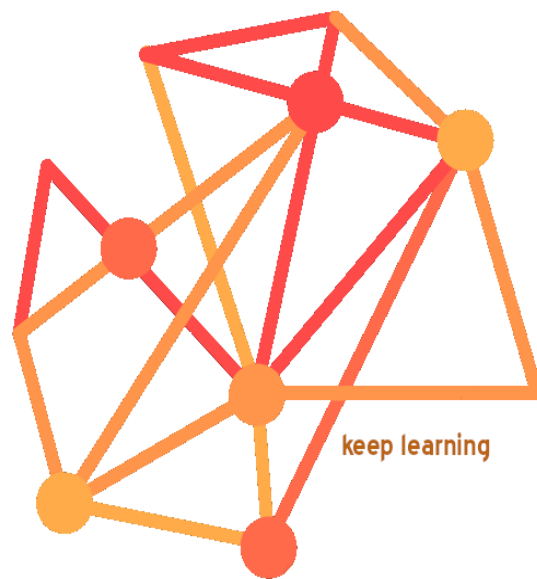


# Introducción PHP



Juan Carlos Pérez Rodríguez

## Sumario

Introducción.....	4
Tipos de datos y variables.....	6
Tipos de datos primitivos.....	7
strict_types=1.....	8
Variables asignadas por referencia.....	11
Variables predefinidas.....	12
Ámbito de una variable.....	12
Eliminar variables.....	13
Casting de variables.....	15
Constantes.....	16
Operadores.....	16
Cadenas en PHP.....	19
Comillas dobles.....	19
Uso de llaves.....	20
Comillas simples.....	20
Caracteres de escape.....	21
Variables de variables.....	21
Arrays en PHP.....	23
Arrays comunes, índices numéricos.....	23
Arrays asociativos.....	24
Recorrer array.....	25
Comportamiento LIFO con array.....	29
Comportamiento FIFO con array.....	30
Otras funciones con array.....	31
Funciones.....	35
¿ arrays por copia o referencia ?.....	36
Inclusión de ficheros externos.....	37
Formularios.....	38
Pasar variables por URL con PHP.....	39
Método GET, método POST.....	42
El método GET.....	42
El método POST.....	43
Enviar un formulario.....	44
Objetos.....	52
Métodos y atributos estáticos.....	55
self.....	55
uso de this y self.....	56
Comparando objetos.....	56
.....	59
Herencia en php.....	60
parent.....	61

Juan Carlos Pérez Rodríguez

# Introducción

PHP, acrónimo recursivo en inglés de PHP Hypertext Preprocessor (procesador de hipertexto), es un lenguaje de programación de propósito general de código del lado del servidor originalmente diseñado para el desarrollo web de contenido dinámico. Fue uno de los primeros lenguajes de programación del lado del servidor que se podían incorporar directamente en un documento HTML en lugar, de llamar a un archivo externo que procese los datos. El código es interpretado por un servidor web con un módulo de procesador de PHP que genera el HTML resultante.

PHP ha evolucionado por lo que ahora incluye también una interfaz de línea de comandos que puede ser usada en aplicaciones gráficas independientes. Puede ser usado en la mayoría de los servidores web al igual que en muchos sistemas operativos y plataformas sin ningún costo.

Una página PHP generalmente consiste de una página HTML con comandos PHP incrustados en ella. El servidor web procesa los comandos PHP y envía la salida al visualizador (browser). Un ejemplo de una página PHP sencilla sería la siguiente:

```
<html>

<head> <title>Hello, world</title> </head>

<body>

  <?php echo "Hello, world!"; ?>

</body>

</html>
```

El comando echo de PHP produce la salida que se inserta en la página HTML. Note que el código PHP se escribe dentro de los delimitadores `<?php` y `?>`.

Las instrucciones se separan con “;”, en el caso de ser la última instrucción no es necesario el punto y coma.

Los comentarios en PHP pueden ser:

Como en C o C++, /\*...\*/ ó //

Otro tipo de comentario de una línea es #, que comentará la línea en la que aparezca pero sólo hasta el tag ?> que cierra el código php.

Vamos a realizar nuestro primer guión php y haremos uso de la función phpinfo()

Nota: en php.net tenemos la descripción de las diferentes funciones php La siguiente información es tomada de allí

## phpinfo

(PHP 4, PHP 5, PHP 7)

phpinfo — Muestra información sobre la configuración de PHP

### Descripción

```
bool phpinfo ( [ int $what = INFO_ALL ] )
```

Muestra gran cantidad de información sobre el estado actual de PHP. Incluye información sobre las opciones de compilación y extensiones de PHP, versión de PHP, información del servidor y entorno (si se compiló como módulo), entorno PHP, versión del OS, rutas, valor de las opciones de configuración locales y generales, cabeceras HTTP y licencia de PHP.

Vamos a crear una carpeta para alojar proyectos php en nuestro usuario

y la enlazaremos desde la raíz de nuestro servidor apache:

```
mkdir $HOME/proyectos
```

```
sudo ln -s $HOME/proyectos /var/www/html/proyectos
```

Crearemos un fichero: \$HOME/proyectos/info.php

y pondremos:

```
<?php
    echo "<p style='background: pink'>Soy alumno. Este es mi primer script php";
    echo "</p>";
    phpinfo();
```

?>

Y ahora accederemos con el navegador a:

<http://localhost/proyectos/info.php>

● **Práctica 1:** Crear el script como se ha comentado sustituyendo “alumno” por nuestro nombre completo. Tomar captura de pantalla del resultado

## Tipos de datos y variables

En PHP todas las variables comienzan con el símbolo del dólar \$ y no es necesario definir una variable antes de usarla. Una variable como cualquier etiqueta en php empieza por una letra o guion bajo, seguido por cualquier número de letras, números o guiones bajos. Usando una expresión regular, se representaría de la siguiente manera: `[a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*`

Observar `0x7f-0xff` hace referencia al ASCII extendido ( de 127 a 255 ) y en el texto anterior se entiende como cualquier carácter también a esos códigos

De facto significa que podemos poner cualquier carácter español sin problemas. Ej.

Ejemplos de nombres de variables:

//las siguientes son todas válidas:

\$pañito = 5;

\$exponente<sup>a</sup> = 4;

\$celdaÇ = 2;

\$admiraја = 8;

\$interrogaçe = 9;

\$4site = 'aun no'; // inválido; comienza con un número

\$\_4site = 'aun no'; // válido; comienza con un carácter de subrayado

\$täyte = 'mansikka'; // válido; 'ä' es ASCII (Extendido) 228

Una misma variable puede contener un número y luego puede contener caracteres. El intérprete establecerá el tipo de dato internamente al hacer la inicialización de la variable ( tipado dinámico )

```
$variable = "texto"; //estamos con una variable de texto
```

```
$otravariabale = 8; // estamos con una variable numérica
```

A lo anterior le podemos poner la salvedad de los parámetros de función recibidos si estamos en el modo estricto

## Tipos de datos primitivos

escalares: boolean, integer, float, string

compuestos: array, object, callable, iterable

especiales: resource, NULL

Lo siguiente es tomado de php.net:

**Nota:** Para comprobar el tipo y el valor de una [expresión](#), utilice la función [var\\_dump\(\)](#).

Para obtener una representación legible por humanos de un tipo con propósitos de depuración, use la función [gettype\(\)](#). Para comprobar un cierto tipo, *no* emplee [gettype\(\)](#), si no las funciones [is\\_tipo](#). Algunos ejemplos:

```
<?php
$un_bool = TRUE;    // un valor booleano
$un_str  = "foo";   // una cadena de caracteres
$un_str2 = 'foo';   // una cadena de caracteres
$un_int  = 12;      // un número entero

echo gettype($un_bool); // imprime: boolean
echo gettype($un_str);  // imprime: string

// Si este valor es un entero, incrementarlo en cuatro
if (is_int($un_int)) {
    $un_int += 4;
}

// Si $un_bool es una cadena, imprimirla
// (no imprime nada)
if (is_string($un_bool)) {
    echo "Cadena: $un_bool";
}
?>
```

Para saber si es de un tipo de datos en concreto usaremos pues:

`is_array, is_bool, is_float, is_int, is_string, is_null, is_object, is_resource`

- **Práctica 2:** Crear el script anterior. Modificarlo para sumar a `$un_str` el valor de `$un_int` y mostrarlo en pantalla ¿qué ocurre ? .  
Sumar `$un_str` con `$un_str2` ¿qué ocurre ? ¿ se puede concatenar una cadena con comillas simples con una con comillas dobles ?

## strict\_types=1

Hemos dicho que php es de tipado dinámico. Pero podemos en las funciones forzar a que los parámetros sean como esperamos que deben ser:

```
<?php
    declare( strict_types=1);

    function sum( int $a, int $b): int{
        return $a + $b;
    }

    sum(1,2);
?>
```

Podemos observar que tenemos que establecer que sea estricto el chequeo de los tipos: `strict_types=1`

Es fundamental tener en cuenta que **DEBE SER LO PRIMERO QUE ESTABLECEMOS EN EL FICHERO**

Con `strict_types` una variable no puede pasar un valor distinto a una función o retornar un valor distinto del tipo que se haya definido.

Por otro lado también podemos observar que establecemos el tipo de datos devuelto mediante el símbolo “:” y el tipo de datos antes del cuerpo de la función

Probar el código y ver el resultado en el navegador:

```
<?php
    declare( strict_types=1);
?>
<!DOCTYPE html>
```



```

<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
  </head>
  <body>
    <?php

        function sum( int $a, int $b): int {
            return $a + $b;
        }

        echo "<p> la suma de uno más dos es: ";
        $resultado = sum(1,2);
        print sum(1,2);
        echo "</p>"

    ?>

  </body>
</html>

```

● **Práctica 3:** Realizar el código anterior y tomar captura de pantalla del resultado. ¿qué es lo que ha ocurrido ?. Poner código html antes de la declaración de strict\_types y probar de nuevo ¿ qué ocurre ahora ?

Vamos a probar las implicaciones de declare strict\_types

( para más información leer: <http://php.net/manual/es/control-structures.declare.php> respecto a strict\_types )

En la documentación nos dice que strict\_types observa el valor que se le pasa a una función y el valor que esta devuelve, lanzando un error si no se corresponde con lo esperado

Veamos este código:

```

<?php
    declare( strict_types=1);
?>
<!DOCTYPE html>
<html>

```

```

<head>
    <meta charset="UTF-8">
    <title></title>
</head>
<body>
<?php
    function fun( int $a, int $b): int {
        $a = "0";
        //return $a;
        return $b ;
    }
    print fun(1,2);
    //print fun("e",3);
    echo "</p>"
?>
</body>
</html>

```

● **Práctica 4:** Tomar captura de pantalla de: ( y explicar lo ocurrido )

- probar el código anterior ¿ da error ? Por qué ?
- quitar el comentario a: return \$a; ¿ da error ahora ?por qué ?
- quitar comentario a: print fun("e",3); ¿ da error ?

Así pues, debemos tener claro que `strict_types=1` nos cubre respecto a la recepción de un argumento y el valor devuelto por una función pero no nos cubre por cambiar el tipo de una variable que hemos recibido como argumento ( salvo que lo devolvamos )

## Variables asignadas por referencia

En PHP también podemos asignar variables por referencia. En ese caso no se les asigna un valor, sino otra variable, de tal modo que las dos variables comparten espacio en memoria para el mismo dato.

La notación para asignar por referencia es colocar un "&" antes del nombre de la variable.

```
<?php
    $foo = 'Bob'; // Asigna el valor 'Bob' a $foo
    $bar = &$foo; // Referencia $foo vía $bar.
    $bar = "Mi nombre es $bar"; // Modifica $bar...
    echo $foo; // $foo también se modifica.
    echo $bar;
?>
```

Esto dará como resultado la visualización dos veces del string "Mi nombre es Bob". Algo como: Mi nombre es Bob Mi nombre es Bob

● **Práctica 5:** Probar el código anterior. Probar ahora con números ¿también funcionan las referencias ?

● **Práctica \_\_:** `var_dump($mivar)` nos muestra el contenido de la variable \$mivar. Crear un array: `$mivar = [];` Introducir datos: `array_push($mivar,"uno");` y hacer una asignación a otras variables. Una por referencia y la otra por valor:  
`$arr1 = $mivar;`  
`$arr2 = &$mivar;`  
modificar la posición cero de esas variable : `$arr1[0] = "una variación"; $arr2[0] = "variando array2";`  
y mostrar el contenido de `$mivar[0]` y `$arr1[0]`  
¿ qué es lo que ha ocurrido ? ( tomar captura de pantalla y explicarlo )

## Variables predefinidas

PHP proporciona una gran cantidad de variables predefinidas para todos los scripts. Las variables representan de todo, desde variables externas hasta variables de entorno incorporadas, desde los últimos mensajes de error hasta los últimos encabezados recuperados.

Algunos ejemplos:


**\$\_REQUEST** matriz asociativa que contiene los datos enviados por los formularios y las cookies

**\$\_SERVER** matriz asociativa con información sobre cabeceras, rutas, etc suministrada por el servidores

ej.

URL	<b>\$_SERVER[PHP_SELF]</b>
http://www.example.com/ejemplo.php	/ejemplo.php
http://www.example.com/ejercicios/ejemplo.php	/ejercicios/ejemplo.php

**\$GLOBALS** matriz asociativa con todas las variables disponibles en el ámbito global

 **Práctica 6:** Hacer un script php que haga echo de **\$\_SERVER** y de **\$\_SERVER[PHP\_SELF]** tomar captura de pantalla de los resultados

Hemos hablado de variables globales, vamos a entender pues el ámbito de una variable

## Ámbito de una variable

En PHP, todas las variables creadas en la página, fuera de funciones, son variables globales a la página. Por su parte, las variables creadas dentro de una función son variables locales a esa función.

Las variables globales se pueden acceder en cualquier lugar de la página, mientras que las variables locales sólo tienen validez dentro de la función donde han sido creadas. De modo que una variable global la podemos acceder dentro de cualquier parte del código, mientras que si intentamos acceder a una variable local fuera de la función donde fue creada, nos encontraremos con que esa variable no tiene contenido alguno.

Ahora bien, si intentamos acceder a una variable global dentro de una función, en principio también nos encontraremos con que no se tiene acceso a su valor. Esto es así en PHP por motivos de claridad del código, para evitar que se pueda prestar a confusión el hecho de usar dentro de una función una variable que no ha sido declarada por ningún sitio cercano.

Para acceder a una variable global dentro de una función, especificaremos que variables globales queremos usar mediante la palabra reservada: `global`

```
function mifuncion(){
    global $mivariable, $otravariable;
    //con esa línea dentro de la función, declaramos el uso de variables globales
    echo $mivariable;
    echo $otravariable;
}
```

Otra alternativa es hacer uso de `$GLOBALS`

```
function mifuncion(){
    //estoy dentro de la función, para acceder a las variables utilizo $GLOBALS
    echo $GLOBALS["mivariable"];
    echo $GLOBALS["otravariable"];
}
```

Podemos observar que podemos acceder a `$GLOBALS` directamente dentro de una función ¿y cómo puede ser eso si es a su vez, una variable global ? Para las otras variables globales hemos visto que tenemos que establecer que son globales. Aquí es donde entra el concepto de: **variables superglobales**. Estas variables mantienen información del sistema, se definen automáticamente en un ámbito global y a las que se puede acceder desde cualquier punto del código PHP.

## Eliminar variables

La función: `unset()` nos permite destruir variables

Documentación de php.net:

**`unset()`** destruye las variables especificadas.

El comportamiento de **`unset()`** dentro de una función puede variar dependiendo de qué tipo de variable que se está tratando de destruir.

Si una variable global es **`unset()`** dentro de una función, solo la variable local es destruida. La variable en el entorno de la llamada mantendrá el mismo valor anterior a la llamada a **`unset()`**.

```
<?php
function destruir_foo()
{
    global $foo;
    unset($foo);
}

$foo = 'bar';
destruir_foo();
echo $foo;
?>
```

El resultado del ejemplo sería:

bar

Si desea aplicar **unset()** a una variable global dentro de una función, puede usar la matriz [\\$GLOBALS](#) para hacerlo:

```
<?php
function foo()
{
    unset($GLOBALS['bar']);
}

$bar = "algo";
foo();
?>
```

Hemos de entender que ejecutar unset() es un paso más que igualar a null una variable:

```
$variable = null;
unset($variable);
```

Cuando establecemos unset() borramos de la tabla de símbolos de php la variables. Así si en las sentencias de antes preguntamos por var\_dump():

```
$variable = null;
var_dump($variable);
unset($variable);
var_dump($variable);
```

obtendremos en el segundo caso un mensaje diciendo que la variable no está definida

● **Práctica 7:** Visualizar lo anterior ¿ se encuentran diferencias entre null y unset() ? Tomar captura de pantalla

## Casting de variables

Debido a ser un lenguaje de tipado dinámico no se precisa hacer cast a una variable Sin embargo si se quiere se puede forzar mediante el uso de paréntesis y poniendo en su interior el tipo de dato deseado:

```
$c = $a + (int) $b;
```

En el ejemplo anterior se está forzando a \$b para que sea tratada como un entero

Veamos un ejemplo completo:

```
<?php
    $unavar = 1.3;
    var_dump($unavar);
    echo "<br>";
    $unavar = (int) $unavar;
    var_dump($unavar);
?>
```

● **Práctica 8:** Ejecutar el script anterior ¿ hay alguna diferencia antes y después del cast ? Tomar captura de pantalla

aprovechamos para observar el uso de var\_dump Veamos lo que dice php.net:

```
void var_dump ( mixed $expression [, mixed $... ] )
```

Esta función muestra información estructurada sobre una o más expresiones incluyendo su tipo y valor. Las matrices y los objetos son explorados recursivamente con valores sangrados para mostrar su estructura.

En PHP 5 todas las propiedades públicas, privadas y protegidas de los objetos serán devueltas en la salida.

## Constantes

Para crear una constante haremos uso de la función: `define()` Estas etiquetas que generaremos no harán uso de: `$` para obtener el dato que tienen dentro:

```
define('PI', 3.14159);  
echo 'El valor de PI es: ' . PI ;
```

Las reglas para el nombre de las constantes sigue la misma expresión regular que para las variables: `[a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*`

A partir de php 5 ya se puede hacer uso de la palabra reservada `const`:

```
const PI = 3.14159;
```

- **Práctica \_:** ¿ qué ámbito tienen las constantes ? ¿ realmente no se puede poner varios valores en un constante ?  
Probar fuera de una función a crear constante:  
`const PULGADA = 2.53;`  
ahora tratar de establecerla de nuevo mediante:  
`PULGADA = 7;`  
`const PULGADA = 8;`  
`$PULGADA = 9;`  
hacer echo en cada caso.  
Crear la constante en ámbito global ( fuera de función ) ¿ se puede acceder dentro de una función ? ¿ se puede establecer: `const PULGADA = 10` dentro de una función ?  
Tomar capturas de pantalla en cada caso y explicar lo que ha ocurrido

## Operadores

No difieren en general de los que pudiéramos encontrar en Java o C salvo:

**\*\* Exponenciación:** `2 ** 3` es elevar 2 a la tercera potencia: `23`

**. Concatenación** En php podemos concatenar textos mediante el punto: `“.”`

```
<?php  
$apellidos="Arcadia Manices";  
$nombre="Amilcar";  
echo "Hola mi nombre es:" . $nombre . " y mis apellidos son:" . $apellidos;  
?>
```



**.= Concatenación y asignación** En php podemos concatenar textos mediante el punto: "." ahora bien, entonces existe el equivalente a +=, -=, etc para: .= significando:

```
$cadena = "Hola ";  
$cadena .= "mundo";  
echo $cadena // devolverá: hola mundo
```

**● Práctica 9:** Crear un script que muestre las potencias del número 2 desde 2<sup>1</sup> hasta 2<sup>9</sup> hacer uso del operador: \*\*  
Ir concatenando las salidas en pantalla de las potencias en una string mediante el operador de concatenación y asignación: .=

Al igual que en Javascript hay diferencia entre la comparación: == y la **comparación ===** Siendo el segundo que además de ser iguales, también sean exactamente del mismo tipo

```
$a = 2;  
$b = "2";  
$a == $b // devuelve true  
$a === $b // devuelve false
```

**<=> Comparador de orden.** (PHP 7, Spaceship operator)

Este operador sirve para comparar dos variables (\$a y \$b) y devolverá -1 si \$a es < \$b, 0 cuando sean iguales y 1 cuando \$a sea mayor que \$b.

```
echo 3 <=> 4; // -1  
echo 3.4 <=> 3.4; // 0  
echo 4 <=> 3; // 1
```

**?? uno o el otro** (PHP 7, operador coalescente)

El operador de fusión de null (??) se ha añadido como aliciente sintáctico para el caso común de la necesidad de utilizar un operador ternario junto con **isset()**. **Devuelve su primer operando si existe y no es NULL**; de lo contrario devuelve su segundo operando.

```

<?php
// Obtener el valor de $_GET['usuario'] y devolver 'nadie'
// si no existe.
$nombre_usuario = $_GET['usuario'] ?? 'nadie';
// Esto equivale a:
$nombre_usuario = isset($_GET['usuario']) ? $_GET['usuario'] : 'nadie';

// La fusión se puede encadenar: esto devolverá el primer
// valor definido de $_GET['usuario'], $_POST['usuario'],
// y 'nadie'.
$nombre_usuario = $_GET['usuario'] ?? $_POST['usuario'] ?? 'nadie';
?>

```

● **Práctica 10:** Crear un programa en php que obtenga la descomposición de un número que esté almacenado en la variable: \$numero Por ejemplo: \$numero = 3102 Se pretende que se utilicen en el programa los operadores: . = , \*\*  
Para el ejemplo anterior se debe mostrar en pantalla: 2 \* 1 + 0 \* 10 + 1 \* 100 + 3 \* 1000

Antes hemos nombrado isset(). Hay otra función relacionada pero diferente de isset() llamada empty()

Ejecutemos el siguiente código:

```

<?php
$var = "";

if(empty($var)){ // true because "" is considered empty

    echo '<br>empty($var) para $var="" ' ;

}else{
    echo '<br>!empty($var) para $var="" ' ;
}

if(isset($var)){ //true because var is set
    echo '<br>isset($var) para $var="" ' ;
}else{
    echo '<br> !isset($var) para $var="" ' ;
}

```

```

if(empty($otherVar)){ //true because $otherVar is null
    echo '<br>empty($otherVar) para $otherVar que no se ha establecido ';
} else {
    echo '<br> !empty($otherVar) para $otherVar que no se ha establecido ';
}

if(isset($otherVar)){ //false because $otherVar is not set
    echo '<br>isset($otherVar) para $otherVar que no se ha establecido ';
} else {
    echo '<br> !isset($otherVar) para $otherVar que no se ha establecido ';
}
?>

```

Observamos que `isset()` es equivalente a `!empty()` salvo en el caso de una variable que es comillas vacías. En ese caso `isset()` devolverá `false` mientras que `!empty()` devolverá `true`

 **Práctica 11:** Ejecutar el script y tomar captura de pantalla

## Cadenas en PHP

Uno de los tipos de datos más corrientes en la mayoría de los lenguajes son los strings. Para asignar a una variable un contenido de tipo cadena, lo escribiremos entre comillas, valiendo tanto las comillas dobles como las comillas simples.

### Comillas dobles

Si usamos comillas dobles para delimitar cadenas de PHP haremos que el lenguaje se comporte de una manera más "inteligente". Lo más destacado es que las variables que coloquemos dentro de las cadenas se sustituirán por los valores. Ej:

```

<?php
    $nombre = "Riquelme";
    echo "hola amigo $nombre";
?>

```

Observando la salida vemos que con comillas dobles nos ha interpretado el contenido de la variable en el mensaje en pantalla

## Uso de llaves

Tratar de interpretar dentro de comillas dobles un array asociativo ( índice alfanumérico al estilo de mapa ) puede generar problemas que se resuelven mediante llaves.

Ej.

```
$array = array('uno' => 1, 'dos' => 2, 'tres' => 40, 'cuatro' => 55);  
$cadena = "La posición 'tres' contiene el dato $array['tres']"; //esto
```

● **Práctica 12:** Probar el script anterior y observar que ocurre. ¿ qué mensaje de error se observa ?

El problema anterior se resuelve usando llaves:

```
<?php  
$array = array('uno' => 1, 'dos' => 2, 'tres' => 40, 'cuatro' => 55);  
$cadena = "La posición 'tres' contiene el dato {$array['tres']}";  
echo $cadena;  
?>
```

## Comillas simples

Al usar comillas simples ya no se trata de obtener el contenido. Ahora volcará el texto literal que hemos escrito. Ej.

```
<?php  
$nombre = "Riquelme";  
echo 'hola amigo $nombre';  
?>
```

Observar que si estamos introduciendo un texto que no queremos que interprete los contenidos de variables nos interesa más hacer uso de las comillas simples ya que el intérprete no tendrá gasto computacional tratando de sustituir variables por su valor

## Caracteres de escape

Para incluir el símbolo \$, la contrabarra y otros caracteres utilizados por el lenguaje dentro de las cadenas y no confundirlos se usan los caracteres de escape.

Para insertar un caracter de escape tenemos que indicarlo comenzando con el símbolo de la contrabarra (barra invertida) y luego el del caracter de escape que deseemos usar.

Los caracteres de escape disponibles dependen del tipo de literal de cadena que estemos usando. En el **caso de las cadenas con comillas dobles** se permiten muchos más caracteres de escape. Los encuentras en la siguiente tabla:

<code>\n</code>	Salto de línea
<code>\r</code>	Retorno de carro
<code>\t</code>	Tabulador horizontal
<code>\v</code>	Tabulador vertical (incluido en PHP 5.2.5)
<code>\e</code>	Tecla de escape (incluido en PHP 5.4.4)
<code>\f</code>	Tecla de avance de página (incluido en PHP 5.2.5)
<code>\\</code>	Contrabarra o barra invertida
<code>\\$</code>	Símbolo de dólar
<code>\"</code>	Comillas dobles

Estos cambios de línea y tabulaciones tienen únicamente efecto en el código y no en el texto ejecutado por el navegador. En otras palabras, si queremos que nuestro texto ejecutado cambie de línea hemos de introducir un echo "`<br>`" y no "`\n`"

## Variables de variables

Observar lo siguiente:

```
$variable='dato';
```

```
$dato = 5;
```

Ahora mismo ya no debíamos tener dudas que lo anterior hace referencia al nombre de dos variables. Una variable llamada: `$variable` que almacena la palabra: `dato`

y otra variable llamada: `$dato` que almacena el número 5

Ahora bien, ¿ cómo interpretar la siguiente expresión en el contexto anterior?

```
${$variable}
```

Pues bien, primero se transforma la parte interior: `$variable` y se obtiene la palabra `dato` Así que nos encontramos con: `$dato` y esa variable sabemos que tiene el valor 5

**Nota: no es necesario poner las llaves en el ejemplo anterior**

Ejecutar lo siguiente y comprobarlo:

```
<?php
    $variable = 'dato';
    $dato = 5;
    echo $$variable.'<br>';
?>
```

● **Práctica 13:** Probar el script anterior y observar que ocurre. Probar ahora con llaves: `${$variable}` ¿ hay diferencia ?

Mirar el siguiente código:

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

    }

    echo "<br> $dato3 ";
    echo "<br> $dato8 ";
?>
```

● **Práctica 14:** Toma el código anterior e introduce una expresión “variable de variables” que permita definir las variables: `$dato0`, `$dato1`, ..., `$dato9` Cada una de ellas con el valor correspondiente: 0, 1,...,9

## Arrays en PHP

Un array en PHP es en realidad un mapa ordenado. Un mapa es un tipo de datos que asocia valores con claves. Este tipo se optimiza para varios usos diferentes; se puede emplear como un array, lista (vector), tabla asociativa (tabla hash - una implementación de un mapa), diccionario, colección, pila, cola, y posiblemente más. Ya que los valores de un array pueden ser otros arrays, también son posibles árboles y arrays multidimensionales.

**Nota:** Los arrays se copian por valor no por referencia. Para tomar referencia usamos: `&$miarray`

### Arrays comunes, índices numéricos

```
$array = array("foo", "bar", "hello", "world");
```

En el ejemplo anterior podemos observar que nos estamos apoyando en una función llamada: `array()` y que le estamos dando directamente los valores. Así en la posición 0 estará “foo” en la 1 estará “bar”, etc

Podemos adulterar ese comportamiento introduciendo en el momento que queramos un índice. Así pues vemos el siguiente ejemplo:

```
$array = array(
    "a",
    "b",
    6 => "c",
    "d",
);
```

## Arrays asociativos

Los pares clave-valor los establecemos así: clave => valor

y separamos por comas.

```
<?php
$array = array(
    "foo" => "bar",
    "bar"=>"foo",
);
```

//PHP 5.4

```
$array = [
    "foo"=>"bar",
    "bar" => "foo",
];
?>
```

Observemos el siguiente trozo de código:

```
<?php
$array = [];
$array[2]="mensaje";
$array[7]="lalala!";
$array[]="yepa yepa!!";
var_dump($array);
?>
```

- **Práctica 15:** ejecutar el script anterior ¿ se muestran las posiciones anteriores a la 2 ? ¿ y entre la 2 y la 7 ?. Realizar el mismo script pero en lugar de crear el array mediante los corchetes: \$array = [] hacerlo con la función array() ¿ hay diferencias en la salida en pantalla ? Ejecutar var\_dump(\$array) después de cada asignación de un valor al array. Tomar captura de pantalla de los resultados

Vemos que inicialmente establecemos un array vacío. Luego incorporamos en las posiciones 2 y 7 texto y al no establecer ningún índice en la línea siguiente simplemente se agrega a continuación del valor de índice mayor, al estilo de un push()



- **Práctica \_\_:** crear un array asociativo dejando sin poner en algunas ocasiones la parte de la clave dejando únicamente el valor ( al estilo de si fuera un array no asociativo ) hacer un `var_dump()` y recorrerlo con un `for` ( no con un `foreach` ) ¿ muestra algún valor ? ¿ genera error ?

## Recorrer array

Si se quiere hacer uso de un contador y acceder por índice numérico a un array con todos sus valores definidos ( no se ha saltado ninguna key ) :

```
<?php
$array = array('perro', 'gato', 'avestruz');
$array_num = count($array);
for ($i = 0; $i < $array_num; $i++){
    print $array[$i];
}

?>
```

Por medio de `foreach`:

```
<?php
$array = array('perro', 'gato', 'avestruz');
foreach ($array as $val) {
    print $val;
}

?>
```

Con lo anterior únicamente estamos tomando el valor de cada elemento. Si quisiéramos disponer del par clave => valor:

```
<?php
$array = array('perro', 'gato', 'avestruz');
foreach ($array as $key => $val) {
    print "<br>array[ $key ] = $val";
}

?>
```

- **Práctica 16:** Ejecutar el script anterior. ¿ Tenemos que usar los nombres de variables `$key` y `$val` ? Sustituir por otros nombres de variables y ver si hay algún problema

Ahora bien, ¿ qué respecto a borrar o modificar elementos del array y recorrerlos con un foreach ? Para entender mejor empezemos con un ejemplo:

```
<?php
$array = [];
for($i=0;$i<5;$i++){
    $array[] = "a" . $i;
}

$j=count($array);
foreach( $array as $key => $val){
    $j--;
    unset($array[$j]);
    echo "<br>";
    var_dump($array);
    echo "<br> $key => $val ";
    echo "<br>";
}
?>
```

● **Práctica 17:** Ejecutar el script anterior. En Java eliminar elementos de un array en un foreach implica un error ¿ también en php ? Tomar captura de pantalla del resultado

Observando el resultado de la ejecución podemos ver que se va eliminando del último al primer elemento del array siendo como resultado un array vacío. Sin embargo nos sigue mostrando en cada iteración la clave y valor, como si no hubieran sido borrados

¿ Cómo es eso ? Lo que está ocurriendo es que **foreach hace una copia del array original para hacer las iteraciones**. Así que observamos los datos de esa copia que no ha sido borrada

En parte pudiera no estar del todo mal ese comportamiento, ya que no tendrá lugar una excepción por estar iterando sobre un elemento que ya no existe. Por otro lado perdemos la posibilidad de tener la información actualizada durante la iteración del array. Así pues en lugar de tomar: \$val para ese caso, podríamos, por ejemplo, acceder a: \$array[\$key]

En el siguiente ejemplo empezamos con un array inicializado cada valor a la letra “a”:

```
$array = ["a","a","a","a","a"];
```

Progresivamente en cada iteración de foreach se va cambiando el último valor a: “a4”

luego el penúltimo valor a: “a3” y así con todos los elementos.

En el ejemplo se muestra el resultado indeseado de: \$val y el que queremos de: \$array[\$key]

```
<?php
$array = ["a", "a", "a", "a", "a"];

$j=count($array);
foreach( $array as $key => $val){
    $j--;
    $array[$j] .= $j;
    echo "<br>";
    var_dump($array);
    echo "<br> $key => $val"; //esta línea no tiene el efecto deseado
    echo "<br> $key => $array[$key]"; // aquí sí
    echo "<br>";
}
?>
```

● **Práctica 18:** Ejecutar el script anterior. Modificar los echo para que se sepa cuando llamamos a \$array ( recordar que con comillas simples no interpreta ) Tomar captura de pantalla

Otra alternativa es hacer uso de las referencias ( un puntero a la posición del array )


```
<?php
$array = ["a", "a", "a", "a", "a"];

$j=count($array);
foreach( $array as $key => &$amp;val){
    $j--;
    $array[$j] .= $j;
    echo "<br>";
    var_dump($array);
    echo "<br> $key => $val";
    echo "<br> $key => $array[$key]";
    echo "<br>";
}
?>
```

Vemos que ahora lo hace bien. Sin embargo hay que tener un poco de cuidado con las referencias. Una vez finalizado el foreach la variable \$val sigue definida y sigue apuntando al último elemento del array. Así el siguiente código tienen problemas:

```
<?php
$arr = array(1, 2, 3, 4);
foreach ($arr as &$amp;val) {
    $val = $val * 2;
}

foreach ($arr as $key => $val) {
    echo "{$key} => {$val} <br>";
    print_r($arr);
    echo "<br><br>";
}
?>
```

 **Práctica 19:** Ejecutar el script anterior. Tomar captura de pantalla del resultado

Como estamos encadenando dos foreach y usamos \$val en ambos, ocurre que en el segundo foreach estamos guardando en la posición de memoria del último elemento que se recorrió en el primer foreach ( observar que el primer foreach toma la referencia y el segundo no lo hace )

Es una buena práctica ejecutar unset() respecto a esa referencia y evitar así problemas. El ejemplo anterior quedaría:

```
<?php
$arr = array(1, 2, 3, 4);
foreach ($arr as &$amp;val) {
    $val = $val * 2;
}
unset($val); // $val ya no apunta al último elemento del array
foreach ($arr as $key => $val) {
    echo "{$key} => {$val} <br>";
    print_r($arr);
    echo "<br><br>";
}
?>
```

**Nota:** Se suele considerar óptimo el uso de foreach para recorrer un array y se suele recomendar este método respecto a otras opciones. Pero eso nos lleva en ocasiones a tener que introducir: break o return, ya que foreach únicamente tiene la condición de parada de finalización del array

## Comportamiento LIFO con array

Podemos obtener el comportamiento de una pila en un array mediante las funciones:

`array_pop()` // devuelve el último elemento de un array. Si el array está vacío devuelve null

Ej.

```
<?php
    $arr= ["1", "2", "3", "4"];
    $va = array_pop($arr);
    echo "el array ahora queda: <br>";
    print_r($arr);
    echo "<br>el valor extraido es: " . $va;
?>
```

● **Práctica 20:** Ejecutar el script anterior. ¿ qué valor devuelve ?Tomar captura de pantalla

para agregar a la pila podemos hacer uso de:

`$arr[] = valordeseado;`

o

`array_push($arr, valordeseado);`

Se recomienda hacer uso de la primera expresión cuando es un único valor el que se quiere agregar. Cuando queremos agregar varios con push:

`array_push($arr, "dato1", "dato2");`

la sentencia anterior agrega tanto dato1 como dato2

● **Práctica 21:** Crear un script que por medio de un bucle for que vaya de 1 a 10 agregue esos números en un array En cada iteración mostrar el contenido del array. Después en un bucle

for de 1 a 5 ir ejecutando sentencias array\_pop() y mostrar como queda el array en cada iteración

## Comportamiento FIFO con array

Mediante las funciones: array\_shift(), array\_unshift()

**array\_shift()** Quita un elemento del principio del array y lo devuelve acortando el array un elemento ( quedando en el array los restantes al quitado )

Ej.

```
<?php
$arr= ["1","2","3","4"];
$val = array_shift($arr);
echo "el array ahora queda: <br>";
print_r($arr);
echo "<br>el valor extraido es: " . $val;
?>
```

array\_unshift() Añade elementos al inicio de un array.

```
<?php
$arr= ["1","2","3","4"];
$val = array_unshift($arr, "7", "8");
echo "el array ahora queda: <br>";
print_r($arr);
?>
```

● **Práctica 22:** Crear un script que por medio de un bucle for que vaya de 1 a 10 agregue esos números en un array mediante array\_unshift() En cada iteración mostrar el contenido del array. Después en un bucle for de 1 a 5 ir ejecutando sentencias array\_shift() y mostrar como queda el array en cada iteración

● **Práctica 23:** Haz una página PHP que utilice foreach para mostrar todos los valores del array \$\_SERVER en una tabla con dos columnas. La primera columna debe contener el nombre de la variable, y la segunda su valor

## Otras funciones con array

Hay muchas funciones a analizar. Una descripción en: <http://es.php.net/manual/es/ref.array.php>

Vamos a ver alguna por encima:

### in\_array()

Comprueba si un valor existe en un array. Devuelve boolean

Ej.

```
<?php
$os = array("Mac", "NT", "Irix", "Linux");
if (in_array("Irix", $os)) {
    echo "Existe Irix";
}
if (in_array("mac", $os)) {
    echo "Existe mac";
}
?>
```

Como se puede ver recibe dos parámetros, el primero el valor a buscar, el segundo el array

Hay opción para un tercer parámetro. El tercer parámetro sería un boolean que comprueba también la coincidencia de tipos si está establecido a TRUE.

Ej.

```
<?php
$a = array('1.10', 12.4, 1.13);

if (in_array('12.4', $a, true)) {
    echo "Se encontró '12.4' con comprobación estricta\n";
}

if (in_array(1.13, $a, true)) {
    echo "Se encontró 1.13 con comprobación estricta\n";
}
?>
```

### array\_search()

Busca un valor determinado en un array y devuelve la primera clave correspondiente en caso de éxito o FALSE si no.

Ej.

```

<?php
$array = array(0 => 'azul', 1 => 'rojo', 2 => 'verde', 3 => 'rojo');

$clave = array_search('verde', $array);
echo $clave . "<br>";
$clave = array_search('marrón', $array);
if( $clave === FALSE)
    echo "no se ha localizado el valor";
else
    echo $clave;
?>

```

Observar que preguntamos por el resultado de \$clave comparando con FALSE mediante === Esto es porque pudiera devolver un valor no booleano que se evalúe como FALSE

### array\_values()

Devuelve un array indexado con los valores del array que se le pase como parámetro. Observar que mediante esta función podríamos eliminar los huecos generados en un array por unset()

Ej.

```

<?php
$array = array('azul', 'rojo', 'verde', 'amarillo', 'blanco');

unset($array[2]);
unset($array[3]);

print_r($array);

//ahora usando array_values "limpiamos"
$array = array_values($array);
echo "<br>";
print_r($array);
?>

```

● **Práctica 24:** Ejecutar los códigos de in\_array(), array\_search(), array\_values() tomar captura del resultado de la ejecución

● **Práctica 25:** Rellenar un array con 10 números aleatorios entre 20 y 25 ( hacer uso de: rand ( int \$min , int \$max ) : int ) y luego hacer uso de array\_search() para localizar el valor: “22” Se debe mostrar en pantalla el array completo y el valor devuelto por array\_search()



`array array_keys ( array $array [, mixed $search_value = NULL [, bool $strict = FALSE ]])`

Devuelve todas las claves de un array o un subconjunto de claves

El segundo y tercer parámetro determinarían si lo que queremos es un subarray con las claves para el valor establecido como segundo parámetro. El tercer parámetro establece si la comparación con ese valor debe o no tener en cuenta los tipos de datos

```
<?php
    $a=array(10,20,30,"10");
    print_r(array_keys($a,"10",false));
?>
```

El anterior ejemplo nos mostrará el array: [0,3]

esto es porque al buscar el dato: "10" le hemos especificado que no queremos la verificación de tipos. De haber puesto true únicamente nos habría devuelto la posición 3

`usort()`

usort — Ordena un array según sus valores usando una función de comparación definida por el usuario

Veamos ejemplo:

```
<?php
function cmp($a, $b)
```

```

{
    if ($a == $b) {
        return 0;
    }
    return ($a < $b) ? -1 : 1;
}

$a = array(3, 2, 5, 6, 1);
usort($a, "cmp");

foreach ($a as $valor) {
    echo " $valor, ";
}

?>

```

Podemos observar el comportamiento clásico de las funciones de comparación. Donde basta que sean mayores que cero para que el primer elemento sea considerado mayor, igual a cero para que se consideren iguales, o menor que cero para que el primer elemento se considere menor

● **Práctica 26:** Variar el ejemplo anterior para que se haga uso del operador nave espacial: `<=>`

● **Práctica 27:** Crear un array con los valores: [7,2,8,1,9,4] Hacer búsqueda con `array_search()` de: 4  
 Ordenar el array con `usort` y repetir la búsqueda de: 4  
 Mostrar los array antes y después de ordenación así como lo que devuelve `array_search()`

# Funciones

Empecemos por un ejemplo:

```
<?php
function sumar($a, $b, $print = false): float
{
    $suma = $a + $b;
    if ($print) {
        echo "resultado suma: $suma <br>";
    }
    return $suma;
}

$sum1=sumar(1,2);
$sum2=sumar(4,5,true);

echo "las operaciones para sum1 y sum2 dan: $sum1 , $sum2";
?>
```

Establecer tipo de parámetros primitivos y tipo de datos de devolución primitivo es a partir de php 7 así pues en general, podemos no definir el tipo de parámetro, como se puede observar en el ejemplo. Siendo así **no es posible una sobrecarga de funciones** como podemos ver en otros lenguajes como Java **Si enviamos más parámetros que los que corresponde a una función, esta seguirá funcionando igualmente**

También podemos observar el parámetro: `$print = false` Estamos estableciendo un valor por defecto para el parámetro de tal forma que podemos hacer la llamada a la función con o sin ese parámetro.

● **Práctica 28:** Modifica el código anterior y quita el valor por defecto del parámetro `$print`. Ejecuta el programa y toma captura de pantalla de los mensajes del IDE y responde: ¿ se obtiene resultado o se detiene el programa ?

Las funciones en php hacen por defecto el paso por valor. Así que lo que recibimos es una copia del valor original. Esto no es aplicable a los objetos que siempre se pasan por referencia.

Veamos un ejemplo:

```
<?php
function modify(int $a): void {
    $a = 3;
}
$a = 2;
modify($a);
print_r($a);
?>
```

● **Práctica 29:** Probar el código anterior. Observamos que no se ha modificado el valor de la variable después de la ejecución de la función. Así que ¿estamos en un caso de paso por valor o por referencia? Tomar captura de pantalla

Veamos ahora el paso por referencia:

```
<?php
function modify(int &$a): void {
    $a = 3;
}
$a = 2;
modify($a);
print_r($a);
?>
```

● **Práctica 30:** Ejecutar el ejemplo y observar que ahora la variable sí se ve modificada. Tomar captura de pantalla

## ¿ arrays por copia o referencia ?

Cuando se viene de Java o C estamos acostumbrados a que el paso de un array sea por referencia, sin embargo en PHP no es así. Probar el siguiente código:

```
<?php
function modify(array $arr): void {
    $arr[] = 4;
}
$a = [1];
```

```
    modify($a);  
    print_r($a);  
    ?>
```

Observamos que sigue imprimiendo 1 como único dato del array. Modificando la definición de la función así:

```
function modify(array &$arr): void {
```

● **Práctica 31:** Hacer lo anterior, comprobar el resultado. Ahora debiera mostrar todos los datos del array. Tomar captura de pantalla.

## Inclusión de ficheros externos

Conforme vayan creciendo los programas que hagas, verás que resulta trabajoso encontrar la información que buscas dentro del código. En ocasiones resulta útil agrupar ciertos grupos de funciones o bloques de código, y ponerlos en un fichero aparte. Posteriormente, puedes hacer referencia a esos ficheros para que PHP incluya su contenido como parte del programa actual.

Para incorporar a tu programa contenido de un archivo externo, tienes varias posibilidades:

**include:** Evalúa el contenido del fichero que se indica y lo incluye como parte del fichero actual, en el mismo punto en que se realiza la llamada. La ubicación del fichero puede especificarse utilizando una ruta absoluta, pero lo más usual es con una ruta relativa. En este caso, se toma como base la ruta que se especifica en la directiva `include_path` del fichero `php.ini`. Si no se encuentra en esa ubicación, se buscará también en el directorio del guión actual, y en el directorio de ejecución.

**include\_once:** Si por equivocación incluyes más de una vez un mismo fichero, lo normal es que obtengas algún tipo de error (por ejemplo, al repetir una definición de una función). `include_once` funciona exactamente igual que `include`, pero solo incluye aquellos ficheros que aún no se hayan incluido.

**require:** Si el fichero que queremos incluir no se encuentra, `include` da un aviso y continua la ejecución del guión. La diferencia más importante al usar `require` es que en ese caso, cuando no se puede incluir el fichero, se detiene la ejecución del guión.

**require\_once**. Es la combinación de las dos anteriores. Asegura la inclusión del fichero indicado solo una vez, y genera un error si no se puede llevar a cabo.

Ejemplo de include:

```
vars.php:
<?php

    $color = 'verde';
    $fruta = 'manzana';

?>

test.php:
<?php

    echo "Una $fruta $color"; // Una
    include 'vars.php';
    echo "Una $fruta $color"; // Una manzana verde

?>
```

● **Práctica 32:** Hacer lo anterior, pero usando require en lugar de include. Para que se note la diferencia la llamada de require debiera ser a un nombre de fichero incorrecto. Por ejemplo haremos que llame a: vars1.php cuando como sabemos el fichero es vars.php  
Tomar captura de pantalla

## Formularios

## Pasar variables por URL con PHP

Bucles y condiciones son muy útiles para procesar los datos dentro de un mismo script. Sin embargo, en un sitio Internet, las páginas vistas y los scripts utilizados son numerosos. Muy a menudo necesitamos que nuestros distintos scripts estén conectados unos con otros y que se sirvan de variables comunes. Por otro lado, el usuario interacciona por medio de formularios cuyos campos han de ser procesados para poder dar una respuesta. Todo este tipo de factores dinámicos han de ser eficazmente regulados por un lenguaje como PHP

Para pasar las variables de una página a otra lo podemos hacer introduciendo dicha variable dentro del enlace hipertexto de la página destino. La sintaxis sería la siguiente:

```
<a href="destino.php?variable1=valor1&variable2=valor2&...">Mi enlace</a>
```

Vemos que después del nombre de la página agregamos un interrogante el nombre de la variable y su valor mediante el símbolo igual. Para encadenar con el siguiente utilizamos: &

Veamos un ejemplo rápido. Vamos a crear una página llamada index.php que contendrá:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
  </head>
  <body>
    <a href="index.php?saludo=hola&texto=Esto es una variable texto">Paso
variables saludo y texto a la página destino.php</a>
    <br>
    <?php

    $saludo = $_GET["saludo"];
    $texto = $_GET["texto"];
    echo "Variable saludo: $saludo <br>";
    echo "Variable texto: $texto <br>";

  ?>
</body>
</html>
```

Hemos hecho uso de GET . Sabemos que podemos hacer uso de otros métodos. Pero antes de continuar analicemos mejor de lo que estamos hablando

Observar que hemos hecho uso de la variable superglobal: `$_GET`

Debemos tener cuidado con el paso de algunos símbolos cuando pasamos una variable por url, ya que pueden no ser recibidos correctamente. Para evitar eso podemos hacer uso de la función: **`urlencode()`**

En el manual de php nos dice sobre `urlencode()`:

### **`urlencode`**

Codifica como URL una cadena. Esta función es conveniente como método práctico para pasar variables a la siguiente página

Devuelve una cadena en la que todos los caracteres no-alfanuméricos excepto `_-.` han sido reemplazados con un signo de porcentaje (%) seguido por dos dígitos hexadecimales y los espacios son codificados como signos de suma (+). Esta es la misma codificación usada en los datos publicados desde un formulario WWW, es decir, el mismo mecanismo usado para el tipo de medios `application/x-www-form-urlencoded`.

Un detalle interesante es que algo que le aplicamos `urlencode()` y luego recogermos con `$_GET` luego no tenemos que pasarle un `decode`. Y por la forma de trabajo de `$_POST` (sabemos que por defecto los formularios se envían como `application/x-www-form-urlencoded`) tampoco es necesario `decode`. Específicamente en el manual de php ponen una advertencia:

**ADVERTENCIA:** Las superglobales `$_GET` y `$_REQUEST` ya están decodificadas. El uso de `urldecode()` en un elemento en `$_GET` o `$_REQUEST` puede tener resultados inesperados y peligrosos

En otro caso una vez se ha codificado con `urlencode()` tenemos el reverso: **`urldecode()`**

```
<?php
$query = "my=apples&are=green+and+red";

foreach (explode('&', $query) as $chunk) {
    $param = explode("=", $chunk);

    if ($param) {
        printf("Value for parameter \"%s\" is \"%s\"<br/>\n", urldecode($param[0]), urldecode($param[1]));
    }
}
?>
```

Vamos a hacer una prueba:

```
<?php
```



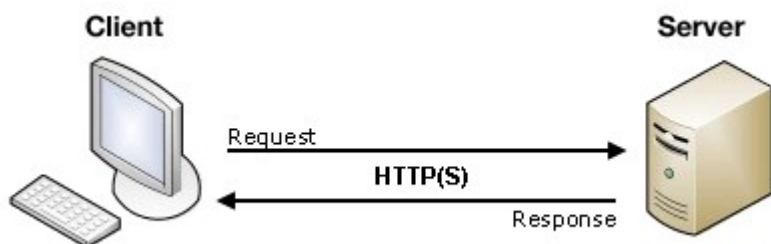
```
echo "<a href=index.php?prueba='Pasando datos diría.. que hay que usar urlencode'>pasando datos</a>";  
$conUrlEncode = urlencode('Pasando datos diría.. que hay que usar urlencode');  
  
$recibido = $_GET["prueba"] ?? "nadita";  
  
echo "<h3>se ha recibido:</h3>";  
echo "prueba: ". $recibido . "<br>";  
?>
```

● **Práctica 33:** Hacer lo anterior, pero se debe comprobar la diferencia de pasar el texto con urlencode y sin urlencode. Así que se propone poner dos parámetros: prueba y prueba2 uno de ellos con urlencode y el otro sin él pasando en ambos casos el mismo texto en el value. Tomar captura de pantalla de lo obtenido

● **Práctica 34:** recorrer el array \$\_GET con un foreach y mostrar el conjunto de clave valor para la actividad anterior

## Método GET, método POST

La web se basa en una arquitectura cliente / servidor muy básica que se puede resumir de la siguiente manera: un cliente (normalmente un navegador Web) envía una petición a un servidor (la mayoría de las veces un servidor web como [Apache](#) , [Nginx](#) , [IIS](#) , [Tomcat](#) , etc.), utilizando el [protocolo HTTP](#) . El servidor responde a la solicitud utilizando el mismo protocolo.



En el lado del cliente, un formulario HTML no es más que una manera fácil de usar conveniente para configurar una petición HTTP para enviar datos a un servidor. Esto permite al usuario proporcionar información a ser entregada en la petición HTTP.

### El método GET

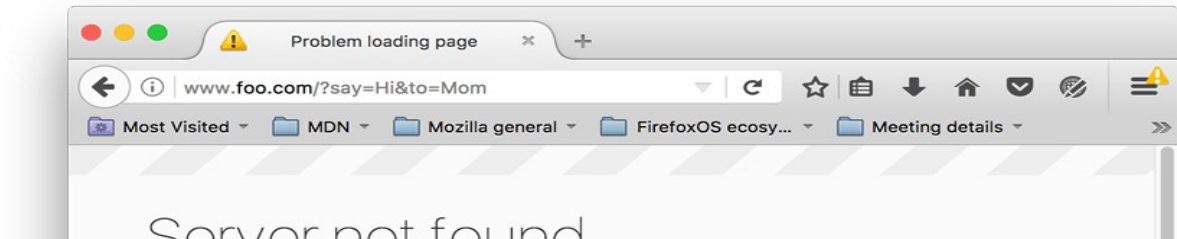
GET es el método utilizado por el navegador que pedir el servidor para enviar de vuelta un recurso dado: "Hey servidor, quiero conseguir este recurso." En este caso, el navegador **envía un cuerpo vacío**. Debido a que el cuerpo está vacío, si un formulario se envía utilizando este método, **los datos enviados al servidor se anexan a la URL**.

Veamos un formulario y como enviaría los datos en la url:

```
<Form action = "http://foo.com" method = "get">
  <Div>
    <Label for = "dice"> Lo saludo qué quiere decir? </ Label>
    <Input name = "decir" id = "decir" value = "Hola">
  </ Div>
  <Div>
    <Label for = "para"> ¿Quién usted quiere decir que a? </ Label>
    <Input name = "a" value = "mamá">
  </ Div>
</ Div>
```

```
<> Botón enviar mis saludos </ botón>
</ Div>
</ Form>
```

Al haber usado GET veremos en la url la info que hemos enviado:



Los datos se añade a la URL como una serie de pares de nombre / valor. Después de la dirección web URL ha terminado, se incluye **un signo de interrogación ( ? )** seguido de los pares de nombre / valor, **cada uno separado por un signo ( & )**. En este caso estamos pasando dos piezas de datos en el servidor:

say, Que tiene un valor de Hi

to, Que tiene un valor de Mom

La solicitud HTTP ( lo que le enviamos al servidor ) se ve así:

```
GET /? = Decir Hola & a = mamá HTTP / 1.1
```

```
Anfitrión: foo.com
```

## El método POST

En esta ocasión le estamos diciendo al servidor: **dame una respuesta que tenga en cuenta los datos que te estoy enviando en el cuerpo de la petición HTTP**. Si un formulario se envía utilizando este método, **los datos se anexan al cuerpo de la petición HTTP**

El mismo formulario de antes ahora utilizando POST daría lugar a la siguiente solicitud HTTP:

```
POST / HTTP/1.1
Anfitrión: foo.com
Content-Type: application / x-www-form-urlencoded
Content-Length: 13

decir = Hi & a = mamá
```

Observar que ahora los datos van en el cuerpo del mensaje HTTP

Content-Length es una cabecera que indica el tamaño del cuerpo

Content-Type Es una cabecera que indica el tipo de recurso que se está enviando al servidor

## Enviar un formulario

En html para los formularios hacemos uso de la etiqueta form

```
<form action=<?php echo $_SERVER['SCRIPT_NAME'] ?> method="get">
  First name: <input type="text" name="fname"><br>
  Last name: <input type="text" name="lname"><br>
  <input type="submit" value="Submit">
</form>

<?php
  $fname = $_GET["fname"] ?? "vacío";
  $lname = $_GET["lname"] ?? "vacío";
  echo "Variable fname: $fname <br>";
  echo "Variable lname: $lname <br>";

?>
```

Veamos el script anterior. Enviamos las variables fname y lname ¿ a qué página ? Mediante la variable `$_SERVER` tenemos acceso a: `SCRIPT_NAME` que nos da el nombre del actual script. De tal forma que estamos enviando el formulario a la propia página en la que estamos

También observamos que hemos usado el método GET así que se está enviando en la cabecera http la información de las variables. Recordar que debemos usar habitualmente POST ya que semánticamente estamos enviando información al servidor, no tanto la solicitud de un recurso. Por otro lado **imaginemos que la URL sea cacheada y por tanto sea accesible por cualquier persona (imaginar que google observa la url y la sigue mostrando entre los resultados del buscador)**

Podríamos haber hecho lo anterior con post como hemos recomendado. Veamos:

```
<form action=<?php echo $_SERVER['SCRIPT_NAME'] ?> method="post">
  First name: <input type="text" name="fname"><br>
  Last name: <input type="text" name="lname"><br>
  <input type="submit" value="Submit">
</form>

<?php
  $fname = $_POST["fname"] ?? "vacío";
  $lname = $_POST["lname"] ?? "vacío";
  echo "Variable fname: $fname <br>";
  echo "Variable lnam: $lname <br>";

?>
```

Hay otra variable superglobal que nos serviría para recoger el formulario indistintamente de que fuera get o post: `$_REQUEST` El ejemplo anterior:

```
<form action=<?php echo $_SERVER['SCRIPT_NAME'] ?> method="post">
  First name: <input type="text" name="fname"><br>
  Last name: <input type="text" name="lname"><br>
  <input type="submit" value="Submit">
</form>

<?php
  $fname = $_REQUEST["fname"] ?? "vacío";
  $lname = $_REQUEST["lname"] ?? "vacío";
  echo "Variable fname: $fname <br>";
  echo "Variable lnam: $lname <br>";

?>
```

En `$_REQUEST` se almacenan los valores de `$_POST`, `$_GET`, `$_COOKIES`

**Observar que si usamos `$_REQUEST` podremos acceder al valor de la variable recibida indistintamente de que se cambie el método get a post o vicerversa.**

Ahora bien, hay que tener en cuenta que son arrays distintos. Si modificamos `$_GET` por ejemplo escribimos:

```
$_GET['mivar']= 'contenido';
```

eso no agregará la variable `mivar` a `$_REQUEST`

ejemplo.

```
<form action=<?php echo $_SERVER['SCRIPT_NAME'] ?> method="post">
  First name: <input type="text" name="fname"><br>
  Last name: <input type="text" name="lname"><br>
  <input type="submit" value="Submit">
</form>

<?php
  $_POST['var1']="un texto prueba";
  echo '$_POST: ';
  print_r($_POST);
  echo '<br>$_REQUEST: ';
  print_r($_REQUEST);
?>
```

Por otro lado un inconveniente que tiene hacer uso de `$_REQUEST` es que no sabes el método de procedencia de la variable ( si fue post o get )

Siempre que sea posible, es preferible validar los datos que se introducen en el navegador antes de enviarlos. Para ello deberás usar código en lenguaje Javascript.

Si por algún motivo hay datos que se tengan que validar en el servidor, por ejemplo, porque necesites comprobar que los datos de un usuario no existan ya en la base de datos antes de introducirlos, será necesario hacerlo con código PHP en la página que figura en el atributo `action` del formulario.

En este caso, una posibilidad que deberás tener en cuenta es usar la misma página que muestra el formulario como destino de los datos. Si tras comprobar los datos, éstos son correctos, se reenvía a otra página. Si son incorrectos, se rellenan los datos correctos en el formulario y se indican cuáles son incorrectos y por qué.

Para hacerlo de este modo, tienes que comprobar si la página recibe datos (hay que mostrarlos y no generar el formulario), o si no recibe datos (hay que mostrar el formulario). Esto se puede hacer utilizando la función `isset` con una variable de las que se deben recibir (por ejemplo, poniéndole un nombre al botón de enviar y comprobando sobre él). En el siguiente código de ejemplo se muestra cómo hacerlo.

```
<?php
    if (isset($_POST['enviar'])) {
        $nombre = $_POST['nombre'];
        $modulos = $_POST['modulos'];
        print "Nombre: ".$nombre."<br />";
        foreach ($modulos as $modulo) {
            print "Modulo: ".$modulo."<br />";
        }
    }
    else {
        <?php echo $_SERVER['PHP_SELF'];?>"
method="post">
    Nombre del alumno: <input type="text" name="nombre" /><br />
    <p>Módulos que cursa:</p>
    <input type="checkbox" name="modulos[]" value="DSW" />
    Desarrollo web en entorno servidor<br />
    <input type="checkbox" name="modulos[]" value="LND" />
    Lenguaje de Marcas<br />
    <br />
    <input type="submit" value="Enviar" name="enviar"/>
    </form>
<?php
    }
?>
```

Observar que hemos puesto **el mismo name: modulos[]**. Hay que tener en cuenta que **esto es necesario en php para enviar los datos como elementos de un mismo array** ( que en el lado del servidor se reciban como elementos de un mismo array ) Sin embargo, esto no es así en otros lenguajes. Por ejemplo, **en Java podríamos haber puesto: name="modulos"**

**Nota importante:** `$_SERVER['PHP_SELF']` puede tener problemas de seguridad:

La variable `$_SERVER["PHP_SELF"]` puede ser utilizada por hackers informáticos.

Si utilizas `PHP_SELF` en tu página, entonces un usuario puede ingresar un slash (/) y luego unos comandos de ejecución "Cross Site Scripting" (XSS).

Cross-site scripting (XSS) es un tipo de vulnerabilidad en seguridad computacional encontrado usualmente en aplicaciones Web. XSS les permite a los atacantes inyectar scripts del lado del cliente en páginas Web que son vistas por otros usuarios.

Asumiendo que tenemos el siguiente formulario en una página llamada “test\_form.php”:

```
<form method="post" action="<?php echo $_SERVER["PHP_SELF"];?>">
```

Ahora, si un usuario ingresa la URL normal en la barra de direcciones `http://www.ejemplo.com/test_form.php`, el código de arriba será traducido de la siguiente manera:

```
<form method="post" action="test_form.php">
```

Hasta ahora, todo bien. Sin embargo, considerando que un usuario ingresa la siguiente URL en la barra de direcciones:

```
http://www.example.com/test_form.php/%22%3E%3Cscript%3Ealert('hacked')%3C/script%3E
```

En este caso, el código anterior se traduce en lo siguiente:

```
<form method="post" action="test_form.php"><script>alert('hacked')</script>
```

Este código añade un tag HTML y un comando de alerta y, cuando la página cargue, el código Javascript será ejecutado (el usuario verá un mensaje de alerta). Este es sólo un simple e indefenso ejemplo de cómo la variable `PHP_SELF` puede ser explotada.

Debes ser consciente de que cualquier código JavaScript puede ser añadido dentro del tag `<script>`. Un hacker puede redireccionar al usuario a un archivo en otro servidor, archivo que puede contener código malicioso que altere las variables globales o que envíe los datos del formulario a otra dirección para almacenar los datos del usuario, por poner un ejemplo.

¿Cómo evitamos ser explotados con `$_SERVER["PHP_SELF"]`?:

Los exploits para `$_SERVER["PHP_SELF"]` pueden ser evitados utilizando la función `htmlspecialchars()`. Para ello, el código del formulario debe verse de la siguiente manera:

```
<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
```

La función `htmlspecialchars()` convierte los caracteres especiales a entidades HTML. Ahora, si el usuario intenta realizar un exploit de la variable `PHP_SELF`, el resultado será lo siguiente:

```
<form method="post" action="test_form.php/&quot;&gt;&lt;script&gt;alert('hacked')&lt;/script&gt;">
```

Así el intento de exploit falla, sin daño de por medio.



Vamos a variar el ejemplo anterior revisando los datos que se obtienen antes de mostrarlos. Concretamente, se tiene que comprobar que el nombre no esté vacío, y que se haya seleccionado como mínimo uno de los módulos.

Además, en el caso de que falte algún dato, deberá generarse el formulario rellenando aquellos datos que el usuario haya introducido correctamente.

Lo primero que hacer es validar los datos. En el ejemplo propuesto será algo así:

```
if (!empty($_POST['modulos']) && !empty($_POST['nombre'])) {  
    // Aquí se incluye el código a ejecutar cuando los datos son correctos  
}  
  
else {  
    // Aquí generamos el formulario, indicando los datos incorrectos  
    // y rellenando los valores correctamente introducidos  
}
```

Para que el usuario no pierda los datos correctamente introducidos después de enviar el formulario, utiliza el atributo value en las entradas de texto (utilizamos la función isset para comprobar que la variable que queremos mostrar existe, para que no nos muestre un mensaje de error si es la primera vez que se carga la página y todavía no ha enviado nada el formulario):

```
Nombre del alumno:  
    <input type="text" name="nombre" value="<?php if (isset  
($_POST['nombre'])) echo $_POST['nombre'];?>" />
```

Y el atributo checked en las casillas de verificación:

```
<input type="checkbox" name="modulos[]" value="DWES"  
    <?php  
        if(in_array("DWS", $_POST['modulos']))  
            echo 'checked="checked"';  
    ?>  
>
```

Fíjate en el uso de la función `in_array()` para buscar un elemento en un array.

Para indicar al usuario los datos que no ha rellenado (o que ha rellenado de forma incorrecta), deberás comprobar si es la primera vez que se visualiza el formulario, o si ya se ha enviado. Se puede hacer por ejemplo de la siguiente forma:

```
Nombre del alumno:
```

```
<input type="text" name="nombre" value="<?php echo $_POST['nombre'];?>" />
<?php
    if (isset($_POST['enviar']) && empty($_POST['nombre']))
        echo "<span style='{color:red}'> &lt;-- Debe introducir un
nombre!!</span>"
?><br />
```

Bien, empezamos a tener los conocimientos básicos para empezar a trabajar. La mejor forma de asentar esos conocimientos es realizar ejercicios

● **Práctica 35:** Realiza una página con un formulario que se llame a si misma para mostrar la tabla de un número introducido por el usuario. Se deberá controlar que el usuario haya introducido un número entero positivo. Hacer uso para ello de la función: `is_int()` buscando su funcionamiento en el manual oficial: [php.net](http://php.net)

● **Práctica 36:** Realizar una página con un formulario que se llame a si misma donde el usuario introduzca en un input una cadena de números separada por espacios ( ej: 2 5 8 7 3 4 ) y muestre un número por línea, mostrando primero los números impares y luego los pares. ( hacer uso de la función `usort()` y de la función `explode()` )

Para el siguiente ejercicio haremos uso de las funciones `htmlspecialchars()` para asegurar `PHP_SELF` Y las funciones:

`preg_match()` para expresiones regulares

`filter_var()` con `FILTER_VALIDATE_EMAIL` para validar correo electrónico

Observar la siguiente página:

## Validación de formulario

\* campo obligatorio

Nombre:  \*

Correo:  \*

Página web:

Comentario:

Genero: ☐ Mujer ☐ Hombre ☐ Otro \*

PH

50

**Datos ingresados:**

- **Práctica 37:** Realizar una página como la anterior que se valide a si misma. Obligando que el correo sea válido, que el nombre no sea vacío al igual que el género. Si los datos están correctamente introducidos se mostrarán por debajo de “Datos ingresados” si no superan la validación se dirá los campos que no la superan con texto en rojo

# Objetos

PHP ha ido incorporando diferentes herramientas de programación desde sus inicios. Así la programación orientada a objetos aparece en PHP 3.0 .

Desde que nosotros ya hemos visto tal forma de programar veamos ahora simplemente como se implementa en php

Empezaremos con el clásico ejemplo de Persona:

```
<?php
class Persona{
    private $nombre;
    private $apellidos;

    function __construct(string $nombre="", string $apellidos="") {
        $this->nombre = $nombre;
        $this->apellidos = $apellidos;
    }

    public function __toString():string{
        return $this->nombre . " " . $this->apellidos;
    }

    public function setNombre(string $nombre):void{
        $this->nombre = $nombre;
    }
}

$p = new Persona("Alfon", "Martín");

$p->setNombre("Armando");

echo "<br> { $p } ";
?>
```

El acceso a atributos y métodos de un objeto es mediante el operador: ->

`$p->setNombre()` es la forma de acceder pues al método setNombre() del objeto \$p

Vemos en el ejemplo anterior que utilizamos la palabra reservada **class** para definir la clase y la palabra **new** para crear el objeto, al igual que en otros lenguajes ( Java, C#, ... )

El método: **\_\_toString()** permite a una clase decidir cómo comportarse cuando se le trata como un string. Por ejemplo, lo que echo \$obj; mostraría. Este método debe devolver un string, si no se emitirá un nivel de error fatal. Forma parte de los llamados: métodos mágicos (métodos que no se deben implementar en las clases salvo que se desee la funcionalidad “mágica” asociada a ellos )

<http://php.net/manual/es/language.oop5.magic.php#object.tostring>

Los modificadores **private** y **public** que definen la visibilidad externa e interna de los atributos y métodos también se comportan como en otros lenguajes. Así con el ejemplo anterior no es posible acceder a: \$p->nombre al ser de tipo privado

Veamos las diferencias entre public, protected y private:

**Public:** podemos acceder a las propiedades y métodos desde cualquier lugar, desde la clase actual, clases que heredan de la clase actual y desde otras clases.

**Protected:** se puede acceder al atributo o método desde la clase que lo define y desde cualquier otra que herede de esta clase.

**Private:** los atributos o métodos solo son accesibles desde la clase que los define.

Mediante **function \_\_construct()** establecemos el constructor para la clase

Observar que hemos dicho “el constructor” como en php no existe la sobrecarga de funciones únicamente tendremos un constructor. Por supuesto podemos acudir a establecer valores por defecto y comportamientos diferentes según los parámetros recibidos para emular ese comportamiento

Ok. Ahora que sabemos de que se trata un método constructor podemos aprender a usar un método destructor. Los métodos destructores, son lo contrario a un constructor. Los constructores se disparan cuando el objeto se crea, mientras que los destructores se disparan en cuanto se borra de

memoria. Si bien estos métodos no suelen utilizarse a menudo como sí pasa con los constructores, no está de más entender su funcionamiento.

Primero que nada **para destruir un objeto tenemos que usar la función de PHP `unset()`** que sirve para borrar cualquier tipo de variable de memoria, incluso un objeto. Esto puede resultar útil para liberar memoria en nuestro script.

Así que para destruir nuestro objeto agregaremos lo siguiente:

```
<?php
class Persona{
    private $nombre;
    private $apellidos;

    function __construct(string $nombre="", string $apellidos="") {
        $this->nombre = $nombre;
        $this->apellidos = $apellidos;
    }

    function __destruct(){
        echo "<br> destruyendo la persona: " . $this->toString();
    }

    public function toString():string{
        return $this->nombre . " " . $this->apellidos;
    }

    public function setNombre(string $nombre):void{
        $this->nombre = $nombre;
    }
}

$p = new Persona("Alfon", "Martín");
echo "<br> {"$p->toString()} ";
unset($p);
echo "<br> ";

$q = new Persona("a", "b", "c");
print_r($q);

unset($q);

?>
```

## Métodos y atributos estáticos

Declarar propiedades o métodos de clases como estáticos los hacen accesibles sin la necesidad de instanciar la clase. Una propiedad declarada como static no puede ser accedida con un objeto de clase instanciado (aunque un método estático sí lo puede hacer).

Ejemplo:

```
<?php
class Foo {
    public static function unMetodoEstatico() {
        echo "<br>Mensaje desde un método estático";
    }
}

Foo::unMetodoEstatico();
$nombre_clase = 'Foo';
$nombre_clase::unMetodoEstatico();
?>
```

Observar que usamos el operador de resolución de ámbito “::”

El Operador de Resolución de Ámbito o en términos simples, el doble dos-puntos, es un token que permite acceder a elementos estáticos, constantes, y sobrescribir propiedades o métodos de una clase. Cuando se hace referencia a estos items desde el exterior de la definición de la clase, se utiliza el nombre de la clase. En el ejemplo anterior lo observamos en: Foo:unMetodoEstatico();

Veamos ahora: \$nombre\_clase::unMetodoEstatico(); Primero \$nombre\_clase se convierte en: ‘Foo’ de esa forma obtenemos: Foo:unMetodoEstatico() llamando así al método de la clase

### self

La palabra reservada self la usaremos cuando queramos acceder a una constante o un método estático desde dentro de la clase

```
<?php
class MiClase {
    public static $valor_estatico = "variable estática";
}

echo MiClase::$valor_estatico;
```

?>

## uso de this y self

Se usa `$this` para hacer referencia al objeto (instancia) actual, y se utiliza `self::` para referenciar a la clase actual. Se utiliza `$this->nombre` para nombres no estáticos y se utiliza `self::nombres` para nombres estáticos.

```
<?php
class MiClase {
    private $valor_no_estatico = 1;
    private static $valor_estatico = 2;

    function __construct() {
        echo $this->valor_no_estatico . ' ' . self::$valor_estatico;
    }
}

$p = new MiClase();
?>
```

## Comparando objetos

A veces tienes dos objetos y quieres saber su relación exacta. Para eso, en PHP puedes utilizar los operadores `==` y `===`

Si utilizas el operador de comparación `==`, comparas los valores de los atributos de los objetos. Por tanto dos objetos serán iguales si son instancias de la misma clase y, además, sus atributos tienen los mismos valores.

```
$p = new Producto();

$p->nombre = 'Samsung Galaxy';

$a = clone $p ;

// El resultado de comparar $a == $p da verdadero

// pues $a y $p son dos copias idénticas
```



Sin embargo, si utilizas el operador `===`, el resultado de la comparación será true sólo cuando las dos variables sean referencias al mismo objeto.

```
$p = new Producto();  
$p->nombre = 'Samsung Galaxy';  
$a = clone($p);  
  
// El resultado de comparar $a === $p da falso  
  
// pues $a y $p no hacen referencia al mismo objeto  
  
$a = &$p;  
  
// Ahora el resultado de comparar $a === $p da verdadero  
  
// pues $a y $p son referencias al mismo objeto
```

¿ qué implica clone en el código anterior ? Pues nos hace una copia “superficial (shallow)” de los métodos del objeto clonado ( no es “deep” clone ) Así lo que se copia es la referencia de los atributos.

Ejemplo:

```
<?php  
  
class C{  
    public $variableC="paquito";  
    public function __construct(string $dato) {  
        $variableC = $dato;  
    }  
}  
  
class B{  
  
    public $variableB;  
    public $variableBB;  
  
}  
  
$b = new B();  
$b->variableBB = new C("varB");  
$b->variableB = "mi texto";  
$c = clone $b;  
$b->variableBB->variableC = "modificadoCenB";
```

```
print_r($b);  
echo "<br>";  
print_r($c);
```

?>

En el ejemplo anterior observamos que al modificar variableC nos está afectando a las dos variables: \$b y \$c , mostrando que \$c contiene atributos apuntadores idénticos que los atributos de \$b

Nosotros podemos establecer el comportamiento de clone si creamos la función: `__clone()`

Tal función se llama automáticamente al usar la palabra clave clone

El siguiente código ( tomado de php.net ) hace una variante de `__clone()` bastante interesante para lograr un deep clone

```
<?php  
class clone_base  
{  
    public function __clone()  
    {  
        $object_vars = get_object_vars($this);  
  
        foreach ($object_vars as $attr_name => $attr_value)  
        {  
            if (is_object($this->$attr_name))  
            {  
                $this->$attr_name = clone $this->$attr_name;  
            }  
        }  
    }  
}
```

```

else if (is_array($this->$attr_name))
{
    // Note: This copies only one dimension arrays
    foreach ($this->$attr_name as &$attr_array_value)
    {
        if (is_object($attr_array_value))
        {
            $attr_array_value = clone $attr_array_value;
        }
        unset($attr_array_value);
    }
}
}
}
}
}
?>

```

Observar: `get_object_vars()` Tal función te devuelve los atributos no estáticos accesibles de un objeto. Devuelve null si el parámetro que se le pasa no es un objeto

## Herencia en php

La herencia es un principio de programación bien establecido y PHP hace uso de él en su modelado de objetos. Este principio afectará la manera en que muchas clases y objetos se relacionan unas con otras.

Por ejemplo, cuando se extiende una clase, la subclase hereda todos los métodos públicos y protegidos de la clase padre. A menos que una clase sobrescriba esos métodos, mantendrán su funcionalidad original.

Esto es útil para la definición y abstracción de la funcionalidad y permite la implementación de funcionalidad adicional en objetos similares sin la necesidad de reimplementar toda la funcionalidad compartida.

Para crear clases que hereden de otra utilizamos la palabra reservada: **extends**

### Ejemplo

```
<?php

class Foo
{

    public function printItem($string)
    {
        echo 'Foo: ' . $string . '<br>';
    }

    public static function printPHP()
    {
```

```

        echo 'PHP is great.' . '<br>';
    }
}

class Bar extends Foo
{
    public function printItem($string)
    {
        echo 'Bar: ' . $string . '<br>';
    }
}

$foo = new Foo();
$bar = new Bar();
Foo::printPHP('baz');

$bar->printItem('baz');
Bar::printPHP();
?>

```

En el ejemplo anterior se puede observar como se sobrescribe un método heredado y como un método que únicamente se ha declarado en el padre también forma parte de la clase

## parent

Por medio de `parent` podemos acceder a variables y funciones del padre desde dentro de la clase:

```

<?php
class A{
    function datos(){
        echo "<br> datos de A";
    }

    function __construct(){
        echo "<br> A creado";
    }
}

class B extends A{
    function __construct(){
        //parent::__construct();
        parent::datos();
        echo "<br> datos de B";
    }
}

```

```
}  
  
$b = new B();  
?>
```

En el anterior ejemplo se llama a `parent::datos()` en el constructor de B. Observar que a diferencia de otros lenguajes no se ejecuta primero el constructor de la clase padre A. Pero podemos llamar al constructor del padre si lo queremos descomentando: `parent::__construct()`