bien un paragraphe avec le contenu de notre variable. En fait, on exprimera tout ce qui est html entre des quotes - ' - et on concatène à cela le contenu nos variables. Par exemple, si je veux afficher un nom dans une page html seulement avec du code PHP, je peux faire :

```
<?php
        $nom="ouam";
        $loisir="ouej";
        echo '<p> Bonjour, je suis '. $nom . ' ! J'aime bien '. $loisir '</p>';
?>
```

ET C TOU.
Pour résumer, on concatène notre html entre quotes avec des variables à l'aide de points.

3.2 Formulaires HTML et récupération en PHP

3.2.1 Récupération de données d'un formulaire

Vous vous souvenez des formulaires qu'on remplissait en HTML ? Eh ben on va récupérer les valeurs saisies, super non ?? En fait c'est très simple ; déjà, on regarde de quelle manière on a envoyé nos informations (GET ou POST), ensuite on récupère dans une variable cette information :

```
<?php
        //on a envoyé nos informations par POST
        $info=$_POST['nomDeLaVariable'];
        echo $info;
?>
```

Rappelez vous : dans la partie HTML, on donnait un attribut *name* à chaque input. C'est cette valeur qu'il faut mettre dans le *$_POST['nomDeLaVariable']*[1].

Enfin, il faut bien vérifier si l'utilisateur a bien mis une valeur à la variable ; on ne cherche pas à récupérer du *rien*. Dans ce cas, on utilise la condition *isset(variable)* :

```
<?php
        if(isset($_POST['nomDeLaVariable']))
        {
                echo $_POST['nomDeLaVariable'];
        }
        else
        {
                echo 'Alors espèce de petit malin, on ne remplit pas le formulaire?????';
        }
?>
```

3.3 Expressions régulières et vérification de données

Ah, les expressions régulières. Il y a des utilisateurs réguliers de Linux ici ? Non ? bon.
Les expressions régulières permettent en fait de vérifier, ou de rechercher des patterns dans des chaines de

1. ou $_GET

caractère, par exemple vérifier si une adresse est bien une adresse mail, si le lien est un lien http :, etc... Avant de voir comment utiliser les expressions régulières en PHP, apprenons à les utiliser tout court.

3.3.1 Expressions régulières

Dans le cours, on a utilisé des expressions régulières de type PCRE ; on va donc en faire de même ici.

Trouver un mot Une expression régulière est séparée par des #.

```
#mot#
```

Par exemple, l'expression régulière ci-dessus va vérifier si le mot *mot*ăse trouve dans la chaine de caractère. On peut appliquer des |, qui correspondent à un OU logique.

```
#mot|oui|merci#
```

Cette expression va vérifier si le mot *mot*, ou *oui*, ou *merci* se trouve dans la chaine de caractères.

Positionnement du mot On peut vérifier la position de l'expression grâce aux symboles ^ et *$*. le ^ vérifie si l'expression qui suit se trouve au début de la chaine de caractères, alors que le *$* vérifie si l'expression qui précède se trouve en fin de la chaine.

```
#^bonjour#
// "bonjour toi" est vrai
// "toi bonjour" est faux
#non$#
// "j'ai dit non" est vrai
// "non non j'ai dit" est faux
```

3.4 Connexion à une base de données et requêtes SQL

3.4.1 Connexion à une base de données

```
<?php
        $host = "localhost";
        $user = "root";
        $bdd = "td7"; // le nom de votre base de données
        $passwd = "";

        $co = mysqli_connect($host , $user , $passwd, $bdd) or die("erreur de connexion");
?>
```

3.4.2 Faire une requête

Après s'être connecté, on a notre connexion dans la variable $co. Pour faire une requete, il faut utiliser *mysqli_query( )*. Cette fonction prend en paramètre une connexion ($co) et la requete que l'on souhaite faire, sous la forme d'une chaîne de caractères.

```
<?php
        $res = mysqli_query($co, "SELECT truc FROM machin WHERE ...);
?>
```

Note : on peut aussi faire de la concaténation de chaines, comme vu plus haut.
On peut aussi faire des INSERT, et là, un petit rappel de la syntaxe s'impose :

```
INSERT INTO [table auquel on souhaite faire un insert] (colonne1,colonne2,..)
              VALUES (<valeur pour col1>, <valeur pour col2>)
```

Et voilà.

3.4.3 Récupérer les résultats d'une requête

Faire des requetes c'est bien, récuperer les résultats c'est mieux.
En fait, on va récupérer les résultats dans un tableau, grâce à *mysqli_fetch_assoc( ).*
En reprenant l'exemple ci dessus, on a donc :

```
<?php
        $res = mysqli_query($co, "SELECT truc FROM machin WHERE ...);
        $row = mysqli_fetch_assoc($res);

        // on a maintenant notre résultat dans le row
        $truc = $row['truc'];
?>
```

3.5 Failles de sécurité

3.5.1 Failles XSS

Une faille XSS profite de la fonction *echo( )* et d'un formulaire pour afficher du code HTML non voulu, par exemple un pop-up.
Pour bloquer ces failles, il existe 3 fonctions php :

- htmlspecialchars() transforme les caractères spéciaux en texte tout court
- htmlentities() transforme aussi les balises html en texte tout court
- strip_tags() supprime carrément les balises.

3.5.2 Failles SQL

Une injection SQL va essayer d'introduire, voir meme de détruire des données dans une base de données. Là aussi, on profite d'un formulaire pour modifier une requête SQL et en faire ce que l'on veut. Là aussi, on va utiliser une fonction pour rendre la requete sécurisée ; c'est la fonction *mysqli_real_escape_string()*. On met notre chaine de caractères en parametre et on récupere la chaine de caractère. youpi
super
trop cool