PHP

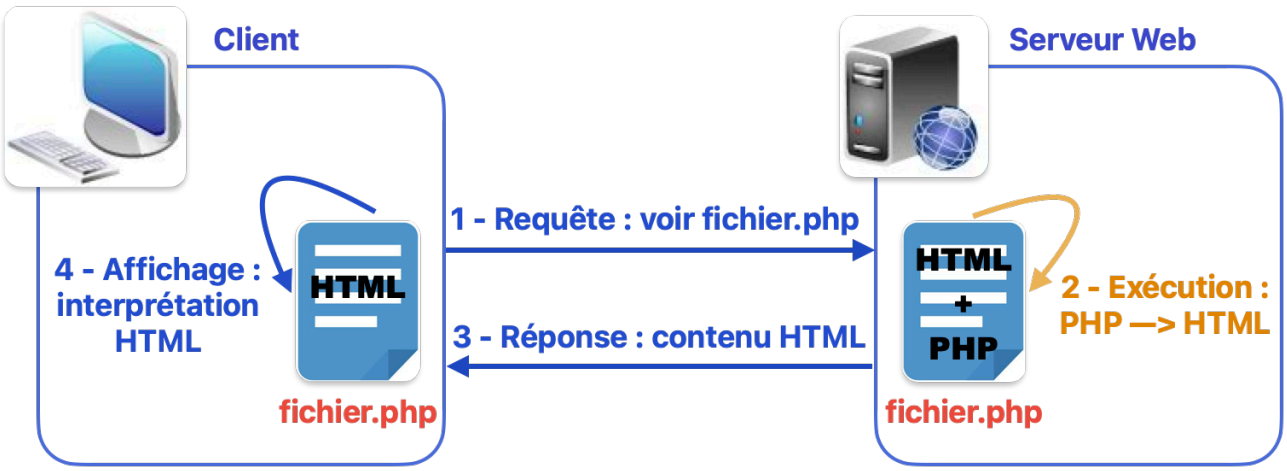
Introduction

Un site web créé uniquement avec des langages qui s'exécutent côte client sera statique tandis qu'un langage créé avec des langages qui s'exécutent côté client et des langages qui s'exécutent côté serveur sera généralement dynamique.
Le PHP permet de créer des pages qui vont être générées dynamiquement.

Contrairement au HTML et au CSS qui sont de véritables standards, le PHP et le MySQL ont de nombreux concurrents : Python, Ruby voire JavaScript pour le PHP et PostGreSQL, Microsoft SQL Server ou encore MariaDB pour le MySQL.

La grande majorité des navigateurs ne sont capables de comprendre et de n'exécuter que du code HTML, CSS et JavaScript. Un navigateur est ainsi incapable de comprendre du code PHP.
Lorsqu'un navigateur demande à un serveur de lui servir une page, le serveur va donc se charger d'exécuter tout code qui ne serait pas compréhensible par le navigateur (en utilisant au besoin un interpréteur).
Une fois ces opérations effectuées, le serveur va renvoyer le résultat (la page demandée après interprétation) sous forme de code compréhensible par le navigateur, c'est-à-dire principalement en HTML. Le navigateur va alors afficher la page au visiteur. Cette page va être unique puisqu'elle a été générée par le serveur pour un utilisateur spécifique et en tenant compte de données particulières.
Si en revanche la page demandée par le visiteur ne contient aucun code nécessitant l'intervention du serveur, alors le serveur va la renvoyer telle quelle au navigateur qui va l'afficher à l'utilisateur.

Les opérations réalisées côté serveur vont être transparentes pour le visiteur et que celui-ci ne va jamais avoir accès ni pouvoir consulter le code exécuté côté serveur car une fois les opérations terminées, le serveur ne va renvoyer que du code compréhensible par le navigateur.
C'est la raison pour laquelle lorsqu'on analyse le code d'une page, on ne trouvera jamais d'instructions PHP mais seulement généralement du code HTML, CSS et JavaScript qui sont des langages qui s'exécutent côté client.



Lorsque l'on code, on peut travailler soit en local, c'est-à-dire en hébergeant nous-mêmes tous les fichiers sur nos propres machines (hors ligne), soit en production (serveur) sur des fichiers qui sont hébergés sur un serveur distant et potentiellement accessibles à tous.
On ne travaillera évidemment pas sur un site live car on n'en a pas mais plutôt en local. Cependant, on sait que le PHP et le MySQL vont s'exécuter côté serveur.
Il va donc nous falloir recréer une architecture serveur sur nos propres machines avec les logiciels adaptés afin de pouvoir tester nos codes PHP.

L'installation de PHP en local sur l'ordinateur permet de développer et de tester en toute sécurité une application Web sans affecter le système en direct. Pour travailler avec PHP en local, on doit disposer des logiciels suivants :
- PHP
- Un serveur Web prenant en charge PHP (serveur Web Apache par exemple).
- Un serveur de base de données (serveur de base de données MySQL ou MariaDB).

En règle générale, on installera pas tous ces logiciels séparément car leur connexion est délicate. Par conséquent, il est plus facile d'installer un progiciel (produit logiciel : package) tout-en-un comprenant PHP, un serveur Web et un serveur de base de données.
L'un des environnements de développement PHP les plus populaires est XAMPP.
XAMPP est une distribution (open source) Apache facile à installer qui contient PHP, MariaDB et le serveur Web Apache. XAMPP prend en charge Windows, Linux et macOS.

- **X :** Cross Platform
- **A :** Apache Server
- **M :** MariaDB
- **P :** PHP
- **P :** Perl

Installer XAMPP

Lien de téléchargement

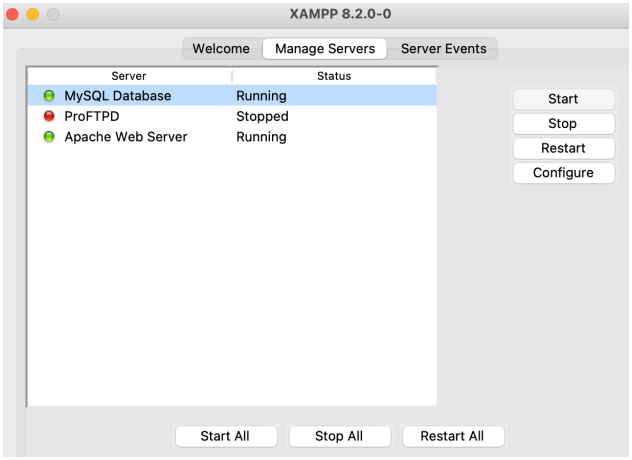
- Télécharger la bonne version de XAMPP
- Exécuter le programme d'installation avec les options par défaut
- Lancer XAMPP
- Démarrer les services via le manager

Il faudra enregistrer tous les fichiers PHP dans le dossier « xampp » et le sous-dossier « htdocs ».

Un site Web sera donc une combinaison de fichiers dans htdocs et d'une base de données MySQL. Chaque site Web se trouvera dans son propre dossier dans htdocs et aura sa propre base de données MySQL distincte.

Démarrer le logiciel Xampp.

Démarrer les serveurs, ou au contraire les arrêter, avec des voyants lumineux indiquant si les serveurs fonctionnent (voyants verts) ou pas (voyants rouges). Démarrer les serveurs si ceux-ci sont arrêtés.



Welcome to XAMPP for OS X 8.2.0

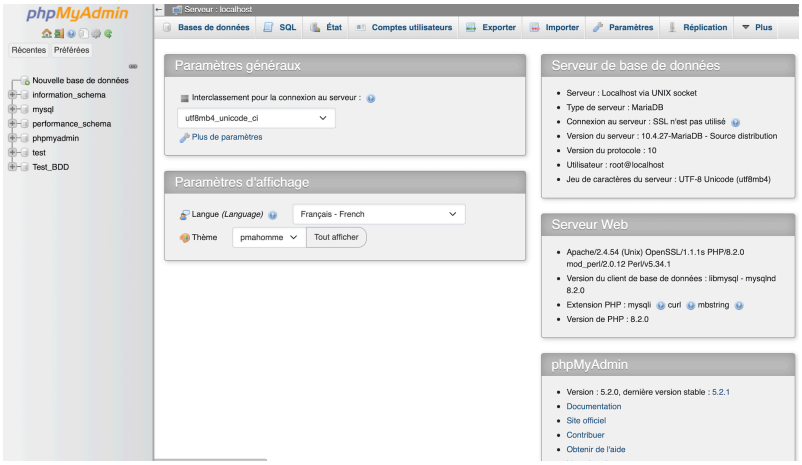
You have successfully installed XAMPP on this system! Now you can start using Apache, MariaDB, PHP and other components. You can find more info in the FAQs section or check the HOW-TO Guides for getting started with PHP applications.

XAMPP is meant only for development purposes. It has certain configuration settings that make it easy to develop locally but that are insecure if you want to have your installation accessible to others.

Start the XAMPP Control Panel to check the server status.

Community

XAMPP has been around for more than 10 years – there is a huge community behind it. You can get involved by joining our Forums, liking us on Facebook, or following our exploits on Twitter.



Pour commencer :

- Créer un dossier dans htdocs appelé "exemple1".
- Créer ensuite une base de données appelée exampleBDD (si nécessaire).
- Ouvrir un navigateur Web (Safari, Chrome, Firefox) et accéder à http://localhost/example1/
- Pour les bases de données Xampp MySQL, par défaut :
    - Username : root
    - Password : (est vide)

Il n'y a pas de mot de passe par défaut pour XAMPP MySQL. Pour le modifier, on peut utiliser l'anglet phpMyAdmin.

PHP

Un script PHP est exécuté sur le serveur et le résultat HTML brut est renvoyé au navigateur.
Un script PHP est écrit dans des fichiers dédiés, c'est-à-dire des fichiers qui ne vont contenir que du PHP, ou intégrer le PHP au sein des fichiers HTML.

Les fichiers qui contiennent du PHP vont devoir être enregistrés avec l'extension `.php`. Dans le cas où on intègre du code PHP dans un fichier HTML, il faudra également changer son extension en `.php`.

Syntaxe :

Un script PHP peut être placé n'importe où dans le document.

Un script PHP commence par
`<?php`
et se termine par
`?>`

```
<?php
// Code PHP à écrire ici
?>
```

Un fichier PHP contient normalement des balises HTML et du code de script PHP.
Ci-dessous, on a un exemple de fichier PHP simple, avec un script PHP qui utilise une fonction PHP intégrée "echo" pour afficher le texte "Lycée Charles Carnus" sur une page Web :

```
<!DOCTYPE html>
<html>
<body>

<h1>Ma page Web écrite en PHP</h1>

<?php
echo "Lycée Charles Carnus";
?>

</body>
</html>
```

Remarque : les instructions PHP se terminent par un point-virgule (`;`).

Sensibilité à la casse PHP

En PHP, les mots-clés (par exemple if, else, while, echo, etc.), les classes, les fonctions et les fonctions définies par l'utilisateur ne sont pas sensibles à la casse. Dans l'exemple ci-dessous, les trois instructions echo ci-dessous sont égales et autorisées :

```
<!DOCTYPE html>
<html>
<body>

<?php
ECHO "Lycée Charles Carnus<br>";
echo "Lycée Charles Carnus<br>";
EcHo "Lycée Charles Carnus<br>";
?>

</body>
</html>
```

Remarque : Cependant ; tous les noms de variables sont sensibles à la casse !

Les commentaires

Un commentaire dans le code PHP est une ligne qui n'est pas exécutée dans le cadre du programme. Son seul but est de faciliter la compréhension de code.
Il existe plusieurs façons de commenter le code :

```
<!DOCTYPE html>
<html>
<body>

<?php
// Commentaire sur une ligne

# Commentaire sur une ligne

/*
Commentaire
sur
plusieurs
lignes
*/
?>

</body>
</html>
```

Les variables

Les variables sont des "conteneurs" pour stocker des informations.

Créer (déclarer) des variables PHP

En PHP, une variable commence par le signe $, suivi du nom de la variable :

```php
<?php
$txt = "Bonjour";
$x = 12;
$y = 10.5;
?>
```

Remarque : Contrairement à d'autres langages de programmation, PHP (comme en Python) n'a pas de commande pour déclarer une variable. Elle est créée au moment où on lui affecte une valeur pour la première fois.

Règles pour les variables PHP :
- Une variable commence par le signe $, suivi du nom de la variable
- Un nom de variable doit commencer par une lettre ou le caractère de soulignement (underscore)
- Un nom de variable ne peut pas commencer par un chiffre
- Un nom de variable ne peut contenir que des caractères alphanumériques et des traits de soulignement (A-z, 0-9 et _)
- Les noms de variables sont sensibles à la casse ($age et $AGE sont deux variables différentes)

Variables de sortie

L'instruction PHP echo est souvent utilisée pour afficher des données à l'écran. L'exemple suivant montre comment afficher du texte et une variable :

```php
<?php
$txt = "LTP Carnus";
echo "Je suis étudiant au $txt!";
?>
```

Ou

```php
<?php
$txt = "LTP Carnus";
echo "Je suis étudiant au " . $txt . "!";
?>
```

```php
<?php
$x = 5;
$y = 4;
echo $x + $y;
?>
```

Portée des variables

En PHP, les variables peuvent être déclarées n'importe où dans le script. La portée d'une variable est la partie du script où la variable peut être référencée/utilisée. PHP a trois portées de variables différentes :

- local
- global
- static

Une variable déclarée en dehors d'une fonction a une portée globale et n'est accessible qu'en dehors d'une fonction :

```php
<?php
$x = 5; // variable globale

function myTest() {
    // l'utilisation de x dans cette fonction générera une erreur
    echo "<p>La variable x à l'intérieur de la fonction est : $x</p>";
}
myTest();

echo "<p>La variable x en dehors de la fonction est : $x</p>";
?>
```

Une variable déclarée dans une fonction a une PORTÉE LOCALE et n'est accessible qu'au sein de cette fonction :

```php
<?php
function myTest() {
    $x = 5; // variable locale
    echo "<p>La variable x à l'intérieur de la fonction est : $x</p>";
}
myTest();

// l'utilisation de x en dehors de la fonction générera une erreur
echo "<p>La variable x en dehors de la fonction est : $x</p>";
?>
```

Remarque : On peut avoir des variables locales portant le même nom dans différentes fonctions, car les variables locales ne sont reconnues que par la fonction dans laquelle elles sont déclarées.

Le mot-clé global

Le mot-clé global est utilisé pour accéder à une variable globale à partir d'une fonction. Pour ce faire, on utilise le mot-clé global avant les variables (à l'intérieur de la fonction) :

```php
<?php
$x = 5;
$y = 10;

function myTest() {
    global $x, $y;
    $y = $x + $y;
}

myTest();
echo $y; // Sortie 15
?>
```

PHP stocke également toutes les variables globales dans un tableau appelé $GLOBALS[index]. L'index contient le nom de la variable. Ce tableau est également accessible depuis les fonctions et peut être utilisé pour mettre à jour directement les variables globales. L'exemple ci-dessus peut être réécrit comme ceci :

```php
<?php
$x = 5;
$y = 10;

function myTest() {
    $GLOBALS['y'] = $GLOBALS['x'] + $GLOBALS['y'];
}

myTest();
echo $y; // Sortie 15
?>
```

Le mot-clé statique

Normalement, lorsqu'une fonction est terminée/exécutée, toutes ses variables sont supprimées. Cependant, on veut parfois qu'une variable locale ne soit PAS supprimée. On en a besoin pour un autre travail. Pour cela, on utilise le mot-clé static lors de la première déclaration de la variable :

```php
<?php
function myTest() {
    static $x = 0;
    echo $x;
    $x++;
}

myTest();
myTest();
myTest();
?>
```

Ensuite, chaque fois que la fonction est appelée, cette variable contiendra toujours les informations qu'elle contenait depuis le dernier appel de la fonction.

Remarque : La variable est toujours locale à la fonction.

echo et print

Il existe deux méthodes de base pour obtenir une sortie : echo et print.
echo et print sont plus ou moins les mêmes. Ils sont tous deux utilisés pour afficher des données à l'écran. Les différences sont minimes : echo n'a pas de valeur de retour tandis que print a une valeur de retour de 1, il peut donc être utilisé dans des expressions. echo peut prendre plusieurs paramètres (bien qu'une telle utilisation soit rare) tandis que print peut prendre un argument. echo est légèrement plus rapide que print.

L'instruction echo

L'instruction echo peut être utilisée avec ou sans parenthèses : echo ou echo().

Afficher le texte
L'exemple suivant montre comment générer du texte avec la commande echo (le texte peut contenir du balisage HTML) :

```
<?php
echo "<h2>PHP</h2>";
echo "Charles Carnus<br>";
echo "C'est aussi un texte<br>";
echo "Cette", "chaîne", "a été", "faite", "avec plusieurs paramètres".;
?>
```

Affichage des variables
L'exemple suivant montre comment générer du texte et des variables avec l'instruction echo :

```
<?php
$txt1 = "PHP";
$txt2 = "Informatique et réseaux";
$x = 5;
$y = 4;

echo "<h2>" . $txt1 . "</h2>";
echo "Etudier le PHP en " . $txt2 . "<br>";
echo $x + $y;
?>
```

L'instruction print

L'instruction print peut être utilisée avec ou sans parenthèses : print ou print().

Afficher le texte

L'exemple suivant montre comment générer du texte avec la commande d'impression (le texte peut contenir du balisage HTML) :

```php
<?php
print "<h2>PHP</h2>";
print "Bonjour<br>";
print "C'est parfait!";
?>
```

Affichage des variables

L'exemple suivant montre comment générer du texte et des variables avec l'instruction print :

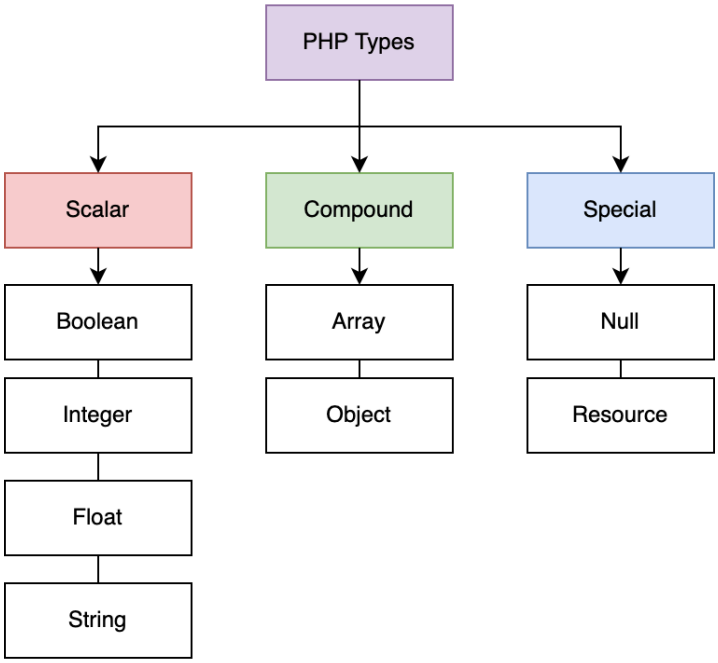
```php
<?php
$txt1 = "PHP";
$txt2 = "Informatique";
$x = 5;
$y = 4;

print "<h2>" . $txt1 . "</h2>";
print "PHP en " . $txt2 . "<br>";
print $x + $y;
?>
```

Types de données

Les variables peuvent stocker des données de différents types, et différents types de données peuvent faire différentes choses. PHP prend en charge les types de données suivants :

- String (Chaîne)
- Integer (Entier)
- Float (nombres à virgule flottante - également appelés doubles)
- Boolean (booléen)
- Array (tableau)
- Object (objet)
- NULL (nul)
- Resource (Ressource)



String (Chaîne)

Une chaîne est une séquence de caractères, comme "Hello world!". Une chaîne peut être n'importe quel texte entre guillemets. On peut utiliser des guillemets simples ou doubles :

```php
<?php
$x = "Hello world!";
$y = 'Hello world!';

echo $x;
echo "<br>";
echo $y;
?>
```

Integer (entier)

Un type de données entier est un nombre non décimal compris entre -2 147 483 648 et 2 147 483 647.

Règles pour les entiers :

- Un entier doit avoir au moins un chiffre
- Un entier ne doit pas avoir de point décimal
- Un entier peut être positif ou négatif
- Les nombres entiers peuvent être spécifiés en notation décimale (base 10), hexadécimale (base 16), octale (base 8) ou binaire (base 2)

Dans l'exemple suivant, $x est un entier. La fonction PHP var_dump() renvoie le type de données et la valeur :

```php
<?php
$x = 2023;
var_dump($x);
?>
```

Float

Un flottant (nombre à virgule flottante) est un nombre avec un point décimal ou un nombre sous forme exponentielle.
Dans l'exemple suivant, $x est un flottant. La fonction PHP var_dump() renvoie le type de données et la valeur :

```php
<?php
$x = 12.406;
var_dump($x);
?>
```

Exemples :

```php
<?php
$x = 2023;
var_dump(is_numeric($x));

$x = "2023";
var_dump(is_numeric($x));

$x = "20.23" + 10;
var_dump(is_numeric($x));

$x = "Bonjour";
var_dump(is_numeric($x));
?>
```

```php
<?php
// Cast float to int
$x = 123465.789;
$int_cast = (int)$x;
echo $int_cast;

echo "<br>";

// Cast string to int
$x = "123465.789";
$int_cast = (int)$x;
echo $int_cast;
?>
```

Boolean

Un booléen représente deux états possibles : TRUE (VRAI) ou FALSE (FAUX).

```php
$x = true;
$y = false;
```

Les booléens sont souvent utilisés dans les tests conditionnels.

Array (tableau)

Un tableau stocke plusieurs valeurs dans une seule variable. Dans l'exemple suivant, $matieres est un tableau. La fonction PHP var_dump() renvoie le type de données et la valeur :

```php
<?php
$matieres = array("Informatique","ESLA","Physique");
var_dump($matieres);
?>
```

Object

Une classe est un modèle pour les objets et un objet est une instance d'une classe. Lorsque les objets individuels sont créés, ils héritent de toutes les propriétés et comportements de la classe, mais chaque objet aura des valeurs différentes pour les propriétés.
On suppose qu'on a une classe nommée Ordinateur. Un ordinateur peut avoir des propriétés comme marque, modèle, processeur, couleur … On peut définir des variables comme $marque, $processeur …, pour contenir les valeurs de ces propriétés. Lorsque les objets individuels (Apple, Dell, HP, Asus …) sont créés, ils héritent de toutes les propriétés et comportements de la classe, mais chaque objet aura des valeurs différentes pour les propriétés. Si on crée une fonction __construct(), PHP appellera automatiquement cette fonction lorsqu'on crée un objet à partir d'une classe.

```php
<?php
class Ordinateur {
    public $processeur;
    public $marque;
    public function __construct($processeur, $marque) {
        $this->processeur = $processeur;
        $this->marque = $marque;
    }
    public function message() {
        return " Mon ordinateur est de marque " . $this->marque . " " .
$this->processeur . "!";
    }
}

$monOrdinateur = new Ordinateur("M2", "Apple");
echo $monOrdinateur -> message();
echo "<br>";
$monOrdinateur = new Ordinateur("i7", "HP");
echo $monOrdinateur -> message();
?>
```

NULL (Valeur NULLE)

Null est un type de données spécial qui ne peut avoir qu'une seule valeur : NULL.
Une variable de type de données NULL est une variable à laquelle aucune valeur n'est affectée.

Conseil : Si une variable est créée sans valeur, la valeur NULL lui est automatiquement affectée.
Les variables peuvent également être vidées en définissant la valeur sur NULL :

```php
<?php
$x = "Bonjour";
$x = null;
var_dump($x);
?>
```

Resource (Ressource)

Le type de ressource spécial n'est pas un type de données réel. C'est le stockage d'une référence à des fonctions et ressources externes à PHP.
Un exemple courant d'utilisation du type de données de ressource est un appel de base de données.

Manipulation des chaînes (string)

strlen() - Renvoie la longueur d'une chaîne
str_word_count() - Compte les mots dans une chaîne
strrev() - Inverse une chaîne
strpos() - Rechercher un texte dans une chaîne
str_replace() - Remplace le texte dans une chaîne

```php
<?php
echo strlen("Hello world!"); // sortie : 12
echo str_word_count("Hello world!"); // sortie : 2
echo strrev("Hello world!"); // sortie : !dlrow olleH
echo strpos("Hello world!", "world"); // sortie : 6
echo str_replace("world", "Carnus", "Hello world!"); // sortie Hello
Carnus!
?>
```

Fonctions mathématiques

```php
<?php
echo(pi()); // Sortie 3.1415926535898
echo(min(0, 150, 30, 20, -8, -200)); // Sortie -200
echo(max(0, 150, 30, 20, -8, -200)); // Sortie 150
echo(abs(-6.7)); // Sortie 6.7
echo(sqrt(64)); // Sortie 8
echo(round(0.60)); // Sortie 1
echo(round(0.49)); // Sortie 0
echo(rand());
echo(rand(10, 100));

// define(name, value, case-insensitive)
define("MAVARIABLE", "Charles Carnus");
echo GREETING;
define("GREETINGS", "Charles Carnus1", true);
echo greetings;
define("ordinateurs", [
    "Apple",
    "HP",
    "DELL"
]);
echo ordinateurs[0];
?>
```

Opérateurs arithmétiques

Opérateur	Nom	Syntaxe
+	Addition	$x + $y
-	Subtraction	$x - $y
*	Multiplication	$x * $y
/	Division	$x / $y
%	Modulo	$x % $y
**	Exponentiel	$x ** $y

Opérateurs d'affectation

L'opérateur d'affectation de base en PHP est "=". Cela signifie que l'opérande de gauche est défini sur la valeur de l'expression d'affectation à droite.

Affectation	Equivalent à ...
x = y	x = y
x += y	x = x + y
x -= y	x = x - y
x *= y	x = x * y
x /= y	x = x / y
x %= y	x = x % y

Opérateurs de comparaison

Opérateur	Nom	Exemple	Résultat
==	Egal	$x == $y	true si $x est égal à $y
===	Identique	$x === $y	true si $x est égal à $y, et sont du même type
!=	Différent	$x != $y	true si $x est différent de $y
<>	Différent	$x <> $y	true si $x est différent de $y
!==	Non identique	$x !== $y	true si $x est différent de $y, ou ne sont pas du même type
>	Supérieur	$x > $y	true si $x est supérieur à $y
<	Inférieur	$x < $y	true si $x est inférieur à $y
>=	Supérieur ou égal	$x >= $y	true if $x est supérieur ou égal à $y
<=	Inférieur ou égal	$x <= $y	true si $x est inférieur ou égal à $y
<=>	Spaceship	$x <=> $y	Retourne un entier inférieur, égal ou supérieur à zéro, selon que $x est inférieur, égal ou supérieur à $y.

Opérateurs logiques

Opérateur	Nom	Exemple	Résultat
and	And	$x and $y	true si $x et $y sont vrais (true)
or	Or	$x or $y	true si $x ou $y est vrai (true)
xor	Xor	$x xor $y	true si $x ou $y est vrai (true), mais pas les deux
&&	And	$x && $y	true si $x et $y sont vrais (true)
||	Or	$x || $y	true si $x ou $y est vrai (true)
!	Not	!$x	true si $x est faux (false)

Opérateurs de chaîne

Opérateur	Nom	Exemple	Résultat
.	Concaténation	$txt1 . $txt2	Concaténation de $txt1 et $txt2
.=	Affectation avec concaténation	$txt1 .= $txt2	Ajoute $txt2 à $txt1

Conditions if...else...elseif

```php
<?php
$t = date("H");

if ($t < "18") {
    echo "Bonjour!";
}
?>
```

```php
<?php
$t = date("H");

if ($t < "18") {
    echo "Bonjour!";
} else {
    echo "Bonsoir!";
}
?>
```

```php
<?php
$t = date("H");

if ($t < "10") {
    echo "Bonjour!";
} elseif ($t < "14") {
    echo "Bonne journée!";
} else {
    echo "Bonsoir!";
}
?>
```

Switch

```php
<?php
$favcolor = "red";

switch ($favcolor) {
    case "red":
        echo "Your favorite color is red!";
        break;
    case "blue":
        echo "Your favorite color is blue!";
        break;
    case "green":
        echo "Your favorite color is green!";
        break;
    default:
        echo "Your favorite color is neither red, blue, nor green!";
}
?>
```

Les boucles

While

```php
<?php
$x = 1;

while($x <= 5) {
    echo "The number is: $x <br>";
    $x++;
}
?>
```

do...while

```php
<?php
$x = 1;

do {
    echo "The number is: $x <br>";
    $x++;
} while ($x <= 5);
?>
```

For

```php
<?php
for ($x = 0; $x <= 10; $x++) {
    echo "The number is: $x <br>";
}
?>
```

foreach

La boucle foreach parcourt un bloc de code pour chaque élément d'un tableau.
La boucle foreach ne fonctionne que sur les tableaux et est utilisée pour parcourir chaque paire clé/valeur d'un tableau.

```
foreach ($array as $value) {
    code à exécuter;
    }
```

```
<?php
$colors = array("red", "green", "blue", "yellow");

foreach ($colors as $value) {
    echo "$value <br>";
}
?>
```

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");

foreach($age as $x => $val) {
    echo "$x = $val<br>";
}
?>
```

Break

L'instruction break peut également être utilisée pour sortir d'une boucle. Cet exemple sort de la boucle lorsque x est égal à 4 :

```
<?php
for ($x = 0; $x < 10; $x++) {
    if ($x == 4) {
        break;
    }
    echo "The number is: $x <br>";
}
?>
```

Continue

L'instruction continue interrompt une itération (dans la boucle), si une condition spécifiée se produit, et continue avec l'itération suivante dans la boucle.
Cet exemple ignore la valeur de 4 :

```php
<?php
for ($x = 0; $x < 10; $x++) {
    if ($x == 4) {
        continue;
    }
    echo "The number is: $x <br>";
}
?>
```

Break et Continue dans une boucle While

```php
<?php
$x = 0;

while($x < 10) {
    if ($x == 4) {
        break;
    }
    echo "The number is: $x <br>";
    $x++;
}
?>
```

```php
<?php
$x = 0;

while($x < 10) {
    if ($x == 4) {
        $x++;
        continue;
    }
    echo "The number is: $x <br>";
    $x++;
}
?>
```

Les fonctions

```
function functionName() {
    code à exécuter;
}
```

Remarque : Un nom de fonction doit commencer par une lettre ou un trait de soulignement. Les noms de fonction ne sont PAS sensibles à la casse.

Dans l'exemple ci-dessous, on crée une fonction nommée "ecrireMsg()". L'accolade ouvrante ({) indique le début du code de la fonction et l'accolade fermante (}) indique la fin de la fonction. La fonction affiche "Bonjour!". Pour appeler la fonction, on écrit simplement son nom suivi de parenthèses () :

```
<?php
function ecrireMsg() {
    echo "Bonjour!";
}

ecrireMsg(); // Appel de la fonction
?>
```

Les arguments d'une fonction

Les informations peuvent être transmises aux fonctions via des arguments.
Les arguments sont spécifiés après le nom de la fonction, entre parenthèses. On peut ajouter autant d'arguments qu'on le souhaite, séparés par une virgule.
L'exemple suivant a une fonction avec un argument ($prenom). Lorsque la fonction fPrenom() est appelée, on transmet également un prénom (par exemple Charles), et le nom est utilisé à l'intérieur de la fonction, qui affiche plusieurs prénoms différents, mais un nom de famille égal :

```
<?php
function fPrenom($prenom) {
    echo "$prenom Carnus.<br>";
}

fPrenom("Charles");
fPrenom("François");
fPrenom("Julia");
?>
```

```
<?php
function fPrenom($prenom, $annee) {
    echo "$prenom Carnus. Né(e) en $annee <br>";
}

fPrenom("Charles", "1850");
fPrenom("François", "1985");
fPrenom("Julia", "2003");
?>
```

Valeur d'arguments par défaut

L'exemple suivant montre comment utiliser un paramètre par défaut. Si on appelle la fonction definirNote() sans arguments, elle prend la valeur par défaut comme argument :

```php
<?php declare(strict_types=1); // exigence stricte
function definirNote(int $minnote = 50) {
    echo "La note est : $minnote <br>";
}

definirNote(10);
definirNote(); // utilisera la valeur par défaut de 50
definirNote(35);
?>
```

Valeurs renvoyées par les fonctions

Pour laisser une fonction renvoyer une valeur, on utilise l'instruction `return` :

```php
<?php declare(strict_types=1); // exigence stricte
function somme(int $x, int $y) {
    $z = $x + $y;
    return $z;
}

echo "5 + 12 = " . somme(5, 12) . "<br>";
echo "7 + 15 = " . somme(7, 15) . "<br>";
echo "20 + 4 = " . somme(20, 4);
?>
```

Passer des arguments par référence

Les arguments sont généralement passés par valeur, ce qui signifie qu'une copie de la valeur est utilisée dans la fonction et que la variable qui a été passée dans la fonction ne peut pas être modifiée. Lorsqu'un argument de fonction est passé par référence, les modifications apportées à l'argument modifient également la variable qui a été passée. Pour transformer un argument de fonction en référence, l'opérateur & est utilisé :

```php
<?php
function plus_dix(&$valeur) {
    $valeur += 10;
}

$nombre = 5;
plus_dix($nombre);
echo $nombre;
?>
```

Les tableaux

Un tableau stocke plusieurs valeurs dans une seule variable :

```php
<?php
$ordinateur = array("HP", "Apple", "Dell");
echo "Marques " . $ordinateur[0] . ", " . $ordinateur[1] . " and " .
$ordinateur[2] . ".";
?>
```

```php
<?php
$ordinateur = array("HP", "Apple", "Dell");
echo count($ordinateur);
?>
```

```php
<?php
$ordinateur = array("HP", "Apple", "Dell");
$tailleTab = count($ordinateur);

for($x = 0; $x < $tailleTab; $x++) {
    echo $ordinateur[$x];
    echo "<br>";
    }
?>
```

```php
<?php
for ($row = 0; $row < 4; $row++) {
    echo "<p><b>Numéro de ligne $row</b></p>";
    echo "<ul>";
    for ($col = 0; $col < 3; $col++) {
        echo "<li>".$ordinateur[$row][$col]."</li>";
    }
    echo "</ul>";
}
?>
```

Organisation du répertoire de travail et inclusion des fichiers

Exemple

```
.
├── index.php
├── functions.php
├── inc
│   ├── footer.php
│   └── header.php
└── public
    ├── css
    │   └── style.css
    └── js
        └── app.js
```

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-
scale=1.0" />
        <link rel="stylesheet" href="public/css/style.css" />
        <title>PHP include Example</title>
    </head>
    <body>
        <h1>PHP include</h1>
        <p>This shows how the PHP include construct works.</p>

        <script src="public/js/app.js"></script>
    </body>
</html>
```

Exemples

Exemple d'application avec un formulaire (un seul fichier)

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="utf-8">
</head>

<body style="background-color:#d0ece7;">

<h1 style="text-align:center;">Ma page Web écrite en PHP</h1>

<div style="background-color:#b80b28;color:white;padding:20px;">
  <h2>BTS SN--IR, cours PHP.</h2>
  <p>Lycée Charles Carnus.</p>
  <p>Rodez</p>
</div>


<form method="post">
 <p>Votre nom : <input type="text" name="nom" /></p>
 <p>Numéro de votre département : <input type="text" name="dep" /></p>
 <p><input type="submit" value="Valider"></p>
</form>

Bonjour, <?php echo $_POST['nom']; ?>.
Votre département est <?php echo (int)$_POST['dep']; ?>.

</body>
</html>
```

Exemple d'application avec un formulaire (2 fichiers)

Index.php

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="utf-8">
</head>

<body style="background-color:#d0ece7;">

<h1 style="text-align:center;">Ma page Web écrite en PHP</h1>

<div style="background-color:#b80b28;color:white;padding:20px;">
  <h2>BTS SN--IR, cours PHP.</h2>
  <p>Lycée Charles Carnus.</p>
  <p>Rodez</p>
</div>


<form action="action.php" method="post">
 <p>Votre nom : <input type="text" name="nom" /></p>
 <p>Numéro de votre département : <input type="text" name="dep" /></p>
 <p><input type="submit" value="Valider"></p>
</form>

</body>
</html>
```

action.php

```
Bonjour, <?php echo htmlspecialchars($_POST['nom']); ?>.
Votre département est <?php echo (int)$_POST['dep']; ?>.
```

Exemple d'application avec un formulaire de calcul

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="utf-8">
</head>
<body style="background-color:#d0ece7;">

<h1 style="text-align:center;">Ma page Web écrite en PHP</h1>

<div style="background-color:#b80b28;color:white;padding:20px;">
  <h2>BTS SN--IR, cours PHP.</h2>
  <p>Lycée Charles Carnus.</p>
  <p>Rodez</p>
</div>

<!-- Form -->
<form method="get">
    <div style="color:red;padding:10px;">
        <label for="nombre1">Nombre 1 :</label>
        <input type="number" id="nombre1" name="nb1">
    </div>
    <div style="color:red;padding:10px;">
        <label for="nombre2">Nombre 2 :</label>
        <input type="number" id="nombre2" name="nb2">
    </div>
    <button type="submit" style="color:darkblue;padding:5px;">Calculer</
button>
</form>

<!-- PHP -->
<?php

$total = $_GET["nb1"]+ $_GET["nb2"];
echo "<p>Résultat du calcul est : $total</p>";
echo " <hr style=height:2px;color:gray;background-color:gray> ";
echo "Lycée Charles Carnus";

$txt1 = "PHP";
$txt2 = "Informatique et réseaux";
$x = 5;
$y = 4;

echo "<h2>" . $txt1 . "</h2>";
echo "Etudier le PHP en " . $txt2 . "<br>";
echo $x + $y . "<br>";
var_dump($x);

?>
</body>
</html>
```

Validation de formulaire

Dans cette partie, on abordera la validation de formulaire PHP, comment valider les données de formulaire et comment afficher des messages d'erreur si les entrées de l'utilisateur ne sont pas valides.

Lors du traitement d'un formulaire, il est essentiel de valider les entrées de l'utilisateur pour s'assurer que les données sont dans un format valide. Il existe deux types de validations; côté client et côté serveur :

- La validation côté client est effectuée dans les navigateurs Web des utilisateurs. Pour valider les données côté client, on peut utiliser la validation HTML5 ou JavaScript. La validation côté client vise à aider les utilisateurs légitimes à entrer des données dans le format valide avant de les soumettre au serveur. Cependant, la validation côté client n'empêche pas les utilisateurs malveillants de soumettre des données susceptibles d'exploiter l'application.
- La validation côté serveur valide les données sur le serveur Web à l'aide de PHP. Pour valider les données en PHP, on peut utiliser les fonctions filter_var() et filter_input().

Exemple de validation de formulaire

On va créer un formulaire d'abonnement par e-mail qui comprend une fonction de validation. Le formulaire contient les éléments d'entrée de nom et d'e-mail et un bouton d'envoi :

Si on entre pas le nom et/ou l'e-mail et on clique sur le bouton d'inscription, le formulaire affichera les messages d'erreur. De plus, si on entre une adresse e-mail invalide, le formulaire affichera un message d'erreur différent.
A noter qu'on n'utilise pas la validation côté client pour ce formulaire afin de faciliter le test. En pratique, on doit également utiliser la validation côté client.

```
.
├── css
│   └── style.css
├── inc
│   ├── get.php
│   ├── post.php
│   ├── header.php
│   ├── footer.php
│   └── .htaccess
└── index.php
```

Fichier	Répertoire	Description
index.php	.	Code principal du formulaire
header.php	inc	Code d'en-tête
footer.php	inc	Code de pied de page
get.php	inc	Formulaire d'inscription par e-mail
post.php	inc	Code pour gérer la soumission du formulaire
style.css	css	Code CSS

header.php

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <link rel="stylesheet" href="css/style.css">
    <title>Subscribe</title>
</head>
<body>
    <main>
```

footer.php

```
    </main>
    </body>
    </html>
```

index.php

```
<?php

require __DIR__ . '/inc/header.php';

$errors = [];
$inputs = [];

$request_method = strtoupper($_SERVER['REQUEST_METHOD']);

if ($request_method === 'GET') {
    // show the form
    require __DIR__ . '/inc/get.php';
} elseif ($request_method === 'POST') {
    // handle the form submission
    require    __DIR__ .  '/inc/post.php';
    // show the form if the error exists
    if (count($errors) > 0) {
        require __DIR__ . '/inc/get.php';
    }
}

require __DIR__ .  '/inc/footer.php';
```

Tout d'abord, on charge le code des fichiers header.php et footer.php en utilisant l'instruction require en haut et en bas du fichier pour générer l'en-tête et le pied de page.
Puis, on définit le tableau $errors pour stocker les messages d'erreur et le tableau $inputs pour stocker les valeurs de formulaire saisies. Si un élément d'entrée contient des données non valides, le fichier index.php affichera la valeur saisie stockée dans les $inputs.

Après, on affiche le formulaire si la méthode de requête HTTP est GET en chargeant le fichier get.php.
Enfin, on charge le code dans le post.php pour gérer la soumission du formulaire si la méthode de requête HTTP est POST. Si le formulaire contient des erreurs, les $errors ne seront pas vides. Dans ce cas, on affiche à nouveau le formulaire avec les messages d'erreur stockés dans le tableau $errors et les valeurs saisies stockées dans les tableaux $inputs.

get.php

```
<form action="<?php echo htmlspecialchars($_SERVER['PHP_SELF']) ?>"
method="post">
    <header>
        <h1>Get FREE Updates</h1>
        <p>Join us for FREE to get email updates!</p>
    </header>
    <div>
        <label for="name">Name:</label>
        <input type="text" name="name" id="name" placeholder="Full Name"
value="<?php echo $inputs['name'] ?? '' ?>" class="<?php echo
isset($errors['name']) ? 'error' : ''  ?>">
        <small><?php echo $errors['name'] ?? '' ?></small>
    </div>
    <div>
        <label for="name">Email:</label>
        <input type="text" name="email" id="email" placeholder="Email
Address" value="<?php echo $inputs['email'] ?? '' ?>" class="<?php echo
isset($errors['email']) ? 'error' : '' ?>">
        <small><?php echo $errors['email'] ?? '' ?></small>
    </div>
    <button type="submit">Subscribe</button>
</form>
```

On commence par remplir les éléments d'entrée name et email avec les valeurs saisies stockées dans le tableau $inputs uniquement si ces valeurs existent.
Puis, on affiche les messages d'erreur stockés dans le tableau $errors s'ils existent.

post.php

Le post.php valide les données du formulaire à l'aide des fonctions filter_input() et filter_var().

```
<?php

const NAME_REQUIRED = 'Please enter your name';
const EMAIL_REQUIRED = 'Please enter your email';
const EMAIL_INVALID = 'Please enter a valid email';

// sanitize and validate name
$name = filter_input(INPUT_POST, 'name', FILTER_SANITIZE_STRING);
$inputs['name'] = $name;

if ($name) {
    $name = trim($name);
    if ($name === '') {
        $errors['name'] = NAME_REQUIRED;
    }
} else {
    $errors['name'] = NAME_REQUIRED;
}

// sanitize & validate email
$email = filter_input(INPUT_POST, 'email', FILTER_SANITIZE_EMAIL);
$inputs['email'] = $email;
if ($email) {
    // validate email
    $email = filter_var($email, FILTER_VALIDATE_EMAIL);
    if ($email === false) {
        $errors['email'] = EMAIL_INVALID;
    }
} else {
    $errors['email'] = EMAIL_REQUIRED;
}
?>

<?php if (count($errors) === 0) : ?>
    <section>
        <h2>
            Thanks <?php echo htmlspecialchars($name) ?> for your
subscription!
        </h2>
        <p>Please follow the steps below to complete your
subscription:</p>
        <ol>
            <li>Check your email (<?php echo htmlspecialchars($email) ?
>) - Find the message sent from webmaster@phptutorial.net</li>
            <li>Click to confirm - Click on the link in the email to
confirm your subscription.</li>
        </ol>
    </section>

<?php endif ?>
```

Formulaire de contact

Créer un formulaire de contact en PHP qui inclut la validation du formulaire, l'envoi d'e-mails, etc.

Introduction au formulaire de contact

Un formulaire de contact permet aux visiteurs d'un site web de laisser des messages. En règle générale, un formulaire de contact contient les champs de saisie Nom, E-mail, Objet et Message.

Les visiteurs doivent remplir ces champs et cliquer sur le bouton soumettre (ou envoyer) pour envoyer un message. En PHP, on peut valider les données du formulaire et envoyer le message saisi à une adresse e-mail prévue.
Le formulaire de contact est une cible pour les spammeurs qui utilisent des spambots pour envoyer des messages non sollicités à des fins publicitaires, de phishing, de diffusion de logiciels malveillants, etc.

Un spambot est un logiciel qui automatise les activités de spam comme remplir et soumettre automatiquement des formulaires de contact. Pour créer un formulaire de contact sans spam, on peut y ajouter un captcha. Cependant, parfois, les captchas sont impossibles à lire. En conséquence, ils créent une expérience terrible pour les utilisateurs légitimes.

Pour éviter d'utiliser un captcha tout en protégeant le contact du spam, on peut utiliser un honeypot (pot de miel) pour tromper les spambots.
Un honeypot est un champ du formulaire que le visiteur ne peut pas voir, mais que les spambots peuvent voir. Lorsqu'un spambot voit le champ honeypot, il remplit le champ avec une valeur. En PHP, on peut vérifier si le honeypot a une valeur et ignorer l'envoi du message. Pour créer un honeypot, on doit disposer d'une classe CSS qui masque complètement le champ du honeypot comme suit :

```
.user-cannot-see {
    display:none
}
```

Et on aura un champ honeypot sur le formulaire :

```
<label for="nickname" aria-hidden="true" class="user-cannot-see">
Nickname
    <input type="text"
           name="nickname"
           id="nickname"
           class="user-cannot-see"
           autocomplete="off"
           tabindex="-1">
</label>
```

On note que le nom du honeypot doit avoir l'air légitime. Récemment, le spambot est devenu plus intelligent et peut détecter le honeypot. Pour faire face à ces robots spammeurs intelligents, on a besoin d'un honeypot intelligent. Par exemple, un honeypot intelligent peut avoir un nom différent pour chaque demande. De plus, son emplacement sur le formulaire est modifié de manière aléatoire.
Le formulaire de contact a les fonctionnalités suivantes :

- Validation du formulaire
- Envoi de message par e-mail
- Eviter les SPAM
- Empêcher la double soumission

Tout d'abord, créer les dossiers et fichiers suivants :

```
├── config
│   └── app.php
├── css
│   └── style.css
├── inc
│   ├── footer.php
│   ├── get.php
│   ├── header.php
│   ├── mail.php
│   └── post.php
└── index.php
```

header.php

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <link rel="stylesheet" href="css/style.css">
    <title>PHP Contact Form</title>
</head>
<body>
    <main>
```

footer.php

```
    </main>
</body>
</html>
```

get.php

```
<?php if (isset($message)) : ?>
    <div class="alert alert-success">
        <?= $message ?>
    </div>
<?php endif ?>

<form action="index.php" method="post">
    <header>
        <h1>Send Us a Message</h1>
    </header>

    <div>
        <label for="name">Name:</label>
        <input type="text" value="<?= $inputs['name'] ?? '' ?>"
name="name" id="name" placeholder="Full name">
        <small><?= $errors['name'] ?? '' ?></small>
    </div>

    <div>
        <label for="email">Email:</label>
        <input type="email" name="email" id="email" value="<?=
$inputs['email'] ?? '' ?>" placeholder="Email address">
        <small><?= $errors['email'] ?? '' ?></small>
    </div>

    <div>
        <label for="subject">Subject:</label>
        <input type="subject" name="subject" id="subject" value="<?=
$inputs['subject'] ?? '' ?>" placeholder="Enter a subject">
        <small><?= $errors['subject'] ?? '' ?></small>
    </div>

    <div>
        <label for="message">Message:</label>
        <textarea id="message" name="message" rows="5"><?=
$inputs['message'] ?? '' ?></textarea>
        <small><?= $errors['message'] ?? '' ?></small>
    </div>

    <label for="nickname" aria-hidden="true" class="user-cannot-see">
Nickname
        <input type="text" name="nickname" id="nickname" class="user-
cannot-see" tabindex="-1" autocomplete="off">
    </label>

    <button type="submit">Send Message</button>
</form>
```

post.php

```php
<?php

// check the honeypot
$honeypot = filter_input(INPUT_POST, 'nickname',
FILTER_SANITIZE_STRING);
if ($honeypot) {
    header($_SERVER['SERVER_PROTOCOL'] . ' 405 Method Not Allowed');
    exit;
}

// validate name
$name = filter_input(INPUT_POST, 'name', FILTER_SANITIZE_STRING);
$inputs['name'] = $name;
if (!$name || trim($name) === '') {
    $errors['name'] = 'Please enter your name';
}

// validate email
$email = filter_input(INPUT_POST, 'email', FILTER_SANITIZE_EMAIL);
$inputs['email'] = $email;
if ($email) {
    $email = filter_var($email, FILTER_SANITIZE_EMAIL);
    if (!$email) {
        $errors['email'] = 'Please enter a valid email';
    }
} else {
    $errors['email'] = 'Please enter an email';
}

// validate subject
$subject = filter_input(INPUT_POST, 'subject', FILTER_SANITIZE_STRING);
$inputs['subject'] = $subject;
if (!$subject || trim($subject) === '') {
    $errors['subject'] = 'Please enter the subject';
}

// validate message
$message = filter_input(INPUT_POST, 'message', FILTER_SANITIZE_STRING);
$inputs['message'] = $message;
if (!$message || trim($message) === '') {
    $errors['message'] = 'Please enter the message';
}
```

Le post.php vérifie le honeypot et renvoie le code d'état HTTP 405 s'il détecte le spambot. Sinon, il nettoie et valide les champs de saisie, y compris le nom, l'e-mail, l'objet et le message.

config.php

Le fichier config.php stocke les informations de configuration, par exemple l'adresse e-mail du destinataire :

```
<?php

return [
    'mail' => [
        'to_email' => ‘webmaster@carnus.fr'
    ]
];
```

mail.php

Le mail.php obtient l'adresse e-mail du destinataire à partir du fichier de configuration app.php. Il envoie au message saisi dans le formulaire de contact à l'aide de la fonction mail() :

```
<?php

// get email from the config file
$config = require_once __DIR__ . '/../config/app.php';
$recipient_email = $config['mail']['to_email'];

// contact information
$contact_name = $inputs['name'];
$contact_email = $inputs['email'];
$message = $inputs['message'];
$subject = $inputs['subject'];

// Email header
$headers[] = 'MIME-Version: 1.0';
$headers[] = 'Content-type: text/html; charset=utf-8';
$headers[] = "To: $recipient_email";
$headers[] = "From: $contact_email";
$header = implode('\r\n', $headers);

mail($recipient_email, $subject, $message, $header);
```

index.php

```php
<?php

session_start();

$errors = [];
$inputs = [];

$request_method = strtoupper($_SERVER['REQUEST_METHOD']);

if ($request_method === 'GET') {

    // show the message
    if (isset($_SESSION['message'])) {
        $message = $_SESSION['message'];
        unset($_SESSION['message']);
    } elseif (isset($_SESSION['inputs']) && isset($_SESSION['errors']))
{
        $errors = $_SESSION['errors'];
        unset($_SESSION['errors']);
        $inputs = $_SESSION['inputs'];
        unset($_SESSION['inputs']);
    }
    // show the form
    require_once __DIR__ . '/inc/get.php';
} elseif ($request_method === 'POST') {
    // check the honeypot and validate the field
    require_once __DIR__ . '/inc/post.php';

    if (!$errors) {
        // send an email
        require_once __DIR__ . '/inc/mail.php';
        // set the message
        $_SESSION['message'] =  'Thanks for contacting us! We will be in
touch with you shortly.';
    } else {
        $_SESSION['errors'] =   $errors;
        $_SESSION['inputs'] =   $inputs;
    }

    header('Location: index.php', true, 303);
    exit;
}
```

Tout d'abord, afficher le formulaire de contact si la méthode de requête HTTP est GET. Obtenir également les données $message, $errors et $inputs de la $_SESSION.
Puis, gérer la soumission du formulaire si la méthode de requête HTTP est POST, envoyer un e-mail s'il n'y a pas d'erreur et rediriger vers le formulaire de contact. Noter qu'on utilise la technique PRG (Post-Redirect-Get) pour éviter le problème de double soumission.