Tabla de contenido

Variables.

Identificadores de Variables

En PHP, los identificadores de variables deben seguir ciertas reglas:

1. **Comienzo**: Deben comenzar con el símbolo $ seguido de una letra o un guion bajo (_).
2. **Contenido**: Pueden contener letras, números y guiones bajos.
3. **Sensibilidad a Mayúsculas**: Son sensibles a mayúsculas y minúsculas ($edad y $Edad son diferentes).
4. **Evitar Palabras Reservadas**: No deben usar palabras reservadas de PHP como function, class, etc.

Declaración de Variables

Las variables en PHP se declaran simplemente asignándoles un valor.
No es necesario especificar el tipo de dato, ya que PHP es un lenguaje de tipado dinámico.

```php
<?php
$nombre = "Juan"; // String
$edad = 25; // Entero
$salario = 45000.50; // Float
$esEmpleado = true; // Booleano
?>
```

Tipos de datos en PHP.

PHP soporta los siguientes tipos de datos:

STRING: secuencia de caracteres.

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

INTEGER: números enteros.

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

FLOAT: números con decimales.

```php
<?php
$decimal = 123.45;
?>
```

BOOLEAN: valores booleanos, TRUE o FALSE.

```php
<?php
$esVerdadero = true;
?>
```

ARRAY: lista de valores.

```php
<?php
$frutas = array("Manzana", "Banana", "Cereza");
?>
```

NULL: representa una variable sin valor.

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

Tipado dinámico.

PHP asigna el tipo de dato automáticamente basado en el valor asignado. Esto permite flexibilidad, pero también puede llevar a errores si no se maneja con cuidado.

```php
<?php
$variable = "Hola"; // String
$variable = 10; // Integer
?>
```

Variables variables.

Una variable variable toma el valor de una variable y lo trata como el nombre de otra variable. Esto se logra utilizando dos signos pesos ($$).

```php
<?php
$a = 'hola';
$$a = 'mundo';

echo $a; // Salida: hola
echo $$a; // Salida: mundo
echo $hola; // Salida: mundo
?>
```

En este ejemplo:

- $a contiene el valor 'hola'.
- $$a se convierte en $hola y se le asigna el valor 'mundo'.

Este tipo de variables se suelen utilizar mucho en el procesamiento eficiente de datos contenidos en arreglos y en la generación de variables de forma dinámica dentro de funciones.

Función VAR_DUMP

Se utiliza para obtener el tipo y valor de una variable.

```php
<?php
$variable = 10.5;
var_dump($variable); // Salida: float(10.5)
?>
```

Constantes.

Una constante es un identificador (nombre) para un valor simple.
Una vez definida, su valor no puede cambiar durante la ejecución del script.
Las constantes son útiles para almacenar valores que no deben ser modificados, como configuraciones o valores fijos.

Declaración.

Las constantes en PHP se pueden declarar de dos maneras:

1.Usando define():

```php
<?php
define("NOMBRE_CONSTANTE", "valor");
echo NOMBRE_CONSTANTE; // Salida: valor
?>
```

2.Usando la palabra clave const (solo dentro de clases):

```php
<?php
class MiClase {
    const MI_CONSTANTE = 'valor';
}
echo MiClase::MI_CONSTANTE; // Salida: valor
?>
```

Reglas para Nombres de Constantes

- Deben comenzar con una letra o un guion bajo (_).
- Pueden contener letras, números y guiones bajos.

- Son sensibles a mayúsculas y minúsculas (CONSTANTE y constante son diferentes).
- Por convención, los nombres de constantes suelen escribirse en mayúsculas

Acceso Global

Las constantes tienen un alcance global y pueden ser accedidas desde cualquier parte del script, sin importar el ámbito en el que fueron definidas.

Constantes Predefinidas

PHP tiene varias constantes predefinidas que proporcionan información sobre el entorno de ejecución, como PHP_VERSION, PHP_OS, y PHP_EOL.

Constantes Mágicas

PHP también tiene constantes mágicas que cambian dependiendo de dónde se utilicen. Algunas de las más comunes son:

- __LINE__: El número de línea actual en el script.
- __FILE__: La ruta completa y el nombre del archivo.
- __DIR__: El directorio del archivo.
- __FUNCTION__: El nombre de la función.
- __CLASS__: El nombre de la clase.
- __METHOD__: El nombre del método de la clase.
- __NAMESPACE__: El nombre del espacio de nombres actual

Operadores aritméticos.

Los operadores aritméticos en PHP se utilizan para realizar operaciones matemáticas comunes.

Acá están los operadores aritméticos básicos:

1. **Adición (+)**: Suma dos valores.

```php
<?php
$a = 10;
$b = 5;
$resultado = $a + $b;
echo $resultado; // Salida: 15
?>
```

2. **Sustracción (-)**: Resta un valor de otro.

```php
<?php
$a = 10;
$b = 5;
$resultado = $a - $b;
echo $resultado; // Salida: 5
?>
```

3. **Multiplicación (*)**: Multiplica dos valores.

```php
<?php
$a = 10;
$b = 5;
$resultado = $a * $b;
echo $resultado; // Salida: 50
?>
```

4. **División (/)**: Divide un valor por otro.

```php
<?php
$a = 10;
$b = 5;
$resultado = $a / $b;
echo $resultado; // Salida: 2
?>
```

5. **Módulo (%)**: Devuelve el resto de una división.

```php
<?php
$a = 10;
$b = 3;
$resultado = $a % $b;
echo $resultado; // Salida: 1
?>
```

6. **Exponenciación (**)**: Eleva un número a la potencia de otro.

```php
<?php
$a = 2;
$b = 3;
$resultado = $a ** $b;
echo $resultado; // Salida: 8
?>
```

Uso de Operadores Aritméticos con Asignación

PHP permite combinar operadores aritméticos con el operador de asignación (=) para realizar operaciones y asignar el resultado en una sola expresión.

Adición y Asignación (+=)

```php
<?php
$a = 10;
$a += 5; // Equivalente a $a = $a + 5
echo $a; // Salida: 15
?>
```

Sustracción y Asignación (-=)

```php
<?php
$a = 10;
$a -= 5; // Equivalente a $a = $a - 5
echo $a; // Salida: 5
?>
```

Multiplicación y Asignación (*=)

```php
<?php
$a = 10;
$a *= 5; // Equivalente a $a = $a * 5
echo $a; // Salida: 50
?>
```

División y Asignación (/=)

```php
<?php
$a = 10;
$a /= 5; // Equivalente a $a = $a / 5
echo $a; // Salida: 2
?>
```

Módulo y Asignación (%=)

```php
<?php
$a = 10;
$a %= 3; // Equivalente a $a = $a % 3
echo $a; // Salida: 1
?>
```

Precedencia de Operadores

La precedencia de operadores determina el orden en que se evalúan las expresiones.
Los operadores aritméticos tienen una precedencia específica que puede afectar el resultado de las operaciones.

```php
<?php
$resultado = 10 + 5 * 2; // Multiplicación se evalúa primero
echo $resultado; // Salida: 20
?>
```

Para cambiar el orden de evaluación, se pueden usar paréntesis.

```php
<?php
$resultado = (10 + 5) * 2; // Suma se evalúa primero
echo $resultado; // Salida: 30
?>
```

Operaciones con Tipos de Datos Mixtos

PHP permite realizar operaciones aritméticas con diferentes tipos de datos, como enteros y flotantes. Sin embargo, es importante tener en cuenta cómo PHP maneja la conversión de tipos.
Para estos casos siempre se mantendrá el tipo de dato que mas precisión aporte al resultado.

```php
<?php
$a = 10; // Entero
$b = 5.5; // Flotante
$resultado = $a + $b;
echo $resultado; // Salida: 15.5
?>
```

Operadores relacionales.

Los operadores relacionales, también conocidos como operadores de comparación, se utilizan para comparar dos valores.
El resultado de una comparación es un valor booleano (true o false).

Lista de Operadores Relacionales

1. **Igual (==)**: Devuelve true si los valores son iguales.

```php
<?php
$a = 5;
$b = 5;
var_dump($a == $b); // Salida: bool(true)
?>
```

2. **Idéntico (===)**: Devuelve true si los valores y los tipos son iguales.

```php
<?php
$a = 5;
$b = "5";
var_dump($a === $b); // Salida: bool(false)
?>
```

3. **Diferente (!= o <>)**: Devuelve true si los valores son diferentes.

```php
<?php
$a = 5;
$b = 3;
var_dump($a != $b); // Salida: bool(true)
?>
```

4. **No idéntico (!==)**: Devuelve true si los valores o los tipos son diferentes.

```php
<?php
$a = 5;
$b = "5";
var_dump($a !== $b); // Salida: bool(true)
?>
```

5. **Mayor que (>)**: Devuelve true si el valor de la izquierda es mayor que el de la derecha.

```php
<?php
$a = 5;
$b = 3;
var_dump($a > $b); // Salida: bool(true)
?>
```

6. **Menor que (<)**: Devuelve true si el valor de la izquierda es menor que el de la derecha.

```php
<?php
$a = 5;
$b = 3;
var_dump($a < $b); // Salida: bool(false)
?>
```

7. **Mayor o igual que (>=)**: Devuelve true si el valor de la izquierda es mayor o igual que el de la derecha.

```php
<?php
$a = 5;
$b = 5;
var_dump($a >= $b); // Salida: bool(true)
?>
```

8. **Menor o igual que (<=)**: Devuelve true si el valor de la izquierda es menor o igual que el de la derecha.

```php
<?php
$a = 5;
$b = 5;
var_dump($a <= $b); // Salida: bool(true)
?>
```

9. **Nave espacial (<=>)**: Devuelve -1, 0, o 1 si el valor de la izquierda es menor, igual o mayor que el de la derecha (introducido en PHP 7).

```php
<?php
$a = 5;
$b = 3;
echo $a <=> $b; // Salida: 1 (porque $a es mayor que $b)
?>
```

Operadores lógicos.

Los operadores lógicos se utilizan para combinar declaraciones condicionales y evaluar expresiones booleanas.
Los operadores lógicos en PHP son:

1. **AND (&& y and)**: Devuelve true si ambas expresiones son verdaderas.

```php
<?php
$a = true;
$b = false;
var_dump($a && $b); // Salida: bool(false)
var_dump($a and $b); // Salida: bool(false)
?>
```

2. **OR (|| y or)**: Devuelve true si al menos una de las expresiones es verdadera.

```php
<?php
$a = true;
$b = false;
var_dump($a || $b); // Salida: bool(true)
var_dump($a or $b); // Salida: bool(true)
?>
```

3. **XOR (**xor**)**: Devuelve true si una y solo una de las expresiones es verdadera.

```php
<?php
$a = true;
$b = false;
var_dump($a xor $b); // Salida: bool(true)
?>
```

4. **NOT (!)**: Invierte el valor de una expresión booleana.

```php
<?php
$a = true;
var_dump(!$a); // Salida: bool(false)
?>
```

Precedencia de Operadores

La precedencia de operadores determina el orden en que se evalúan las expresiones. Los operadores lógicos && y || tienen una precedencia mayor que and y or.

```php
<?php
$a = false || true; // Equivalente a $a = (false || true)
$b = false or true; // Equivalente a ($b = false) or true
var_dump($a); // Salida: bool(true)
var_dump($b); // Salida: bool(false)
?>
```

Cortocircuito

Los operadores lógicos en PHP utilizan cortocircuito, lo que significa que la evaluación se detiene tan pronto como se determina el resultado.

```php
<?php
function foo() {
    echo "foo() llamado\n";
    return true;
}

$a = false && foo(); // foo() no se llama
$b = true || foo(); // foo() no se llama
$c = false and foo(); // foo() no se llama
$d = true or foo(); // foo() no se llama
?>
```

Uso de Paréntesis

Para evitar confusiones y errores de precedencia, es recomendable usar paréntesis para agrupar expresiones lógicas.

```php
<?php
$a = true;
$b = false;
$c = true;
if (($a && $b) || $c) {
    echo "La condición es verdadera.";
} else {
    echo "La condición es falsa.";
}
// Salida: La condición es verdadera.
?>
```

Estructura de control FOR.

El bucle FOR en PHP se utiliza para ejecutar un bloque de código un número determinado de veces.
La sintaxis básica es la siguiente:

```php
for (inicialización; condición; incremento/decremento) {
    // Código a ejecutar
}
```

- **Inicialización**: Se ejecuta una vez al comienzo del bucle. Aquí se suele inicializar una variable de control.
- **Condición**: Se evalúa antes de cada iteración. Si la condición es verdadera, el bucle continúa; si es falsa, el bucle termina.
- **Incremento/Decremento**: Se ejecuta al final de cada iteración. Aquí se suele incrementar o decrementar la variable de control.

Ejemplo básico:

```php
for ($i = 0; $i < 10; $i++) {
    echo "El valor de i es: $i\n";
}
```

En este ejemplo, el bucle se ejecuta 10 veces, imprimiendo los valores del 0 al 9.

Bucles anidados.

Podemos anidar uno o mas bucles dentro de otros.

```php
for ($i = 0; $i < 3; $i++) {
    for ($j = 0; $j < 3; $j++) {
        echo "i = $i, j = $j\n";
    }
}
```

Control de flujo.

Podemos usar break y continue para controlar el flujo dentro del bucle:

- break: Termina el bucle inmediatamente.
- continue: Salta a la siguiente iteración del bucle.

```php
for ($i = 0; $i < 10; $i++) {
    if ($i == 5) {
        break; // Termina el bucle cuando i es 5
    }
    if ($i % 2 == 0) {
        continue; // Salta las iteraciones donde i es par
    }
    echo "El valor de i es: $i\n";
}
```

Uso en arreglos.

El bucle FOR es útil para iterar sobre arrays cuando se conoce el número de elementos:

```php
$array = [1, 2, 3, 4, 5];
for ($i = 0; $i < count($array); $i++) {
    echo "Elemento $i: " . $array[$i] . "\n";
}
```

Manipulación compleja de variables.

Podemos manipular múltiples variables en la sección de inicialización y de incremento/decremento:

```php
for ($i = 0, $j = 10; $i < 10; $i++, $j--) {
    echo "i = $i, j = $j\n";
}
```

Bucles invertidos.

Podemos usar un bucle FOR para contar hacia atrás.

```php
for ($i = 10; $i > 0; $i--) {
    echo "Cuenta regresiva: $i\n";
}
```

Consideraciones de rendimiento.

- **Evitar recalcular la longitud del array en cada iteración**: En lugar de for ($i = 0; $i < count($array); $i++), usar for ($i = 0, $len = count($array); $i < $len; $i++).
- **Usar bucles** foreach **para iterar arrays**: Aunque el bucle for es potente, foreach es más eficiente y legible para iterar arrays.

Estructura de control IF.

La estructura IF en PHP se utiliza para ejecutar un bloque de código solo si una condición es verdadera.
La sintaxis básica es:

```
if (condición) {
    // Código a ejecutar si la condición es verdadera
}
```

Ejemplo de uso:

```
$edad = 20;

if ($edad >= 18) {
    echo "Eres mayor de edad.";
}
```

En este ejemplo, el mensaje “Eres mayor de edad.” se imprimirá solo si la variable $edad es mayor o igual a 18.

Estructura IF/ELSE.

Para manejar condiciones alternativas, se puede usar ELSE:

```
if (condición) {
    // Código a ejecutar si la condición es verdadera
} else {
    // Código a ejecutar si la condición es falsa
}
```

Ejemplo de uso.

```php
$edad = 16;

if ($edad >= 18) {
    echo "Eres mayor de edad.";
} else {
    echo "Eres menor de edad.";
}
```

Estructura IF/ELSEIF/ELSE.

Para manejar múltiples condiciones, debemos usar ELSEIF:

```php
if (condición1) {
    // Código a ejecutar si la condición1 es verdadera
} elseif (condición2) {
    // Código a ejecutar si la condición2 es verdadera
} else {
    // Código a ejecutar si ninguna de las condiciones anteriores es verdadera
}
```

Ejemplo de uso:

```php
$nota = 85;

if ($nota >= 90) {
    echo "Excelente";
} elseif ($nota >= 75) {
    echo "Bueno";
} else {
    echo "Necesita mejorar";
}
```

Estructuras IF anidadas.

Podemos anidar estructuras IF dentro de otras:

```php
$edad = 20;
$es_estudiante = true;

if ($edad >= 18) {
    if ($es_estudiante) {
        echo "Eres un estudiante adulto.";
    } else {
        echo "Eres un adulto.";
    }
} else {
    echo "Eres menor de edad.";
}
```

Uso de operadores lógicos.

Se pueden combinar múltiples condiciones usando los operadores lógicos.

```php
$edad = 20;
$es_estudiante = true;

if ($edad >= 18 && $es_estudiante) {
    echo "Eres un estudiante adulto.";
}
```

Consideraciones de Rendimiento

- **Evaluación de Cortocircuito**: PHP utiliza evaluación de cortocircuito, lo que significa que, en una expresión lógica, si la

primera condición es suficiente para determinar el resultado, las condiciones restantes no se evalúan.

- **Legibilidad del Código**: Aunque puedes anidar muchas estructuras IF, es importante mantener el código legible. Considera refactorizar el código en funciones si se vuelve demasiado complejo.

Estructura de control WHILE.

El bucle WHILE en PHP se utiliza para ejecutar un bloque de código repetidamente mientras una condición sea verdadera.
La sintaxis básica es:

```
while (condición) {
    // Código a ejecutar mientras la condición sea verdadera
}
```

Ejemplo de uso:

```
$i = 0;

while ($i < 10) {
    echo "El valor de i es: $i\n";
    $i++;
}
```

En este ejemplo, el bucle se ejecuta mientras $i sea menor que 10, imprimiendo los valores del 0 al 9.

Estructura DO WHILE.

El bucle DO-WHILE es similar al WHILE, pero garantiza que el bloque de código se ejecute al menos una vez, independientemente de si la condición es verdadera o falsa al inicio.

```
do {
    // Código a ejecutar al menos una vez
} while (condición);
```

Ejemplo de uso:

```
$i = 0;

do {
    echo "El valor de i es: $i\n";
    $i++;
} while ($i < 10);
```

Estructuras WHILE anidadas.

Podemos anidar bucles WHILE dentro de otros bucles WHILE:

```
$i = 0;

while ($i < 3) {
    $j = 0;
    while ($j < 3) {
        echo "i = $i, j = $j\n";
        $j++;
    }
    $i++;
}
```

Control de flujo.

Podemos usar break y continue para controlar el flujo dentro del bucle:

- break: Termina el bucle inmediatamente.
- continue: Salta a la siguiente iteración del bucle.

```php
$i = 0;

while ($i < 10) {
    if ($i == 5) {
        break; // Termina el bucle cuando i es 5
    }
    if ($i % 2 == 0) {
        $i++;
        continue; // Salta las iteraciones donde i es par
    }
    echo "El valor de i es: $i\n";
    $i++;
}
```

Uso de arreglos.

El bucle WHILE es útil para iterar sobre arrays cuando no conocemos el número exacto de elementos o cuando necesitamos manipular el índice manualmente:

```php
$array = [1, 2, 3, 4, 5];
$i = 0;

while ($i < count($array)) {
    echo "Elemento $i: " . $array[$i] . "\n";
    $i++;
}
```

Aspectos avanzados, condiciones complejas.

Se puede usar condiciones complejas dentro del WHILE utilizando operadores lógicos como && (AND), || (OR) y ! (NOT):

```php
$i = 0;
$j = 10;

while ($i < 10 && $j > 0) {
    echo "i = $i, j = $j\n";
    $i++;
    $j--;
}
```

Bucles invertidos.

Se puede usar la estructura WHILE para contar hacia atrás.

```php
$i = 10;

while ($i > 0) {
    echo "Cuenta regresiva: $i\n";
    $i--;
}
```

Consideraciones de Rendimiento

- **Evitar bucles infinitos**: Asegurarse de que la condición del WHILE eventualmente se vuelva falsa para evitar bucles infinitos que pueden colgar nuestro script.
- **Optimización de Condiciones**: Colocar las condiciones más probables para salir del bucle al principio para mejorar el rendimiento.

Funciones.

Una función en PHP es un bloque de código reutilizable que realiza una tarea específica.

Las funciones permiten organizar el código de manera más estructurada, facilitando su mantenimiento y reutilización en diferentes partes de un programa.
La sintaxis básica para definir una función en PHP es:

```php
function nombreDeLaFuncion($parametro1, $parametro2, ...) {
    // Código a ejecutar
    return $valorDevuelto; // Opcional
}
```

- function: Palabra clave para definir una función.
- nombreDeLaFuncion: Nombre de la función, que debe ser único y descriptivo.
- $parametro1, $parametro2, ...: Parámetros opcionales que la función puede recibir.
- return: Palabra clave para devolver un valor desde la función (opcional).

Ejemplo de uso:

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

echo saludar("Juan"); // Imprime "Hola, Juan!"
```

Uso de Parámetros y Valores por Defecto

Podemos definir parámetros con valores por defecto:

```php
function saludar($nombre = "Mundo") {
    return "Hola, $nombre!";
}

echo saludar(); // Imprime "Hola, Mundo!"
echo saludar("Juan"); // Imprime "Hola, Juan!"
```

Funciones con Parámetros por Referencia

También podemos pasar parámetros por referencia para que la función pueda modificar el valor de la variable original:

```php
function incrementar(&$valor) {
    $valor++;
}

$numero = 5;
incrementar($numero);
echo $numero; // Imprime 6
```

Cuando anteponemos el carácter & al parámetro, le damos permiso a la función para que modifique la variable original que se usó para invocarla, en nuestro ejemplo, es la variable número.
Si le quitamos el & y volvemos a ejecutar el código, el pasaje del parámetro ya no es por referencia, es por valor, y en ese caso veremos que la función ya no puede modificar la variable número.

Funciones que retornan más de un valor.

En PHP, una función no puede retornar múltiples valores directamente, pero puedes lograr un resultado similar utilizando arrays u objetos.

Veamos cómo se hace.

Usando Arrays
La forma más común de retornar múltiples valores es empaquetarlos en un array y luego descomponerlos al recibirlos.

```php
function obtenerDatos() {
    $nombre = "Juan";
    $edad = 25;
    $ciudad = "Buenos Aires";
    return array($nombre, $edad, $ciudad);
}

// Descomponer el array en variables individuales
list($nombre, $edad, $ciudad) = obtenerDatos();

echo "Nombre: $nombre, Edad: $edad, Ciudad: $ciudad";
```

En este ejemplo, la función obtenerDatos retorna un array con tres elementos.
Luego, usamos LIST para descomponer el array en variables individuales.

Usando Arrays Asociativos
Otra opción es retornar un array asociativo, lo que puede hacer que el código sea más legible.

```php
function obtenerDatos() {
    return array(
        "nombre" => "Juan",
        "edad" => 25,
        "ciudad" => "Buenos Aires"
    );
}

$datos = obtenerDatos();

echo "Nombre: " . $datos["nombre"] . ", Edad: " . $datos["edad"] . ", Ciudad: " . $datos["ciudad"];
```

Funciones nativas de PHP.

Funciones de manipulación de cadenas de texto.

STRLEN: Devuelve la longitud de una cadena.

```php
$cadena = "Hola Mundo";
echo strlen($cadena); // Imprime 10
```

STRPOS: Encuentra la posición de la primera aparición de una subcadena.

```php
$cadena = "Hola Mundo";
echo strpos($cadena, "Mundo"); // Imprime 5
```

STR_REPLACE: Reemplaza todas las apariciones de una subcadena con otra.

```php
$cadena = "Hola Mundo";
echo str_replace("Mundo", "PHP", $cadena); // Imprime "Hola PHP"
```

Funciones del tipo IS_*.

PHP proporciona una serie de funciones is_* para verificar tipos específicos de variables. Aquí tienes algunas de las más comunes:

IS_ARRAY: Verifica si una variable es un array.

```php
$variable = [1, 2, 3];
echo is_array($variable); // Imprime 1 (true)
```

IS_BOOL: Verifica si una variable es un Booleano.

```php
$variable = true;
echo is_bool($variable); // Imprime 1 (true)
```

IS_INT: Verifica si una variable es un entero.

```php
$variable = 123;
echo is_int($variable); // Imprime 1 (true)
```

IS_FLOAT: Verifica si una variable es un número de punto flotante.

```php
$variable = 1.23;
echo is_float($variable); // Imprime 1 (true)
```

IS_STRING: Verifica si una variable es una cadena de texto.

```php
$variable = "Hola";
echo is_string($variable); // Imprime 1 (true)
```

IS_NULL: Verifica si una variable es NULL.

```php
$variable = null;
echo is_null($variable); // Imprime 1 (true)
```

Sesiones.

Una sesión en PHP permite almacenar datos específicos del usuario en el servidor y mantenerlos disponibles a través de diferentes páginas de un sitio web.
A diferencia de las cookies, que se almacenan en el lado del cliente, las sesiones son más seguras ya que los datos se almacenan en el servidor.

La web utiliza el protocolo HTTP, que, entre sus características, está la de no guardar ningún estado, es decir, no conserva ninguna información sobre conexiones anteriores.
No sabe qué página visitamos antes de la actual, no sabe tampoco si ya hemos estado con anterioridad en dicho sitio ni si hemos enviado o no datos a la página.
Con el tiempo surgió la necesidad de conocer este tipo de información, y surgieron las famosas cookies.
Una cookie es un fragmento de información que se guarda en el ordenador del usuario, y depende de cada navegador.
Cada uno guarda sus propias cookies y no las comparte con los otros navegadores, y tiene su propia forma de guardarlas, aunque por lo general son archivos de texto.
Cuando ingresamos a un sitio web, el navegador revisa si tiene guardada alguna cookie asociada a dicha web, o mejor dicho asociada al dominio, y de existir, manda los datos al servidor junto con la petición.
Las páginas que ponen una cookie también le indican al navegador hasta cuando son válidas.
Un dominio solo puede acceder a las cookies que guardo, no a las cookies de los demás dominios.
Hasta acá, las únicas maneras de obtener datos de los visitantes, era mediante la URL o mediante el uso de cookies. El problema es que ambos recursos son inseguros, fácilmente modificables. La URL se puede modificar desde la barra del navegador y las cookies, al ser archivos guardados en el equipo del usuario, son fácilmente modificables.
Por lo que se necesitaba un método más seguro, que no sea modificable por los usuarios, fue así que se crearon las sesiones.

Una sesión en PHP es una cadena de caracteres aleatorios que forman una identificación única para cada visitante, dicha cadena de caracteres es el ID de sesión.
Cuando a un usuario se le asigna un id de sesión, el servidor crea un archivo en su sistema donde irá almacenando todos los datos que se necesiten guardar.

En futuras visitas, el servidor puede, mediante este identificador, poder saber de qué usuario se trata y la forma de obtener esa información es mediante el envío del id de sesión mediante una cookie o por la URL.
Resumiendo, cuando visitamos un sitio, el servidor nos asigna un id de sesión y crea un archivo en su sistema o base de datos. Los datos de nuestra visita así sean contraseñas o información que intercambiamos con el servidor, solo están disponibles en dicho servidor y lo único que se almacena en forma local en nuestro equipo, es una cookie con el id asignado, ningún otro dato sensible.

Implementación Básica

Iniciar una Sesión
Para iniciar una sesión en PHP, se utiliza la función session_start(). Esta función debe ser llamada al principio del script, antes de cualquier salida HTML.

```php
<?php
session_start();
?>
```

Establecer Variables de Sesión
Las variables de sesión se almacenan en el array superglobal $_SESSION.

```php
<?php
session_start();
$_SESSION["usuario"] = "Juan";
$_SESSION["edad"] = 25;
?>
```

Acceder a Variables de Sesión
Podemos acceder a las variables de sesión en cualquier página que haya iniciado la sesión.

```php
<?php
session_start();
echo "Usuario: " . $_SESSION["usuario"]; // Imprime "Usuario: Juan"
echo "Edad: " . $_SESSION["edad"]; // Imprime "Edad: 25"
?>
```

Destruir una Sesión

Para destruir una sesión y eliminar todas las variables de sesión, se utiliza session_destroy().

```php
<?php
session_start();
session_destroy();
?>
```

Ejemplo de uso.

Página 1: Establecer Variables de Sesión

```php
<?php
session_start();
$_SESSION["usuario"] = "Juan";
$_SESSION["edad"] = 25;
?>
<!DOCTYPE html>
<html>
<body>
    <p>Sesión iniciada. <a href="pagina2.php">Ir a la página 2</a></p>
</body>
</html>
```

Página 2: Acceder a Variables de Sesión

```php
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>
    <p>Usuario: <?php echo $_SESSION["usuario"]; ?></p>
    <p>Edad: <?php echo $_SESSION["edad"]; ?></p>
    <p><a href="cerrar_sesion.php">Cerrar sesión</a></p>
</body>
</html>
```

Página 3: Cerrar Sesión

```php
<?php
session_start();
session_destroy();
?>
<!DOCTYPE html>
<html>
<body>
    <p>Sesión cerrada. <a href="pagina1.php">Volver a la página 1</a></p>
</body>
</html>
```