CMS - WORDPRESS

Module 15 : WordPress et PHP

Objectifs

- Découvrir l'interaction entre WordPress et PHP
- Découvrir le codex
- Découvrir la boucle et ses fonctions
  - bloginfo()
  - wp_nav_menu()
- Découvrir les filtres
- Découvrir les hooks
- Découvrir les actions
- Découvrir les globales
- Découvrir les classes
- Découvrir WP_Query
- Découvrir wpdb
- Découvrir les principales méthodes de requêtes pour la base de données

Le codex WordPress : guide de référence

- Le codex WordPress est le guide de référence officiel de WordPress.
- http://codex.wordpress.org
- http://codex.wordpress.org/fr:Accueil (version fr)
- https://developer.wordpress.org/reference,
- Le codex regroupe toutes les bonnes pratiques, servant à la modification de WordPress, à l'installation, à la configuration.
- C'est la documentation officielle pour les développeurs.
- WordPress est un CMS écrit en PHP
- Guide PHP: http://php.net/manual/fr/langref.php

Les fonctions d'inclusion

- Dans les anciens thèmes, WordPress utilisait les fonctions include() propres à PHP pour faire appel à un autre fichier.
  - <?php include(TEMPLATEPATH."/mon-fichier.php"); ?>
- Dans les thèmes récents, WordPress utilise ses propres fonctions d'inclusion :
  - **get_header()** : pour appeler le fichier header.php.
  - **get_footer()** : pour appeler le fichier footer.php.
  - **get_sidebar()** : pour appeler le fichier sidebar.php.
  - **get_searchform()** : pour appeler le fichier searchform.php.
  - **comments_template()** : pour appeler le fichier comments.php.
  - **get_template_part('slug')** : pour appeler le fichier slug.php.
  - **get_template_part('slug','nom')** : pour appeler le fichier slug-nom.php.
- *Exemples:*
  - <?php get_searchform(); ?>
    - La fonction get_searchform() fait appel au fichier searchform.php et affiche le formulaire de recherche.
  - <?php get_template_part('template-parts/content', 'page'); ?>
    - La fonction get_template_part() du fichier page.php, fait appel au fichier template-parts/content-page.php.

Les fonctions d'inclusion

- Selon les thèmes et l'utilisation des formats, dans certains thèmes, cf fonction suivante :
  - <?php get_template_part('content', get_post_format()); ?>
- La fonction get-template_part() fait appel au fichier content-*nomduformat*.php si les formats sont utilisés lors de l'édition des articles dans le bloc **Format**.
- La fonction get_post_format() permet de récupérer le nom du format.
- Si **Audio** est sélectionné dans le bloc **Format**, la fonction get_template_part() fait alors appel au fichier **content-audio.php**.
- Les formats doivent être activés par du code PHP dans le fichier **functions.php**, cela dépend donc des thèmes. L'utilisation des formats peut être différente d'un thème à l'autre.
- Vous pouvez très bien appeler le fichier **content-audio.php** en utilisant la forme suivante :
  - <?php get_template_part('content', 'audio'); ?>
  - Référence au codex : http://codex.wordpress.org/Include_Tags

Les marqueurs conditionnels

- Les marqueurs conditionnels permettent d'effectuer des actions dans des cas particuliers.
- Les conditions dans WordPress sont les mêmes qu'en PHP : if, elseif, else, endif.
- Elles s'écrivent de deux façons, la deuxième étant la plus fréquente dans WordPress :
  - if(ma_condition){ }elseif(ma_condition){ }else{ }
  - ou : if (ma_condition) : elseif (ma_condition) : else : endif;
- Il existe des fonctions conditionnelles propres à WordPress qui renvoient true ou false. Elles sont directement liées au type de fichier PHP (modèle de page) qui s'affichera sur le navigateur.
- Par exemple, is_home() vérifie que vous êtes sur la page d'accueil, ce qui correspond au fichier d'affichage index.php.

Les marqueurs conditionnels

- Voici quelques fonctions :
  - is_home() : pour la page principale d'accueil et les pages paginées, ce qui correspond au fichier index.php.
  - is_frontpage() : pour la page principale d'accueil, sans la pagination, ce qui correspond au fichier index.php.
  - is_page() : pour les pages, ce qui correspond au fichier page.php. Accepte comme argument l'id, le titre ou le permalien.
  - is_single() : pour les articles, ce qui correspond au fichier single.php. Accepte comme argument l'id, le titre ou le permalien.
  - is_tax() : pour les taxonomies s'il s'agit d'une catégorie ou d'une étiquette. Attention, cette fonction retourne false sur une page de catégorie et d'étiquette.

Les marqueurs conditionnels

- Voici quelques fonctions :
  - is_category() : pour les catégories, ce qui correspond au fichier category.php dans certains thèmes. Accepte comme argument l'id, le nom de la catégorie ou le permalien.
  - is_tag() : pour les étiquettes, ce qui correspond au fichier tag.php dans certains thèmes. Accepte comme argument l'id, le nom de l'étiquette ou le permalien.
  - is_author() : pour les auteurs, ce qui correspond au fichier author.php ou archive.php dans certains thèmes. Accepte comme argument l'id, le nom de l'auteur ou le permalien.
  - is_archive() : pour une page de type archive, ce qui correspond au fichier archive.php.
  - is_404() : pour une page 404, ce qui correspond au fichier 404.php.

Les marqueurs conditionnels

- Voici quelques fonctions :
  - is_search() : pour une recherche, ce qui correspond au fichier search.php.
  - is_paged() : pour une page paginée, ce qui correspond au fichier archive.php ou à la page d'accueil si celle-ci est paginée. Cela ne se réfère pas à un article ou une page dont le contenu serait divisé par l'utilisation de la balise HTML (quicktags) <!--nextpage-->.
  - comments_open() : quand les commentaires sont ouverts et acceptés, ce qui correspond au fichier comments.php.
  - have_posts() : quand il y a des articles. En général, cette fonction s'utilise avant la boucle WordPress (the loop) qui récupère le contenu des articles et des pages.
  - has_post_thumbnail() : quand l'article a une image. En général cette fonction s'utilise dans la boucle qui récupère le contenu des articles et des pages.
  - function_exists('nom_de_la_fonction') : quand la fonction existe (fonction propre à PHP).

  - Retrouvez la liste complète des marqueurs conditionnels sur le codex : http://codex.wordpress.org/fr:Marqueurs_conditionnels

La boucle et ses fonctions

- Boucle de WordPress = boucle while() en PHP.
- Récupère le contenu des pages et articles grâce aux fonctions WordPress have_posts() et the_posts().
- Présente dans presque tous les fichiers du thème.
- Sa particularité est d'afficher le contenu en fonction du fichier PHP (modèle de page) où elle se trouve, tout en tenant compte des différentes conditions et fonctions qu'elle intègre.
  - <?php while (have_posts()) : the_post(); //mes fonctions pour afficher du contenu endwhile; ?>
- À l'intérieur de la boucle :
  - Insérez des fonctions principales d'affichage propres à WordPress afin d'afficher le contenu,
  - Servez-vous des fonctions principales de récupération de variables pour créer vos propres fonctions.
- À l'intérieur d'une boucle, on peut mettre une fonction d'inclusion qui fait appel à un autre fichier.

La boucle et ses fonctions

- **Les principales fonctions d'affichage dans une boucle**
- Les fonctions d'affichage sont propres à WordPress et affichent directement le résultat.
- Ces fonctions commencent généralement par le préfixe **the_**.
  - **the_ID()** : affiche l'id d'un article ou d'une page. <?php the_ID(); ?>
  - **the_title()** : affiche le titre d'un article ou d'une page. <?php the_title($before,$after,$echo); ?>
  - La fonction peut recevoir des arguments facultatifs :
    - **$before** : texte avant le titre, en général du HTML comme les balises H.
    - **$after** : texte après le titre, en général la ou les balises HTML fermantes. Attention à respecter la structure HTML : si vous utilisez pour $before une balise <h1> ouvrante, il faut absolument que $after utilise la balise </h1> fermante.
    - **$echo** : accepte une valeur booléenne true ou false pour l'affichage. Par défaut : true.
  - **the_content()** : affiche le contenu d'un article ou d'une page. Vous pouvez ajouter la balise HTML (quicktags) <!--more-->. Un lien **Lire la suite** s'affiche alors pour que le lecteur puisse lire la fin de l'article ou de la page.

La boucle et ses fonctions

- **Les principales fonctions d'affichage dans une boucle**
- **the_content()** : affiche le contenu d'un article ou d'une page. Vous pouvez ajouter le commentaire HTML (quicktags) **<!--more-->**. Un lien **Lire la suite** s'affiche alors pour que le lecteur puisse lire la fin de l'article ou de la page. <?php the_content($more,$aftermore); ?>
  - La fonction peut recevoir des arguments :
    - **$more** : texte à afficher pour le lien permettant de voir la suite du contenu.
    - **$aftermore** : accepte une valeur booléenne true ou false, pour l'affichage du texte $more. Par défaut : false.
- **the_excerpt()** : affiche l'extrait d'un article ou d'une page, si le bloc **Extrait** est renseigné.

```
<?php the_excerpt(); ?>
```

- **the_permalink()** : affiche l'URL du lien d'un article ou d'une page.

La boucle et ses fonctions

- **Les principales fonctions d'affichage dans une boucle**
- **the_post_thumbnail()** : affiche l'image à la Une d'un article ou d'une page, si le bloc **Image à la Une** contient une image. S'utilise avec la fonction **has_post_thumbnail(),** permettant de vérifier la présence d'une image.
  - Exemple : <?php if(has_post_thumbnail()){   the_post_thumbnail();   } ?>
  - La fonction peut recevoir des arguments :
    - **$size** : le format de l'image. Accepte thumbnail, medium, large ou full.
    - **$arguments** : accepte un tableau (array) avec différents paramètres : largeur, hauteur, src, class, alt, title.
  - Pour utiliser cette fonction dans le thème, il faut la déclarer dans le fichier **functions.php**
  - add_theme_support('post-thumbnails');
  - En général, la plupart des thèmes gèrent et intègrent cette fonction.
- **the_author()** : affiche le nom de l'auteur sous forme de lien cliquable renvoyant sur la page de l'auteur.

```
<?php the_author(); ?>
```

La boucle et ses fonctions

- **the_time**() : affiche la date de publication d'un article ou d'une page. <?php the_time($d); ?>
  - La fonction peut recevoir un argument :
    - $d : accepte les formats de date PHP. Par défaut, la configuration de l'administration définie dans le menu **Réglages - Général - Format de date**.
- **the_category**() : affiche les catégories sous forme de liens renvoyant vers la page qui liste les articles appartenant à cette catégorie.
    - <?php the_category($separator,$parents); ?>
  - La fonction peut recevoir des arguments :
    - $**separator** : texte ou caractère s'affichant entre les liens. Généralement la virgule « , » est utilisée.
    - $**parents** : affiche la relation parent/enfant, accepte multiple ou single.
- **the_tags**() : affiche les étiquettes de l'article ou de la page sous forme de liens renvoyant vers la page qui liste les articles correspondant à cette étiquette.
  - La fonction peut recevoir des arguments :
    - $**before** : texte, avec ou sans HTML, avant le premier lien ; par défaut « tags : ».
    - $**separator** : texte ou caractère s'affichant entre les liens ; par défaut, la virgule « , ».
    - $**after** : texte après le dernier lien et/ou balise HTML fermante. Attention à respecter la structure HTML : si vous utilisez pour $**before** une balise ouvrante, il faut absolument que $after utilise la balise fermante.

La boucle et ses fonctions

- **edit_post_link()** : affiche le lien vers l'administration de la page ou de l'article concerné.
  - La fonction peut recevoir des arguments :
    - $**name** : titre du lien ; par défaut « éditer ».
    - $**before** : texte avant le lien, en général du HTML comme la balise .
    - $**after** : texte après le lien et ou la balise HTML fermante. Attention à respecter la structure HTML : si vous utilisez pour $before une balise ouvrante, il faut absolument que $after utilise la balise fermante.

La boucle et ses fonctions

- **wp_link_pages**() : affiche des liens d'un article ou d'une page suivante ou précédente.
  - La fonction peut recevoir un argument :
    - **$arguments** : accepte un tableau avec différents arguments. Par défaut :
    - **before** : texte HTML avant le premier lien.
    - **after** : texte HTML après le dernier lien.
    - **link_before** : texte HTML avant le lien.
    - **link_after** : texte HTML après le lien.
    - **next_or_number** : indique le numéro des pages ou d'un texte, accepte next ou number.
    - **separator** : texte ou caractère s'affichant entre les liens.
    - **nextpagelink** : texte vers la page suivante.
    - **previouspagelink** : texte vers la page précédente.
    - **pagelink** : nombre de liens de pages, accepte un nombre. « % » correspond à un nombre de liens illimité.
    - **echo** : accepte une valeur booléenne true ou false pour l'affichage. Par défaut : true.

La boucle et ses fonctions

- Les fonctions pour récupérer des variables sont propres à WordPress et permettent de récupérer le contenu sous forme de variables.
- Les fonctions commencent généralement par le préfixe **get_** et nécessitent d'utiliser la fonction **echo** propre à PHP pour afficher le résultat.
- Récupérer des variables d'articles ou de pages vous est utile lors de l'élaboration de vos propres fonctions.
- get_the_ID() : récupère l'id d'un article ou d'une page. <?php $id=get_the_ID(); ?>
- get_the_title() : récupère le titre d'un article ou d'une page.

<?php $title=get_the_title($id); ?>

  - La fonction peut recevoir un argument :
    - $**id** : l'id d'un article ou d'une page. En absence d'id, la fonction retourne le titre de la page courante.

La boucle et ses fonctions

- get_the_content() : récupère le contenu d'un article ou d'une page. Ajouter le commentaire HTML <!--more-->. Un lien **Lire la suite** s'affiche alors pour que le lecteur puisse lire la fin d'un article ou d'une page.

  <?php $content=get_the_content($more, $aftermore); ?>

  - La fonction peut recevoir des arguments :
    - $**more** : texte à afficher pour le lien permettant de voir la suite du contenu.
    - $**aftermore** : accepte une valeur booléenne true ou false pour l'affichage du texte $more. Par défaut : false.
- get_the_excerpt() : récupère l'extrait d'un article ou d'une page.
- get_permalink() : récupère l'URL du lien d'un article ou d'une page.
  - La fonction peut recevoir un argument :
    - $id : l'id d'un article ou d'une page. En absence d'id, la fonction retourne le lien de la page courante.

La boucle et ses fonctions

- get_the_category() : récupère le ou les liens des catégories d'un article ou d'une page sous forme de lien HTML <a href> renvoyant sur la page qui liste les articles appartenant à cette catégorie. <?php $category=get_the_category($id); ?>
  - La fonction peut recevoir un argument :
    - $id : l'id d'un article ou d'une page.
- get_the_tags() : le ou les liens des étiquettes d'un article ou d'une page sous forme de lien HTML <a href> renvoyant sur la page qui liste les articles correspondant à cette étiquette. <?php $tags=get_the_tags($id); ?>
  - La fonction peut recevoir un argument :
    - $**id** : l'id d'un article ou d'une page.

La boucle et ses fonctions

- get_edit_post_link() : récupère le lien vers l'administration d'un article ou d'une page concerné sous forme de lien HTML <a href>.
  - <?php $edit=get_edit_post_link($name,$before,$after,$id); ?>
  - La fonction peut recevoir des arguments :
    - **$name** : titre du lien ; par défaut « éditer ».
    - **$before** : texte avant le lien, en général du HTML comme la balise .
    - **$after** : texte après le lien. Attention à respecter la structure HTML : si vous utilisez pour $before une balise ouvrante, il faut absolument que $after utilise la balise fermante.
    - **$id** : l'id d'un article ou d'une page.
    - Les deux exemples suivants affichent la même chose :
      - <?php the_title(); ?>
      - <?php echo get_the_title(); ?>

Les fonctions pour les textes dans les fichiers PHP

- Insérer du texte brut dans les fichiers PHP est possible, malheureusement les caractères spéciaux et accentués ne sont pas pris en compte. Il faut utiliser les codes ASCII ou les convertir en UTF-8, grâce à une fonction PHP classique ou bien utiliser les fonctions de WordPress.
- L'avantage d'utiliser les fonctions de WordPress est que celles-ci ont été spécialement conçues pour gérer le multilingue, grâce à un fichier de traduction.
- Deux fonctions WordPress sont récurrentes dans la plupart des thèmes : __() et _e().
  - __() : retourne une variable.
  - _e() : fait un echo de la variable, ce qui revient à faire echo __().
    - <?php _e($text, $domain); ?> <?php __($text, $domain); ?>
  - Les fonctions acceptent deux arguments :
    - **$text** : le texte à traduire.
    - **$domain** : le nom clé du fichier où se trouvent les chaînes de traduction.
  - Exemple : <?php _e('Nothing Found', 'twentytwelve'); ?>
  - Ce code affiche à l'écran le texte « Nothing Found », mais le texte apparaît en français : « Rien Trouvé ».

Les fonctions pour les textes dans les fichiers PHP

- WordPress cherche le fichier twentyseventeen.po se trouvant dans le dossier **wp-content/language/themes/twentyseventeen-fr_FR.po**, pour les thèmes natifs de WordPress.
- Pour les thèmes personnalisés, le fichier de langue se trouve directement dans le dossier **wp-content/themes/mon-theme/language/fr_FR.mo**.

Les fonctions pour les textes dans les fichiers PHP

- Pour les thèmes personnalisés, le fichier de langue se trouve directement dans le dossier :

wp-content/themes/mon-theme/language/fr_FR.mo.

  - **fr_FR** indique que le fichier correspond au français. Vous pouvez donc trouver, lors d'une installation de WordPress en espagnol, un fichier s'appelant twentytseventeen-es_ES.po, par exemple.
  - Si aucun fichier de langue n'existe, la fonction prend pour texte le premier argument $text. De base, WordPress utilise l'anglais, ainsi le fichier en_US n'existe pas.
  - WordPress sait que vous utilisez la version française grâce au fichier **wp-config**.
  - La ligne **define('WPLANG', 'fr_FR');** indique que le site est en français.
  - Si vous supprimez de la fonction « fr_FR », soit define('WPLANG', '');, votre site WordPress devient alors intégralement anglophone.

Les fonctions pour les textes dans les fichiers PHP

- Il existe d'autres fonctions pour les textes dans WordPress, qui fonctionnent de la même façon : _n(), _x(), _ex(), _nx()
  - _n() : permet de retourner la forme au singulier ou au pluriel, selon une quantité : si la quantité est 1, le mot sera au singulier ; si la quantité est supérieure à 1, le mot sera au pluriel. Renvoie une variable.
  - _x() : permet de préciser un contexte pour l'internationalisation, si vous avez besoin de traduire un mot de deux façons différentes. Renvoie une variable.
  - _ex() : fait un echo de la fonction _x() ; revient à faire echo _x().
  - _nx() : est un mélange des deux fonctions précédentes.
- Les fonctions _en() et enx() n'existent pas.

Les chemins dans les URLs

- Plusieurs fonctions propres à WordPress permettent d'afficher les chemins des différents dossiers.
- Voici les différentes fonctions :
  - site_url() : renvoie l'adresse du site web sous forme de variable.
    - Exemple : http://www.monsite.com
  - home_url() : renvoie l'adresse de la racine du site WordPress sous forme de variable.
    - Exemple : http://www.monsite.com ou http://www.monsite.com/dossier-Wordpress
  - admin_url() : renvoie l'adresse de l'administration sous forme de variable.
    - Exemple : http://www.monsite.com/wp-admin
  - includes_url() : renvoie le chemin du dossier wp-includes sous forme de variable.
    - Exemple : http://www.monsite.com/wp-includes

Les chemins dans les URLs

- content_url() : renvoie le chemin du dossier wp-content sous forme de variable.
  - Exemple : http://www.monsite.com/wp-content
- plugins_url() : renvoie le chemin du dossier plugins sous forme de variable.
  - Exemple : http://www.monsite.com/wp-content/plugins
- theme_url() : renvoie le chemin du dossier themes sous forme de variable.
  - Exemple : http://www.monsite.com/wp-content/themes
- wp_upload_dir() : renvoie le chemin du dossier uploads sous forme de variable.
  - Exemple : http://www.monsite.com/wp-content/uploads

Les chemins dans les URLs

- Vous trouvez également les fonctions :
  - get_template_directory_uri() : dans le cas d'un thème enfant, renvoie l'URL du dossier du thème parent, sinon renvoie le dossier du thème actif.
    - Exemple : s'il s'agit d'un thème enfant : http://www.monsite.com/wp-content/themes/mon-theme
    - sinon : http://www.monsite.com/wp-content/themes/mon-theme
  - get_stylesheet_directory_uri() : dans le cas d'un thème enfant, renvoie l'URL du dossier du fichier style.css enfant, sinon renvoie le dossier du fichier style.css du thème actif.
    - Exemple : S'il s'agit d'un thème enfant : http://www.monsite.com/wp-content/themes/mon-theme-enfant
    - sinon : http://www.monsite.com/wp-content/themes/mon-theme
- Les fonctions bloginfo($param) et get_blog info($param) peuvent également renvoyer un chemin d'URL.

La fonction bloginfo()

- bloginfo() = récupére diverses informations sur le blog.
- get_bloginfo() = renvoie une variable
- bloginfo() affiche le résultat.
- La fonction demande un argument obligatoire : $param, pour renvoyer un résultat :
  - <?php bloginfo($param); ?>
  - <?php $bloginfo=get_bloginfo($param); echo $bloginfo ?>
  - bloginfo('name') : permet d'afficher le titre du site. Cf champ **Titre du site** dans **Réglages - Général**. Cf table wp_options.
  - bloginfo('description') : permet d'afficher le slogan du site. Cf. champ **Slogan** dans **Réglages - Général**, Info dans la table wp_options.
  - bloginfo('wpurl') : permet d'afficher l'URL du site. Cf. **Adresse web de WordPress** dans **Réglages - Général**, table wp_options. Cette fonction équivaut à faire un echo de la fonction site_url().

La fonction bloginfo()

- bloginfo('url') : permet d'afficher l'adresse de la page d'accueil, parfois différente de l'URL du site. Cf. **Adresse web du site** dans **Réglages - Général**, table wp_options. Cette fonction équivaut à faire un echo de la fonction home_url().
- bloginfo('admin_email') : permet d'afficher l'adresse e-mail du site. Cf. **Adresse de messagerie** dans **Réglages - Général**, table wp_options.
- bloginfo('charset') : permet d'afficher le charset qui définit l'encodage des caractères de la page, par défaut utf-8. Dans le fichier header.php, on retrouve cette balise HTML méta charset, par exemple : <meta charset="<?php bloginfo('charset'); ?>">. Vous pouvez modifier le charset dans le fichier wp_config.php.
- bloginfo('version') : permet d'afficher la version de votre site WordPress. Vous pouvez aussi utiliser la variable $wp_version, en tant que globale. Cf. table wp_options.

La fonction bloginfo()

- bloginfo('html_type') : permet d'afficher le type de page HTML, par défaut text/html. Cf. table wp_options.
- bloginfo('text_direction') : permet d'afficher la direction du texte. Par défaut : ltr(left to right).
- bloginfo('language') : permet d'afficher la langue du site. Vous pouvez la modifier dans le fichier wp-config.php.
- bloginfo('stylesheet_url') : permet d'afficher l'URL de la feuille de style style.css du thème actif. Équivaut à un echo de la fonction get_stylesheet_uri().
  - Exemple : http://www.monsite.com/wp-content/themes/mon-theme/style.css
- bloginfo('stylesheet_directory') : permet d'afficher l'URL du dossier de la feuille de style style.css du thème actif. Équivaut à un echo de la fonction get_stylesheet_directory_uri().
  - Exemple : http://www.monsite.com/wp-content/themes/mon-theme

La fonction bloginfo()

- bloginfo('template_url') ou bloginfo ('template_directory') : permet d'afficher l'URL du thème actif. Dans le cas d'un thème enfant, bloginfo('template_url')renvoie le dossier du thème parent. Équivaut à echo de la fonction get_template_ directory_uri().
  - Exemple : http://www.monsite.com/wp-content/themes/mon-theme
- bloginfo('pingback_url') : permet d'afficher l'URL du fichier XML-RPC Pingback. Dans le fichier header.php, on retrouve cette ligne de code : <link rel="pingback" href="<?php bloginfo('pingback_ url'); ?>" />
  - Exemple : http://www.monsite.com/xmlrpc.php
- bloginfo('atom_url') : permet d'afficher l'URL du flux Atom.
  - Exemple : http://www.monsite.com/feed/atom

La fonction bloginfo()

- bloginfo('rdf_url') : permet d'afficher l'URL du flux RDF/RSS 1.0.
  - Exemple : http://www.monsite.com/feed/rfd
- bloginfo('rss_url') : permet d'afficher l'URL du flux RSS 0.92.
  - Exemple : http://www.monsite.com/feed/rss
- bloginfo('rss_url2') : permet d'afficher l'URL du flux RSS 2.0.
  - Exemple : http://www.monsite.com/feed
- bloginfo('comments_atom_url') : permet d'afficher les commentaires du flux Atom.
  - Exemple : http://www.monsite.com/comments/atom
- bloginfo('comments_rss2_url') : permet d'afficher les commentaires du flux RSS 2.0.
  - Exemple : http://www.monsite.com/comments/feed

- Les fonctions bloginfo('siteurl') et bloginfo('home') sont des fonctions dépréciées par WordPress, utilisez plutôt **bloginfo('url').**

WordPress et PHP

La fonction wp_nav_menu()

- WordPress gère très facilement les menus et permet d'en créer à volonté via le panel d'administration dans **Apparence - Menus**.
- La fonction wp_nav_menu() permet d'afficher n'importe quel menu créé dans l'administration **Apparence - Menus**. Le menu principal du blog utilise donc cette fonction. Elle se trouve en général dans le fichier **header.php**, mais cela peut varier selon les thèmes.
- Dans la plupart des cas, si aucun menu personnel n'est créé en tant que menu principal, WordPress affiche automatiquement les pages par ordre alphabétique.
  - <?php wp_nav_menu($defaults) ; ?>
- La fonction peut recevoir un ou plusieurs arguments optionnels, classés à l'intérieur d'un tableau.
- Notez qu'un menu se compose de la structure HTML suivante :

```php
<?php

$defaults = array(
 'theme_location' => '',
 'menu' => '',
 'container' => 'div',
 'container_class' => '',
 'container_id' => '',
 'menu_class' => 'menu',
 'menu_id' => '',
 'echo' => true,
 'fallback_cb' => 'wp_page_menu',
 'before' => '',
 'after' => '',
 'link_before' => '',
 'link_after' => '',
 'items_wrap' => '<ul id="%1$s" class="%2$s">%3$s</ul>',
 'depth' => 0,
 'walker' => ''
);

wp_nav_menu( $defaults );

?>
```

```html
<ul>
   <li><a href='http://monlien.com'>nom de mon lien</a></li>
   <li><a href='http://monlien2.com'>nom de mon lien2</a></li>
</ul>
```

La fonction wp_nav_menu()

- **menu_id** : accepte un nom d'id pour la balise HTML <ul> englobant les balises HTML <li>.
  - Exemple : 'menu_id' => 'id-du-menu'
- **echo** : accepte true ou false. Permet d'afficher ou d'utiliser le menu en tant que variable, par défaut : true.
  - Exemple : 'echo' => '0' ou 'echo' => 'false'
- **fallback_cb** : accepte un nom, retourné si le menu n'existe pas. Utilisez false pour désactiver cet argument, par défaut : wp_page_menu.
  - Exemple : 'fallback_cb' => 'false'

La fonction wp_nav_menu()

- **before** : accepte un texte s'affichant avant la balise HTML <a href> (lien de chaque onglet).
  - Exemple : 'before' => '>'
- **after** : accepte un texte s'affichant après la balise HTML <a href> (lien de chaque onglet).
  - Exemple : 'after' => '<'
- **link_before** : accepte un texte s'affichant avant le texte du lien dans la balise HTML <a href> (lien de chaque onglet). Le texte est donc cliquable.
  - Exemple : 'link_before' => '>'
- **link_after** : accepte un texte s'affichant après le texte du lien dans la balise HTML <a href>(lien de chaque onglet). Le texte est donc cliquable.
  - Exemple : 'link_after' => '<'

La fonction wp_nav_menu()

- Voici le détail des différents arguments :
- theme_location : accepte le nom de l'emplacement. Ce champ permet d'enregistrer un menu à un emplacement précis.
- Dans l'administration l'emplacement apparaît dans **Apparence - Menus - Gérer les emplacements**. Il doit au préalable être défini grâce à la fonction register_nav_menus() Exemple : 'menu' => 'le-nom-de-emplacement-de-mon-menu'
- menu : accepte le nom ou l'id d'un menu créé au préalable ou existant dans l'administration, via **Apparence - Menus**.
  - Exemple : 'menu' => 'le-nom-de-mon-menu'

```
<span>
   <ul>
        <li><a href='http://monlien.com'>nom de mon lien</a></li>
        <li><a href='http://monlien2.com'>nom de mon lien2</a></li>
   </ul>
</span>
```

La fonction wp_nav_menu()

- **container** : accepte le nom de la balise HTML contenant le menu. Par défaut : . Pour n'avoir aucune balise, mettez la valeur à false.
  - Exemple : 'container' => 'span'
- **container_class** : accepte un nom de classe pour la balise contenant le menu.
  - Exemple : 'container_class' => 'classe-du-conteneur'
- **container_id** : accepte un nom d'id pour la balise contenant le menu.
  - Exemple : 'container_id' => 'id-du-conteneur'

```
<span class='class-du-conteneur'>
   <ul>
        <li><a href='http://monlien.com'>nom de mon lien</a></li>
        <li><a href='http://monlien2.com'>nom de mon lien2</a></li>
   </ul>
</span>
```

```
<span id='id-du-conteneur'>
   <ul>
        <li><a href='http://monlien.com'>nom de mon lien</a></li>
        <li><a href='http://monlien2.com'>nom de mon lien2</a></li>
   </ul>
</span>
```

La fonction wp_nav_menu()

- items_wrap : accepte le même format que la fonction PHP sprintf(). Ce type de format peut inclure des variables : %1$s qui correspond à l'argument menu_id, %2$s qui correspond à l'argument menu_class, et %3$s à la valeur de la balise HTML <li> contenant le lien de l'onglet. Par défaut : <ul id="%1$s" class="%2$s">%3$s
- Exemples :
  - 'items_wrap' => '%3$s'. Dans ce cas, seule la liste avec les balises HTML <li>, sans les balises HTML <ul> l'englobant, s'affiche. Il faut donc ajouter les balises HTML <ul> dans le fichier appelant la fonction.

```
<ul>
   <?php wp_nav_menu(array('items_wrap' => '%3$s')); ?>
</ul>
```

  - 'items_wrap' => '
Menu : %3$s
'. Dans ce cas, vous pouvez placer un mot qui s'intègre dans la liste avec des balises HTML <li>.
  - Cet exemple donne le code HTML suivant :

```
<ul>
   <?php
wp_nav_menu(
array('items_wrap' => '<ul><li>Menu : </li>%3$s</ul>')
);
   ?>
</ul>
```

```
<ul>
   <li>Menu :</li>
   <li><a href='http://monlien.com'>nom de mon lien</a></li>
   <li><a href='http://monlien2.com'>nom de mon lien2</a></li>
</ul>
```

La fonction wp_nav_menu()

- **depth** : accepte un chiffre permettant de gérer les niveaux du menu.
  - Par défaut : 0, qui affiche tout le menu. 1 désactive tous les sous-menus à partir du niveau 1.
- Dans le cas de sous-sous-menus, 2 désactive tous les sous-sous-menus à partir du niveau 2 et laisse les sous-menus de niveau 1.
- -1 affiche tous les onglets au même niveau.
  - Exemple : 'depth' => '1'
- Structure d'un menu sur 1 niveau :
- **walker** : accepte un objet, pas une chaîne de caractères. Permet de personnaliser entièrement votre menu. Par défaut : new Walker_Nav_Menu.
  - Exemple : 'walker' => 'new mon-objet'

```
<ul>
   <li><a href='http://monlien.com'>nom de mon lien</a></li>
     <ul>
         <li>
<a href='http://monlien-niveau1.com'>nom de mon lien niveau1</a>
         </li>
     </ul>
   <li><a href='http://monlien2.com'>nom de mon lien2</a></li>
</ul>
```

La fonction wp_nav_menu()

- Référence au codex pour plus de détails sur la fonction wp_nav_menu() :
  - https://developer.wordpress.org/reference/functions/wp_nav_menu/
- La fonction wp_nav_menu() applique automatiquement des classes CSS prédéfinies.

Démonstration

La fonction wp_nav_menu()

Les hooks (filtres et actions)

- Les hooks (crochet ou hameçon) vous permettent de personnaliser votre site WordPress
- 2 types de hooks dans WordPress : les **actions** et les **filtres**.
- Les hooks sont indispensables lors de la création de fonctions dans un thème via le fichier functions.php (parfois fichier d'un dossier annexe) ou lors de la création d'extensions.
- Ils ont été créés pour ne pas modifier les fichiers sources de WordPress, qui seraient écrasés lors d'une mise à jour.
- Les hooks accrochent les fonctions au core de WordPress.
- **Les actions**
  - Les hooks d'action permettent de charger des fonctions WordPress à un moment donné ou de créer vos propres actions.
  - Par exemple, pour charger des fichiers CSS, ou pour ajouter une action lors d'un clic sur un bouton...

Les hooks (filtres et actions)

- Pour ajouter une action, utilisez la fonction :
  - **$hook** : argument obligatoire. Accepte un nom d'action (cf. la liste : http://codex.wordpress.org/Plugin_API/Action_Reference, http://adambrown.info/p/wp_hooks) ou le nom d'une action créée dans une extension ou un thème grâce à la fonction do_action().
  - **$function_name** : argument obligatoire. Accepte le nom de la fonction où se passe l'action.
  - **$priority** : argument optionnel. Sert à spécifier l'ordre dans lequel doit être exécutée l'action.
    - Par défaut : 10. Plus le chiffre est bas, plus l'action se déroule tôt. Les actions avec la même priorité s'exécutent dans l'ordre de leur apparition.
  - **$accepted_args** : argument optionnel. Sert à spécifier le nombre d'arguments passant dans la fonction $function_name.

```
<?php add_action( $hook, $function_name, $priority,
$accepted_args ); ?>
```

Les hooks (filtres et actions)

- *Exemple :*

```
<?php
   ma_fonction_change_hook(){
      // mon action
   }
   add_action('hook_existant','ma_fonction_change_hook');
?>
```

- La fonction do_action() permet d'exécuter une action que vous avez créée.

```
<?php do_action( $hook, $arg ); ?>

<?php do_action( $hook, $arg1, $arg2, ... ); ?>
```

  - **$hook** : argument obligatoire. Accepte le nom de l'action créée.
  - **$arg**, $arg1, $arg2, ... : argument optionnel. Accepte plusieurs arguments à faire passer dans la fonction créée.

Les hooks (filtres et actions)

- *Exemple :*
- Créez une fonction action :

```php
<?php
   ma_fonction_action( $arg1,$arg2 ){
      // mon action
   }
?>
```

- Ajoutez la fonction aux actions à effectuer :

```php
<?php add_action( 'mon_hook', 'ma_fonction_action', '10', '2' ); ?>
```

- Utilisez l'action :

```php
<?php do_action( 'mon_hook', $arg1, $arg2 ); ?>
```

- La fonction do_action_ref_array($hook, $args) s'utilise de la même façon, à ceci près que **$args** accepte un tableau. Certaines actions nécessitent l'usage de cette fonction.

Les filtres

- Les **hooks filtres** permettent de modifier des fonctions existantes de WordPress avant leur affichage, ou lors de l'enregistrement dans la base de données.
- Par exemple, pour modifier la fonction de WordPress the_excerpt(), affichant le résumé d'un article ou pour organiser le code HTML d'une fonction...
- Les **hooks filtres** retournent une **valeur**, contrairement à une action.
- Pour ajouter un filtre, il faut utiliser la fonction : add_filter()
  - **$hook** : argument obligatoire. Accepte un nom de filtre (cf liste : http://codex.wordpress.org/Plugin_API/Filter_Reference, http://adambrown.info/p/wp_hooks) ou le nom d'une action créée dans une extension ou un thème grâce à la fonction apply_filter().
  - **$function_name** : argument obligatoire. Accepte le nom de la fonction où se passe le filtre.
  - **$priority** : argument optionnel. Sert à spécifier l'ordre dans lequel doit être exécuté le filtre. Par défaut : 10. Plus le chiffre est bas, plus le filtre se déroule tôt. Les filtres avec la même priorité s'exécutent dans l'ordre de leur apparition.
  - **$accepted_args** : argument optionnel. Sert à spécifier le nombre d'arguments passant dans la fonction $function_name.

```
<?php add_filter( $hook, $function_name, $priority,
$accepted_args ); ?>
```

Les filtres

- *Exemple*

```php
<?php
    ma_fonction_change_hook(){
        // mon filtre
    }
    add_filter('hook_existant','ma_fonction_change_hook');
?>
```

- La fonction apply_filters() permet d'exécuter un filtre que vous avez créé.
  - **$hook** : argument obligatoire. Accepte le nom du filtre créé.
  - **$arg**, $arg1, $arg2, ... : argument optionnel. Accepte plusieurs arguments à faire passer dans la fonction créée.

```php
<?php apply_filters( $hook, $arg ); ?>

<?php apply_filters( $hook, $arg1, $arg2, ... ); ?>
```

Les filtres

- *Exemple*
- Créez une fonction filtre :

```
<?php
    ma_fonction_filtre( $arg1,$arg2 ){
       // mon filtre

    return $filtre;
    }
?>
```

Exemple

```
class MaClasse {

  function __construct() {
//constructeur
add_action( 'nom_hook', array( $this, 'mon_action' ) );
  }

  function mon_action() {
//mon action
  }

}
```

- Ajoutez la fonction aux filtres à exécuter :

```
<?php add_filter( 'mon_hook', 'ma_fonction_filtre', '10', '2' ); ?>
```

- Utilisez le filtre :

```
<?php apply_filters( 'mon_hook', $arg1, $arg2 ); ?>
```

Ajouter des filtres et des actions dans une classe

- Voici la structure pour ajouter des hooks dans vos thèmes et extensions en programmation orientée objet. Gardez la même structure pour les fonctions de suppression.

```
ma_fonction( 'nom_hook', array( $this, 'mon_hook' ) );
```

```
class MaClasse {

  function __construct() {
//constructeur
add_action( 'nom_hook', array( $this, 'mon_action' ) );
  }

  function mon_action() {
//mon action
  }

}
```

```
class MaClasse {

  function __construct() {
//constructeur
apply_filter( 'nom_hook', array( $this, 'mon_filtre' ) );
  }

  function mon_filtre() {
//mon filtre
  }

}
```

Supprimer les filtres et les actions

- Pour supprimer une action ou un filtre, utilisez ces fonctions:

```
<?php remove_action( $hook, $function_name, $priority ); ?>
<?php remove_all_actions( $hook, $priority ) ?>
```

```
<?php remove_filter( $hook, $function_name, $priority ); ?>
<?php remove_all_filters( $hook, $priority ); ?>
```

- Si un hook est activé avec une priorité différente de celle par défaut, il faut la spécifier lors de la suppression.

Tester les filtres et les actions

- Pour tester une action ou un filtre, utilisez ces fonctions conditionnelles selon les cas :
- À noter que pour ces fonctions,
  - **$hook** est un argument obligatoire,
  - tandis que $function_name est optionnel.

```
<?php has_action( $hook, $function_name ); ?>

<?php has_filter( $hook, $function_name ); ?>
```

- *Exemple*
- S'il y a le filtre mon_filtre dans mon_hook, alors le code s'exécute.

```
<?php
   if (has_filter( 'mon_hook', 'mon_filtre' ) ){
//code à exécuter
   }
?>
```

- Si mon_hook a des actions attachées à lui, alors le code s'exécute.

```
<?php if (has_action( 'mon_hook') ){ //code à exécuter } ?>
```

Tester les filtres et les actions

- Il est aussi possible de connaître le hook dans lequel vous êtes, cela marche pour les actions comme pour les filtres :
- Si l'action en cours est mon_hook, alors le code s'exécute.
- Vous pouvez également tester le nombre de fois qu'une action a été exécutée. La fonction did_action() peut renvoyer un chiffre.
- Si l'action mon_hook a été exécutée une fois, alors le code s'exécute.

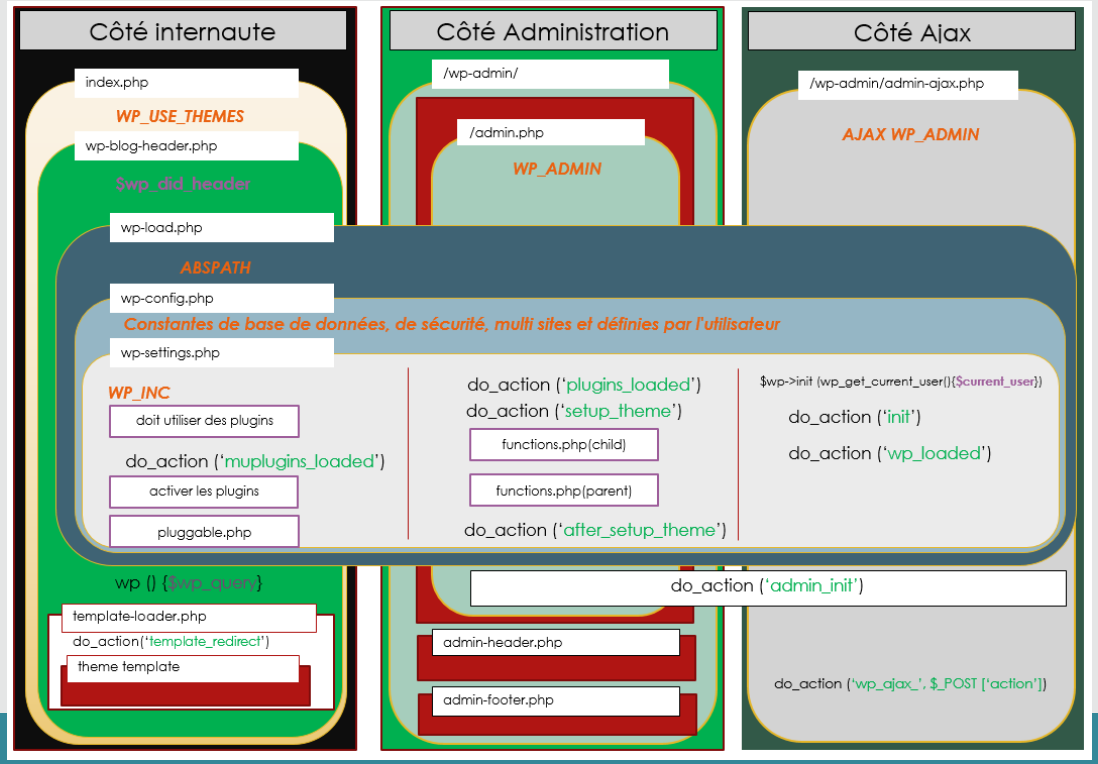
```
<?php if (current_filter()='mon_hook'){ //code à exécuter } ?>
```

```
<?php if ( did_action( 'mon_hook' )==1 ) { //code à exécuter } ?>
```

WordPress et PHP

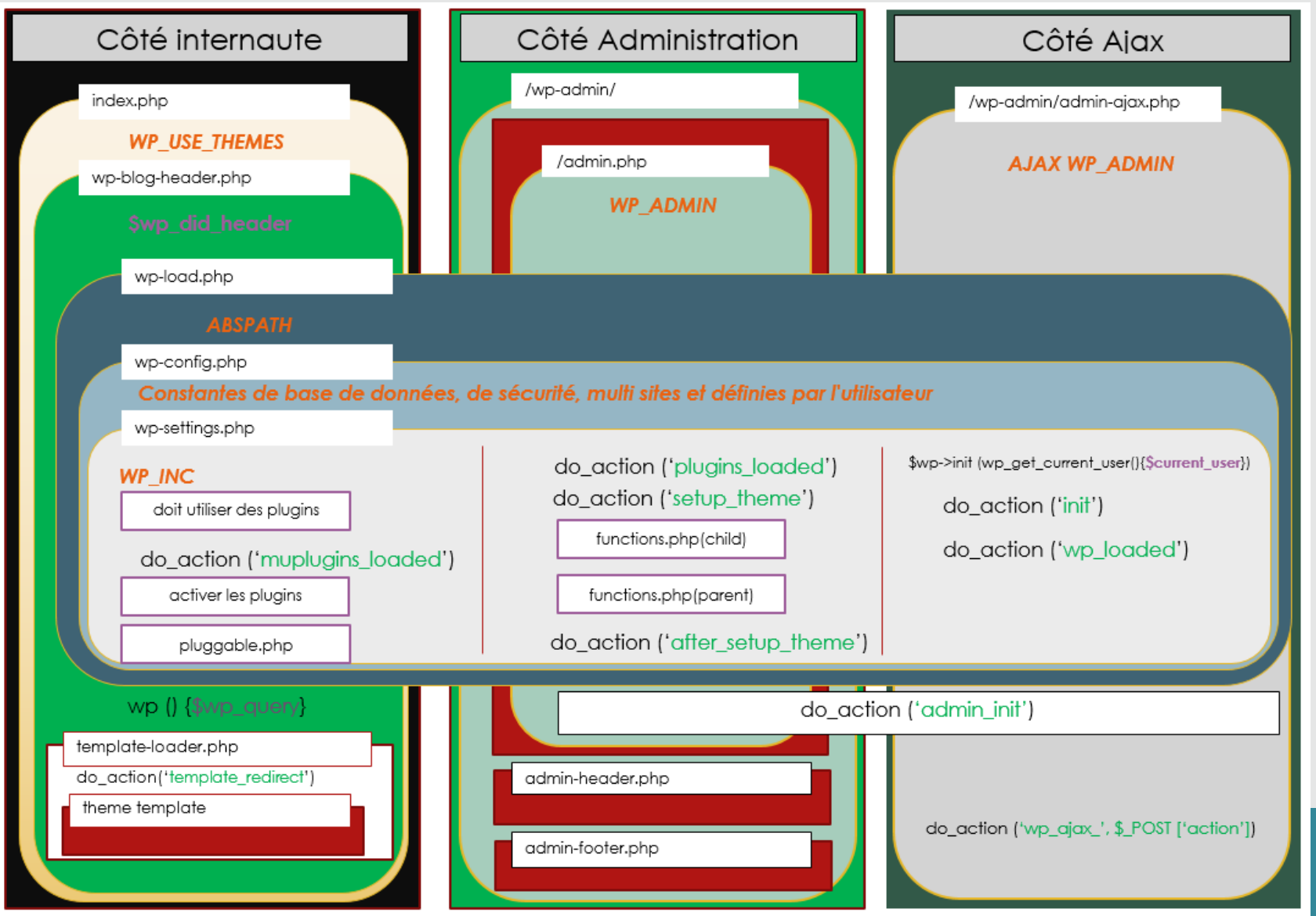
L'ordre d'exécution des actions

- Les hooks lors du chargement de WordPress s'exécutent dans un ordre précis. Vous aurez besoin de connaître cet ordre pour exécuter au bon moment vos actions.
- Pour cela il vous est nécessaire de vous référer au codex : http://codex.wordpress.org/Plugin_API/Action_Reference
- Voici l'ordre d'exécution des principales actions :



WordPress et PHP

L'ordre d'exécution des actions



L'ordre d'exécution des actions

- **Rappel** : certaines actions nécessitent l'usage de la fonction do_action_ref_array($hook, $args);.
- Les globales vous permettent de connaître la liste des actions et des filtres que votre site charge, et savoir le nombre de fois que les actions sont exécutées.
- Voici quelques actions :
  - **after_setup_theme** : s'effectue directement après le chargement du fichier **functions.php**. Cette action est utilisée pour les arguments et options du thème. C'est le premier **hook** sur les thèmes. À ce stade, le rôle de l'utilisateur n'est pas identifié.
  - **init** : s'effectue après que WordPress ait fini de charger, mais avant que les headers soient envoyés. Les rôles ont été identifiés. Cette action est utilisée généralement pour initialiser les extensions ou intercepter les variables **$_GET** et **$_POST**.
  - **widgets_init** : s'effectue après le chargement de WordPress, juste après l'action init. Cette action est utilisée pour initialiser les widgets.
  - **wp_loaded** : s'effectue après que WordPress a fini de charger toutes les extensions et le thème.
  - **wp** : s'effectue une fois que l'objet $wp est chargé. Cette action est idéale pour filtrer ou valider certaines requêtes.

L'ordre d'exécution des actions

- **after_switch_theme** : s'effectue à chaque fois qu'un thème est activé.
- **template_redirect** : s'effectue juste avant que WordPress détermine la page à charger. Cette action est utilisée pour faire une redirection.
- **wp_enqueue_scripts** : s'utilise pour ajouter des scripts JavaScript et des styles CSS à WordPress dans la partie internaute.
- **wp_head** : s'utilise pour ajouter des éléments entre les balises HTML <head></head>. Cette action permet d'ajouter des éléments à la fonction wp_header().
- **wp-footer** : s'utilise juste avant la balise HTML </body>. Cette action permet d'ajouter des éléments à la fonction wp_footer().
- **admin_init** : s'utilise avant n'importe quel hook quand vous accédez à l'administration.
- **admin_menu** : s'utilise lors du chargement du menu d'administration. Cette action s'utilise lors de l'ajout d'un onglet au menu d'administration par exemple.

L'ordre d'exécution des actions

- **load-(page)** : s'utilise pour exécuter une action seulement si la page d'une extension précise est chargée.
- **admin-head-(plugin_page)** : s'utilise pour ajouter une action entre les balises HTML <head></head> seulement si la page d'une extension précise est chargée.
- **admin_enqueue_scripts** : s'utilise pour ajouter des scripts JavaScript et des styles CSS à WordPress dans l'administration.
- **login_enqueue_scripts** : s'utilise pour ajouter des scripts JavaScript et des styles CSS à WordPress dans la page de connexion.
- **wp_ajax_(action)** : s'utilise pour créer des fonctions Ajax pour les utilisateurs connectés.
- **wp_ajax_nopriv_(action)** : s'utilise pour créer des fonctions Ajax pour les utilisateurs non connectés.

WordPress et PHP

Les globales

- Les globales sont utilisées partout dans WordPress. Elles permettent de récupérer une majorité des données de WordPress. Il vaut mieux éviter de modifier les globales directement. Il est préférable d'utiliser des fonctions spécifiques de WordPress.
- Pour récupérer les globales, déclarez-les, comme en PHP :

```
<?php global $ma_variable; ?>
```

- La globale **$post** s'utilise à l'intérieur d'une boucle et renvoie toutes les informations concernant l'article ou la page. **$post** utilise l'objet WP_Query et les requêtes sur le contenu.

```
<?php global $post; ?>
```

- Les fonctions à l'intérieur des boucles de WordPress utilisent également $post pour récupérer les informations. Cette globale est généralement chargée directement par les modèles de page, inutile de la déclarer une deuxième fois.
- **$post** permet de récupérer les informations sous forme de variables.
- La fonction **get_the_id()** = $post->id par exemple.

Les globales

- **$post->ID** : renvoie l'id de la page ou de l'article.
- **$post->ancestors** : renvoie sous forme de tableau la hiérarchie parent/enfant(s).
- **$post->post_author** : renvoie l'id de l'auteur.
- **$post->post_content** : renvoie le contenu de l'article ou de la page.
- **$post->post_date** : renvoie la date utilisée par le serveur au format: yyyy-mm-dd hh:mm:ss. Exemple : 2018-02-11 15:51:27.
- **$post->post_date_gmt** : renvoie la date GMT (*Greenwich Mean Time*) utilisée par le serveur.
- **$post->post_excerpt** : renvoie le résumé d'un article ou d'une page.
- **$post->guid** : renvoie le lien de l'article ou de la page. Ne renvoie pas le permalien.

Les globales

- **$post->post_modified** : renvoie la date de la dernière modification de l'article ou de la page, utilise la date du serveur.
- **$post->post_modified_gmt** : renvoie la date GMT de la dernière modification de l'article ou de la page, utilise la date du serveur.
- **$post->post_parent** : renvoie l'id du parent s'il y en a un.
- **$post->post_type** : renvoie le type.
- **$post->post_title** : renvoie le titre de l'article ou de la page.
- Voir la liste complète : http://codex.wordpress.org/Function_Reference/$post
- Vous pouvez aussi afficher le tableau retourné par la globale, avec la fonction PHP print_r() :

```php
<?php

global $post;
print_r($post);

?>
```

Les globales pour les serveurs

- Ces globales servent à connaître le serveur utilisé par WordPress. Elles retournent une valeur booléenne true ou false.
  - **$is_apache** : serveur Apache.
  - **$is_IIS** : Microsoft Internet Information Services (IIS).
  - **$is_iis7** : Microsoft Internet Information Services (IIS) v7.x.
- Les globales pour les versions de WordPress servent à connaître différentes informations sur les versions utilisées.
  - **$wp_version** : la version de WordPress installée.
  - **$wp_db_version** : le numéro de version de la base de données.
  - **$tinymce_version** : la version installée de TinyMCE.
  - **$required_php_version** : la version requise pour PHP.
  - **$required_mySQL_version** : la version requise pour MySQL.

Les globales diverses

- **$super_admins** : renvoie un tableau avec les id des administrateurs ayant le statut Super-admin, s'utilise uniquement pour le multisite.
- **$wp_query** : est la globale, instance de la classe wp_query renvoyant toutes les informations d'un article ou d'une page.
- **$wp_rewrite** : est la globale, instance de la classe wp_rewrite servant à la gestion des règles de réécriture, permettant d'utiliser les permaliens.
- **$wpdb** : est la globale, instance de la classe wpdb servant à faire des requêtes à la base de données.
- **$wp_roles** : est la globale, instance de la classe wp_roles renvoyant les informations concernant les rôles.
- **$wp_filter** : renvoie un tableau listant tous les filtres et les actions utilisés.
- **$wp_actions** : renvoie un tableau avec toutes les actions utilisées et le nombre de fois où elles ont été déclenchées.

Les classes de WordPress

- WordPress est construit sur le principe de la programmation orientée objet (POO).
- Il utilise de nombreuses classes qui appartiennent généralement aux API de WordPress. Les API sont un ensemble de classes, de méthodes et de fonctions qui permettent de coder plus simplement.
  - Exemple : API des widgets, API des shortcodes, API sur la base de données...
- Les classes permettent de structurer WordPress.
- De nombreuses fonctions instancient directement une classe, et permettent d'utiliser certaines méthodes à travers d'autres fonctions.
- Certaines globales permettent d'instancier directement une classe.
- Nous pouvons donc récupérer des informations de plusieurs manières :
  - par des fonctions WordPress,
  - par des globales,
  - en utilisant directement des objets.

Les classes de WordPress

- Les objets s'utilisent comme en PHP, par instance de classe :
- Voici la liste des classes qu'utilise WordPress : http://codex.wordpress.org/Class_Reference
  - La classe **WP_Roles** : permet de gérer les rôles.
  - La classe **WP_User** : permet de gérer les rôles attribués aux utilisateurs.
  - La classe **WP_Post** : récupére les informations d'un article ou d'une page.
  - La classe **WP_Query** : permet de faire des requêtes sur n'importe quelles informations concernant les articles ou les pages.
  - La classe **wpdb** : permet d'interagir avec la base de données grâce à des requêtes SQL.
  - La classe **WP_Widget** : permet de gérer et de créer des widgets.
  - La classe **WP_Rewrite** : permet de gérer les règles d'écriture concernant les permaliens.
  - La classe **WP_Object_Cache** : permet de mettre en cache le résultat des requêtes à la base de données, afin de ne pas la saturer.

```
<?php $Nom_de_l_objet = new Nom_de_la_classe; ?>
```

La classe WP_Query et les requêtes sur le contenu

```php
<?php $the_query = new WP_Query( $args ); ?>
```

```php
<?php
$the_query = new WP_Query($args);

if($the_query->have_posts()){
   while ($the_query->have_posts()){
      $the_query->the_post();
         //fonctions ex: the_title();
   }
}
?>
```

- La classe **WP_Query** permet de faire des requêtes sur la base de données et sur la table **wp_posts**, et de récupérer n'importe quel contenu concernant les articles et les pages, grâce à la méthode **the_post**().
- **WP_Query** avec la méthode **the_post**() renvoie un **tableau** avec le contenu des articles et des pages, cela permet de faire une boucle et de récupérer le contenu à l'aide de fonctions d'affichage (ex : **the_title**() pour le titre).
- De nombreuses fonctions utilisent la classe **WP_Query**.
- La boucle principale de WordPress utilise la méthode **the_post**(), ainsi que les fonctions demandant des informations sur le contenu d'un article ou d'une page, mais également les marqueurs conditionnels. L'ensemble représente une partie de l'API **WP_Query**.

La classe WP_Query et les requêtes sur le contenu

```php
<?php
$the_query = new WP_Query($args);

if($the_query->have_posts()){
   while ($the_query->have_posts()){
      $the_query->the_post();
         //fonctions ex: the_title();
   }
}
?>
```

- La boucle avec la classe **WP_Query** et la méthode **the_post**() fonctionne exactement comme la boucle de WordPress, la boucle accepte donc les mêmes fonctions d'affichage.
- Cette boucle peut s'utiliser en tant que boucle secondaire à la boucle principale.
- Attention un trop grand nombre de requêtes à la base de données peut créer des ralentissements sur le site.
- Voici le lien du codex sur les **shortcodes** : http://codex.wordpress.org/Class_Reference/WP_Query

La classe WP_Query et les requêtes sur le contenu

- Les méthodes
  - Voici les méthodes les plus fréquemment utilisées:
    - have_posts() : permet de tester avant ou dans la boucle, si le tableau renvoyé contient des articles. Elle est également présente avant la boucle WordPress et peut être considérée comme marqueur conditionnel.
    - the_post() : permet d'initialiser la globale **$post** et de récupérer tous les éléments d'articles ou de pages grâce à des fonctions.

La classe WP_Query et les requêtes sur le contenu

- Les arguments
  - La classe **WP_Query** a besoin d'au moins un argument pour s'afficher.
  - **Trier par catégories**
    - **cat** : accepte un ou plusieurs id de catégories. Permet de récupérer tous les articles appartenant à la catégorie correspondante.

```
$query = new WP_Query( 'cat=4' );
$query = new WP_Query( 'cat=2,6,17,38' );
```

    - Pour exclure des catégories, mettez un signe moins « - » devant l'id :
    - **category_name** : accepte le nom clé d'une ou de plusieurs catégories. Permet de récupérer tous les articles appartenant à la catégorie ou aux catégories correspondantes.

```
$query = new WP_Query( 'cat=-12,-34,-56' );
```

    - **category__and** : accepte un tableau contenant les id de catégories. Permet de récupérer les articles appartenant à toutes les catégories dont l'id est présent dans le tableau.
      - Dans l'exemple, les articles appartiennent à la catégorie 2 et 6.

```
$query = new WP_Query( array( 'category__and' => array(2,6)));
```

```
$query = new WP_Query( 'category_name=ma-cat' );
$query = new WP_Query( 'category_name=ma-cat,actus' );
```

La classe WP_Query et les requêtes sur le contenu

- Les arguments
  - **Trier par catégories**
    - **category__in** : accepte un tableau contenant les id de catégories. Permet de récupérer tous les articles appartenant à l'une ou l'autre catégorie.
    - Dans l'exemple, les articles appartiennent à la catégorie 2 ou 6.

```
$query = new WP_Query( array( 'category__in' => array(2,6)));
```

    - **category__not_in** : accepte un tableau contenant les id de catégories à exclure. Permet de récupérer tous les articles n'appartenant pas à ces catégories.
    - Dans l'exemple, les articles n'appartiennent pas à la catégorie 2 ou 6.

```
$query = new WP_Query( array( 'category__not_in' => array( 2, 6)));
```

La classe WP_Query et les requêtes sur le contenu

- Les arguments
  - **Trier par étiquettes**
    - **tag_id** : accepte un ou plusieurs id d'étiquettes. Permet de récupérer tous les articles liés aux étiquettes correspondant à ces id.
    - **tag** : accepte le nom clé de l'étiquette.
    - **tag__and** : accepte un tableau contenant les id d'étiquettes. Permet de récupérer tous les articles qui ont pour étiquettes l'une et l'autre des étiquettes contenus dans le tableau.
    - **tag__in** : accepte un tableau contenant les id des étiquettes. Permet de récupérer tous les articles qui ont pour étiquettes l'une ou l'autre des étiquettes contenues dans le tableau.
    - **tag__not_in** : accepte un tableau contenant les id d'étiquettes à exclure.

La classe WP_Query et les requêtes sur le contenu

- Les arguments
  - **Trier par auteurs**
    - **author** : accepte un ou plusieurs id d'auteurs. Permet de récupérer tous les articles des auteurs correspondants.
    - **author_name** : accepte le nom clé de l'auteur (si le nom est un nom composé, il a des tirets).
      - Exemple : http://www.monsite.com/author/**jean-paul**/
    - **author__in** : accepte un tableau contenant les id des auteurs.
    - **author__not_in** : accepte un tableau contenant les id des auteurs à exclure.

La classe WP_Query et les requêtes sur le contenu

- **Trier par pages et par articles**

Par défaut, le post_type est égal à post. Pour utiliser ces arguments pour les pages, il faut définir le post_type à page.

- **p** ou **page_id** : accepte un ou plusieurs id de pages ou d'articles.
- **name** : accepte le nom clé (l'identifiant) de la page ou de l'article.
  - Exemple : http://www.monsite.com/**titre-de-ma-page**/
- **post_parent** : accepte l'id de page ou de l'article parent et renvoie tous les enfants.
- **post_parent__in** : accepte un tableau contenant les id des parents et renvoie tous les enfants.
- **post_parent__not_in** : accepte un tableau contenant les id des parents à exclure.

La classe WP_Query et les requêtes sur le contenu

- **Choisir l'ordre d'affichage**

Comme dans une requête SQL classique, un ordre d'affichage (**order by**) est défini.

- **order** : accepte les valeurs **ASC** et **DESC**. Permet de récupérer un tableau dans un ordre ascendant ou descendant selon le champ souhaité.
- **orderby** : accepte comme arguments id, author, title, name, date, parent,modified, comment_count, menu_ordrer, meta_value, meta_value_num, post_in.
- Permet de choisir le ou les champs que vous souhaitez ordonner.

```
<?php
$query = new WP_Query(
                array ( 'orderby' => 'title', 'order' => 'DESC' )
        );
?>
```

La classe WP_Query et les requêtes sur le contenu

- **Plus de filtres**
  - Les arguments pour la classe **WP_Query** sont nombreux
  - Il existe aussi des filtres pour les taxonomies, les métadonnées (champs personnalisés), les dates, la pagination, le cache, les statuts des articles..., mais aussi pour des articles (post_type) personnalisés et pour les taxonomies personnalisées.
  - Vous pouvez exécuter des requêtes complexes, exactement comme en SQL.

  - Pour plus de renseignements, consultez le codex : http://codex.wordpress.org/Class_Reference/WP_Query

La classe WP_Query et les requêtes sur le contenu

- **La concaténation d'arguments**
  - Il est possible de concaténer les arguments de requête autrement qu'avec un tableau :
  - Utilisez le signe « **&** » entre les différents arguments, de la même façon que vous passez des variables dans les URL.
  - *Exemple*
    - `<?php $query = new WP_Query('cat=1&orderby=name&order=ASC'); ?>`
  - Cependant, utiliser les arguments en tableau quand ils sont nombreux est plus simple à ranger et évite les erreurs.

La classe WP_Query et les requêtes sur le contenu

- Plusieurs boucles sur la même page
  - Pour inclure plusieurs boucles dans la même page, après chaque boucle **réinitialisez l'objet WP_Query**, car la boucle a modifié la globale **$post** selon les critères de la requête. Cela évite les conflits.
  - La fonction wp_reset_postdata() est à placer juste après votre boucle.
  - <?php wp_reset_postdata(); ?>
- D'autres fonctions pour faire des requêtes
  - Les fonctions get_posts() et query_posts() permettent de faire des requêtes sur le contenu, et utilisent donc la classe **WP_Query**.
  - **$args** accepte des arguments de requête similaires.
  - Cf. codex : http://codex.wordpress.org/Template_Tags/get_posts et http://codex.wordpress.org/Function_Reference/query_posts

La classe WP_Query et les requêtes sur le contenu

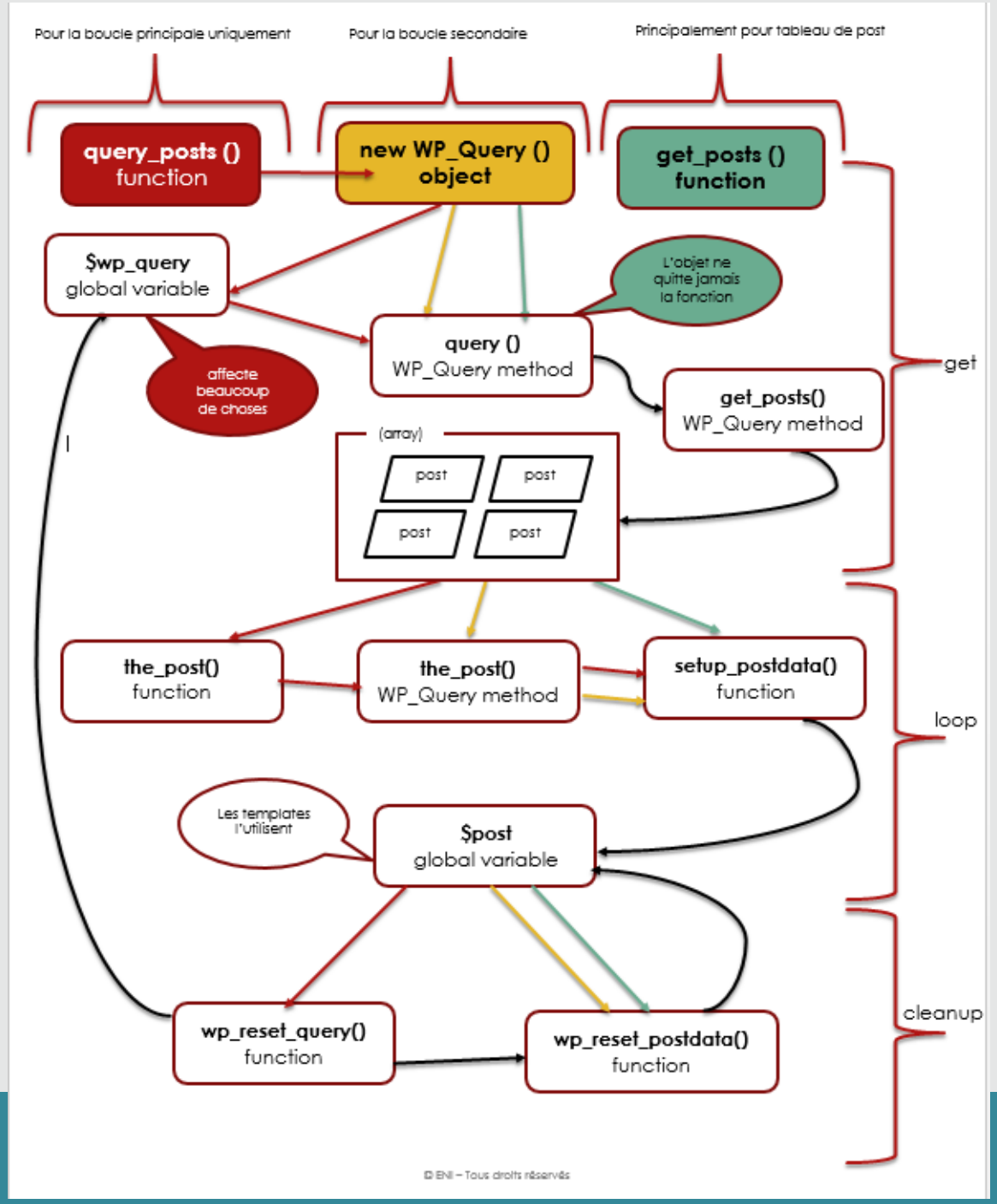
- get_posts() : permet de faire des requêtes simples.
- setup_postdata($objet_array) permet d'utiliser les fonctions d'affichage de WordPress proprement.
  - Cela charge toutes les données de l'objet.
  - Il faut déclarer la globale **$post** en **premier** lieu, car la requête doit modifier la globale $post et charger les critères de la requête.
- query_posts() : sert à modifier la requête principale de WordPress et à la remplacer. Cette fonction s'utilise uniquement en tant que boucle principale.
- La fonction wp_reset_query() sert à rétablir la requête d'origine comme le fait la fonction wp_reset_postdata() avec la globale $post.
- Il ne faut surtout pas utiliser cette fonction pour les extensions, vous devez l'utiliser avec précaution, dans des cas bien précis !

```php
<?php
global $post;
$query = get_posts( $args );

if($query){
   foreach($query as $post){
      setup_postdata($post);
      //fonctions ex: the_title();
   }
}
?>
```

```php
<?php

query_posts($args);

if ( have_posts() ) :

   while (have_posts()) : the_post();

      //fonctions ex: the_title();

   endwhile;

endif;

wp_reset_query();

?>
```

La classe WP_Query et les requêtes sur le contenu

- L'objet **WP_Query** est l'objet renvoyant toutes les informations des articles et pages. Grâce à lui vous pouvez faire des requêtes à la base de données. Beaucoup de fonctions font appel à cet objet et l'utilisent de façon similaire, afin d'obtenir les mêmes résultats.
- Le schéma suivant vous permet de mieux comprendre comment WordPress fonctionne avec cet objet :



La classe wpdb et les requêtes au format SQL

- Les méthodes de la classe **wpdb** ne s'utilisent pas directement, mais à travers la globale **$wpdb**, et permettent de communiquer avec la base de données grâce à des requêtes SQL.
  - **<?php global $wpdb; ?>**
- **$wpdb** permet d'accéder à toutes les tables de la base de données. Cela est utile pour créer vos propres tables.
  - Voici le lien du codex sur l'objet wpdb : http://codex.wordpress.org/Class_Reference/wpdb

La classe wpdb et les requêtes au format SQL

- Les méthodes utiles
  - **$wpdb->nom_de_la_table** (avec le nom de la table sans le préfixe) : permet d'obtenir le nom de la table avec le préfixe.
    - Exemple : dans le cas d'un préfixe wp_, vous avez comme nom pour la table posts : wp_posts. Pour récupérer la variable sous la forme : wp_posts, faites $wpdb->posts.
  - **$wpdb->prefix** : permet d'obtenir le préfixe des tables.
    - Exemple : dans le cas d'un préfixe wp_, $wpdb->prefix renvoie wp_.
  - **$wpdb->prepare($req,$value,...)** : permet de se protéger contre les injections SQL malveillantes. S'utilise lors d'insertion de variables dans la base de données et équivaut à utiliser la fonction PHP **mySQL_real_escape_string().**
    - **$req** : accepte une requête SQL ou une méthode de requête.
    - **$value** : accepte autant de valeurs à sécuriser. Identifiez dans la requête **$req** les champs à insérer et remplacez-les par **%s** pour une variable textuelle ou un nombre décimal, **%d** pour un nombre entier, **%f** pour un nombre approximatif.
  - Ensuite, mettez les valeurs correspondant à la suite de la requête dans l'ordre d'apparition.
    - **$wpdb->escape($var)** : équivaut à exécuter la fonction PHP addslashes() avant d'insérer la variable.
    - **$wpdb->show_errors()** et **$wpdb->hide_errors()** : permet d'afficher ou de cacher les erreurs SQL.

```php
<?php
$prefixe= $wpdb->prefix;
//echo $prefixe renvoie le nom de votre préfixe
?>
```

```php
<?php
$titre=$wpdb->escape($titre);
?>
```

Les principales méthodes de requêtes et leurs arguments

- **$wpdb->query($query)** : permet d'exécuter n'importe quelle requête SQL.
- **$query** : accepte n'importe quelle requête SQL.
  - *Exemple*

```
$wpdb->get_results($query_select, $type)
```

  - **$wpdb->get_results()** : permet d'exécuter une requête SELECT et de récupérer des données sous forme de tableau.
- **$query_select** : accepte une requête de type SELECT.
- **$type** : accepte le type de tableau que renverra la requête. Par défaut OBJECT, correspondant à un tableau d'objets, OBJECT_K un tableau associatif d'objets avec les noms des lignes comme clés, ARRAY_A, un tableau associatif avec les noms des colonnes comme clés, ARRAY_N, un tableau multidimensionnel avec des clés numériques.
- *Exemple*
  - **$wpdb->insert()** : permet d'exécuter une requête INSERT et d'insérer des données.
  - **$wpdb->replace()** : permet d'exécuter une requête REPLACE et de mettre à jour les données d'une ligne, ou d'en insérer une nouvelle.

```
$wpdb->insert($table, $data, $format)
$wpdb->replace($table, $data, $format)
```

```php
<?php
$wpdb->query(
 $wpdb->prepare(
 "
 DELETE FROM $wpdb->postmeta
 WHERE post_id =%d
 AND meta_key =%s
 ",
 13, 'actus'
 )
);
?>
```

```php
<?php
$wpdb->get_results(
 "
 SELECT *
 FROM $wpdb->posts
 WHERE post_status = 'draft'
 AND post_author = 5
 "
 );
?>
```

Les principales méthodes de requêtes et leurs arguments

- **$table** : accepte le nom de la table.
- **$data** : accepte un tableau avec pour clé(s) le ou les nom(s) de colonne(s) et comme valeurs les données à insérer.
- **$format** (optionnel) : accepte un tableau ou des variables correspondant au champ à insérer $data telles que **%s** pour une variable textuelle ou un nombre décimal, **%d** pour un nombre entier, **%f** pour un nombre approximatif. Ensuite, mettez les valeurs correspondant à la suite de la requête dans l'ordre d'apparition.
- *Exemples*

```php
<?php
$wpdb->insert(
 $wpdb->nom_de_table,
 array(
 'colonne1' => 'valeur textuelle',
 'colonne2' => 123
 ),
 array(
 '%s',
 '%d'
 )
);
?>
```

```php
<?php
$wpdb->replace(
 $wpdb->nom_de_table,
 array(
 'id' => 1,
 'colonne1' => 'valeur textuelle'
 ),
 array(
 '%d',
 '%s',
 )
)
?>
```

Les principales méthodes de requêtes et leurs arguments

- **$wpdb->update**() : permet d'exécuter une requête UPDATE et de mettre à jour des données.
- **$wpdb->delete**() : permet d'exécuter une requête DELETE et de supprimer des données.

```
$wpdb->update( $table, $data, $where, $format, $where_format)
$wpdb->delete($table, $where, $where_format)
```

- **$table** : accepte le nom de la table.
- **$data** : accepte un tableau avec pour clé(s) le ou les nom(s) de colonne(s) et comme valeurs les nouvelles données à insérer.
- **$where** : accepte un tableau avec pour clé(s) le ou les nom(s) de colonne(s) et ayant pour valeurs la référence à la ou aux valeurs de la ligne à remplacer.
- **$format (optionnel)** : accepte un tableau ou des variables correspondant au champ à insérer $data telles que **%s** pour une variable textuelle ou un nombre décimal, **%d** pour un nombre entier, **%f** pour un nombre approximatif.
- **$where_**format (optionnel) : accepte un tableau ou des variables correspondant au champ à insérer $where telles que **%s** pour une variable textuelle ou un nombre décimal, **%d** pour un nombre entier, **%f** pour un nombre approximatif.

```php
<?php
$wpdb->update(
 $wpdb->nom_de_table,
 array(
 'colonne1' => 'valeur textuelle',
 'colonne2' => 'nombre entier'
 ),
 array( 'ID' => 1 ),
 array(
 '%s',
 '%d'
 ),
 array( '%d' )
);
?>
```

```php
<?php
$wpdb->delete(
    $wpdb->nom_de_table,
    array( 'ID' => 1 ),
    array( '%d' )
);
?>
```

Créer des tables pour les plugins avec la fonction dbdelta()

- Lors de la création d'extensions, vous pouvez créer vos propres tables, afin d'y stocker des informations.
- La fonction dbdelta($sql) permet de créer et de mettre à jour des tables très facilement.
- La fonction vérifie si la table respecte la même structure de champ. Si celle-ci varie, elle modifie ou ajoute les champs nécessaires, mais n'en supprime pas.
- La fonction dbdelta($sql) ne fait pas partie de l'API de WordPress et nécessite l'ajout du fichier **wp-admin/includes/upgrade.php**, par une fonction d'inclusion.
- **$sql** nécessite une requête SQL de type **CREATE**.

Exemple

```
ma_fonction(){
   global $wpdb;

   $sql = "CREATE TABLE $wpdb->prefix.'nom_de_table' (
 id mediumint(9) NOT NULL AUTO_INCREMENT,
 time datetime DEFAULT '0000-00-00 00:00:00' NOT NULL,
 name tinytext NOT NULL,
 text text NOT NULL,
 url VARCHAR(55) DEFAULT '' NOT NULL,
 UNIQUE KEY id (id)
   );";

   require_once( ABSPATH . 'wp-admin/includes/upgrade.php' );
   dbDelta( $sql );
}
```

La classe WP_rewrite et la réécriture d'URL

- Les méthodes de la classe **WP_rewrite** ne s'utilisent pas directement, mais à travers la globale **$wp_rewrite**.
- Cela permet d'ajouter des règles d'écriture, grâce à des méthodes et fonctions spécifiques.
  - <?php global $wp_rewrite; ?>
- Il est obligatoire d'utiliser cet objet si vous utilisez les permaliens (autres que ceux par défaut) pour la réécriture d'URL (*URL rewriting*), et si vous souhaitez passer des variables dans vos URL, ou si vous utilisez une structure d'articles, une taxonomie personnalisée, une boucle personnalisée nécessitant une pagination...
- Si vous utiliser le fichier **.htaccess** pour faire passer vos règles de réécriture d'URL, il vaut mieux utiliser les différentes méthodes de l'objet **WP_rewrite**.
- WordPress utilise ce système pour éviter les conflits avec les différentes extensions ou thèmes, et la réécriture systématique du fichier **.htaccess**. Pour éviter les conflits, les règles d'écriture sont enregistrées en base de données, dans la **table d'options**.
- En local, pensez à activer le module **rewrite_module** d'Apache.
- Voici le lien du codex sur l'objet WP_rewrite : http://codex.wordpress.org/Class_Reference/WP_Rewrite

La classe WP_rewrite et la réécriture d'URL

```php
<?php

function custom_rewrite_tag() {
   add_rewrite_tag('%film_title%', '([^&]+)');
}
add_action('init', 'custom_rewrite_tag', 10, 0);

?>
```

```php
<?php
global $wp_query;
$film_title=$wp_query->query_vars['film_title'];
?>
```

- Passer une variable dans une URL avec la fonction add_rewrite_tag()
- Pour faire passer une variable dans une URL, il faut utiliser 2 fonctions WordPress indissociables : la fonction add_rewrite_tag() qui permet de créer la variable et la fonction add_rewrite_rule()pour réécrire l'URL en ajoutant une règle.
  - <?php add_rewrite_tag($tag, $regex); ?>
  - **$tag** : accepte le nom clé de la variable que vous utilisez, entouré du caractère **%**. Cela est comparable à celui que vous trouvez dans les permaliens **%postname**%.
- Voici la liste des marqueurs par défaut : http://codex.wordpress.org/Using_Permalinks#Structure_Tags
- Cette fonction utilise la méthode $wp_rewrite->add_rewrite_tag(). La fonction add_rewrite_tag() doit être utilisée à l'intérieur d'une fonction ou d'une méthode appelée par un hook d'action : init.
- Pour récupérer la valeur de la variable **$tag**, il faut utiliser l'objet **WP_Query**.
  - $wp_query->query_vars[$tag]
- Remarque : l'utilisation de $_GET sur une URL réécrite ne fonctionne pas, même si les variables sont présentes.
- *Exemple*
  - Pour faire passer dans une URL une variable qui a comme nom clé film_title : Cf code 1
  - Pour récupérer la variable dans les modèles de page : Cf code 2

La classe WP_rewrite et la réécriture d'URL

```
<?php add_rewrite_rule($regex, $redirect, $after); ?>
```

- La fonction WordPress add_rewrite_rule() permet d'ajouter une règle d'écriture aux URL.
  - **$regex** : accepte une expression régulière (regex) équivalente à l'URL $redirect.
  - **$redirect** : accepte l'URL à réécrire. Peut utiliser le tableau $matches[] contenant les données récupérées par l'expression régulière $regex.
  - **$after** : accepte 'top' ou 'bottom' afin de spécifier si la règle doit être avant toutes les autres règles ou si elle doit être à la suite. À utiliser dans de rares cas.

```
function my_rewrite_rules(){
    global $wp_rewrite;
    add_rewrite_rule('^film/([^/]*)/?',
        'index.php?page_id=12&film_title=$matches[1]','top');
}
add_action('init', 'my_rewrite_rules');
```

- La fonction add_rewrite_rule() doit être utilisée à l'intérieur d'une fonction ou d'une méthode appelée par un hook d'action init.
  - Cette règle réécrit l'URL : www.monsite.com/index.php?page_id=44&film_title=titre-de-mon-film en : www.monsite.com/film/titre-de-mon-film

La classe WP_rewrite et la réécriture d'URL

```
$wp_rewrite->flush_rules();
```

- Régénérer les règles d'écriture
  - flush_rules() : permet d'actualiser les règles d'écriture se situant dans la base de données.
  - Ainsi la fonction de l'exemple précédent devient :

```
function my_rewrite_rules(){
   global $wp_rewrite;
   add_rewrite_rule('^film/([^/]*)/?',
       'index.php?page_id=12&film_title=$matches[1]','top');
   $wp_rewrite->flush_rules();
}

add_action('init', 'my_rewrite_rules');
```

- flush_rewrite_rules() = retirer les règles d'écriture et les régénérer entièrement
  - Cette fonction est utilisée, en général, avec une structure d'articles ou une taxonomie personnalisée.

Les shortcodes

- Le shortcode est un code simplifié entre crochets, propre à WordPress, de la forme : **[monshortcode]**
- Il permet d'afficher le résultat d'une fonction réalisée en PHP. Il s'insère dans les pages, les articles...
- Il peut s'insérer automatiquement dans vos pages ou dans vos articles. Par exemple lorsque vous créez une galerie, le shortcode s'insère dans votre éditeur WYSIWYG, et sur la page vous voyez votre galerie photo.
  - *Exemple :* [gallery]
- Le shortcode permet d'afficher la galerie photo.
- Le shortcode peut aussi accepter des arguments ou récupérer du contenu.
  - *Exemple :* [gallery columns='2' size='large']
  - La galerie est alors sur deux colonnes dans une grande taille.
- [tag] mon contenu [/tag]
  - Il est possible de récupérer le contenu entre les deux shortcodes.
  - Certaines extensions nécessitent l'utilisation de ces arguments pour fonctionner, cf. documentation.
  - Le shortcode est pratique pour les personnes qui ne maitrisent pas le code.
  - Les shortcodes sont utiles lors de la création d'extensions afin de simplifier les tâches de celle-ci.

Les shortcodes

- Voici la liste des shortcodes de WordPress par défaut :
  - **[audio]** : permet d'ajouter une musique.
  - **[caption]** : permet d'ajouter une description.
  - **[embed]** : permet d'intégrer du contenu nécessitant la balise HTML embed.
  - **[gallery]** : permet de créer une galerie photo.
  - **[video]** : permet d'ajouter une vidéo.
  - Voici le lien du codex sur les shortcodes : http://codex.wordpress.org/Shortcode_API

Créer des shortcodes

- Pour créer des shortcodes, utilisez la fonction :
  - <?php add_shortcode('$nom_du_shortcode','$nom_de_ma_fonction'); ?>
    - $nom_du_shortcode : accepte le nom du shortcode.
    - $nom_de_ma_fonction : accepte le nom de la fonction.
  - *Exemple*
  - Pour créer un shortcode du type : [monshortcode] qui renvoie la phrase : « je suis un shortcode ! », créez le code suivant :

```
<?php
function exemple_shortcode(){
    return "je suis un shortcode! ";
}
add_shortcode( 'monshortcode', 'exemple_shortcode' );
?>
```

  - Placez le shortcode **[monshortcode]** dans vos pages ou articles, pour que la phrase apparaisse.

Ajouter des arguments aux shortcodes

```
<?php shortcode_atts($tab, $atts, $nom_du_shortcode); ?>
```

- Pour faire passer des arguments, sous forme de variables, et les récupérer dans votre fonction, utilisez la fonction : shortcode_atts()
  - **$tab** : accepte un tableau avec comme clés les noms des champs et comme valeurs les valeurs par défaut.
  - **$atts** : accepte le nom de variable passé dans la fonction.
  - **$nom_du_shortcode** : accepte le nom du shortcode (optionnel).
- La fonction PHP extract() permet de retourner le nom des champs sous forme de variables.
- *Exemple*
  - Une fonction :

```
<?php
function exemple_shortcode( $atts ) {
   extract(
 shortcode_atts( array(
 'valeur1' => 'valeur1_defaut',
 'valeur2' => 'valeur2_defaut'
 ),
 $atts,
 'monshortcode'
 );
   );

 return $value1.' '.$value2;
}
add_shortcode( 'monshortcode', 'exemple_shortcode' );
?>
```

Le shortcode :

```
[ monshortcode valeur1='aaa' valeur2='bbb' ]
```

Le résultat est : aaa bbb

Récupérer du contenu

- Pour récupérer du contenu entre deux shortcodes, passez une autre variable dans votre fonction.
- *Exemple*
  - Une fonction :

```
function exemple_shortcode( $atts, $content="" ) {
 return "content = $content";
}
add_shortcode( 'monshortcode', 'exemple_shortcode' );
```

  - Le shortcode :

```
[monshortcode]mon contenu[/monshortcode]
```

  - Le résultat est : mon contenu.

Récupérer du contenu

- Utiliser les shortcodes dans des fichiers PHP
  - **Il ne faut pas utiliser un shortcode directement dans un fichier PHP, car celui-ci est reconnu en tant que texte et affiche une erreur.**
  - Pour utiliser un shortcode dans un fichier PHP, utilisez la fonction do_shortcode().

```
<?php
if( shortcode_exists(monshortcode)){
    echo do_shortcode('[monshortcode]');
}
?>
```

  - Voici le lien du codex : https://developer.wordpress.org/reference/functions/do_shortcode
  - Les shortcodes ne s'utilisent pas non plus dans les widgets texte, même si ce type de widget accepte du code HTML. Il faut utiliser une extension de type widget qui accepte les **shortcodes** ou une extension de type widget qui lit le code PHP, auquel cas, il faut utiliser la fonction do_shortcode().

Démonstration