**Plantilla para Propuesta de Implementación de un Analizador Léxico, Sintáctico y Semántico**

**Implementación de una herramienta GUI con el fin de validar el léxico, sintaxis y semántica del lenguaje de programación PHP**

**1. Introducción**

PHP (Hypertext Preprocessor) es un lenguaje de programación de propósito general, utilizado en el desarrollo web del lado del servidor [1]. Fue creado en 1994 por Rasmus Lerdorf y ha evolucionado hasta convertirse en uno de los lenguajes más populares para construir sitios web dinámicos y aplicaciones web robustas. PHP es un lenguaje interpretado y orientado a objetos. Su importancia radica en la gran cantidad de frameworks disponibles además de que se integra fácilmente con bases de datos y otros servicios web.

El proyecto que se presenta tiene como objetivo desarrollar una herramienta que permita validar el código PHP en tres niveles fundamentales: léxico, sintáctico y semántico. Esto significa que la herramienta podrá identificar errores en la escritura del código (léxico), en la estructura o reglas del lenguaje (sintaxis) y en el sentido o significado correcto del código (semántica).

La interfaz gráfica de usuario (GUI) de la herramienta de validación de PHP estará diseñada para ser intuitiva y facilitar la interacción del usuario con el proceso de análisis del código. Entre las funcionalidades principales que ofrecerá la GUI se encuentran:

**Editor de Código Integrado**  
Un espacio donde el usuario pueda escribir o pegar código PHP para ser analizado.

**Carga y Guardado de Archivos**  
Permite abrir archivos PHP existentes desde el sistema y guardar los cambios realizados en el editor.

**Resaltado de Errores en el Código**  
Destaca visualmente las partes del código donde se encuentran los errores, facilitando la corrección por parte del usuario.

**Reporte de Resultados**  
Genera un informe exportable (por ejemplo, en formato texto o PDF) con el resumen del análisis realizado, útil para documentar el proceso o compartir con el equipo.

**2. Analizador Léxico**

**Reconocimiento de tokens válidos y de error:**

Identificar correctamente los tokens de PHP que abarcaremos en este proyecto según su especificación, incluyendo palabras reservadas, operadores, identificadores, números, cadenas, comentarios, espacios en blanco y símbolos especiales. Cuando el analizador encuentra caracteres o secuencias que no corresponden a ningún token válido del lenguaje, debe generar un token de error y producir mensajes personalizados que informen con claridad la naturaleza del error, su ubicación exacta (línea y columna)

**Manejo de espacios y comentarios:**

Los espacios en blanco y comentarios son reconocidos y descartados o procesados adecuadamente para no afectar la generación de tokens válidos.

**Salida estructurada:**

La secuencia de tokens generada será utilizada por las etapas posteriores del análisis sintáctico y semántico. Por ello, cada token incluirá información relevante, como el tipo de token, su valor textual y su posición en el código.

**Componentes Principales**

**Variables**

**Prefijo obligatorio $:**

* Todas las variables comienzan con un signo de dólar ($).
* Ejemplo: $nombre, $edad, $x.

**Sensibles a mayúsculas y minúsculas:**

* $Var y $var son variables diferentes.

**Nombres válidos:**

* Después del $, el nombre debe comenzar con una letra o guion bajo (\_).
* Puede contener letras, números y guiones bajos.
* No puede comenzar con un número ni contener caracteres especiales como @, #, -, etc.

**No se necesita declarar tipo**:

* PHP es un lenguaje **débilmente tipado y dinámico**. Una variable puede cambiar de tipo según su valor. [2]

**Tipos de datos**

**1. Primitivos**

**int (enteros):**  
Números sin decimales: -10, 0, 42

**float (o double):**  
Números con decimales: 3.14, -2.7, 1.0e3

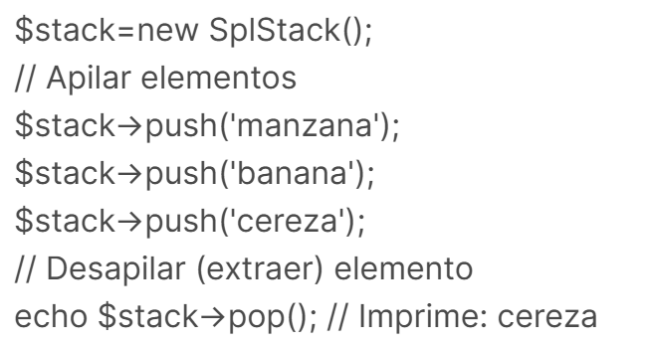
**string:**  
Cadenas de texto: "Hola mundo", 'PHP'

**bool:**  
Booleanos: true, false

**2. Compuestos**

**1. Pila**

SplStack es la implementación de **pila** en php que sigue el principio **LIFO** (*Last In, First Out*), es decir, el último elemento que agregas es el primero que se extrae.

// Pila usando SplStack (Nehemías Lindao)

**2. Array simple**

Un array simple es una colección ordenada de valores indexados por números enteros. Se utiliza cuando los elementos no necesitan una clave personalizada y se accede a ellos mediante su índice numérico. [3]

**Características clave:**

* Índices automáticos (0, 1, 2…).
* Ideal para listas de valores.

// Array simple (Daniel Zavala)

$colores = ["rojo", "verde", "azul"];

**3. Array asociativo**

Un array asociativo almacena datos usando claves de tipo string en lugar de índices numéricos. Esto permite dar contexto al contenido del array, como si fuera una especie de diccionario.

**Características clave:**

* Las claves son nombres personalizados.
* Facilita la representación de entidades con propiedades (como una persona con nombre y edad).

// Array asociativo (Alex Benites)

$persona = ["nombre" => "Laura", "edad" => 25];

**Operadores**

**Operadores Aritméticos en PHP**

**+ : Suma**

**A group of symbols on a white background

AI-generated content may be incorrect.**

**- : Resta**

**A close-up of a number

AI-generated content may be incorrect.**

**\*:Multiplicación**

**A group of black symbols

AI-generated content may be incorrect.**

**/ : División**

**A close-up of a math equation

AI-generated content may be incorrect.**

**% : Módulo (resto de división)**

**A close-up of symbols

AI-generated content may be incorrect.**

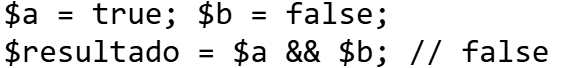
**\*\* : Potenciación**

**A group of black symbols

AI-generated content may be incorrect.**

**Operadores Lógicos en PHP**

**&& : AND lógico**

****

**|| : OR lógico**

**A black text with a white background

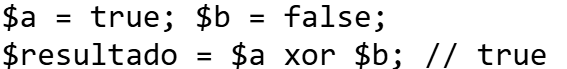
AI-generated content may be incorrect.**

**! : Negación lógica (NOT)**

**A black text with a white background

AI-generated content may be incorrect.**

**xor : XOR lógico (exclusivo)**

****

**Operadores de Asignación en PHP**

**= : Asignación simple**

****

**+= : Suma y asigna**

**A black line on a white background

AI-generated content may be incorrect.**

**-= : Resta y asigna**

**A close-up of a white background

AI-generated content may be incorrect.**

**\*= : Multiplica y asigna**

**A close-up of a sign

AI-generated content may be incorrect.**

**/= : Divide y asigna**

**A black lines on a white background

AI-generated content may be incorrect.**

**%= : Módulo y asigna**

**A close up of a symbol

AI-generated content may be incorrect.**

**\*\*= : Potencia y asigna**

**A black lines on a white background

AI-generated content may be incorrect.**

**Palabras Reservadas**

**if**  
Se utiliza para realizar una condición. Si la condición es verdadera, ejecuta el bloque de código asociado.

**else**  
Se usa junto con if para ejecutar un bloque de código cuando la condición del if es falsa.

**elseif**  
Permite verificar múltiples condiciones dentro de una estructura condicional.

**while**  
Crea un ciclo que se ejecuta mientras la condición sea verdadera.

**for**  
Bucle que se ejecuta un número determinado de veces.

**Foreach** [4]  
Bucle especialmente diseñado para recorrer arrays o colecciones.

**function**  
Declara una función, un bloque de código reutilizable.

**return**  
Devuelve un valor desde una función y termina su ejecución.

**class**  
Define una clase para la programación orientada a objetos.

**public / private / protected**  
Modificadores de acceso que determinan la visibilidad de propiedades o métodos en una clase.

**new**  
Crea una nueva instancia de una clase.

**try / catch / finally**  
Estructura para manejo de excepciones. El código dentro de try se prueba y si ocurre un error, se captura en catch. finally se ejecuta siempre, con o sin error.

**switch**  
Estructura condicional que compara una variable contra varios valores posibles.

**case**  
Define un caso dentro de un switch.

**break**  
Termina la ejecución de un ciclo o una estructura switch.

**include / require**  
Incluyen y ejecutan un archivo PHP externo dentro del script actual. require produce un error fatal si no encuentra el archivo, include solo una advertencia.

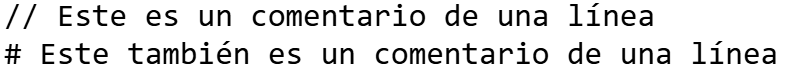
**echo / print**  
Imprime texto o variables en la salida.

**empty**  
Verifica si una variable está vacía.

**exit / die**  
Termina la ejecución del script inmediatamente.

**Comentarios**

**Comentario de una línea:**  
Se usa para escribir comentarios cortos que ocupan solo una línea. Hay dos formas de hacerlo:



**Comentario de múltiples líneas:**

Se usa para escribir comentarios más largos que pueden ocupar varias líneas.

A black text on a white background

AI-generated content may be incorrect.

**Delimitadores**

**Punto y coma ( ; )**  
Se usa para finalizar una instrucción o sentencia en PHP. Es obligatorio después de cada línea de código que ejecuta una acción.

echo "Hola Mundo";

**Llaves ( { } )**  
Delimitan bloques de código, como los de funciones, clases, estructuras condicionales (if, else) y ciclos (for, while).

A close up of a number

AI-generated content may be incorrect.

**Paréntesis ( ( ) )**  
Se usan para encerrar condiciones en estructuras condicionales, parámetros de funciones y llamadas a funciones.

A white background with black text

AI-generated content may be incorrect.

**Corchetes ( [ ] )**  
Se usan para acceder a elementos de arrays o definir arrays.



**3. Analizador Sintáctico**

**Descripción General**

* El analizador sintáctico verificará que las expresiones y estructuras de control de PHP sigan una secuencia lógica y sintácticamente correcta, evaluando la apertura y cierre de bloques, secuencias de operaciones, y estructuras de control.
* PHP tiene una sintaxis flexible, mezcla tipado dinámico con características de tipado fuerte cuando se usa declare(strict\_types=1);.

**Reglas Gramaticales**

* **Declaración de Variables:** La declaración de variables en PHP consiste en crear identificadores precedidos por $ para almacenar datos. PHP asigna el tipo de dato automáticamente según el valor.

$nombre = "Juan";

* **Expresiones Aritméticas:** Validación de operaciones como suma, resta, multiplicación, división, y la precedencia de operadores en PHP.

$total = $a + $b \* $c;

* **Expresiones Booleanas**: Comprobación de expresiones lógicas y operadores de comparación entre valores y variables.

if ($a > 10 && $b != 5) { ... }

* **Estructuras de Datos (ED)**: El analizador reconocerá la estructura y sintaxis para arrays y objetos.

$frutas = array("manzana", "naranja", "pera");

* **Estructuras de Control**: Reglas de sintaxis para if, else, while, for, switch.

for ($i = 0; $i < 10; $i++) { ... }

* **Declaraciones de Funciones**: Comprobación de la correcta declaración y llamado de funciones en PHP.

function saludar($nombre) {

return "Hola, " . $nombre;

}

* **Impresión y solicitud de datos:** Uso de echo y print para salida de datos, y fgets(STDIN) para entradas de consola.

echo "Ingrese su nombre:";

$nombre = readline();

* **Clases y Objetos**: Validación de la definición de clases, instanciación de objetos, y herencia en PHP.

class Persona {

public $nombre;

function \_\_construct($nombre) {

$this->nombre = $nombre;

}

}

$p = new Persona("Luis");

**4. Analizador Semántico**

**Descripción General**

* Este analizador garantizará la coherencia lógica de las operaciones y estructuras de PHP, verificando la correcta asignación de tipos, compatibilidad de valores, uso de estructuras de control, así como retornos adecuados en funciones.

**Reglas Semánticas**

Escoger por cada integrante 2 ejemplos de reglas semánticas. De las siguientes:

* **Tipo de Datos**: Verificación de tipos de datos en operaciones y asignaciones para evitar errores de tipo.

Ejemplo válido:

$x = "5" + 3; // permitido, pero se convierte a int implícitamente

Ejemplo no válido:

$edad = "veinticinco";

$calculo = $edad / 5; // No tiene sentido dividir una cadena no numérica

* **Compatibilidad de Tipos**: Garantizar que las conversiones entre tipos de datos sean válidas en contextos específicos.

Ejemplo válido:

$precio = 12.5;

$cantidad = 2;

$total = $precio \* $cantidad;

Ejemplo no válido:

$activo = true;

$total = $activo \* "hola"; // Resultado incoherente semánticamente

* **Retorno de Funciones**: Comprobación de que el tipo de retorno de una función coincide con lo esperado en su contexto de uso.

Ejemplo válido:

function suma(int $a, int $b): int {

return $a + $b;

}

Ejemplo no válido:

function obtenerNombre(): int {

return "Carlos"; // Retorna un string en vez de un int

}

* **Uso Correcto de Estructuras de Control**: Validación de que instrucciones como break y continue se encuentren correctamente ubicadas dentro de bucles o estructuras de control.

Ejemplo valido:

for ($i = 0; $i < 10; $i++) {

if ($i === 5) {

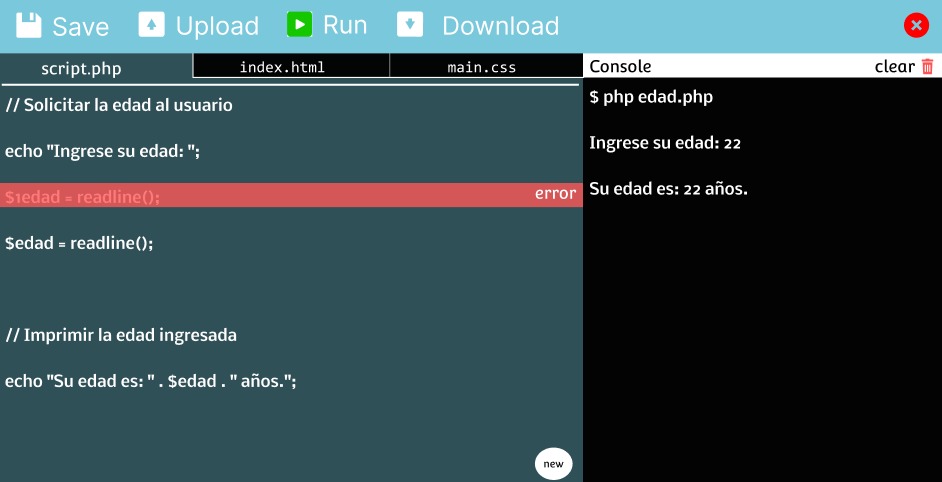
break;

}

}

Ejemplo no válido:

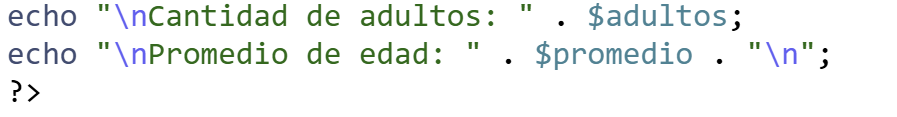
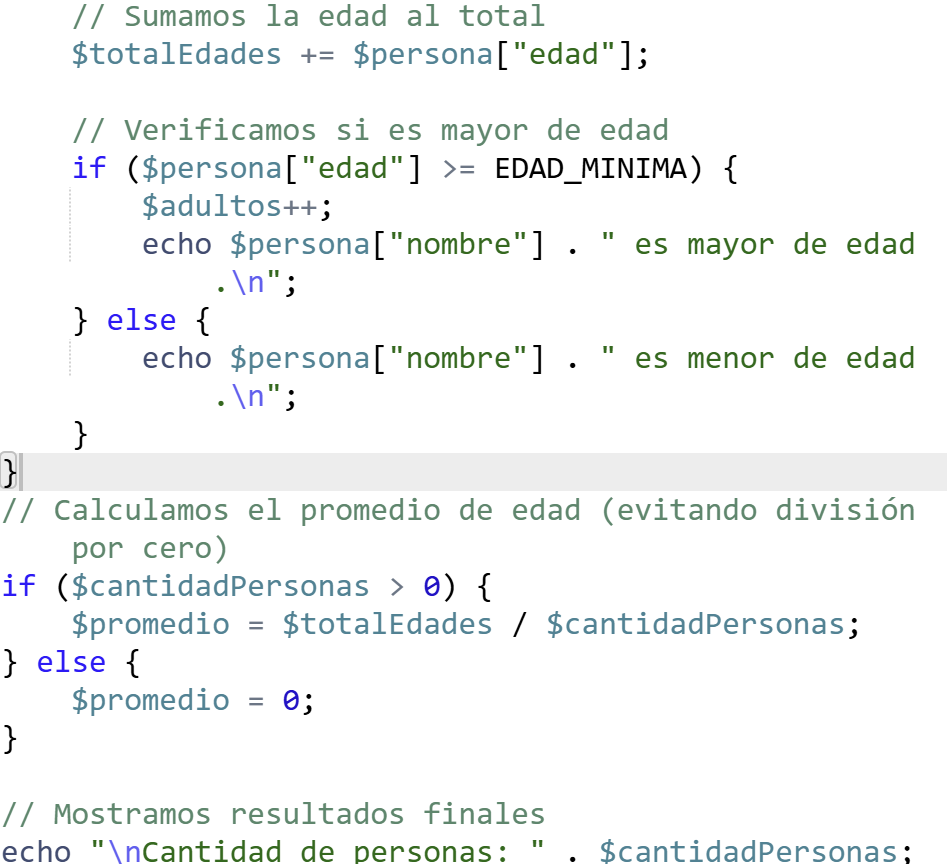
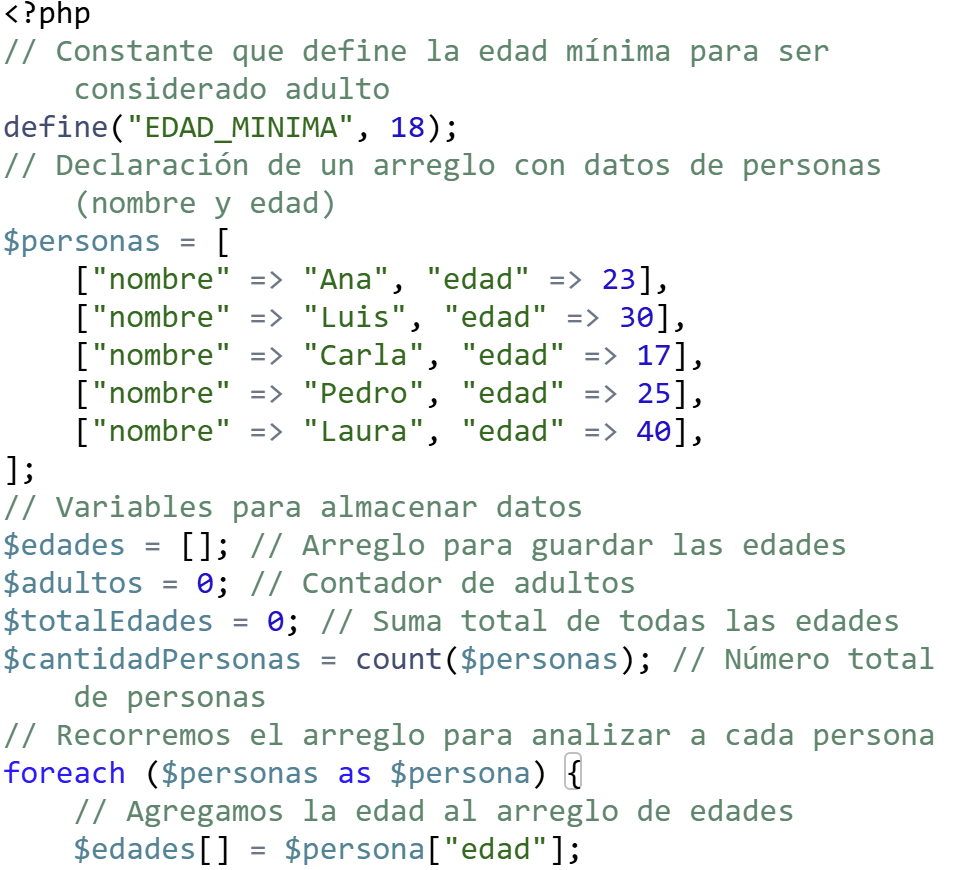
break; // Error: break fuera de un bucle o switch

**5. Diseño Preliminar** [5]

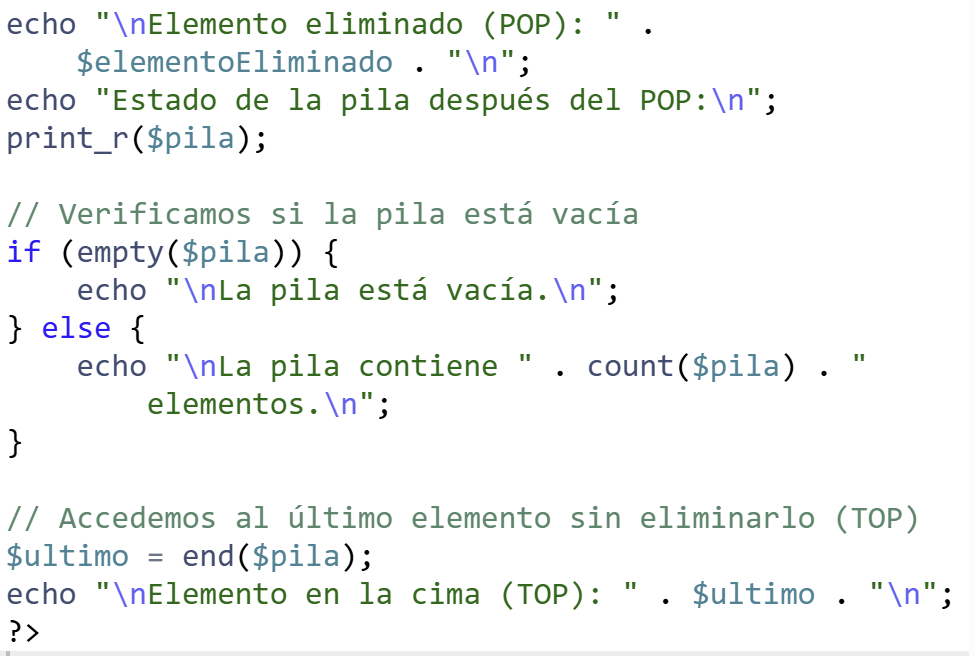
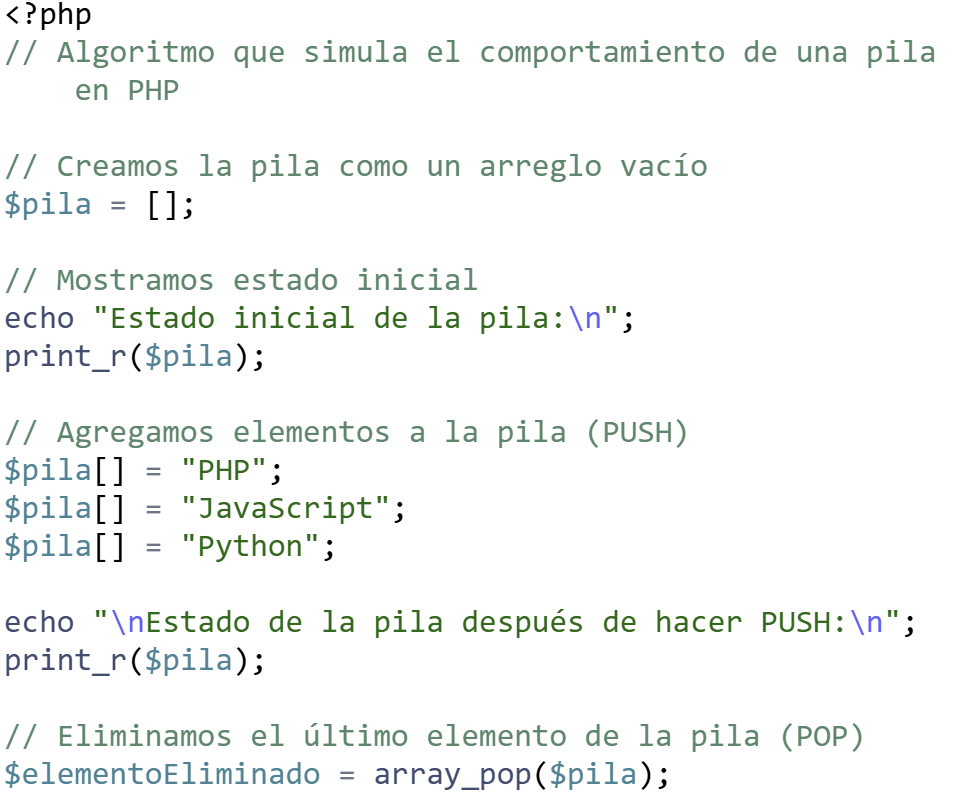
**6. Pruebas y Validación**

**Casos de Prueba**

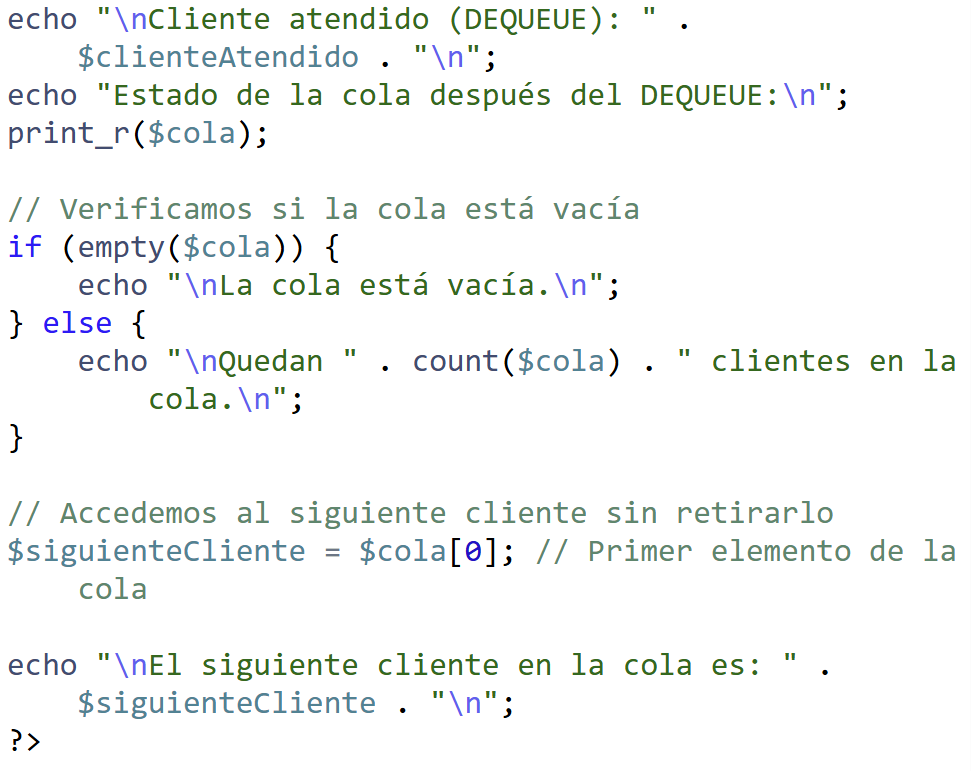
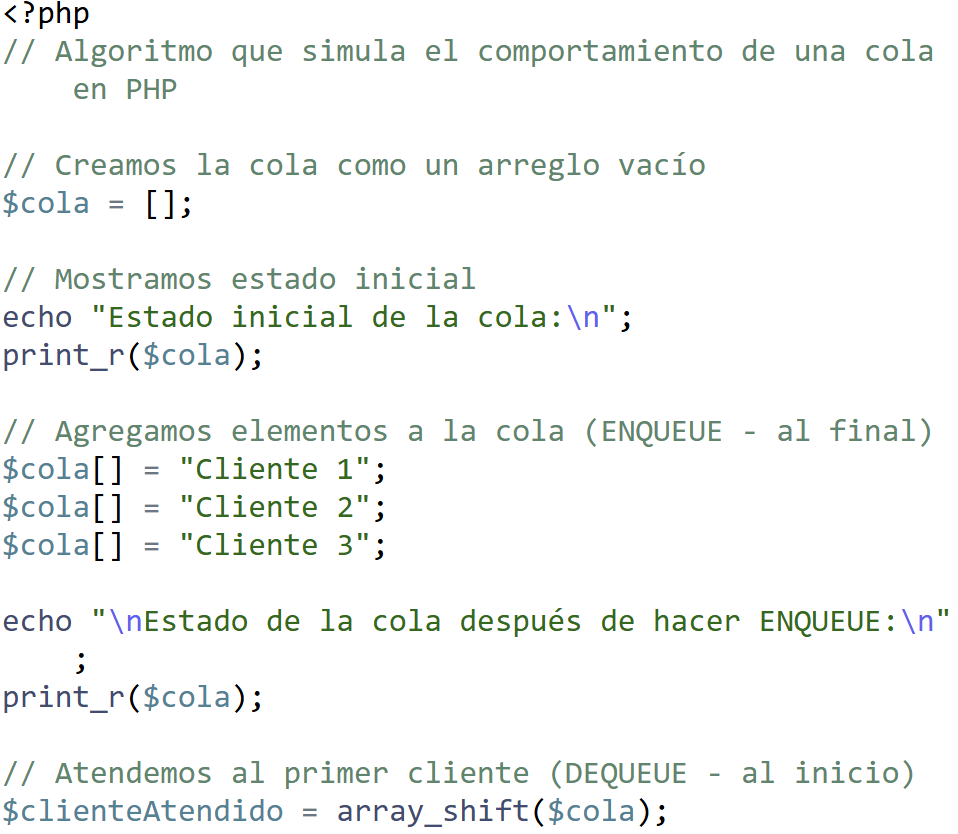
Codigo1- Daniel Zavala



Código 2 – Alex Benítez



Código 3 – Nehemías Lindao



# **Referencias**

|  |  |
| --- | --- |
| [1] | PHP Documentation Group, «Historia de php,» [En línea]. Available: https://www.php.net/manual/es/history.php.php. [Último acceso: 08 06 2025]. |
| [2] | PHP, «Manual de PHP,» [En línea]. Available: https://www.php.net/manual/es/. [Último acceso: 08 06 2025]. |
| [3] | freeCodeCamp, «Los mejores ejemplos de PHP,» 21 03 2023. [En línea]. Available: https://www.freecodecamp.org/espanol/news/los-mejores-ejemplos-de-php/. [Último acceso: 08 06 2025]. |
| [4] | MoureDev, «Curso de PHP Desde Cero,» 2024. [En línea]. Available: https://www.youtube.com/watch?v=nPCJAx5c1uE. [Último acceso: 08 06 2025]. |
| [5] | Group-10, «Figma,» Prototipo de analizador lexico, semantico y sintactico en PHP, 08 06 2025. [En línea]. Available: https://www.figma.com/design/lnL8KOMUirTuvPEnVzxLiq/Prototype-LP?node-id=0-1&t=7s1mLtMG7EC205g7-1. |