# Introduction aux applications web

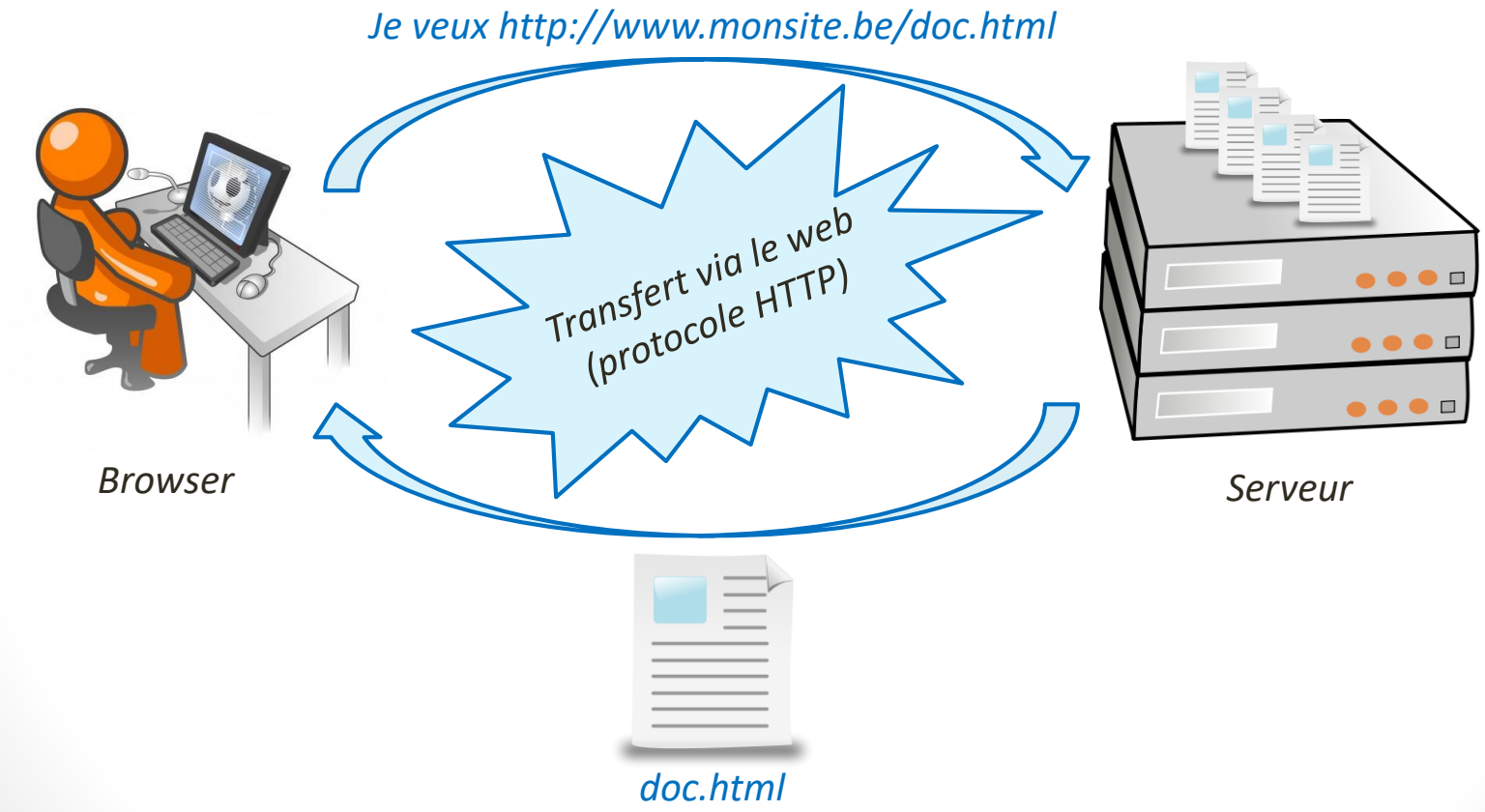
Développement web
HENALLUX — IG3 — 2018-2019

# Au programme…

- **Du site web statique au site web dynamique**
  - Sites web statique, HTML-dynamique, dynamique
- **PHP, un premier aperçu**
  - À quoi sert PHP ? Comment cela fonctionne-t-il ?
- **Architectures distribuées**
  - Un peu de vocabulaire pour parler des applications comportant un aspect « client » et un aspect « serveur ».
- **Organisation du cours**
  - Déroulement du cours, évaluation, etc.

# Un site web

- **Comment fonctionne un site web (version IG1) ?**



# Un site web

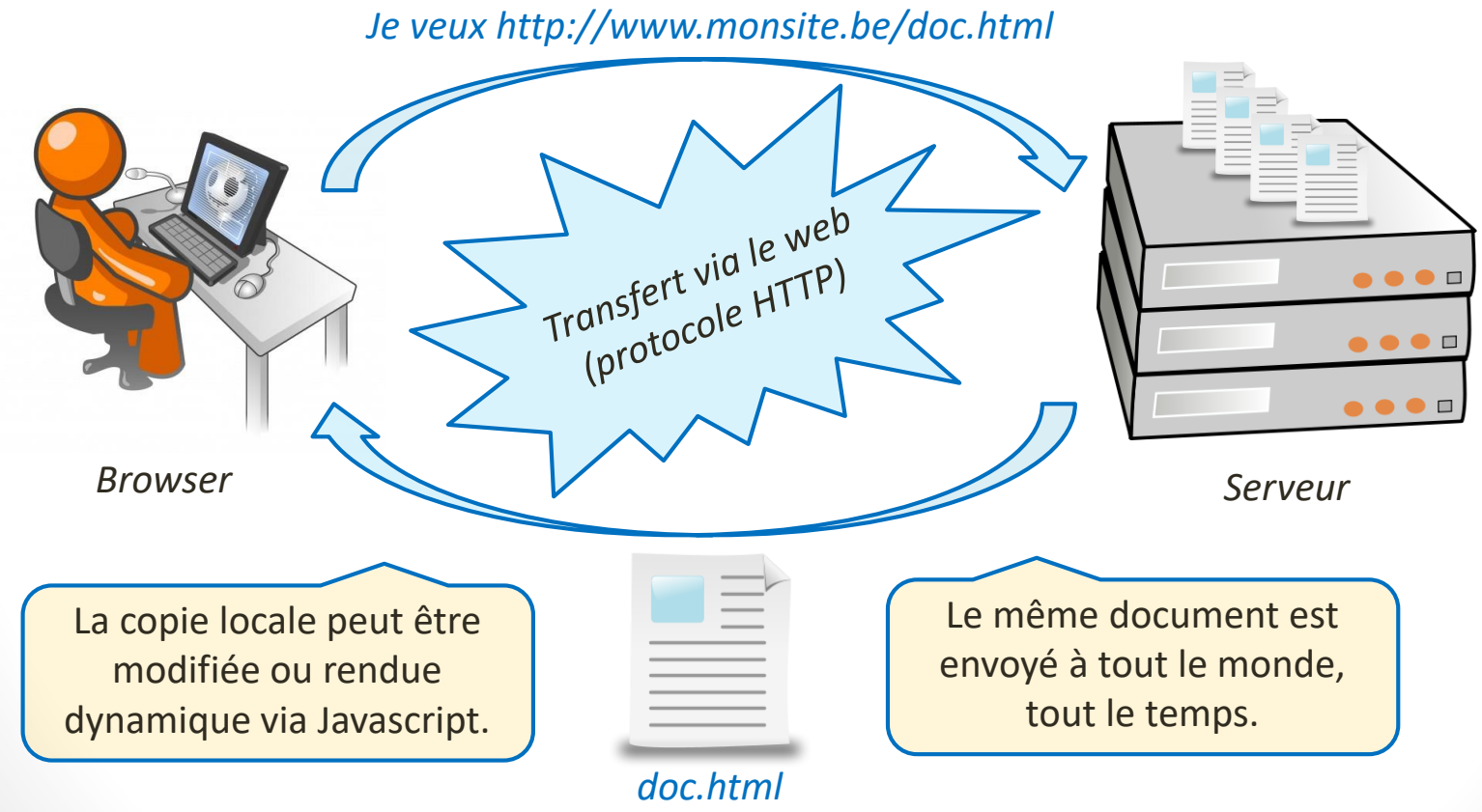
- **Quelles sont les ressources utilisées ?**

  - un navigateur utilisé par l'internaute
    - qui envoie les demandes,
    - réceptionne les réponses et
    - gère l'affichage (standards HTML et CSS).

  - un serveur
    - serveur = à la fois machine physique et logiciel
    - machine physique accessible via internet
    - logiciel-serveur capable de répondre aux demandes de fichiers

# Un site web HTML-dynamique

- **Comment fonctionne un site web (version IG2) ?**

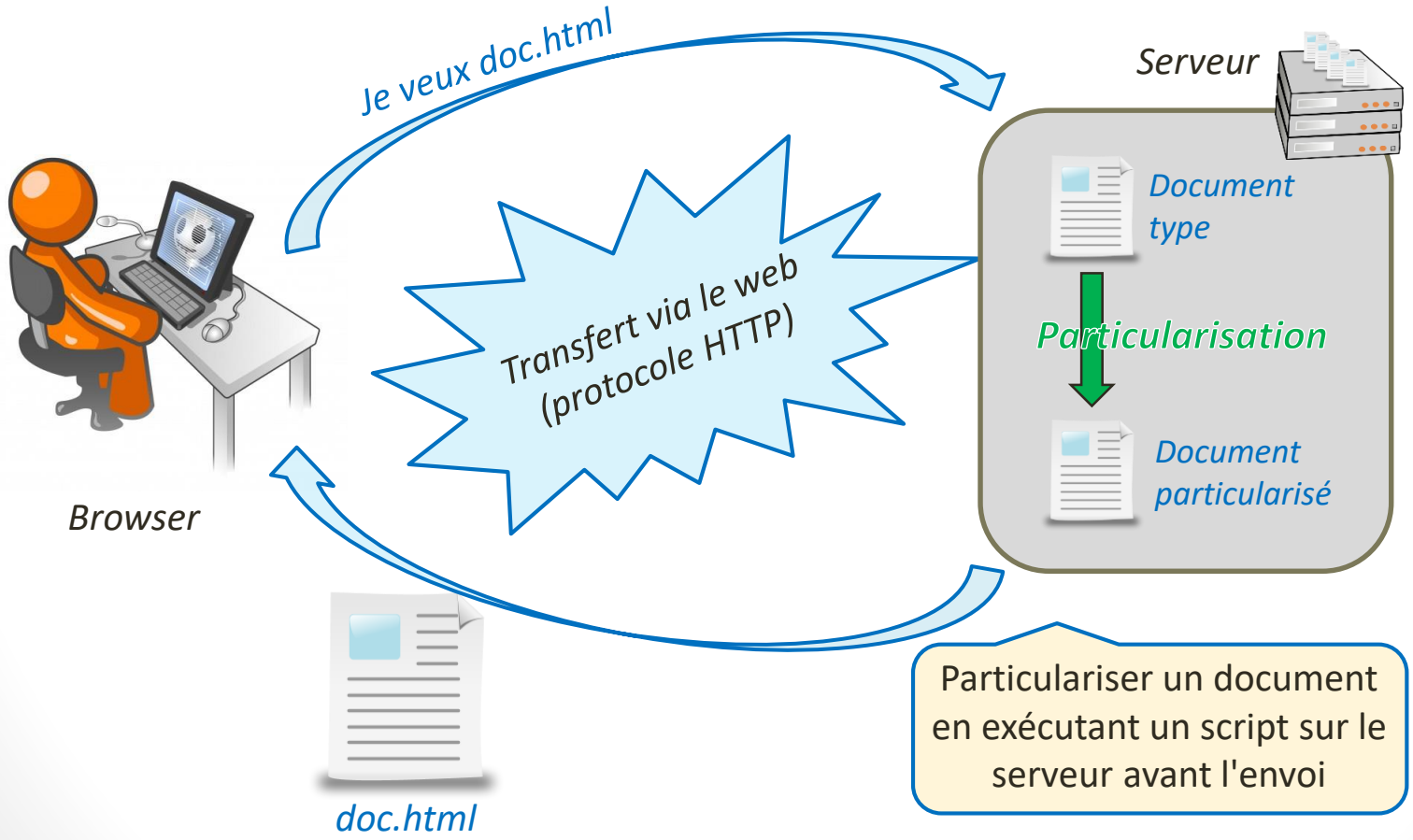


# Un site web HTML-dynamique

- **Quelles sont les ressources utilisées ?**
  - un navigateur utilisé par l'internaute
    - qui envoie les demandes,
    - réceptionne les réponses,
    - gère l'affichage (standards HTML et CSS) et
    - **exécute des scripts Javascript**.
  - un serveur
    - serveur = à la fois machine physique et logiciel
    - machine physique accessible via internet
    - logiciel-serveur capable de répondre aux demandes de fichiers

# Un site dynamique

- **Comment fonctionne un site web (version IG3) ?**



# Un site dynamique

- **Quelles sont les ressources utilisées ?**
  - un navigateur utilisé par l'internaute
    - qui envoie les demandes,
    - réceptionne les réponses,
    - gère l'affichage (standards HTML et CSS) et
    - exécute des scripts Javascript.
  - un serveur
    - serveur = à la fois machine physique et logiciel
    - machine physique accessible via internet
    - logiciel-serveur capable de répondre aux demandes de fichiers
    - **logiciel capable d'exécuter des scripts particularisant ces fichiers**.

      *C'est là qu'intervient PHP…*

# PHP, un premier aperçu

- **PHP « in use »**
  Un exemple d'utilisation
- **Fonctionnement général**
  À quoi sert PHP ?
- **Utilisations de PHP**
  Que peut-on faire avec PHP ?
- **PHP : logiciels compagnons et alternatives**

Ensuite : *Éléments d'architecture distribuée*

# PHP « in use »

- Structure du document de base :

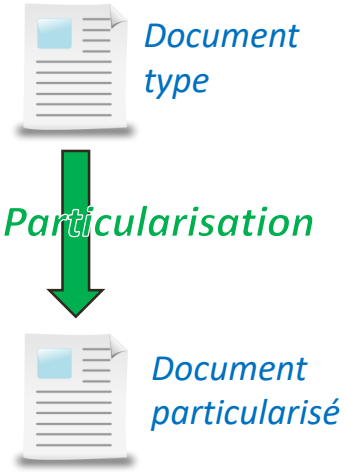
```
…
<h1>Bienvenue !</h1>
<p>Il est actuellement

<?php
  $heure = date('H:i');
  echo $heure;
?>

et tout va bien.</p>
…
```

~~doc.html~~
doc.php

Les scripts PHP peuvent être inclus directement dans le document-type.

# PHP « in use »

- Idée de base :
  - Le document de départ sera à la base un document HTML.
  - On va rendre certaines parties de ce document "variables".
  - Pour définir ces parties, on va donner un script permettant de générer le contenu attendu.



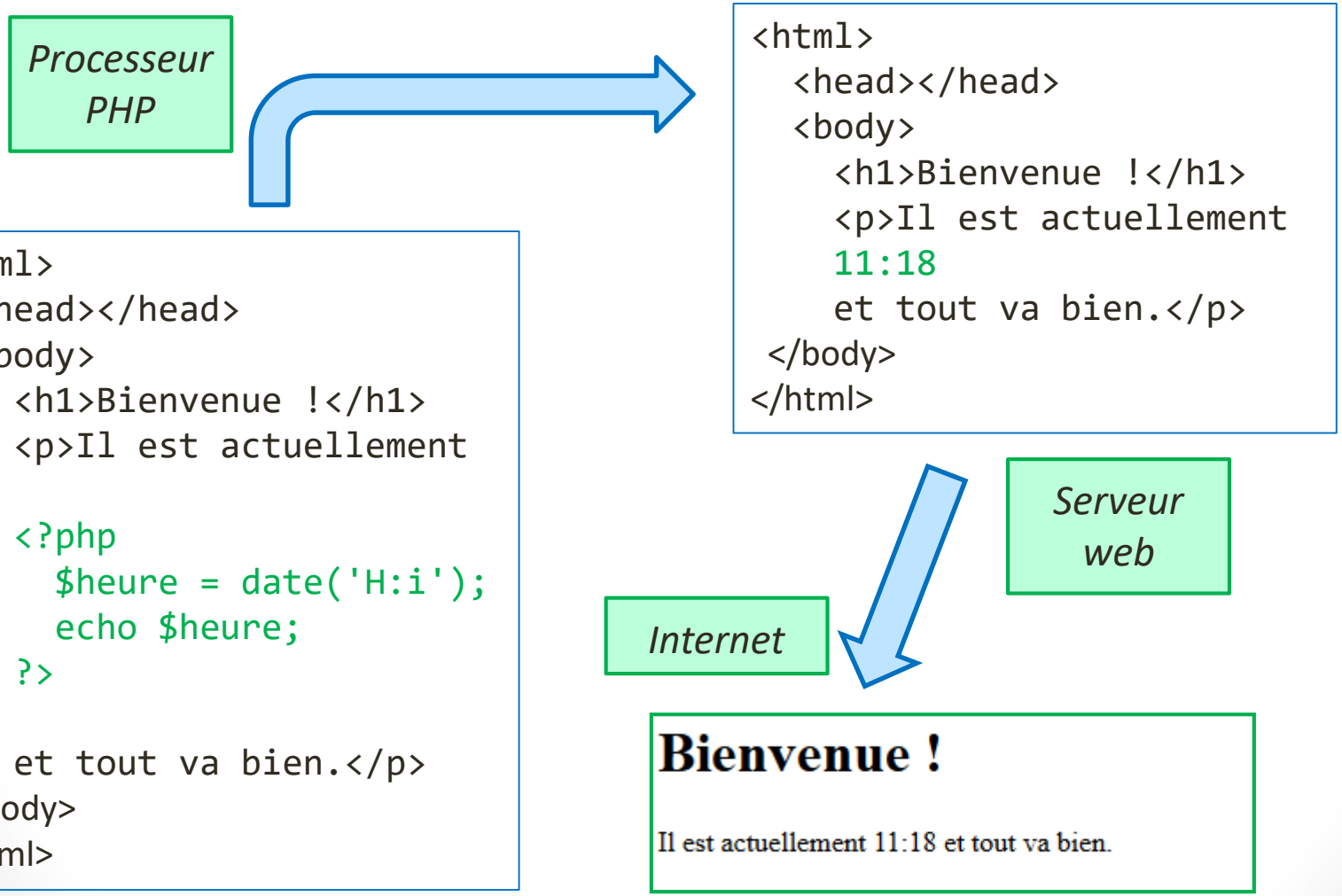

- Exemple :

```
<h1>Bienvenue !</h1>
<p>Il est actuellement 13:44 et tout va bien.</p>
```

Partie variable générée par un script

*Demander l'heure courante au système.*
*Sortir cette heure au format hh:mm.*

# PHP « in use »


Processeur PHP
<html>
<head></head>
<body>
<h1>Bienvenue !</h1>
Il est actuellement
<?php
$heure = date('H:i');
echo $heure;
?>
et tout va bien.
</body>
</html>
<html>
<head></head>
<body>
<h1>Bienvenue !</h1>
Il est actuellement
11:18
et tout va bien.
</body>
</html>
Serveur web
Internet
Bienvenue !
Il est actuellement 11:18 et tout va bien.

# PHP, fonctionnement général

- En cas d'envoi par le client d'une demande pour **un document avec l'extension php**, …

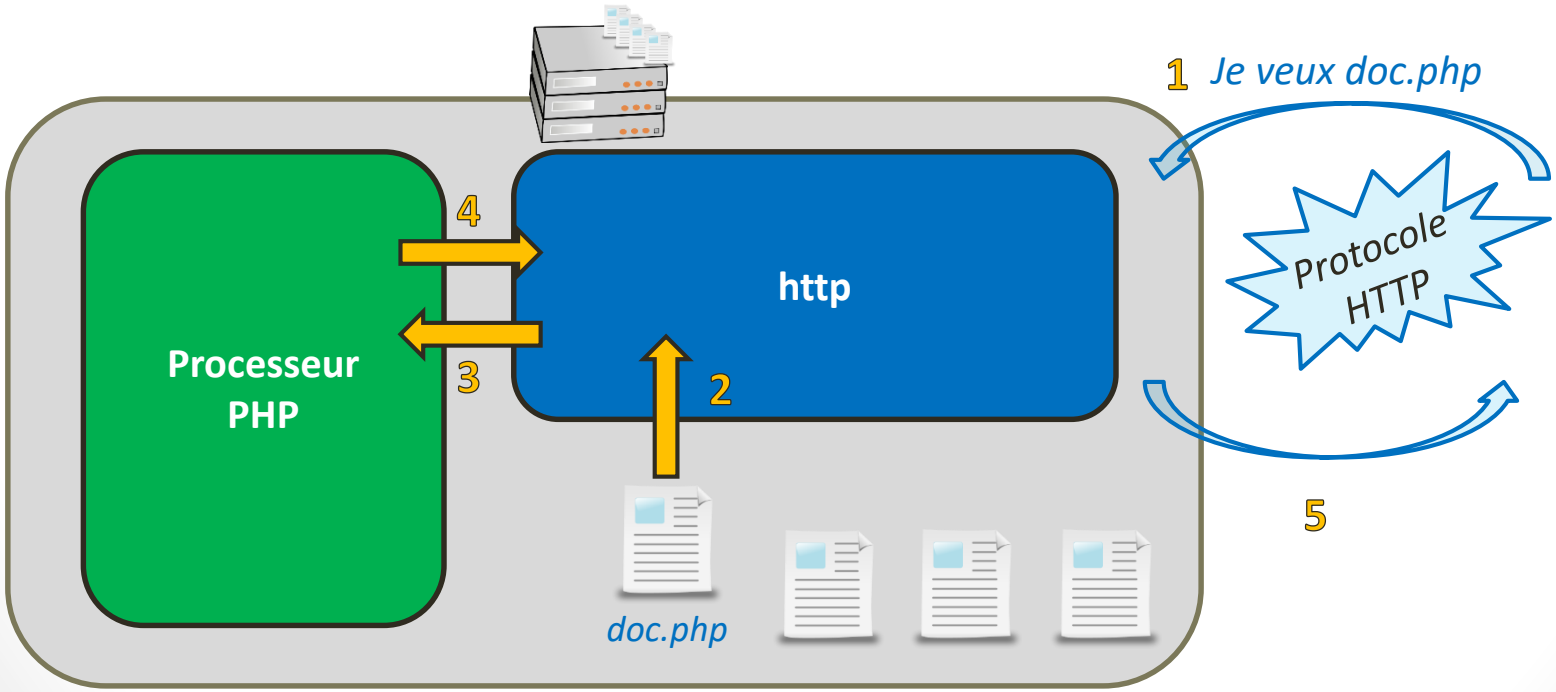
1) tout d'abord, le serveur web reçoit la demande ;
2) le serveur web récupère le fichier demandé ;
3) il passe la main au processeur PHP capable de comprendre et d'exécuter le code PHP ;
4) le code PHP est exécuté et remplacé par son résultat ; le fichier résultant (qui devrait être un fichier HTML correct, incluant peut-être du CSS et du Javascript) est renvoyé au serveur web ;
5) le serveur web envoie le fichier au demandeur via le web.

*Note.* Pour accomplir son travail, le processeur PHP peut éventuellement faire appel à une base de données ou à d'autres éléments présents sur le serveur.

# PHP, fonctionnement général

- Comment cela se passe-t-il du côté serveur ?

  *Note : serveur = machine physique sur laquelle tournent plusieurs programmes qu'on appelle parfois aussi serveurs (http, php…) !*



# PHP, fonctionnement général

- Deux extrêmes pour le code PHP (selon les goûts) :

Tout placer dans le code PHP

```
<?php
  echo '<!doctype html>';
  echo '<html><head></head>';
  echo '<body><p>Je calcule ';
  echo '17 + 15 = ';
  echo 17 + 15;
  echo '.</p></body></html>';
?>
```

Balise fermante optionnelle

N'y placer que le strict nécessaire

```
<!doctype html>
<html><head></head>
<body>
  <p>Je calcule  17 + 15 =
    <?php echo 17 + 15; ?>
  </p>
</body>
</html>
```

La balise fermante ?> absorbe le retour à la ligne qui la suit.

# Utilisations de PHP

- **PHP permet d'écrire des scripts côté serveur**.

  *Par opposition à Javascript : scripts côté client.*

- Le but des scripts PHP est de
  - construire de manière dynamique des pages web,
    - (éventuellement en utilisant une BDD ou d'autres éléments)
  - qui seront transmises au serveur http,
  - qui les enverra aux internautes.

*Note. Rien n'empêche un document-type (sur le serveur) de contenir des scripts PHP qui vont générer une page contenant de l'HTML et du Javascript ! Et ce code Javascript pourrait émettre des requêtes http qui vont revenir au serveur etc.*

# Utilisations de PHP

- (Jusqu'ici) PHP pour construire des **pages web dynamiques**
  - Requête : demande d'une page HTML
  - Réponse : un document HTML « particularisé »

- Il peut **en faire plus sur le serveur** :
  - PHP en tant que langage de programmation à part entière (contrairement à Javascript)
  - Applications plus complexes (gestion de BD) sur le serveur

- Il peut aussi renvoyer **autre chose qu'une page HTML** :
  - Requête : demande spécifique (évt avec arguments)
  - Réponse : un bout de code HTML
    un texte / une chaîne de caractères
    un document XML
    un document json

# Utilisations de PHP

- Contrairement à Javascript, **PHP n'interagit pas directement avec l'utilisateur** car PHP agit *avant* l'envoi de la page.
  - Pas de "alert", pas de "prompt"…
  - Pas d'interaction avec la fenêtre d'affichage, l'historique…

- Contrairement à Javascript, PHP peut
  - créer, ouvrir, lire, écrire et fermer des fichiers sur le serveur ;
  - interagir avec une base de données ;
  - contrôler ce à quoi les utilisateurs ont accès (version utilisateur et version admin d'un forum par exemple) ;
  - …

# Utilisations de PHP

- PHP n'a pas de mémoire ("**stateless**") : il oublie tout une fois le boulot terminé !

  - Un navigateur émet une requête pour un document.
    - Le serveur web reçoit la requête et se rend compte que c'est un document php.
    - Il contacte le processeur php qui génère un document html.
    - Le serveur web envoie le document html vers le navigateur.

  - Si le même navigateur ou un autre navigateur émet une nouvelle requête, celle-ci est traitée de manière indépendante de la première : PHP a tout oublié !

# Logiciels compagnons

- Que veut dire PHP ?

  **PHP** = PHP : a Hypertext Processor

  C'est un acronyme récursif, comme GNU (= Gnu is Not Unix) ou encore Linux (= Linux Is Not UniX).

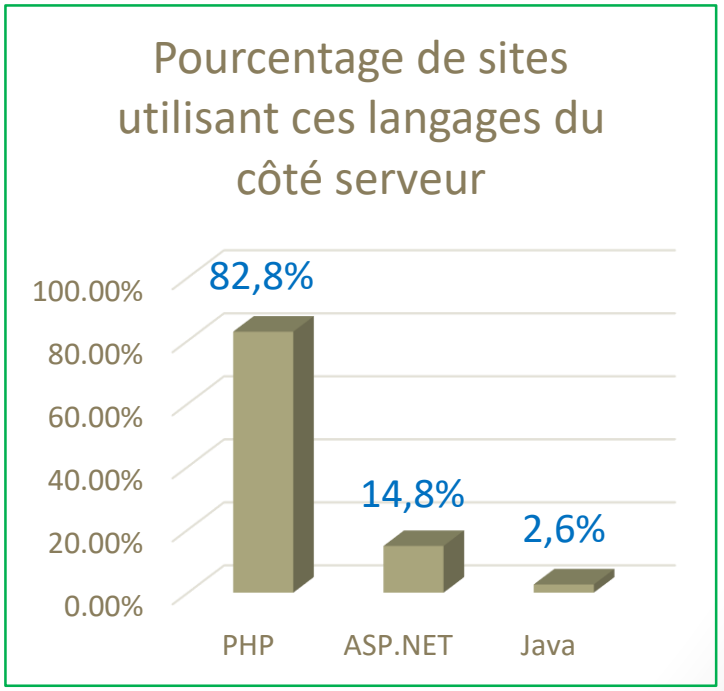
- Logiciel utilisé en conjonction avec PHP : le **serveur web Apache** (« a patchy server »)
  - Plus qu'un simple serveur web
  - Gère automatiquement les fichiers .php (aussi Perl)

- Package all-in-one :
  **Wamp** = **A**pache (http) + **P**HP + **M**ySQL sous **W**indows
  (aussi Xampp avec Perl)
  (configuration dans le fichier php.ini)

# Alternatives

- **Autre « monde » du web** :
  - ASP.NET (Active Server Page) avec du code C# (généralement)
  - géré par un serveur http IIS (Internet Information Service) ou PWS (Personal Web Server)

- Mais PHP est…
  - relativement simple
  - gratuit et ouvert.
  - portable.
  - très utilisé.
    82,8% des sites actuels utilisent du PHP du côté serveur.
    (source : w3techs.com, 09/2017)



# Architectures distribuées

Une application web se divise généralement en deux parties : une partie « client » et une partie « serveur ».

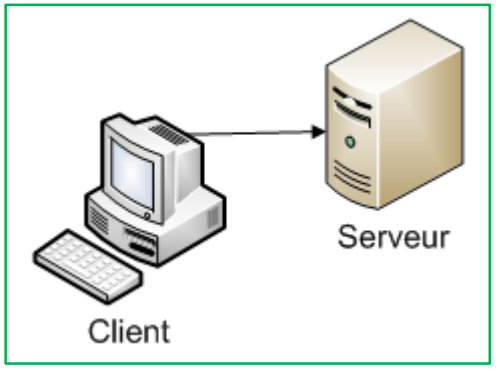
- Quel est le **vocabulaire associé à ce genre d'architecture** ?

Ensuite : *Organisation du cours*

# Architecture client/serveur



- On qualifie parfois ce genre de structure d'**architecture bipartite** (ou **2-tiers**) ou encore d'**architecture client/serveur**.

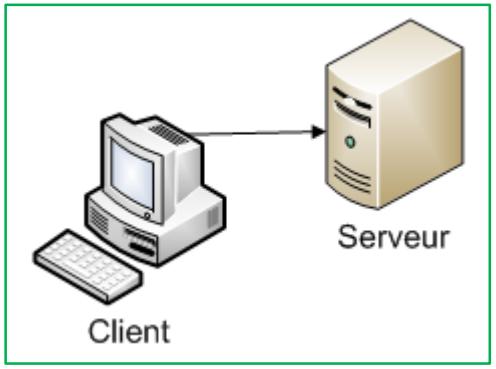



  - D'un côté, le client
    - qui initie le processus de connexion au serveur ;
    - qui envoie des requêtes au serveur et traite les réponses.
  - De l'autre, le serveur.
    - est à l'écoute des demandes de connexions et les gère ;
    - est à l'écoute des requêtes et les gère.

# Architecture client/serveur



- **Exemples d'architectures bipartites** :

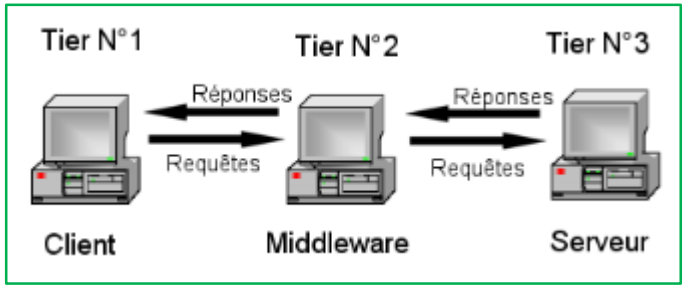



  - des navigateurs accédant à un site web situé sur un serveur (version « naïve »)
  - des terminaux reliés (physiquement) à un serveur central
  - une connexion P2P (peer-to-peer), où les rôles de client et de serveur peuvent alterner

# Architecture tripartite



- On ajoute parfois un 3e niveau gérant les communications et on parle d'**architecture tripartite** (ou **3-tiers**).
  - le client ;
  - le middleware, l'intermédiaire entre le client et le serveur.
  - le serveur.




- Souvent (mais pas toujours), cette découpe correspond à :
  - client = couche de présentation (« vue »)
  - middleware = couche fonctionnelle de traitement (« contrôleur »)
  - serveur = couche des données (« modèle »)

# Architecture tripartite

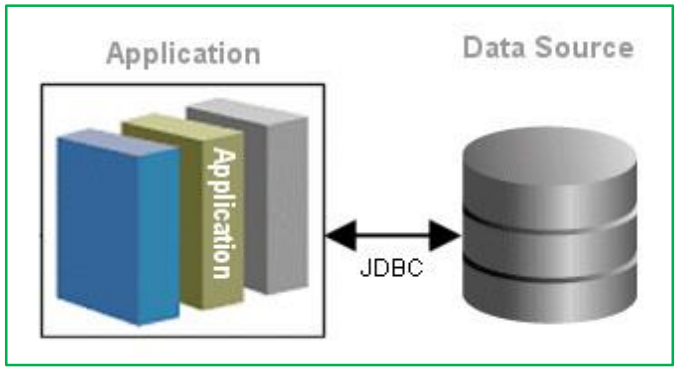


- **Exemples d'architectures tripartites :**
  - Navigateur ↔ Web ↔ Serveur

    *Site web (statique)*

    *Gadget affichant le cours de la Bourse ou la météo*
  - Programme ↔ Interface ↔ Base de données locale

    *Via Open/Java Database Connectivity (ODBC/JDBC)*

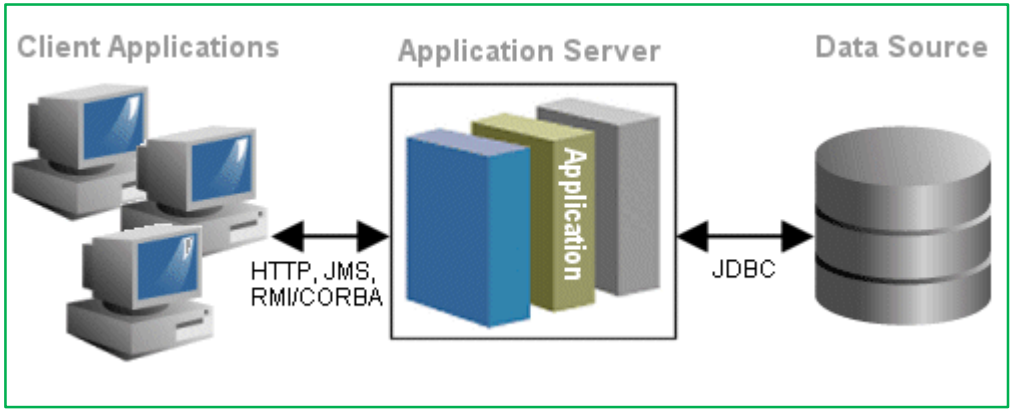


# Architecture tripartite

- **Exemples d'architectures tripartites (cas courant) :**
  - Navigateur ↔ Serveur web ↔ Base de données (similaire à MVC ?)

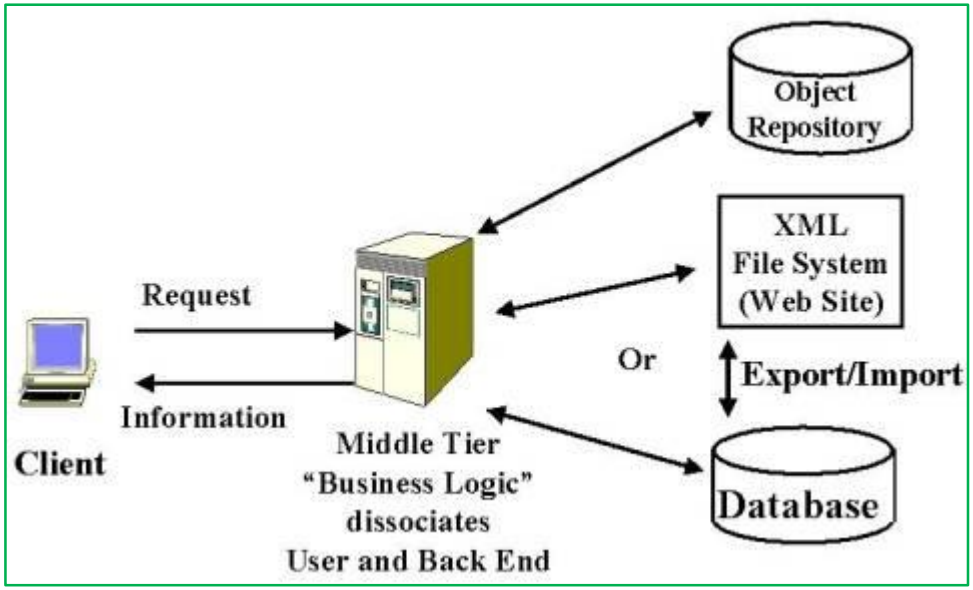
    *Site de vente en ligne*

    *Jeu multiplayer en ligne*



# Architecture tripartite

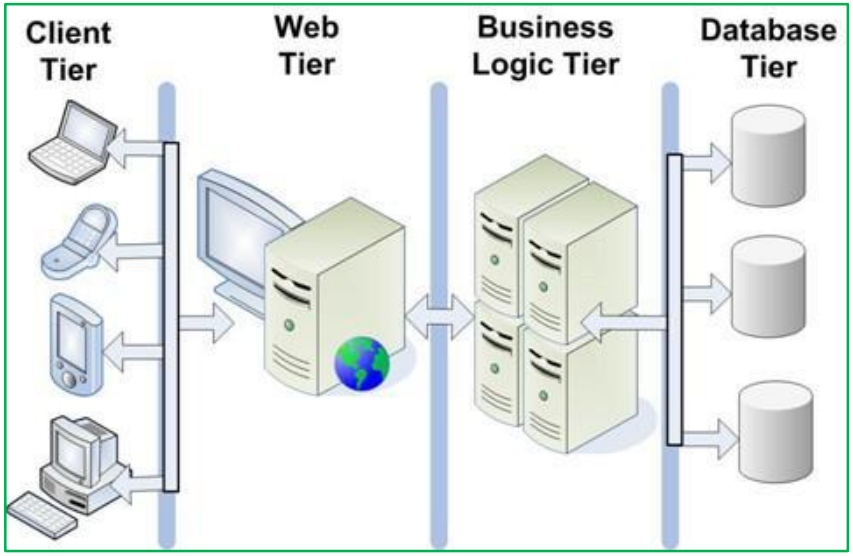
- **Exemples d'architectures tripartites (suite) :**



# Architecture n-tiers

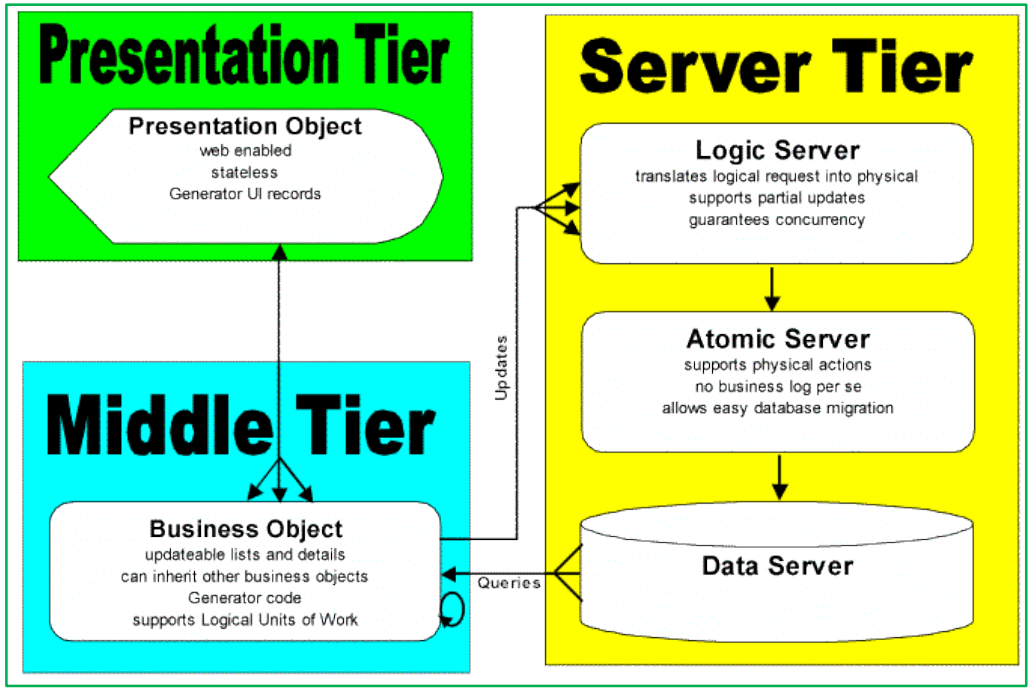
- S'il y a plusieurs autres niveaux intermédiaires, on parle d'**architecture multipartite** (ou **n-tiers**).

- **Exemples d'architectures n-tiers :**
  - Navigateur ↔ Serveur http ↔ Application ↔ Base de données



# Architecture n-tiers

- Un même système peut souvent être découpé de différentes manières selon ce qu'on veut mettre en évidence.



# Un peu plus de vocabulaire…

- Quand une des couches est utilisable/compatible avec plusieurs variantes dans les autres couches, on la qualifie parfois d'**universelle**.


  - Web = middleware universel (grâce à des standards de communication tels que TCP-IP, http, ftp…)

  - Site web = serveur universel (grâce à des standards HTML, CSS, XML, json… compris par tous [?] les navigateurs)

# Différents types de clients

- Le **client** peut être :
  - **léger** : le client se contente de servir d'interface affichant les réponses et envoyant les entrées de l'utilisateur, sans se préoccuper de leur signification.
    *Tous les traitements sont effectués sur le serveur.*
    Ex : site web statique
  - **lourd** : le client effectue lui-même une bonne partie des traitements.
    *Les traitements sont divisés entre le client et le serveur (plus ou moins à parts égales).*
    Ex : jeux Flash avec high scores conservés en ligne

# Différents types de clients

- Le **client** peut être :
  - **riche** : les traitements effectués par le client portent principalement sur l'affichage (graphiques sophistiqués) et la vérification.
    *La grande majorité des traitements sont effectués sur le serveur.*
    Ex : widget d'affichage de la météo ou du cours de la bourse

# Matière du cours



- **Matière/Planning**

| Contenu |
| --- |
| **Introduction et présentation du cours** |
| Labo 1 : introduction, environnement de travail |
| **Bases du langage PHP (1/2)*** |
| Labo 2 : bases de PHP |
| **Bases du langage PHP (2/2)*** |
| Labo 3 : bases de PHP (suite) |
| **Orienté objet en PHP*** |
| Labo 4 : orienté objet en PHP |
| **Échanges de données*** |
| Labo 5 : GET et POST |
| Labo 6 : cookies et sessions |
| Labo 7 : Ajax |

*(*) lire les slides à l'avance*

- Inclut également les matières des années précédentes !
  - IG1 (HTML/CSS) et IG2 (Javascript)

# Planning du cours

| Date | Contenu |
|---|---|
| **20/09 10h40** | **Introduction et présentation du cours** |
| *à domicile* | Labo 1 : introduction, environnement de travail |
| **02/10 8h30** | **Bases du langage PHP (1/2)*** |
| 09/10 8h30 | Labo 2 : bases de PHP |
| **16/10 8h30** | **Bases du langage PHP (2/2)*** |
| 23/10 8h30 | Labo 3 : bases de PHP (suite) |
| **6/11 8h30** | **Orienté objet en PHP*** |
| 13/11 8h30 | Labo 4 : orienté objet en PHP |
| **20/11 8h30** | **Échanges de données*** |
| 4/12 8h30<br>11/12 8h30<br>18/12 8h30 | Labo 5 : GET et POST<br>Labo 6 : cookies et sessions<br>Labo 7 : Ajax |
| | **** Slides à lire à l'avance !*** |

# Évaluation

- **Évaluation**
  - Examen écrit, à court ouvert, avec PC à disposition
  - Matière d'IG1 et d'IG2 incluses !!!

  - Attention : cours ouvert ≠ examen pour touriste !
  - Attention : le respect des règles du Clean Code reste de mise !
  - Attention : la qualité algorithmique est jugée également !

# Évaluation (critères)



| | | | | |
|---|---|---|---|---|
| ***Connaissance du vocabulaire et compréhension des concepts généraux*** | | | | |
| TRÈS INSUFFISANT<br>→ max 6/20 | INSUFFISANT<br>-1.5 | JUSTE SUFFISANT | SATISFAISANT<br>+1.5 | BIEN ACQUIS<br>+2.5 |
| *Concepts théoriques associés au langage PHP, aux applications web et à l'échange de données* | | | | |
| ***Maîtrise du langage PHP*** | | | | |
| TRÈS INSUFFISANT<br>→ max 6/20 | INSUFFISANT<br>-1.5 | JUSTE SUFFISANT | SATISFAISANT<br>+1.5 | BIEN ACQUIS<br>+2.5 |
| *Connaître et savoir utiliser les éléments de base et les particularités du langage PHP (hors OO)* | | | | |
| ***Utilisation de l'orienté objet en PHP*** | | | | |
| TRÈS INSUFFISANT<br>→ max 6/20 | INSUFFISANT<br>-1.5 | JUSTE SUFFISANT | SATISFAISANT<br>+1.5 | BIEN ACQUIS<br>+2.5 |
| *Connaître les mécanismes de l'orienté objet en PHP et savoir les mettre en pratique* | | | | |
| ***Gestion des échanges de données entre client et serveur*** | | | | |
| TRÈS INSUFFISANT<br>→ max 6/20 | INSUFFISANT<br>-1.5 | JUSTE SUFFISANT | SATISFAISANT<br>+1.5 | BIEN ACQUIS<br>+2.5 |
| *Connaître les mécanismes standards d'échange de données et savoir les utiliser en Javascript et en PHP* | | | | |
| ***Clean code et qualité du code*** | | | | |
| TRÈS INSUFFISANT<br>→ max 6/20 | INSUFFISANT<br>*pénalité* | JUSTE SUFFISANT | SATISFAISANT<br>*bonus* | |
| *Respect des règles du « clean code » et qualité des algorithmes et du code produits* | | | | |
| cote finale (valeur indicative) = 10 – 1.5 par INSUFFISANT, + 1.5 par SATISFAISANT, +2.5 par BIEN ACQUIS.<br>Si au moins un TRÈS INSUFFISANT : cote maximale de 6/20. Clean code & qualité modifie la cote finale. | | | | |