

# Funciones

---

- **Funciones**
  - Qué es una función
    - Ventajas de usar funciones
  - Sintaxis
    - Definición de función sin parámetros
    - Definición de función con parámetros
  - Funciones con parámetros y argumentos
    - Parámetros con valores predeterminados
    - Pasar argumentos por valor y por referencia
    - Número variable de argumentos (funciones variádicas)
  - Funciones recursivas
    - Qué son
    - Cómo funcionan
    - Casos de uso
  - Funciones anónimas (closures)
    - Qué son
    - Sintaxis
    - Casos de uso
    - Ventajas
  - Funciones de retorno de funciones (funciones de orden superior)
    - Retornar funciones desde otras funciones
    - Sintaxis
    - Casos de uso
  - Scope de las variables
  - Funciones predefinidas
  - Retorno de valores
    - Sintaxis
  - Cómo llamar a una función
    - Sintaxis
      - Con parámetros
      - Sin parámetros

## Qué es una función

Una función es un bloque de código que realiza una tarea específica o un conjunto de tareas relacionadas. Es una forma de organizar y reutilizar código al dividir un programa en tareas más pequeñas y manejables. Las funciones pueden recibir argumentos (datos de entrada) y pueden devolver un resultado (valor de salida) después de realizar su tarea. Las funciones juegan un papel fundamental en la estructura y modularidad de un programa.

### Ventajas de usar funciones

- **Reutilización de código:** Una de las principales ventajas de las funciones es que permiten escribir una vez y usar muchas veces. Al definir una función para una tarea específica, puedes llamarla desde

diferentes partes del programa, evitando la duplicación de código y facilitando el mantenimiento y las actualizaciones.

- **Modularidad:** Las funciones ayudan a dividir un programa en módulos más pequeños y autónomos. Esto facilita la comprensión del código, ya que cada función se encarga de una tarea específica y bien definida.
- **Facilita el debugging:** Al utilizar funciones, es más fácil identificar y corregir errores en el código. Si una función no produce el resultado esperado, puedes enfocarte en la lógica de esa función en lugar de tener que revisar todo el programa.
- **Mejora la legibilidad:** Las funciones pueden recibir nombres descriptivos que indican claramente lo que hacen. Esto hace que el código sea más legible y comprensible para otros desarrolladores y para ti mismo en el futuro.
- **Abstracción:** Al utilizar funciones, puedes abstraer detalles complejos y proporcionar una interfaz simple para su uso. Esto permite que otros desarrolladores utilicen la función sin necesidad de conocer los detalles internos de su implementación.
- **Facilita la colaboración:** Cuando trabajas en equipo, dividir el código en funciones bien definidas facilita la asignación de tareas y la colaboración en diferentes partes del proyecto.
- **Facilita el mantenimiento:** Si necesitas realizar cambios o mejoras en una funcionalidad específica, solo necesitas modificar la función relacionada en lugar de todo el código.

## Sintaxis

### Definición de función sin parámetros

```
function nombre_funcion() {  
    //código de la función  
}
```

### Definición de función con parámetros

```
function nombreFuncion(parametro1, parametro2, ...) {  
    // Código de la función  
    // Puede incluir operaciones, sentencias, etc.  
}
```

## Funciones con parámetros y argumentos

Las funciones en PHP pueden recibir parámetros, que son variables que se utilizan para pasar datos o valores a la función. Cuando se llama a la función, los valores pasados como argumentos se asignan a los parámetros de la función, y estos valores pueden utilizarse dentro del bloque de código de la función.

### Parámetros con valores predeterminados

En PHP, puedes asignar valores predeterminados a los parámetros de una función. Esto significa que si un argumento no se proporciona al llamar a la función, el parámetro tomará el valor predeterminado especificado.

```
function nombreFuncion($parametro1 = valorPredeterminado) {  
    // Código de la función  
}
```

Ejemplo:

```
function saludar($nombre = "amigo") {  
    echo "¡Hola, $nombre!";  
}  
  
// Llamada a la función sin argumento  
saludar(); // Imprime: ¡Hola, amigo!  
  
// Llamada a la función con argumento  
saludar("Nuria"); // Imprime: ¡Hola, Nuria!
```

## Pasar argumentos por valor y por referencia

Por defecto, en PHP, los argumentos se pasan por valor, lo que significa que se crea una copia del valor original dentro de la función. Si modificas el valor del parámetro dentro de la función, no afectará a la variable original fuera de la función.

Para pasar argumentos por referencia, se utiliza el operador & antes del nombre del parámetro en la definición de la función. Esto permite que la función modifique el valor de la variable original fuera de la función.

```
function nombreFuncion(&$parametro) {  
    // Código de la función  
}
```

Ejemplo:

```
function incrementar(&$numero) {  
    $numero++;  
}  
  
$valor = 10;  
incrementar($valor);  
echo $valor; // Imprime: 11
```

## Número variable de argumentos (funciones variádicas)

PHP también permite crear funciones que acepten un número variable de argumentos. Esto se logra utilizando el operador ... antes del nombre del último parámetro en la definición de la función.

```
function nombreFuncion($parametro1, $parametro2, ...$parametrosVariadicos) {  
    // Código de la función  
}
```

Ejemplo:

```
function sumar(...$numeros) {  
    $total = 0;  
    foreach ($numeros as $numero) {  
        $total += $numero;  
    }  
    return $total;  
}  
  
echo sumar(1, 2, 3, 4, 5); // Imprime: 15
```

## Funciones recursivas

Qué son

En programación, una función recursiva es aquella que se llama a sí misma durante su ejecución. Es decir, en lugar de realizar un bucle o iteración, la función se invoca a sí misma para resolver un problema más pequeño o similar.

Cómo funcionan

En programación, una función recursiva es aquella que se llama a sí misma durante su ejecución. Es decir, en lugar de realizar un bucle o iteración, la función se invoca a sí misma para resolver un problema más pequeño o similar.

Ejemplo:


Un ejemplo clásico de función recursiva es el cálculo del factorial de un número. El factorial de un número entero positivo  $n$ , denotado como  $n!$ , es el producto de todos los números enteros positivos desde 1 hasta  $n$ .

```
function factorial($n) {  
    if ($n === 0 || $n === 1) {  
        return 1; // Caso base  
    } else {  
        return $n * factorial($n - 1); // Llamada recursiva  
    }  
}  
  
echo factorial(5); // Imprime: 120 (5! = 5 * 4 * 3 * 2 * 1 = 120)
```

## Casos de uso

Las funciones recursivas son útiles en situaciones en las que un problema puede descomponerse en problemas más pequeños y similares, y donde es más sencillo resolver cada subproblema individualmente. Algunos casos de uso comunes de funciones recursivas incluyen:

- **Cálculos matemáticos:** Como se mostró en el ejemplo del factorial, funciones recursivas son adecuadas para problemas matemáticos como cálculos de potencias, números de Fibonacci, etc.
- **Estructuras de datos:** La recursión es una técnica común para recorrer y manipular estructuras de datos complejas, como árboles y grafos.
- **Búsqueda:** En algoritmos de búsqueda y recorrido, como la búsqueda en profundidad (DFS) y la búsqueda en anchura (BFS), las funciones recursivas pueden ser muy útiles para visitar todos los nodos de una estructura de datos.
- **Problemas dividir y conquistar:** Algoritmos que siguen el enfoque de "dividir y conquistar" a menudo utilizan funciones recursivas para resolver subproblemas.

 **ADVERTENCIA** es importante tener en cuenta que las funciones recursivas deben diseñarse cuidadosamente para evitar bucles infinitos y asegurarse de que haya un caso base que termine la recursión. Además, en algunos casos, las funciones iterativas (bucles) pueden ser más eficientes que las funciones recursivas, por lo que es necesario evaluar cada situación y escoger la mejor opción para cada problema.

## Funciones anónimas (closures)

### Qué son

En programación, una función anónima, también conocida como cierre o closure, es una función que no tiene un nombre explícito. A diferencia de las funciones tradicionales que se definen mediante la palabra clave `function` seguida de un nombre, las funciones anónimas se definen de manera más compacta y no llevan un nombre.

### Sintaxis

```
$variableFuncion = function($parametro) {  
    // Código de la función anónima  
};
```

### Ejemplo:

```
$sumar = function($a, $b) {  
    return $a + $b;  
};  
  
echo $sumar(5, 3); // Imprime: 8
```

## Casos de uso

Las funciones anónimas son útiles en situaciones donde necesitas una función temporal o ad hoc que no requiere un nombre único.

- **Callbacks:** Puedes pasar una función anónima como argumento a otras funciones, lo que permite crear callbacks flexibles y personalizados.
- **Iteraciones:** En ciertos contextos, las funciones anónimas son útiles para realizar iteraciones sobre una lista de elementos o una colección de datos.
- **Filtros y mapeo:** Las funciones anónimas se pueden usar para filtrar o mapear elementos de una lista, por ejemplo, en funciones como `array_filter` y `array_map`.
- **Manejo de eventos:** En entornos web, las funciones anónimas pueden utilizarse para manejar eventos de forma rápida y sencilla.

## Ventajas

- **Concisión:** Las funciones anónimas permiten definir lógica en un lugar específico sin la necesidad de crear una función con un nombre.
- **Flexibilidad:** Al ser funciones sin nombre, se pueden utilizar de manera más flexible y directa en diferentes contextos.
- **Closures:** Las funciones anónimas pueden capturar variables del entorno circundante, lo que las convierte en closures y les permite mantener un estado interno incluso después de que la función haya finalizado.
- **Callbacks:** Son especialmente útiles para crear callbacks en tiempo real para ciertas operaciones o procesamiento de datos.



**ADVERTENCIA** Es importante tener en cuenta que pueden ser menos legibles cuando su lógica es compleja. En tales casos, es mejor considerar la creación de una función con nombre para mejorar la claridad del código.

## Funciones de retorno de funciones (funciones de orden superior)

Las funciones de retorno de funciones, también conocidas como funciones de orden superior, son aquellas funciones que pueden devolver otras funciones como resultado. En PHP, esto es posible debido a que las funciones son tratadas como ciudadanos de primera clase, lo que significa que pueden ser asignadas a variables, pasadas como argumentos a otras funciones y también retornadas desde funciones.

### Retornar funciones desde otras funciones

Para retornar una función desde otra función, simplemente se debe definir la función interna dentro del cuerpo de la función externa y luego retornarla como resultado. La función interna se convertirá en una función de orden superior que puede ser utilizada en otros lugares del código.

### Sintaxis

```
function funcionExterna() {  
    // Código de la función externa  
    return function() {
```

```
        // Código de la función interna
    };
}
```

Ejemplo:

```
function crearFuncionSumar() {
    return function($a, $b) {
        return $a + $b;
    };
}

$sumar = crearFuncionSumar();
echo $sumar(5, 3); // Imprime: 8
```

## Casos de uso

- **Closures personalizadas:** Puedes devolver closures personalizadas desde funciones, lo que te permite crear funciones específicas en tiempo de ejecución según las necesidades del contexto.
- **Callbacks dinámicos:** Al devolver funciones desde otras funciones, puedes proporcionar callbacks dinámicos que realizan diferentes acciones según el contexto.
- **Configuraciones dinámicas:** Las funciones de retorno son útiles cuando necesitas devolver configuraciones dinámicas que se utilizan en diferentes partes del código.
- **Funciones de alto nivel:** Las funciones de retorno son fundamentales en la programación de alto nivel, donde la abstracción y la composición son esenciales.
- **Estrategias:** Puedes definir diferentes estrategias o algoritmos y seleccionar la estrategia adecuada según las condiciones.

## Scope de las variables

En PHP, el alcance o scope de las variables y constantes también se refiere a la visibilidad y accesibilidad de estas dentro de un programa. Indica qué partes del código pueden acceder y utilizar una determinada variable o constante, así como cuándo se crea y destruye su espacio de almacenamiento.

- **Scope global:** Las variables y constantes declaradas fuera de cualquier función tienen un alcance global. Esto significa que se pueden acceder a ellas desde cualquier parte del programa, ya sea dentro de funciones, bloques de código o en el ámbito global. Las variables globales permanecen en memoria durante toda la ejecución del programa y pueden ser accedidas y modificadas en cualquier momento.
- **Scope local:** Las variables y constantes declaradas dentro de una función tienen un alcance local. Esto significa que solo pueden ser accedidas y utilizadas dentro de la función en la que se declaran. Estas variables locales solo existen mientras la función está en ejecución y se destruyen una vez que la función finaliza su ejecución. Cada función tiene su propio scope local, lo que significa que las variables con el mismo nombre pueden existir en diferentes funciones sin causar conflictos.

```
<?php
$globalVariable = "Soy una variable global"; // Variable global
```

```
function myFunction() {
    $localVariable = "Soy una variable local"; // Variable local dentro de la
función

    echo $GLOBALS['globalVariable'] . "\n"; // Accediendo a la variable global
desde la función
    echo $localVariable . "\n"; // Accediendo a la variable local desde la función
}

echo $globalVariable . "\n"; // Accediendo a la variable global desde el ámbito
global
echo $localVariable . "\n"; // Error: Undefined variable, no se puede acceder
desde el ámbito global

myFunction(); // Llamando a la función para imprimir los valores de las variables
dentro de ella
?>
```



**TIP** recuerda que para acceder a una variable global dentro de una función en PHP, se puede utilizar la matriz superglobal \$GLOBALS, que contiene todas las variables globales accesibles desde cualquier parte del código.

## Funciones predefinidas

PHP incluye una amplia variedad de funciones predefinidas que cubren diversas áreas y funcionalidades, lo que facilita el desarrollo de aplicaciones web y otros tipos de programas. Estas funciones predefinidas se agrupan en categorías según su propósito y uso.

- **strlen()**: Devuelve la longitud de una cadena.
- **count()**: Cuenta el número de elementos en un array o el número de caracteres en una cadena.
- **date()**: Devuelve la fecha y hora actual formateada según el formato especificado.
- **str\_replace()**: Reemplaza todas las apariciones de un texto en una cadena con otro texto.
- **implode()**: Combina los elementos de un array en una cadena.
- **file\_get\_contents()**: Lee un archivo y devuelve su contenido como una cadena.
- **json\_encode()**: Convierte una variable PHP en una cadena JSON.
- **mysqli\_query()**: Ejecuta una consulta SQL en una base de datos MySQL.



**PARA SABER MÁS** puedes encontrar más información en la sección '[Internal \(built-in\) functions](#)' del manual oficial de PHP

## Retorno de valores

### Sintaxis

En PHP, una función puede devolver un valor utilizando la palabra clave return. El valor que se devuelve puede ser de cualquier tipo de datos válido en PHP, como enteros, cadenas, booleanos, arrays, objetos, entre otros. El valor devuelto por la función puede ser utilizado en otras partes del código o asignado a una variable para su posterior uso.



Es importante tener en cuenta que una función puede tener múltiples puntos de retorno, es decir, puede utilizar `return` en diferentes lugares del código para devolver diferentes valores según ciertas condiciones. Una vez que una función encuentra una instrucción `return`, la ejecución de la función se detiene y se devuelve el valor especificado. Además, si una función no contiene una declaración `return`, automáticamente devuelve el valor `NULL`.

```
function nombreFuncion(parametro1, parametro2, ...) {  
    // Código de la función  
  
    // Valor que se va a devolver  
    return $valor;  
}
```

```
function nombreFuncion($parametro) {  
    if ($parametro) {  
        return $valor;  
    } else {  
        return $valor2;  
    }  
}
```

Ejemplo:

```
function sumar($a, $b) {  
    $resultado = $a + $b;  
    return $resultado;  
}  
  
$valorSuma = sumar(5, 3);  
echo $valorSuma; // Imprime: 8
```

```
function obtenerMensaje($hora) {  
    if ($hora < 12) {  
        return "Buenos días";  
    } elseif ($hora < 18) {  
        return "Buenas tardes";  
    } else {  
        return "Buenas noches";  
    }  
}  
  
$mensaje = obtenerMensaje(15); // Devuelve "Buenas tardes"  
echo $mensaje;
```

## Cómo llamar a una función

Para llamar a una función en PHP, simplemente es necesario escribir el nombre de la función seguido de paréntesis. Si la función recibe parámetros, estos se deben proporcionar dentro de los paréntesis, separados por comas, de acuerdo con los parámetros definidos en la declaración de la función.

### Sintaxis

#### Con parámetros

```
nombreFuncion(argumento1, argumento2);
```

Ejemplo:

```
function saludar($nombre) {  
    echo "¡Hola, $nombre!";  
}  
  
saludar("Nuria"); // Llamada a la función con el argumento "Nuria". Imprime:  
¡Hola, Nuria!
```

#### Sin parámetros

```
nombreFuncion();
```

Ejemplo:

```
function saludar() {  
    echo "¡Hola, bienvenido!";  
}  
  
saludar(); // Llamada a la función sin argumentos. Imprime: ¡Hola, bienvenido!
```