

## PROGRAMACION

### Incremento / Decremento

```
$X = 5;  
$resultado = --$X;  
echo $resultado;  
echo $X;
```

Salida:

4

4

**Pre-decremento:** Decrementa primero y luego usa el nuevo valor.

```
$X = 5;  
$resultado = $X--;  
echo $resultado;  
echo $X;
```

Salida:

5

4

**Post-decremento:** Usa el valor original y luego decrementa.

### Operadores de comparación

#### Y LÓGICA (AND)

A	B	RES.
FALSE	FALSE	FALSE
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE
TRUE	TRUE	TRUE

#### O LÓGICA (OR)

A	B	RES.
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	TRUE

#### NEGACIÓN LÓGICA (NOT)

A	RES.
FALSE	TRUE
TRUE	FALSE

### Switch

```
1  <?php  
2  
3  $x = 5;  
4  $y = 7;  
5  
6  switch($x){  
7      case 5:  
8          echo $x * $y;  
9          break;  
10     case 7:  
11         echo $y - $x;  
12         break;  
13     default:  
14         echo $y - $x;;  
15         break;  
16 }  
17  
18 ?>
```

Acciones diferentes  
para cada opcion

Valores posibles  
para \$x

break; delimita las  
acciones

Acción por defecto

## Operadores de comparación

```
1 <?php
2
3 $x = 5;
4
5 echo $x == 7; // igual a -> muestra FALSE
6 echo $x == "5"; // igual a -> muestra TRUE
7 echo $x === "5"; // estrictamente igual a -> muestra FALSE
8 echo $x != 7; // distinto de -> TRUE
9 echo $x != "5"; // distinto de -> FALSE
10 echo $x !== "5"; // estrictamente distinto a -> muestra TRUE
11
12 ?>
```

## Operadores lógicos

FALSE TRUE

```
1 <?php
2
3 $x = 5;
4 $y = 7;
5
6 echo (($x > 7) and ($x > 4)); // muestra FALSE
7 echo (($x > 7) && ($x > 4)); // muestra FALSE
8
9 echo (($x > 7) or ($x > 4)); // muestra TRUE
10 echo (($x > 7) || ($x > 4)); // muestra TRUE
11
12 echo !(($x > 7) && ($x > 4)); // muestra TRUE
13 echo !(($x > 7) || ($x > 4)); // muestra FALSE
14
15 ?>
```

'and' equivale a '&&'

'or' equivale a '||'

el signo de exclamación significa 'negación' y su efecto es invertir el resultado

## While

Si la condición es TRUE repetimos el bloque

```
1 <?php
2
3 $x = 1;
4
5 while($x <= 5) {
6     echo "el número es: $x";
7     $x++;
8 }
9
10 ?>
```

Bloque de sentencias que se deben repetir. Se define entre llaves.

El programa muestra un número diferente en cada iteración

Continuamos por aquí una vez que la condición es FALSE

Debe haber al menos una acción que afecte a la condición.

## Do..while

las acciones se ejecutarán al menos 1 vez, aunque la condición sea FALSE

Bloque de sentencias que se deben repetir. Se define entre llaves.

Continuamos por aquí una vez que la condición es FALSE

```
1  <?php
2
3  $x = 1;
4
5  do{
6      echo "el número es: $x";
7      $x++;
8  }while($x <= 5)
9
10 ?>
```

El programa muestra un número diferente en cada iteración

Debe haber al menos una acción que afecte a la condición.

Si la condición es TRUE repetimos el bloque

## For

Inicializa el contador de bucle (\$x) y pone el valor inicial a 0

Continúa el bucle siempre y cuando \$x sea menor o igual a 10

Aumentar el valor del contador de bucle en 1 por cada iteración

```
1  <?php
2
3  for ($x = 0; $x <= 10; $x++) {
4      echo "El número es: $x <br>";
5  }
6
7  ?>
```

Continuamos por aquí cuando \$x > 10

El programa mostrará un valor diferente en cada iteración

## Arrays indexados

```
1  <?php
2  $arr = []; // crea un array sin elementos
3
4  $arr[0] = 15; // añade 15 al primer elemento
5  $arr[3] = 25; /* OJO! añade 25 a un elemento nuevo con índice 3, el índice
6  |         |         |         | 3 ya no representa el 4º elemento!!! */
7
8  $arr[3] = 30; /* modifica el elemento con índice 3 */
9  $arr2 = [10, 20, 30, 40]; /* crea un array y lo puebla no valores */
10
11  var_dump($arr); /* muestra el array con su estructura */
12  echo $arr2[3]; /* muestra el elemento con índice 3 */
```

## Arrays asociativos

```
1  <?php
2  $arr = [
3      'nombre' => "David",
4      'DNI' => "123456789Z"
5  ];
6
7  $arr['Apellido'] = 'Soler'; // añade el elemento 'apellido'
8  $arr['Apellido'] = 'Pérez'; /* modifica el elemento con
9      |         |         |         |         |         |         |
10     clave 'Apellido' */
11
12 var_dump($arr); /* muestra el array con su estructura */
13 echo $arr['nombre']; /* muestra el elemento con
14     |         |         |         |         |         |         |
15     clave 'nombre' */
```

## Arrays indexados Multidimensionales

```
<?php
$arr = [
    ['Juan', 'Perez', 25],
    ['Maria', 'Gomez', 30],
    ['Carlos', 'Rodriguez', 35]
];

$arr[] = ['Luis', 'Gonzalez', 40]; //añade un nuevo elemento al final del array (será otro array con 3 elementos)
$arr[0][1] = 'Pérez'; //modifica el valor de la posición 1 del array 0 (se nos había olvidado poner el acento en Pérez)

var_dump($arr); //muestra el contenido del array
echo $arr[1][0]; //muestra 'Maria'
```

## Arrays asociativos multidimensionales

```
1  <?php
2  $arr = [
3      'temperatura' => [10, 12, 14, 13],
4      'Humedad' => [90, 87, 75, 60]
5  ];
6
7  $arr['presion'] = [714, 825, 904, 1024]; // añade el elemento 'presion'
8  $arr['presion'][1] = 920; /* modifica el elemento con
9      |         |         |         |         |         |         |
10     clave 'presion', y dentro de este elemento, el dato 825, es decir, la
11     celda 1, con el nuevo valor 920 */
12
13 var_dump($arr); /* muestra el array con su estructura */
14 echo $arr['Humedad'][3]; /* muestra el elemento con clave 'Humedad', y
15     |         |         |         |         |         |         |
16     dentro de éste, el elemento con índice 3, es decir, el dato 60*/
```

## Recorrido de un array

### for – 1 dimensión indexado

```
<?php
    $arr = [10, 20, 30, 40, 50];

    for ($i = 0; $i < count($arr); $i++) {
        |     echo $arr[$i] . "<br>";
    } //imprime los valores del array
```



for – 2 dimensiones indexado

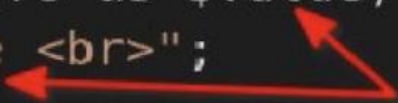
```
<?php
$arr = [
    [10, 20, 30, 40, 50],
    ["A", "B", "C", "D", "E"]
]; //array multidimensional

for ($i = 0; $i < count($arr); $i++) {
    for ($j = 0; $j < count($arr[$i]); $j++) {
        echo $arr[$i][$j] . " ";
    }
    echo "<br>";
} //recorrido de array multidimensional
```

Foreach – 1 dimensión indexado

```
<?php
$colors = ["red", "green", "blue", "yellow"];

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

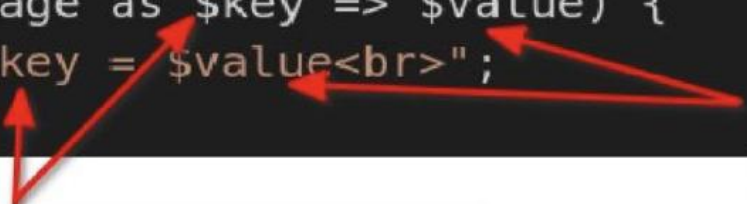


\$value contiene el valor de la celda en cada iteración

Foreach – 1 dimensión asociativo

```
$age = ["Peter"=>"35", "Ben"=>"37", "Joe"=>"43"];

foreach($age as $key => $value) {
    echo "$key = $value<br>";
}
```



Contiene la clave de cada elemento

Contiene los valores de cada elemento

## Foreach – 2 dimensiones indexado

```
<?php
$arr = [
    [10, 20, 30, 40, 50],
    ["A", "B", "C", "D", "E"]
];

foreach ($arr as $element) {
    echo "<h2>Elementos del array</h2>";
    echo "<ul>";
    foreach ($element as $value) {
        echo "<li>$value</li>";
    }
    echo "</ul>";
}

?>
```

### Elementos del array

- 10
- 20
- 30
- 40
- 50

### Elementos del array

- A
- B
- C
- D
- E

## Foreach – 2 dimensiones asociativo

```
<?php
$arr = [
    "Juan" => [
        "edad" => 25,
        "altura" => 1.75,
        "peso" => 70
    ],
    "Maria" => [
        "edad" => 30,
        "altura" => 1.60,
        "peso" => 60
    ]
];

foreach ($arr as $persona => $data) {
    echo "$persona: <br>";
    foreach ($data as $key => $value) {
        echo "$key: $value <br>";
    }
}
```

Juan:  
edad: 25  
altura: 1.75  
peso: 70  
Maria:  
edad: 30  
altura: 1.6  
peso: 60

## Funciones

```
<?php
    $fruits = ["apple", "orange", "banana", "pear"];

    $var1 = implode(" ", $fruits);
    echo $var1 . "<br>";

    $var2 = explode(" ", $var1);
    var_dump($var2);

    array_pop($fruits);
    var_dump($fruits);

    array_push($fruits, "kiwi");
    var_dump($fruits);

    array_shift($fruits);
    var_dump($fruits);

    array_unshift($fruits, "apple");
    var_dump($fruits);
?>
```

### EJEMPLO

apple, orange, banana, pear

```
/var/www/html/index.php:15:
array (size=4)
  0 => string 'apple' (length=5)
  1 => string 'orange' (length=6)
  2 => string 'banana' (length=6)
  3 => string 'pear' (length=4)

/var/www/html/index.php:18:
array (size=3)
  0 => string 'apple' (length=5)
  1 => string 'orange' (length=6)
  2 => string 'banana' (length=6)

/var/www/html/index.php:21:
array (size=4)
  0 => string 'apple' (length=5)
  1 => string 'orange' (length=6)
  2 => string 'banana' (length=6)
  3 => string 'kiwi' (length=4)

/var/www/html/index.php:24:
array (size=3)
  0 => string 'orange' (length=6)
  1 => string 'banana' (length=6)
  2 => string 'kiwi' (length=4)

/var/www/html/index.php:27:
array (size=4)
  0 => string 'apple' (length=5)
  1 => string 'orange' (length=6)
  2 => string 'banana' (length=6)
  3 => string 'kiwi' (length=4)
```

implode: Convierte un array en string.

explode: Convierte un string en array.

array\_pop: Elimina el ultimo elemento y devuelve su valor. Reduce la longitud del array en uno.

```
$frutas = ["manzana", "banana", "cereza"];
$ultimaFruta = array_pop($frutas);

echo $ultimaFruta; // Salida: cereza
print_r($frutas); // Salida: Array ( [0] => manzana [1] => banana )
```

array\_push: Añade un elemento al final.

array\_shift: Elimina el 1<sup>er</sup> elemento de un array y devuelve su valor.

```
$frutas = ["manzana", "banana", "cereza"];
$primeraFruta = array_shift($frutas);

echo $primeraFruta; // Salida: manzana
print_r($frutas); // Salida: Array ( [0] => banana [1] => cereza )
```

```
$colores = ["rojo" => "#FF0000", "verde" => "#00FF00", "azul" => "#0000FF"];
$primerColor = array_shift($colores);

echo $primerColor; // Salida: #FF0000
print_r($colores);
// Salida: Array ( [verde] => #00FF00 [azul] => #0000FF )
```

array\_unshift: Añade 1 o + elementos al inicio del array.

```
$frutas = ["banana", "cereza"];
array_unshift($frutas, "manzana", "naranja");

print_r($frutas);
// Salida: Array ( [0] => manzana [1] => naranja [2] => banana [3] => cereza )
```

```
$colores = ["verde" => "#00FF00", "azul" => "#0000FF"];
array_unshift($colores, "rojo");

print_r($colores);
// Salida: Array ( [0] => rojo [verde] => #00FF00 [azul] => #0000FF )
```

array\_splice: Elimina una porción de un array, opcionalmente, la reemplaza con otro elemento.

```
<?php
$fruits = ["Bannana", "Orange", "Apple", "Mango"];
array_splice($fruits,2,0,["Lemon","Kiwi"]); var_dump($fruits);
/*
en la posición 2, es decir 'Apple' elimina 0 elementos y añade
"Lemon" y "Kiwi", resultando en:
["Bannana", "Orange", "Lemon", "Kiwi", "Apple", "Mango"]
*/
array_splice($fruits,1,2); var_dump($fruits);
/*
desde la posición 1, es decir, 'Orange, elimina 2 elementos y
no añade nada mas, resultando en:
["Bannana", "Kiwi", "Apple", "Mango"]
*/
```

Array\_slice: Extraer una porción del array sin modificar el original.

```
$frutas = ["manzana", "banana", "cereza", "durazno"];
$porcion = array_slice($frutas, 1, 2);

print_r($porcion);
// Salida: Array ( [0] => banana [1] => cereza )
```

```
$frutas = ["manzana", "banana", "cereza", "durazno"];
$porcion = array_slice($frutas, 2);

print_r($porcion);
// Salida: Array ( [0] => cereza [1] => durazno )
```



```
$frutas = ["manzana", "banana", "cereza", "durazno"];
$porcion = array_slice($frutas, -2, 1);

print_r($porcion);
// Salida: Array ( [0] => cereza )
```

```
$frutas = ["manzana", "banana", "cereza", "durazno"];

// Usando array_slice()
$porcion = array_slice($frutas, 1, 2);
print_r($frutas); // El array original no cambia
// Salida: Array ( [0] => manzana [1] => banana [2] => cereza [3] => durazno )

// Usando array_splice()
$eliminadas = array_splice($frutas, 1, 2);
print_r($frutas); // El array original se modifica
// Salida: Array ( [0] => manzana [1] => durazno )
```

```
$fruits = ["Bannana", "Orange", "Apple", "Mango"];
$var = array_slice($fruits, 1);
var_dump($var);
/*
Corta desde la posición 1 incluido hasta el final del
array, y genera un nuevo array, resultando en:
["Orange", "Apple", "Mango"]
*/
$var = array_slice($fruits, 1, 2);
var_dump($var);
/*
corta desde la posición 1 incluida, 2 elementos
y genera un nuevo array, resultando en:
["Orange", "Apple"]
*/
```

Array\_merge: Combinar 2 o + arrays. Si las claves son numéricas, los valores se agregan en orden. Si las claves son asociativas, las claves duplicadas se sobrescriben con los valores del ultimo array que las contiene-.

```
$myGirls = ["Cecile", "Lone"];
$myBoys = ["Emil", "Tobias", "Linus"];
$myChildren = array_merge($myGirls, $myBoys);
var_dump($myChildren);
/* Nuevo array $myChildren cuyo contenido es
["Cecile", "Lone", "Emil", "Tobias", "Linus"]
| */
```

```
$array1 = [1, 2, 3];
$array2 = [4, 5, 6];

$resultado = array_merge($array1, $array2);

print_r($resultado);
```

Salida:

```
Array (
    [0] => 1
    [1] => 2
    [2] => 3
    [3] => 4
    [4] => 5
    [5] => 6
)
```

```
$array1 = ["a" => "apple", "b" => "banana"];
$array2 = ["b" => "blueberry", "c" => "cherry"];

$resultado = array_merge($array1, $array2);

print_r($resultado);
```

Salida:

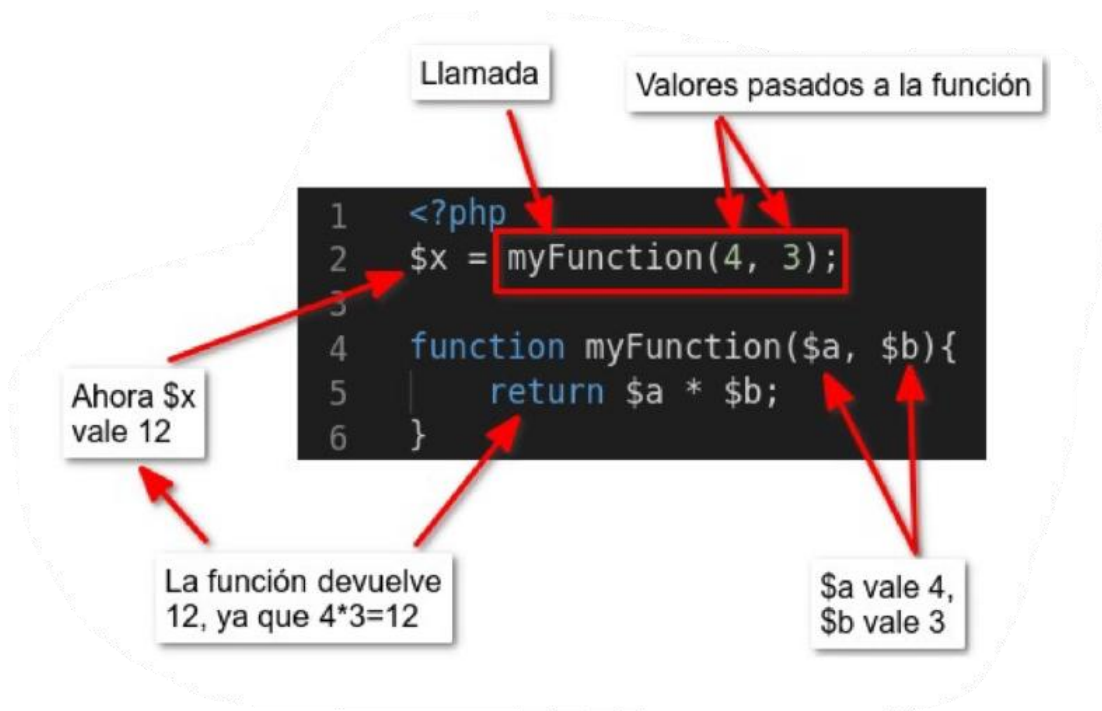
```
Array (
    [a] => apple
    [b] => blueberry
    [c] => cherry
)
```

```
$array1 = [1, 2];  
$array2 = [3, 4];  
$array3 = [5, 6];  
  
$resultado = array_merge($array1, $array2, $array3);  
  
print_r($resultado);
```

Salida:

```
Array (  
[0] => 1  
[1] => 2  
[2] => 3  
[3] => 4  
[4] => 5  
[5] => 6  
)
```

Llamada a funciones



Parámetros por defecto: Valor que toma el parámetro en caso de ser omitido el argumento relacionado.

En caso de no tener valor por defecto y no se haya proporcionado argumento → Genera error.

```
function saludar($nombre = "Juan") {  
    echo "Hola, $nombre!";  
}
```

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

```
// Esto es incorrecto  
function ejemplo($param1 = "valor", $param2) {  
    // código  
}
```

```
// Esto es correcto  
function ejemplo($param2, $param1 = "valor") {  
    // código  
}
```

```
function saludar($nombre = "Juan", $edad = 25) {  
    echo "Hola, $nombre! Tienes $edad años.";  
}  
  
saludar();          // Imprime: Hola, Juan! Tienes 25 años.  
saludar("Ana");     // Imprime: Hola, Ana! Tienes 25 años.  
saludar("Carlos", 30); // Imprime: Hola, Carlos! Tienes 30 años.
```

```
function mostrarDatos($datos = []) {  
    print_r($datos);  
}  
  
mostrarDatos();          // Imprime: Array()  
mostrarDatos([1, 2, 3]); // Imprime: Array([0] => 1 [1] => 2 [2] => 3)
```

Argumentos pasados por valor / por referencia

Por valor: La función recibe una copia del valor de la variable. Cualquier cambio que realices dentro de la función NO AFECTA a la variable original.

```
function duplicarPorValor($numero) {  
    $numero *= 2; // Cambia solo la copia  
}  
  
$valor = 10;  
duplicarPorValor($valor);  
  
echo $valor; // Salida: 10 (el valor original no cambió)
```

Por referencia: La función recibe una referencia a la variable original. Cualquier cambio que realices dentro de la función afecta directamente a la variable fuera de la función.

```
function duplicarPorReferencia(&$numero) {  
    $numero *= 2; // Cambia el valor original  
}  
  
$valor = 10;  
duplicarPorReferencia($valor);  
  
echo $valor; // Salida: 20 (el valor original cambió)
```

Combinación:

```
function sumarPorValor($num) {  
    $num += 10;  
}  
  
function sumarPorReferencia(&$num) {  
    $num += 10;  
}  
  
$a = 5;  
$b = 5;  
  
sumarPorValor($a); // $a no se modifica  
sumarPorReferencia($b); // $b se modifica  
  
echo "Valor de a: $a\n"; // Salida: Valor de a: 5  
echo "Valor de b: $b\n"; // Salida: Valor de b: 15
```

Array de argumentos

```
function mostrarNombres($nombres) {  
    foreach ($nombres as $nombre) {  
        echo $nombre . "\n";  
    }  
}  
  
$misNombres = ["Juan", "Ana", "Carlos"];  
mostrarNombres($misNombres);
```

Salida

Juan

Ana



Carlos

Func\_num\_args: Obtener el numero de argumentos. Saber cuántos argumentos se pasaron a la función.

```
function contarArgumentos() {  
    $numArgs = func_num_args(); // Devuelve el número de argumentos  
    echo "Número de argumentos: " . $numArgs;  
}  
  
contarArgumentos(1, 2, 3, 4); // Imprime: Número de argumentos: 4
```

Func\_get\_args: Manejar múltiples argumentos. Devuelve un array con todos los argumentos pasados a la función.

```
function imprimirArgumentos() {  
    $argumentos = func_get_args(); // Devuelve un array con los argumentos  
    foreach ($argumentos as $arg) {  
        echo $arg . "\n";  
    }  
}  
  
imprimirArgumentos("Hola", "Mundo", 123); // Imprime: Hola, Mundo, 123
```

Diagrama de anotación de código PHP con explicaciones de funciones nativas:

- Pasamos 3 argumentos en la llamada a la función (señalando a `foo(1, 2, 3);`)
- no definimos ningún parámetro (señalando a `function foo()`)
- función nativa para obtener el número de argumentos pasados (señalando a `func_num_args()`)
- función nativa para acceder a un argumento concreto (señalando a `func_get_arg(1)`)
- función nativa para obtener el array de argumentos (señalando a `func_get_args()`)

```
1 <?php  
2 foo(1, 2, 3);  
3 function foo()  
4 {  
5     $numargs = func_num_args();  
6     echo "Número de argumentos: $numargs <br>";  
7     if ($numargs >= 2) {  
8         echo "El segundo argumento es: " . func_get_arg(1) . "<br>";  
9     }  
10    $arg_list = func_get_args();  
11    for ($i = 0; $i < $numargs; $i++) {  
12        echo "El argumento $i es: " . $arg_list[$i] . "<br>";  
13    }  
14 }
```

Global: Acceder a una variable definida fuera de una función. Permite que la función acceda y modifique una variable global, definida fuera del ámbito de la función.

```
$mensaje = "Hola desde fuera de la función.";  
  
function mostrarMensaje() {  
    global $mensaje; // Hace que la variable global $mensaje esté disponible dentro  
    echo $mensaje;  
}  
  
mostrarMensaje(); // Imprime: Hola desde fuera de la función.
```

```
$mensaje = "Hola desde fuera de la función.";

function mostrarMensaje() {
    echo $GLOBALS['mensaje']; // Accede a la variable global usando $GLOBALS
}

mostrarMensaje(); // Imprime: Hola desde fuera de la función.
```

```
$numero = 10;

function cambiarValor() {
    global $numero;
    $numero = 20;
}

cambiarValor();
echo $numero; // Imprime: 20
```

#### Funciones nativas

pi() → Numero PI.

round() → Redonde al entero más próximo.

Ceil() → Redondea al entero más próximo hacia arriba.

Floor() → Redonde entero más próximo hacia abajo.

Sqrt() → Raiz cuadrada.

Sin(), cos() → Seno y coseno.

Min(), max() → Mínimo y máximo de una sucesión numérica.

Rand() → Valor aleatorio.

Strlen() → Longitud del string.

Strpos() → Buscar un texto dentro de otro y retorna su posición.

Printf() → Imprime string con formato.

Substr() → Extrae parte de un string creando otro.

Str\_replace() → Reemplaza partes del texto por otras.

Strtoupper() → Convierte en mayúsculas.

Strtolower() → Convierte en minúsculas.

Trim() → Elimina los espacios vacíos al principio y al final del string.