DEFINIR ESTÁNDARES DE CODIFICACIÓN DE ACUERDO A PLATAFORMA DE DESARROLLO ELEGIDA

Presentado por:
Ana Karina Yepes Gómez c.c. 43612987
Jheison Fernando Rojas Rendon c.c. 1053846295
Adan de Jesús Restrepo Zapata c.c. 71290603
Jesús Harvey Bernal López. c.c. 1069725992

Ficha: 2627092

TECNOLOGÍA EN ANÁLISIS Y DESARROLLO DE SOFTWARE
SENA REGIONAL QUINDÍO.
2024

Tabla de contenido

1. INTRODUCCIÓN 5
2. OBJETIVOS 6
   2.1. Objetivo General: 6
   2.2. Objetivos Específicos: 6
3. Estándares de codificación para el lenguaje php 7
   3.1. PSR (PHP Standards Recommendations) 7
   3.2. Symfony Coding Standards 7
   3.3. Zend Framework Coding Standard 7
   3.4. WordPress Coding Standards 7
   3.5. Drupal Coding Standards 7
   3.6. PHP CodeSniffer 7
   3.7. PHP-CS-Fixer 7
4. Elección de Estándares y Herramientas 8
   4.1. Instalar Visual Studio Code. 8
   4.2. Instalar la Extensión de PHP. 8
   4.3. Configurar la Extensión de PHP 8
   4.4. Instalar Extensiones Complementarias 9
      4.4.1. Debugger for Chrome: 9
      4.4.2. HTML CSS Support: 9
      4.4.3. GitLens: 9
   4.5. Crear y Abrir Proyectos PHP 9
   4.6. Empezar a Codificar 9
5. Declaración de variables en PHP 9
   5.1. Nombres de las variables 9
      5.1.1. Sintaxis Básica 9
      5.1.2. Tipos de Datos 10
      5.1.3. Reglas de Nombres de Variables 11
      5.1.4. Variables Predefinidas (Superglobals) 12
   5.1.5. Variables Variables 12
      5.1.6. Ejemplo Completo 13
6. Declaración de funciones en PHP 14
   6.1. Declaración de Funciones 14

6.1.1.Sintaxis Básica 14
6.1.2. Ejemplo Sencillo 14
6.1.3. Funciones con Parámetros 14
6.1.4. Funciones con Valores por Defecto 15
6.1.5. Funciones con Retorno de Valores 15
6.1.6. Funciones con Tipado Estricto (PHP 7+) 16
6.1.7. Funciones Anónimas (Closures) 16
6.1.8. Funciones Flecha (PHP 7.4+) 16
6.1.9. Ejemplo Completo 17
7. Indicaciones y mejores prácticas para programar PHP 18
7.1. Organización del Código 18
7.1.1. Uso de Estructuras de Directorios 18
7.1.2. Seguir las Convenciones de Nombres 18
7.3. Comentarios y Documentación 19
7.4. Seguridad 19
7.4.1. Validación y Saneamiento de Datos 19
7.4.2. Uso de Sentencias Preparadas para SQL 19
7.5. Errores y Manejo de Excepciones 20
7.6. Configuración y Entorno 20
7.6.1.Usar Archivos de Configuración 20
7.7. Reutilización del Código 21
7.7.1. Funciones y Clases 21
7.8. Uso de Herramientas de Desarrollo 21
7.8.1. Composer 21
7.8.2. PSR (PHP Standard Recommendations) 21
7.9. Testing 22
7.10. Optimización y Buenas Prácticas 22
7.10.1. Mantén el código limpio y legible. 22
8. Estándares de codificación para el lenguaje javascript 24
8.1. Declaración de variables 24
8.1.1. Nombres de variables 24
8.1.2. Valores de variables 24
8.1.3. Declaración de multiples variables 24
8.2. Declaración de funciones 25
8.2.1. Nombres de funciones 25

8.2.2. Llaves y espaciado de funciones 25
8.2.3. Tipos de funciones 26
8.2.4. Argumentos de una función 26
8.2.5. Construcción de cadenas 26
8.3. Operaciones 26
8.3.1. Operaciones lógicas de igualdad y desigualdad 27
9. BIBLIOGRAFÍA 28

1. INTRODUCCIÓN

En la documentación del material de formación (Aplicacion del paradigma orientado a objetos, s.f.) se habla del uso de Programación Orientada a Objetos (POO) cuando se aplican las características de abstracción, encapsulación, herencia y polimorfismo. La POO encierra datos (atributos) y métodos (comportamientos) que hacen parte del objeto y que se encuentran relacionados entre sí.

La POO se caracteriza por la encapsulación de datos, ocultación de datos y mecanismos de acceso, inicialización automática y sobrecarga de operadores, además de implementar la herencia y ligadura dinámica. Entre los lenguajes que soportan estas características se incluyen C++, Smalltalk, Object Pascal y Java, aunque existe un gran número de lenguajes de programación basados en objetos y orientados a objetos.

Para el desarrollo de esta evidencia se toma como lenguaje principal el PHP, HTML, CCS y JavaScript para la ejecución del proyecto en desarrollo que busca mejorar el servicio de Vigilancia y seguridad privada a los usuarios, prestando la función a través de un aplicativo web que sirve como intermediario entre cliente y prestador de servicio, donde tanto el cliente como la Empresa podrán optimizar de manera satisfactoria la adquisición y prestación de servicios que les brinden una mejora en los procesos donde el cliente tendrá la oportunidad de acceder al aplicativo web para encontrar variedad de opciones de productos y servicios que necesite para su consumo que se ajusten a sus necesidades y al mismo tiempo la empresa de seguridad contará con la oportunidad de administrar la información necesaria para la comercialización de sus productos y servicios.

En este trabajo exploraremos los Estándares de codificación para PHP, y Javascript, dos lenguajes de programación o de secuencias de comandos que permite implementar funciones complejas en páginas web, como crear contenido de actualización dinámica, controlar multimedia, animar imágenes, entre otras.

2. OBJETIVOS

2.1. Objetivo General:

- Realizar un informe técnico con el estándar de codificación del lenguaje a utilizar en el desarrollo del software

2.2. Objetivos Específicos:

- Explorar el tema de la programación orientada a objetos
- Con base al lenguaje de programación seleccionado para la ejecución del proyecto, definir los estándares de codificación, que serán la base del desarrollo del proyecto.

3. Estándares de codificación para el lenguaje php

Los estándares de codificación son pautas que ayudan a mantener la consistencia y la legibilidad del código en un proyecto. Para PHP, hay varios estándares populares que se utilizan ampliamente en la comunidad de desarrollo. Aquí hay algunos de los más comunes:

3.1. PSR (PHP Standards Recommendations)

Las PSR son una serie de recomendaciones emitidas por el Grupo de Trabajo de PHP-Framework Interoperability para estandarizar el desarrollo en PHP. Algunas de las PSR más importantes son:
PSR-1: Básico de codificación: define las pautas básicas para la creación de archivos PHP.
PSR-2: Estilo de codificación: define un conjunto de reglas de estilo para mejorar la legibilidad del código.
PSR-4: Autocarga de clases: define un estándar para la autocarga de clases basado en espacios de nombres.
PSR-12: Estándar de codificación extendido: amplía las reglas de PSR-2 para mejorar aún más la consistencia del código.

3.2. Symfony Coding Standards

Symfony es un popular framework PHP y tiene sus propias pautas de codificación. Estas normas están diseñadas para ser compatibles con las PSR y abarcan aspectos más allá de las reglas básicas de codificación.

3.3. Zend Framework Coding Standard

Zend Framework también tiene sus propias normas de codificación. Estas normas están diseñadas para ser consistentes con las PSR y proporcionan pautas adicionales para el desarrollo en el contexto del framework Zend.

3.4. WordPress Coding Standards

WordPress es otro gran proyecto PHP con sus propias pautas de codificación. Estas normas están diseñadas para ser consistentes con las PSR y están específicamente adaptadas para el desarrollo de plugins, temas y núcleo de WordPress.

3.5. Drupal Coding Standards

Drupal, un popular sistema de gestión de contenido, también tiene sus propias normas de codificación. Estas normas están diseñadas para ser compatibles con las PSR y se enfocan en la creación de módulos, temas y el núcleo de Drupal.

3.6. PHP CodeSniffer

PHP CodeSniffer es una herramienta que se puede utilizar para verificar si un código cumple con ciertos estándares de codificación, incluidos los mencionados anteriormente. Puedes configurarlo para que se ajuste a las pautas específicas que prefieras para tu proyecto.

3.7. PHP-CS-Fixer

PHP-CS-Fixer es otra herramienta útil que puede ayudar a mantener el código PHP en línea

con los estándares de codificación. Puedes usarlo para aplicar automáticamente las correcciones necesarias para que tu código cumpla con ciertos estándares.

4. Elección de Estándares y Herramientas

La elección de los estándares de codificación y las herramientas dependerá del proyecto gespecífico y de las preferencias del equipo de desarrollo. Es importante elegir y aplicar consistentemente un conjunto de estándares para mantener la calidad y la legibilidad del código en el tiempo.

Para este proyecto de desarrollo de software utilizamos el Visual Studio Code como editor de código ligero, junto con la extensión de PHP del mismo. Para mantener un código limpio, consistente y fácil de entender en PHP, es importante seguir estándares de codificación. Aunque Visual Studio Code (VS Code) es principalmente un editor de código y no impone directamente estándares de codificación, puedes configurarlo para que te ayude a mantener los estándares que elijas.

Visual Studio Code es un editor de código ligero, potente y altamente personalizable que es muy popular entre los desarrolladores de PHP debido a su flexibilidad y extensibilidad. Aquí te muestro cómo configurar Visual Studio Code para trabajar con PHP:

4.1. Instalar Visual Studio Code.

Descargarlo e instalarlo desde su sitio web oficial: Visual Studio Code.

4.2. Instalar la Extensión de PHP.

Visual Studio Code tiene una amplia gama de extensiones disponibles que pueden mejorar la experiencia de desarrollo. Para trabajar con PHP, puedes instalar la extensión oficial de PHP.

- Abre Visual Studio Code.
- Ve a la pestaña de extensiones (icono de cubo en la barra lateral izquierda) o presiona Ctrl+Shift+X.
- Busca "PHP" en el campo de búsqueda.
- Haz clic en "Instalar" en la extensión de PHP ofrecida por "Felix Becker" (es la extensión oficial de PHP para Visual Studio Code).

4.3. Configurar la Extensión de PHP

Una vez instalada la extensión de PHP, puedes personalizar su configuración según tus necesidades. Puedes acceder a la configuración de la extensión de PHP a través del menú "Archivo" > "Preferencias" > "Configuración" o usando Ctrl+,.

Algunas configuraciones que puedes ajustar incluyen:

- Ruta al ejecutable de PHP.
- Ruta al archivo de configuración de PHP.

- Configuración de la depuración de PHP (Xdebug u otra herramienta).
- Opciones de formato de código PHP.
- Opciones de análisis de código PHP.

4.4. Instalar Extensiones Complementarias

Además de la extensión de PHP, puedes instalar otras extensiones que pueden ser útiles para el desarrollo de PHP, como:

4.4.1. Debugger for Chrome: Si trabajas con aplicaciones web PHP y JavaScript, esta extensión te permite depurar código JavaScript en el navegador Chrome.
4.4.2. HTML CSS Support: Ofrece funciones de autocompletado y resaltado de sintaxis para HTML y CSS.
4.4.3. GitLens: Si trabajas con Git, esta extensión mejora la integración de Git en Visual Studio Code.

4.5. Crear y Abrir Proyectos PHP

Una vez que hayas configurado Visual Studio Code y instalado las extensiones necesarias, puedes crear o abrir proyectos PHP directamente en Visual Studio Code. Puedes abrir una carpeta existente o crear un nuevo proyecto PHP desde cero.

4.6. Empezar a Codificar

¡Ahora estás listo para empezar a codificar en PHP con Visual Studio Code! Puedes escribir y editar archivos PHP, depurar tu código, administrar tu proyecto y mucho más, todo desde un entorno unificado y altamente personalizable.

Visual Studio Code ofrece una gran cantidad de características y opciones de personalización que pueden mejorar tu productividad como desarrollador de PHP. Experimenta con diferentes extensiones y configuraciones para encontrar la combinación que mejor se adapte a tus necesidades y estilo de trabajo.

5. Declaración de variables en PHP

5.1. Nombres de las variables

En PHP, la declaración de variables es bastante sencilla y flexible. A continuación, se explican los conceptos básicos sobre cómo declarar y utilizar variables en PHP:

5.1.1. Sintaxis Básica

En PHP, las variables se declaran utilizando el signo de dólar ($) seguido del nombre de la variable. El nombre de la variable puede contener letras, números y el carácter de subrayado (_), pero debe

comenzar con una letra o un subrayado.

```php
<?php
$variable = "Valor";
?>
```

5.1.2. Tipos de Datos

PHP es un lenguaje de tipado dinámico, lo que significa que no necesitas declarar explícitamente el tipo de una variable. PHP determinará el tipo de datos en función del valor asignado a la variable.

Tipos de datos básicos:

Enteros (integer):

```php
<?php
$numero = 123;
?>
```

Flotantes (float/double):

```php
<?php
$decimal = 12.34;
?>
```

Cadenas de texto (string):

```php
<?php
$texto = "Hola, mundo!";
?>
```

Booleanos (boolean):

```php
<?php
$verdadero = true;
$falso = false;
?>
```

Arrays:

```php
<?php
$array = array(1, 2, 3);
?>
```

Objetos (objects):

```php
<?php
class Persona {
    public $nombre;
    public $edad;
}

$persona = new Persona();
$persona->nombre = "Juan";
$persona->edad = 25;
?>
```

NULL:

```php
<?php
$variable = null;
?>
```

5.1.3. Reglas de Nombres de Variables

El nombre de una variable debe comenzar con una letra o un guion bajo (_).
Puede contener letras, números y guiones bajos después del primer carácter.
PHP es sensible a mayúsculas y minúsculas, por lo que `$variable` y `$Variable` se consideran diferentes variables.

```php
<?php
$variable1 = "Hola";
$Variable1 = "Mundo";

echo $variable1; // Imprime "Hola"
echo $Variable1; // Imprime "Mundo"
?>
```

5.1.4. Variables Predefinidas (Superglobals)

PHP proporciona una serie de variables predefinidas, también conocidas como superglobals, que están disponibles en todos los ámbitos (global, local, etc.) sin necesidad de declararlas explícitamente.

Algunas de las superglobals más comunes son:

- $_GET: Contiene los parámetros de la URL.
- $_POST: Contiene los datos enviados por un formulario mediante el método POST.
- $_REQUEST: Contiene datos de $_GET, $_POST y $_COOKIE.
- $_SERVER: Contiene información del servidor y del entorno de ejecución.
- $_SESSION: Contiene los datos de la sesión.
- $_COOKIE: Contiene las cookies.

```php
<?php
echo $_SERVER['HTTP_USER_AGENT']; // Muestra el user agent del cliente
?>
```

5.1.5. Variables Variables

PHP permite crear variables variables, es decir, el nombre de una variable puede ser dinámico.

```php
<?php
$nombreVariable = "variableDinamica";
$$nombreVariable = "Valor Dinámico";

echo $variableDinamica; // Imprime "Valor Dinámico"
?>
```

5.1.6. Ejemplo Completo

Ejemplo que muestra varias declaraciones y usos de variables en PHP:

php Copiar código

```php
<?php
// Declaración de variables
$nombre = "Carlos";
$edad = 30;
$altura = 1.75;
$esProgramador = true;

// Uso de arrays
$lenguajes = array("PHP", "JavaScript", "Python");

// Uso de objetos
class Persona {
    public $nombre;
    public $edad;
}

$persona = new Persona();
$persona->nombre = "Ana";
$persona->edad = 28;

// Impresión de variables
echo "Nombre: " . $nombre . "<br>";
echo "Edad: " . $edad . "<br>";
echo "Altura: " . $altura . "m<br>";
echo "Es programador: " . ($esProgramador ? "Sí" : "No") . "<br>";

// Impresión de arrays
echo "Lenguajes de programación: " . implode(", ", $lenguajes) . "<br>";

// Impresión de objetos
echo "Nombre de la persona: " . $persona->nombre . "<br>";
echo "Edad de la persona: " . $persona->edad . "<br>";

// Uso de variables variables
$variable = "nombre";
$$variable = "Roberto";
echo "Nombre dinámico: " . $nombre . "<br>"; // Imprime "Roberto"
?>
```

6. Declaración de funciones en PHP

En PHP, las funciones son bloques de código reutilizables que se pueden definir y llamar en cualquier parte del script. A continuación, se detalla cómo declarar y utilizar funciones en PHP:

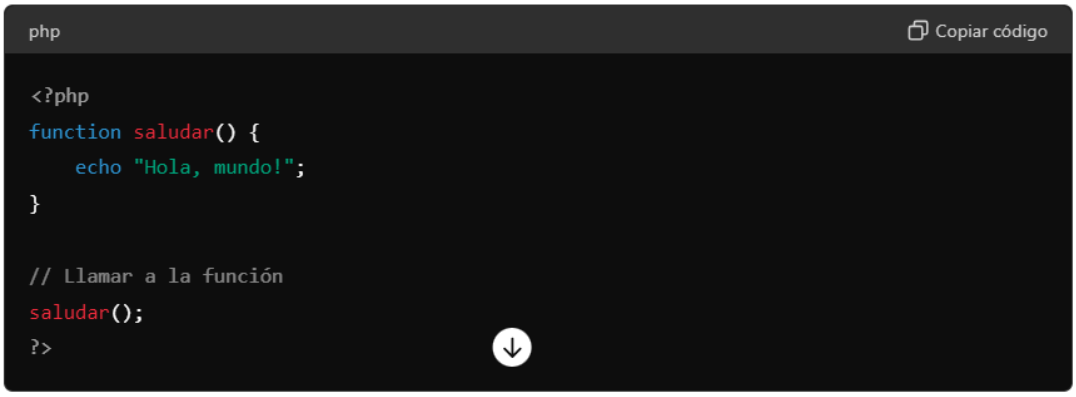
6.1. Declaración de Funciones

Para declarar una función en PHP, se utiliza la palabra clave function, seguida del nombre de la función, paréntesis (que pueden incluir parámetros) y un bloque de código delimitado por llaves {}.

6.1.1.Sintaxis Básica

6.1.2. Ejemplo Sencillo

A continuación, se muestra un ejemplo básico de una función que imprime un mensaje:



6.1.3. Funciones con Parámetros

Las funciones pueden aceptar parámetros que permiten pasar información al bloque de código dentro de la función. Los parámetros se definen dentro de los paréntesis después del nombre de la función.

Ejemplo con Parámetros

```php
<?php
function saludar($nombre) {
    echo "Hola, " . $nombre . "!";
}

// Llamar a la función con un argumento
saludar("Carlos");
?>
```

6.1.4. Funciones con Valores por Defecto

Puedes definir valores por defecto para los parámetros. Si no se pasa un argumento al llamar a la función, se usará el valor por defecto.

Ejemplo con Valores por Defecto

```php
<?php
function saludar($nombre = "mundo") {
    echo "Hola, " . $nombre . "!";
}

// Llamar a la función sin argumento
saludar(); // Imprime "Hola, mundo!"

// Llamar a la función con un argumento
saludar("Carlos"); // Imprime "Hola, Carlos!"
?>
```

6.1.5. Funciones con Retorno de Valores

Las funciones pueden devolver valores usando la palabra clave `return`. Cuando se llama a una función que devuelve un valor, puedes capturar ese valor en una variable.

Ejemplo con Retorno de Valores

```php
<?php
function sumar($a, $b) {
    return $a + $b;
}

// Llamar a la función y capturar el valor retornado
$resultado = sumar(5, 3);
echo "La suma es: " . $resultado; // Imprime "La suma es: 8"
?>
```

6.1.6. Funciones con Tipado Estricto (PHP 7+)

PHP 7 introdujo el tipado estricto, que permite especificar los tipos de los parámetros y el tipo de retorno de una función. Para habilitar el tipado estricto, se debe declarar al inicio del archivo PHP.

Ejemplo con Tipado Estricto

php Copiar código

```php
<?php
declare(strict_types=1);

function sumar(int $a, int $b): int {
    return $a + $b;
}

// Llamar a la función
$resultado = sumar(5, 3);
echo "La suma es: " . $resultado; // Imprime "La suma es: 8"
?>
```

6.1.7. Funciones Anónimas (Closures)

PHP también soporta funciones anónimas, que son funciones sin nombre. Estas son útiles para pasar como parámetros a otras funciones o asignar a variables.

Ejemplo de Función Anónima

php Copiar código

```php
<?php
$saludar = function($nombre) {
    echo "Hola, " . $nombre . "!";
};

// Llamar a la función anónima
$saludar("Carlos");
?>
```

6.1.8. Funciones Flecha (PHP 7.4+)

PHP 7.4 introdujo las funciones flecha, que son una sintaxis más corta para definir funciones anónimas. Usan la palabra clave `fn`.

Ejemplo de Función Flecha

```php
<?php
$suma = fn($a, $b) => $a + $b;

// Llamar a la función flecha
echo $suma(5, 3); // Imprime "8"
?>
```

6.1.9. Ejemplo Completo

Ejemplo completo que muestra varios aspectos de la declaración y uso de funciones en PHP:

```php
<?php
// Declarar una función sin parámetros
function saludar() {
    echo "Hola, mundo!<br>";
}

// Llamar a la función
saludar();

// Declarar una función con parámetros
function saludarConNombre($nombre) {
    echo "Hola, " . $nombre . "!<br>";
}

// Llamar a la función con un argumento
saludarConNombre("Carlos");

// Declarar una función con un valor por defecto
function saludarPorDefecto($nombre = "mundo") {
    echo "Hola, " . $nombre . "!<br>";
}

// Llamar a la función sin argumento
saludarPorDefecto();

// Llamar a la función con un argumento
saludarPorDefecto("Ana");

// Declarar una función que devuelve un valor
function multiplicar($a, $b) {
    return $a * $b;
}

// Llamar a la función y capturar el valor retornado
$resultado = multiplicar(4, 5);
echo "El resultado de la multiplicación es: " . $resultado . "<br>";

// Declarar una función con tipado estricto
declare(strict_types=1);
function dividir(int $a, int $b): float {
    return $a / $b;
}

// Llamar a la función con tipado estricto
$resultadoDivision = dividir(10, 2);
echo "El resultado de la división es: " . $resultadoDivision . "<br>";

// Declarar una función anónima
$saludarAnonimo = function($nombre) {
    echo "Hola, " . $nombre . "!<br>";
};

// Llamar a la función anónima
$saludarAnonimo("Pedro");

// Declarar una función flecha
$suma = fn($a, $b) => $a + $b;

// Llamar a la función flecha
echo "La suma es: " . $suma(3, 7) . "<br>";
?>
```

Este ejemplo completo ilustra cómo declarar y usar funciones en PHP, incluyendo funciones con parámetros, valores por defecto, retorno de valores, tipado estricto, funciones anónimas y funciones flecha.

7. Indicaciones y mejores prácticas para programar PHP

Programar en PHP de manera efectiva y siguiendo buenas prácticas es fundamental para desarrollar aplicaciones web seguras, mantenibles y eficientes. A continuación, se presentan algunas indicaciones y mejores prácticas para programar en PHP:

7.1. Organización del Código

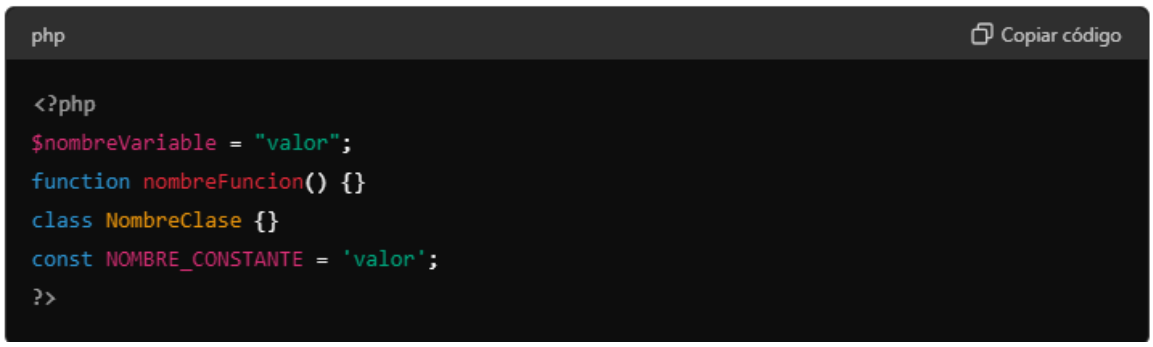
7.1.1. Uso de Estructuras de Directorios

Organiza tu proyecto en carpetas lógicas como controllers, models, views, helpers, etc. Esto facilita la navegación y el mantenimiento del código.

7.1.2. Seguir las Convenciones de Nombres

Variables y funciones: Utiliza el estilo `camelCase`.

Clases: Utiliza `PascalCase`.
Constantes: Utiliza `SNAKE_CASE`.



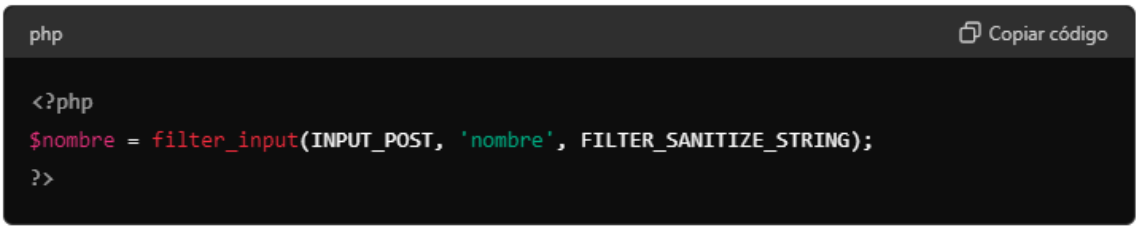
7.3. Comentarios y Documentación

Usa comentarios para explicar partes complejas del código y docblocks para funciones, métodos y clases.

7.4. Seguridad

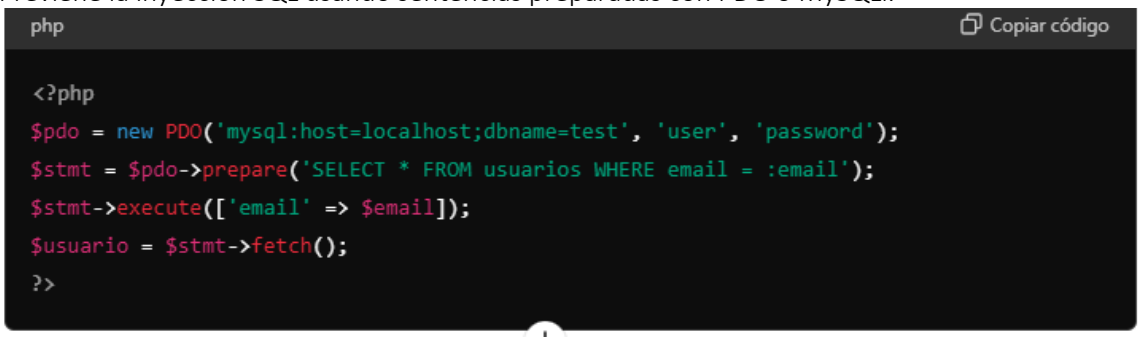
7.4.1. Validación y Saneamiento de Datos

Siempre valida y desinfecta la entrada del usuario para prevenir inyecciones de código y otros ataques.



7.4.2. Uso de Sentencias Preparadas para SQL

Previene la inyección SQL usando sentencias preparadas con PDO o MySQLi.



7.5. Errores y Manejo de Excepciones

Maneja adecuadamente los errores y excepciones para mejorar la robustez del código.

php Copiar código

```php
<?php
try {
    // Código que puede lanzar una excepción
    $pdo = new PDO('mysql:host=localhost;dbname=test', 'user', 'password');
} catch (PDOException $e) {
    echo 'Error de conexión: ' . $e->getMessage();
}
?>
```

7.6. Configuración y Entorno

7.6.1.Usar Archivos de Configuración

Almacena configuraciones y credenciales en un archivo separado, como `.env`, y utiliza un cargador de configuración.

plaintext Copiar código

```plaintext
DB_HOST=localhost
DB_USER=user
DB_PASS=password
```

php Copiar código

```php
<?php
$dotenv = Dotenv\Dotenv::createImmutable(__DIR__);
$dotenv->load();

$host = $_ENV['DB_HOST'];
$user = $_ENV['DB_USER'];
$password = $_ENV['DB_PASS'];
?>
```

7.7. Reutilización del Código

7.7.1. Funciones y Clases

Escribe funciones y clases reutilizables para evitar la duplicación del código.

```php
<?php
class Usuario {
    private $nombre;
    private $email;

    public function __construct($nombre, $email) {
        $this->nombre = $nombre;
        $this->email = $email;
    }

    public function getNombre() {
        return $this->nombre;
    }

    public function getEmail() {
        return $this->email;
    }
}
?>
```

7.8. Uso de Herramientas de Desarrollo

7.8.1. Composer

Usa Composer para manejar dependencias y autoloading.

```bash
composer init
composer require monolog/monolog
```

7.8.2. PSR (PHP Standard Recommendations)

Adopta las recomendaciones PSR para estandarizar el código. Por ejemplo, sigue PSR-4 para autoloading de clases.

```json
{
   "autoload": {
      "psr-4": {
         "App\\": "src/"
      }
   }
}
```

7.9. Testing

Escribe pruebas unitarias y funcionales para asegurar la calidad del código. PHPUnit es una herramienta comúnmente utilizada.

```bash
composer require --dev phpunit/phpunit
```

Ejemplo de prueba unitaria:

```php
<?php
use PHPUnit\Framework\TestCase;

class UsuarioTest extends TestCase {
    public function testGetNombre() {
        $usuario = new Usuario("Carlos", "carlos@example.com");
        $this->assertEquals("Carlos", $usuario->getNombre());
    }
}
?>
```

7.10. Optimización y Buenas Prácticas

7.10.1. Mantén el código limpio y legible.

Evita el uso de include y require en favor de require_once y include_once para evitar múltiples inclusiones.

Optimiza las consultas a la base de datos y usa caché donde sea posible.

Ejemplo Completo de Aplicación PHP

Un ejemplo básico de aplicación siguiendo buenas prácticas:

```php
<?php
// Autoloading con Composer
require 'vendor/autoload.php';

// Configuración del entorno
$dotenv = Dotenv\Dotenv::createImmutable(__DIR__);
$dotenv->load();

// Conexión a la base de datos usando PDO
try {
    $pdo = new PDO(
        "mysql:host=" . $_ENV['DB_HOST'] . ";dbname=" . $_ENV['DB_NAME'],
        $_ENV['DB_USER'],
        $_ENV['DB_PASS']
    );
} catch (PDOException $e) {
    die('Error de conexión: ' . $e->getMessage());
}

// Modelo
class Usuario {
    private $pdo;
```

```php
    public function __construct($pdo) {
        $this->pdo = $pdo;
    }

    public function obtenerUsuarios() {
        $stmt = $this->pdo->query('SELECT * FROM usuarios');
        return $stmt->fetchAll(PDO::FETCH_ASSOC);
    }
}

// Controlador
class UsuarioController {
    private $usuarioModel;

    public function __construct($usuarioModel) {
        $this->usuarioModel = $usuarioModel;
    }

    public function mostrarUsuarios() {
        $usuarios = $this->usuarioModel->obtenerUsuarios();
        require 'views/usuarios.php';
    }
}
```

```php
?>
<!-- views/usuarios.php -->
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Lista de Usuarios</title>
</head>
<body>
    <h1>Lista de Usuarios</h1>
    <ul>
        <?php foreach ($usuarios as $usuario): ?>
            <li><?php echo htmlspecialchars($usuario['nombre']); ?></li>
        <?php endforeach; ?>
    </ul>
</body>
</html>

<?php
// Index
$usuarioModel = new Usuario($pdo);
$usuarioController = new UsuarioController($usuarioModel);
$usuarioController->mostrarUsuarios();
?>
```

Este ejemplo ilustra cómo estructurar una aplicación PHP utilizando buenas prácticas de programación, incluyendo la organización del código, la seguridad, el manejo de errores, la reutilización del código, y el uso de herramientas modernas.

8. Estándares de codificación para el lenguaje javascript

8.1. Declaración de variables

Declarar variables es una de las cosas más importantes a la hora de codificar, por eso es necesario tener en cuenta las siguientes premisas:

8.1.1. Nombres de variables

Los nombres de variables deben ser escritos con notación camelCase.

```
var test;  // Nombre de variable correcto
var i_am_bad; //  Nombre de variable incorrecto
var iAmFine; //  Nombre de variable correcto
```

8.1.2. Valores de variables

Los espacios entre el nombre y el valor de una variable son muy importantes, puesto que mejoran la legibilidad del código.

```
var test = 1;  // Nombre de variable correcto
var iAmFine = true; //  Nombre de variable correcto
```

8.1.3. Declaración de multiples variables

Cuando se declaran multiples variables deben declararse todas con la misma sentencia var, una variable por linea y con coma (,) al final a menos que sea la ultima variable, en cuyo caso se usará punto y coma (;) al final de la linea. A partir de la segunda linea debe usarse 4 espacios antes del nombre de variable para mejorar la legibilidad del código fuente. El orden de las variables debe ser alfabetico de ser posible.

• Una declaración erronea es de la siguiente manera:

```
// declaración erronea
var bad1 = 1;
var iAmFine = false;
var sampleVariable = "wrong";
```

• Una declaración correcta es de la siguiente manera:

```
// declaración correcta
var iAmFine = true, // usa coma
   greatDay = 1, // usa coma
         sampleVariable = true; // usa punto y coma por que es la ultima variable
```

Nota: Es aconsejable que cada variable tenga un comentario acerca de su objetivo, de manera que aumente la comprensibilidad del código.

Variables de tipo colección

Cuando la variable a declarar contiene un elemento de tipo colección (arreglo y objeto) y su contenido excede los 70 caracteres o tiene más de 2 elementos debe tener cada item de la colección en una linea diferente para mejorar la legibilidad del código.

```
// declaración correcta
var elements = [], // colección vacia
   gender = ["male", "female"], // colección con 2 elementos
   users = {
      "Hugo": "Duck 1",
```

```
    "Paco": "Duck 2",
    "Luis": "Duck 3"
  }, // colección con más de 2 elementos
  attributes = [
    "fuerza",
    "velocidad",
    "inteligencia",
    "carisma",
  ],
  base64_logo_parts = [
'','FBMV
EUAAAA/mUPidDHiLi5Cn0XkNTPmeUrkdUg/m0Q0pEfcpSbwaVdKskg+lUP4zA/iLi3m','sSHkOjVAmET
dJSjtYFE/lkPnRj3sWUs8kkLeqCVIq0fxvhXqUkbVmSjwa1n1yBLepyX1xxP','0xRXqUkboST9KukpHpUbu
vRrzrhF/ljbwaljuZFM4jELaoSdLtElJrUj1xxP6zwzfqSU4i0',
    // ...
    // more lines
    // ...

'/0rbNvHVxiJywa8yS2KDfV1dfbu31H8jF1RHiTKtWYeHxUvq3bn0pyjCRaiRU6aDO+gb3aE','fEeVNsDg
m8zzLy9egPa7Qt8TSJdwhjplk06HH43ZNJ3s91KKCHQ5x4sw1fRGYDZ0n1L4FKb','9/BP5JLYxToheoFCV
xz57PPS8UhhEpLBVeAAAAAElFTkSuQmCC'
  ]; // colección con más de 70 caracteres
```

8.2. Declaración de funciones

Uno de los elementos más importantes de javascript son las funciones, éstas son conocidas como ciudadanos de primer tipo. Para la declaración de funciones es necesario tener en cuenta las siguientes premisas:

8.2.1. Nombres de funciones

Los nombres de funciones deben ser escritos con notación camelCase.

```
function testFunction() {
  // code here
}
```

8.2.2. Llaves y espaciado de funciones

Las llaves de las funciones deben empezar en la misma linea que se declara la función y debe haber un espacio entre el parentesis de cierre de argumentos y la llave de inicio de cuerpo de función, así:

```
// A. Declaración de función
// B. Parentesis de argumentos
// C. Espacio
// D. Llave de inicio de cuerpo de función.
// ------- A ------- --- B --- C D
function testFunction(username) {
  console.log('Hello ' + username + '...');
}
```

8.2.3. Tipos de funciones

En javascript las funciones pueden ser funciones como tal o funciones como variables. En caso de que sean funciones como variables es necesario usar punto y coma (;) despues del cuerpo de la función para finalizar la sentencia.

```
function testFunction() {
   console.log('Hello Test...');
}

var anotherTestFunction = function() {
   console.log('Hello Test...');
}; // ; para fin de sentencia.
```

8.2.4. Argumentos de una función

Para mejorar la legibilidad del código javascript los argumentos de funciones deben ser nombres con notación camelCase. Si son varios argumentos deben estar separados por coma (,) y un espacio, así:

```
function testFunction(arg1, arg2, arg3) {
   console.log('Hello Test...');
}

var testFunction = function(arg1, arg2) {
   console.log('Hello Test...');
}; // ; para fin de sentencia.
```

Si unicamente hay un argumento no hay espacios a ningún lado, así:

```
function testFunction(arg1){
   console.log('Hello Test...');
}
```

8.2.5. Construcción de cadenas

Deben ser usadas comilas simples (') puesto de ésta manera se reducen los problemas a la hora de escapar las comillas dobles (") usadas cuando se genera HTML y mejora la legibilidad.

```
// Incorrecto
var container0 = "<div class=\"another-container\" data-type=\"html\"></div>";
var container1 = "<div class="+'""'+"another-container"+'""'+
   " data-type="+'""'+"html"+'""'+"></div>";
// Correcto
var containerString = '<div class="container" data-type="html"></div>';
```

Nota: Para los casos en que no se estan construyendo elementos del DOM también deben ser usadas las comillas simples por uniformidad en el código.

8.3. Operaciones

Todas las operaciones deben tener espacios entre los operadores y los operandos para mejorar la legibilidad del código. Si se están usando parentesis no es necesario usar espacio entre los parentesis y los operandos.

```
var v1 = b * h / 2,
   v2 = (location.toLowerCase() === 'medellin') ? 'frijoles' : 'lentejas';
```

8.3.1. Operaciones lógicas de igualdad y desigualdad

Deben usarse los comparadores estrictos (aquellos que comparan valor y tipo ===, !==) para todas las operaciones lógicas de comparación.

```
function factorial(x) {
   if(x === 0){
      return 1;
   } else if(x !== 0) {
      return factorial(x - 1) * x;
   }
}

factorial(10);
```

9. BIBLIOGRAFÍA

PHP Standards Recommendations (PSR). Sitio Web Oficial: https://www.php-fig.org/

Repositorio en GitHub: https://github.com/php-fig

Symfony Coding Standards https://symfony.com/doc/current/contributing/code/standards.html

WordPress Coding Standards https://developer.wordpress.org/coding-standards/wordpress-coding-standards/

Aplicación del paradigma orientado a objetos. (s.f.). Obtenido de https://sena.territorio.la/content/index.php/institucion/Titulada/institution/SENA/Tecnologia/228118/Contenido/OVA/CF28/index.html#/introduccion

Estandares de codificacion para el lenjuaje JavaScript. (s.f.). Obtenido de https://github.com/0granada/js-coding-standards

mnd web docs. (s.f.). Obtenido de https://developer.mozilla.org/es/docs/Learn/JavaScript/First_steps/What_is_JavaScript

Rumbaugh, J., Jacobson, I. & Booch, G. (2004). El Lenguaje Unificado de Modelado. Elements, 30. http://portal.acm.org/citation.cfm?id=993859&dl=