PHP PANDAS

LE LANGAGE DE PROGRAMMATION PHP POUR TOUS ET TOUTES

DAYLE REES & NICOLAS DUPUY

PHP Pandas (FR)

Le langage de programmation PHP pour tous et toutes.

Dayle Rees and Nicolas Dupuy

This book is for sale at http://leanpub.com/php-pandas-fr

This version was published on 2015-10-07

Table des matières

Remerciements

Je voudrais tout d'abord remercier ma copine Emma, non seulement pour avoir supporté mes plaisanteries de nerd mais aussi pour avoir réalisé les formidables clichés de panda roux de cet ouvrage (et des autres) ! Je t'aime, Emma !

Merci également à mes parents, qui ont soutenu mon intérêt pour ces grilles de maths de mon enfance depuis trente ans ! Merci aussi à eux d'avoir acheté environ un milliard de copies de mon premier bouquin pour les offrir à toute la famille !

Je souhaitais remercier aussi tou(te)s mes fantastiques collègues à JustPark pour leurs encouragements continus ! Vous êtes les meilleur(e)s !

Enfin, merci à tous ceux et toutes celles qui ont acheté mes précédents ouvrages, *Code Happy* et *Code Bright*, et à toute la communauté Laravel. Sans votre soutien, je n'aurais jamais eu assez de confiance pour continuer à écrire.

Errata

Ça a beau être mon troisième livre et j'ai sans doute un peu progressé en rédaction mais je peux vous assurer que vous trouverez beaucoup, beaucoup d'erreurs dans le présent ouvrage.

Vous pouvez donner un coup de main et soutenir ce livre en m'envoyant par email toute erreur que vous y trouveriez à me@daylerees.com[1] en précisant le titre de section concernée.

Les erreurs seront corrigées au fur et à mesure de leur identification. Les corrections seront publiées dans le cadre des mises à jour ultérieures du livre.

[1] mailto:me@daylerees.com

Retours

De la même manière, vous pouvez me faire part de toute remarque personnelle concernant le contenu de ce livre en m'envoyant un email à me@daylerees.com[2] ou en tweetant à @daylerees.

Je m'efforcerai de répondre à tous les mails qui me sont envoyés.

[2] mailto:me@daylerees.com

Traductions

Si vous souhaitez traduire PHPandas dans votre langage, merci de m'envoyer un email à me@daylerees.com[3] avec les raisons qui vous motivent. J'offre une répartition des profits de la version traduite sur une base de 50/50, la copie traduite étant vendue au même prix que la copie anglaise d'origine.

Merci de noter que cet ouvrage a été rédigé en format markdown.

[3] mailto:me@daylerees.com

1. Introduction

Hé, salut ! Regardez-moi ça, le plus beau ET/OU le plus magnifique lecteur de la planète ! (Valable pour les lectrices aussi). Bravo à vous d'avoir acheté PHPandas et d'avoir fait ce tout premier pas vers une carrière de développeur/développeuse de renommée internationale.

Qui suis-je ? Et bien, c'est une question toute simple ! Je m'appelle Dayle et je suis votre auteur pour cette aventure. J'écris des livres pour débutants depuis quelques années maintenant et j'ai eu ainsi l'occasion d'embarquer d'autres lecteurs et lectrices, charmant(e)s comme vous, dans des aventures leur permettant d'acquérir de nouvelles connaissances. Nous allons ainsi faire plein de découvertes ensemble et, tout au long du voyage, je resterai, soyez-en sûrs, à vos côtés.

> Pourquoi écris-tu parfois comme si tu avais une case en moins ?

Pardon ? Ah, ça… Et bien, voyez-vous, c'est la seule façon d'écrire que je connaisse. Donc si vous cherchez un livre technique pétri d'austérité scientifique professorale (Est-ce que ça se dit ? Je l'espère), et bien je crains que vous ne soyez mal tombé. J'écris mes livres pour les gens. J'aime bien m'imaginer que nous sommes juste des potes assis au pub, en train de discuter de PHP devant une pinte de Special Bre… Fosters.

Pour être tout à fait franc, les débutants pour lesquels j'écris ont plutôt tendance à apprécier mon style d'écriture. Ils ne cherchent pas à obtenir de ce bouquin un diplôme en maths, ils veulent juste apprendre deux ou trois choses sur PHP, et ça, je peux m'y engager !

Et, tenez, tant qu'on y est, vous aurez sûrement remarqué que, là, maintenant, nous sommes en train de *discuter*. Pas très courant chez les autres auteurs, non ? C'est que, en fait, j'ai ce pouvoir magique qui vous donne la possibilité de me parler et de me poser vos questions *directement*.

> Un instant, comment faites-vous pou…

Secret professionnel. Navré, nous ne pouvons pas encore partager ce genre de choses mais ça ne vous met pas en joie de pouvoir faire *vraiment* partie de cette aventure et de ne pas être simple spectateur ?

> Oui, j'imagine… OK, je veux bien tenter le coup…

Super.

Bon, maintenant, dans n'importe quel autre livre, ç'aurait été le moment parfait pour tout vous dire sur PHP, son histoire, ses applications, son auteur et un million d'autres choses. Je pense qu'il est maintenant bien identifié que je ne suis pas le plus traditionnel des auteurs et que ce genre de chapitres n'est pas mon truc. Vous avez acheté ce livre pour apprendre PHP donc je pars du principe que vous avez déjà développé par vous-même un peu de curiosité pour ce langage. Voici donc tout ce dont, selon moi, vous aurez besoin.

PHP est un langage de programmation qui fait fonctionner l'essentiel des sites présents sur ce bon vieux Web. Il a été rédigé à l'origine par un gars dont le nom est Rasmus Lerdorf et que vous pouvez voir invariablement souriant dans à peu près toutes les images de lui disponibles sur Google. Soyons clairs, Rasmus est un gars génial et, à ma façon, je le remercie chaque jour pour ce langage qui m'a donné un métier, mais je pense que c'est vraiment tout ce qu'il vous faut savoir sur lui. D'autres livres en profiteraient probablement pour vous indiquer sa marque de céréales préférée mais, plutôt que de faire ça, pourquoi ne pas plonger directement dans l'apprentissage de PHP ?

Ce livre est donc à destination de débutants ***absolus***. En d'autres termes, si vous n'avez jamais essayé de programmer avant, c'est votre jour de chance, mon ami(e) ! Si vous avez déjà touché un peu à la chose, ce sera également parfait. Enfin, si vous êtes un expert PHP, alors ce sera sans doute l'occasion de rafraîchir vos connaissances et, qui sait, de glaner deux ou trois astuces en chemin.

J'ai fait appel à ma copine, mes collègues non-technicien(ne)s et quelques personnes au hasard dans la rue et je leur ai soumis mon livre comme cobayes pour voir un peu comment il était reçu par des personnes sans aucune connaissance préalable de PHP. Mes cobayes ont transformé l'essai avec brio donc maintenant, c'est à votre tour, squik squik !

Mon objectif est que ce livre devienne le plus agréable, le plus factuel et le plus impressionnant des livres consacrés à PHP sur le marché. Je veux qu'il soit **le** livre qu'on recommende à toute personne qui souhaite devenir développeur ou développeuse PHP. J'ai travaillé très dur pour le rendre accessible à tous et à toutes donc si l'aventure vous plaît, merci de le tweeter, de le bloguer, d'acheter des copies pour vos ami(e)s et votre famille, ou encore de l'imprimer et de vous en servir pour baffer les gens que vous croisez dans la rue.

Ce livre est un livre de syntaxe PHP. Il ne vous apprendra donc pas à faire des sites (ce sera l'objet d'un autre livre de cette série sur lequel je travaille en ce moment). En revanche, il vous déploie la toute première étape de votre apprentissage des fondamentaux du langage, de telle sorte que lorsque vous en viendrez à construire votre premier site Web, vous serez @%£^ de prêts, baby !

Si, à la lecture, vous trouvez qu'une information fait défaut, qu'un chapitre n'est

pas très clair ou si quoi que ce soit d'autre vous chiffonne à son propos, alors n'hésitez pas et envoyez-moi un email à `me@daylerees.com` pour me le faire savoir ! Je suis incroyablement réactif, pour ne pas dire responsive (grâce à toutes mes *media queries*… ah ah… Blague de programmeur) et je veux vraiment que ce livre soit parfait pour tous et toutes.

Si vous lisez ce livre et que vous ne trouvez rien à redire, et bien… envoyez-moi un mail pour me dire combien vous l'avez apprécié ! Je serais très heureux d'avoir vos retours.

Et maintenant, ne perdons pas plus de temps. Vous avez plein de choses à apprendre. Tournez la page, imaginez le thème de Jurassic Park au moment où les portes s'ouvrent et préparez-vous à entrer dans le monde du développement !

2. Installation

Avant de commencer à travailler en PHP, il faut l'installer. Car, voyez-vous, PHP est une application comme les autres. Il doit être installé sur notre système de manière à pouvoir traiter du code… PHP.

Les méthodes d'installation peuvent varier sensiblement selon le système d'exploitation utilisé. C'est pour cette raison que je vous fournis ci-dessous trois guides d'installation de PHP complètement différents. La première section vous explique comment installer PHP dans un environnement Linux, en l'occurrence Ubuntu, en raison de sa popularité. La deuxième section vous explique cette fois-ci comment installer PHP dans un système Mac OSX d'Apple. Enfin, la troisième section vous explique comment installer PHP dans un système d'exploitation Windows.

Nous n'installerons que la version console de PHP. Nous n'installerons pas encore de serveur web (ce sera l'objet d'un prochain livre). La version console de PHP est tout ce dont nous avons besoin pour commencer notre processus d'apprentissage.

Souvenez-vous, ne lisez que la section correspondant à votre propre ordinateur. Une fois PHP installé, allez directement au chapitre suivant.

Linux

La meilleure manière d'installer PHP dans une distribution basée sur du Linux est de recourir à un gestionnaire de paquets (*package manager*, en anglais). Le gestionnaire de paquets disponible dépend en grande partie de la distribution Linux que vous avez retenue. J'ai choisi pour ma part de vous donner les instructions d'installation de PHP sur Ubuntu qui est une des distributions Linux les plus populaires.

Ubuntu utilise le gestionnaire de paquets `apt` pour installer ses paquets. Pour installer la version console de PHP, il faut que nous installions le paquet `php5-cli`. Allons-y. Ouvrez tout d'abord un nouveau terminal. Saisissez ensuite les instructions suivantes.

```
1 $ sudo apt-get install php5-cli
```

Vous n'avez pas besoin de saisir le signe dollar, qui n'est que l'invite de commande du terminal vous indiquant que vous saisissez votre contenu dans la console. Une fois que vous avez tapé sur la touche Entrée, apt récupère le paquet de l'application PHP et l'installe pour vous.

Et c'est tout ! Vous avez terminé. Enfin, normalement. Vérifions quand même, vous voulez bien ? Saisissez simplement…

```
$ php -v
```

Cette commande est utilisée pour afficher la version PHP actuellement installée. Vous devriez donc voir quelque chose de semblable à ceci.

```
PHP 5.5.13 (cli) (built: Jun  5 2014 19:13:23)
Copyright (c) 1997-2014 The PHP Group
Zend Engine v2.5.0, Copyright (c) 1998-2014 Zend Technologies
```

Votre version ne sera probablement pas tout à fait la même mais, après tout, nous sommes tous différents, non ? Dans l'exemple ci-dessus, la version PHP est `5.5.13`. Avec un peu de chance, votre numéro de version de PHP sera `5.4.0` ou même supérieur.

Si votre version n'est pas la bonne, alors il vous faudra consulter la documentation correspondant à votre distribution Linux pour connaître la méthode d'installation de la version correspondante.

Si tout est opérationnel, n'hésitez pas, sautez au chapitre suivant, vous avez fini !

Mac OSX

Dans le système d'exploitation Macintosh, PHP est pré-installé. Tenez, ouvrez l'application Terminal et saisissez ceci pour connaître la version de PHP que vous utilisez.

```
$ php -v
```

Ne saisissez pas le signe dollar, c'est simplement l'invite de commande du terminal ! Vous devriez voir s'afficher quelque chose comme ça, mais un peu différent.

```
PHP 5.4.24 (cli) (built: Jan 19 2014 21:32:15)
Copyright (c) 1997-2013 The PHP Group
Zend Engine v2.4.0, Copyright (c) 1998-2013 Zend Technologies
```

La version PHP dans l'exemple ci-dessus est `5.4.24`. Du moment que votre version de PHP est supérieur à `5.4`, tout va bien et vous pouvez vous rendre directement au chapitre suivant.

Si ce n'est pas le cas, pas de panique. Nous pouvons recourir à un gestionnaire de paquets tiers pour OSX qui se chargera de nous installer une version plus récente de PHP.

Nous allons en l'occurrence utiliser un gestionnaire de paquets appelé 'Homebrew' ou, plus simplement, 'Brew'. Pour installer Homebrew, suivez les instructions présentes sur le site suivant :

brew.sh[1]

Je préfère ne pas recopier ici les instructions car elles changent d'une sortie de version à l'autre. Une fois Homebrew installé, il est temps d'installer une version plus récente de PHP. Je vous recommande d'installer la version `5.5`. Vous pouvez le faire très simplement à l'aide de la commande suivante.

```
$ brew install php55
```

Vous devez ensuite préciser l'emplacement de cette version de PHP dans la variable PATH de votre système. Ne vous tracassez pas trop et tapez simplement ceci.

```
$ PATH=~/usr/local/Cellar/php55/5.5.13/bin:$PATH
```

Vous aurez peut-être à mettre à jour le numéro de version pour qu'il coïncide avec la version de PHP que Homebrew a installé sur votre système. Essayons maintenant d'afficher une nouvelle fois notre version de PHP utilisée.

```
$ php -v
```

Croisons les doigts et vous aurez normalement une version de PHP supérieure à `5.4`. Si c'est bien le cas, allez directement au chapitre suivant.

[1] http://brew.sh/

Windows

L'installation de PHP dans Windows est un tout petit peu plus compliquée, en tout cas pour moi. J'ai pu tester les instructions ci-dessous sur ma machine Windows 10, mais si vous rencontrez des difficultés à reproduire ces étapes, faites-le moi savoir et je chercherai quelqu'un de plus fûté en Windows pour mettre à jour cette section.

Pour commencer, allez sur :

http ://windows.php.net/download

Téléchargez-y l'archive zip de la dernière version PHP 5.4 et supérieure. Une fois l'archive téléchargée, dézippez-là dans un dossier qui fasse sens. J'ai par exemple choisi de l'extraire ici :

```
C:\Users\Dayle\PHP
```

Il va vous falloir une invite de commande pour pouvoir exécuter les scripts présents dans ce livre. Voici donc un bon moyen de lancer une invite de commande sous Windows.

Faites un clic-droit sur votre Bureau, ou sur n'importe quel dossier de votre choix et choisissez 'Créez un raccourci'. Dans le cartouche de saisie qui s'affiche, tapez :

```
cmd.exe
```

Cliquez sur Suivant et nommez votre raccourci "PHP".

Pour finir, faites à nouveau un clic-droit, cette fois-ci sur votre raccourci, et cliquez sur 'Propriétés'. Dans l'onglet 'Raccourci', modifiez le champ 'Démarrer dans :' pour qu'il corresponde à l'endroit où vous avez extrait l'archive PHP. Cliquez sur 'OK' quand c'est fait.

Double-cliquez sur votre raccourci PHP et vous devriez être accueilli par une invite de commande. Saisissez…

```
php -v
```

… et vous devriez voir s'afficher les informations de votre version PHP. Assurez-vous bien que la version en question est supérieure ou égale à PHP 5.4 et passez au chapitre suivant.

À nouveau : désolé pour la qualité un peu brute de décoffrage de ce sous-chapitre. Je n'utilise plus Windows comme machine de développement depuis un bon nombre d'années maintenant. Si quelqu'un connaît un meilleur moyen de lancer PHP sous Windows, soyez gentil de m'envoyer les instructions par mail et vous recevrez en retour vos 5 minutes de gloire au sein de ce chapitre !

3. Trouver les réponses

Je sais. Un peu gnangnan comme titre, non ? Mais il faut me faire confiance : ce que je vais vous dire est extrêmement important. Ce chapitre traite en effet de vous, développeur / développeuse plein(e) d'avenir, et de votre confiance en vous-même. Apprendre est toujours difficile - mais ne vous inquiétez pas, je suis là pour vous aider.

Les développeurs sont des robots.

Qu'est-ce qui vous a poussé à (vouloir) devenir développeur ? Attendez ! Laissez-moi deviner… Un soir, vous avez vu un développeur PHP du genre rockstar s'extraire nonchalamment d'une limousine pour s'engouffrer dans un des spots de nuit les plus chauds de New York, commander 5 bouteilles de Cristal et passer la soirée à glander avec Jay-Z et le fantôme de Tupac.

C'est vrai que la vie d'un développeur est *glamour*. Tenez, moi, par exemple, je rédige les chapitres de ce livre pendant les 5 seules heures de sobriété de ma journée.

Bon, alors, reprenons. Vous avez probablement vu un développeur écrire du code et vous avez pensé…

> Bon dieu, ce dèv doit être un robot. Les gars de son genre connaissent tous ces mots de code et ces fonctions et comment tout ça fonctionne.

Souvent, les personnes sans aucune connaissance du développement pensent que les développeurs sont des espèces de génies avec des diplômes honorifiques en mathématiques. C'est sans doute vrai pour quelques développeurs mais en tout cas, ce n'est pas vrai me concernant. J'aime à penser que d'autres développeurs sont dans mon cas.

La vérité, c'est que nous ne sommes pas parfaits. Nous en sommes même très loin. Si vous pensez que les développeurs PHP connaissent toutes ces fonctions et ces bouts de code par cœur, vous faites erreur - et, en plus, vous vous persuadez à tort que vous ne serez jamais capable de suivre leur rythme.

C'est tout simplement faux. Nous ne mémorisons pas tout. En fait, l'essentiel du code que nous utilisons chaque jour est tiré de sources extérieures. Nous sommes des guerriers Google. Il existe par exemple des fonctions en PHP qui agissent de la

manière la plus simple qui soit sur des lignes de texte - et pourtant je dois consulter quasiment chaque semaine la documentation PHP pour me rappeler l'ordre dans lequel il faut leur passer leurs paramètres.

Quand je suis complètement coincé, je tente ma chance sur Google pour voir si un autre développeur est tombé sur le même casse-tête. Le plus souvent, je trouve une solution adaptée qu'un autre développeur a découverte avant moi, ou alors suffisamment d'informations pour me donner la direction d'une solution possible. Bien sûr, ceci marche dans les deux sens et, pour ma part, je tente de donner mes propres solutions à la communauté. Je poste mes réponses sur Stack Overflow et je contribue à des forums et à des discussions le plus souvent possible. C'est très important de donner en retour à la communauté.

Je vous le disais, nous ne sommes pas des robots. Nous ne connaissons pas tout de notre langage de programmation et nous n'avons pas la solution à tous les problèmes. Cependant, nous sommes de fantastiques enquêteurs. Nous sommes des opportunistes. Nous sommes des résolveurs de problèmes, aux ressources infinies ou presque. Nous sommes des développeurs.

L'art de Googler

Quand on vous dit de "googler" quelque chose, c'est assez facile de le prendre comme une insulte. Ou, au mieux, du sarcasme… En fait, au moins pour nous autres développeurs, ce n'est pas le but. Si Google est notre page d'accueil, c'est bien pour une (bonne) raison. Tenez, regardons un peu comment on peut y trouver les réponses aux problèmes de développement les plus fréquents.

Mettons que nous soyons en train d'écrire un programme et que nous ayons besoin d'inverser une phrase, de telle sorte que 'Vive les pandas !' deviennent ' ! sadnap sel eviV'. Nous ne savons pas du tout par où commencer pour résoudre ce problème. Nous venons tout juste de commencer le PHP.

Nous savons simplement que, en PHP, une séquence de texte est appelée 'chaîne de caractères' ('string' en anglais). Nous le savons parce que nous n'avons pas baissé les bras et que nous avons terminé ce livre un peu barré avec ses exemples de pandas et, justement, nous avons découvert les chaînes de caractères dans un de ses chapitres (plus loin). On est OK jusqu'ici ?

Donc nous savons ce que nous cherchons à faire. Nous voulons inverser une chaîne de caractères. Faisons-en donc une requête de recherche pour Google.

```
1 inverser chaîne de caractères
```

Nan, attendez ! Le souci avec cette recherche c'est qu'il existe des milliers de langages de programmation potentiellement concernés. Juré, ça fait un bout de temps qu'il y a des ordinateurs !

Donc si nous cherchons ‘inverser chaîne de caractères’, nous allons obtenir des réponses touchant à C++, ASP.NET, Erlang, ce que vous voulez. Notre affaire ici, c’est le PHP. On n’a pas à se préoccuper des autres langages. On prendra le temps de jouer avec eux quand on sera devenus de vrais cerveaux PHP. En attendant, corrigeons notre recherche en précisant le langage concerné.

```
1 php inverser chaîne de caractères
```

Parfait. Jetons un œil aux résultats que nous propose Google. C’est d’ailleurs sans doute le meilleur moment pour vous annoncer que je ne travaille pas pour Google et que je ne touche aucune commission. N’hésitez pas à utiliser Bing si vous préférez, mais vous vous retrouverez peut-être au final à acheter une remorque à cheval d’occasion plutôt qu’à trouver une fonction inversant une chaîne de caractères… Bref, alors, ces résultats ?

Inverser chaîne de caractères - PHP

http ://php.net/manual/fr/function.strrev.php[1]

Vous vous débrouillez en anglais ? Tentez votre chance dans cette langue, vous aurez encore plus de résultats, comme par exemple :

Reverse a string with php - Stack Overflow

http ://stackoverflow.com/questions/11100634/reverse-a-string-with-php[2]

En posant la bonne question, nous recevons ainsi des ressources concrètes et utiles en retour. Le manuel PHP (parfois appelé les docs de l’API PHP) et Stack Overflow (en anglais) sont deux des ressources Internet les plus efficaces pour résoudre des problèmes PHP. Attention, je ne dis pas qu’elles proposent toujours la bonne réponse. Qui plus est, il existe d’autres sites très utiles mais je suis sûr que vous vous rendrez rapidement compte, à l’usage, que vos recherches vous ramènent souvent sur ces deux sites.

Pour l’heure, nous cherchons donc une espèce d’outil pour réaliser une inversion de chaîne de caractères. Nous ne cherchons donc pas à résoudre un problème abstrait, nous savons précisément ce que nous voulons.

[1] http://php.net/manual/fr/function.strrev.php

[2] http://stackoverflow.com/questions/11100634/reverse-a-string-with-php

Continuez et cliquez sur ce premier lien. La ravissante page du manuel PHP dédiée à une fonction appelée `strrev()` nous accueille. Pas besoin de savoir ce qu'est une fonction pour l'instant. Ne vous inquiétez pas si tout ça vous passe au-dessus de la tête.

Une fois que vous serez au parfum du rôle des fonctions, vous verrez que cette page du manuel PHP nous présente tout ce dont nous avons besoin pour utiliser cette fonction `strrev()`, avec même des exemples d'utilisation concrets.

Une fois encore, en posant la bonne question, nous avons eu en retour toutes les réponses nécessaires à la poursuite de notre travail. Nous ne connaissions évidemment pas du tout cette fonction `strrev()` avant ; cependant, nous savions précisément quel était notre problème. C'était amplement suffisant pour nous mener vers la solution. Et peu importe si nous devons revenir à cette page plus tard.

Peut-être, en effet, que nous n'utiliserons pas cette fonction suffisamment souvent pour avoir vraiment besoin de nous en souvenir dans les détails. En revanche, si on s'aperçoit qu'on utilise de plus en plus cette fonction et qu'on se rend fréquemment sur la page concernée, et bien très vite, on deviendra indépendant sur la question : on pensera instantanément 'Hé, je devrais utiliser cette fonction `strrev()` dont je me sers tout le temps et dont je sais exactement ce qu'elle fait'. Cela deviendra de la 'mémoire musculaire' et ça fera partie de votre boîte à outils personnelle.

En résumé, la leçon que j'espère que vous retiendrez de ce chapitre est qu'il ne faut pas paniquer. Vous ne retiendrez pas tout et c'est tout à fait normal de demander de l'aide. C'est même tout à fait humain, comme c'est humain d'apprendre de ses propres expériences.

Félicitations ! Vous êtes un humain, pas un robot.

4. Fichiers

Attention, scoop. Le code PHP est stocké dans des fichiers. J'en suis navré mais c'est vrai ! Préparez-vous donc à travailler avec beaucoup, beaucoup de fichiers. Bon, en fait, parfois ce ne sera qu'un fichier mais, plus tard, vous travaillerez forcément avec beaucoup, beaucoup de fichiers !

Cette vérité choquante révélée, il est temps pour vous de créer votre premier fichier PHP.

> Dayle, c'est bon, je connais les bases d'un système de fichiers d'ordinateur.

Bravo mon pote ! C'est très bien mais en fait ce n'est pas vraiment de ça qu'il est question ici. En fait, la plupart des fichiers PHP ont quelque chose en commun. Je veux parler de la balise de script PHP.

Regardons d'un peu plus près ce petit bonhomme.

Exemple 01 : balise PHP .

```
1 <?php
```

Splendide, n'est-ce pas ? Admirez la brillance de cette fourrure… Un spécimen tout simplement fantastique.

> Euh… je…

Quoi ? Cette balise ne vous inspire pas la même fascination que moi ? Faites-moi confiance, après plusieurs années de développement en PHP, vous aussi, vous la trouverez splendide. Vous la verrez même les yeux fermés, la nuit, pendant votre sommeil. C'est votre meilleure amie. Elle vous permet d'utiliser PHP.

Bref, j'ai toujours préféré commencer par un exemple concret donc tentons quelque chose ensemble. Créez un nouveau fichier et appelez-le `test.php`. Les fichiers PHP prennent souvent l'extension `.php`. En toute franchise, cette extension n'est pas nécessaire pour exécuter le fichier mais autant s'y tenir, ne serait-ce que parce que, dans le cas contraire, les développeurs seniors se moqueront de vous, vous voleront votre déjeuner et vous feront pleurer. Allez, je plaisante… les développeurs sont des gens très sympa. Mais utilisez quand même l'extension `.php`.

Pour commencer, saisissons les mots suivants…

Exemple 02 : Du texte.

```
1 Vive les pandas !
```

...dans le fichier et enregistrons-le.

Super, maintenant exécutons le fichier. Pour ce faire, nous pouvons appeler l'application `php` en ligne de commande ou le shell Unix et passer le nom du fichier en paramètre. Par exemple, sur mon Mac, je saisis la ligne suivante.

Exemple 03 : Exécuter un fichier PHP.

```
1 php test.php
```

Vous devriez voir la phrase `Vive les pandas !` s'afficher sur l'écran. Ceci est dû au fait que tout ce qui est à l'intérieur de nos magnifiques balises PHP est affiché quand l'application est exécutée. Essayons autre chose. Servons-nous de notre première balise PHP.

Modifions le fichier pour qu'il corresponde à ceci.

Exemple 04 : segment PHP.

```
1 <?php
2
3 // Vive les pandas !
4
5 ?>
6 Vive les pandas !
```

Exécutons à nouveau le fichier. Qu'obtenons-nous comme sortie ?

Exemple 05 : Résultat.

```
1 Vive les pandas !
```

> Hé, attendez ! Où est passé le reste ?

Bien vu, mon bientôt-développeur ! Il manque effectivement une partie à notre fichier. Ceci est, là encore, dû au fait que tout ce qui se trouve entre nos balises PHP est interprété comme du code PHP et est donc traité en conséquence.

Du coup, qu'est-ce que c'est au final que ces balises PHP ? Et bien, vous avez déjà fait la connaissance de la balise PHP d'ouverture. Vous vous souvenez de notre

splendide amie `<?php` ? Cette balise `<?php` indique le début de notre code PHP. Mais alors quand est-ce qu'il finit ? Et bien, c'est là où la balise `?>` intervient.

Maintenant que vous savez comment les balises PHP fonctionnent, c'est très facile pour nous d'identifier où se loge le code PHP dans ce fichier. C'est donc la ligne suivante.

Exemple 06 : Commentaire.

```
1 // Vive les pandas !
```

Que fait cette ligne au juste ? Absolument rien, en fait. C'est ce qu'on appelle un commentaire. Les développeurs s'en servent pour documenter leur propre code sans que ce contenu ne s'affiche. Pas de panique. Nous ne pencherons sur les commentaires un peu plus tard.

Et bien, ce fut un joli petit chapitre, non ? Maintenant, place aux bonnes nouvelles. Dans le chapitre suivant, vous allez écrire vos premières ***vraies*** lignes de code PHP.

Excité ? Alors, pourquoi attendre ? Tournez cette page.

5. Notions arithmétiques de base

Allez, je suis sûr que vous avez déjà entendu dire que la programmation, en fait, c'est que des maths. Pas vrai ? Bon, gagné, c'est l'heure des maths. C'est parti.

$$\left|\sum_{i=1}^{n} a_i b_i\right| \le \left(\sum_{i=1}^{n} a_i^2\right)^{1/2} \left(\sum_{i=1}^{n} b_i^2\right)^{1/2}$$

Maintenant résolvez X.

Allez, je plaisante. D'ailleurs, il n'y a même pas de X dans cette équation. En fait, ce n'est même pas une équation, c'est vous dire si ma blague était mauvaise… Mais bon, qui a dit que toutes les blagues devaient forcément être à mourir de rire ? Pour vous dire la vérité, je n'ai absolument aucune idée de ce que ce truc ci-dessus est censé faire. Nous ne sommes pas tous des gourous en maths.

Instructions

Essayons quelque chose qui est un peu plus proche de mon niveau de mathématiques. Vous savez comment faire un fichier PHP et vous savez comment ouvrir et fermer des balises PHP. Donc prenons directement un fichier PHP et appelons-le `math.php`. En voici le contenu.

Exemple 01 : Addition.

```php
<?php

3 + 3;

?>
```

Alors, en fait, attendez un peu. On ne va rien générer après notre code PHP. Du coup, pourquoi s'embêter avec la balise de fermeture ? Pour être tout à fait franc, la plupart des développeurs PHP ignorent cette balise PHP fermante si rien ne suit le code PHP correspondant. Autant le faire nous aussi.

Exemple 02 : **La balise de fermeture est inutile**.

```
1 <?php
2
3 3 + 3;
```

Bien mieux !

OK, donc, au cas où votre niveau de mathématiques ne serait pas aussi pointu que le mien, laissez-moi vous aider un peu. Quand vous additionnez trois et trois, vous obtenez six. Voilà, maintenant vous êtes fin prêt.

La ligne `3 + 3;` contient une instruction. C'est une ligne de code PHP qui va être évaluée par PHP. Elle se finit normalement par un point-virgule, comme ceci : `;`. Au début, ces points-virgules, vous n'allez pas arrêter de les oublier mais ne vous inquiétez pas, très vite, vous terminerez toutes vos phrases avec ;

Partons du principe que vous maîtrisez maintenant les additions de base : que pensez-vous que va générer l'exécution de ce fichier ?

> 7 virgule 5.

Et bien, cher lecteur, euh, un peu spécial, voyons si vous avez raison. Lançons `php math.php` pour voir ce qui se passe.

Exemple 03 : **Résultat**.

```
1 [rien]
```

Woh ! Absolument rien. Ce langage est vraiment stupide. Laissons tomber. OK, je plaisantais encore. Oui, j'ai un sens de l'humour très particulier, mais pas de stress, vous vous y habituerez.

Comment se fait-il que n'ayons obtenu aucun résultat ? Et bien, c'est parce que nous n'avons pas dit à PHP de générer quoi que ce soit. PHP est très obéissant, il suffit de lui demander de nous donner la réponse. Pour ce faire, nous allons utiliser `echo`. C'est une structure du langage PHP qui nous permet de visualiser le résultat d'une instruction.

Modifions donc notre instruction pour y inclure `echo`.

Exemple 04 : L'instruction echo.

```
<?php

echo 3 + 3;
```

Et voilà. Nous avons placé `echo` juste avant l'expression dont nous voulons voir le résultat. Lançons à nouveau notre application ! C'est parti…

Exemple 05 : Résultat.

```
6
```

Woohoo ! Six ! **MÊME PAS SEPT VIRGULE CINQ** ! Voilà, là, c'est ce que je disais. Nous pouvons enfin visualiser le résultat de l'évaluation PHP de notre première instruction. Plutôt excitant, non ?

> J'aurais pu faire la même chose avec une calculatrice.

Je sais, je sais. Ce n'est pas vraiment de la physique quantique. La physique quantique est traitée dans un autre chapitre de ce livre… Attendez, j'ai déjà fait cette blague dans un autre de mes livres. Il faut vraiment que je me renouvelle.

Opérateurs arithmétiques

Je sais bien que notre exemple de `3 + 3` est du code très simple mais nous allons vite arriver à plus grand et mieux. Saviez-vous qu'il existait d'autres opérateurs arithmétiques ? Je suis sûr que ça va vous rappeler quelque chose.

```
+ Addition
- Soustraction
* Multiplication
/ Division
% Modulo
```

Allez, je suis sûr que vous avez déjà rencontré ces opérateurs. Je me doute bien que la multiplication et la division ont ici des signes un peu différents de ceux que vous avez pu apprendre à l'école. Ils sont en fait communs à la plupart des langages de programmation et vous verrez que le signe de division, par exemple, est vraiment

plus facile à saisir sous cette forme sur un clavier. Mais ne vous tracassez pas, vous les aurez vite retenus et adoptés.

Il se peut en revanche que vous ne connaissiez pas l'opérateur 'Modulo' - mais il est très simple à expliquer. On l'utilise pour calculer le reste d'une division. Par exemple, l'opération '3 % 2' donnera en résultat '1'. Dans la pratique, on l'utilise pour déterminer si un nombre est pair ou impair, en le divisant simplement par deux.

Et si on donnait du vrai grain à moudre à PHP maintenant ?

Exemple 06 : Maths plus complexes.

```
1  <?php
2
3  echo 4 + 3 * 2 / 1;
```

Alors, quel est le résultat ? Cela peut être un peu difficile de le calculer de tête parce que nous ne savons pas vraiment dans quel ordre traiter les calculs internes. Faut-il additionner 4 et 3 pour commencer ? Ou peut-être diviser 2 par 1 d'abord. Hmmm. Subtil !

Bien sûr, en mathématiques, on nous a appris à utiliser des parenthèses pour séparer les différentes parties d'une opération. On peut faire la même chose avec PHP. Essayons pour voir.

Exemple 07 : Parenthèses pour différencier les parties d'une opération.

```
1  <?php
2
3  echo (4 + 3) * (2 / 1);
```

Là, nous sommes sûrs que `4 + 3` et `2 / 1` sont d'abord évalués puis que leurs résultats respectifs sont multipliés. Super, lançons notre script et regardons le résultat…

Exemple 08 : Résultat.

```
1  14
```

Génial… mais est-ce qu'on n'a pas un peu triché ? Quel résultat obtiendrions-nous sans les parenthèses ? Enlevons-les à nouveau.

Exemple 09 : Sans parenthèses.

```
<?php

echo 4 + 3 * 2 / 1;
```

Alors, ce résultat ? Lançons notre script.

Exemple 10 : Résultat.

```
10
```

C'est un nombre complètement différent. A quoi est-ce dû ? En fait, c'est tout simplement que PHP ne gère pas les opérateurs dans le même ordre. Prenons donc un peu de temps pour apprendre ces ordres d'opérateurs en PHP.

Voilà comment PHP définit les ordres d'importance de ces opérateurs.

```
* Multiplication
/ Division
% Modulo
+ Addition
- Soustraction
```

L'opérateur à la priorité la plus grande se trouve en haut de cette liste. Ce qui signifie que quand PHP examine `4 + 3 * 2 / 1` il calcule tout d'abord le résultat de `3 * 2 = 6`, puis celui de `6 / 1 = 6` et enfin celui de `4 + 6` pour nous donner, au final, le résultat `10`.

Pour ma part, quand j'écris du code contenant des opérations mathématiques, je préfère utiliser systématiquement des parenthèses pour éviter toute confusion. Je trouve aussi que cela clarifie l'intention du code et que ça en rend la lecture plus facile.

Procédures

Le code PHP est analysé de manière procédurale. Cela signifie qu'il est lu et exécuté instruction après instruction. S'il est possible d'inclure plusieurs instructions sur une même ligne, c'est assez rare chez les développeurs PHP. Cela signifie donc que nous pouvons aborder le code ligne par ligne. On peut voir ce principe en action en ajoutant des instructions à notre fichier PHP. Essayons ceci par exemple.

Exemple 11 : Instructions multiples.

```
<?php

echo 2 + 2;
echo 3 + 3;
echo 4 + 4;
echo 5 + 5;
```

Exécutons maintenant le fichier…

Exemple 12 : Résultat.

```
46810
```

QUARANTE-SIX MILLE GIGAWATTS !?

Du calme, lecteur, lectrice ! Nous avons simplement demandé à PHP de générer le résultat, pas d'insérer des espaces ou des lignes dans ce résultat. PHP a donc bien calculé correctement les valeurs. Si nous introduisons des espaces dans le résultat fourni par PHP de cette manière…

4 6 8 10

…nous voyons bien que les résultats sont en effet corrects. C'est juste que PHP est très obéissant et n'a généré les valeurs que les unes après les autres, comme demandé.

J'ai déjà pu indiquer que PHP est un langage flexible et plutôt indulgent. Mettons cette affirmation à l'épreuve, pour voir, vous voulez bien ? Jusqu'ici toutes nos instructions ne comportaient qu'un unique espace entre chaque 'mot' (ou nombre). Ajoutons des espaces de manière complètement aléatoire pour voir ce qui se produit. Voici notre code modifié.

Exemple 14 : Espaces.

```
<?php

echo    2          +   2;
echo                   3   +3;
echo 4+4;
echo    5+     5        ;
```

Manifestement, ce n'est pas très beau… mais si vous deviez exécuter ce code en l'état, vous verriez qu'il fonctionnerait parfaitement. PHP ne soucie en effet pas du nombre d'espaces entre les mots présents dans son code. Il s'en accommode tout simplement. (Insérer ici un chien cool avec des lunettes noires…)

Vous remarquerez également que certaines opérations arithmétiques, comme `4+4` ne requièrent aucun espace. Ce principe ne s'applique toutefois pas à toutes les variations syntaxiques. Regardez par exemple l'extrait de code suivant.

Exemple 15 : Pas d'espace après echo.

```
1  <?php
2
3  echo5 + 5;
```

Si vous essayez d'exécuter ce script, vous verrez que PHP affiche un avertissement, 'Use of undefined constant echo5 - assumed 'echo5'', soit, en français, 'Utilisation d'une constante echo5 indéfinie - 'echo5' supposé'. Ceci est dû au fait que PHP ne sait pas ce que le mot `echo5` lui demande de faire. Pour éviter ce genre d'écueil, il est donc préférable de toujours insérer un espace entre les mots.

Pour ce qui est des instructions, si nous avions quelque penchant masochiste, nous pourrions choisir de toutes les placer sur une unique ligne. Voici un exemple :

Exemple 16 : Instructions multiples, ligne unique.

```
1  <?php echo 2 + 2; echo 3 + 3; echo 4 + 4; echo 5 + 5;
```

Ce code PHP est tout à fait valide mais vous trouverez peu de développeurs qui en suivent le principe. Placer une instruction par ligne rend le code beaucoup plus lisible et la compréhension du fichier source beaucoup plus facile. Cela évite aussi toutes sortes de problèmes pour les systèmes de gestion de versions !

Nous avons donc compris que PHP ignore les espaces multiples présents dans son code. Sachez que, qui plus est, il considère qu'un retour à la ligne est un espace. En d'autres termes, l'extrait de code suivant est tout à fait correct.

Exemple 17 : Une instruction, plusieurs lignes.

```
1  <?php
2
3  echo
4  2
5  +
6  2
7  ;
```

Vous ne me croyez pas ? Allez-y, essayez ! Mais, là encore, si le code est bien fonctionnel, il n'est pas franchement très agréable à lire. Donc, si je vous surprends à rédiger du code de cette manière, c'est fessée direct !

Il existe toutefois un usage bien pratique à ce retour à la ligne. En effet, si la ligne est excessivement longue, c'est un autre problème de lisibilité qui se pose. Il peut être résolu en insérant un retour à la ligne là où vous estimez qu'est atteinte une longueur de ligne standard pour la lecture. Beaucoup de développeurs intègrent en complément quatre espaces (ou une tabulation paramétrée) à la ligne suivante pour en indiquer la continuation avec la précédente.

Voici un exemple de retours à la ligne utilisés pour améliorer la lisibilité d'une instruction.

Exemple 18 : Césure de ligne impeccable.

```
1  <?php
2
3  echo (3 * 5) / (7 / 12) * (7 * 6) + (7 % 3)
4          + (6 + 7) * (12 / 3);
```

On est dans des maths plutôt costaudes ici mais, avec un peu de chance, vous trouverez l'ensemble plus lisible.

Il est aussi intéressant de noter que vous pouvez placer des lignes vides au sein de votre code, là encore pour le rendre plus lisible. Voici un exemple.

Exemple 19 : Plus de retours à la ligne pour une meilleure lisibilité.

```
1  <?php
2
3  echo 3 + 2;
4
5  echo 7 * 7;
6
7  echo 5;
```

Vous le voyez, PHP est extrêmement flexible mais n'oubliez pas de saisir ce point-virgule en fin de ligne, car, dans le cas contraire, il ne vous le pardonnerait jamais.

JAMAIS ;

6. Variables & affectation

On rentre enfin dans le vif du sujet ! Les variables sont une partie extrêmement utile - et copieusement sollicitée - de la boîte à outils des développeurs. On commence, si vous le voulez bien.

Les petites boîtes

Essayez d'imaginer que les variables sont de toutes petites boîtes dans lesquelles on peut ranger des choses. Les variables sont des mots qui commencent par le signe dollar, `$`. Voyons tout de suite un exemple.

Exemple 01 : **Affectation de base**.

```
1  <?php
2
3  $three = 3;
```

Imaginez que la variable `$three` est une petite boîte : ici, nous venons simplement de mettre la valeur `3` dedans. C'est à ça que sert ce signe "égal". En maths, on utilise le signe "égal" pour indiquer le résultat d'une opération mais, en PHP, c'est une toute autre histoire.

En PHP le signe égal, `=`, est appelé un opérateur d'affectation. Il est utilisé pour ***définir*** quelque chose. Ici, nous disons à PHP de ***définir*** la variable `$three` comme le chiffre `3`.

Si vous exécutez le script créé ci-dessus, vous verrez que PHP ne génère rien. C'est dû au fait qu'une affectation… et bien, n'est que cela : une affectation. Nous n'avons pas demandé à PHP de *produire* quoi que ce soit. En revanche, maintenant que nous avons paramétré la variable `$three` à `3`, nous pouvons y appliquer la structure de langage `echo`.

Exemple 02 : Faire ‘echo’ sur une valeur.

```php
<?php

// Définir notre variable à la valeur trois.
$three = 3;

// Générer la valeur de la variable.
echo $three;
```

Nous définissons tout d’abord notre variable et ensuite nous utilisons la structure de langage `echo` pour produire la valeur qu’elle contient. Si nous exécutons notre code, le résultat est bien `3`.

C’est génial parce que ça veut dire que nous pouvons donner des surnoms aux choses. Vous savez, comme on faisait avec ces petites teignes à l’école… Par exemple, le nombre ‘3.14159265359’ est un très beau nombre aux yeux des adorateurs de cercles mais il est incroyablement difficile à retenir, on est d’accord ? Donnons-lui un surnom. Nous l’appellerons Pete. Non, attendez, j’ai une meilleure idée.

Exemple 03 : Un nom de variable approprié.

```php
<?php

$pi = 3.14159265359;
```

Voilà, nous avons créé une nouvelle variable `$pi` qui contient la valeur `3.14159265359`. Ça signifie que nous pouvons utiliser cette variable n’importe où dans notre code, par exemple pour effectuer des calculs. Voici des exemples.

Exemple 04 : Utiliser des variables dans des instructions.

```php
<?php

// Affecter pi à une variable.
$pi = 3.14159265359;

// Effectuer des calculs de circonférence.
echo $pi * 5;
echo $pi * 3;
```

Une fois `$pi` paramétré, nous pouvons ainsi l'utiliser dans d'autres instructions pour effectuer des calculs.

Il est possible de déclarer et d'affecter autant de variables que souhaité mais il existe un certain nombre de règles à suivre pour en choisir les noms. Les noms de variables peuvent en effet contenir des chiffres, des lettres et des tirets bas (*underscores*) mais ils ***doivent*** absolument commencer par, soit une lettre, soit un tiret bas - jamais un chiffre ! Ils sont par ailleurs sensibles à la casse, ce qui signifie que `$panda` est différent de `$pAnda`. Voici quelques exemples.

Exemple 05 : Nommer des variables.

```php
<?php

$panda = 1;     // Correct
$Panda = 1;     // Correct
$_panda = 1;    // Correct
$pan_da = 1;    // Correct
$pan_d4 = 1;    // Correct
$pan-da = 1;    // Incorrect
$4panda = 1;    // Incorrect
```

Même si, on l'a vu, les noms de variables peuvent contenir des tirets bas et commencer par des capitales, l'usage commun pour mieux différencier les mots consiste à recourir à un format de nommage appelé "casse de chameau". Pas de panique, nous n'aurons pas besoin de chameau.

Les noms en *casseDeChameau* commencent par une lettre en bas de casse (minuscule). Les variables dont le nom contient plusieurs mots auront chacun de leurs mots suivant le tout premier en capitales. Voici quelques exemples.

Exemple 06 : Noms de variables en casseDeChameau.

```php
<?php

$earthWormJim = 1;
$powerRangers = 1;
$spongeBobSquarePants = 1;
```

Vous vous rappelez comment nos instructions retournent une valeur ? Et bien, figurez-vous que nos affectations sont aussi des instructions. Vous voyez ce que cela signifie ? C'est ça, elles retournent elles aussi une valeur. Nous pouvons même le prouver en utilisant notre bon vieille structure de langage `echo`.

Exemple 07 : Instructions retournant une valeur.

```php
<?php

echo $panda = 1337;
```

Le résultat affiché est le nombre `1337`. Ceci s'explique par le fait que l'affectation de la variable `$panda` est effectuée avant que sa valeur ne soit affichée. Cette séquence de traitement nous permet d'ailleurs de pouvoir utiliser une astuce assez fine. Ce n'est pas quelque chose que vous utiliserez souvent mais je pense que ça reste un bon petit truc à connaître, au cas où. Allez-y, jetez un œil à cet exemple.

Exemple 08 : Affectation multiple.

```php
<?php

$firstPanda = $secondPanda = $thirdPanda = 1337;
```

Bon, l'extrait de code ci-dessus peut paraître un peu fracassé mais il devient plus cohérent si vous le lisez de droite à gauche. La variable `$thirdPanda` se voit affecter la valeur `1337`, puis la variable `$secondPanda` prend la valeur de `$thirdPanda` et, enfin, la variable `$firstPanda` reçoit la valeur de `$secondPanda`. En d'autres termes, toutes les variables sont définies comme la valeur finale. Impeccable, non ?

Exactement mon type

Jusqu'ici, nous n'avons travaillé qu'avec des chiffres. Ce serait d'un ennui inimaginable si c'étaient les seuls types de valeur que nous puissions utiliser, vous ne trouvez pas ? Il est donc temps d'examiner les autres possibilités. Voici quelques-uns des types de valeur utilisés au sein d'applications PHP.

- nombres entiers
- nombres décimaux
- booléens
- chaînes de caractères
- null
- tableaux

Il en existe encore quelques-unes mais ne compliquons pas les choses tout de suite. Il faut vraiment apprendre les choses petit à petit. Ne risquons pas la surcharge d'informations !

Jetons un œil à ces types, l'un après l'autre. Nous avons tout d'abord les nombres entiers (*integer* en anglais). Ce sont des nombres sans décimales, comme nous avons pu en utiliser dans les exemples précédents.

Exemple 09 : Nombres entiers.

```php
<?php

$panda = 2;
$redPanda = -23;
```

Les nombres à décimales (*floats* en anglais) sont, et bien, des nombres avec des décimales après leur virgule. Ayant des décimales, ils contiennent donc des fractions. Ils peuvent être utilisés exactement comme des nombres entiers. En fait, on en a même déjà utilisé un. Vous vous souvenez de notre ami `$pi` ? Et oui, c'était un nombre à décimales. Et maintenant, on passe à autre chose, d'accord ?

Exemple 10 : Nombres à décimales.

```php
<?php

$panda = 2.34;
$redPanda = -23.43;
```

Les booléens sont des types de données binaires. Mais pas de stress ! On ne va pas se lancer dans de l'arithmétique binaire… C'est simplement une manière de dire que ces données peuvent avoir une valeur parmi deux possibles. Un booléen peut ainsi être soit `true` (vrai), soit `false` (faux). Nous verrons un peu plus loin comment on peut utiliser ces valeurs booléennes pour modifier le flux de notre application.

Exemple 11 : Booléens

```php
<?php

$panda = false;
$redPanda = true;
```

Ensuite, nous avons la valeur 'chaîne de caractères' (*string*, en anglais). Les chaînes de caractères sont utilisées pour stocker un mot, un caractère ou une séquence de texte. Les chaînes de caractères sont très spécifiques, à tel point que j'ai choisi de leur consacrer un (court) chapitre entier. Nous y reviendrons donc !

Exemple 12 : Chaînes de caractères.

```php
<?php

$panda = 'Normal Panda';
$redPanda = "Red Panda";
```

'Null' est une valeur un peu spéciale. C'est… rien. Nul. Zéro. Bon, en fait, ce n'est pas zéro. Zéro est une valeur numérique et nous pouvons donc utiliser les nombres entiers pour ça. Les 'nulls' sont très précisément 'rien'. 'Null' est aussi la valeur de toute variable avant son affectation. C'est une valeur extrêmement pratique et vous la rencontrerez beaucoup prochainement.

Exemple 13 : Les valeurs 'null'.

```php
<?php

$noPanda = null;
```

Enfin, les tableaux (*arrays*, en anglais) sont un autre type de valeurs un peu spéciales. En fait, pour tout vous dire, c'est mon type de valeur préféré. À tel point que, là encore, j'ai choisi de leur consacrer un chapitre entier. Pour l'heure, tout ce que vous avez besoin de savoir, c'est que c'est une valeur qui contient un ensemble d'autres valeurs. Dingue ! C'est quasi du Inception, non ?

Exemple 14 : Tableaux.

```php
<?php

$countThePandas = [1, 2, 3];
$morePandas = array(5, 6, 7, 8);
```

Affectation avancée

Dans un chapitre précédent, nous avons vu les différents types d'opérateurs qui peuvent être utilisés avec les variables et nous avons même, allez, disons-le, maîtrisé l'opérateur d'affectation. Mais alors que se passe-t-il quand on les met ensemble ? Est-ce que ça va créer un nouveau trou noir et absorber tout l'univers ? Je me sens un peu aventureux, on tente le coup ?

Exemple 15 : Affectation avec addition.

```php
<?php

// Définir une valeur.
$panda = 3;

// Essayer de créer un nouveau trou noir.
$panda += 1;

// L'univers prêt à résister, faire un var_dump de la valeur.
var_dump($panda);
```

On commence par définir une variable avec la valeur de nombre entier ‘trois’. Ensuite, on insère l’opérateur d’addition juste avant l’opérateur d’affectation et on affecte la valeur de nombre entier ‘un’ à l’ensemble.

Nous pouvons utiliser la fonction `var_dump()` (nous aurons l’occasion de parler des fonctions plus tard !) pour connaître non seulement la valeur contenue par la variable mais aussi son type !

Qu’obtenons-nous de ce ‘dump’ ?

Exemple 16 : Résultat.

```
int(4)
```

Génial ! L’univers est sauvé. On dirait bien que nous obtenons un quatre, non ? Et bien, je suppose que c’est plutôt normal… Nous savons en effet que `$a + $b` retourne une valeur sans la définir ; et nous savons aussi que l’opérateur d’affectation sert justement à définir la valeur des variables. Ici, nous faisons les deux en même temps. Nous disons à PHP de paramétrer la valeur de `$panda` à sa valeur en cours, à laquelle nous ajoutons ‘un’.

Vous pouvez utiliser cette syntaxe avec n’importe lequel des opérateurs que nous avons vus jusqu’ici. Il y a un seul truc à retenir : ne placez jamais l’opérateur *de l’autre côté* du signe ‘égal’. Faites-moi confiance, j’ai déjà essayé. S’est alors ouverte une porte sur d’obscurs bas-fonds dont se sont échappées des créatures mi-hommes, mi-dinosaures qui ont fondu sur Cardiff pour en terroriser les habitants. Ce n’est qu’à l’aide d’un lance-flammes maison (propulsé par PHP) que j’ai pu repousser les assauts de ces vicieux démons. Je ne souhaite ça à personne d’autre, et encore moins à vous. Soyez prudent(e)s, donc !

Ensuite, nous avons l'opérateur d'incrémentation. En fait, n'oublions pas tout d'abord l'opérateur de décrémentation. On a tendance à lui accorder moins d'attention. Tiens, d'ailleurs, mettons un peu en avant ses atouts.

Comme toujours, je préfère recourir à un exemple. Considérons l'extrait de code suivant.

Exemple 17 : – Après.

```php
<?php

// Définir une valeur.
$panda = 3;

// Diminuer la valeur.
$panda--;

// 'Dumper' la valeur.
var_dump($panda);
```

Là, juste au milieu, vous le voyez ? Ce splendide opérateur de décrémentation… Il suffit simplement d'insérer deux signes 'moins' après la variable. Et ça fait quoi, au juste ? Et bien, voici le résultat de notre bout de code.

Exemple 18 : Résultat.

```
int(2)
```

Comme vous pouvez le voir, la valeur de `$panda` a été dimininuée de un. C'est un raccourci très rapide pour diminuer une valeur. De la même manière, si vous utilisez un `++`, vous augmenterez la valeur concernée. Notez bien toutefois que ce sont les deux seuls opérateurs qui fonctionnent ainsi. Ne faites pas les malins en essayant d'utiliser l'opérateur de multiplication. Ça ne fonctionnera pas du tout comme vous le souhaitez !

Tenez, je me demande ce qui se passerait si nous placions l'opérateur *avant* la variable. Tentons le coup, vous voulez bien ?

Exemple 19 : – Avant.

```php
<?php

// Définir une valeur.
$panda = 3;

// Diminuer la valeur.
--$panda;

// 'Dumper' la valeur.
var_dump($panda);
```

Alors, quelle est la réponse ? Vous ne trouvez pas ça excitant ?

Exemple 20 : Résultat.

```
int(2)
```

Euh… c'est exactement pareil. L'exemple fut peu passionnant, non ? En fait, j'ai un petit secret à partager : ce n'est pas la même chose. Certes, la valeur qui nous est renvoyée semble être la même mais mon exemple ne lui rend pas vraiment justice.

Construisons un exemple différent. Nous allons afficher l'état d'une valeur **avant** que l'opérateur soit utilisé. Nous inspecterons ensuite le résultat de l'instruction **quand** le calcul est utilisé puis, pour terminer, nous vérifierons la valeur **après** l'utilisation de l'opérateur. A priori, aucune raison que la valeur 'd'après' soit différente.

Exemple 21 : Les étapes de –.

```php
<?php

// Définir une valeur.
$panda = 3;

// 'Dump' AVANT.
var_dump($panda);

// 'Dump' PENDANT.
var_dump(--$panda);

// 'Dump' APRÈS.
var_dump($panda);
```

Maintenant exécutons le code. Quelles sont les trois valeurs qui nous sont retournées ?

Exemple 22 : Résultat.

```
int(3)
int(2)
int(2)
```

La première valeur est donc ‘trois’. Nous nous y attendions - après tout, nous n’avons fait que de la définir comme telle, non ? Le résultat de l’instruction utilisant l’opérateur de décrémentation est égal à ‘deux’. Le résultat de la valeur est également ‘deux’. Ça signifie donc que la valeur est diminuée à la deuxième ligne.

Maitenant, déplaçons l’opérateur de l’autre côté de la valeur, d’accord ? Comme ceci :

Exemple 23 : Les étapes de – deuxième partie.

```
<?php

// Définir une valeur.
$panda = 3;

// 'Dump' AVANT.
var_dump($panda);

// 'Dump' PENDANT.
var_dump($panda--);

// 'Dump' APRÈS.
var_dump($panda);
```

Regardez attentivement pour bien identifier la différence. Jetons à nouveau un œil.

Exemple 24 : Résultat.

```
int(3)
int(3)
int(2)
```

Hé !? La valeur du milieu, là, est différente ! Pourquoi n’a-t-elle pas été diminuée ? Et bien, en fait, en intervertissant l’opérateur, nous avons demandé à PHP de diminuer

la valeur **APRÈS** la ligne en cours. Le résultat de l'opération de cette ligne est la même valeur que la valeur initiale.

Laissez-moi résumer…

- **$value–** - Modifie la valeur *après* la ligne en cours.
- **–$value** - Modifie la valeur de la ligne en cours.

En quoi est-ce utile ? Et bien, voici un premier cas pratique pour vous. Si vous êtes un peu créatifs, je suis sûr que vous en trouverez d'autres. En utilisant l'opérateur qui modifie **après** la ligne en cours, nous pouvons définir une autre variable avec cette valeur et diminuer la valeur originelle sur la même ligne. Comme ceci :

Exemple 25 : Affecter et augmenter.

```php
<?php

// Définir une valeur.
$panda = 3;

// Affecter puis augmenter.
$pandaFriend = $panda++;
```

Ce que nous avons fait ici, c'est d'économiser une ligne. C'est une espèce de raccourci, si vous voulez. Voici à quoi ça ressemblerait si nous n'utilisions pas cet opérateur d'incrémentation.

Exemple 26 : Explication de l'incrémentation.

```php
<?php

// Définir.
$panda = 3;

// Affecter.
$pandaFriend = $panda;

// Incrémenter.
$panda = $panda + 1;
```

Dans un autre chapitre, consacré, celui-ci, aux boucles, vous trouverez un autre exemple d'utilisation de cet opérateur. Mais, pour l'heure, dans le chapitre suivant, nous allons nous pencher sur les chaînes de caractères.