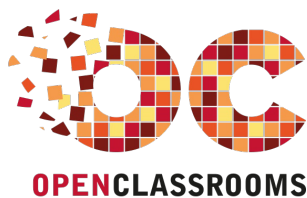


Les captchas anti-bot

Par nax



www.openclassrooms.com

*Licence Creative Commons 7 2.0
Dernière mise à jour le 14/11/2010*

Sommaire

Sommaire	2
So-pix.fr site web de graphisme ma brute WIP Play Station 2 Slim	1
Les captchas anti-bot	3
Partie 1 : Créer un captcha	3
Les captchas textuels	4
Recopier un mot	4
L'organisation du script	4
Le formulaire	4
L'algorithme du captcha	4
Vérification	5
Recopier un mot (dictionnaire)	6
Recopier un mot aléatoire	6
Une séquence de caractères aléatoires	7
Une séquence de lettres	7
Une question mathématique	8
Les captchas graphiques	9
Une image simple	10
La chaîne de caractères	10
Fonction de génération de l'image	11
Une image masquée	12
Les bandes noires	12
Un fond hachuré	13
Une image utilisant une police ttf	13
Écrire le captcha dans une police ttf	13
Matrice de convolution	15
Les matrices de convolution	15
En PHP, ça donne quoi ?	15
Déformation	17
Rotation des lettres	17
Accessibilité : créer un fichier son	19
Introduction à la manipulation de fichiers binaires	20
Les fichiers binaires	20
Découverte du format .wav	21
Obtenir la structure du fichier	21
Un exemple de fichier .wav	22
Lecture d'un fichier binaire	23
Lecture d'un fichier .wav	25
isWav	25
getHeader	26
getData	27
Écriture d'un fichier .wav	27
listWav	28
toWav	28
Intégration dans notre script de captcha	30
formulaire.php	30
captcha.php	30
wav.php	30

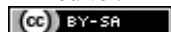


Les captchas anti-bot



Mise à jour : 14/11/2010

Difficulté : Difficile



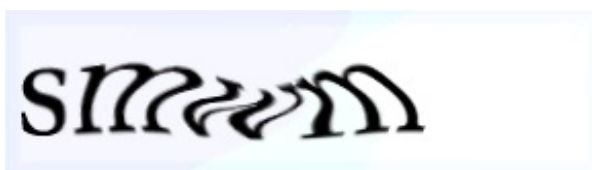
Votre site est envahi par des inscriptions de *bots* (des robots qui s'inscrivent plusieurs fois à des sites dans le but de publier des messages publicitaires sur un livre d'or ou un forum, ou pour utiliser des comptes mail) ?

Il existe de nombreuses solutions pour empêcher ces bots d'agir. Dans ce tutoriel, vous apprendrez à en coder quelques-unes en PHP.

En plus de vous apprendre à créer des *captchas*, ce cours va vous apprendre - je l'espère - à utiliser des outils méconnus de PHP et des notions informatiques un peu plus complexes. 😊 Dans le premier chapitre, vous apprendrez à réaliser un captcha textuel sous la forme d'une question, puis vous découvrirez comment manipuler le hasard avec PHP et générer des chaînes de caractères aléatoires.

Dans le deuxième chapitre, nous nous concentrerons sur la bibliothèque GD et sur quelques fonctions intéressantes : utiliser une police *true type*, créer des effets tels que des rotations et des flous avec des matrices de convolution, etc.

Enfin, dans le troisième chapitre, nous aborderons la manipulation de données binaires : d'abord la théorie, puis la pratique avec PHP.



Partie 1 : Créer un captcha

Un **captcha** est un test destiné à différencier un humain d'une machine.

Ces tests, généralement présentés sous la forme d'une image déformée ou d'une question, sont utilisés sur Internet dans les formulaires pour éviter les envois par des robots.

Ils sont utilisés contre le spam, dans les forums ou dans les commentaires d'articles de blogs, dans les formulaires d'inscription, contre les tentatives d'attaque par force brute et également contre la participation automatique à des sondages.

Les captchas textuels

Un captcha textuel est un système qui demande une information à l'utilisateur à partir d'une question que seul un humain peut comprendre.

Par exemple : « *combien font 3 plus 5 ?* » Même si un ordinateur pourrait faire le calcul, on suppose qu'il ne comprend pas la question.

Voici quelques exemples :

- « Recopiez le mot *maison* » ;
- « Entrez la somme de 5 et 6 ».

Recopier un mot

Dans cette première partie, nous allons réaliser un captcha qui demande à l'utilisateur de taper un mot.

Vous pouvez voir un exemple sur le formulaire du forum ubuntu-fr : <http://forum.ubuntu-fr.org/>.

L'organisation du script

Nous allons utiliser trois fichiers PHP :

- [formulaire.php](#) (le formulaire) ;
- [verification.php](#) (la page de vérification) ;
- [captcha.php](#) (le fichier pour l'algorithme du captcha).

Le formulaire

Il s'agit d'un formulaire HTML tout simple avec un champ text pour le captcha.

Code : HTML

```
<form action="verification.php" method="post">
  <p>
    <label for="nom">Votre nom</label>
    <input type="text" name="nom" id="nom" /><br />

    <label for="captcha">Recopiez le mot : ""</label>
    <input type="text" name="captcha" id="captcha" /><br />

    <input type="submit" value="envoyer" />
  </p>
</form>
```

Nous écrirons le mot une fois l'algorithme du captcha réalisé.

L'algorithme du captcha

Nous avons deux possibilités : générer un mot au hasard ou en sélectionner un parmi une liste prédéfinie. Bien que la seconde solution soit la plus facile, nous verrons les deux.

Un mot parmi une liste

La liste de mots sera un tableau. Pour choisir aléatoirement un mot de ce tableau, on utilise la fonction `array_rand()`.

Code : PHP

```
<?php
function motListe()
{
    $liste = array('internet', 'captcha', 'robot');
    return $liste[array_rand($liste)];
}

function captcha()
{
    return motListe();
}
?>
```

Il faut maintenant afficher ce mot dans `formulaire.php` et l'enregistrer quelque part pour que `verification.php` puisse vérifier que l'utilisateur a saisi le bon mot.

On utilise les sessions pour transmettre le mot d'une page à l'autre.

Il faut penser à ajouter `<?php session_start(); ?>` en haut de chaque page (`formulaire.php` et `verification.php`). Le fichier `captcha.php` étant inclus dans `formulaire.php`, la session sera déjà initialisée et il n'y aura donc pas besoin d'ajouter cette ligne de code.

Dans formulaire.php

On inclut la page `captcha.php` avec `<?php require('captcha.php'); ?>` et on écrit le mot dans le label avec :

Code : PHP

```
<label for="captcha">Recopiez le mot : "<?php echo captcha(); ?>
"</label>
```

Dans captcha.php

Il faut penser à sauvegarder le mot dans une session ; on modifie donc la fonction `captcha()` :

Code : PHP

```
<?php
function captcha()
{
    $mot = motListe();
    $_SESSION['captcha'] = $mot;
    return $mot;
}
?>
```

Vérification

Maintenant, nous allons nous occuper de `verification.php`.

Il s'agit juste de vérifier que `$_POST['captcha']` est égale à `$_SESSION['captcha']`.

Voici le code de `verification.php` :

Code : PHP

```
<?php
if(!empty($_POST['captcha']) && !empty($_POST['nom']))
{
    if($_POST['captcha'] == $_SESSION['captcha'])
        echo 'Le captcha est bon, votre nom est ' . $_POST['nom'];
    else
        echo 'Le captcha n\'est pas bon.';
}
else
    echo 'Il faut remplir tous les champs.';
?>
```



Pensez à appeler la fonction `session_start()` au début du script !

Exemple

Recopier un mot (dictionnaire)

Au lieu de saisir vous-même la liste de mots, il peut être amusant d'en choisir un au hasard dans un dictionnaire.

Il suffit juste de trouver un fichier texte « dictionnaire ». Pour cela, je vous conseille [Google](#).

Pour ma part, je prends ce dictionnaire `dic-iso.text` (version UTF-8 : `dico-utf8.text`).

Ensuite, il suffit de modifier dans le fichier `captcha.php` la fonction `motListe()`.

Code : PHP

```
<?php
function motListe()
{
    $liste = file('dico.text');
    return trim($liste[array_rand($liste)]);
}
?>
```

Il est nécessaire d'utiliser la fonction `trim()` car le mot contient le `\n` de fin de ligne.

Exemple

Recopier un mot aléatoire

Nous allons, dans cette partie, demander au visiteur de recopier un mot non pas choisi dans une liste, mais généré au hasard. On va donc écrire une nouvelle fonction dans `captcha.php` : `motHasard($n)`.

Cette fonction génère un mot au hasard de `$n` caractères.

Il existe des dizaines (des centaines ?) de méthodes pour générer un mot au hasard ; nous allons en étudier quelques-unes, c'est

assez amusant. 😊

Une séquence de caractères aléatoires

Une méthode utilise les fonctions de [hash](#). Bien que ce ne soit pas leur but premier, ces fonctions permettent d'obtenir des séquences de caractères aléatoires si l'on passe en paramètre une chaîne aléatoire.

Bon, on tourne un peu en rond puisqu'il nous faut une chaîne aléatoire au départ ! Mais PHP propose plein de fonctions pour cela :

- `uniqid` ;
- `mt_rand` ;
- `microtime` ;
- ...

Modifions notre fichier `captcha.php` pour obtenir ceci :

Code : PHP

```
<?php
function motHasard($n)
{
    // Séquence aléatoire
    return substr(md5(uniqid()), 0, $n);
}

function captcha()
{
    $mot = motHasard(6);
    $_SESSION['captcha'] = $mot;
    return $mot;
}
?>
```

Bien sûr, vous pouvez utiliser d'autres combinaisons dans la fonction `motHasard()`. En voici quelques-unes :

Code : PHP

```
<?php
function motHasard($n)
{
    // Séquence aléatoire (en choisir une)
    $mot = substr(md5(uniqid()), 0, $n);
    $mot = substr(sha1(rand()), 0, $n);
    $mot = substr(strrev(time()), 0, $n);
    $mot =
    substr(str_pad(next(explode('.', microtime(true))), $n, '0'), 0, $n);
    $mot = str_pad(mt_rand(0, pow(10, $n)-1), $n, '0');
    return $mot;
}
?>
```

Une séquence de lettres

Dans la partie précédente, nous avons surtout travaillé avec des chaînes de chiffres (ou de caractères hexadécimaux). Maintenant, nous allons voir comment générer une chaîne de caractères alphabétiques aléatoires.

Il s'agira uniquement de modifier la fonction `motHasard()`.

On peut, par exemple, piocher au hasard dans un tableau de caractères :

Code : PHP

```
<?php
function motHasard($n)
{
    $lettres =
    array_merge(range('a','z'),range('A','Z'),range('0','9'));
    $nl = count($lettres)-1;
    $mot = '';
    for($i = 0; $i < $n; ++$i)
        $mot .= $lettres[mt_rand(0,$nl)];
    return $mot;
}

?>
```

Maintenant, nous allons coder une fonction qui donne un mot « prononçable », c'est-à-dire dans lequel les voyelles sont suivies de consonnes.

La subtilité est de définir un tableau de voyelles et un de consonnes.

Code : PHP

```
<?php
function motHasard($n)
{
    $voyelles = array('a', 'e', 'i', 'o', 'u', 'ou',
    'io', 'ou', 'ai');
    $consonnes = array('b', 'c', 'd', 'f', 'g', 'h', 'j', 'k', 'l',
    'm', 'n', 'p', 'r', 's', 't', 'v', 'w',
    'br', 'bl', 'cr', 'ch', 'dr', 'fr', 'dr', 'fr', 'fl',
    'gr', 'gl', 'pr', 'pl', 'ps', 'st', 'tr', 'vr');

    $mot = '';
    $nv = count($voyelles)-1;
    $nc = count($consonnes)-1;
    for($i = 0; $i < round($n/2); ++$i)
    {
        $mot .= $voyelles[mt_rand(0,$nv)];
        $mot .= $consonnes[mt_rand(0,$nc)];
    }

    return substr($mot,0,$n); // Comme certaines syllabes font plus
    d'un caractère, on est obligé de couper pour avoir le nombre exact
    de caractères.
}

?>
```

Exemple

Maintenant, à vous de choisir la fonction que vous préférez. L'avantage est qu'avec l'agencement des fichiers que nous avons défini au début ([formulaire.php](#), [captcha.php](#), [verification.php](#)), vous pouvez générer n'importe quel type de captcha, il n'y aura qu'une seule fonction à changer (la fonction captcha()). Vous pouvez donc utiliser ce script dans n'importe lequel de vos sites (voire l'ajouter à un formulaire déjà existant).

Une question mathématique

Dans cette partie, nous allons demander à l'utilisateur de répondre à une question mathématique.

Par exemple : « *combien font deux plus huit ?* » Malgré l'apparente simplicité de cette question, elle permet d'éliminer les robots

car ils ne peuvent pas *comprendre* le sens de la question.

On génère deux nombres au hasard, on calcule leur somme, et on affiche une question en toutes **lettres**.

Voici une idée de code :

Code : PHP

```
<?php
function captchaMath()
{
    $n1 = mt_rand(0,10);
    $n2 = mt_rand(0,10);
    $nbrFr =
    array('zero', 'un', 'deux', 'trois', 'quatre', 'cinq', 'six', 'sept', 'huit', 'neuf', 'dix');
    $resultat = $n1 + $n2;
    $phrase = $nbrFr[$n1] . ' plus ' . $nbrFr[$n2];

    return array($resultat, $phrase);
}

function captcha()
{
    list($resultat, $phrase) = captchaMath();
    $_SESSION['captcha'] = $resultat;
    return $phrase;
}
?>
```

J'utilise la structure de langage `list()` que vous n'avez peut-être pas encore vue : elle transforme un tableau en une liste de variables.

Il n'y a pas besoin de modifier le fichier `verification.php`.

Dans le fichier `formulaire.php`, vous pouvez remplacer :

Code : PHP

```
<label for="captcha">Recopiez le mot : "<?php echo captcha(); ?>"</label>
```

par :

Code : PHP

```
<label for="captcha">Combien font <?php echo captcha(); ?></label>
```

Sur cette base de code, vous pouvez inventer plein de types de captchas. 😊

Exemple

Maintenant que nous avons vu comment réaliser un captcha textuel, nous allons nous intéresser aux images.

Mais avant cela, sachez que les captchas textuels sont les plus simples à réaliser... et à contourner.

Lisez cet article qui illustre bien le problème : [http://www.seoblackout.com/2007/12/30/\[...\]ions-calculs/](http://www.seoblackout.com/2007/12/30/[...]ions-calculs/).

Les captchas graphiques

Dans ce chapitre, nous allons créer une image anti-bot (un captcha) : ce chapitre utilise l'architecture de fichiers fournie dans le chapitre *Les captchas textuels*. Il est donc important d'avoir lu au moins la première partie.

Les captchas *graphiques* sont très difficilement lisibles pour un robot : très peu arrivent à identifier les caractères, surtout s'ils sont déformés ou en partie masqués.

Nous utiliserons tout au long de ce tutoriel la [bibliothèque GD](#). Le tuto officiel du site sur cette bibliothèque devrait vous aider à l'installer et à l'utiliser (je vous recommande cependant de lire un peu la [doc](#)).



Voici la première et la dernière image que nous allons créer.



Mise en garde : les captchas présentés ici ne sont pas très résistants face aux robots et ne feront pas le poids face à un robot développé exprès contre eux, gardez cela à l'esprit. Le but de ce tuto est avant tout de vous enseigner comment réaliser un captcha, pas d'en fournir un à toute épreuve. Le dernier captcha réalisé est toutefois plutôt efficace.

Une image simple

Dans un premier temps, nous allons réaliser une image simple contenant une certaine chaîne de caractères. La chaîne de caractères sera générée par une fonction que nous avons vue dans le premier chapitre, nous nous concentrerons ainsi sur l'image.

La chaîne de caractères

On utilisera la fonction :

Code : PHP

```
<?php
function nombre ($n)
{
    return str_pad(mt_rand(0,pow(10,$n)-1) , $n, '0', STR_PAD_LEFT) ;
}
?>
```

Renseignez-vous sur les fonctions utilisées ([str_pad](#), [mt_rand](#) et [pow](#)) si besoin.

La différence avec nos autres scripts (chapitre 1) est que [captcha.php](#) génère maintenant une image, il faut donc le modifier comme ceci :

Code : PHP

```
<?php

function image ($mot)
{
}

function nombre ($n)
{
    return str_pad(mt_rand(0,pow(10,$n)-1) , $n, '0', STR_PAD_LEFT) ;
}

function captcha ()
{
    $mot = nombre(5) ;
```

```
$_SESSION['captcha'] = $mot;
image($mot);
}

header("Content-type: image/png");
captcha();
?>
```

Fonction de génération de l'image

On crée une fonction **image(\$mot)** qui prend en paramètre la chaîne de caractères à écrire dans l'image. Cette fonction générera l'image et l'affichera.

Pour connaître la largeur de notre image, on va utiliser la taille du mot, en admettant qu'un caractère fait 10 pixels de largeur (en comptant les espaces et les marges).
Pour la hauteur, on utilisera une image de 20 pixels.

Notre fichier ressemble donc à cela :

Code : PHP

```
<?php
function nombre($n) {
    // [...]
}

function image($mot)
{
    $largeur = strlen($mot) * 10;
    $hauteur = 20;
    $img = imagecreate($largeur, $hauteur);
    $blanc = imagecolorallocate($img, 255, 255, 255);
    $noir = imagecolorallocate($img, 0, 0, 0);

    imagepng($img);
    imagedestroy($img);
}

function captcha()
{
    $mot = nombre(5);
    $_SESSION['captcha'] = $mot;
    image($mot);
}

header("Content-type: image/png");
captcha();
?>
```

Il faut maintenant écrire le mot sur l'image, grâce à la fonction **imagestring()**.

Pour centrer le mot dans l'image, il faut réfléchir un peu. 😊 La fonction **imagestring()** a besoin des coordonnées (abscisse et ordonnée) auxquelles sera placé le texte. Cette position correspond au coin supérieur gauche de notre mot.

Pour l'ordonnée (axe vertical), on prend donc le milieu de l'image moins la hauteur d'un caractère divisée par deux. Pour la position horizontale, c'est un peu plus compliqué.

J'ai trouvé empiriquement que `<?php strlen($mot) / 2; ?>` était une abscisse acceptable.

On trace aussi une bordure extérieure pour rendre notre captcha plus joli avec la fonction **imagerectangle()**.

Code : PHP

```
<?php

function image($mot)
{
    $largeur = strlen($mot) * 10;
    $hauteur = 20;
    $img = imagecreate($largeur, $hauteur);
    $blanc = imagecolorallocate($img, 255, 255, 255);
    $noir = imagecolorallocate($img, 0, 0, 0);
    $milieuHauteur = ($hauteur / 2) - 8;
    imagestring($img, 6, strlen($mot) / 2, $milieuHauteur, $mot,
    $noir);
    imagerectangle($img, 1, 1, $largeur - 1, $hauteur - 1, $noir); //
    La bordure

    imagepng($img);
    imagedestroy($img);
}
?>
```

Il faut maintenant adapter un peu [formulaire.php](#). Il ne faut plus inclure [captcha.php](#), mais l'utiliser comme l'adresse de l'image.

Code : HTML

```
<label for="captcha">Recopiez le mot : </label>
```



Il faut donc mettre `<?php session_start(); ?>` en haut de [captcha.php](#) et enlever `<?php require('captcha.php'); ?>` de [formulaire.php](#).

Exemple

Nous avons maintenant un captcha fonctionnel, bien que sa sécurité soit un peu faible.

Une image masquée

Les captchas graphiques sont un peu plus sûrs que les captchas textuels, mais il faut impérativement masquer le texte ou le déformer, car les robots arrivent de plus en plus à décoder les images.

Nous allons donc masquer notre image en y ajoutant des bandes noires horizontales et un fond hachuré.

Les bandes noires

Pour ajouter des bandes noires, on utilise la fonction `imageline()`. On va dessiner une barre aléatoire et une barre standard. À vous d'adapter le script pour obtenir ce que vous voulez.

Pour la barre aléatoire, on choisit deux hauteurs au hasard et on trace la ligne d'un bout à l'autre de l'image. La barre standard coupera l'image en deux.

Code : PHP - barre standard

```
<?php imageline($img, 2, $milieuHauteur + 8, $largeur - 2,
    $milieuHauteur + 8, $noir); ?>
```

Code : PHP - barre aléatoire

```
<?php imageline($img, 2,mt_rand(2,$hauteur), $largeur - 2,
mt_rand(2,$hauteur), $noir); ?>
```

Exemple

Un fond hachuré

Pour obtenir un fond hachuré, on utilisera une boucle. Il suffit juste d'une variable x que l'on incrémentera jusqu'à ce qu'elle atteigne la largeur de l'image.

Code : PHP - Le Fond Hachuré

```
<?php
// Le fond hachuré
for($x = 5; $x < $largeur; $x+=6)
{
    imageline($img, $x,2,$x-5,$hauteur,$noir);
}
?>
```

Exemple

Vous pouvez facilement adapter ces scripts en modifiant quelques valeurs et la couleur pour avoir des captchas originaux.

Une image utilisant une police ttf

Nous allons maintenant utiliser une police ttf à la place de la police par défaut. Vous pouvez prendre une police plus difficile à lire par un robot.

Tout d'abord, téléchargez une police ttf sur Internet : une police manuscrite est conseillée car elle est plus difficilement reconnaissable par un robot. Vous pouvez aller voir sur Dafont.com. Je vais utiliser la police [Smartie](#).



Faites attention aux licences des polices, certaines interdisent une utilisation commerciale.

Écrire le captcha dans une police ttf

Il faut tout d'abord que la police (le fichier ttf) se trouve dans le même dossier que [captcha.php](#).

Nous allons utiliser une fonction très pratique, [imagegettfbbox](#), qui nous retournera la taille du bloc de texte écrit avec la police. Cette fonction nous permet de calculer plus facilement les dimensions de l'image.

J'utilise [imagegettfbbox](#) comme cela :

Code : PHP - imagegettfbbox

```
<?php
$size = 32;
$box = imagegettfbbox($size, 0, './smartie.ttf', $mot);
?>
```

La [doc](#) nous indique ce que contient le tableau que retourne `imagettfbbox()` :

Citation : Doc PHP

`imagettfbbox()` returns an array with 8 elements representing four points making the bounding box of the text:
0 lower left corner, X position
1 lower left corner, Y position
2 lower right corner, X position
3 lower right corner, Y position
4 upper right corner, X position
5 upper right corner, Y position
6 upper left corner, X position
7 upper left corner, Y position

Étant donné que je n'ai pas indiqué d'angle, nous n'avons besoin que de quatre valeurs pour calculer la largeur et la hauteur.

Code : PHP - Calcul de la hauteur/largeur

```
<?php
$largeur = $box[2] - $box[0];
$hauteur = $box[1] - $box[7];
?>
```

Le code de la fonction `image()` est donc :

Code : PHP - Fonction Image

```
<?php

function image($mot)
{
    $size = 32;
    $marge = 5;

    $box = imagettfbbox($size, 0, './smartie.ttf', $mot);
    $largeur = $box[2] - $box[0];
    $hauteur = $box[1] - $box[7];

    $img = imagecreate($largeur+$marge*2, $hauteur+$marge*2);
    $blanc = imagecolorallocate($img, 255, 255, 255);
    $noir = imagecolorallocate($img, 0, 0, 0);

    imagepng($img);
    imagedestroy($img);
}
?>
```

Il faut maintenant écrire le texte grâce à la fonction `imagettftext`.

Code : PHP

```
<?php
imagettftext($img, $size, 0, $marge, $hauteur+$marge, $noir,
'./smartie.ttf', $mot);
?>
```

Pour le paramètre y , il faut essayer diverses valeurs. J'ai pour ma part mis `<?php $hauteur+$marge; ?>`, mais cela dépend de la police que vous utilisez ; essayez de trouver une valeur qui convient et gardez-la.

Exemple

Vous pouvez ajouter à ce captcha les barres horizontales que nous avons incluses précédemment. Et n'hésitez pas à changer les couleurs.

Matrice de convolution

Les matrices de convolution

Les matrices de convolution sont des modifications appliquées à une image : elles sont souvent utilisées dans les logiciels de retouche d'image sous le nom de *filtres*. Le but d'une matrice est d'appliquer une modification de couleur sur chaque pixel de l'image en réalisant une opération par rapport à la couleur des pixels adjacents. Pour un flou, par exemple, on fera la moyenne arithmétique des couleurs (représentées par les nombres *rgb*) des pixels adjacents.

Pour schématiser la modification, on utilise un tableau à deux dimensions (3 x 3 généralement). L'élément (également appelé *case*) central représente le pixel que l'on est en train de modifier. Les autres éléments représentent quant à eux les pixels adjacents. On affecte à chaque élément un nombre représentant le coefficient du pixel. Ce coefficient permet de réaliser une moyenne.

Exemple : le flou gaussien

$$\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$$

On applique la matrice à chaque pixel de l'image, en la centrant sur le pixel traité. Les valeurs de la matrice sont le poids des pixels de l'image. Le point central est le poids du pixel traité, les autres points ceux des pixels entourant ce dernier. On aura donc un problème pour les pixels du contour qui n'ont pas forcément de pixels adjacents, mais plusieurs solutions existent : ignorer ces pixels, considérer que les pixels adjacents sont noirs, modifier les coefficients pour ne pas tenir compte du bord, etc.

Si l'on applique la matrice précédente à un pixel comme celui-ci (les valeurs correspondent à un niveau de gris pour simplifier) :

$$\begin{bmatrix} 68 & 125 & 210 \\ 127 & 34 & 222 \\ 235 & 65 & 128 \end{bmatrix}$$

la couleur du pixel central sera :

$$((1 \times 68) + (2 \times 125) + (1 \times 210) + (2 \times 127) + (4 \times 34) + (2 \times 222) + (1 \times 235) + (2 \times 65) + (2 \times 128)) = 1855$$

On voit bien que la valeur dépasse la valeur maximale pour une couleur (255) : c'était prévisible étant donné que la somme des coefficients est différente de 1. Une solution consiste à diviser tous les coefficients par 16 (1/16, 2/16, etc.) pour obtenir la matrice suivante :

$$\begin{pmatrix} \frac{1}{16} & \frac{2}{16} & \frac{1}{16} \\ \frac{2}{16} & \frac{4}{16} & \frac{2}{16} \\ \frac{1}{16} & \frac{2}{16} & \frac{1}{16} \end{pmatrix}$$

On peut également ajouter des modificateurs (diviseurs et décalages). On donnera la valeur 16 au diviseur pour le flou gaussien car la somme des coefficients n'est pas égale à 1.

$$\frac{1855}{16} = 116$$

Ce qui est alors la valeur finale de la couleur du pixel. Pour une image en couleur, on applique la matrice à chaque composante (rouge, vert, bleu).

Le décalage est une valeur qui s'ajoute au résultat de la division. En général, on lui donne la valeur 0.

En PHP, ça donne quoi ?

Pour notre captcha, on va utiliser une matrice de flou pour déformer notre image. PHP propose une fonction toute faite pour les matrices : `imageconvolution()`.



Imageconvolution est une fonction récente (PHP >= 5.1.0), il est donc possible qu'elle ne fonctionne pas chez votre hébergeur.

Flou gaussien

Voici la fonction qui permet d'ajouter un flou gaussien.

La matrice est, je vous le rappelle, $\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$.

J'utilise cette fois la police [AixDarbotzcumi](#).

Code : PHP

```
<?php
function image($mot)
{
    $size = 32;
    $marge = 5;

    // Flou Gaussien
    $matrix_blur = array(
        array(1,2,1),
        array(2,4,2),
        array(1,2,1));

    $box = imagettfbbox($size, 0, './aix.ttf', $mot);
    $largeur = $box[2] - $box[0];
    $hauteur = $box[1] - $box[7];

    $img = imagecreate($largeur+$marge*2, $hauteur+$marge*2);
    $blanc = imagecolorallocate($img, 255, 255, 255);
    $noir = imagecolorallocate($img, 0, 0, 0);

    imagettftext($img, $size, 0,$marge,$hauteur+$marge, $noir,
    './aix.ttf', $mot);
    imageconvolution($img, $matrix_blur,16,0); // On applique la
    matrice, avec un diviseur 16

    imagepng($img);
    imagedestroy($img);
}
?>
```

Exemple

Vous pouvez utiliser la matrice du flou classique : $\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$.

Code : PHP

```
<?php
```



```

$matrix_blur = array(
    array(1,1,1),
    array(1,1,1),
    array(1,1,1));

imageconvolution($img, $matrix_blur,9,0);
imageconvolution($img, $matrix_blur,9,0); // Appliquée deux fois,
l'effet sera plus prononcé
?>

```

Vous pouvez obtenir de meilleurs captcha avec un fond gris et en appliquant le flou plusieurs fois :

Exemple

Si le sujet vous intéresse, voici un article sur les matrices de convolution : [http://tuxy2885.free.fr/index.php?cat= \[...\] s_convolution](http://tuxy2885.free.fr/index.php?cat= [...] s_convolution).

Déformation

Maintenant, nous nous attaquons à quelque chose d'un peu plus compliqué : la déformation d'image. Déformer son image est une sécurité supplémentaire lorsque l'on crée un captcha.

Rotation des lettres

L'un des paramètres de la fonction `imageffttext()` est un angle servant à incliner le mot. Il nous suffit alors d'écrire les lettres une par une avec un angle aléatoire compris entre - 35° et + 35°.

Pour placer les lettres une par une, on calculera auparavant la largeur d'une lettre.

J'utilise la police `angelina`.

Pour calculer la largeur d'une lettre, on divise la largeur de l'image par le nombre de lettres :

Code : PHP

```

<?php
$box = imageftbbox($size, 0, $font, $mot);
$largeur = $box[2] - $box[0];
$hauteur = $box[1] - $box[7];
$largeur_lettre = round($largeur/strlen($mot));
?>

```

Ensuite, on parcourt le mot lettre par lettre :

Code : PHP

```

<?php
for($i = 0; $i < strlen($mot);++$i)
{
    $l = $mot[$i];
    $angle = mt_rand(-35,35); // Angle au hasard
}
?>

```

Il ne reste plus qu'à écrire la lettre sur l'image :

Code : PHP

```
<?php
imaggittfttext($img,$size,$angle,($i*$largeur_lettre)+$marge,
$hauteur+mt_rand(0,$marge/2),$noir, $font, $l);
?>
```

J'ai ajouté une valeur aléatoire à la hauteur pour que les lettres ne soient pas toutes alignées.

Le code final de la fonction image est le suivant :

Code : PHP - image()

```
<?php
function image($mot)
{
    $size = 32;
    $marge = 15;
    $font = './angelina.ttf';

    $box = imaggittfbbox($size, 0, $font, $mot);
    $largeur = $box[2] - $box[0];
    $hauteur = $box[1] - $box[7];
    $largeur_lettre = round($largeur/strlen($mot));

    $img = imagecreate($largeur+$marge, $hauteur+$marge);
    $blanc = imagecolorallocate($img, 255, 255, 255);
    $noir = imagecolorallocate($img, 0, 0, 0);

    for($i = 0; $i < strlen($mot);++$i)
    {
        $l = $mot[$i];
        $angle = mt_rand(-35,35);
        imaggittfttext($img,$size,$angle,($i*$largeur_lettre)+$marge,
$hauteur+mt_rand(0,$marge/2),$noir, $font, $l);
    }

    imagepng($img);
    imagedestroy($img);
}
?>
```

On peut même rajouter un flou gaussien, des barres horizontales, une taille aléatoire pour les lettres, des couleurs différentes, etc.

Code : PHP

```
<?php

function image($mot)
{
    $size = 32;
    $marge = 15;
    $font = './angelina.ttf';

    $matrix_blur = array(
        array(1,1,1),
        array(1,1,1),
        array(1,1,1));

    $box = imaggittfbbox($size, 0, $font, $mot);
    $largeur = $box[2] - $box[0];
```

```

$hauteur = $box[1] - $box[7];
$largeur_lettre = round($largeur/strlen($mot));

$img = imagecreate($largeur+$marge, $hauteur+$marge);
$blanc = imagecolorallocate($img, 255, 255, 255);
$noir = imagecolorallocate($img, 0, 0, 0);

$couleur = array(
    imagecolorallocate($img, 0x99, 0x00, 0x66),
    imagecolorallocate($img, 0xCC, 0x00, 0x00),
    imagecolorallocate($img, 0x00, 0x00, 0xCC),
    imagecolorallocate($img, 0x00, 0x00, 0xCC),
    imagecolorallocate($img, 0xBB, 0x88, 0x77));

for($i = 0; $i < strlen($mot);++$i)
{
    $l = $mot[$i];
    $angle = mt_rand(-35,35);

    imagefttext($img,mt_rand($size-7,$size),$angle,($i*$largeur_lettre)+$marge,
    $hauteur+mt_rand(0,$marge/2),$couleur[array_rand($couleur)], $font, $l);
}

    imageline($img, 2,mt_rand(2,$hauteur), $largeur+$marge,
    mt_rand(2,$hauteur), $noir);
    imageline($img, 2,mt_rand(2,$hauteur), $largeur+$marge,
    mt_rand(2,$hauteur), $noir);

    imageconvolution($img, $matrix_blur,9,0);
    imageconvolution($img, $matrix_blur,9,0);

    imagepng($img);
    imagedestroy($img);
}
?>

```

Exemple

Vous venez de découvrir comment réaliser un captcha graphique ; ces captchas sont largement utilisés sur Internet bien que cela pose un problème : l'accessibilité. En effet, certaines personnes malvoyantes sont dans l'incapacité de lire votre captcha. Pour pallier ce problème, nous allons voir comment générer un fichier son qui lira le captcha. Les robots sont bien souvent incapables de comprendre un fichier sonore (d'autant plus s'il est en français).

Accessibilité : créer un fichier son

Nous nous attaquons maintenant à la partie la plus compliquée de ce cours : la génération d'un fichier son qui lit notre captcha. Ce chapitre est difficile car nous allons manipuler des **données binaires**, et non des **fichiers textes**.

Un fichier binaire ne s'ouvre et ne se lit pas de la même manière qu'un fichier texte. On n'y écrit pas non plus de la même manière.

L'intérêt de réaliser un fichier **.wav** est de permettre l'accès à votre site aux personnes malvoyantes lorsque votre captcha est une image. Ces personnes disposent souvent d'un navigateur vocal et / ou braille, mais un captcha n'étant pas censé pouvoir être lu par une machine, l'ordinateur de ces personnes ne pourra pas le lire. Un fichier son va permettre l'accès d'un plus grand nombre de personnes à votre site.

Introduction à la manipulation de fichiers binaires

Les fichiers binaires

Manipuler des fichiers binaires est loin d'être aussi facile que des fichiers textes. En effet, un fichier binaire se résume à une suite de 0 et de 1. À partir de là, un certain nombre de problèmes se posent pour le programmeur. Nous allons y répondre progressivement en étudiant chacun d'entre eux.

Comment délimiter les différentes parties d'un fichier ?

Dans un fichier texte, vous pouvez enregistrer une ligne comme cela :

Citation : Fichier texte

```
55;mon fichier texte;12.36;147;1
```

Et à la lecture du fichier, on peut extraire facilement chaque variable grâce aux points-virgules, aux retours à la ligne, ou à n'importe quel délimiteur.

Pour un fichier binaire, c'est différent.

Toutes les informations sont transformées en valeurs numériques, puis en valeurs binaires :

Citation : Fichier binaire

```
0101101101011110001010101110100100110110
```

Impossible de délimiter chaque information : *comment savoir où commence et où se termine chaque information ?*

La première chose à faire est de délimiter le fichier en **octets**, c'est-à-dire en groupes de **8 bits** ou encore, pour faire le lien avec les 0 et les 1, en groupes de 8 chiffres binaires (0 ou 1).

On peut également découper en groupes de 2 chiffres hexadécimaux, ce qui revient au même comme nous allons le voir.

Lorsqu'on lit ce fichier, on obtient donc quelque chose comme ça :

Citation : Fichier binaire

```
01011011 01011110 00101010 11101001 00110110
```

Citation : Fichier binaire (représentation en base 16)

```
5B 5E 2A E9 36
```

Pour obtenir notre information, on peut dès lors demander à lire les x premiers octets puis les x suivants, et ainsi de suite.

Cependant, comment savoir ce qui correspond à ces octets ?

Il n'y a pas de solution miracle, il faut connaître la *structure* du fichier que l'on essaye de lire, ou, si c'est notre propre type de fichier, en définir une.

Par *structure*, je veux dire par exemple « savoir que les 2 premiers octets représentent un nombre, les 3 suivants représentent chacun un caractère », etc.

En effet, chaque type de **donnée informatique** (entier, booléen, flottant) est codé en base 2 (binaire) sur un certain nombre d'octets (nous y reviendrons plus loin). C'est-à-dire qu'à chaque type de données que vous pouvez manipuler avec votre programme correspond un certain nombre de bits (donc d'octets). Connaître la structure d'un fichier, c'est savoir quels octets contiennent quel type de données et ce que ces données renseignent dans le fichier.

Tout cela vous semblera plus clair lorsque nous aurons découvert la structure d'un fichier **.wav**.

Découverte du format .wav

Dans cette partie, nous allons découvrir la structure du type de fichier **.wav-pcm**, car c'est celui-ci que nous utiliserons pour notre fichier son.

Obtenir la structure du fichier

Si vous voulez comprendre comment est fait un format de fichier, vous devrez rechercher les informations concernant sa structure sur le Web. Vous trouverez ces informations chez le propriétaire du format ou sur différents sites, le plus simple étant de **chercher sur Google** ; par exemple, avec les mots-clés « *specifications wave* », on obtient beaucoup de résultats ! Le format **.wav** étant très utilisé, de nombreux sites peuvent vous documenter.

Voici la structure d'un fichier **.wav** (trouvée sur : http://fr.wikipedia.org/wiki/WAVEform_audio_format) :

Format				
Nom	Nombre d'octets	Type	Endian	Description
Bloc de déclaration d'un fichier au format WAVE				
Chunk ID	4	4 char	big	4 caractères constants RIFF : 0x52494646
Chunk Size	4	int 32	little	Taille du fichier - 8 octets des deux premiers blocs
Format	4	4 char	big	4 caractères constants WAVE : codé 0x57415645
Bloc décrivant le format audio				
SubChunk 1 ID	4	4 char	big	4 caractères constants "fmt " : codé 0x666d7420
SubChunk 1 Size	4	int 32	little	Taille du sous-bloc "SubCHunk 1" - 8 octets, vaut 16 pour le PCM
Audio Format	2	int 16	little	Type de compression audio, 1 pour PCM
Num Channels	2	int 16	little	Nombre de canaux (1 = Mono, 2 = Stéréo, ...)
Sample Rate (Frequence)	4	int 32	little	Fréquence d'échantillonnage (nombre d'échantillons par seconde)
Byte Rate (BytePerSec)	4	int 32	little	Nombre d'octets par seconde = Frequence * BitsPerSample/8 * NumChannels
Block Align (BytePerSample)	2	int 16	little	Nombre d'octets par échantillon (tous canaux confondus) = Nombre de canaux * Nombre d'octets par échantillons
Bits per Sample	2	int 16	little	Nombre de bits par échantillon
Bloc des données				

SubChunk 2 ID	4	4 char	big	4 caractères constants : "data" : 0x64617461
SubChunk 2 Size	4	int 32	little	Taille des données = NbCanaux * NbSamples * BitsPerSample/8 = taille du fichier - 44 octets (taille de l'en-tête, rappelez-vous-en !)
Datas (données)	FileSize - 44	...	little	Les données !

Image utilisateur

Petite parenthèse : qu'est-ce que l'endian ?

J'ai mis un lien pour l'expliquer, mais si vous ne l'avez pas lu, je vais l'expliquer brièvement ici. L'endian est la manière de représenter les données en binaire : ici, nous ne nous occuperons que de *big endian* et de *little endian* pour des entiers.

Un entier peut être représenté sur plusieurs octets s'il est supérieur à 255, valeur maximale que peut représenter un octet (1111111) ; il faut donc le représenter sur 2, 4, 8 ou 16 octets. On parle d'entiers 8 bits, 16 bits, 32 bits, 64 bits...

Bits	Octets	Valeurs représentées (entier positif)
8	1	0 à 2^{8-1} soit [0 ; 255]
16	2	[0 ; 2^{16-1}] soit [0 ; 65535]
32	4	[0 ; 2^{32-1}] soit [0 ; 4294967295]
64	8	[0 ; 2^{64-1}] soit [0 ; 18446744073709551615]

Dans le format **.wav**, la plupart des entiers sont codés sur 4 ou 2 octets.

Pour coder un nombre sur plusieurs octets, on le convertit en binaire et on enregistre les octets (groupe de 8, 1 ou 0) à la suite. Si le nombre d'octets que l'on s'est donné pour enregistrer l'entier est supérieur au nombre d'octets qu'il faut pour l'enregistrer, on ajoute des bits nuls (0). L'endian caractérise l'ordre dans lequel sont enchaînés les octets. En *big-endian*, le bit de poids fort est en premier ; en *little-endian*, c'est l'inverse.

Exemple

Prenons le nombre 2084 en décimal, valant 824 en hexadécimal : représenté sur 4 octets, on a **00 00 08 24** en hexadécimal. Bits de poids fort en rouge, et bits de poids faible en bleu.

Il sera égal à **00000000 00000000 00001000 00100100** en binaire (codé sur 4 octets).

Enregistré en *big endian*, il apparaîtrait comme cela dans le fichier :

Citation : Big Endian

00000000 00000000 00001000 00100100 ou **00 00 08 24** (hex)

(hex) : notation hexadécimale que l'on trouve dans les éditeurs hexadécimaux, plus simple à afficher car deux chiffres hexadécimaux représentent un octet.

S'il était codé en *little endian*, l'octet de poids fort apparaîtrait en dernier.

Citation : Little Endian

00100100 00001000 00000000 00000000 ou **24 08 00 00** (hex)

Un exemple de fichier .wav

Maintenant, je pense qu'il faudrait vous montrer un exemple de fichier `.wav` pour bien comprendre.

Code : Autre

```
52 49 46 46 24 08 00 00 57 41 56 45 66 6d 74 20 10 00 00 00 01 00 02 00
22 56 00 00 88 58 01 00 04 00 10 00 64 61 74 61 00 08 00 00 00 00 00
24 17 1e f3 3c 13 3c 14 16 f9 18 f9 34 e7 23 a6 3c f2 24 f2 11 ce 1a 0d
```

J'ai écrit ce fichier dans un éditeur hexadécimal (par exemple hexedit pour Windows, ghex pour Gnome, khexedit pour KDE). Avec un éditeur hexadécimal, vous pouvez voir les données binaires (contrairement à un éditeur de texte).



Un nombre hexadécimal de deux chiffres représente en fait un octet du fichier. Les 4 premiers octets contiennent le mot "RIFF", comme on le voit à droite du logiciel. Ensuite, les 4 octets suivants contiennent la taille du fichier **moins 8 octets**. Cependant, l'entier est codé en **little endian**, ce qui veut dire qu'il faut lire le nombre comme cela :

Citation : Chunk Size

24 08 00 00 correspond à 00 00 08 24 en hexadécimal (824)

Sortez votre calculatrice, et vous obtenez 2084 octets (le fichier a été coupé à 72 octets pour simplifier).

Cette valeur peut être obtenue directement depuis l'éditeur hexadécimal dans la case 32 bits (4 octets) en sélectionnant le premier octet du nombre et en activant le décodage **little endian** / petit boutiste.



Nous avons ensuite :

Code : Autre

```
52 49 46 46 24 08 00 00 57 41 56 45 66 6d 74 20 10 00 00 00 01 00 02 00
R I F F | 2048 | W A V E f m t ?
| 16 | | 1 | | 2 |
...
```

Essayez de continuer ainsi tout le fichier. 😊 Sachant que le fichier est composé de deux canaux et que BitsPerSample vaut « 10 00 », soit 16, un échantillon sera composé de BytePerSample * NbChannels octets ($16/8 * 2$), soit 4 octets.

Maintenant que nous avons vu la théorie, nous allons passer à la pratique avec PHP !

Lecture d'un fichier binaire

Maintenant que vous avez compris comment était organisé un fichier binaire, nous allons voir comment en lire un avec PHP. J'ai créé un petit fichier pour nos tests avec un éditeur hexadécimal : vous pouvez faire de même ou le télécharger à cette adresse : <http://tutos.alwaysdata.net/captcha/sound1/test.wav>.

Code : Autre - Fichier test.wav

```
52 49 46 46 24 08 00 00
```

Ce fichier contient les 4 caractères « RIFF » en *big endian* et un entier sur 4 octets en *little endian* (2084 en décimal).

Notre but est de lire ce fichier avec PHP et d'afficher les 4 caractères, et l'entier en décimal. Nous allons utiliser la fonction `unpack`.

Elle prend en paramètre le format (la structure, si vous préférez) et les données binaires.

Pour le format, il faut consulter la documentation de la fonction `pack` (nous la verrons plus tard). On apprend que le format doit être constitué d'un caractère pour identifier le type de données, suivi d'un nombre indiquant la quantité (optionnel). Les différents types de formats sont séparés par des « / ».

Nous, nous avons 4 caractères et un entier sur 32 bits (4 octets) en *little endian*. Selon la doc, le format devrait être quelque chose de ce type :

Citation : Format unpack

```
c4riff/Vint
```

J'ai donné les noms riff et int aux variables qui seront créées par unpack.

Maintenant, nous avons besoin des données ; pour les obtenir, nous allons lire le fichier avec `fread()`, mais nous allons l'ouvrir avec `fopen()` en mode **binaire**, en rajoutant **b** dans l'option d'ouverture !

Code : PHP

```
<?php
$file = fopen('test.wav', 'rb'); // Ouvert en mode lecture / binaire
$data = fread($file, 8); // Fichier de 8 octets
$infos = unpack('c4riff/Vint', $data);
?>
```

Avec ce code, les données sont stockées dans le tableau infos. Il ne reste plus qu'à les afficher !

Code : PHP

```
<?php
$file = fopen('test.wav', 'rb'); // Ouvert en mode lecture / binaire
$data = fread($file, 8); // Fichier de 8 octets
$infos = unpack('c4riff/Vint', $data);
var_dump($infos);
echo '<br />';
for($i = 0; $i < 4; ++$i)
{
    echo chr(current($infos));
    next($infos);
}

echo $infos['int'];
?>
```

Le tableau infos contient :

Code : PHP

```
array(5) {
```



```

        ["riff1"]=>    int(82)
        ["riff2"]=>    int(73)
        ["riff3"]=>    int(70)
        ["riff4"]=>    int(70)
        ["int"]=>     int(2084)
    }

```

Avec `chr()`, on obtient le caractère correspondant à la valeur ASCII indiquée en paramètre : cela permet d'afficher RIFF.

Ce code ne présente aucune autre difficulté, et maintenant vous savez lire un fichier binaire !

Lecture d'un fichier .wav

Bien ! Maintenant, nous allons lire un fichier `.wav` et donc implémenter quelques fonctions :

- `bool isWav(string $file)` : vérifie si le fichier est un fichier `.wav` ;
- `array getHeader(string $file)` : retourne l'en-tête du fichier ;
- `string getData(string $file)` : retourne les données brutes (binaires) du fichier `.wav`.

J'utilise des fichiers `.wav` générés par le site <http://www.research.att.com/~ttsweb/tts/demo.php> (usage non commercial uniquement).

J'ai enregistré des fichiers « 1 », « 2 », « 3 », etc. pour mes tests.

J'ai créé un dossier dans lequel j'ai mis `wav.php` qui contient les fonctions pour générer le son, et un dossier `sounds` qui contient les fichiers `.wav`.

Vous pouvez placer dans ce dossier nos fichiers `captcha.php`, `formulaire.php` et `verification.php`.

isWav

Pour cette fonction, nous allons lire certains blocs de l'en-tête du fichier et vérifier que :

- les 4 premiers octets contiennent la chaîne **RIFF** ;
- les octets 4 à 8 contiennent un entier égal à la taille du fichier moins 8 ;
- les octets 8 à 12 contiennent la chaîne **WAVE** ;
- les octets 12 à 16 contiennent la chaîne **fmt** ;
- les octets 36 à 40 contiennent la chaîne **data** ;
- les octets 40 à 44 contiennent la taille du fichier moins 44.

Si toutes ces conditions sont réunies, on est en face d'un fichier `.wav` et l'on retourne `true`.

Code : PHP

```

<?php
function isWav($file)
{
    if(is_file($file) && is_readable($file))
    {
        $res = fopen($file, 'rb');
        $data = fread($res, 16);
        $h = unpack('H8riff/Vfile_size/H8wave/H8fmt', $data);

        if($h['riff'] === '52494646' &&
            $h['file_size'] === filesize($file) - 8 &&
            $h['wave'] === '57415645' &&
            $h['fmt'] === '666d7420')
        {

            fseek($res, 36); // position au bloc data

            $data = fread($res, 8);
            fclose($res);

```

```

$sh = unpack('H8data/Vdata_size',$data);

if($sh['data'] === '64617461' &&
    $sh['data_size'] === filesize($file) - 44)
    return true;
else
    return false;
}
else{
    fclose($res);
    return false;
}
}
else{
    return false;
}
}
}??>

```

J'ai utilisé le format H8 pour les chaînes de caractères : comme cela, j'ai une chaîne qui contient la forme hexadécimale telle que l'on peut la voir dans un éditeur du même type. On traduit donc **RIFF** par « 52494646 », pareil pour **WAVE**, **fmt** et **data**. On vérifie aussi la taille du fichier.

getHeader

Cette fonction retournera un tableau donné par un unpack() de l'en-tête du fichier **.wav**.

Il s'agit juste de définir le format du paramètre pour unpack() ; avec les informations sur le fichier **.wav**, ça ne devrait pas poser de problème.

On vérifiera que c'est bien un fichier **.wav** avec isWav().

Code : PHP

```

<?php
function getHeader($file)
{
    if(isWav($file))
    {
        $res = fopen($file,'rb');
        $data = fread($res, 44);
        fclose($res);

        // Riff chunk descriptor
        $entete_unpack = 'H8FileTypeBlocID/VFileSize/H8FileFormatID';
        // Sub Chunk fmt
        $entete_unpack
        .= '/H8FormatBlocID/VBlocSize/vAudioFormat/vNbrCanaux/VFrequence/VBytePerSec/vByt
        $entete_unpack .= '/vBitsPerSample';
        // Sub Chunk data
        $entete_unpack .= '/H8DataBlocID/VDataSize';
        return unpack($entete_unpack,$data);
    }
    else{
        return false;
    }
}
}
}??>

```

On lit les 44 premiers octets (taille de l'en-tête), puis on utilise unpack sur les données binaires.

J'ai utilisé un format en me basant sur les tableaux du format **.wav** (Wav#En-t.C3.AAte_de_fichier_WAV et

[http://ccrma.stanford.edu/CCRMA/Course \[...\] s/WaveFormat/](http://ccrma.stanford.edu/CCRMA/Course [...] s/WaveFormat/)).

Vous pouvez vous amuser à faire :

Code : PHP

```
<pre>
<?php var_dump(getHeader('sounds/1.wav')); ?>
</pre>
```

pour voir les informations du fichier **.wav** !

getData

getData() est la fonction la plus simple à coder :

- on vérifie que le fichier est un fichier **.wav** ;
- on lit du 44^e octet à la fin du fichier ;
- on retourne les données brutes (pas de unpack).

Code : PHP

```
<?php
function getData($file)
{
    if(isWav($file))
    {
        $res = fopen($file, 'rb');
        fseek($res, 44);
        return fread($res, filesize($file) - 44);
    }
    else{
        return false;
    }
}
?>
```

Maintenant, nous avons toutes les fonctions pour lire n'importe quel fichier **.wav** !

Dans la partie suivante, nous allons voir comment en créer un.

Écriture d'un fichier .wav

Nous allons maintenant voir comment créer un fichier **.wav** ! Nous allons utiliser la fonction **pack** et les fonctions de lecture que nous venons de créer.

Pack fonctionne comme unpack() mais de façon inverse : elle prend en paramètre des entiers, des chaînes de caractères, n'importe quelle variable et crée une chaîne binaire.

Notre but est en fait de combiner plusieurs fichiers **.wav** pour en créer un nouveau qui « lira » notre captcha. Dans cet exemple, nous allons lire un nombre, on combinera donc des fichiers « 1 », « 2 », « 3 », etc. J'ai créé des fichiers comme cela avec <http://www.research.att.com/~ttsweb/tts/demo.php>.

Maintenant, la liste des fonctions à créer :

- array listWav(\$mot) : liste les fichiers qu'il faut assembler pour obtenir le mot ;
- string toWav(\$mot) : renvoie les données binaires d'un fichier **.wav** créé à partir de la liste des fichiers retournée par listWav.

Vous ne devriez rencontrer aucune difficulté à écrire la fonction `listWay` :

Code : PHP

Code : PHP

www.openclassrooms.com

```

    if ($nbFiles == 1)
        return file_get_contents(current($list));

    // On se base donc sur le premier fichier de la liste et on
    vérifie le reste :
    $infos = getHeader(current($list));
    for ($i = 1; $i < $nbFiles; ++$i)
    {
        $h = getHeader($list[$i]);
        if ($h['AudioFormat'] != $infos['AudioFormat'])
            die('AudioFormat in '.$list[$i].' different');
        if ($h['NbrCanaux'] != $infos['NbrCanaux'])
            die('NbrCanaux in '.$list[$i].' different');
        if ($h['Frequence'] != $infos['Frequence'])
            die('Frequence in '.$list[$i].' different');
        if ($h['BytePerSec'] != $infos['BytePerSec'])
            die('BytePerSec in '.$list[$i].' different');
        if ($h['BytePerBloc'] != $infos['BytePerBloc'])
            die('BytePerBloc in '.$list[$i].' different');
        if ($h['BitsPerSample'] != $infos['BitsPerSample'])
            die('BitsPerSample in '.$list[$i].' different');

        foreach ($list as $file)
        {
            $datas .= getData($file);
        }

        $datasize = strlen($datas);
        $filesize = 36 + $datasize;
        $entete_pack = 'H8VH8H8VvvVVvvH8V';

        $file = pack($entete_pack,
            $infos['FileTypeBlocID'], $filesize, $infos['FileFormatID'],
            $infos['FormatBlocID'], $infos['BlocSize'],
            $infos['AudioFormat'],
            $infos['NbrCanaux'], $infos['Frequence'],
            $infos['BytePerSec'],

            $infos['BytePerBloc'], $infos['BitsPerSample'], $infos['DataBlocID'],
            $datasize) . $datas;

        return $file;
    }
    else{
        return false;
    }
}

session_start();
header('Content-type: audio/x-wav');
header('Content-Disposition: attachment; filename="captcha.wav"');
echo toWav($_SESSION['captcha']);

/* Pour enregistrer le fichier, on aurait fait :

$filename = uniqid().'wav';
$res = fopen($filename, 'wb');
fwrite($res, toWav($_SESSION['captcha']));
fclose($res);

*/
?>

```


Vous pouvez tester le script en allant sur la page [wav.php](#) et en remplaçant `<?php echo toWav($_SESSION['captcha']); ?>` par `<?php echo toWav('cequevousvoulez0123456789'); ?>`,

du moment que les fichiers sons ([a.wav](#), [b.wav](#), etc.) existent.

Intégration dans notre script de captcha

À présent, nous allons intégrer nos fonctions qui génèrent des fichiers [.wav](#) dans notre script de captcha.

Il suffit juste de mettre un lien vers [wav.php](#) pour la lecture du fichier [.wav](#).

J'utilise cette image pour le lien :  (GNU/LGPL).

formulaire.php

On ajoute juste un lien vers [wav.php](#) :

Code : PHP

```
<label for="captcha">Recopiez le mot :  <a href="wav.php"></a></label>
```

Source du fichier.

captcha.php

Je n'ai rien eu à changer dans [captcha.php](#).

Source du fichier.

wav.php

J'ai choisi de ne pas enregistrer le fichier et d'utiliser la fonction toWav() de cette manière :

Code : PHP

```
<?php

function toWav() {
    // [...]

    $file = pack($entete_pack,
        $infos['FileTypeBlocID'], $filesize, $infos['FileFormatID'],
        $infos['FormatBlocID'], $infos['BlocSize'], $infos['AudioFormat'],
        $infos['NbrCanaux'], $infos['Frequence'], $infos['BytePerSec'],

        $infos['BytePerBloc'], $infos['BitsPerSample'], $infos['DataBlocID'],
        $datasize) . $datas;

    return $file;
}

?>
```

Pour ainsi pouvoir envoyer le fichier en dehors de la fonction :

Code : PHP

```
<?php
session_start();
```

```
header('Content-type: audio/x-wav');  
header('Content-Disposition: attachment; filename="captcha.wav");  
echo toWav($_SESSION['captcha']);  
?>
```

[Source du fichier.](#)

Exemple

Vous avez dorénavant un captcha à la fois résistant et accessible !

Ce chapitre vous a permis - je l'espère - d'apprendre à manipuler les fichiers binaires. Vous pouvez utiliser le même principe pour éditer des fichiers [.mp3](#) ou [.zip](#) et même des images ([.png](#), [.jpeg](#)...) sans passer par une bibliothèque.

J'espère que ce cours vous aura appris bien plus que la création de captchas : c'est en effet un type d'application qui permet d'utiliser des notions de programmation originales et intéressantes.

Si vous avez apprécié ce tutoriel, merci de laisser un petit message en commentaire. Si vous avez des difficultés, n'hésitez pas à me contacter par MP et / ou à demander de l'aide sur le forum.

Merci.