I. Introduction

1) Statique - dynamique

On considère qu'il existe deux types de sites web : les sites [ ] et les sites [ ].

- Les **sites** [ ] sont des sites réalisés uniquement à l'aide des langages HTML et CSS. Ils fonctionnent très bien mais leur contenu ne peut pas être mis à jour automatiquement : il faut que le propriétaire du site (le webmaster) modifie le code source pour y ajouter des nouveautés. Ce n'est pas très pratique quand on doit mettre à jour son site plusieurs fois dans la même journée! Les sites statiques sont donc bien adaptés pour réaliser des sites « vitrine », pour présenter par exemple son établissement. Ce type de site se fait de plus en plus rare aujourd'hui, car dès que l'on rajoute un élément d'interaction (comme un formulaire), on ne parle plus de site statique mais de site dynamique.
- Les **sites** [ ] sont plus complexes, ils utilisent d'autres langages en plus de HTML et CSS, tels que PHP et MySQL. Le contenu de ces sites web est dit « dynamique » parce qu'il peut changer sans l'intervention du webmaster. La plupart des sites web que vous visitez aujourd'hui sont des sites dynamiques (ce qui n'est pas le cas du site NSI). Le seul prérequis pour apprendre à créer ce type de site est de savoir réaliser des sites statiques en HTML et CSS ce qui a été fait précédemment.

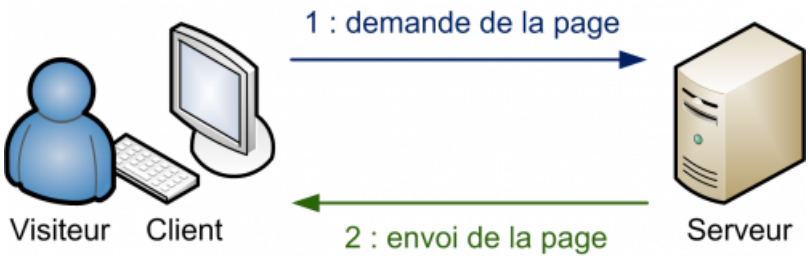
2) Fonctionnement général

Internet est un réseau composé d'ordinateurs. Ceux-ci peuvent être classés en deux catégories :

- [ ] : ce sont les ordinateurs des internautes comme vous. Votre ordinateur fait donc partie de la catégorie des clients.
- [ ] : ce sont des ordinateurs puissants qui stockent et délivrent des sites web aux internautes, c'est-à-dire aux clients.

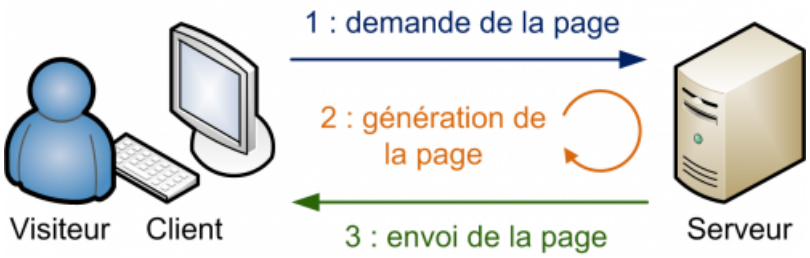
Dans le cas d'un **site statique** :

- le [ ] demande au [ ] de voir une page Web ;
- le [ ] répond en envoyant au [ ] la page demandée.



Dans le cas d'un **site dynamique** il y a une étape intermédiaire supplémentaire :

- le [ ] demande au [ ] de voir une page Web ;
- le [ ] prépare la page spécialement pour le [ ] ;
- le [ ] répond en envoyant au [ ] la page demandée.



Dans toutes ces communications, c'est le protocole HTTP ou HTTPS qui est utilisé.

3) Le PHP

PHP (sigle récursif pour [ ]) est un langage de scripts généraliste et Open Source, spécialement conçu pour le développement d'applications web. Il peut être intégré facilement au HTML.

Exemple 1 :

```
<!DOCTYPE html>
<html>
  <head>
    <title>Exemple PHP</title>
  </head>
  <body>
  <?php
    echo "Ceci est du PHP";
  ?>
  </body>
</html>
```

Le code PHP est inclus entre une balise de début et une balise de fin qui permettent au serveur web de passer en mode PHP.

Ce qui distingue PHP des langages de script comme le Javascript (que nous avons vu précédemment), est que le code est exécuté sur , générant ainsi le HTML, qui sera ensuite envoyé au client. Le client ne reçoit que le résultat du script, sans aucun moyen d'avoir accès au code qui a produit ce résultat. Vous pouvez configurer votre serveur web afin qu'il analyse tous vos fichiers HTML comme des fichiers PHP. Ainsi, il n'y a aucun moyen de distinguer les pages qui sont produites dynamiquement des pages statiques.

Pour les sites Web et applications Web (script côté serveur), tâche qui est de loin la plus répandue, vous avez besoin de trois choses : , un et un . Vous avez probablement un navigateur, et si vous avez installé WAMP ou XAMPP comme demandé, vous disposez aussi du serveur Web Apache. Vous pouvez aussi louer un espace à une société. De cette façon, vous n'aurez pas à mettre en place PHP, mais uniquement à écrire vos scripts, les charger sur le serveur et voir le résultat sur votre navigateur.

PHP permet de faire bien d'autres choses que nous ne verrons pas ici.

On considère que les fichiers terminés par l'extension `.php` sont traités par PHP. Sur la plupart des serveurs, c'est la configuration par défaut. Si votre serveur web supporte PHP, vous n'avez rien à faire. Simplement, il faut mettre vos fichiers avec l'extension `.php` dans le répertoire lu par votre serveur (par défaut `/var/www/html` sous linux, `C:\wamp64\www` sous windows, `/htdocs` sous mac). Vous lancez ensuite un navigateur avec comme adresse `localhost/` suivi du nom de votre fichier, et le serveur va automatiquement l'exécuter avec PHP. Il n'y a pas de compilation, ou d'installation compliquée. Gardez en tête que les fichiers sont comparables à des fichiers HTML, dans lesquels vous allez utiliser des balises magiques, qui feront beaucoup de choses pour vous.

II. Exemples de pages en PHP

Exemple 1 (suite) : On reprend l'exemple de la partie précédente. Créer un fichier `exemple1.php` dans le bon répertoire, copier le code à l'intérieur, sauvegarder, puis taper dans la barre d'adresse du navigateur

Ce script ne fait qu'afficher « » à l'aide de l'instruction `echo`. Ce n'est pas très dynamique nous allons voir quelque chose de plus puissant dans l'exemple qui suit.

Exemple 2 : Nous allons vérifier le type de navigateur que le visiteur de notre site utilise. Pour cela, nous allons accéder aux informations que le navigateur du visiteur nous envoie, lors de sa requête HTTP. Cette information est stockée dans la variable $_SERVER['HTTP USER AGENT'].
$_SERVER est une variable spéciale de PHP, qui contient toutes les informations relatives au serveur web. C'est une variable réservée de PHP, qui est super-globale (ces variables sont automatiquement créées par PHP à chaque fois qu'une page est chargée. Elles existent donc sur toutes les pages et sont accessibles partout : au milieu de votre code, au début, dans les fonctions, etc.).
Créer un fichier `exemple2.php` toujours dans le répertoire `www` contenant le code suivant :

```
<?php echo $_SERVER['HTTP_USER_AGENT']; ?>
```

On devrait obtenir par exemple un affichage du type :

```
Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:70.0) Gecko/20100101 Firefox/70.0
```

III. Transmettre des données via l'URL

Une URL, Uniform Resource Locator, sert à représenter une adresse sur le web.
Dans cet exemple, les informations après le point d'interrogation sont des données que l'on fait transiter d'une page à une autre.

https://www.leboncoin.fr/recherche/?text=205&locations=r_3

Ce que l'on voit après le point d'interrogation sont des paramètres que l'on envoie à la page PHP. Celle-ci peut récupérer ces informations dans des variables.
Le point d'interrogation sépare le nom de la page PHP des paramètres.
Ensuite, ces derniers s'enchaînent selon la forme **nom=valeur** et sont séparés les uns des autres par le symbole &.

Dans l'exemple ci dessus il y a variables, qui ont pour nom et pour valeurs respectives

Nous allons créer des liens en HTML pour transmettre des paramètres d'une page à l'autre.
Dans un premier temps créer un fichier `index.html` qui contient le code suivant :

```
<a href="description.php?nom=Morane&amp;prenom=Bob&amp;age=42">Qui suis-je ?</a>
```

Remarque : en HTML, on demande à ce que les & soient écrits `&` dans le code source. Cela permet de passer la validation W3C.
Nous faisons un lien dans `index.html` vers `description.php` avec transmission des informations via l'URL.
Nous devons maintenant récupérer les informations dans la page `description.php`.
La page qui réceptionne les paramètres description.php va automatiquement créer un tableau au nom un peu spécial : $_GET. Il s'agit d'un tableau associatif (en python on dirait un dictionnaire) dont les clés correspondent aux noms des paramètres envoyés en URL.
Par rapport à l'exemple on a le schéma suivant :

Nom	Valeur
$_GET['nom']	
$_GET['prenom']	
$_GET['age']	

On peut donc récupérer les informations transmises.
Créer le fichier `description.php` et copier le code suivant qui va permettre d'afficher les renseignements sur la personne. Pour simplifier et gagner du temps on ne met pas toutes les informations d'en tête d'une vraie page HTML.

```
<p>Bonjour vous vous appelez <?php echo $_GET['prenom'] . ' ' . $_GET['nom']; ?>
et vous avez <?php echo $_GET['age']; ?> ans </p>
```

Remarque : En php, le point entre deux chaînes de caractères permet de les concaténer (comme le en python).

Attention : il ne faut **JAMAIS** faire confiance aux variables qui transitent de page en page, comme $_GET.
Avec l'exemple précédent, nous allons avoir une URL du type :
http://localhost/description.php?nom=Morane&prenom=Bob&age=42
Mais on peut changer les paramètres directement dans la barre d'adresse et saisir par exemple :
http://localhost/description.php?nom=Bracame&prenom=Edouard&age=42
On va obtenir un affichage avec les nouvelles valeurs !
Moralité on ne peut pas avoir confiance dans les données qu'on reçoit. N'importe quel visiteur peut les changer.

IV. Les formulaires

Les formulaires constituent le principal moyen pour vos visiteurs d'entrer des informations sur un site. Les formulaires permettent de créer une interactivité, par exemple pour accéder à une zone membre en saisissant son login et mot de passe.

1) Les bases du formulaire

En HTML, pour insérer un formulaire, on se sert de la balise `<form>`.
On a la structure de base suivante :

```
<form method="post" action="cible.php">
<p>
   On place ici les éléments du formulaire.
</p>
</form>
```

Dans la balise `<form>`, il y a deux attributs très importants à connaître que nous allons voir.

2) L'attribut method

Il existe plusieurs moyens d'envoyer les données du formulaire. On peut en employer deux.

get : les données transiteront par l'URL comme vu précédemment. On pourra les récupérer grâce au tableau . Cette méthode est assez peu utilisée car on est assez limité pour envoyer beaucoup d'informations dans l'URL.

post : les données ne transiteront pas par l'URL, l'utilisateur ne les verra donc pas passer dans la barre d'adresse. On pourra les récupérer grâce au tableau $_POST. Cette méthode permet d'envoyer autant de données que l'on veut, ce qui fait qu'on la privilégie le plus souvent. Néanmoins, à part si on utilise le protocole HTTPS, les données ne sont pas plus sécurisées qu'avec la méthode GET.

En général la méthode que l'on utilise est post, on écrira donc :

3) L'attribut action

L'attribut action sert à définir la page appelée par le formulaire. C'est cette page qui recevra les données du formulaire et qui sera chargée de les traiter. En effet, la page contenant le formulaire ne fait aucun traitement particulier, mais une fois le formulaire envoyé (lorsqu'on a cliqué sur le bouton « Valider »), le visiteur est redirigé vers la page `cible.php` qui reçoit les données du formulaire.

4) Exemple

Nous allons créer une page qui vous demande votre nom et qui va l'afficher sur la page `cible.php`.
On créé une première page `exemple3.html` avec le code suivant :

```
<p>
   Bienvenue sur mon site<br/>
   Veuillez taper votre nom :
</p>

<form method="post" action="cible.php">
<p>
   <input type="text" name="nom" />
   <input type="submit" value="Valider" />
</p>
</form>
```

Ensuite on va créer la page `cible.php` contenant le code suivant. Comme on a utilisé la méthode `post`, on va récupérer les informations dans le tableau . La clé est donnée par l'attribut `name` de l'élément `<input>`, c'est à dire

```
<p>Bonjour !</p>
<p>Vous vous appelez <?php echo  ; ?> !</p>
<p>pour revenir à la page exemple3.html <a href="exemple3.html">clique ici</a>.</p>
```

Il n'y a plus qu'à taper dans la barre d'adresse du navigateur pour tester le fonctionnement.

5) Failles de sécurité

Il ne faut pas non plus faire confiance aux paramètres passés aux formulaires. On peut en effet envoyer des informations qui ne sont pas celle attendues ou ne pas envoyer toutes les informations attendue.

Un grand classique : La faille XSS (pour cross-site scripting) est une technique qui consiste à injecter du code HTML contenant du JavaScript dans vos pages pour le faire exécuter à vos visiteurs. On peut arriver à récupérer les informations personnelles contenues dans un cookie par exemple ...
Pour éviter ce genre de chose il faut protéger le code HTML en l'échappant, c'est-à-dire en affichant les balises (ou en les retirant) plutôt que de les faire exécuter par le navigateur en utilisant la fonction **htmlspecialchars**.

6) Un autre exemple : TP mot de passe

On veut réaliser une page « secrète » qui n'est accessible que par mot de passe.
On va commencer par créer une page `accueil.html` que vous devez compléter :

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <title>Accueil</title>
    </head>
    <body>
        <p>Veuillez entrer le mot de passe pour passer à la page secrète :</p>
        <form method="      " action="                " >
            <p>
                <input type="password" name="mot_de_passe" />
                <input type="submit" value="Valider" />
            </p>
        </form>
        <p> Page uniquement accessible aux élèves de NSI ! </p>
    </body>
</html>
```

Puis une autre page `pageSecrete.php` également à compléter :

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <title>Page NSI</title>
    </head>
    <body>
        <?php
            if (isset($_POST[              ]) AND $_POST[              ] == "NSI42") {
                // Si le mot de passe est bon, on affiche les informations de la page
        ?>
        <h1>Page des premières NSI :</h1>
        <p><strong>Vous êtes trop fort !</strong></p>
        <p>Cette page est réservée aux premières NSI <br/>
        N'oubliez pas de consulter régulièrement le site des NSI</p>
        <?php
        }
        else { // Sinon, on affiche un message d'erreur
            echo '<p>Mot de passe incorrect</p>';
        }
        ?>
    </body>
</html>
```

Remarques : Tout ce qui suit un // est un commentaire (comme après un en python)
La fonction `isset` teste si la variable passée en paramètre existe.
On retrouve une structure connue `if else` : c'est une
Le mot de passe apparaît en clair dans le code source, mais ce n'est pas très gênant puisqu

Complément : Dans la pratique, c'est souvent la même page que se charge d'afficher le formulaire et de le traiter. Reprendre l'exemple ci-dessus avec cette méthode.
Indication : on pourra commencer par tester si la variable `$_POST['mot_de_passe']` est définie.