Variables, expresiones, y funciones

Objetivo

Manejar los elementos que forman el lenguaje de programación PHP como el manejo de variables, expresiones, operadores, funciones y estructuras de control.

Variables, expresiones y funciones

Manejo de variables

Como se menciono antes, el manejo de las variables en PHP es muy diferente a otros lenguajes como C, Java o Visual Basic. Por ejemplo las variables son case sensitive y no hay tipos de datos predefinidos.

Declaración de variables

En PHP no se declaran las variables antes de utilizarlas, al momento de asignarles un valor se crean automáticamente, las variables siempre llevan como prefijo el carácter $. Otro aspecto peculiar de PHP es que los nombres de las variables son sensibles a mayúsculas/minúsculas o *Case Sensitive*. Aquí tenemos un ejemplo muy sencillo del manejo de variables:

```
<?php
$variable1 = "texto";
$numero1 = 10;
$Numero1 = 20.1569;
$bandera1 = true;
$bandera2 = false;
?>
```

Reglas para asignar nombre a las variables

Los nombre deben iniciar con una letra o guión bajo (_).
Los caracteres validos para el nombre pueden ser letras, números o guiones bajos.
No son validos caracteres como +, -, *, etc.
No hay un límite del número de caracteres para el nombre de las variables.

Variables, expresiones y funciones

Asignación por valor y por referencia

Los valores de las variables se pueden manejar por valor o por referencia, el default es por valor.

Por valor

Cuando se asigna una variable por valor, el contenido es copiado a la variable, si se altera el valor original, el valor de la copia no se ve afectado. Ejemplo:

```
<?php
$numero1 = 10;
$numero2 = $numero1;
echo $numero1;// 10
echo $numero2;// 10
$numero1 = 20;
echo $numero1;// 20
echo $numero2;// 10
?>
```

Por referencia

Si la asignación se hace por referencia, entonces PHP en lugar de copiar el valor original solo crea una apuntador a la variable original. De esta manera si el valor original cambia, también cambia el valor de la referencia. Para asignar un valor por referencia se utiliza el carácter &, por ejemplo:

```
<?php
$numero1 = 10;
$numero2 = &$numero1;
echo $numero1;// 10
echo $numero2;// 10
$numero1 = 20;
echo $numero1;// 20
echo $numero2;// 20
?>
```

Variables, expresiones y funciones

Nombres de variables variables

PHP permite utilizar el valor de una variable para asignar el nombre a otra variable, para hacer esto se utiliza doble el carácter $. Por ejemplo:

```
<?php
$nombre = "Juan";
$$nombre = "Perez";
echo $nombre;// Juan
echo $Juan;// Perez - Note que la J esta en mayúsculas
$nombre = "Jose";
echo $nombre;// Jose
echo $Juan;// Perez
?>
```

En este caso PHP para `$$nombre` crea una nueva variable llamada `"Juan"` ya que ese es el valor asignado a la variable `$$nombre`.

Variables, expresiones y funciones

Tipos de Datos

Los tipos de datos que soporta PHP son los siguientes:
• boolean
• string
• array
• integer
• float
• NULL
• object
• resource

boolean

Una variable boleana puede contener un `true` o `false` o bien `TRUE` o `FALSE`, en este caso PHP no es case sensitive. Comúnmente este tipo de datos se utiliza para evaluar una expresión y controlar si un bloque de código se ejecuta o no, esta evaluación se hace regularmente con los operadores de igualdad `==` o negación `!` o `not`.

Ejemplo:

```
<?php
$bandera = true;
if ($bandera == true)
{
echo "\$bandera es true !";
}
if ($bandera)// Esta manera es la recomendada
{
echo "\$bandera es true !";
}
$bandera = false;
if ($bandera == false)
{
echo "\$bandera es false !";
}
if (!$bandera)// Esta manera es la recomendada
{
echo "\$bandera es false !";
}
?>
```

Variables, expresiones y funciones

El resultado de la variable boleana también se puede asignar como resultado de la evaluación de una expresión:

```
<?php
$valor = 10;
$bandera = ($valor > 5);
if ($bandera)
{
echo "\$bandera es true !";
}
if (!$bandera)
{
echo "\$bandera es false !";
}
?>
```

Los siguientes valores se consideran como `false` cuando se evalúan en una expresión boleana:

- El numero integer cero
- El numero float cero
- El string vacío o "0" (cero)
- Un arreglo vacío
- Un objeto vacío
- El tipo NULL

Variables, expresiones y funciones

Cualquier otro valor a los anteriores se evalúa como un `true`.

```
<?php
$valorInteger = 0;
$valorFloat = 0.0;
$valorStringCero = "0";
$valorStringSinDatos = "";
$valorNULL = NULL;
if (!$valorInteger)
{
echo "\$valorInteger es false !";
}
if (!$valorFloat)
{
echo "\$valorFloat es false !";
}
if (!$valorStringCero)
{
echo "\$valorStringCero es false !";
}
if (!$valorStringSinDatos)
{
echo "\$valorStringSinDatos es false !";
}
if (!$valorNULL)
{
echo "\$valorNULL es false !";
}
if (!$valorStringVacio)
{
echo "\$valorStringVacio es false !";
}
if (!$valorArregloVacio())
{
echo "\$valorArregloVacio() es false !";
}
?>
```

is_bool()
Para verificar si una variable es del tipo boolean podemos utilizar la función `is_bool()`, la cual regresa true si la variable es boleana, false de lo contrario. Ejemplo:

```
is_bool(variable)
```

Variables, expresiones y funciones

string

Para PHP una variable `string` es una serie de caracteres, no hay límite para el número de caracteres. Algo que hay que considerar es que PHP no soporta Unicodigo. Un string puede estar delimitado de tres maneras:
• Apóstrofes
• Comillas
• heredoc

Ejemplo:

```
<?php
$stringComillas = "String con  comillas ";
$stringApostrofes =  String con "apostrofes" ;
$stringHeredoc = <<<IDENTIFICADOR
Este es un string largo de "varios"  renglones
definido con la sintaxis heredoc.
IDENTIFICADOR;
echo "\$stringComillas = $stringComillas <br>";
echo "\$stringApostrofes = $stringApostrofes <br>";
echo "\$stringHeredoc = $stringHeredoc <br>";
?>
```

Podemos insertar comillas en un string definido con apostrofes y viceversa sin mayores complicaciones. Note que en el caso de la sintaxis `heredoc`, no se proporciona un identificador al inicio y final de la definición del string.

Es muy importante que la ultima línea del identificador no contenga ningún carácter además del identificador y el punto y coma, no debe haber espacios, ni tabs, no se debe identar la ultima línea de texto.

Variables, expresiones y funciones

Caracteres de escape
En caso de que necesitemos incluir apostrofes o comillas en un string definido por esos mismos delimitadores, tenemos que utilizar los caracteres de escape. PHP cuenta con varios caracteres que se llaman de escape, por ejemplo, para insertar comillas dentro de comillas utilizamos un carácter de escape.

Carácter de escape	Descripción
\n	Salto de línea o linefeed
\r	Salto de carro o carriage return
\t	Tabulador
\\	Backslash
\$	Símbolo monetario
\"	Comillas

```
<?php
$stringComillasAnidadas = "String con \"comillas\" anidadas";
$stringApostrofesAnidadas = ´String con \´apostrofes\´ anidadas´;
$stringRutaTipoDOS = "C:\\Windows\\System32";
echo "\$stringComillasAnidadas = $stringComillasAnidadas <br>";
echo
"\$stringApostrofesAnidadas = $stringApostrofesAnidadas <br>";
echo
"\$stringRutaTipoDOS = $stringRutaTipoDOS <br>";
?>
```

is_string()

Para verificar si una variable es del tipo boolean podemos utilizar la función `is_string()`, la cual regresa trae si la variable es boleana, false de lo contrario. Ejemplo:

```
is_string(variable)
```

Variables, expresiones y funciones

array

En PHP un arreglo es un mapa ordenado de pares de valores y llaves, cada llave tiene un valor asociado. La sintaxis para crear un arreglo es la siguiente:

```
array( [key =>] value, ...[keyN =>] valueN)
```

`key` es la llave y puede ser un string o numero entero positivo, `value` es el valor y puede ser cualquier valor. Por default el primer elemento de un arreglo empieza en cero, y no en uno. Ejemplos:

```
$arreglo1 = array("elemento1" => "valor1", "elemento2" =>
"valor2");
$arreglo2 = array(0 => 100, 1=> 200);
```

Se puede combinar el tipo de llaves con strings, números y boleanos. Además se puede iniciar el arreglo en con un elemento diferente al cero y saltarse a un numero de elemento especifico:

```
$arreglo3 = array(1 => 100, 2 => "valor2", "elemento3" => 300,
10 => true);
```

Se puede omitir la parte de la llave y solo asignar los valores.

```
$arreglo4 = array(100,200,300);
```

Tambien podemos utilizar una variable para asignar el valor de un elemento en el arreglo:

```
$variable = 500;
$arreglo5 = array(10,"valor2",$variable);
```

Se puede agregar un nuevo elemento después de declarado de esta manera:

```
$arreglo6 = array(10,"valor2");
$arreglo6[] = 20; // Este es el elemento 3 en el arreglo
$arreglo6[] = "valor3"; // Este es el elemento 4 en el arreglo
$arreglo6["valor4"] = "valor4";// Este es el elemento 5 en el
arreglo
```

Si queremos desplegar el valor de un elemento definido como un string, por ejemplo:
`$arreglo1["elemento1"]`

Podemos hacerlo de esta manera: `echo $arreglo1["elemento1"] . "<br>";`

O de esta otra manera: `echo "$arreglo1[elemento1] <br>";`

Note que en el último caso no es necesario y no es valido delimitar entre comillas el valor de `elemento1`.

Estas opciones no son validas para referenciar elementos de un arreglo, y se dispara un error al ejecutarlas:

```
echo "$arreglo1["elemento1"] <br>";
echo "$arreglo1[\"elemento1\"] <br>";
echo "$arreglo1[ elemento1 ] <br>";
```

Tambien es posible concatenar dos arreglos para formar un tercero, por ejemplo:

```
$arreglo1 = array("elemento1" => "valor1", "elemento2" =>
"valor2");
$arreglo2 = array("elemento3" => "valor3", "elemento4" =>
"valor4");
$arreglo3 = $arreglo1 + $arreglo2;
```

En caso de que los arreglos contengan valores que se duplican, tiene precedencia el primer arreglo:

```
$arreglo1 = array("elemento1" => "valor1", "elemento2" =>
"valor2");
$arreglo2 = array("elemento2" => "valor2a", "elemento3" =>
"valor3");
$arreglo3 = $arreglo1 + $arreglo2; // Se ignora "eleento2" =>
"valor2a"
```

Ejemplos:

```
<?php
$arreglo1 = array("elemento1" => "valor1", "elemento2" =>
"valor2");
echo $arreglo1["elemento1"]; // valor1
echo $arreglo1["elemento2"]; // valor2
$arreglo2 = array(0 => 100, 1=> 200);
echo $arreglo2[0]; // 100
echo $arreglo2[1]; // 200
$arreglo3 = array(1 => 100, 2 => "valor2", "elemento3" => 300,
10 =>
true);
echo $arreglo3[1]; // 100
echo $arreglo3[2]; // valor2
echo $arreglo3["elemento3"]; // 300
echo $arreglo3[10]; // 1 (true = 1)
$arreglo4 = array(100,200,300);
echo $arreglo4[0]; // 100
echo $arreglo4[1]; // 200
echo $arreglo4[2]; // 300
$arreglo5 = array(100,10 => 200,300);
echo $arreglo5[0]; // 100
echo $arreglo5[10]; // 200
echo $arreglo5[11]; // 300
$variable = 500;
$arreglo6 = array(10,"valor2",$variable);
echo $arreglo6[0]; // 10
echo $arreglo6[1]; // "valor2"
echo $arreglo6[2]; // 500
$arreglo7 = array(10,"valor2");
$arreglo7[] = 20;
$arreglo7[] = "valor3";
$arreglo7["valor4"] = "valor4";
echo $arreglo7[0]; // 10
echo $arreglo7[1]; // "valor2"
echo $arreglo7[2]; // 20
echo $arreglo7[3]; // "valor3"
echo $arreglo7["valor4"]; "valor4"
?>
```

is_array()
Para verificar si una variable es del tipo array podemos utilizar la función `is_array()`, la cual regresa true si la variable es array, false de lo contrario. Ejemplo:

```
is_array(variable)
```

Variables, expresiones y funciones

integer y float

Una variable de tipo `integer` es la que tiene un valor positivo o negativo entero, esto es sin decimales. El tamaño exacto varia según la plataforma donde esta corriendo PHP, el valor máximo aproximado es de 32 billones.
Una variable tipo `float` tiene un valor con valores positivos o negativos y con decimales. El número máximo de decimales varia según la plataforma donde esta corriendo PHP, el valor máximo aproximado es de 14 decimales.
Ejemplo:

```
<?php
$valorIntegerPositivo = 100;
$valorIntegerNegativo = -100;
$valorFloatPositivo = 100.544875;
$valorFloatNegativo = -100.544875;
$valorFloatE1 = 1.2e3;
$valorFloatE2 = 7E-10;
?>
```

Cuando un número `integer` se sale del rango permitido, PHP lo convierte automáticamente al tipo `float`.

is_int(), is_float(), is_real(), is_numeric()

Para verificar si una variable es del tipo integer o float podemos utilizar varias funciones:
• `is_int()`
Regresa true si la variable es un número entero, false de lo contrario.
• is_integer()
Esta función en un alias de is_int(), funciona de la misma manera.
• `is_float()`
Regresa true si la variable es un numero float, false de lo contrario.
• `is_real()`
Esta función en un alias de is_float(), funciona de la misma manera.
• `is_numeric()`

Esta función determina si una variable es numérica (integer o float) o si un string representa un numero.

Variables, expresiones y funciones

NULL

NULL es un valor especial que significa que una variable no tiene valor, una variable se considera NULL en estos casos:

• Se le asigno explícitamente el valor de NULL
• No ha sido declarada
• Se le aplica la función `unset()`

```
<?php
$variableNULL1;
$variableNULL2 = NULL;
$variableNULL3 = 10;
unset($variableNULL3);
is_null(variableNULL1);// FALSE
is_null(variableNULL2);// TRUE
is_null(variableNULL3);// TRUE
is_null(variableNULLx);// TRUE
?>
```

is_null()

Para verificar si una variable es NULL podemos utilizar la función `is_null()`, regresa true si la variable es NULL, false de lo contrario.

Variables, expresiones y funciones

object

Una variable del tipo `object` es aquella que representa una instancia o copia de una clase de PHP. Una clase es la definición de un objeto. Ejemplo:

```
<?php
class elObjeto
{
function miFuncion()
{
// codigo...
}
}
$miObjeto = new elObjeto;
$miObjeto->miFuncion();
?>
```

is_object()

Para verificar si una variable es de este tipo, podemos utilizar la función `is_object()`, regresa true si la variable representa una instancia de un objeto, false de lo contrario. Ejemplo:

```
is_object($variable)
```

resource

Una variable del tipo `resource` es una referencia de un recurso externo, los cuales son creados en PHP por cierto tipo de funciones. Ejemplos de recursos externos son archivo abiertos o conexiones con bases de datos.

is_resource()

Para verificar si una variable es de este tipo, podemos utilizar la función `is_resource()`, regresa true si la variable representa una instancia de un objeto, false de lo contrario. Ejemplo:

```
is_resource($variable)
```

Variables, expresiones y funciones

Manipulando los tipos de datos

Hay varias funciones para manipular los tipos de datos:

- gettype()
- settype()
- Type Casting

gettype()

Si necesitamos desplegar el tipo asignado a una variable, podemos utilizar la función `gettype()`, por ejemplo:

```
gettype($variable)
```

Esta función regresa uno de estos valores: boolean, integer, double (es lo mismo que float), string, array, object, resource, null y unknown type.

Los valores que regresa la función son strings, y no deben ser utilizados en una expresión para determinar el tipo de datos que contiene una variable. Para lo anterior se debe utilizar las funciones is_int(), is_float(), is_numeric(), is_string(), is_null(), is_array(), is_object() e is_resource().

```
<?php
$boleano = false;
$texto = "texto";
$entero = 10;
$flotante = 4587.1154
$arreglo = array();
$nulo = NULL;
echo "\$boleano = " . gettype($boleano);// boolean
echo "\$texto = " . gettype($texto);//string
echo "\$entero = " . gettype($entero);//integer
echo "\$flotante = " . gettype($flotante);//double
echo "\$arreglo = " . gettype($arreglo);//array
echo "\$nulo = " . gettype($nulo);//NULL
echo "\$noDefinida = " . gettype($noDefinida);//NULL
// OJO echo "\$noDefinida = gettype($noDefinida)"; NO ES VALIDO
!!!
```

```
// Esto no es recomendable
if (gettype($nulo) == "NULL")
{
echo "gettype() - Nulo !<br>";
}
// Esta es la manera recomendada
if (is_null($nulo))
{
echo "is_null() - Nulo !";
}
?>
```

Variables, expresiones y funciones

settype()

Esta función nos permite especificar el tipo de datos que contiene una variables, regresa true si logra definir el tipo de datos, false de lo contrario. Su sintaxis es la siguiente:

```
settype(variable, "tipo_de_dato")
```

Los tipos de datos que se pueden especificar son los siguientes:

- "boolean" o "bool"
- "integer" o "int"
- "float"
- "string"
- "array"
- "object"
- "null"

```
<?php
$boleano = true;
$texto = "texto";
$texto2 = "10abc";
$entero = 10;
$flotante = 4587.1154;
$arreglo = array();
$nulo = NULL;
settype($boleano,"string"); // 1
settype($texto,"integer"); // 0
settype($texto2,"integer"); // 10
settype($entero,"array"); // array
settype($flotante,"integer"); // 4587
settype($arreglo,"string"); // Array
settype($nulo,"float"); // 0
?>
```

Variables, expresiones y funciones

Type Casting

Es posible especificar el tipo de dato de una variable al momento de asignarle un valor, de esta manera no es necesario utilizar `settype()`, a esto se le llama *type casting*, por ejemplo:

```
<?php
$texto = "10";
$numero = (integer) $texto; // $numero = 10
?>
```

Las opciones que se pueden especificar son las siguientes:

- (int), (integer) - integer
- (bool), (boolean) - boleano
- (float), (double), (real) - float
- (string) - string
- (array) - array
- (object) - object

Para convertir un valor numérico en string, también se puede hacer de esta manera:

```
<?php
$numero = 10;
$texto = "$numero"; // $texto = "10"
?>
```

Constantes

define()

Como en otros lenguajes, una vez que se le ha asignado un valor a una constante en PHP esta no puede cambiar en el resto del código. Los nombres de las constantes son case-sensitive por default y se recomienda que estén en mayúsculas, las constantes se definen con la función `define()`. cuya sintaxis es:

```
define(nombre_de_la_constante, valor [, case_insensitive])
```

A una constante se le pueden asignar los siguientes valores: boolean, integer, float y string. Para desplegar una constante no se utiliza el símbolo $, solo el nombre de la constante.
Aquí tenemos algunos ejemplos:

```
<?php
define("MENSAJE_DE_ERROR", "Se ha presentado un error");
define("IVA", 0.15);
define("NO_ES_CASESENSITIVE", "valor_no_caseSensitive", true);
echo MENSAJE_DE_ERROR; // "Se ha presentado un error"
echo IVA; // 0.15
echo iva; // "iva"
echo NO_ES_CASESENSITIVE; // "valor_no_caseSensitive"
echo no_es_casesensitive; // "valor_no_caseSensitive"
// OJO esto no es valido:
echo "MENSAJE_DE_ERROR <br>"; // "MENSAJE_DE_ERROR"
?>
```

constant()

Otra manera de obtener el valor de una constante es con la función `constant()`:

```
<?php
define("MENSAJE_DE_ERROR", "Se ha presentado un error.");
define("IVA", 0.15);
echo constant("MENSAJE_DE_ERROR"); // "Se ha presentado un
error."
echo constant("IVA"); // 0.15
echo constant("iva"); // No es valido arreja una advertencia
// OJO esto no es valido:
echo "constant(MENSAJE_DE_ERROR) <br>"; //
"constant(MENSAJE_DE_ERROR)"
?>
```

get_defined_constants()

Para desplegar una lista con todas las constantes definidas podemos utilizar la función `get_defined_constants()`, esta regresa las constantes predefinidas de PHP y las que hayamos definido en el código.

Esta función regresa un arreglo con los nombres de las constantes y sus valores, para desplegar esta información podemos utilizar un `foreach`, ejemplo:

```
<?php
foreach (get_defined_constants() as $constante => $valor)
{
echo "<strong>$constante</strong> = $valor<br>";
}
?>
```

defined()

Si queremos verificar si una constante esta definida o no, podemos utilizar la función defined(), la cual regresa un true si esta definida, false de lo contrario.

```
<?php
define("MENSAJE_DE_ERROR", "Se ha presentado un error.");
if (defined("MENSAJE_DE_ERROR"))
{
echo MENSAJE_DE_ERROR; // "Se ha presentado un error."
}
?>
```

Funciones especiales para el manejo de variables

Entre las funciones especiales de PHP que facilitan manipular las variables tenemos las siguientes:

* isset
* empty
* unset
* get_defined_vars
*print_r
* var_dump
* var_export

isset

Determina si una variable existe o no, si existe regresa un true, false de lo contrario. Esta función puede verificar una lista de variables, si todas las variables existen regresa un true, si una o mas de las variable no existen regresa un false.

```
<?php
$variableExiste1 = 10;
$variableExiste2 = 10;
if (isset($variableExiste1))
{
echo "\$variableExiste1 = $variableExiste1";
}
if (!isset($variableNoExiste))
{
echo "\$variableNoExiste No existe !!!!";
}
if (isset($variableExiste1, $variableExiste2))
{
echo "Todas las variables existen.";
}
if (!isset($variableExiste1, $variableExiste2,
$variableNoExiste))
{
echo "Una o mas variables no existen !!!!";
}
?>
```

empty

Determina si una variable esta vacía o no, si esta vacía regresa un true, false de lo contrario. Una variable esta vacía cuando se cumple una de estas condiciones:

- Nunca se ha declarado
- No se le ha asignado ningún valor
- Es igual a cero numérico 0
- Es igual a cero string "0"
- Es igual a NULL

```
<?php
$variableExiste1 = 0;
$variableExiste2 = 10;
if (empty($variableExiste1)) // true
{
echo "\$variableExiste1 es empty !!!!<br>";
}
$variableExiste3;
if (empty($variableExiste3)) // true
{
echo "\$variableExiste3 es empty !!!!<br>";
}
if (empty($variableNoExiste)) // true
{
echo "\$variableNoExiste es empty !!!!<br>";
}
?>
```

get_defined_vars

Regresa un arreglo multidimensional con todas las variables definidas, estas incluyen las variables predefinidas por PHP y las que hayamos definido nosotros. Ejemplo:

```
<?php
foreach (get_defined_vars() as $variable => $valor)
{
echo "<strong>$variable</strong> = $valor<br>";
}
?>
```

unset

Desecha una o varias variables, la sintaxis es:

```
unset($variable1,$variable2,.....,$variableN)
```

Tambien podemos usar esta función para eliminar un elementos de un arreglo. Ejemplo:

```
<?php
$variable1 = 10;
$variable2 = 20;
$variable3 = 20;
$arreglo = array(1,2,"elemento3");
unset(variable1);
unset(variable2, variable3);
// Esto elimina el tercer elemento "elemento3"
// En un arreglo el primer elemento inicia en cero
unset($arreglo[2]);
// OJO esto no es valido en un unset:
// unset($arreglo["elemento3"]);
?>
```

print_r

Esta función regresa información legible sobre una variable. Esta información la despliega directamente al navegador, no es necesario utilizar echo. Si se le pide información de un numero o string, simplemente regresa el valor que tiene asignado. En el caso de un arreglo, regresa una descripción de cada llave y su respectivo valor.

```
<?php
$numero = 10;
$texto = "texto";
$arreglo = array("elemento1" => "valor1", "elemento2" =>
"valor2",
"elemento3" => "valor3");
print_r($numero); // 10
print_r($texto); // texto
print_r($arreglo); // Array ( [elemento1] => valor1 [elemento2]
=> valor2
// [elemento3] => valor3 )
?>
```

var_dump

Regresa información sobre una o mas variables, a diferencia de `printr`, además del valor de las variables, esta regresa el tipo de dato.

```
<?php
$numero = 10;
$texto = "texto";
$arreglo = array("elemento1" => "valor1", "elemento2" =>
"valor2",
"elemento3" => "valor3");
var_dump($numero); // int(10)
var_dump($texto); // string(5)"texto"
var_dump($arreglo); // array(3) { ["elemento1"]=> string(6)
"valor1"
// ["elemento2"]=> string(6) "valor2" ["elemento3"]=> string(6)
"valor3" }
?>
```

var_export

Regresa información sobre una o varias variables.

```
<?php
$numero = 10;
$texto = "texto";
$arreglo = array("elemento1" => "valor1", "elemento2" =>
"valor2",
"elemento3" => "valor3");
var_export($numero); // 10
var_export($texto); //  texto
var_export($arreglo); // array ( 'elemento1' => 'valor1',
// 'elemento2' => 'valor2', 'elemento3' => 'valor3', )
?>
```

Operadores

Operadores aritméticos

Como su nombre lo indica, estos operadores se utilizan para realizar operaciones aritméticas. Los operadores son los siguientes:

Operador	Descripción
+	Suma $resultado = $numero1 + $numero2
-	Resta $resultado = $numero1 - $numero2
*	Multiplicación $resultado = $numero1 * $numero2
/	División $resultado = $numero1 / $numero2
%	Remanente $resultado = $numero1 % $numero2

Operadores de comparación

Operadores de comparacion

Operador	Descripción El resultado de la comparación entre dos variables o expresiones
==	Igual `$valor1 == $valor2` TRUE si `$valor1` es igual a `$valor2`
===	Idéntico `$valor1 === $valor2` TRUE si `$valor1` es igual y además es del mismo tipo que `$valor2`
!=	Diferente `$valor1 != $valor2` TRUE si `$valor1` es diferente de `$valor2`
<>	Diferente `$valor1 <> $valor2` TRUE si `$valor1` es diferente de `$valor2`
!==	No idéntico `$valor1 !== $valor2` TRUE si `$valor1` es diferente o si no es del mismo tipo que `$valor2`
<	Menor `$valor1 < $valor2` TRUE si `$valor1` es menor que `$valor2`
>	Mayor `$valor1 > $valor2` TRUE si `$valor1` es mayor que `$valor2`
<=	Menor o igual `$valor1 <= $valor2` TRUE si `$valor1` es menor o igual que `$valor2`
>=	Mayor o igual `$valor1 >= $valor2` TRUE si `$valor1` es mayor o igual que `$valor2`
?	Terciario `$resultado = (expresion1) ? (expresion2) : (expresion3)` En este caso se evalúa `expresion1`, si evalúa TRUE entonces se evalúa `expresion2` y el resultado se asigna a `$resultado`. Si `expresion1` evalúa a FLASE, entonces se evalúa `expresion2` y el resultado se asigna a `$resultado`.

Operadores lógicos

PHP nos ofrece los siguientes operadores lógicos:

Operador	Descripción El resultado de la comparación entre dos variables o expresiones
`and`	TRUE si ambas variables o expresiones son a su vez TRUE.
`&&`	Equivalente al `and`, pero con mayor precedencia.
`or`	TRUE si cualquiera de las variables o expresiones son a su vez TRUE.
`||`	Equivalente al `or`, pero con mayor precedencia.
`!`	Negación, TRUE si la variable o expresión no es TRUE.
`xor`	TRUE si alguna de las variables o expresiones es a su vez TRUE, pero si mas de una variable o expresión es TRUE, entonces `xor` regresa FALSE. También si ambas variables o expresiones son FLASE, regresa un FALSE.

Operadores de asignación

PHP permite la asignación de valores con el operador =, o bien con operadores combinados. Por ejemplo:

```
<?php
$numero1 = 10;
$texto1 = "string1...";
$numero1 += 10; // $numero1 = 20
$numero1 -= 5; // $numero1 = 15
$numero1 *= 2; // $numero1 = 30
$numero1 /= 2; // $numero1 = 15
$texto1 .= "string2..."; // $texto1 = "string1...string2"
$numero2 = ($numero3 = 5) + 5; // $numero2 = 10, $numero3 = 5
?>
```

Operadores unitarios

Como en otros lenguajes, PHP soporta los operadores unitarios para incrementar y decrementar variables. Los operadores unitarios son los siguientes:

Operador	Descripción
`++$variable`	Incrementa el valor de `$variable` y regresa el valor de la variable ya incrementada.
`$variable++`	Incrementa el valor de `$variable` y regresa el valor de la variable antes de ser incrementada.
`--$variable`	Decrementa el valor de `$variable` y regresa el valor de la variable ya decrementada.
`$variable--`	Decrementa el valor de `$variable` y regresa el valor de la variable antes de ser decrementada.

por ejemplo:

```
<?php
$numero1 = 1;
$texto1 = "a";
++$numero1; // 2
$numero1++; // 2, aunque el valor internamente si cambia a 3
--$numero1; // 2
$numero1--; // 2, aunque el valor internamente si cambia a 1
++$texto1; // b
$texto1++; // b, aunque si el valor internamente si cambia a c
// OJO en el caso de caracteres, no se decrementa
--$texto1; // c
$texto1--; // c
?>
```

Funciones

Como muchos otros lenguajes PHP soporta la creación de funciones definidas por el usuario, la sintaxis básica es la siguiente:

```
<?php
function nombre($argumento1, ..., $argumentoN)
{
//codigo php...
return $valor-que-regresa;
}
?>
```

Si se requiere se le puede enviar parametros a la función, si una función tiene que regresar un resultado se utiliza el `return`, aquí tenemos un ejemplo:

```
<?php
$valor1 = 5;
$valor2 = 5;
$valor3 = multiplica1($valor1, $valor2);
echo $valor3; // 25
multiplica2(10,10);
multiplica3();
function multiplica($numero1,$numero2)
{
return $numero1 * $numero2;
}
function multiplica2($numero1,$numero2)
{
echo $numero1 * $numero2; // 100
}
```

```
function multiplica3()
{
echo 100 * 100; // 10000
}
?>
```

Argumentos por valor

Por default, los argumentos que utiliza una función los recibe por valor, esto es, recibe una copia del valor original. De esta manera si la función altera el valor de la copia, el original no se ve afectado.

```
<?php
$valor1 = 5;
$valor2 = 5;
echo $valor1; // 5
echo $valor2; // 5
$valor3 = multiplica($valor1, $valor2);
echo $valor3; // 100
echo $valor1; // 5
echo $valor2; // 5
function multiplica($valor1,$valor2)
{
$valor1 += 5;
$valor2 += 5;
return $valor1 * $valor2;
}
?>
```

Argumentos por referencia

En caso que sea necesario una función puede recibir un valor como una referencia que apunta hacia el valor original. De esta manera al cambiar el valor que recibe afecta el valor original, lo que se hace en este caso es agregar el carácter & al declarar los argumentos:

```
<?php
$valor1 = 5;
$valor2 = 5;
echo $valor1; // 5
echo $valor2; // 5
$valor3 = multiplica($valor1, $valor2);
echo $valor1; // 10
echo $valor2; // 10
echo $valor3; // 100
function multiplica(&$valor1, &$valor2)
{
$valor1 += 5;
$valor2 += 5;
return $valor1 * $valor2;
}
?>
```

Argumentos con valores default

PHP permite asignar un valor defautl a un argumento al momento que se declara. La utilidad de este valor default es que si se omite el valor al mandar llamar la función automáticamente se le asigna el valor default al argumento.

```
<?php
$valor1 = 5;
$valor2 = 5;
$valor3 = multiplica($valor1);
echo $valor1; // 5
echo $valor2; // 5
echo $valor3; // 25
function multiplica($valor1, $valor2 = 5)
{
return $valor1 * $valor2;
}
?>
```

Si se van a asignar valores default a uno o varios argumentos primero se deben declarar los argumentos que no tienen valores default, y posteriormente los que si tienen.

```
<?php
// Correcto
function multiplica($valor1, $valor2 = 5)
{
return $valor1 * $valor2;
}
// Incorrecto
function multiplica($valor2 = 5, $valor1)
{
return $valor1 * $valor2;
}
?>
```

Alcance de las variables en las funciones

Las variables que se declaran fuera de una función en una pagina PHP se consideran globales, esto es, se pueden referenciar en cualquier parte del código.
Sin embargo, si una función utiliza una variable que esta declarada también fuera de la función, entonces se crea una variable local con el mismo nombre.

```
<?php
$valor1 = 5;
$valor2 = 5;
$valor3 = multiplica();
echo "Despues de mandar llamar la funcion<br>";
echo "\$valor1 = $valor1<br>"; // 5
echo "\$valor2 = $valor2<br>"; // 5
echo "\$valor3 = $valor3<br>"; // 25 no es 100
function multiplica()
{
/*
Se crean nuevas variables $valor1 y $valor2 con alcance local.
Las variables globales no se ven afectadas.
En lugar de que $valor1 y $valor2 se incrementen a 10, se les
asigna un 5
*/
$valor1 += 5;
$valor2 += 5;
return $valor1 * $valor2;
}
?>
```

Hay dos maneras de referenciar las variables globales en las funciones: con el keyword `global` y con el arreglo `$GLOBALS[]`.

```
<?php
function multiplica2()
{
global $valor1, $valor2;
$valor1 += 5;
$valor2 += 5;
return $valor1 * $valor2;
}
function multiplica3()
{
$GLOBALS["valor1"] += 5;
$GLOBALS["valor2"] += 5;
return $GLOBALS["valor1"] * $GLOBALS["valor2"];
}
?>
```

Variables estáticas

Otro concepto que maneja PHP es de la variables estáticas, las cuales se manejan solo dentro de una función. Las variables normales pierden su valor una vez que se termina de ejecutar la función, las variables estáticas conservan su valor.
Para utilizar una variable estética se utiliza el keyword `static`, por ejemplo:

```
<?php
function incrementa()
{
static $valor2 = 0;
++$valor1;
++$valor2;
echo "\$valor1 = $valor1<br>";
echo "\$valor2 = $valor2<br><br>";
}
?>
```

En este ejemplo cada vez que se mande llamar la función `incrementa()`, `$valor2` se incrementa en uno y conserva su valor al salir de la función. Por otra parte, `$variable1` se incrementa en uno y pierde su valor al salir de la función, por lo que siempre tiene un valor de 1.

Funciones variables

El concepto de funciones variables consiste en que cuando se tiene una variable con paréntesis PHP buscara y ejecutara la función con el nombre al cual evalúa la variable.

```
<?php
$variable1 = "multiplica";
$variable1(); // se ejecuta la funcion multiplica()
function multiplica()
{
echo 5 * 5 . "<br><br>";
}
?>
```

En este caso se despliega en pantalla el resultado de la multiplicación.

function_exists

Esta función nos sirve para determinar si una función ha esta definida o no, de ser así regresa un TRUE, de lo contrario regresa un FALSE.

```
<?php
if (function_exists("multiplica"))
{
echo "Si existe la funcion multiplica()";
}
if (! function_exists("suma"))
{
echo "No existe la funcion suma()";
}
function multiplica()
{
echo 5 * 5 . "<br><br>";
}
?>
```

get_defined_functions

Esta función regresa un arreglo con todas las funciones que están definidas.

NOTA

Esta función puede tardar mucho en desplegar el resultado ya que se incluyen todas la funciones de PHP además de las que haya definido el usuario.

```
<?php
$funciones = get_defined_functions();
?>
```

Funciones especiales para el manejo de strings

trim

Remueve los espacios en blanco a la derecha e izquierda de un string, tambien se le puede especificar una lista de caracteres a remover.

```
<?php
$texto1 = " direccion@dominio.com ";
$texto1 = trim($texto1); // "direccion@dominio.com"
$texto2 = "abcdefghi";
$texto2 = trim($texto2,"a..e"); // fghi
?>
```

strlen

Obtiene cuantos caracteres hay en un string.

```
<?php
$texto1 = "123456789";
echo strlen($texto1); // 9
?>
```

strpos

Encuentra la posición de la primera ocurrencia de un string, en caso que no encuentre el string regresa un FALSE.
El primer carácter de un string empieza a contar desde el cero, así que para no confundir entre la posición cero y el FALSE (que internamente es un cero) se utiliza el operador `===` para determinar si regreso o no un FALSE.

```
<?php
$texto = "123456789";
$posicion = strpos($texto,"5");// regresa 4
if(!$posicion === false)
{
echo "Posicion del caracter \"5\" en $texto = $posicion";
}
?>
```

Para buscar un carácter especial en un string, por ejemplo, comillas se utiliza su carácter de escape:

```
$posicion = strpos($texto,"\"");
```

NOTA
Esta función es case sensitive.

strrpos

Funciona igual que `strpos`, pero encuentra la última ocurrencia del string.

strstr

Encuentra la primer ocurrencia de un string y regresa el resto del string incluyendo el que se esta buscando.

```
<?php
$texto = "123456789";
$textoExtraido = strstr($texto,"5");// regresa 56789
if(!$textoExtraido === false)
{
echo "\$textoExtraido = $textoExtraido";
}
?>
```

NOTA
Esta función es case sensitive.

strrchr

Funciona igual que `strstr`, pero encuentra la última ocurrencia del string.

substr

Regresa parte de un string, la parte del string que regresa se define en base al carácter donde inicia la extracción, mas el numero de caracteres que se indiquen que se quiere extraer. La sintaxis es:

```
substr(string-en-el-que-extraemos,caracter-de-inicio,cuantos-caracteres-aextraer)
```

En esta función el carácter de inicio puede ser un numero positivo o negativo, el carácter positivo inicia desde cero en el primer carácter. El carácter negativo inicia desde -1 en el ultimo carácter, el penúltimo carácter seria el -2, y así sucesivamente.

```
<?php
$texto = "1234567890";
$textoExtraido = substr($texto,4); // 567890
$textoExtraido = substr($texto,0,5); // 12345
$textoExtraido = substr($texto,-1); // 0
$textoExtraido = substr($texto,-3,2); // 89
?>
```

substr_count

Esta función regresa el numero de veces que un string ocurre en otro.

```
<?php
$texto = "a b c d a";
$cuantasOcurre = substr_count($texto,"a"); // 2
?>
```

NOTA
Esta función es case sensitive.

substr_replace

Remplaza un string con otro, la sintaxis es:

```
substr_replace(string-en-el-que-remplazamos, string-que-
remplaza-alexistente,caracter-de-inicio, cuantos-caracteres-a-
remplazar)
```

En esta función el carácter de inicio puede ser un número positivo o negativo, el carácter positivo inicia desde cero en el primer carácter. El carácter negativo inicia desde -1 en el ultimo carácter, el penúltimo carácter seria el -2, y así sucesivamente.

```
<?php
$texto = "1234567890";
$textoRemplazado = substr_replace($texto, "abc", 0,3); //
abc4567890
$textoRemplazado = substr_replace($texto, "abc", 4,3); //
1234abc890
?>
```

str_repeat

Repite un string un numero determinado de veces.

```
<?php
echo str_repeat("-_-",5); // -_--_--_--_--_-
?>
```

str_shuffle

Aleatoriamente combina los caracteres de un string.

```
<?php
echo str_shuffle("a1b2c3d4e5"); // 4a52c1b3de (varia al
ejecutarse)
?>
```

ucfirst

Convierte el primer carácter de un string en mayúsculas.

```
<?php
echo ucfirst("esta es una linea"); // Esta es una linea
?>
```

ucwords

Convierte el primer carácter de cada palabra en un string en mayúsculas.

```
<?php
echo ucwords("esta es una linea"); // Esta Es Una Linea
?>
```

strtolower

Convierte un string a minúsculas.

```
<?php
echo strtolower("EN MINUSCULAS"); // en minusculas
?>
```

strtoupper

Convierte un string a mayúsculas.

```
<?php
echo strtoupper("en mayusculas"); // EN MAYUSCULAS
?>
```

wordwrap

Aplica un salto de renglón en un número determinado de caracteres utilizando un carácter especifico.

```
<?php
$texto8 = "Linea de texto muy larga con mucho texto";
$texto9 = wordwrap($texto8, 10);
echo "<pre>$texto9\n</pre>";
/*
Regresa:
Linea de
texto muy
larga con
mucho
texto
*/
?>
```

number_format

Aplica formato numérico a una variable, la sintaxis es:

```
number_format(numero, cuantos-decimales, caracter-de-decimales ,
caracterde-miles)
```

```
<?php
$numero1 = 1500000.566;
echo number_format($numero1, 2, "." , ","); // 1,500,000.57
?>
```

implode

Convierte un arreglo es un string concatenando los elementos y separándolos por un carácter determinado, la sintaxis es:

```
implode(caracter-separador, arreglo)
```

```
<?php
$arreglo = array(10,"elemento 2",30);
$arregloConvertido = implode(",",$arreglo); // 10,elemento 2,30
?>
```

explode

Convierte los elementos delimitados por un carácter de un string en un arreglo, la sintaxis es:

```
explode(caracter-separador, string)
```

```
<?php
$texto = "10,elemento 2,30";
$textoConvertido = explode(",",$texto); // Array ( [0] => 10 [1]
=>
elemento 2 [2] => 30 )
?>
```

htmlentities

Convierte los caracteres aplicables a entidades HTML, por ejemplo:

```
<?php
$texto = htmlentities("Entidades HTML: <, >, &");
echo $texto; // Entidades HTML: <, >, &
/*
Al examinar el codigo fuente los caracteres se convierten a:
Entidades HTML: &lt;, &gt;, &amp;
*/
?>
```

html_entity_decode

Convierte entidades HTML a sus respectivos caracteres, por ejemplo:

```
<?php
$texto = html_entity_decode("Entidades HTML: &lt;, &gt;,
&amp;");
echo $texto; // Entidades HTML: <, >, &
/*
Al examinar el codigo fuente los caracteres se convierten a:
Entidades HTML: <, >, &
*/
?>
```

urlencode

Convierte o codifica los caracteres que aplican para que sean validos en un URL, por ejemplo:

```
<?php
$nombre = "Fulano de Tal";
$comentarios = "Renglon 1...\nRenglon2";
$texto13 = "<a href=\"procesa.php?nombre=" . urlencode($nombre)
.
"&comentarios=" . urlencode($comentarios) . "\">Liga</a>";
/*


Al examinar el codigo fuente se convierte a:
<a
href="procesa.php?nombre=Fulano+de+Tal&comentarios=Renglon+1...%
0ARenglon2"
>Liga</a>
*/
?>
```

urldecode

Convierte o decodifica caracteres de un URL en caracteres regulares, por ejemplo:

```
<?php
$nombre = urldecode("Fulano+de+Tal"); // Fulano de Tal
$comentarios = urldecode("Renglon+1...%0ARenglon2"); // Renglon
1...
Renglon2
?>
```

nl2br

Inserta saltos de línea HTML (
) en cada nueva línea, por ejemplo:

```
<?php
$texto14 = "linea 1\nlinea 2\nlinea 3";
echo nl2br($texto14);
/* Al examinar el codigo fuente econtramos los <br>:
linea 1<br />
linea 2<br />
linea 3
*/
?>
```

NOTA

`<br />` da el mismo resultado que `<br>`. La primera se apega al standard XHTML, la segunda al HTML.

Controlando la ejecución del código

Como otros lenguajes de programación PHP cuenta varios mecanismos para controlar la ejecución del código.
Para controlar que secciones de código se ejecutan y cuales no, podemos utilizar:

- `if`
- `else`
- `elseif`
- `switch`

Para manejar ciclos o bucles tenemos:

- `while`
- `do..while`
- `for`
- `foreach`

if

Esta instrucción ejecuta una línea o un bloque de código dependiendo de si se cumple o no una o varias condiciones. La sintaxis básica para ejecutar una sola línea de código es la siguiente:

```
if(expresion)
     codigo php
```

Si se van ejecutar varias líneas se requieren las llaves para delimitar el código:

```
if(expresion)
{
   código php...
   código php...
 }
```

else

Si no se cumple una condición del `if`, podemos utilizar esta instrucción para ejecutar una línea o un bloque de código. La sintaxis básica para ejecutar una sola línea de código es la siguiente:

```
if(expresion)
  código php
else
  código php
```

Si se van ejecutar varias líneas se requieren las llaves para delimitar el código:

```
if(expresion)
{
  codigo php
}
else
{
   código php
}
```

elseif

Aunque se pueden anidar varios `if` y `else`, se puede utilizar el `elseif` para mejorar la legibilidad del código.
La sintaxis básica para ejecutar una sola línea de código es la siguiente:

```
if(expresion1)
  código php
elseif(epxresion2)
  código php
elseif(epxresionN)
  código php
else
   código php
```

Si se van ejecutar varias líneas se requieren las llaves para delimitar el código:

```
if(expresion1)
{
  código php...
  código php...
}
elseif(expresion2)
{
   código php...
   código php...
}


elseif(expresionN)
{
   código php...
   código php...
}
else
{
   código php...
   código php...

}
```

Ejercicio - Validando una página de login

El siguiente ejercicio consiste en capturar los valores de una forma HTML que incluye los elementos más comúnmente utilizados.

switch

Cuando se tiene una lista larga de `if..elseif..else` el código se puede volver difícil de mantener y leer, switch puede ser una mejor opción.

```
switch(expresion)
{
case 0:
código php...
break;
case 1:
código php...
break;
case N:
código php...
break;
default:
código php...
}
```

`break` sirve para detener la ejecución del código una vez que se cumple la condición.
El valor de cada `case` debe ser un numero o string, no se recomienda utilizar expresiones ya que puede marcar errores de sintaxis o bien es mejor utilizar un `if`.

while

Esta instrucción ejecuta una línea o un bloque de código repetidamente hasta que se cumple una condición. La sintaxis básica para ejecutar una sola línea de código es la siguiente:

```
while(expresion)
  código php
```

Para ejecutar varias líneas de código se utilizan las llaves:

```
while(expresion)
{
   código php
}
```

do...while

A diferencia del `while`, este estatuto verifica la condición al final del ciclo en lugar de al inicio.

```
do
{
    código php
} while (expresión);
```

En un `while` podemos "brincarnos" la ejecución de cierto bloque de código con `continue`, por ejemplo:

```
while(expresion)
{
   código php
   if(expresion)
   {
      continue;
   }
   código php
}
```

También podemos detener la ejecución del ciclo en cualquier momento con `break`, ejemplo.

```
while(expresion)
{
   código php
   if(expresion)
   {
      break;
   }
   código php
}
```

for

Se utiliza frecuentemente para ejecutar un bloque de código un número definido de veces, los criterios que utiliza para lo anterior comúnmente son variables numéricas. La sintaxis de este comando es un poco más compleja:

```
for(expresion1; expresion2; expresion3)
{
código php
código php
}
```

`expresion1` se ejecuta solo al inicio del ciclo, `expresion2` se evalúa en cada ciclo y si evalúa a TRUE se ejecuta el código. `expresion3` se ejecuta en cada ciclo. Ejemplo:

```
for($contador = 1; $contador <= 10; $contador++)
{
código php
código php
}
```

En este ejemplo la condición para ejecutar el ciclo es que `$contador` sea menor a 11, la variable se inicializa en 1 y se incrementa en uno en cada ciclo.
En el `for` se pueden utilizar también el `continue` y `break`.

foreach

Se utiliza para obtener los elementos de un arreglo, un arreglo esta formado por pares de llaves y sus respectivos valores, por ejemplo:

```
$arreglo = array("llave1" => "valor1", "llave2" => "valor2");
```

Para obtener la información de los elementos podemos utilizar dos sintaxis:

```
foreach($arreglo as $valor)
```

Extrae únicamente los valores de las llaves: `valor1` y `valor2`

```
foreach($arreglo as $llave => $valor)
```

Extrae tanto los nombres de las llaves como sus valores.

Ejercicio - Utilizando funciones de ciclos

El siguiente ejercicio consiste en capturar los valores de una forma HTML que incluye los elementos más comúnmente utilizados.

exit()

Este estatuto termina la ejecución del script de manera inmediata, y puede desplegar un numero o string antes de terminar la ejecución del código. Si se utiliza un número se debe utilizar uno entre 1 y 254.

```
exit("Mensaje de salida");
exit(150);
```

Un uso común es el de terminar la ejecución del script si algo lo suficientemente grave ha ocurrido, para lo cual se puede utilizar combinándolo con el llamado de una función, por ejemplo:

```
$resultado = conexionBD() or exit("Error en la conexion.");
```

En este ejemplo si la función `conexionBD()` regresa un TRUE continua la ejecución del código, si regresa un FALSE se ejecuta el `exit()`.

die()

Es un alias de `exit()` y funciona de la misma manera.

Manejando código externo

PHP nos permite incluir archivos externos que contengan código en la página actual, para esto podemos utilizar los siguientes estatutos:

- `requiere()`
- `require_once()`
- `include()`
- `include_once()`

require()

Incluye y ejecuta un archivo externo en la pagina, los archivos puede estar en el mismo directorio o se puede indicar un URL completo indicando su ubicación. Hay varias maneras de especificar el archivo externo:

```
require(´externo.php´);
require ’externo.php’;
require "externo.php";
require $variableExterno;
```

El archivo externo se "vacía" en el lugar donde se mande llamar con `require()`, si el archivo externo no existe se dispara un error.

Las variables que se encuentren en el archivo externo asumen el alcance o *scope* del lugar donde se mandan llamar. Por ejemplo, si se utiliza el `require()` dentro de una función, entonces las variables son locales a la función.
El archivo externo se puede vaciar mas de una vez en la misma pagina, por lo que hay que tener precaución, ya que se pueden alterar los valores de las variables y redefinir funciones.

require_once()

Funciona de la misma manera que `require()`, solo que únicamente incluye el archivo externo una sola vez en la pagina. Se puede utilizar `require_once()` varias veces indicando el mismo archivo externo y solo se incluye una vez.

include()

Funciona de la misma manera que `require()`, solo que si no se encuentra el archivo externo no marca error, solo genera una advertencia de manera que no se detiene la ejecución del código.

include_once()

Funciona de la misma manera que `require_once()`, solo que si no se encuentra el archivo externo no marca error, solo genera una advertencia de manera que no se detiene la ejecución del código.