Guía Completa de PHP

November 25, 2025

Contents

1 Sintaxis y fundamentos del lenguaje

1.1 Sintaxis básica de PHP

El lenguaje PHP se caracteriza por su sintaxis sencilla y su capacidad de integrarse fácilmente con HTML. Todo código PHP debe estar encerrado entre las etiquetas <?php y ?>, lo que permite combinar lógica y presentación en un solo archivo.

```php
<?php
echo "Hola, mundo!";
?>
```

Listing 1: Ejemplo básico de PHP

1.2 Variables

Las variables en PHP comienzan siempre con el signo $ y no requieren declaración explícita de tipo, lo que facilita su uso y flexibilidad.

```php
$nombre = "Juan";
$edad = 25;
```

Listing 2: Declaración de variables

1.3 Tipos de datos

PHP soporta diversos tipos de datos básicos, entre ellos enteros, flotantes, booleanos, cadenas de texto, arrays, objetos y el valor nulo (NULL).

```php
$numero = 10;
$precio = 15.5;
$activo = true;
$texto = "Hola";
$arreglo = [1,2,3];
```

Listing 3: Ejemplo de tipos de datos

1.4 Operadores

Entre los operadores fundamentales están:

- **Aritméticos**: +, -, *, /, %
- **Comparación**: ==, ===, !=, !==, >, <, >=, <=
- **Lógicos**: &&, ||, !

1.5 Condicionales

Permiten la ejecución de bloques de código según condiciones evaluadas.

```php
if ($edad >= 18) {
    echo "Mayor de edad";
} elseif ($edad >= 13) {
    echo "Adolescente";
} else {
    echo "Niño";
}
```

Listing 4: Ejemplo de condicional if-elseif-else

1.6 Switch

El `switch` es útil para evaluar un valor contra múltiples casos posibles.

```php
$color = "rojo";
switch($color){
    case "rojo":
        echo "Color rojo";
        break;
    case "azul":
        echo "Color azul";
        break;
    default:
        echo "Otro color";
}
```

Listing 5: Ejemplo de switch

1.7 Bucles

PHP ofrece varias estructuras para iterar código: `for`, `while` y `foreach`.

```php
for($i=0; $i<5; $i++){
    echo $i;
}

$numero = 0;
while($numero < 5){
    echo $numero;
    $numero++;
}

$frutas = ["manzana", "pera"];
foreach($frutas as $fruta){
    echo $fruta;
}
```

Listing 6: Ejemplos de bucles

2 Arrays y manejo de colecciones

2.1 Arrays en PHP

Los arrays son estructuras esenciales para manejar colecciones de datos.

```php
$numeros = [1,2,3,4,5];
```

Listing 7: Definición de un array simple

2.2 Arrays indexados

Los arrays indexados son listas numeradas, donde se accede a los elementos mediante índices.

```php
$colores = ["rojo", "verde", "azul"];
echo $colores[1]; // verde
```

Listing 8: Array indexado con acceso a elemento

2.3 Arrays asociativos

Permiten asociar claves con valores, facilitando el acceso por identificadores legibles.

```php
$persona = [
    "nombre" => "Ana",
    "edad" => 30
```

```
];
echo $persona["nombre"]; // Ana
```

Listing 9: Array asociativo

2.4 Arrays multidimensionales

Son arrays que contienen otros arrays, útiles para representar estructuras más complejas.

```
$personas = [
    ["nombre" => "Ana", "edad" => 30],
    ["nombre" => "Luis", "edad" => 25]
];
echo $personas[1]["nombre"]; // Luis
```

Listing 10: Array multidimensional

2.5 Funciones de arrays

PHP ofrece diversas funciones para manipular arrays, entre ellas:

Función	Descripción
count()	Devuelve el número de elementos en un array
array_push()	Agrega uno o más elementos al final del array
array_pop()	Elimina y devuelve el último elemento del array
array_merge()	Combina dos o más arrays en uno solo
in_array()	Verifica si un valor existe dentro de un array

```php
$frutas = ["manzana", "pera"];
array_push($frutas, "plátano");
echo count($frutas); // 3
```

Listing 11: Uso de funciones para arrays

3 Funciones y modularidad

3.1 Definición y uso

Las funciones permiten encapsular código para reutilización y mejorar la organización.

```php
function saludar($nombre){
    return "Hola, $nombre";
}
echo saludar("Juan");
```

Listing 12: Definición y llamada a función

3.2 Paso de parámetros

Por defecto los parámetros se pasan por valor, pero pueden pasarse por referencia usando &.

```php
function aumentar(&$numero){
    $numero += 10;
}
$n = 5;
aumentar($n);
echo $n; // 15
```

Listing 13: Parámetros por referencia

3.3 Funciones como argumentos

PHP permite funciones anónimas y pasar funciones como argumentos, facilitando la programación funcional.

```php
function aplicar($func, $valor){
    return $func($valor);
}
echo aplicar(fn($x) => $x*2, 5); // 10
```

Listing 14: Función que recibe otra función

3.4 Funciones definidas por el usuario

Las funciones se definen con la palabra clave `function` y pueden retornar valores con `return`.

3.5 Manejo de errores dentro de funciones

Es recomendable controlar errores o condiciones inesperadas dentro de las funciones para evitar fallos.

```php
function dividir($a, $b){
    if($b == 0) return "Error: División por 0";
    return $a/$b;
}
echo dividir(10,0);
```

Listing 15: Ejemplo de manejo simple de error

4 Programación Orientada a Objetos (POO)

4.1 Clases y objetos

La POO en PHP permite definir clases como plantillas para crear objetos con propiedades y métodos.

```php
class Persona {
    public $nombre;
    public function saludar(){
        return "Hola, $this->nombre";
    }
}
$persona = new Persona();
$persona->nombre = "Ana";
echo $persona->saludar();
```

Listing 16: Definición básica de clase y objeto

4.2 Propiedades y métodos

Las propiedades son variables que describen al objeto; los métodos son funciones que definen su comportamiento.

4.3 Constructores y destructores

Se usan para inicializar objetos al crear instancias y realizar limpieza al destruirlos.

```php
class Persona {
    public function __construct($nombre){
        $this->nombre = $nombre;
    }
    public function __destruct(){
        echo "Objeto destruido";
    }
}
$persona = new Persona("Luis");
```

Listing 17: Ejemplo de constructor y destructor

4.4 Visibilidad

Controla el acceso a propiedades y métodos con `public`, `private` y `protected`.

```php
class Test {
    private $dato;
    public function setDato($v){ $this->dato = $v; }
    public function getDato(){ return $this->dato; }
}
```

Listing 18: Ejemplo de encapsulación

4.5 Herencia

Permite crear nuevas clases que heredan propiedades y métodos de otras.

```
class Animal {
    public function comer(){ echo "Comiendo"; }
}
class Perro extends Animal {}
$perro = new Perro();
$perro->comer();
```

Listing 19: Herencia básica

4.6 Polimorfismo

Clases derivadas pueden redefinir métodos para comportarse de forma distinta.

```
class Gato extends Animal {
    public function comer(){ echo "Gato comiendo"; }
}
```

Listing 20: Polimorfismo mediante redefinición

4.7 Clases abstractas

Definen métodos que deben implementarse en clases hijas, sirviendo como plantilla.

```
abstract class Figura {
    abstract public function area();
}
class Rectangulo extends Figura {
    private $a, $b;
    public function
       __construct($a,$b){$this->a=$a;$this->b=$b;}
    public function area(){return $this->a*$this->b;}
}
```

Listing 21: Clase abstracta y herencia

4.8 Interfaces

Especifican métodos que una clase debe implementar sin definir su funcionalidad.

```
interface Volador {
    public function volar();
}
class Pajaro implements Volador {
    public function volar(){ echo "Vuela"; }
}
```

Listing 22: Ejemplo de interfaz

4.9 Traits

Permiten reutilizar fragmentos de código en múltiples clases de manera horizontal.

```
trait Saludo {
    public function saludar(){ echo "Hola"; }
}
class Persona { use Saludo; }
```

Listing 23: Uso de Traits

4.10 Métodos mágicos

Son funciones especiales que facilitan comportamientos automáticos como construcción, destrucción, acceso a propiedades y más.

```
class Test {
    public function __toString(){ return "Objeto Test"; }
}
echo new Test(); // Objeto Test
```

Listing 24: Ejemplo de método mágico __toString

4.11 Clases anónimas

Clases sin nombre que pueden instanciarse directamente, útiles para objetos temporales.

```php
$obj = new class {
    public function saludar(){ echo "Hola"; }
};
$obj->saludar();
```

Listing 25: Clase anónima

4.12 Namespaces

Permiten organizar el código en espacios de nombres para evitar conflictos y mejorar modularidad.

```php
namespace Proyecto;
class Usuario {}
$u = new \Proyecto\Usuario();
```

Listing 26: Uso de namespaces

5 Comunicación con el cliente (formularios y peticiones)

5.1 GET y POST

Los formularios en HTML pueden enviar datos al servidor usando los métodos GET (por URL) o POST (en el cuerpo de la petición).

```
<form method="POST" action="procesar.php">
<input name="nombre">
<input type="submit">
</form>
```

Listing 27: Formulario con método POST

```
echo $_POST['nombre'];
```

Listing 28: Recepción de datos en PHP

5.2 Superglobales

PHP provee variables superglobales para acceder a datos de formularios y otros datos de la petición: $_GET, $_POST, $_REQUEST.

5.3 Formularios en un solo archivo

Se puede manejar formulario y procesamiento en un solo script verificando el método de la petición.

```php
if($_SERVER["REQUEST_METHOD"]=="POST"){
    echo $_POST['nombre'];
}
```

Listing 29: Procesamiento en el mismo archivo

5.4 Validación de formularios

Es fundamental limpiar y validar los datos recibidos para evitar problemas de seguridad o errores.

```php
$nombre = filter_input(INPUT_POST, "nombre",
   FILTER_SANITIZE_STRING);
```

Listing 30: Sanitización de entrada

5.5 Subida de archivos

PHP permite manejar archivos enviados por formularios con la variable $_FILES.

```php
if(isset($_FILES['archivo'])){
    move_uploaded_file($_FILES['archivo']['tmp_name'],
       'uploads/'.$_FILES['archivo']['name']);
}
```

Listing 31: Guardar archivo subido

6 Cookies y almacenamiento en navegador

Las cookies permiten almacenar información en el navegador del usuario y recuperarla en siguientes visitas.

```
setcookie("usuario","Juan", time()+3600); %\quad% // 1 hora
echo $_COOKIE["usuario"];
setcookie("usuario","",time()-3600); %\quad% // eliminar
```

Listing 32: Crear, leer y eliminar cookies

7 Sesiones y persistencia en servidor

Las sesiones almacenan información en el servidor, asociada a un usuario mediante un identificador.

```
session_start();
$_SESSION['usuario'] = "Ana";
echo $_SESSION['usuario'];
session_unset();
session_destroy();
```

Listing 33: Ejemplo básico de sesión

8 Validación avanzada

8.1 Expresiones regulares

PHP soporta expresiones regulares para validar patrones complejos en cadenas.

```php
$patron = "/^[a-z]+$/";
if(preg_match($patron, "hola")) echo "Válido";
```

Listing 34: Validación con expresión regular

9 Bases de Datos con PHP

9.1 Conexión con PDO

PDO es una interfaz flexible para conectarse a múltiples bases de datos.

```
$pdo = new
   PDO("mysql:host=localhost;dbname=test","root","");
```

Listing 35: Conexión con PDO

9.2 Consultas simples

Se pueden ejecutar consultas y recuperar resultados de forma sencilla.

```
$stmt = $pdo->query("SELECT * FROM usuarios");
while($fila = $stmt->fetch()){
    echo $fila['nombre'];
}
```

Listing 36: Consulta y recorrido de resultados

9.3 Instrucciones preparadas

Para evitar inyecciones SQL, se usan consultas preparadas con parámetros.

```
$stmt = $pdo->prepare("INSERT INTO usuarios(nombre) VALUES
    (:nombre)");
$stmt->execute(['nombre'=>"Ana"]);
```

Listing 37: Inserción segura con consulta preparada

9.4 Transacciones

Permiten ejecutar múltiples operaciones como una sola unidad, asegurando integridad.

```
$pdo->beginTransaction();
try {
    $pdo->exec("UPDATE cuentas SET saldo=saldo-100 WHERE
        id=1");
    $pdo->exec("UPDATE cuentas SET saldo=saldo+100 WHERE
        id=2");
    $pdo->commit();
} catch(Exception $e){
    $pdo->rollBack();
}
```

Listing 38: Uso de transacciones

10 Librerías y herramientas externas

10.1 Composer

Composer es el gestor de dependencias estándar en PHP, facilitando la instalación y actualización de librerías.

```
composer require phpmailer/phpmailer
```

Listing 39: Instalación de librería con Composer

10.2 PHPMailer

PHPMailer es una librería popular para enviar correos electrónicos con mayor facilidad y características avanzadas.

```
use PHPMailer\PHPMailer\PHPMailer;
$mail = new PHPMailer();
$mail->setFrom("from@example.com");
$mail->addAddress("to@example.com");
$mail->Subject = "Asunto";
$mail->Body = "Mensaje";
$mail->send();
```

Listing 40: Ejemplo básico de PHPMailer

10.3 PHPUnit

Es un framework para realizar pruebas unitarias que ayudan a garantizar la calidad del código.

11 Manejo general de errores

El manejo adecuado de errores es fundamental para programas robustos y confiables. PHP permite capturar excepciones usando try-catch.

```
try {
    $resultado = 10/0;
} catch(Exception $e){
    echo "Error: ".$e->getMessage();
}
```

Listing 41: Ejemplo de manejo de excepciones