

PHP

INICIACIÓN a PHP	6
Fragmentos PHP	6
Fragmentos PHP y fragmentos HTML	6
Código HTML de páginas HTML válidas	6
Generar el código HTML de páginas HTML válidas	7
Enlaces a hojas de estilo	7
El delimitador <? ... ?>	8
El delimitador <?= ... ?>	8
Diferencias entre echo y print	8
print_r vs var_dump	8
Comentarios en páginas PHP	8
Tipos de datos	10
Cadenas	10
Comillas dentro de cadenas	10
Diferencias entre comillas simples y dobles	10
Comillas en código html / css	10
Caracteres especiales	10
Concatenar cadenas	10
Añadiendo saltos de línea	11
Variables	11
Usar variables	12
Concatenar variables y cadenas	12
Variables en cadenas	12
Tipos de variables	12
Operaciones aritméticas	12
Nombres de variables	13
Unir variables y texto	13
Tipos de variables	13
Variables lógicas (boolean)	13
Variables enteras (integer)	13
Notación complemento a dos	14
Variables decimales (float)	14
Variables de cadenas (string)	14
Conversiones de tipos	14
Conversión Explícita	15
Conversión Automática	15
Variables como variables lógicas	15
Qué es una matriz	15
Crear una matriz	16
Matrices asociativas	16

Matrices multidimensionales	16
Imprimir todos los valores de una matriz: la función print_r()	16
Añadir elementos a una matriz	17
Unión de matrices	17
Más información general sobre las matrices	17
Borrar una matriz o elementos de una matriz	17
Copiar una matriz	17
Variables predefinidas	18
Constantes y constantes predefinidas	18
Constantes	18
Definir constantes	18
Constantes predefinidas	19
Lista completa de constantes predefinidas	19
Operaciones aritmética	19
Números	19
Operadores aritméticos	19
Resto de una división	20
Paréntesis	20
Operadores combinados	20
Redondear un número	20
Potencias	21
Máximo y mínimo	21
Formatear un número	21
Números aleatorios	21
Operaciones lógicas	22
Comparaciones	22
Operadores lógicos	23
Diferencia entre and y && y entre or y	23
Estructuras de Control	24
Bloques de instrucciones	24
Sentencia condicional if ... elseif ... else ...	24
Sentencia condicional if ... else ...	24
Notación abreviada .. ? ... : ...	25
Sentencia condicional if ... elseif ... else ..	25
Sintaxis alternativa sentencia condicional if ... elseif ... else ...	26
Sentencia condicional switch	27
Sintaxis alternativa sentencia condicional switch	27
Bucle for	27
Ejemplos de bucles incorrectos	29
Sintaxis alternativa bucle for	29
Contador	29
Acumulador	29
Bucles anidados	29
Bucles anidados con variables independientes	29

Bucles anidados con variables dependientes	30
Bucle while	30
Sintaxis alternativa bucle while	30
Bucle do ... while	30
Bucle foreach	31
Sintaxis alternativa bucle foreach	32
Funciones de matrices/arrays	32
Comparar arrays	32
array_diff	32
array_diff_assoc	32
array_diff_key	32
Unir arrays	33
Operador suma	33
array_merge	33
array_merge_recursive	33
array_combine	33
Rellenar arrays	33
array_fill	33
range	33
Dividir arrays	34
array_slice	34
array_splice	34
array_chunk	34
Modificar elementos en arrays	34
array_shift	34
array_unshift	35
array_pop	35
array_push	35
array_flip	35
Contar elementos de un array	35
Contar cuántas veces aparece cada valor en un array	35
Máximo y mínimo	36
Ordenar un array	36
Buscar un valor en un array	37
Reindexar un array	37
Barajar los elementos de un array	37
Extraer al azar un elemento de un array	37
Eliminar valores repetidos	37
Funciones	37
La sintaxis clásica de la función es:	38
La sintaxis moderna de la función es:	38
Funciones con argumentos	39
Pasar argumentos por valor	39
Pasar argumentos por referencia	40

Valores de argumentos predeterminados	40
Tipos admitidos	41
Funciones variables	41
Ámbito de las variables	41
Anónimas	43
Bibliotecas	43
Controles en Formularios	59
Introducción	59
Atributo method	59
Formulario con get	60
Formulario con post	61
Botón Enviar	62
Caja de texto, caja de contraseña y área de texto	65
Casilla de verificación	67
Botón radio	68
Menú (select)	70
Control oculto (input hidden)	72
Imagen (input image)	72
Archivo (input file)	74
Recogida de datos	75
Matriz \$_REQUEST	75
Referencia a \$_REQUEST dentro y fuera de cadenas	75
Comprobación de existencia	78
1-Controles vacíos == los controles no contengan ningún valor.	78
2- Controles inexistentes: isset()	79
Seguridad en las entradas	81
Script PHP vulnerable (login.php)	82
Ejemplo de ataque:	82
Eliminar espacios en blanco iniciales y finales: trim(\$cadena)	83
Utilización de variables	84
htmlspecialchars()	86
Funciones de recogida de datos	88
Comprobación de datos	90
Comprobación de números con is_numeric() y ctype_digit()	91
Comprobación de números con is_numeric()	91
Comprobación de números enteros positivos con ctype_digit()	91
Funciones is_	91
Funciones ctype_	93
Funciones filter_	94
Funciones xxx_exists()	95
Función function_exists()	95
Función array_key_exists()	96
El protocolo HTTP	96
Solicitudes HTTP	96

Respuestas HTTP	97
Cabeceras: la función header()	98
Función header()	98
Redirecciones: header("Location:...")	101
Resto del programa tras la redirección	103
Crear con PHP otros tipos de archivos	105
Crear hojas de estilo CSS	106
Crear imágenes SVG	106
SESIONES	108
MOTIVACIÓN	108
Partes del trabajo con sesiones	109
Creación o apertura de la sesión	109
Consideraciones importantes:	110
El SID (Session ID) no se muestra explícitamente, pero está implícito en la función session_start().	110
¿Dónde está físicamente el SID?	111
Utilización de la sesión	111
Destrucción o cierre de la sesión	115
Nombre de sesión	116
Borrar elementos de la sesión	117
Directiva session.save_handler	119
Seguridad en las sesiones	122
Sesión Time-Outs.	123
Regenerar el Session ID.	123
Cuándo regenerar el ID de sesión:	123
Destruir la sesión.	124
Utilizar almacenamiento permanente.	125

INICIACIÓN a PHP

Nota: PHP requiere que las instrucciones **terminen en punto y coma** al final de cada sentencia. La etiqueta de cierre de un bloque de código de PHP automáticamente implica un punto y coma, por tanto no es necesario usar un punto y coma para cerrar la última línea de un bloque de PHP.

Fragmentos PHP

- ☐ `<?php` (etiqueta inicial)
- ☐ `?>` (etiqueta final)
- ☐ Los fragmentos de PHP no pueden anidarse, es decir, no puede haber un fragmento dentro de otro.
- ☐ Si el programa termina con un fragmento PHP, ese último fragmento PHP no necesita cerrarse. **RECOMENDACIÓN , cerrar siempre los fragmentos**

Fragmentos PHP y fragmentos HTML

- ☐ Los fragmentos PHP tienen que generar las etiquetas HTML que se necesiten para una correcta visualización de la página web en el navegador.
- ☐ Como la página PHP se lee secuencialmente, el código HTML generado por los fragmentos PHP y el incluido en los fragmentos HTML se encuentran en el mismo orden en que se encontraban los fragmentos en la página PHP.
- ☐ En un fragmento PHP no pueden escribirse etiquetas HTML sueltas; el código HTML debe generarse siempre con instrucciones de PHP.

Código HTML de páginas HTML válidas

- ☐ Si queremos crear páginas HTML válidas debemos generar código HTML válido, es decir, generar todas las etiquetas HTML necesarias respetando las reglas sintácticas.
- ☐ La etiqueta `<!DOCTYPE ... >` (línea 1) es obligatoria
- ☐ La etiqueta `<html> ... </html>` (líneas 2 a 13) engloba todo el documento html.
- ☐ El encabezado `<head> ... </head>` .
- ☐ Las etiquetas `<meta>` están pensadas para proporcionar información sobre el documento a los programas que analicen la página y por ello existen muchas etiquetas `<meta>` diferentes.
- ☐ La etiqueta de título `<title> ... </title>` contiene el texto que se muestra en la barra de título de la ventana del navegador. La etiqueta `title` es obligatoria y debe incluirse en todas las páginas web.
- ☐ El enlace a una hoja de estilo no es obligatorio.
- ☐ El cuerpo (`<body> </body>`) contiene lo que se verá en la ventana del navegador.

- ☐ La etiqueta de párrafo `<p> ... </p>` contiene el texto que se muestra en la ventana del navegador

Generar el código HTML de páginas HTML válidas

- ☐ La página PHP debe incluir o generar todo el código HTML necesario.
- ☐ EJEMPLO

```
<?php
print "<!DOCTYPE html>
<html lang=\"es\">
<head>
  <meta charset=\"utf-8\">
  <title>Página HTML válida</title>
  <meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0\">
</head>

<body>
  <p>Esta página es una página HTML válida.</p>
</body>
</html>";
?>
```

Enlaces a hojas de estilo

- ☐ Una página PHP puede enlazar a una hoja de estilo CSS exactamente igual que una página HTML, es decir, enlazando a la hoja de estilo mediante la etiqueta `<link>`.
- ☐ EJEMPLO

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>Página HTML válida</title>
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
<?php
print " <link rel=\"stylesheet\" href=\"estilo.css\" title=\"Color\">\n";
?>
</head>

<body>
<?php
print " <p>Esta página es una página HTML válida.</p>\n";
?>
</body>
</html>
```

El delimitador <? ... ?>

- ☐ El delimitador <? ... ?> no se debe utilizar. Se comenta porque todavía se puede encontrar en código escrito hace muchos años.
- ☐ Mediante la directiva short_open_tag (etiqueta de apertura abreviada) se puede permitir el uso del delimitador <? además del delimitador <?php. Si la directiva short_open_tag tiene el valor On (lo que está desaconsejado)

El delimitador <?= ... ?>

- ☐ El delimitador <?= es una abreviatura de la expresión <?php echo que se utiliza en muchos frameworks de PHP en los documentos que sirven de plantilla de generación de interfaces,

Diferencias entre echo y print

- ☐ Ambas funciones nos permiten mostrar cadenas de textos y ambas funciones no llevan paréntesis al momento de llamarlas
- ☐ int print (string \$arg)
- ☐ void echo (string \$arg1 [, string \$...])
- ☐ print imprime una cadena, echo puede imprimir más de una separadas por coma
- ☐ print devuelve un valor int que según la documentación siempre es 1, por lo que puede ser utilizado en expresiones mientras que echo es tipo void, no hay valor devuelto y no puede ser utilizado en expresiones:

print_r vs var_dump

- ☐ Estas dos funciones imprimen los detalles de una variable, incluyendo su valor, en un formato legible por el humano
- ☐ Si el valor de la variable es una cadena de texto, var_dump imprime la cadena entre dobles comillas, print_r no.
- ☐ print_r no imprime nada visible para false y cadenas vacías.
- ☐ var_dump proporciona información sobre el tamaño y tipo de datos de la variable y, en el caso de arrays y objetos, de los elementos que la componen. print_r no da información sobre el tamaño de la variable ni sobre el tipo de datos
- ☐ print_r puede devolver el resultado en lugar de imprimirlo si se proporciona el segundo parámetro como true

Comentarios en páginas PHP

En un fragmento PHP se pueden comentar líneas de código utilizando:

- ☐ // para comentar el resto de la línea (como en C++)
- ☐ # para comentar el resto de la línea (como en la shell de Unix o en Perl)
- ☐ /* */ para delimitar varias líneas (como en C) Estos comentarios no se añaden al código HTML generado por la página, por lo que no pueden verse en el navegador

Estos comentarios no se añaden al código HTML generado por la página, por lo que no pueden verse en el navegador.

IMPORTANTE; Si se quieren escribir comentarios en los fragmentos HTML, hay que utilizar la etiqueta de comentarios de HTML . Estos comentarios, como todo el código HTML situado en los fragmentos HTML, se incluyen sin modificaciones en el resultado, por lo que pueden verse en el navegador.

Tipos de datos

Cadenas

- ☐ En PHP, las cadenas de texto se delimitan por comillas (dobles o simples)
- ☐ Si las comillas de apertura y cierre no son del mismo tipo (una de ellas doble y otra simple), se produce un error de sintaxis.

Comillas dentro de cadenas

- ☐ Si una cadena está delimitada por comillas dobles, en su interior puede haber cualquier número de comillas simples, y viceversa.
- ☐ Lo que no puede haber en una cadena es una comilla del mismo tipo que las que delimitan la cadena.
- ☐ Para poder escribir en una cadena una comilla del mismo tipo que las que delimitan la cadena, se debe utilizar los caracteres especiales \' o \".
- ☐ Pero los caracteres especiales \" y \' no se pueden utilizar para delimitar cadenas

Diferencias entre comillas simples y dobles

- ☐ **PHP no sustituye las variables que se encuentran dentro de** cadenas delimitadas con comillas simples, mientras que sí que lo hace (pero no siempre) si se utilizan **comillas dobles**

Comillas en código html / css

- ☐ En el código HTML generado con PHP también se pueden escribir comillas simples o dobles.

Caracteres especiales

- ☐ En PHP los nombres de variables empiezan por el carácter dólar (\$). Si a esa variable no se le ha dado valor anteriormente, se producirá un aviso.
- ☐ Para poder escribir el carácter \$ se tiene que anteponer el carácter contrabarra (\).

Concatenar cadenas

- ☐ El operador . (punto) permite concatenar dos o más cadenas.
- ☐ El operador de asignación con concatenación (.=) permite concatenar una cadena a otra y asignarla a esta:

Añadiendo saltos de línea

- ☐ \n para realizar saltos de línea puede dar problemas de compatibilidad entre sistemas operativos ya que la forma de realizar un salto de línea varía en cada uno. **PHP_EOL** selecciona automáticamente el correcto.

```
<?php
```

```
echo '<p>En un lugar de la Mancha,</p>' . PHP_EOL . '<p>de  
cuyo nombre no quiero acordarme...</p>';
```

```
?>
```

Variables

- ☐ En PHP el programador puede dar el **nombre** que quiera a las variables, con algunas **restricciones**:
 - ☐ Los nombres de las variables tienen que empezar por el carácter \$
 - ☐ A continuación tiene que haber una letra (mayúscula o minúscula) o un guión bajo (_).
 - ☐ El resto de caracteres del nombre pueden ser números, letras o guiones bajos.
- ☐ Los nombres de variables pueden contener caracteres no ingleses (vocales acentuadas, eñes (ñ) o cedillas (ç), etc.) pero se aconseja no hacerlo
- ☐ En los nombres de las variables, PHP **distingue entre mayúsculas y minúsculas**
- ☐ Si el nombre de la variable contiene varias palabras, se aconseja utilizar la **notación "camel case"**, es decir, escribir la primera palabra en minúsculas y la primera letra de las siguientes palabras en mayúsculas.
- ☐ PHP define automáticamente una serie de variables (son las llamadas variables predefinidas). Los nombres de estas variables siguen siempre el mismo patrón: empiezan por un guión bajo y se escriben en mayúsculas (por ejemplo, **\$_REQUEST**, **\$_SERVER**, **\$_SESSION**, etc.). **Para evitar conflictos** con variables predefinidas que se creen en el futuro, **se recomienda no crear en los programas variables con nombres que sigan ese mismo patrón.**
- ☐ En PHP, no es necesario declarar las variables: al inicializarlas queda especificado el tipo.
- ☐ Cualquier variable puede cambiarse de tipo con funciones como intval(), floatval() o strval():
- ☐ Para averiguar si una variable existe podemos usar la función **isset()**, que nos devuelve **true si la variable existe** y false en caso contrario. **Del mismo modo**, hay otra función muy útil, **unset()**, que **hace desaparecer a una variable ya definida** y libera la memoria que ocupaba:
- ☐ Los tipos de datos predefinidos en PHP son:
 - ☐ integer (entero)
 - ☐ float (real)
 - ☐ bool (booleano)
 - ☐ string (cadena)
 - ☐ array

Usar variables

- ☐ Para guardar un valor en una variable se utiliza el operador de asignación (=) escribiendo a la izquierda únicamente el nombre de la variable y a la derecha el valor que queremos guardar
 - ☐ Si queremos guardar un número, no hace falta poner comillas
 - ☐ Si queremos guardar una cadena de texto hay que poner comillas (dobles o simples).
- ☐ En el lado izquierdo de la asignación (=) no se puede escribir más que el nombre de una variable.
- ☐ En la parte derecha de la asignación se pueden escribir expresiones matemáticas. Esas expresiones pueden contener variables.
- ☐ En PHP una igualdad no es una ecuación matemática, sino una asignación (el resultado de la derecha se almacena en la variable de la izquierda).

Concatenar variables y cadenas

- ☐ El operador . (punto) permite concatenar dos o más cadenas o variables (pudiendo ser resultados de operaciones).
- ☐ El operador . (punto) se puede utilizar en la instrucción print. En el ejemplo siguiente se concatenan una cadena, una variable y una cadena.

Variables en cadenas

- ☐ Depende del tipo de variable utilizado.

Tipos de variables

- ☐ En el caso de números, cadenas o matrices de una dimensión, las variables se puede insertar directamente:
- ☐ En el caso de matrices de dos o más dimensiones, las variables no se puede insertar directamente:
 - ☐ Una solución a este problema es sacar la matriz de la cadena:
 - ☐ Otra solución sin necesidad de sacar la matriz de la cadena es rodear la variable con llaves ({ }):

Operaciones aritméticas

- ☐ En el interior de las cadenas no se realizan operaciones aritméticas.
- ☐ Si se quiere mostrar el resultado de operaciones matemáticas, es necesario efectuar las operaciones fuera de las cadenas
- ☐ Se recomienda no escribir las operaciones entre paréntesis cuando no sea necesario.
- ☐ Siempre se pueden utilizar variables auxiliares que guarden los resultados y utilizar las variables auxiliares en la cadena

Nombres de variables

- ☐ Si se quiere escribir el nombre de una variable, es decir, para que PHP no sustituya la variable por su valor, hay que escribir una contrabarra (\) antes de la variable.

Unir variables y texto

- ☐ Si el tamaño está almacenado en una variable, no se puede juntar la variable con los caracteres ya que se interpretaría como una variable que no está definida y toma el valor vacío
- ☐ Si el tamaño está almacenado en una variable, se pueden utilizar llaves o sacar la variable de la cadena:
- ☐ Dentro de las llaves no se pueden realizar operaciones. Si se realizan operaciones, se puede producir un error de sintaxis u obtener resultados no esperados:

Tipos de variables

- ☐ Los tipos de variables básicos son los siguientes:
 - ☐ lógicas o booleanas (boolean)
 - ☐ enteros (integer)
 - ☐ decimales (float)
 - ☐ cadenas (string)
 - ☐ matrices (arrays)
- ☐ Existen además los tipos:
 - ☐ objetos (object)
 - ☐ recursos (resource)
 - ☐ nulo (null)

Variables lógicas (boolean)

- ☐ Las variables de tipo lógico sólo pueden tener el valor true (verdadero) o false (falso). Se suelen utilizar en las estructuras de control.
- ☐ Estos valores se pueden escribir en mayúsculas o minúsculas o combinando ambas, aunque se recomienda utilizar minúsculas
- ☐ : No es necesario comparar una variable lógica con true o false, podemos emplear variables lógicas directamente en la condición del if.

Variables enteras (integer)

- ☐ Las variables de tipo entero pueden guardar números enteros (positivos o negativos)
- ☐ Definición del tamaño de variables **PHP_INT_SIZE**
- ☐ Valor más grande almacenable **PHP_INT_MAX**.

Notación complemento a dos

- ☐ Se utiliza la notación complemento a dos, en la que la primera mitad de los valores representan los valores positivos y la segunda mitad de los valores representan los valores negativos

Variables decimales (float)

- ☐ Las variables de tipo decimal (float) pueden guardar números decimales (positivos o negativos).
- ☐ El separador de la parte entera y la parte decimal es el punto (.), no la coma (,).
- ☐ Las variables decimales utilizan un número de bytes fijo que depende del ordenador, **(no existen constantes predefinidas para conocer su tamaño)**
- ☐ Normalmente las variables decimales siguen la norma IEEE 754, concretamente el formato denominado "doble precisión", que emplea 8 bytes (64 bits) para almacenar un número.
 - ☐ De esos 64 bits, uno se utiliza para el signo, 53 para la parte fraccionaria y 11 para el exponente. Eso hace que el valor más grande que se pueda almacenar sea aproximadamente $1,8 \cdot 10^{308}$
- ☐ Si se intenta guardar un número demasiado grande positivo PHP no puede manejarlo y lo que hace es sustituirlo por la constante predefinida INF.

Variables de cadenas (string)

- ☐ Las variables de tipo cadena pueden guardar caracteres.
- ☐ PHP no impone ningún límite al tamaño de las cadenas.
- ☐ El juego de caracteres que utiliza PHP viene determinado en principio por el juego de caracteres que utiliza el fichero fuente del programa.
- ☐ Las funciones de tratamiento de cadenas no están preparadas para tratar la diversidad de juegos de caracteres
- ☐ Se puede acceder a caracteres individuales indicando la posición del carácter, como si se tratara de una matriz de una dimensión en la que el primer carácter ocupa la posición 0.
- ☐ Si se indica una posición mayor que la longitud de la cadena, la cadena se alarga con espacios hasta llegar a ese valor:
- ☐ Si en una posición se guarda una cadena vacía, la cadena se acorta eliminando el carácter de esa posición

Conversiones de tipos

- ☐ PHP permite convertir variables de un tipo en otro tipo o considerar variables de un tipo como otro.

Conversión Explícita

- ☐ Este tipo de conversión puede realizarse mediante type casting y funciona de la misma manera como funciona en el lenguaje C, anteponiendo el tipo deseado entre paréntesis justo antes de la variable.
- ☐ Tipos de cast en PHP.

Expression	Final type
(int), (integer)	integer
(bool), (boolean)	boolean
(float), (double), (real)	float
(string)	string
(array)	array
(object)	object

- ☐ Para que la variable en sí cambie de tipo está la función settype().

Conversión Automática

- ☐ La conversión automática de tipos se realiza cuando se necesita un tipo específico de datos en una expresión

Variables como variables lógicas

- ☐ Si una variable entera o decimal cuyo valor sea igual a cero se considera como variable lógica, se considera que tiene el valor false. Si una variable entera o decimal distinta de cero se considera como variable lógica, se considera que tiene el valor true.
- ☐ Si una cadena vacía se considera como variable lógica, se considera que tiene el valor false. Si una cadena no vacía se considera como variable lógica, se considera que tiene el valor true.
- ☐ Las cadenas no vacías se consideran como true, aunque tengan un valor que puedan confundirnos:
- ☐ Las matrices se consideran siempre como true:

Qué es una matriz

- ☐ Una matriz es un tipo de variable que permite almacenar simultáneamente varios datos diferentes, a los que se accede mediante un índice, numérico o de texto
- ☐ En PHP, una matriz es un tipo de variable muy flexible, ya que podemos añadir, modificar, eliminar o reordenar los elementos de forma individual. **Además los elementos pueden ser de tipos de datos diferentes**
- ☐ Si los elementos de una matriz son datos de tipos simples (booleanos, enteros, decimales o cadenas), sólo se necesita un índice para identificar los datos. Se dice

entonces que las matrices son unidimensionales. A las **matrices de una dimensión** también se les llama **vectores**.

- ☐ Si los elementos de una matriz son a su vez también matrices, se necesitan varios índices para identificar a los datos. Se dice entonces que **las matrices son multidimensionales**.

Crear una matriz

- ☐ En la notación compacta, las matrices se crean empleando corchetes (`[]`).
- ☐ Los elementos de la matriz deben separarse con comas.
- ☐ Tras el último elemento se puede escribir o no una coma, pero **la coma final no crea un nuevo elemento**
- ☐ Para hacer referencia a los valores individuales de la matriz, se deben utilizar índices, que se escriben entre corchetes (`[]`).
- ☐ Si al crear la matriz no se han indicado otros valores de índices, el primer término tiene el índice `[0]`, el segundo tiene el índice `[1]`, etc.
- ☐ Para referirse a un elemento individual, hay que indicar siempre el índice correspondiente.
- ☐ Si se solicita un valor no definido de una matriz, se produce un aviso (`undefined offset`). **Los avisos no interrumpen la ejecución del programa**
- ☐ Se puede crear una matriz vacía (para añadirle posteriormente elementos).
- ☐ También se pueden crear matrices asignando directamente un elemento de la matriz.

Matrices asociativas

- ☐ Las matrices de PHP son matrices asociativas, es decir, que los índices no tienen por qué ser correlativos, ni siquiera tienen por qué ser números.
- ☐ Al crear matrices asociativas, debemos indicar el valor de los índices, utilizando la notación `$índice => $valor`
- ☐ PHP **sustituye las referencias a valores de matrices de una dimensión dentro de las cadenas**, por lo que **no es necesario concatenar cadenas y referencias a matrices**, pero **los índices deben escribirse sin comillas**, aunque sean cadenas.
- ☐ Si la referencia a un valor de una matriz está fuera de una cadena o entre llaves, los índices que son cadenas deben escribirse con comillas.

Matrices multidimensionales

- ☐ PHP no sustituye las referencias a valores de matrices de más de una dimensión dentro de las cadenas, por lo que se necesita o bien utilizar llaves, o bien concatenar cadenas y referencias a matrices

Imprimir todos los valores de una matriz: la función `print_r()`

- ☐ La instrucción `print` permite imprimir valores individuales de una matriz, pero no matrices completas.

- ☐ La función `print_r($variable [, $devolver])` permite imprimir todos los valores de una matriz de forma estructurada. En general, `print_r()` imprime cualquier variable compuesta de forma legible.
- ☐ Aunque `print_r()` genera espacios y saltos de línea en el código fuente de la página para indicar el anidamiento, `print_r()` no genera etiquetas html, por lo que el navegador no muestra esos espacios y saltos de línea.
- ☐ Si se añade un segundo argumento con el valor `true`, `print_r()` no imprime nada pero devuelve el texto que se imprimiría si no estuviera el argumento `true`

Añadir elementos a una matriz

- ☐ En la notación compacta, se pueden añadir elementos a una matriz indicando o no el índice del nuevo elemento:
 - ☐ Si no se indica el índice, el nuevo elemento toma como índice el siguiente al mayor de los existentes (o el índice 0 si no había ningún valor numérico)
 - ☐ Indicando el índice, se asigna el elemento de la matriz

Unión de matrices

- ☐ Los operadores `+` (suma) `+=` (suma combinada) realizan la unión de dos matrices. La unión de dos matrices contiene **todos los elementos de la primera matriz y únicamente los elementos de la segunda matriz cuyo índice no se encuentra en la primera matriz**.
 - ☐ Si los índices no coinciden, la unión contendrá todos los elementos:
 - ☐ El operador combinado `+=` realiza también la unión de dos matrices (y modifica, en su caso, la primera matriz).

Más información general sobre las matrices

- ☐ Cada elemento de la matriz se comporta como una variable independiente y se pueden almacenar datos de tipos distintos en una misma matriz.
- ☐ Los valores de los índices numéricos suelen ser siempre positivos, pero PHP también permite que sean negativos.
- ☐ PHP no permite valores decimales en los índices numéricos. En caso de usarlos, PHP los trunca automáticamente a enteros

Borrar una matriz o elementos de una matriz

- ☐ La función `unset()` permite borrar una matriz o elementos de una matriz.
- ☐ Si se intenta borrar un elemento no definido, PHP no genera ningún aviso.

Copiar una matriz

- ☐ Se puede copiar una matriz creando una nueva variable

Variables predefinidas

- ☐ PHP genera automáticamente una serie de variables con diversa información sobre el cliente y el servidor.
 - ☐ **\$_REQUEST** es una matriz asociativa que contiene los datos enviados por los formularios y las cookies guardadas en el ordenador del cliente.
 - ☐ **\$_SERVER** es una matriz asociativa que contiene **información sobre cabeceras, rutas y ubicaciones de scripts suministrada por el servidor** (*pero hay que tener en cuenta que no todos los servidores suministran todos los datos*).

Esta variable se puede utilizar en las páginas que enlazan consigo mismas. Por ejemplo, una página puede contener un formulario que envíe los datos a esa misma página. Para ello, el atributo action del formulario debe contener el nombre de la página. En vez de poner el nombre de la página, se puede utilizar `$_SERVER[PHP_SELF]`. La ventaja es que aunque se cambie el nombre del fichero, el enlace seguirá funcionando.

Constantes y constantes predefinidas

- ☐ PHP permite definir constantes e incluye una serie de constante predefinidas que se pueden utilizar directamente:

Constantes

- ☐ Las constantes son **elementos de PHP que guardan un valor fijo que no se puede modificar a lo largo del programa**. Las constantes **pueden ser definidas por el programa o estar predefinidas por el propio PHP** o por algún módulo. Los nombres de las constantes siguen las **mismas reglas que los nombres de las variables, pero sin el dólar (\$) inicial**. La costumbre es escribir los nombres de las constantes en mayúsculas.
- ☐ El inconveniente de usar constantes es que las constantes no se sustituyen dentro de las cadenas y es necesario sacarlas fuera de las cadenas.

Definir constantes

- ☐ La función **define(nombre_constante, valor_constante)** permite definir constantes
- ☐ Otra sintaxis que nos ofrece PHP es usando la palabra **const**, como en JavaScript
- ☐ **La costumbre es escribir los nombres de las constantes en mayúsculas**. Los nombres de las constantes deben empezar por una letra y tener sólo letras (acentos, etc), números y guiones bajos.
- ☐ Las constantes **no se sustituyen dentro de una cadena** (ni siquiera escribiéndose entre llaves), por lo que es necesario concatenarlas para mostrar su valor.
- ☐ En las constantes, PHP distingue entre minúsculas y mayúsculas:

Constantes predefinidas

Lista completa de constantes predefinidas

- ☐ Depende de los módulos cargados en php.ini
- ☐ La función **get_defined_constants()** devuelve las constantes predefinidas en el servidor que estemos utilizando:
- ☐ **INF**; representa el infinito, es decir, cualquier número demasiado grande (positivo o negativo) para poderse guardar en una variable decimal.
- ☐ **PHP_INT_MAX**; valor del mayor entero que se puede guardar en una variable de tipo entero. Ese valor depende del tamaño en bytes que ocupan las variables en la memoria, que depende del microprocesador, del sistema operativo y de la versión de PHP. **En sistemas de 64 bits, los enteros suelen ocupar 8 bytes (64 bits) y el valor máximo suele ser 9.223.372.036.854.775.807 (2⁶³ - 1)**
- ☐ **PHP_INT_SIZE**; es el tamaño en bytes de las variables de tipo entero, que depende del ordenador, del sistema operativo y de la versión de PHP. En sistema de 64 bits suele ser 8 bytes.

Operaciones aritmética

Números

- ☐ Los números decimales se escriben con punto (.), no con coma (,).
- ☐ Cuando estás **multiplicando números**, el **resultado final** será de tipo **flotante** ***si al menos uno de los operandos era flotante.*** No importa si el número final tiene una parte decimal o no.

Operadores aritméticos

Ejemplo	Nombre	Efecto
-\$a	-\$a Negación	El opuesto de \$a.
\$a + \$b	Suma	Suma de \$a y \$b
\$a - \$b	Resta	Diferencia entre \$a y \$b.
\$a * \$b	Multiplicación	Producto de \$a y \$b.
\$a / \$b	División	Cociente de \$a y \$b.
\$a % \$b	Módulo	Resto de \$a dividido por \$b.
++\$a	Pre-incremento	Incrementa \$a en uno, y luego devuelve \$a.
\$a++	Post-incremento	Devuelve \$a, y luego incrementa \$a en uno.
--\$a	Pre-decremento	Decrementa \$a en uno, luego devuelve \$a.

Ejemplo	Nombre	Efecto
\$a--	Post-decremento	Devuelve \$a, luego decrementa \$a en uno.

- ☐ pre-incremento , incrementa la variable y después se utiliza
- ☐ post-incremento , primero se utiliza y después se incrementa.
- ☐ El operador de ***incremento funciona también con caracteres***, teniendo en cuenta que al incrementar el carácter 'z' se obtiene la cadena 'aa'.
- ☐ El operador de **decremento** no funciona con caracteres

Resto de una división

- ☐ El operador % calcula el resto de una división entera
- ☐ La función **fmod** calcula el resto de una división con números decimales

Paréntesis

- ☐ Los operadores aritméticos tienen el mismo orden de precedencia que en Matemáticas
- ☐ de mayor a menor (los operadores indicados en el mismo grupo se efectúan en el orden en que aparecen en la expresión)
 - ☐ ++ (incremento) -- (decremento)
 - ☐ * (multiplicación) / (división) % (resto)
 - ☐ + (suma) - (resta)
- ☐ Se aconseja **no utilizar paréntesis cuando las operaciones den el mismo resultado** con o sin paréntesis.

Operadores combinados

Ejemplo	Nombre	Equivale a
\$a += \$b	Suma	\$a = \$a + \$b
\$a -= \$b	Resta	\$a = \$a - \$b
\$a *= \$b	Multiplicación	\$a = \$a * \$b
\$a /= \$b	División	\$a = \$a / \$b
\$a %= \$b	Módulo	\$a = \$a % \$b

Redondear un número

- ☐ La función **round(x)** redondea el número x al entero más próximo (toma el número más cercano en la recta numérica)

- ☐ La función **round(x,n)** redondea *x con n decimales* (**si n es negativo redondea a decenas, centenas, etc.**).
- ☐ La función **floor(x)** redondea el número x al entero inferior (es decir, devuelve la parte entera).
- ☐ La función **ceil(x)** redondea el número x al entero superior.

Potencias

- ☐ La función **pow(x, y)** calcula x elevado a y.
- ☐ En PHP 5.6 se introdujo el operador **, equivalente a la función pow().

Máximo y mínimo

- ☐ Las funciones **max()** y **min()** devuelven el máximo y el mínimo, respectivamente, de una lista o matriz de valores.

Formatear un número

- ☐ Para escribir un número con los símbolos de separación de decimales y de miles españoles, es decir, una coma (,) para separar la parte entera de la decimal y un punto (.) para separar las cifras de la parte entera en grupos de tres, se puede utilizar la función **number_format()**. La función requiere dos o cuatro argumentos:
 - ☐ El primer argumento es el número a formatear.
 - ☐ El segundo argumento es el número de decimales a mostrar (el número se redondea o trunca dependiendo de la longitud del número).
 - ☐ El tercer argumento es el carácter a utilizar como separador de la parte entera de la decimal.
 - ☐ El cuarto argumento es el carácter a utilizar como separador de miles
- ☐ **Si sólo se utilizan dos argumentos**, se utiliza el punto como separador de parte entera y decimal y la coma como separador de miles (**notación inglesa**).

Números aleatorios

- ☐ Para obtener un número entero aleatorio entre dos valores determinados, se pueden utilizar la función **rand()** o la función **mt_rand()**
 - ☐ El primer argumento es el valor mínimo que se quiere obtener
 - ☐ El segundo argumento es el valor máximo que se quiere obtener.
- ☐ Si se llama a las funciones **sin argumentos**, estas *devuelven un número entero aleatorio entre 0 y un valor máximo que es el que devuelve la función **getrandmax()** o **mt_getrandmax()** (este valor depende del sistema operativo)*.
- ☐ **En algunas plataformas** (como en Windows), **getrandmax()** sólo alcanza hasta **32767**. En caso de necesitar un valor mayor de 32767 se debería emplear **mt_rand()** en lugar de **rand()**.

Operaciones lógicas

- ☐ Las operaciones lógicas son expresiones matemáticas cuyo resultado es un **valor booleano** (verdadero o falso, en PHP, true o false). Estas expresiones se utilizan principalmente en las **estructuras de control**.

Comparaciones

Ejemplo	Nombre	Resultado
expresión_1 == expresión_2	Igual	true si expresión_1 es igual a expresión_2.
expresión_1 === expresión_2	Idéntico	true si expresión_1 es igual a expresión_2, y son del mismo tipo. (a partir de PHP 4)
expresión_1 != expresión_2	Diferente	
expresión_1 <> expresión_2		
expresión_1 !== expresión_2	No idénticos	true si expresión_1 no es igual a expresión_2, o si no son del mismo tipo. (a partir de PHP 4)

expresión_1 < expresión_2	Menor que	true si expresión_1 es estrictamente menor que expresión_2.
expresión_1 > expresión_2	Mayor que	true si expresión_1 es estrictamente mayor que expresión_2.
expresión_1 <= expresión_2	Menor o igual que	true si expresión_1 es menor o igual que expresión_2.
expresión_1 >= expresión_2	Mayor o igual que	true si expresión_1 es mayor o igual que expresión_2.

- ☐ En una comparación **las variables pueden aparecer a la derecha de la comparación**.
- ☐ Confundir el operador comparación (==) con el operador de asignación (=), produce resultados inesperados.
- ☐ No se pueden concatenar comparaciones. En estos casos habría que concatenar comparaciones mediante operadores lógicos
- ☐ Hay que tener cuidado al comparar expresiones de tipos diferentes, ya que se pueden producir resultados inesperados.
- ☐ PHP permite comparar expresiones de tipos distintos, y la comparación es correcta siempre que sea posible interpretar un dato de un tipo como un dato de otro tipo.
 - ☐ Podemos comparar números enteros y decimales:

- ☐ Un número escrito como número (sin comillas) o como texto (con comillas) son iguales para PHP ...:
- ☐ pero PHP no efectúa cálculos en las cadenas:
- ☐ Además, hay situaciones especiales. Por ejemplo, cuando una cadena empieza por un número, PHP interpreta la cadena como número y descarta el resto
- ☐ Para evitar problemas con las conversiones automáticas de tipo se pueden utilizar los operadores `===` o `!==`

Operadores lógicos

Ejemplo	Nombre	Resultado
expresión_1 && expresión_2	Y	true si los dos, expresión_1 y expresión_2, son true.
expresión_1 and expresión_2		
expresión_1 expresión_2	O	true si uno de los dos, expresión_1 o expresión_2, es true.
expresión_1 or expresión_2		
expresión_1 xor expresión_2	O exclusivo (Xor)	true si sólo uno de los dos, expresión_1 o expresión_2, es true, pero no ambos.
!expresión	Negación	true si expresión no es true.

- ☐ Al escribir expresiones en las que se combinan varias comparaciones mediante operadores lógicos es conveniente utilizar paréntesis,

Diferencia entre and y && y entre or y ||

- ☐ **&& y ||** tienen **mayor prioridad** que **and y or**. Como además el operador de **asignación** = tiene una **prioridad intermedia**, se pueden producir situaciones inesperadas. **Para evitarlas, se recomienda usar paréntesis.**

<?php

```

$var1 = true;
$var2 = false;
$union = $var1 && $var2;
if ($union) {
    echo "<p>verdadero</p>";
} else {
    echo "<p>falso</p>";
}

```

?>

<p>falso</p>

```

<?php
    $var1 = true;
    $var2 = false;
    ($union = $var1) and $var2;
    if ($union) {
        echo "<p>verdadero</p>";
    } else {
        echo "<p>falso</p>";
    }
?>
<p>verdadero</p>

```

Estructuras de Control

- ☐ Un programa PHP se ejecuta en principio de forma secuencial, desde la primera instrucción hasta la última y de una en una. Las estructuras de control permiten modificar este flujo, eligiendo entre instrucciones alternativas o repitiendo instrucciones.

Bloques de instrucciones

- ☐ Para indicar a PHP que varias instrucciones forman un bloque de instrucciones se utilizan las llaves { y }
- ☐ Se recomienda sangrar las instrucciones que forman el bloque y escribir las llaves en líneas distintas de las instrucciones del bloque.
- ☐ No es necesario escribir un punto y coma (;) después de las llaves y, en general, se recomienda no hacerlo.
- ☐ No se suelen utilizar bloques cuando no hay estructuras de control. Si se quiere indicar visualmente al programador que unas instrucciones están relacionadas entre sí, basta con dejar una línea en blanco en el código ... o añadir líneas de comentarios:

Sentencia condicional if ... elseif ... else ...

- ☐ Las construcciones if, else y elseif permiten condicionar la ejecución de un bloque de sentencias al cumplimiento de una condición.
- ☐ La ejecución de esta construcción es la siguiente:
 - ☐ La condición se evalúa siempre.
 - ☐ Si el resultado es true se ejecuta el bloque de sentencias
 - ☐ Si el resultado es false no se ejecuta el bloque de sentencias.

Sentencia condicional if ... else ...

- ☐ La ejecución de esta construcción es la siguiente:
- ☐ La condición se evalúa siempre.

- ☐ Si el resultado es true se ejecuta solamente el bloque de sentencias 1
- ☐ Si el resultado es false se ejecuta solamente el bloque de sentencias 2.

Notación abreviada .. ? ... : ...

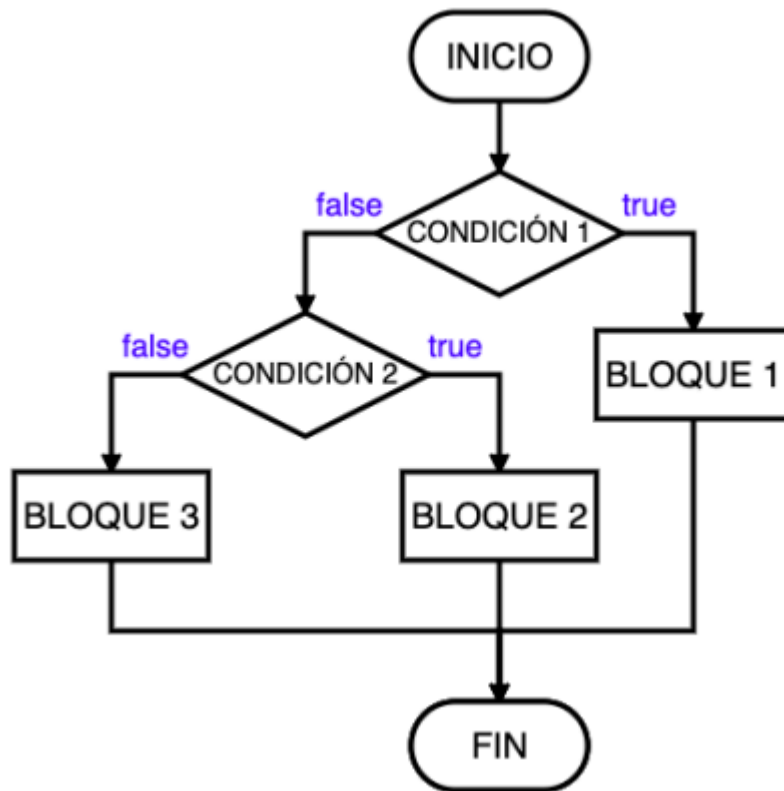
- ☐ Si una sentencia condicional if ... else ... sigue el patrón siguiente:

```
if (condicion_1) {
    $variable = expresion_1;
} else
    $variable = expresion_2;
}
```
- ☐ Entonces se puede sustituir por la construcción más compacta:
\$variable = (condicion_1) ? expresion_1 : expresion_2;
- ☐ que se puede escribir en varias líneas para facilitar la legibilidad:

```
$variable = (condicion_1)
    ? expresion_1
    : expresion_2;
```

Sentencia condicional if ... elseif ... else ..

- ☐ La ejecución de esta construcción es la siguiente:
 - ☐ La condición 1 se evalúa siempre.
 - ☐ Si el resultado es true se ejecuta solamente el bloque_de_sentencias_1
 - ☐ Si el resultado es false se evalúa la condición 2.
 - ☐ Si el resultado es true se ejecuta solamente el bloque_de_sentencias_2
 - ☐ Si el resultado es false se ejecuta solamente el bloque_de_sentencias_3



Sintaxis alternativa sentencia condicional if ... elseif ... else ...

```

<html>
<body>
    <?php if (condición): ?>
        // Código que se ejecuta cuando la condición es cierta
    <?php else: ?>
        // Código que se ejecuta cuando la condición no es cierta
    <?php endif; ?>
</body>
</html>

```

```

<html>
<body>
    <?php if (condición1): ?>
        // Código que se ejecuta cuando la condición 1 es cierta
    <?php elseif (condición2): ?>
        // Código que se ejecuta cuando la condición 2 es cierta
    <?php else: ?>
        // // Código que se ejecuta cuando ninguna condición es
        // cierta
    <?php endif; ?>
</body>
</html>

```

Sentencia condicional switch

- ☐ La sentencia switch es equivalente a una construcción if ... elseif ... en las que las expresiones son comparaciones de igualdad de la misma condición con valores distintos.
- ☐ La sentencia switch ejecuta línea por línea (en realidad, sentencia por sentencia). Al principio, ningún código es ejecutado.
 - ☐ Solo cuando se encuentra una sentencia case cuya expresión se evalúa a un valor que coincida con el valor de la expresión switch, PHP comienza a ejecutar las sentencias. PHP continúa ejecutando las sentencias hasta el final del bloque switch, o hasta la primera vez que vea una sentencia break. Si no se escribe una sentencia break al final de la lista de sentencias de un case, PHP seguirá ejecutando las sentencias. **Es importante no olvidar las sentencias break**
- ☐ Un caso especial es el default. Este caso coincide con cualquier cosa que no se haya correspondido por los otros casos. P

Sintaxis alternativa sentencia condicional switch

- ☐ Advertencia: Cualquier salida (incluyendo espacios en blanco) entre una sentencia switch y el primer case resultará en un error de sintaxis.

Bucle for

```
for (asignacion_inicial; condicion_continuacion; cambio_variable)
{
    bloque_de_sentencias
}
```

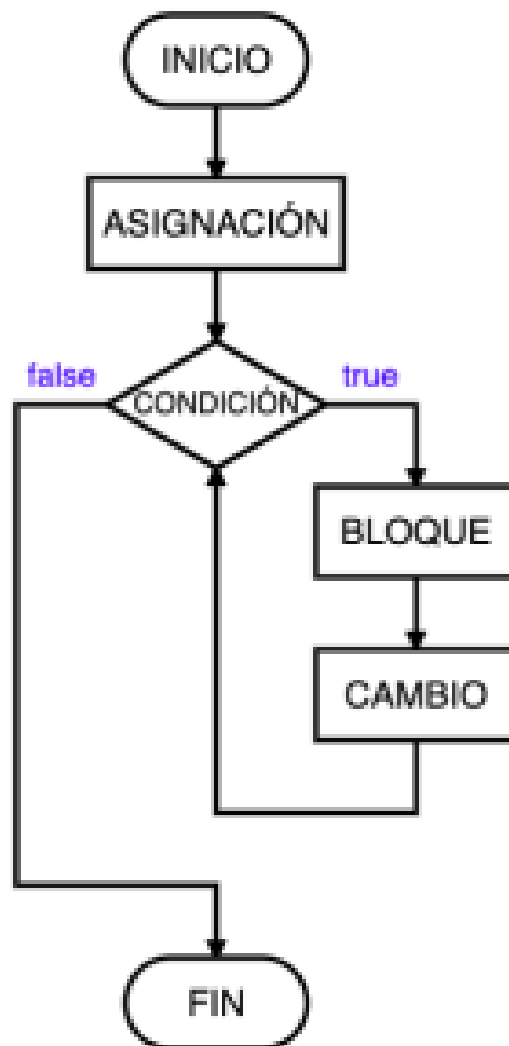
- ☐ La ejecución de esta estructura de control es la siguiente:
 - ☐ Se establece el valor inicial de la variable de control definida en la asignación inicial.
 - ☐ Evalúa la condición de continuación:
 - ☐ si el resultado es true se ejecuta el bloque de sentencias, se efectúa el cambio de la variable de control y se evalúa nuevamente la condición de continuación;
 - ☐ si el resultado es false el bucle se termina
- ☐ Cuando se programa un bucle for hay que tener cuidado en que la condición de continuación vaya a dejar de cumplirse en algún momento, porque si no es así, el bucle no terminaría nunca.
- ☐ Los bucles no tienen por qué ir contando de uno en uno, la expresión de cambio de variable puede ser cualquiera:
- ☐ La variable también puede tomar valores descendientes:
- ☐ Nada impide que la variable de control del bucle se modifique en el cuerpo del bucle, pero eso puede afectar al número de veces que se ejecuta el bucle:

- ☐ La forma más utilizada en programación y que se aconseja acostumbrarse a utilizarla es la que se representa a continuación. Sus ventajas son que empieza a contar en 0, como empiezan de forma predeterminada los arrays de índices numéricos, y que el número de iteraciones aparece en la condición de continuación.

```
<?php
```

```
print "<p>Comienzo</p>";  
for ($i = 0; $i < 5; $i++) {  
    echo "<p>Hola</p>";  
}
```

```
?>
```



Ejemplos de bucles incorrectos

Sintaxis alternativa bucle for

```
for (expr1; expr2; expr3):  
    sentencia  
    ...  
endfor;
```

Contador

- ☐ Se entiende por contador una variable que lleva la cuenta del número de veces que se ha cumplido una condición

Acumulador

- ☐ Se entiende por acumulador una variable que acumula el resultado de una operación
 - ☐ El acumulador se modifica en cada iteración del bucle (en este caso, el valor de la variable de control \$i se añade al acumulador \$suma).
 - ☐ Antes del bucle se debe dar un valor inicial al acumulador (en este caso, 0)

Bucles anidados

- ☐ Un bucle anidado es un bucle que se encuentra incluido en el bloque de sentencias de otro bloque.
- ☐ Los bucles pueden tener cualquier nivel de anidamiento (un bucle dentro de otro bucle dentro de un tercero, etc.).
- ☐ Al bucle que se encuentra dentro del otro se le puede denominar bucle interior o bucle interno. El otro bucle sería el bucle exterior o bucle externo.
- ☐ En los bucles anidados es importante utilizar variables de control distintas, para no obtener resultados inesperados.

Bucles anidados con variables independientes

- ☐ Los bucles anidados con variables independientes son los bucles en los que ninguna de las variables de uno de los bucles interviene ni en la condición de continuación ni en el cambio de variable (expresión de paso) de los otros bucles.
- ☐ En estos casos, si el bucle exterior se ejecuta M veces y el bucle interior se ejecuta N veces cada vez que se ejecuta el bucle exterior, en total el bloque de instrucciones se ejecutará M x N veces.

Bucles anidados con variables dependientes

- ☐ Los bucles anidados con variables dependientes son los bucles en los que la variable de uno de los bucles interviene en la condición de continuación o en la expresión de paso de los otros bucles.
- ☐ En estos casos, es difícil decir a priori cuántas veces se ejecutará el bloque de instrucciones.

Bucle while

```
while (condicion) {  
    bloque_de_sentencias  
}
```

- ☐ La condición se evalúa al principio de cada iteración: si el resultado es true se ejecuta el bloque de sentencias; si el resultado es false el bucle se termina
- ☐ Cuando se programa un bucle while hay que tener cuidado en que la condición deje de cumplirse en algún momento, porque si no es así, el bucle no terminaría nunca.

Sintaxis alternativa bucle while

```
while (expr):  
    sentencias  
    ...  
endwhile;
```

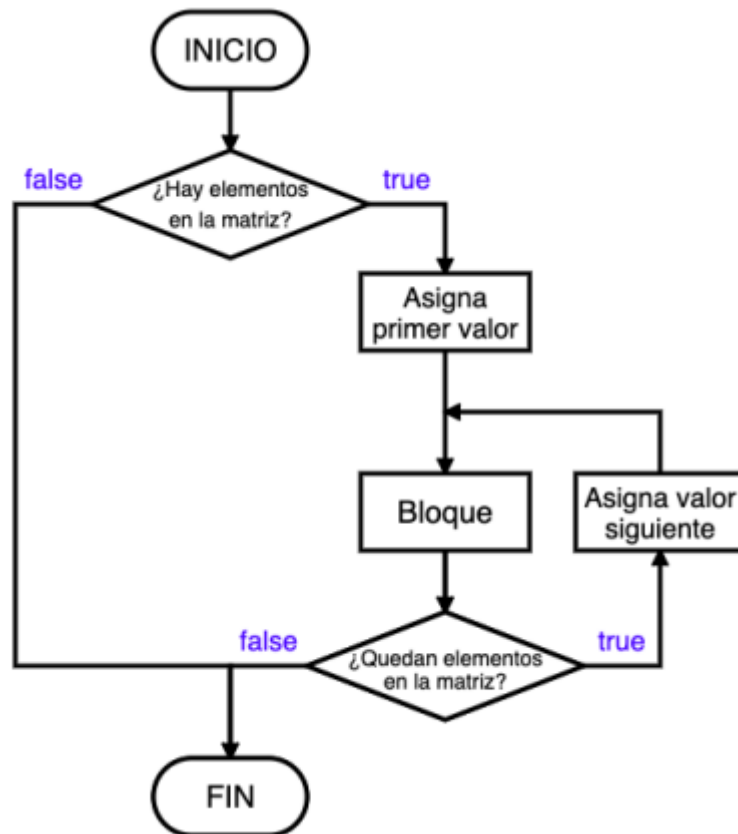
Bucle do ... while

```
do {  
    bloque_de_sentencias  
} while (condicion)
```

- ☐ La condición se evalúa al final de cada iteración: si el resultado es true se ejecuta el bloque de sentencias; si el resultado es false el bucle se termina.
- ☐ El bucle do ... while es muy similar al bucle while, la principal diferencia es que en el bucle **do ... while** el bloque de sentencias se ejecuta **por lo menos una vez mientras** que en el bucle while depende de si la condición es cierta o no la primera vez que se evalúa.
- ☐ En los bucles do ... while más sencillos, antes del bucle se inicializa una variable que se evalúa en la condición y dentro del bucle se modifica la variable, como muestra el ejemplo siguiente

Bucle foreach

- ☐ El bucle foreach es un tipo especial de bucle que permite recorrer estructuras que contienen varios elementos (como matrices, recursos u objetos) sin necesidad de preocuparse por el número de elementos.



```
foreach ($matriz as $valor) {  
    bloque_de_sentencias  
}
```

```
foreach (matriz as $indice => $valor) {  
    bloque_de_sentencias  
}
```

- ☐ La ejecución de esta estructura de control es la siguiente:
 - ☐ Si el array no contiene elementos, el bucle no se ejecuta.
 - ☐ Si el array contiene elementos:
 - ☐ Se asigna el primer valor del array a la variable auxiliar (y en su caso, el primer índice a la otra variable auxiliar)
 - ☐ Se ejecuta el bloque de sentencias
 - ☐ Si el array no contiene más elementos, el bucle deja de ejecutarse.
 - ☐ Si el array todavía contiene más elementos:

- ☐ Se asigna el siguiente valor del array a la variable auxiliar (y en su caso, el siguiente índice a la otra variable auxiliar)
- ☐ Se ejecuta de nuevo el bloque de sentencias.
- ☐ El bucle se ejecuta tantas veces como elementos tiene el array. En cada iteración, las variables \$índice y \$valor van tomando los valores de los índices y del array para ese índice.
- ☐ Si sólo se necesitan los valores almacenados en el array se puede utilizar tanto la primera como la segunda forma. Si se necesitan tanto los índices como los valores se debe utilizar la segunda forma. Si sólo se necesitan los índices también se debe utilizar la segunda forma.

Sintaxis alternativa bucle foreach

```
foreach(expr):
    sentencias
...
endforeach;
```

Funciones de matrices/arrays

Comparar arrays

array_diff

- ☐ **array_diff (array \$array1, array \$array2 [, array \$...])**: array Compara \$array1 con uno o más arrays y devuelve los valores de \$array1 que no estén presentes en cualquiera de los demás arrays. **Se observa que lo que compara son los valores.**

array_diff_assoc

- ☐ **array_diff_assoc (array \$array1, array \$array2 [, array \$...])**: array Compara \$array1 con \$array2 y devuelve la diferencia, pero esta vez **comparando valores y claves.**

array_diff_key

- ☐ **array_diff_key (array \$array1, array \$array2 [, array \$...])**: array Compara las claves de \$array1 con las claves de \$array2 y devuelve la diferencia. Es como la función array_diff(), pero **sólo comparando las claves en lugar de los valores.**

Unir arrays

Operador suma

- ☐ Con el operador de suma +, que ya se ha comentado previamente, se eliminan los elementos duplicados que aparecen en el array a la derecha del operador.

array_merge

- ☐ **array_merge (array \$array1 [, array \$...]): array** Combina dos o más arrays, de modo que los **valores de uno se unen al final del anterior**.
- ☐ Al contrario que con el operador suma, **si los arrays de entrada tienen las mismas claves de tipo string**, el último valor para **esa clave sobrescribirá al anterior**.
- ☐ Pero **si son arrays numéricos**, el **último valor** no sobrescribirá al valor original, sino que **se añadirá al final**.

array_merge_recursive

- ☐ **array_merge_recursive (array \$array1 [, array \$...]): array** Es igual que array_merge(), salvo que **cuando los arrays de entrada tienen las mismas claves de tipo string, los valores de estas claves se unen en un array de forma recursiva**, de forma que si uno de los valores es un array, la función unirá también ésta con la correspondiente entrada de otro array.
- ☐ Si los arrays tienen la **misma clave numérica**, el **valor posterior no sobrescribirá el valor original**, sino que **se añadirá al final**.

array_combine

- ☐ **array_combine (array \$keys, array \$values): array** Crea un array utilizando un array para sus claves y otro array para sus valores. Coge los valores para crear el nuevo array (clave => valor) de los arrays de origen.

Rellenar arrays

array_fill

- ☐ **array_fill (int \$start_index, int \$num, mixed \$value): array** Llena un array \$num veces (debe ser mayor que cero) con el valor \$value desde la clave \$startindex.
- ☐ Si **\$startindex es negativo**, el primer valor tendrá como clave ese valor negativo y los demás comenzarán desde cero.

range

- ☐ **range (mixed \$start, mixed \$end [, number \$step = 1]): array** Esta función genera un array de valores en sucesión aritmética, es decir los valores desde \$start hasta \$end contando de paso en paso \$step (sin superar el valor \$final).
- ☐ **En el caso de usar letras** está limitada a **emplear sólo un carácter**, si se emplea más de uno, sólo tiene en cuenta el primero.

Dividir arrays

array_slice

- ☐ **array_slice** (array \$array, int \$offset [, int \$length = null [, bool \$preserve_keys = false]]): array Extrae una parte de un array, que comienza en \$offset (si es negativo comienza por el final) y con una longitud de \$length.
- ☐ Si \$length es positivo, la secuencia tendrá **tantos elementos como indique el valor**. Si es mayor que el propio array, se devolverán los elementos disponibles en el array.
- ☐ Si \$length es negativo, la secuencia terminará en tantos elementos comenzando por el final del array.
- ☐ Si se omite, contendrá **todos los elementos del array desde \$offset**.
- ☐ El parámetro opcional \$preserve_keys reiniciará los índices numéricos de forma **predeterminada**. Se pueden preservar cambiándolo a true.

array_splice

- ☐ **array_splice** (array &\$input, int \$offset [, int \$length [, mixed \$replacement = array()]]): array Elimina una porción del array y la reemplaza por algo. Comienza la eliminación desde \$offset (si es negativo, comienza por el final).
- ☐ Si se omite \$length, se **elimina todo desde \$offset** hasta el final del array.
- ☐ Si \$length es positivo, se eliminan **tantos elementos como indique su longitud**.
- ☐ Si \$length es negativo, el final de la porción eliminada será de **tantos elementos como indique la longitud desde el final del array**.
- ☐ Si \$length es cero no se elimina ningún elemento.
- ☐ Si se especifica el array \$replacement, los elementos eliminados se reemplazarán por los elementos de este array.
- ☐ Para **eliminar todo desde \$offset** hasta el final de array **cuando se ha especificado \$replacement**, se puede utilizar count(\$input) para \$length.

array_chunk

- ☐ **array_chunk** (array \$array, int \$size [, bool \$preserve_keys = false]): array Divide un array en fragmentos de tamaño \$size. **El último fragmento puede contener menos elementos que size**.
- ☐ Devuelve un array multidimensional indexado numéricamente, empezando desde cero.
- ☐ Si \$preserve_keys es true se mantienen las claves, si no se reindexa numéricamente.

Modificar elementos en arrays

array_shift

- ☐ **array_shift** (array &\$array): mixed Quita el primer valor del \$array y lo devuelve.

- ☐ Los arrays asociativos no varían sus claves, mientras que los numéricos son reindexados y comienzan de cero.
- ☐ Esta función después de usarse ejecuta reset() en el array.

array_unshift

- ☐ **array_unshift (array &\$array, mixed \$value1 [, mixed \$...]): int** Añade los valores *\$value al inicio del array*. Igual que en array_shift(), en los arrays asociativos no varían sus claves, mientras que los numéricos se reindexan.

array_pop

- ☐ **array_pop (array &\$array): mixed** Quita el último valor del array y lo devuelve.
- ☐ Esta función después de usarse ejecuta reset() en el array.

array_push

- ☐ **array_push (array &\$array, mixed \$value1 [, mixed \$...]): int** Inserta uno o más elementos al final de un array. **Viene a ser lo mismo que insertar un elemento a un array \$array[] = valor, pero se pueden insertar varios de vez**. Si se inserta sólo un valor, es mejor utilizar la forma tradicional.

array_flip

- ☐ **array_flip (array \$array): array** Intercambia las claves de un array con sus valores.
- ☐ Los valores del array tienen que ser válidos (de tipo integer o string), sino genera un warning y los elementos en cuestión no se incluirán en el resultado.

Contar elementos de un array

- ☐ La función **count(\$array)** permite contar los elementos de un array.
- ☐ **int count(mixed \$array_or_countable, int \$mode = COUNT_NORMAL)**
- ☐ **En un array multidimensional**, la función **count(\$array)** considera el array como un vector de vectores y **devuelve simplemente el número de elementos del primer nivel**:
- ☐ **Para contar todos los elementos de un array multidimensional**, habría que utilizar el segundo parámetro de la función, el modo, con el valor **COUNT_RECURSIVE**.
- ☐ Si quisiéramos contar únicamente los elementos de un array bidimensional habría que restar los dos resultados anteriores:

Contar cuántas veces aparece cada valor en un array

- ☐ **array_count_values(\$array)** cuenta cuántos valores hay de cada valor en un array.
- ☐ **array_count_values(array \$array): array** El array devuelto tiene como índices los valores del array original y como valores la cantidad de veces que aparece el valor.

Máximo y mínimo

- ☐ La función **max(\$array, ...)** devuelve el valor máximo de un array (o varias).
- ☐ La función **min(\$array, ...)** devuelve el valor mínimo de un array(o varias).
 - ☐ max(array \$values): mixed
 - ☐ min(array \$values): mixed
- ☐ Los valores no numéricos se tratan como 0, pero si 0 es el mínimo o el máximo, la función devuelve la cadena.

Ordenar un array

- ☐ Cuando los **índices** del array que vamos a ordenar **no son importantes** y se pueden modificar, podemos utilizar las funciones **sort(\$array, \$opciones)** y **rsort(\$array, \$opciones)**, que ordenan atendiendo únicamente a los valores del array (no a sus índices), en orden creciente o decreciente, y reindexan el array:
 - ☐ **sort(array &\$array, int \$sort_flags = SORT_REGULAR): bool**: ordena por orden alfabético / numérico de los valores y genera nuevos índices numéricos consecutivos a partir de cero:
 - ☐ **rsort(array &\$array, int \$sort_flags = SORT_REGULAR): bool**: ordena por orden alfabético / numérico inverso de los valores y genera nuevos índices numéricos consecutivos a partir de cero
- ☐ Cuando los **índices** del array que vamos a ordenar **son importantes** y no se deben modificar, podemos **ordenar atendiendo únicamente a los valores del array u ordenar atendiendo únicamente a los índices del array** y ordenar en orden creciente o decreciente, por lo que PHP dispone de cuatro funciones:
 - ☐ **asort(array &\$array, int \$sort_flags = SORT_REGULAR): bool**: ordena por orden alfabético / numérico de los valores:
 - ☐ **arsort(array &\$array, int \$sort_flags = SORT_REGULAR): bool** ordena por orden alfabético / numérico inverso de los valores:
 - ☐ **ksort(array &\$array, int \$sort_flags = SORT_REGULAR): bool**: ordena por orden alfabético / numérico de los índices:
 - ☐ **krsort(array &\$array, int \$sort_flags = SORT_REGULAR): bool**: ordena por orden alfabético / numérico de los índices

Array inicial	sort()	rsort()	asort()	arsort()	ksort()	krsort()
Array ([5] => cinco [1] => uno [9] => nueve)	Array ([0] => cinco [1] => nueve [2] => uno)	Array ([0] => uno [1] => nueve [2] => cinco)	Array ([5] => cinco [9] => nueve [1] => uno)	Array ([1] => uno [9] => nueve [5] => cinco)	Array ([1] => uno [5] => cinco [9] => nueve)	Array ([9] => nueve [5] => cinco [1] => uno)

Buscar un valor en un array

- ☐ La función booleana **in_array(mixed \$needle, array \$haystack, bool \$strict = false)**: **bool** devuelve true si el valor se encuentra en el array. Si el argumento booleano \$tipo es true, in_array() comprueba además que los tipos coincidan.
- ☐ La función **array_search(mixed \$needle, array \$haystack, bool \$strict = false)**: **mixed** busca el valor en el array y, si lo encuentra, devuelve el índice correspondiente, pero si hay varios valores coincidentes sólo devuelve el primero que encuentra.
- ☐ La función **array_keys(\$array[, \$valor[, \$tipo]]**: **array** busca el valor en el array y, si lo encuentra, devuelve un array cuyos valores son los índices de todos los elementos coincidentes

Reindexar un array

- ☐ La función **array_values(array \$array)**: **array** devuelve los valores de un array en el mismo orden que en el array original, pero renumerando los índices desde cero:

Barajar los elementos de un array

- ☐ La función **shuffle(array &\$array)**: **bool** baraja los valores de un array. Los índices del array original se pierden, ya que se renumeran desde cero.

Extraer al azar un elemento de un array

- ☐ La función **array_rand(array \$array, int \$num = 1)**: **mixed** extrae al azar uno de los índices del array.
- ☐ Una vez obtenido el índice se puede obtener el valor correspondiente de la matriz.

Eliminar valores repetidos

- ☐ La función **array_unique(array \$array, int \$sort_flags = SORT_STRING)**: **array** devuelve un array en el que se han eliminado los valores repetidos.

Funciones

Cuando utilizamos en **múltiples ocasiones un mismo fragmento** de código debemos usar **funciones (functions)**: herramienta para encapsular y ejecutar un mismo código.

- VENTAJAS ayuda a que los ficheros tengan un menor tamaño y sean más fáciles de mantener.
- Deben definirse antes de utilizarlas (aunque no sea en el mismo fragmento de código php).
- Las funciones **pueden no tener argumento o tener argumentos por valor o por referencia.**
- Los **nombres de las funciones** siguen las mismas reglas de los identificadores de PHP, es decir, deben **comenzar por una letra o un guión bajo (_) y el resto de**

caracteres pueden ser letras, números o guiones bajos (se pueden utilizar caracteres no ingleses como acentos, eñes, etc)

- Los **nombres** de funciones **no distinguen entre mayúsculas o minúsculas**.
- Se recomienda que los nombres de las funciones sigan el estilo camelCase (es decir, sin espacios ni guiones, con la primera palabra en minúsculas y el resto con la primera letra en mayúsculas).

La sintaxis clásica de la función es:

```
<?php
```

```
// Declarar
```

```
function nombreFuncion($argumento1,$argumento2, ...) {  
    instrucciones de la función;  
    return ...;  
}
```

```
// Llamar
```

```
nombreFuncion($parametros);
```

```
?>
```

La palabra **function** es una palabra reservada:

- **nombreFuncion:** Es el nombre que daremos a la función, podemos escribir aquí cualquier palabra, y este será el nombre que tenga la función.
- **()**: Escribimos luego los paréntesis, y dentro de ellos los argumentos, en caso de que sean necesarios.
 - **\$argumentos:** Son los datos necesarios para poder ejecutar la función, por
 - Una función puede necesitar o no argumentos, por lo que no es obligatorio ponerlos si no es necesario. Si necesita más de un argumento, estos se pondrán separados por comas.
 - El argumento puede ser cualquier variable, la cual la utilizaremos dentro de la función para operar con ella y obtener el resultado.
- **{Instrucciones}**: Entre llaves pondremos las instrucciones que haga falta para ejecutar la función.
- **return** : la palabra clave return va seguido del resultado que devuelve la función.**se asocia automáticamente un tipo de datos a la variable, dependiendo de su valor**

La sintaxis moderna de la función es:

```
<?php
```

```
// Declarar
```

```
function nombreFuncion(tipo_de_parametro $parametros):  
    tipo_return  
{  
    instrucciones de la función  
    return ...;  
}
```

```
// Llamar
```

`nombreFuncion($parametros);`
?>

- Las partes que componen una función son las siguientes:
 - Las 4 primeras líneas es el formato de comentarios. Una función, **por mucha prisa que tengas, debe estar comentada** con el formato del ejemplo.
 - La palabra **function** es una palabra reservada.
 - **A continuación** de la palabra reservada debe de ir el **nombre de la función**.
 - **Después unos paréntesis con los argumentos**. Si no tiene, se dejan vacíos pero siempre tienen que estar presentes.
 - Luego **dos puntos**. Por último el **tipo de valor resultante**.
 - Llaves para envolver el código **{}**.
 - Y dentro del código la **palabra reservada return seguido del valor a devolver**.
- La palabra **return** establece el **valor de respuesta de una función a cualquier variable, y después PHP sale de la función**. Si se usa sólo return, se sale de la función sin devolver ningún valor. **Si se omite return, el valor devuelto será null**.
- **Se recomienda su uso siempre que sea posible**
- En la versión PHP 8 es posible indicar 2 tipos diferentes de datos en el envío de return, incluso tenemos la opción de utilizar mixed, que es equivalente a union type (object|resource|array|string|int|float|bool|null).

Funciones con argumentos

- Se pueden pasar argumentos por valor, por referencia y por valor predeterminado.

Pasar argumentos por valor

- Por defecto los argumentos de las funciones se pasan por valor, **si el valor del argumento dentro de la función cambia, éste no cambia fuera de la función**.
- A la hora de pasar los argumentos, PHP arreglará las incompatibilidades de tipos de forma automática

```
<?php
/**
 * Incrementa uno un número
 * @param {int} $num - Número a incrementar
 * @return {int}
 */
function incrementar(int $num): int
{
    return $num + 1;
}
echo "<p>". incrementar(4.5) . "</p>";
?>
<p>5</p>
```

Pasar argumentos por referencia

- **Para permitir a una función que cambie un argumento**, se tiene que pasar por referencia, y se consigue anteponiendo el símbolo ampersand & en el argumento:

```
<?php
/**
 * Incrementa uno un número
 * @param {array} $animales - Lista de animales
 */
function cambiarAnimales(array &$animales): void
{
    $animales = ["mono", "gorila", "léon"];
}
$array = ["perro", "gato", "avestruz"];
cambiarAnimales($array);
foreach($array as $animal){
    echo "<p>$animal</p>";
}
?>
<p>mono</p>
<p>gorila</p>
<p>léon</p>
```

Valores de argumentos predeterminados

- En PHP se pueden definir funciones con argumentos predeterminados, de manera que **si en la llamada a la función no se envía un argumento, la función asume un valor predeterminado**. Lógicamente, los argumentos predeterminados deben ser los últimos en la lista de argumentos, para evitar ambigüedades.
- Los argumentos predeterminados **se establecen en la definición de la función**, igualando el nombre del argumento a su valor predeterminado.

```
<?php
/**
 * Saluda a una persona
 * @param {string} $nombre - Nombre
 * @return {string}
 */
function saludar(string $nombre = 'Anónimo'): string
{
    return "<p>Hola, persona llamada " . $nombre . ". Por lo que
    veo tu nombre mide " . strlen($nombre) . " caracteres.</p>";
}
echo saludar();
echo saludar("Guillermo");
?>
```

- También se pueden poner arrays y de tipo null como valores predeterminados:

Tipos admitidos

- Las declaraciones de tipos permiten a las funciones exigir que los parámetros proporcionados sean de un tipo determinado.
- Para especificar una declaración de tipo se antepone el nombre del tipo al nombre del parámetro. Se puede hacer que la declaración acepte valores null si el valor predeterminado del parámetro se establece a null.

Funciones variables

- Las funciones variables son llamadas cuando se añade un paréntesis a una variable y se ejecuta su valor.
- Si se llama a una variable que tiene paréntesis anexos a ella, PHP buscará una función con el valor que tiene dicha variable, e intentará ejecutarla.
- Las funciones variables **no funcionan con constructores del lenguaje como echo, print, unset(), isset(), empty(), include, require, etc.**

Ámbito de las variables

- El ámbito de las variables (o scope) divide un script PHP en entornos, lo que afecta a la definición y uso de las variables:
 - **Las variables que se declaran fuera de funciones o clases** se encuentran en el ámbito global (global scope), disponibles en cualquier parte del script.
 - Las funciones son bloques independientes, **las variables que se definen dentro de una función tienen ámbito local (local scope) y no afectan a otras en el script global**, de la misma forma que las variables globales **no están disponibles dentro de las funciones:**

```
<?php
/**
 * Función saludo con educación
 */
function saludar(): void
{
    $saludo = "Hola";
    echo $saludo;
    return;
}
$saludo = "Buenos días";
saludar(); // Devuelve: Hola
echo $saludo; // Devuelve: Buenos días
?>
```

- Si vas a usar variables que están presentes en el código, puedes enriquecer el contenido de la función usando use

```
<?php
$tienda = "frutería";
function () use ($tienda) {
    return "Estoy en la $tienda";
}
```

```
}  
?>
```

- Es posible emplear también una variable global dentro de una función con la palabra global o con el array \$GLOBALS:
 - Si se asigna una referencia a una variable declarada como global dentro de una función, la referencia se verá sólo dentro de la función. **Si se quiere que el resultado sea visible en el ámbito global también se utiliza la matriz \$GLOBALS:**
- Cuando se hace global a una variable, es lo mismo que referenciar el nombre de la misma al elemento GLOBALS en cuestión:
- La diferencia entre utilizar la palabra global y el array \$GLOBALS se produce cuando se hacen referencias dentro de las funciones:

```
<?php  
$x = "Soy X";  
$y = "Soy Y";  
/**  
 * Función devuelve una cadena  
 */  
function funcionUno():  
{  
    global $x, $y;  
    $y = &$amp;x;  
}  
/**  
 * Función devuelve una cadena  
 */  
function funcionDos(){  
    global $x;  
    $GLOBALS['y'] = &$amp;x;  
}  
funcionUno();  
echo "<p>" . $y . "</p>"; // Devuelve: Soy Y  
funcionDos();  
echo "<p>" . $y . "</p>"; // Devuelve: Soy X  
?>  
  
<p>Soy Y</p>  
  
<p>Soy X</p>
```

En funcionUno() se han importado \$x e \$y con la palabra global y se ha referenciado \$y a \$x, pero esto ha ocurrido sólo dentro de la función. En funcionDos() se ha empleado \$GLOBALS con \$y y su valor ha permanecido referenciado después de la función.

Anónimas

- Las funciones anónimas son funciones sin nombre a las que se accede mediante variables o mediante otras funciones

```
<?php

$numeros = [10, 20, 30, 40];

$numerosIncrementados = array_map(function ($numero) {

    return $numero + 1;

}, $numeros);

echo "<pre>";

var_dump($numerosIncrementados);

ech"</pre>";

?>

Array(4) {

    [0]=>

    int(11)

    [1]=>

    int(21)

    [2]=>

    int(31)

    [3]=>

    int(41)

}
```

Bibliotecas

- Las bibliotecas son archivos php que se pueden incluir en cualquier otro archivo php. Las bibliotecas se suelen utilizar para centralizar fragmentos de código que se utilizan en varias páginas. De esa manera, si se quiere hacer alguna modificación, no es necesario hacer el cambio en todas las páginas sino únicamente en la biblioteca.

biblioteca.php

```
<?php
/**
 * Función que imprime
 * @param {string} $titulos - Título
 */
function cabecera(string $titulo)
{
    echo "<!DOCTYPE html>";
    echo "<html lang=\"es\">";
    echo "<head>";
    echo " <meta charset=\"utf-8\">";
    echo " <title>$titulo</title>";
    echo " <meta name=\"viewport\" content=\"width=device-width,
initial-scale=1.0\">";
    echo " <link rel=\"stylesheet\" href=\"estilo.css\"
title=\"Color\">";
    echo "</head>";
    echo "<body>";
    echo " <h1>$titulo</h1>";
}
?>
```

pagina_1.php

```
<?php
include "biblioteca.php";
cabecera("Página de ejemplo");
echo "<p>Esta página es válida</p>";
?>

</body>
</html>
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="utf-8">
<title>Página de ejemplo</title>
<meta name="viewport" content="width=device-width,
initial-scale=1.0">
<link rel="stylesheet" href="estilo.css" title="Color">
</head>
<body>
<h1>Página de ejemplo</h1>
<p>Esta página es válida</p>
</body>
</html>
```

- Se pueden crear todas las bibliotecas que se necesiten e incluir cualquier número de bibliotecas en cualquier punto de un programa. Las bibliotecas pueden a su vez incluir otras bibliotecas.
- Normalmente, las bibliotecas **suelen contener funciones, definiciones de constantes o inicialización de variables**, pero en realidad pueden incluir cualquier tipo de código php, que se ejecutará en la posición en la que se incluya la biblioteca. **También se puede incluir parte de la lógica del programa** (generar salida, modificar variables, etc.), pero **se recomienda que no se hagan las dos cosas en un mismo fichero.**
- Existe una construcción **similar a include**, la construcción **require**. La diferencia con respecto a include es que **require produce un error si no se encuentra el archivo** (y no se procesa el resto de la página), mientras que **include sólo produce un aviso** (y se procesa el resto de la página).
- En un **mismo archivo php** se pueden incluir **varias construcciones include o require**, pero si las bibliotecas incluidas contienen definiciones de funciones, al incluir de nuevo la definición de la función se genera un error.
 - Para que no ocurra este problema se pueden utilizar las funciones **include_once o require_once**, que también incluyen los ficheros pero que, en caso de que los ficheros ya se hayan incluido, entonces no los incluyen.
- Las bibliotecas están habitualmente en el mismo servidor que los programas, pero **podrían estar en otros servidores y acceder a ellas por HTTP**, no como ficheros locales.
 - La directiva de configuración **allow_url_include** **permite acceder a bibliotecas por HTTP (suele estar desactivada)**. Las cuatro construcciones (include, require, include_once y require_once) pueden utilizarse escribiendo entre paréntesis el nombre de los ficheros o sin escribir paréntesis.

<https://www.hostinger.es/tutoriales/como-instalar-composer>

Propuesta de Biblioteca Personal: InventoryManager

Dada la implementación actual del ejercicio, se puede estructurar y encapsular en una biblioteca personalizada llamada InventoryManager. Esta biblioteca contendrá clases y métodos organizados para realizar las operaciones relacionadas con la gestión de inventarios, lo que proporcionará un mejor mantenimiento, reutilización y legibilidad del código. A continuación, se detalla cómo se puede construir esta biblioteca:

1. Estructura de la Biblioteca

La biblioteca se dividirá en clases específicas para cada funcionalidad. La estructura de archivos sería similar a:

```
css Copiar código

InventoryManager/
├── src/
│   ├── Inventory.php
│   ├── InventoryOperations.php
│   └── InventoryReport.php
├── tests/
│   └── InventoryTest.php
├── composer.json
└── README.md
```

InventoryManager/

```
├── src/
|   ├── Inventory.php
|   ├── InventoryOperations.php
|   └── InventoryReport.php
├── tests/
|   └── InventoryTest.php
├── composer.json
└── README.md
```

2. Clases y Funcionalidades

Clase Principal: Inventory

- Descripción: Esta clase almacena el inventario y proporciona métodos para acceder y manipularlo.
- Métodos:
 - `__construct(array $items)`: Constructor para inicializar el inventario.
 - `getAllItems()`: Retorna todos los productos del inventario.
 - `addItem(array $item)`: Añade un nuevo producto al inventario.
 - `removeItem(string $productName)`: Elimina un producto por su nombre.

- getItem(string \$productName): Retorna los detalles de un producto específico.
- getCategories(): Devuelve todas las categorías disponibles.

Clase: InventoryOperations

- Descripción: Realiza operaciones como comparación, unificación y eliminación de duplicados.
- Métodos:
 - compareInventories(array \$inventory1, array \$inventory2): Compara dos inventarios y encuentra diferencias.
 - mergeInventories(array ...\$inventories): Une múltiples inventarios.
 - removeDuplicates(array \$inventory): Elimina duplicados de un inventario.
 - countByCategory(array \$inventory): Cuenta el número de productos por categoría.
 - sortByPrice(array \$inventory, string \$order = "ascendente"): Ordena productos por precio.

Clase: InventoryReport

- Descripción: Se encarga de la generación de informes y estadísticas.
- Métodos:
 - generateReport(array \$inventory): Genera un informe detallado del inventario.
 - exportToPDF(array \$inventory, string \$filename): Exporta el informe a PDF.
 - exportToExcel(array \$inventory, string \$filename): Exporta el informe a Excel.
 - divideInventory(array \$inventory, int \$chunkSize): Divide el inventario en secciones.

3. Implementación de las Clases

A continuación, se presenta un ejemplo de la implementación de estas clases:

Archivo: src/Inventory.php

Archivo: src/Inventory.php

```
php Copiar código

<?php

namespace InventoryManager;

class Inventory {
    private $items;

    public function __construct(array $items) {
        $this->items = $items;
    }

    public function getAllItems(): array {
        return $this->items;
    }

    public function addItem(array $item): void {
        $this->items[] = $item;
    }

    public function removeItem(string $productName): void {
        $this->items = array_filter($this->items, function($item) use ($productName) {
            return $item['Producto'] !== $productName;
        });
    }

    public function getItem(string $productName): ?array {
        foreach ($this->items as $item) {
            if (strcasecmp($item['Producto'], $productName) === 0) {
                return $item;
            }
        }
        return null;
    }

    public function getCategories(): array {
        return array_unique(array_column($this->items, 'Categoría'));
    }
}
```

<?php

namespace InventoryManager;


```

class Inventory {

    private $items;


    public function __construct(array $items) {

        $this->items = $items;

    }


    public function getAllItems(): array {

        return $this->items;

    }


    public function addItem(array $item): void {

        $this->items[] = $item;

    }


    public function removeItem(string $productName): void {

        $this->items = array_filter($this->items, function($item) use ($productName) {

            return $item['Producto'] !== $productName;

        });

    }


    public function getItem(string $productName): ?array {

        foreach ($this->items as $item) {

            if (strcasecmp($item['Producto'], $productName) === 0) {

                return $item;

            }

        }

    }

}

```

```
}  
  
return null;  
  
}
```

```
public function getCategories(): array {  
  
    return array_unique(array_column($this->items, 'Categoría'));  
  
}
```

}chivo: src/InventoryOperations.php

```
php
Copiar código

<?php

namespace InventoryManager;

class InventoryOperations {

    public static function compareInventories(array $inventory1, array $inventory2): array {
        $products1 = array_column($inventory1, 'Producto');
        $products2 = array_column($inventory2, 'Producto');
        return array_diff($products1, $products2);
    }

    public static function mergeInventories(array ...$inventories): array {
        return array_merge(...$inventories);
    }

    public static function removeDuplicates(array $inventory): array {
        $uniqueItems = [];
        $products = [];
        foreach ($inventory as $item) {
            if (!in_array($item['Producto'], $products)) {
                $products[] = $item['Producto'];
                $uniqueItems[] = $item;
            }
        }
        return $uniqueItems;
    }

    public static function countByCategory(array $inventory): array {
        return array_count_values(array_column($inventory, 'Categoría'));
    }

    public static function sortByPrice(array $inventory, string $order = "ascendente"): array {
        usort($inventory, function ($a, $b) use ($order) {
            return $order === "ascendente" ? $a['Precio'] <=> $b['Precio'] : $b['Precio'] <=> $a['Precio'];
        });
        return $inventory;
    }
}
```

<?php

namespace InventoryManager;

```

class InventoryOperations {

    public static function compareInventories(array $inventory1, array $inventory2): array {

        $products1 = array_column($inventory1, 'Producto');

        $products2 = array_column($inventory2, 'Producto');

        return array_diff($products1, $products2);

    }

    public static function mergeInventories(array ...$inventories): array {

        return array_merge(...$inventories);

    }

    public static function removeDuplicates(array $inventory): array {

        $uniqueItems = [];

        $products = [];

        foreach ($inventory as $item) {

            if (!in_array($item['Producto'], $products)) {

                $products[] = $item['Producto'];

                $uniqueItems[] = $item;

            }

        }

        return $uniqueItems;

    }

    public static function countByCategory(array $inventory): array {

        return array_count_values(array_column($inventory, 'Categoría'));

    }

}

```

```
}
```

```
public static function sortByPrice(array $inventory, string $order = "ascendente"): array {  
    usort($inventory, function ($a, $b) use ($order) {  
        return $order === "ascendente" ? $a['Precio'] <=> $b['Precio'] : $b['Precio'] <=>  
$a['Precio'];  
    });  
    return $inventory;  
}  
}
```

Archivo: src/InventoryReport.php

Archivo: src/InventoryReport.php

```
php Copiar código

<?php

namespace InventoryManager;

class InventoryReport {

    public static function generateReport(array $inventory): array {
        $report = [];
        foreach ($inventory as $item) {
            $report[$item['Producto']] = [
                'Precio' => $item['Precio'],
                'Categoría' => $item['Categoría'],
                'Cantidad' => $item['Cantidad']
            ];
        }
        return $report;
    }

    public static function divideInventory(array $inventory, int $chunkSize): array {
        return array_chunk($inventory, $chunkSize);
    }

    public static function exportToPDF(array $inventory, string $filename): void {
        // Lógica para exportar a PDF usando la librería FPDF
    }

    public static function exportToExcel(array $inventory, string $filename): void {
        // Lógica para exportar a Excel usando la librería PhpSpreadsheet
    }
}
```

<?php

namespace InventoryManager;

class InventoryReport {

public static function generateReport(array \$inventory): array {

```

$report = [];

foreach ($inventory as $item) {

    $report[$item['Producto']] = [

        'Precio' => $item['Precio'],

        'Categoría' => $item['Categoría'],

        'Cantidad' => $item['Cantidad']

    ];

}

return $report;

}

public static function divideInventory(array $inventory, int $chunkSize): array {

    return array_chunk($inventory, $chunkSize);

}

public static function exportToPDF(array $inventory, string $filename): void {

    // Lógica para exportar a PDF usando la librería FPDF

}

public static function exportToExcel(array $inventory, string $filename): void {

    // Lógica para exportar a Excel usando la librería PhpSpreadsheet

}

}

```

4. Integración de la Biblioteca

Para integrar esta biblioteca en el proyecto, se debe crear un archivo composer.json y agregar las clases en el namespace InventoryManager. Luego, se puede incluir la biblioteca utilizando el autoload de Composer:

json

A screenshot of a code editor with a dark theme. The title bar shows 'json' and a 'Copiar código' button. The code is a JSON object for composer.json, with syntax highlighting: curly braces are grey, strings are red, and the namespace 'InventoryManager' is green. The code is as follows:

```
{
  "autoload": {
    "psr-4": {
      "InventoryManager\\": "src/"
    }
  }
}
```

```
{
  "autoload": {
    "psr-4": {
      "InventoryManager\\": "src/"
    }
  }
}
```

Después de instalar la biblioteca con Composer (composer dump-autoload), se puede usar en un archivo principal:

Después de instalar la biblioteca con Composer (`composer dump-autoload`), se puede usar en un archivo principal:

```
php Copiar código

<?php

require 'vendor/autoload.php';

use InventoryManager\Inventory;
use InventoryManager\InventoryOperations;
use InventoryManager\InventoryReport;

// Definir inventarios iniciales
$inventario = new Inventory([
    ["Producto" => "Teclado", "Precio" => 20, "Categoría" => "Electrónica", "Cantidad" =>
    // Más productos aquí...
]);

$informe = InventoryReport::generateReport($inventario->getAllItems());
print_r($informe);
```

Con esta estructura modular, la biblioteca `InventoryManager` facilita la gestión de inventarios y es mucho más mantenible.

```
<?php
```

```
require 'vendor/autoload.php';
```

```
use InventoryManager\Inventory;
```

```
use InventoryManager\InventoryOperations;
```

```
use InventoryManager\InventoryReport;
```

```
// Definir inventarios iniciales
```

```
$inventario = new Inventory([
```

```
    ["Producto" => "Teclado", "Precio" => 20, "Categoría" => "Electrónica", "Cantidad" => 4],
```

```
    // Más productos aquí...
```

```
]);
```

```
$informe = InventoryReport::generateReport($inventario->getAllItems());
```

```
print_r($informe);
```

Con esta estructura modular, la biblioteca InventoryManager facilita la gestión de inventarios y es mucho más mantenible.

Controles en Formularios

Introducción

Para que un control envíe información es necesario:

- Que el control esté incluido en un formulario (<form>).
- Que el formulario tenga establecido el atributo action, con la dirección absoluta o relativa del fichero php que procesa la información.
- Que el control tenga establecido el atributo name.
Nota: el atributo name puede contener cualquier carácter (números, acentos, guiones, etc), pero si contiene espacios, PHP sustituye los espacios por guiones bajos (_) al procesar los datos enviados.
- Que el formulario contenga un botón de tipo submit.

Un ejemplo de un formulario válido es:

```
<form action="ejemplo.php">
  <p>Nombre: <input type="text" name="nombre"></p>
  <p><input type="submit" value="Enviar"></p>
</form>
```

El programa que recibe los datos, los guarda automáticamente en la matriz **\$_REQUEST**. Mediante la orden **print_r(\$_REQUEST)** se puede mostrar el contenido de dicha matriz. El siguiente ejemplo muestra lo que escribiría el programa PHP (ejemplo.php) si recibiera la información del formulario anterior (ejemplo.html).

```
<?php
print "<pre>";
print_r($_REQUEST);
print "</pre>";
?>
```

```
Array
(
    [nombre] =>
)
```

El nombre del elemento de la matriz **\$_REQUEST** coincide con el nombre del control (atributo **name**) en el formulario (excepto en el control de tipo imagen).

Atributo method

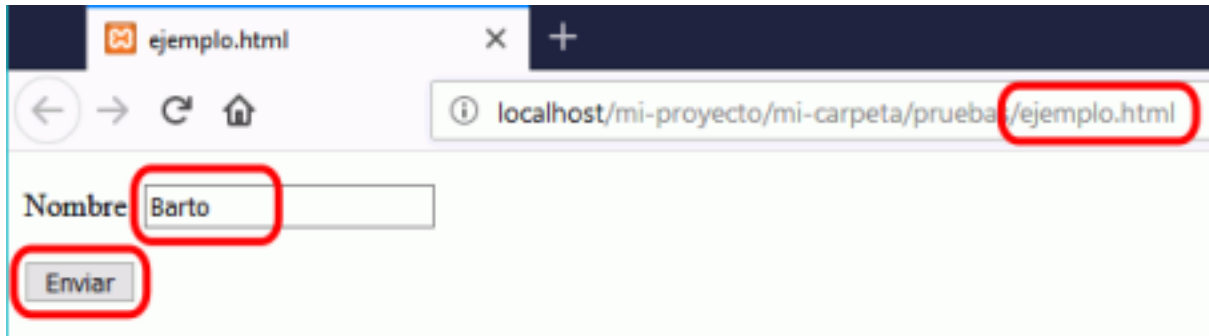
El atributo **method** de la etiqueta **<form>** permite elegir si la información de los controles se incluye en la llamada a la página (**method="get"**) o se proporciona posteriormente (**method="post"**). Si el atributo no está definido, la opción por defecto es get.

Con el valor get se pueden ver en la barra de dirección los nombres de los controles y los valores introducidos por el usuario, mientras que con el valor post no, pero los datos

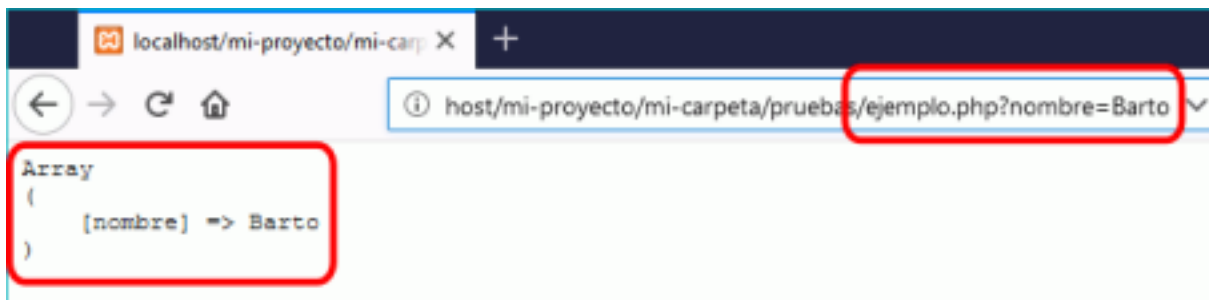
recibidos son idénticos.

Formulario con get

```
<form action="ejemplo.php" method="get">
  <p>Nombre: <input type="text" name="nombre"></p>
  <p><input type="submit" value="Enviar"></p>
</form>
```



```
<?php
print_r($_GET); // Ambos array tienen la misma información
print_r($_REQUEST);
?>
```



Las principales características del envío de formulario por el método get son:

- **Envía la información en la propia URL**, estando limitada a 2000 caracteres.
- **La información es visible por lo que con este método nunca se envía información sensible.**
 - No se pueden enviar datos binarios (archivos, imágenes...).
 - En PHP los datos se pueden recibir con el array asociativo `$_GET` además del `$_REQUEST`.
 - En caso de que haya varios controles que envíen información en un formulario con **get**, los nombres y valores aparecen en la barra de dirección separados por el carácter ampersand (&), como `nombre1=valor1&nombre2=valor2&...`

```
<form action="ejemplo.php" method="get">
  <p>Nombre: <input type="text" name="nombre"></p>
  <p>Apellidos: <input type="text"
name="apellidos"></p> <p><input type="submit"
value="Enviar"></p>
</form>
```

Nombre: Barto

Apellidos: Sintes Marco

Enviar

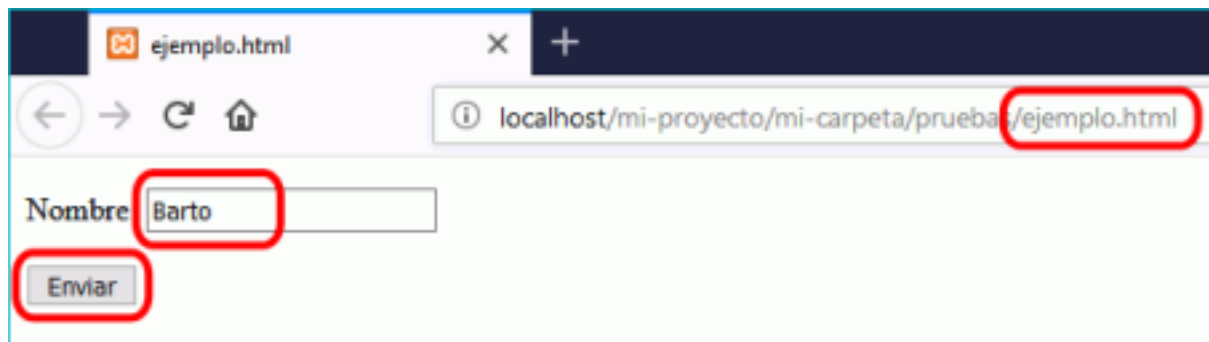
```
<?php
  print_r($_GET); // Ambos array tienen la misma información
  print_r($_REQUEST);
?>
```

Array

```
(
  [nombre] => Barto
  [apellidos] => Sintes Marco
)
```

Formulario con post

```
<form action="ejemplo.php" method="post">
  <p>Nombre: <input type="text" name="nombre"></p>
  <p><input type="submit" value="Enviar"></p>
</form>
```



```
<?php
    print_r($_POST); // Ambos array tienen la misma información
    print_r($_REQUEST);
?>
```

Las principales características del envío de formulario por el método **post** son:

- No tiene límite de cantidad de información a enviar.
- La información proporcionada no es visible, por lo que se puede enviar información sensible.
- **Se puede usar para enviar texto normal, así como datos binarios (archivos, imágenes...).**
- En PHP los datos se pueden recibir con el array asociativo `$ _POST` además del `$ _REQUEST`.

Botón Enviar

Este control se puede crear con la etiqueta **<input>** o con la etiqueta **<button>**. En ambos casos, este control se envía siempre que esté definido el atributo **name** y el valor enviado es el valor del atributo **value** o el contenido de la etiqueta.

```
<input type="submit" value="Submit">
```

```
<?php
    print_r($_REQUEST);
?>
```

```
Array
(
)
```

Desarrollo Web Entorno Servidor I.E.S. Velázquez

```
<input type="submit" value="Enviar" name="boton1">
```

```
<?php  
    print_r($_REQUEST);  
?>
```

```
Array  
(  
    [boton1] => Enviar  
)
```

```
<button type="submit">Submit</button>
```

```
<?php  
    print_r($_REQUEST);  
?>
```

```
Array  
(  
)
```

```
<button type="submit" name="boton2">Enviar</button>
```

```
<?php  
    print_r($_REQUEST);  
?>
```

```
Array  
(  
    [boton2] => Enviar  
)
```

```
<button type="submit" value="Enviar"
name="boton3">Submit</button>
```

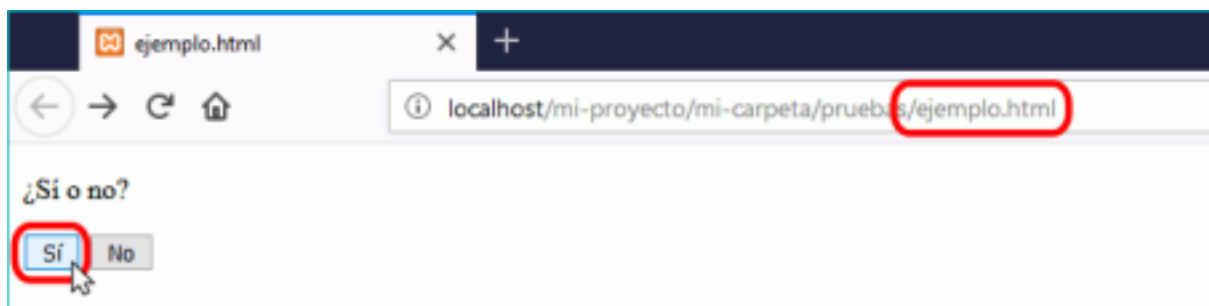
```
<?php
    print_r($_REQUEST);
?>
```

Array

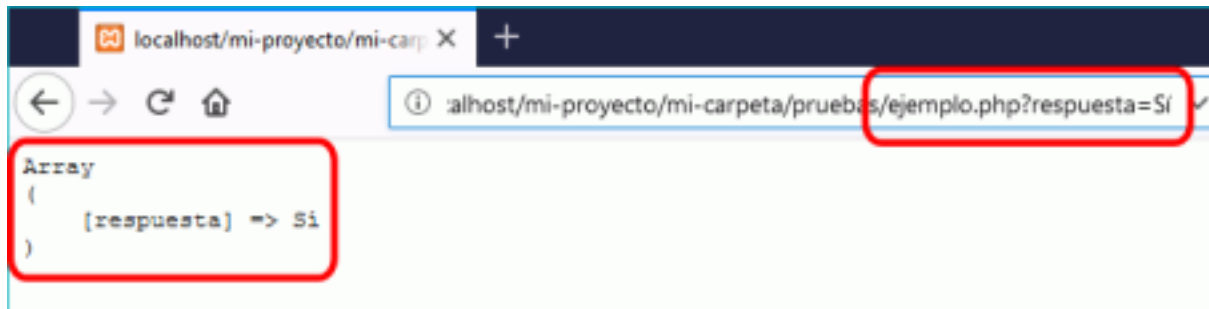
```
(
    [boton3] => Enviar
)
```

Normalmente **no se suele dar el atributo name al botón enviar** ya que la información se envía cuando se hace clic en el botón. En algunos casos, sí que **puede ser conveniente**, por ejemplo **cuando el formulario contiene varios botones y queremos saber cuál se ha pulsado**, como en el ejemplo siguiente, en el que los atributos **name** son iguales, pero los atributos **value** son distintos:

```
<form action="ejemplo.php" method="get">
    <p>¿Sí o no?</p>
    <p>
        <input type="submit" name="respuesta" value="Sí">
        <input type="submit" name="respuesta" value="No">
    </p>
</form>
```



```
<?php
    print_r($_REQUEST);
?>
```

Caja de texto, caja de contraseña y área de texto

Este control se envía siempre. El valor enviado es el contenido de la caja o área.

```
<input type="text" name="cajatexto1">
```

```
<?php
    print_r($_REQUEST);
?>
```

```
Array
(
    [cajatexto1] =>
)
```

```
<input type="text" name="cajatexto2" value="Cualquier valor">
```

```
<?php
    print_r($_REQUEST);
?>
```

```
Array
(
    [cajatexto2] => Cualquier valor
)
```

```
<input type="password" name="contrasena1">
```

```
<?php
    print_r($_REQUEST);
?>
```

```
Array
(
    [contrasena1] =>
```

```
<input type="password" name="contrasena2" value="123456">
```

```
<?php
    print_r($_REQUEST);
?>
```

```
Array
(
    [contrasena2] => 123456
```

```
<textarea rows="4" cols="20" name="area1"></textarea>
```

```
<?php
    print_r($_REQUEST);
?>
```

```
Array
(
    [area1] =>
```

```
<textarea rows="4" cols="20" name="area2"
value="123456">Contenido caja</textarea>
```

```
<?php
    print_r($_REQUEST);
?>
```

```
Array
(
    [area2] => Contenido caja
)
```

Casilla de verificación

Este control se envía solamente si se marca la casilla. El valor enviado es "on" si la casilla no tiene definido el atributo **value** o el valor del atributo **value** si ésta está definida.

```
<input type="checkbox" name="casilla1"> (sin marcar)
```

```
<?php
    print_r($_REQUEST);
?>
```

```
Array
(
)
```

```
<input type="checkbox" name="casilla1"> (marcada)
```

```
<?php
    print_r($_REQUEST);
?>
```

```
Array
(
    [casilla] => on
)
```

```
<input type="checkbox" name="casilla2" value="marcada"> (marcada)
```

```
<?php  
    print_r($_REQUEST);  
?>
```

```
Array  
(  
    [casilla2] => marcada  
)
```

Botón radio

Este control se envía solamente si se marca alguno de los botones radio que forman el control. El valor enviado es "on" si el botón marcado no tiene definido el atributo value o el valor del atributo **value** si éste está definido.

```
<input type="radio" name="radio1"> (sin marcar)  
<input type="radio" name="radio1">
```

```
<?php  
    print_r($_REQUEST);  
?>
```

```
Array  
(  
)
```

```
<input type="radio" name="radio1"> (marcada esta  
opción) <input type="radio" name="radio1">
```

```
<?php  
    print_r($_REQUEST);  
?>
```

```
Array
(
    [radio1] => on
)
```

```
<input type="radio" name="radio1">
<input type="radio" name="radio1"> (marcada esta opción)
```

```
<?php
    print_r($_REQUEST);
?>
```

```
Array
(
    [radio1] => on
)
```

```
<input type="radio" name="radio2" value="uno"> (marcado
uno) <input type="radio" name="radio2" value="dos">
```

```
<?php
    print_r($_REQUEST);
?>
```

```
Array
(
    [radio2] => uno
)
```

```
<input type="radio" name="radio2" value="uno">
<input type="radio" name="radio2" value="dos"> (marcado dos)
```

```
<?php
    print_r($_REQUEST);
?>
```

Array

```
(
    [radio2] => dos
)
```

Menú (select)

Este control envía siempre la opción elegida. **El valor enviado es el contenido de la etiqueta option elegida** si la opción elegida no tiene definido el atributo value o el valor del atributo value si éste está definido.

Si el menú admite selección múltiple, entonces el nombre del menú debe acabar con corchetes ([]) y se envía como una matriz, de tantos elementos como opciones se hayan elegido.

```
<select name="menu1">
    <option></option>
</select>
```

```
<?php
    print_r($_REQUEST);
?>
```

Array

```
(
    [menu1] =>
)
```

```
<select name="menu1">
    <option>Opción 1</option> (seleccionamos opción
    1) <option>Opción 2</option>
</select>
```

```
<?php
    print_r($_REQUEST);
?>
```

```
Array
(
    [menu1] => Opción 1
)
```

```
<select name="menu2">
  <option value="Uno">Opción 1</option> (seleccionamos opción
  1) <option value="Dos">Opción 2</option>
</select>
```

```
<?php
    print_r($_REQUEST);
?>
```

```
Array
(
    [menu1] => Uno
)
```

```
<select name="menu3[]" size="3" multiple>
  <option>Opción 1</option> (seleccionamos la opción 1 y
  3) <option>Opción 2</option>
  <option>Opción 3</option> (seleccionamos la opción 1 y
  3) <option>Opción 4</option>
</select>
```

```
<?php
    print_r($_REQUEST);
?>
```

```
Array
(
    [menu3] => Array
        (
            [0] => Opción 1
            [1] => Opción 2
        )
)
```

Control oculto (input hidden)

Este control se envía siempre y el valor enviado es el valor del atributo **value**.

```
<input type="hidden" name="oculto1">
```

```
<?php
    print_r($_REQUEST);
?>
```

Array

```
(
    [oculto1] =>
)
```

```
<input type="hidden" name="oculto2" value="valor">
```

```
<?php
    print_r($_REQUEST);
?>
```

```
Array
(
    [oculto2] => Valor
)
```

Imagen (input image)

El control de tipo imagen inserta una imagen que funciona como un botón (aunque ni

Firefox ni Internet Explorer le da relieve como a los botones). Al hacer clic en un punto de la imagen es como si se hubiera pulsado a un botón submit y se envían las coordenadas del punto en el que se ha hecho clic (junto con los valores de los otros controles del formulario).

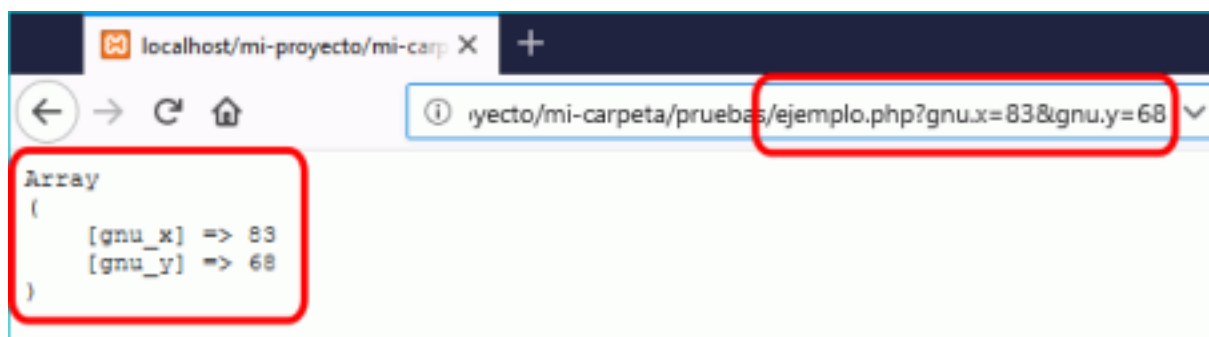
El origen de las coordenadas es el extremo superior izquierdo de la imagen. El valor X aumenta a medida que nos desplazamos a la derecha y el valor Y aumenta a medida que nos desplazamos hacia abajo.

```
<input type="image" name="gnu" src="gnu.png" alt="Logotipo GNU">
```



```
Array
(
    [gnu_x] => 83
    [gnu_y] => 68
)
```

Las coordenadas se reciben en \$_REQUEST en dos elementos que añaden al nombre del control el sufijo _x e _y. En la barra de dirección el nombre aparece con el sufijo .x e .y.



Si se define el atributo **value**, el formulario debe enviar tanto las coordenadas como el nombre del control con el valor del atributo **value**. Dependiendo de la versión del navegador es posible que el value de la imagen no se envíe.

Archivo (input file)

El selector de archivo permite enviar un archivo desde el ordenador del cliente al servidor. En un formulario "normal", este control se envía siempre y el valor enviado es el nombre del archivo elegido.

```
<input type="file" name="archivo">
```

```
<?php
    print_r($_REQUEST);
?>
```

```
Array
(
    [archivo] => texto.txt
)
```

Para que este control envíe toda la información, el formulario debe tener el atributo `enctype` con el valor **multipart/form-data** y ser enviado con el método **POST**. La información se almacena entonces en la matriz **\$_FILES** (pero no en la variable **\$_REQUEST**).

```
<form enctype="multipart/form-data"
action="ejemplo.php" method="post">
    <input type="file" name="archivo2">
</form>
```

```
<?php
    print_r($_FILES);
?>
```

```
Array
(
    [archivo1] => Array
        (
            [name] => loquesea.txt
            [type] => text/plain
            [tmp_name] => C:\ejemplos\texto.txt
            [error] => 0
            [size] => 27890
        )
)
```

Antes de utilizar este control, **hay que configurar en el archivo php.ini el tamaño máximo** de los archivos que se pueden enviar (mediante la directiva **post_max_size**).

Recogida de datos

Matriz \$_REQUEST

- ☐ Cuando se envía un formulario, PHP almacena la información recibida en una matriz llamada **\$_REQUEST**. **El número de valores recibidos y los valores recibidos dependen tanto del formulario como de la acción del usuario.**
- ☐ El atributo **name** del control puede contener cualquier carácter (números, acentos, guiones, etc), pero **si contiene espacios éstos se sustituyen por guiones bajos (_)**.
- ☐ Cada control crea un elemento de la matriz **\$_REQUEST**, que **se identifica como `$_REQUEST[valor_del_atributo_name]`** y que contiene el valor entregado por el formulario (en su caso).
- ☐ *Mientras se está programando, para comprobar que el fichero php está recibiendo la información enviada por el control, lo más fácil es utilizar la función `print_r($matriz)` para mostrar el contenido de la matriz **\$_REQUEST**. Una vez se ha comprobado que la información llega correctamente, la línea se debe comentar o eliminar.*

Referencia a \$_REQUEST dentro y fuera de cadenas

- ☐ Al hacer referencia a los elementos de la matriz **\$_REQUEST**, hay que tener en cuenta si la referencia se encuentra dentro de una cadena o fuera de ella:
 - ☐ Si se hace referencia **dentro de una cadena**, el índice se **debe escribir sin comillas**.
 - ☐ Si se escribe cualquier tipo de comillas (simples o dobles) se **produce un error**.

```
<form action="ejemplo.php" method="post">
    <p>Nombre: <input type="text" name="nombre"
value="Antonio"></p>
    <p><input type="submit" value="Enviar"></p>
</form>
```

```
<?php
    print "<p>Su nombre es $_REQUEST[nombre]</p>";
?>
```

```
<p>Su nombre es Antonio</p>
```

```
<?php
    print "<p>Su nombre es $_REQUEST['nombre']</p>";
?>
```

Parse error: syntax error, unexpected '"', expecting '-' or identifier (T_STRING) or variable (T_VARIABLE) or number (T_NUM_STRING) in **ejemplo.php** on line 2

```
<?php
    print "<p>Su nombre es $_REQUEST['nombre']</p>";;
?>
```

Parse error: syntax error, unexpected '' (T_ENCAPSED_AND_WHITESPACE), expecting '-' or identifier (T_STRING) or variable (T_VARIABLE) or number (T_NUM_STRING) in **ejemplo.php** on line 2

- ☐ **Si se hace referencia fuera de una cadena**, el índice se debe escribir **con comillas dobles o simples**. Si se escribe sin comillas, se produce un aviso

```
<?php
    print "<p>Su nombre es " . $_REQUEST["nombre"] .
"</p>"; ?>
```

```
<p>Su nombre es Antonio</p>
```

```
<?php
    print "<p>Su nombre es " . $_REQUEST['nombre'] .
"</p>"; ?>
```

```
<p>Su nombre es Antonio</p>
```

```
<?php
    print "<p>Su nombre es " . $_REQUEST[nombre] .
"</p>"; ?>
```

Warning: Use of undefined constant nombre - assumed 'nombre' in **ejemplo.php** on line 2

```
<p>Su nombre es Antonio</p>
```

- ☐ **En ningún caso se pueden utilizar los caracteres especiales \" o \', ni dentro ni fuera de las cadenas.**

```
<?php
    print "<p>Su nombre es
$_REQUEST[\"nombre\"]</p>"; ?>
```

Parse error:: syntax error, unexpected ''
(T_ENCAPSED_AND_WHITESPACE), expecting '-' or identifier
(T_STRING) or variable (T_VARIABLE) or number (T_NUM_STRING) in **ejemplo.php** on line 2

```
<?php
    print "<p>Su nombre es
$_REQUEST['nombre']</p>"; ?>
```

```
Parse error:: syntax error, unexpected ''
(T_ENCAPSED_AND_WHITESPACE), expecting '-' or identifier
(T_STRING) or variable (T_VARIABLE) or number (T_NUM_STRING) in
ejemplo.php on line 2
```

Comprobación de existencia

- ☐ Un programa PHP no debe suponer que los controles le van a llegar siempre porque cuando eso no ocurra, el programa no funcionará correctamente.

1-Controles vacíos == los controles no contengan ningún valor.

- ☐ En el programa de ejemplo, aunque el programa funcione correctamente, la respuesta del programa puede confundir al usuario:

```
<form action="ejemplo.php" method="post">
  <p>Nombre: <input type="text" name="nombre"
  value=""></p> <p><input type="submit"
  value="Enviar"></p>
</form>
```

```
<?php
  print "<pre>";
  print_r($_REQUEST);
  print "</pre>";
  print "<p>Su nombre es $_REQUEST[nombre]</p>";
?>
```

```
<pre>
Array
(
    [nombre] =>
)
</pre>
<p>Su nombre es</p>
```

- ☐ Este problema se puede resolver incluyendo una estructura if ... else ... que considere la posibilidad de que no se haya escrito nada en el formulario:

```
<form action="ejemplo.php" method="post">
  <p>Nombre: <input type="text" name="nombre"
  value=""></p> <p><input type="submit"
  value="Enviar"></p>
</form>
```

```
<?php
    print "<pre>";
    print_r($_REQUEST);
    print "</pre>";
    if ($_REQUEST["nombre"] == "") {
        print "<p>No ha escrito ningún nombre</p>";
    } else {
        print "<p>Su nombre es $_REQUEST[nombre]</p>";
    }
?>
```

```
<pre>
Array
(
    [nombre] =>
)
</pre>
<p>No ha escrito ningún nombre</p>
```

2- Controles inexistentes: isset()

- ☐ Un problema más grave es que el programa suponga que existe un control que en realidad no le ha llegado.

Ej. En el caso de las casillas de verificación los formularios envían el control solamente cuando se han marcado.

- ☐ En el siguiente ejemplo el usuario no marca el **checkbox** y la matriz \$_REQUEST no contiene el dato.--> el programa **genera un aviso por hacer referencia a un índice no definido**, aunque se haya incluido la estructura if ... else ...

```
<form action="ejemplo.php" method="post">
    <p>Deseo recibir información:
        <input type="checkbox" name="acepto"> (sin marcar la
        casilla) </p>
    <p><input type="submit" value="Enviar"></p>
</form>
```

```
<?php
    print "<pre>";
    print_r($_REQUEST);
    print "</pre>";
    if ($_REQUEST["acepto"] == "on"){
        print "<p>Desea recibir información</p>";
    }else{
        print "<p>No desea recibir información</p>";
    }
?>
```

```
<pre>
Array
(
)
</pre>
```

Notice: Undefined index: acepto in **ejemplo.php** on line 4

```
<p>No desea recibir información</p>
```

- ☐ Este problema **puede ocurrir** en realidad **en cualquier formulario** (incluso en formularios con cajas de texto que en principio envían siempre el control) **ya que el usuario puede escribir directamente la dirección del programa PHP en el navegador y ejecutar el programa PHP sin pasar por el formulario.**
 - ☐ En ese caso, el programa ,que no recibe ningún control, genera un aviso que hace referencia a un índice no definido
- ☐ **Estos problemas se pueden resolver comprobando que el índice está definido antes de hacer referencia a él**, utilizando la función **isset(\$variable)**, que **admite como argumento una variable** y **devuelve true** si existe **y false** si no existe.

```
<form action="ejemplo.php" method="post">
    <p>Deseo recibir información:
        <input type="checkbox" name="acepto"> (marcada la
        casilla) </p>
    <p><input type="submit" value="Enviar"></p>
</form>
```



```
<?php
    if (isset($_REQUEST["acepto"])){
        print "<p>Desea recibir información</p>";
    }else{
        print "<p>No desea recibir información</p>";
    }
?>
```

```
<p>Desea recibir información</p>
```

- ☐ **Es conveniente efectuar siempre la verificación de existencia para prevenir los casos en que un usuario intente acceder a la página PHP sin pasar por el formulario.**

Seguridad en las entradas

EJ.Un usuario puede insertar código html en la entrada de un formulario, lo que puede acarrear comportamientos inesperados y riesgos de seguridad.

Ejemplo de problema de seguridad: Inyección SQL

Supongamos que tenemos un formulario de inicio de sesión simple donde el usuario ingresa su nombre de usuario y contraseña. El servidor PHP recibe estos datos y los utiliza para autenticar al usuario contra una base de datos.

Este código PHP es vulnerable a inyección SQL porque los datos recibidos del formulario (nombre de usuario y contraseña) se insertan directamente en la consulta SQL sin ninguna validación o sanitización. Un atacante podría manipular la entrada para ejecutar código SQL malicioso.

```
<h2>Iniciar sesión</h2>
```

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

    <label for="username">Nombre de usuario:</label>

    <input type="text" name="username" id="username"
required><br><br>

    <label for="password">Contraseña:</label>

    <input type="password" name="password" id="password"
required><br><br>

    <button type="submit">Iniciar sesión</button>
```

</form>

Script PHP vulnerable (**login.php**)

```
<?php
if (isset($_POST['username']) && isset($_POST['password'])) {
    $username = $_POST['username'];
    $password = $_POST['password'];

    // Conexión a la base de datos (ejemplo simple)
    $conn = new mysqli('localhost', 'root', '', 'testdb');

    if ($conn->connect_error) {
        die("Error de conexión: " . $conn->connect_error);
    }

    // Consulta SQL vulnerable a inyección
    $sql = "SELECT * FROM users WHERE username = '$username' AND password = '$password'";

    $result = $conn->query($sql);

    if ($result->num_rows > 0) {
        echo "¡Inicio de sesión exitoso!";
    } else {
        echo "Nombre de usuario o contraseña incorrectos.";
    }

    $conn->close();
}
?>
```

Ejemplo de ataque:

Un atacante podría ingresar lo siguiente en el campo del nombre de usuario:

```
' OR '1'='1
```

Y cualquier valor en el campo de **contraseña**

El código SQL que se generaría sería:

```
SELECT * FROM users WHERE username = '' OR '1'='1' AND password = 'cualquier_valor';
```

Esta consulta siempre será verdadera porque **'1'='1'** es una condición que se cumple.

Esto podría permitir al atacante iniciar sesión sin conocer el nombre de usuario ni la contraseña válidos.

Eliminar espacios en blanco iniciales y finales: trim(\$cadena)

- ☐ **A veces, al rellenar un formulario, los usuarios escriben por error espacios en blanco al principio o al final de las cajas de texto y si el programa PHP no tiene en cuenta esa posibilidad se pueden obtener resultados inesperados.**

EJ. con resultado erróneo

```
<form action="ejemplo.php" method="post">
  <p>Nombre: <input type="text" name="nombre" value=" ">
  <p><input type="submit" value="Enviar"></p>
</form>
```

```
<?php
  print "<pre>";
  print_r($_REQUEST);
  print "</pre>";
  if ($_REQUEST["nombre"] == ""){
    print "<p>No ha escrito ningún nombre</p>";
  }else{
    print "<p>Su nombre es $_REQUEST[nombre]</p>";
  }
?>
```

```
<pre>
Array
(
    [nombre] =>
)
</pre>
<p>Su nombre es </p>
```

- ☐ **Este problema se puede resolver utilizando la función trim(\$cadena), que elimina los espacios en blanco iniciales y finales y devuelve la cadena sin esos espacios.**

```
<form action="ejemplo.php" method="post">
  <p>Nombre: <input type="text" name="nombre" value=" ">
  <p><input type="submit" value="Enviar"></p>
</form>
```

```
<?php
    print "<pre>";
    print_r($_REQUEST);
    print "</pre>";
    if (trim($_REQUEST["nombre"]) == ""){
        print "<p>No ha escrito ningún nombre</p>";
    }else{
        print "<p>Su nombre es $_REQUEST[nombre]</p>";
    }
?>
```

<pre>

Array

```
(
    [nombre] =>
```

</pre>

<p>No ha escrito ningún nombre**</p>**

Utilización de variables

- ☐ Hemos visto que para resolver los problemas comentados en los apartados anteriores, al incluir cualquier referencia a `$_REQUEST[control]` habría que:
 - ☐ comprobar que el control esté definido con la función **isset()**
 - ☐ eliminar los espacios en blanco iniciales y finales con la función **trim()**
- ☐ Una manera de hacerlo sin complicar excesivamente el programa es guardar los valores de la matriz `$_REQUEST` en variables y realizar todas las comprobaciones al definir esas variables.
- ☐ En el resto del código basta con utilizar la variable en vez del elemento de la matriz `$_REQUEST`.

```

<?php
    print "<pre>";
    print_r($_REQUEST);
    print "</pre>";
    if (isset($_REQUEST["nombre"])){
        $nombre = trim(strip_tags($_REQUEST["nombre"]));
    }else{
        $nombre = "";
    }

    if ($nombre == ""){
        print "<p>No ha escrito ningún nombre</p>";
    }else{
        print "<p>Su nombre es$nombre</p>";
    }
?>

```

```

<pre>
Array
(
    [nombre] =>
)
</pre>
<p>No ha escrito ningún nombre</p>

```

☐ RECORDATORIO La asignación de la variable se puede realizar en una sola instrucción, utilizando la notación abreviada: (condición) ? verdadero : falso;

```

<?php
    print "<pre>";
    print_r($_REQUEST);
    print "</pre>";
    $nombre = isset($_REQUEST["nombre"]) ?
    trim(strip_tags($_REQUEST["nombre"])) : "";

    if ($nombre == ""){
        print "<p>No ha escrito ningún nombre</p>";
    }else{
        print "<p>Su nombre es $nombre</p>";
    }
?>

```

htmlspecialchars()

- ☐ Ciertos caracteres tienen un significado especial en HTML y deben ser representados por entidades HTML si se desea preservar su significado.
- ☐ La función [htmlspecialchars\(\)](#) de PHP realiza la siguiente sustitución de caracteres por su correspondiente entidad HTML:

Carácter	Sustitución
&	&
" (comilla doble)	"
' (comilla simple)	'
< (menor que)	<
> (mayor que)	>

- ☐ La función **htmlspecialchars()** además de la cadena a convertir admite otros parámetros entre el que hay que destacar el segundo que es un entero (\$flags).
- ☐ En este segundo parámetro se especifica cómo manejar las comillas, las secuencias de unidad de código inválidas y el tipo de documento utilizado. Por defecto el valor que tiene es (**ENT_COMPAT | ENT_HTML401**) que convierte las comillas dobles y deja solo las comillas simples y maneja el código como HTML 4.01).

```
<form action="ejemplo.php" method="post">
  <p>Nombre: <input type="text" name="nombre"
value=' "Antonio"'></p>
  <p><input type="submit" value="Enviar"></p>
</form>
```

```

<?php
    print "<pre>";
    print_r($_REQUEST);
    print "</pre>";
    if (isset($_REQUEST["nombre"])){
        $nombre =
htmlspecialchars(trim(strip_tags($_REQUEST["nombre"])))
        ; }else{
        $nombre = "";
    }

    if ($nombre == ""){
        print "<p>No ha escrito ningún nombre</p>";
    }else{
        print "<p>Su nombre es $nombre</p>";
    }
?>

```

```

<pre>
Array
(
    [nombre] => "Antonio"
)
</pre>
<p>Su nombre es &quot;Antonio&quot;</p>

```

NOTA: Existe otra función de PHP **htmlentities()** muy parecida a **htmlspecialchars()**. A diferencia de **htmlspecialchars**, **htmlentities convierte** no sólo los caracteres especiales de la cadena en entidades HTML, sino **todos los caracteres aplicables en entidades HTML**.

```

<?php
    print "<h3>htmlspecialchars</h3>";
    print "<p>";
    print htmlspecialchars("Ejemplo <br>");
    print "</p>";
    print "<p>";
    print htmlspecialchars("µ †");
    print "</p>";

    print "<h3>htmlentities</h3>";
    print "<p>";
    print htmlentities("Ejemplo <br>");
    print "</p>";
    print "<p>";
    print htmlentities("µ †");
    print "</p>";
?>

```

La salida HTML es la misma en ambos casos:

```

<h3>htmlspecialchars</h3>
<p>Ejemplo <br></p>
<p>µ †</p>
<h3>htmlentities</h3>
<p>Ejemplo <br></p>
<p>µ †</p>

```

Pero el código fuente de la página es diferente:

```

<h3>htmlspecialchars</h3>
<p>Ejemplo &lt;br></p>
<p>µ †</p>
<h3>htmlentities</h3>
<p>Ejemplo &lt;br></p>
<p>&micro; &dagger;</p>

```

Funciones de recogida de datos

- ☐ La forma más cómoda de tener en cuenta todos los aspectos comentados en los puntos anteriores es definir una función:
- ☐ Una vez definida la función, al comienzo del programa se almacenarán en variables los datos devueltos por la función.
- ☐ En el resto del programa se trabaja con las variables.
 - ☐ La función tendrá como argumento el nombre del control que se quiere recibir

- ☐ Y devolverá el valor recibido (o una cadena vacía si el control no se ha recibido).
- ☐ Para tratar los datos recibidos se aplicarán únicamente las funciones **htmlspecialchars()** y **trim()**: De esta manera, si se reciben etiquetas (<>), las etiquetas no se borrarán, sino que se conservarán como texto.

```
trim(htmlspecialchars($_REQUEST[$var], ENT_QUOTES, "UTF-8"));
```

- ☐ Si se quisieran borrar las etiquetas, habría que aplicar además la función **strip_tags()**:

```
trim(htmlspecialchars(strip_tags($_REQUEST[$var])), ENT_QUOTES, "UTF-8");
```

EJ.

```
<!DOCTYPE html>
<html>
<head>
  <title>Formulario de ejemplo</title>
</head>
<body>
  <form action="procesar_formulario.php" method="post">
    <label for="nombre">Nombre:</label>
    <input type="text" id="nombre" name="nombre"><br><br>

    <label for="email">Email:</label>
    <input type="email" id="email" name="email"><br><br>
    <input type="submit" value="Enviar">
  </form>
</body>
</html>
```

biblioteca.php

```
<?<?php
function recoger_y_limpiar_datos($var) {
  // Comprobamos si la variable existe en el array $_REQUEST
  if (isset($_REQUEST[$var])) {
    // Limpiamos el dato:
    // - Eliminamos espacios en blanco al principio y al final.
    // - Convertimos caracteres especiales a entidades HTML.
    // - (Opcional) Eliminamos todas las etiquetas HTML.
```

```

$dato_limpio = trim(htmlspecialchars(strip_tags($_REQUEST[$var]), ENT_QUOTES,
"UTF-8"));
// Si quieres mantener las etiquetas, simplemente quita strip_tags():
// $dato_limpio = trim(htmlspecialchars($_REQUEST[$var], ENT_QUOTES, "UTF-8"));
return $dato_limpio;
} else {
return null; // O un mensaje de error si lo prefieres
?>

```

procesar_formulario.php

```

<?php
// Recogemos el nombre y el email del formulario:
include "biblioteca.php";
$nombre = recoger_y_limpiar_datos('nombre');
$email = recoger_y_limpiar_datos('email');

// Validamos los datos (ejemplo simple):
if (empty($nombre) || empty($email)) {
    print_r(nl2br("Formulario no válido.\nPor favor, completa todos los
campos antes de enviarlo.\n"));
} else {
    echo "<br>Nombre: $nombre.<br>";
    echo "Email: ".$email."<br>";
}
?>

```

Comprobación de datos

- ☐ Antes de utilizar en un programa los datos recibidos a través de un formulario, es necesario comprobar si el dato recibido corresponde a lo esperado.
 - ☐ Algunas de las comprobaciones se pueden hacer con comparaciones (si no es vacío si es un valor comprendido entre unos valores determinado, etc.),
 - ☐ Pero **antes de hacer esas comparaciones** hay que **comprobar que es del tipo esperado** (número entero o decimal, texto, etc.) para procesarlo sin error.
- ☐ En esta lección se verán una serie de familias de funciones que permiten comprobar la existencia, el tipo o el contenido de un dato:
 - ☐ funciones is_

- ☐ funciones **ctype_**
- ☐ funciones **filter_**
- ☐ funciones **_exists**
- ☐ **Pero antes** se comenta un caso especial, la **comprobación de números**, para la que se aconseja utilizar las funciones **is_numeric()** y **ctype_digit()**.

Comprobación de números con is_numeric() y ctype_digit()

Cuando se quiere comprobar que un dato recibido a través de un formulario es un número, en el caso de las funciones **is_** o **ctype_**, sólo tiene sentido utilizar dos de ellas; **is_numeric()** y **ctype_digit()**.

Comprobación de números con is_numeric()

- ☐ Para **comprobar si el dato que se ha recibido es un número** (o se puede interpretar como número) se debe utilizar la función **is_numeric(\$valor)**. Esta función lógica **devuelve true** si el argumento se puede interpretar como **número** y **false** si no lo es.
- ☐ El **motivo** por el que se debe utilizar la función **is_numeric(\$valor)**, es que **lo que se recibe a través de un formulario se guarda siempre como cadena** (aunque sea un número).
 - ☐ La función **is_numeric(\$valor)** **es la única que comprueba si el argumento se puede interpretar como número** (aunque el argumento sea de tipo cadena), la función **is_int(\$valor)** o **is_float(\$valor)** hacen solamente comprobaciones sobre el tipo de los datos y devolverán siempre **false**.

Comprobación de números enteros positivos con ctype_digit()

- ☐ Para **comprobar si el dato que se ha recibido es un número entero positivo** (sin decimales) se puede utilizar la función **ctype_digit(\$valor)**.
 - ☐ Esta función lógica **devuelve true si todos los caracteres del argumento son dígitos (0, 1, ..., 9)** y **false** si no lo son.

Funciones is_

Las funciones **is_** son un conjunto de funciones booleanas que devuelven **true** si el argumento es de un tipo de datos determinado y **false** si no lo son.

Función	Tipo de datos	alias (funciones equivalentes)
isset(\$valor)	devuelve si el dato está definido o no	
is_null(\$valor)	null	
is_bool(\$valor)	booleano	

Función	Tipo de datos	alias (funciones equivalentes)
is_numeric(\$valor)	número (puede tener signo, parte decimal y estar expresado en notación decimal, exponencial o hexadecimal).	
is_int(\$valor)	entero	is_integer(\$valor) , is_long(\$valor)
is_float(\$valor)	float	is_double(\$valor) , is_real(\$valor)
is_string(\$valor)	cadena	
is_scalar(\$valor)	escalar (entero, float, cadena o booleano)	
is_array(\$valor)	matriz	
is_callable(\$valor)	función	
is_object(\$valor)	object	
is_resource(\$valor)	recurso	
is_countable(\$valor)	contable (matriz u objeto que implementa Countable)	

- ☐ Si un dato es demasiado grande para el tipo de variable, las funciones devolverán false. Por ejemplo, si se usa como argumento un entero mayor que `PHP_INT_MAX` (2147483647 en Windows), la función `is_int()` devolverá false.

Estas funciones son en general de poca utilidad con datos provenientes de un formulario, ya que estas funciones lo que **comprueban es el tipo de los datos y la información que llega de un formulario es siempre del tipo cadena.**

- ☐ Las dos excepciones serían:
 - ☐ La función `is_numeric()`, que evalúa si el argumento se puede interpretar como número (aunque el tipo sea cadena). Por tanto esta función se puede utilizar para comprobar si un dato recibido es un número.
 - ☐ La función `isset()`, que evalúa si el argumento está o no definido, independientemente de su tipo.

El ejemplo siguiente muestra el uso de la función `is_numeric()`.

```
<form action="ejemplo.php" method="post">
  <p>Número: <input type="text" name="numero"
    value=""></p> <p><input type="submit"
    value="Enviar"></p>
</form>
```

ejemplo.php

```
<?php
print "<pre>";
print_r($_REQUEST);
print "</pre>";
$numero = $_REQUEST["numero"];
if (is_numeric($numero)){
    print "<p>Ha escrito un número: $numero</p>";
}else{
    print "<p>No ha escrito un número: $numero</p>";
}
?>
```

Funciones ctype_

- ☐ Las funciones `ctype_` son un conjunto de funciones booleanas que devuelven si todos los caracteres de una cadena son de un tipo determinado, de acuerdo con el juego de caracteres local. **Estas funciones son las mismas que las que proporciona [la biblioteca estándar de C](#) `ctype.h`.**

Función	Tipo de datos
<code>ctype_alnum(\$valor)</code>	alfanuméricos

ctype_alpha(\$valor)	alfabéticos (mayúsculas o minúsculas, con acentos, ñ, ç, etc)
ctype_cntrl(\$valor)	caracteres de control (salto de línea, tabulador, etc)
ctype_digit(\$valor)	dígitos
ctype_graph(\$valor)	caracteres imprimibles (excepto espacios)
ctype_lower(\$valor)	minúsculas
ctype_print(\$valor)	caracteres imprimibles
ctype_punct(\$valor)	signos de puntuación (caracteres imprimibles que no son alfanuméricos ni espacios en blanco)
ctype_space(\$valor)	espacios en blanco (espacios, tabuladores, saltos de línea, etc)
ctype_upper(\$valor)	mayúsculas
ctype_xdigit(\$valor)	dígitos hexadecimales

- ☐ Estas funciones se pueden aplicar a los datos recibidos de un formulario ya que hacen comprobaciones de todos los caracteres de una cadena. Quizás **la más útil es la función ctype_digit()**, que **evalúa si todos los caracteres son dígitos, por lo que permite identificar los números enteros positivos** (*sin punto decimal ni signo negativo*)

Funciones filter_

- ☐ La función filter más simple es la función **filter_var(\$valor [, \$filtro [, \$opciones]])**, que devuelve los datos filtrados o false si el filtro falla.

Los filtros predefinidos de validación son los siguientes:

Filtro	Tipo de datos
FILTER_VALIDATE_INT	entero
FILTER_VALIDATE_BOOLEAN	booleano
FILTER_VALIDATE_FLOAT	float
FILTER_VALIDATE_REGEXP	expresión regular

FILTER_VALIDATE_DOMAIN	dominio web
FILTER_VALIDATE_URL	URL
FILTER_VALIDATE_EMAIL	dirección de correo
FILTER_VALIDATE_IP	dirección IP
FILTER_VALIDATE_MAC	dirección MAC física

NOTA: El problema de los filtros **FILTER_VALIDATE_INT** y **FILTER_VALIDATE_FLOAT** es que dan false si el argumento es 0, lo que complica su uso para detectar números porque el caso del 0 debe considerarse aparte al hacer la validación.

El ejemplo siguiente muestra el uso de la función **filter_var()** con el argumento **FILTER_VALIDATE_INT**.

```
<form action="ejemplo.php" method="post">
  <p>Número: <input type="text" name="numero"
    value=""></p> <p><input type="submit"
    value="Enviar"></p>
</form>
```

ejemplo.php

```
<?php
print "<pre>";
print_r($_REQUEST);
print "</pre>";
$numero = $_REQUEST["numero"];

if (filter_var($numero, FILTER_VALIDATE_INT)) {
    print "<p>Ha escrito un número entero:
    $numero</p>"; }else{
    print "<p>No ha escrito un número entero:
    $numero</p>"; }
?>
```

Funciones xxx_exists()

Función function_exists()

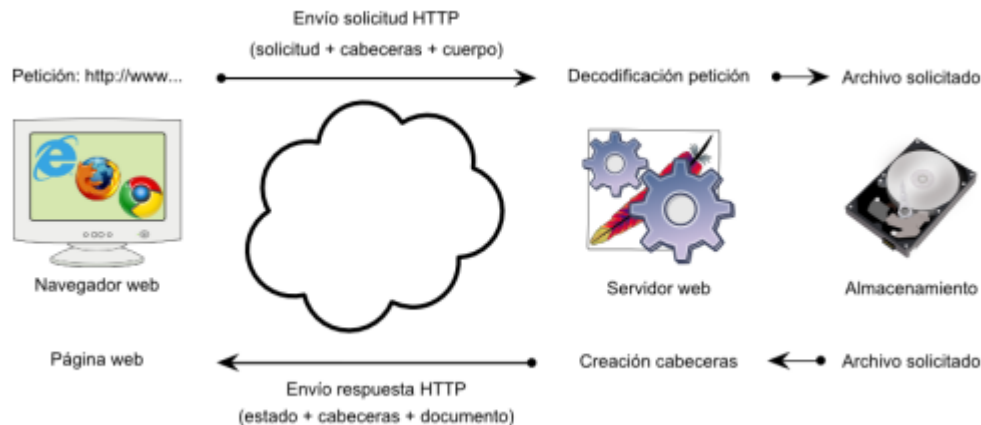
- ☐ La función booleana **function_exists()** devuelve si la función existe o no.

Función array_key_exists()

- ☐ La función booleana **array_key_exists(\$indice, \$matriz)** devuelve si un elemento determinado de una matriz existe o no.
- ☐ Para comprobar si existe un elemento de una matriz también se puede utilizar la función **isset()** comentada anteriormente.

El protocolo HTTP

- ☐ Comunicación entre el cliente y el servidor
- ☐ Cuando un usuario solicita una página web a un servidor, el proceso se puede describir (de forma simplificada) en cuatro pasos:
 - ☐ El navegador solicita al servidor el documento mediante una solicitud HTTP.
 - ☐ El servidor prepara el documento.
 - ☐ El servidor envía el documento al navegador mediante una respuesta HTTP.
 - ☐ El navegador muestra el documento al usuario.



Solicitudes HTTP

- ☐ La solicitud HTTP está formada por varias líneas de texto:
 - La línea de solicitud, que incluye:
 - El método que se va a utilizar (GET, POST, TRACE, etc.).
 - La ubicación del documento solicitado (URI).
 - La versión del protocolo HTTP.

HTML:

```
<a href="https://example.com">Visitar ejemplo.com</a>
```

```
<form action="/submit" method="POST">
```

```
  <input type="text" name="nombre">
```

```
  <button type="submit">Enviar</button>
```

```
</form>
```

Solicitud HTTP (aproximada):

```
POST /submit HTTP/1.1
```

```
Host: example.com
```


Content-Type: application/x-www-form-urlencoded
nombre=JohnDoe (datos del formulario codificados en URL.)

- Los campos de cabecera, que pueden incluir entre otros:
 - Referer: dirección de donde se ha obtenido la dirección del documento solicitado (si una página A contiene un enlace a otra página B, el navegador solicita en la línea de petición la página B, pero también envía la dirección de la página A como referer).
 - User-agent: información sobre el navegador (nombre, versión, etc).
 - Accept: tipos MIME, juegos de caracteres, codificaciones, idiomas, etc. admitidos por el navegador.
 - Cookies: Si el servidor ha almacenado previamente cookies en el cliente, estas se incluyen en las peticiones.
- Una línea en blanco.
- El cuerpo del mensaje, que incluye texto opcional como por ejemplo:
 - datos de un formulario cuyo método sea POST

GET /pagina.html HTTP/1.1

Host: www.ejemplo.com

Referer: <https://www.paginaanterior.com>

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/118.0.0.0 Safari/537.36

Accept:

text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9

Cookie: session_id=123456789;

other_cookie=value

Connection: keep-alive

nombre=Juan&apellido=Pérez&edad=30

IMPORTANTE; **HTML no tiene un control directo sobre** las cabeceras HTTP como **Referer, User-Agent o Cookie**. Estas cabeceras **son** generalmente **agregadas automáticamente** por el navegador.

Respuestas HTTP

- ☐ **El servidor web** es el responsable de generar las respuestas HTTP.
 - ☐ Cuando un cliente (un navegador, por ejemplo) envía una solicitud, el servidor procesa la solicitud y envía una respuesta de acuerdo con las reglas del protocolo HTTP.
- ☐ Los factores que influyen en la respuesta HTTP son:
 - ☐ **La solicitud del cliente:** El método HTTP (GET, POST, PUT, DELETE), la URL, las cabeceras y el cuerpo de la solicitud.

- ☐ **La configuración del servidor:** Las reglas de reescritura, los archivos de configuración, los scripts y aplicaciones que se ejecutan en el servidor.
- ☐ **El contenido del servidor**
- ☐ La respuesta HTTP está formada por varias líneas de texto:
 - La línea de estado, que incluye:
 - La versión del protocolo HTTP.
 - El código de status (403, 404, 500, etc.).
 - El texto asociado al código de status (403=Forbidden, 404=Not found, 500=Internal Server Error, etc.).
 - Los campos de cabecera, que pueden incluir entre otros:
 - Location: permite decirle al cliente que solicite otro documento en lugar del documento solicitado inicialmente.
 - Content: tipo MIME, juego de caracteres, codificación, idioma, etc. del documento enviado.
 - Set-Cookie: permite decirle al cliente que cree una cookie.
 - Una línea en blanco.
 - El cuerpo del mensaje, que incluye el documento solicitado por el cliente.

HTTP/1.1 302 Found

Location: <https://www.nuevositio.com/>

Content-Type: text/html; charset=utf-8

```
<html>
<head>
  <meta http-equiv="Refresh" content="0; URL=https://www.nuevositio.com/">
</head>
<body>
  Redireccionando...
</body>
</html>
```

Cabeceras: la función header()

Función header()

- ☐ Cuando un **servidor envía** una página web al navegador envía **la página web e información adicional** (*el estado y los campos de cabecera*).
- ☐ Tanto el **estado** como los **campos de cabecera** se envían **antes de la página web**.
- ☐ Normalmente, un programa PHP sólo genera la página web y es el servidor el que genera automáticamente la información de estado y los campos de cabecera y los envía antes de enviar el contenido generado por el programa.
- ☐ Pero un programa PHP también puede generar la información de estado y los campos de cabecera, mediante la función header().

Ejemplo

```

<?php
// Configurar la cabecera de estado HTTP
header('HTTP/1.1 200 OK');

// Configurar las cabeceras personalizadas
header('Content-Type: text/html; charset=utf-8');
header('X-Powered-By: MiFramework PHP');

// Enviar una cookie
setcookie('nombre_cookie', 'valor_cookie', time() + 3600); // Expira en
1 hora

// Generar el contenido HTML
echo '<!DOCTYPE html>
<html>
<head>
    <title>Ejemplo de cabeceras en PHP</title>
</head>
<body>
    <h1>¡Hola desde mi servidor PHP!</h1>
    <p>Esta página ha sido personalizada con cabeceras HTTP.</p>
</body>
</html>';
?>

```

Explicación del código:

Configuración del estado HTTP:

header('HTTP/1.1 200 OK');: Establece el código de estado HTTP en 200 (OK), indicando que la solicitud se ha completado con éxito.

Configuración de cabeceras personalizadas:

header('Content-Type: text/html; charset=utf-8');: Especifica que el contenido de la respuesta es HTML y utiliza la codificación UTF-8.

header('X-Powered-By: MiFramework PHP');: Agrega una cabecera personalizada para indicar que la página fue generada por tu propio framework PHP.

Envío de una cookie:

setcookie('nombre_cookie', 'valor_cookie', time() + 3600);: Crea una cookie llamada *nombre_cookie* con el valor *valor_cookie* que expirará en una hora.

Generación del contenido HTML:

Se crea el contenido HTML de la página y se envía al navegador.

En access.log de APACHE podemos ver el estado de la solicitud HTTP y cabecera. En este caso :

**"GET /clase/Propio/Cabecera.php HTTP/1.1" 200 219 "http://localhost/clase/Propio/"
"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/130.0.0.0 Safari/537.36"**

Puntos clave a recordar:

- ☐ **Orden:** Las llamadas a **header()** deben realizarse antes de cualquier otro contenido HTML o PHP, ya que una vez que se envía cualquier salida, no se pueden enviar más cabeceras.
- ☐ **Codificación:** Utilizar la codificación correcta para evitar problemas de visualización de caracteres especiales.
- ☐ **Cookies:** Las cookies se utilizan para almacenar información en el lado del cliente.
- ☐ **Cabeceras personalizadas:** Puedes definir tus propias cabeceras para transmitir información específica a tu aplicación o a terceros.

El ejemplo siguiente muestra un ejemplo de cabecera HTTP, concretamente una redirección. En el ejemplo, la página 1 contiene un enlace a la página 2. Pero como la página genera una cabecera HTTP de redirección a la página 3, al hacer clic en el enlace se muestra directamente la página 3.

cabeceras-header-1-1.php

```
<p>Esta es la página 1.</p>
<p><a href="cabeceras-header-1-2.php">Enlace a la página 2 (que
redirige a la página 3)</a></p>
```

cabeceras-header-1-2.php

```
<?php
header("Location: cabeceras-header-1-3.php");
print "<p>Esta es la página 2</p>";
print "<p>La redirección <strong>NO</strong> se ha
realizado</p>";
print "<p><a href=\"cabeceras-header-1-1.php\">Volver a la página
1</a></p>";
?>
```

cabeceras-header-1-3.php

```
<p>Esta es la página 3.</p>
```

```
<p>La redirección <strong>Sí</strong> se ha realizado.</p>

<p><a href="cabeceras-header-1-1.php">Volver a la página

1</a></p>
```

Nota: Si el programa contiene un error antes de la función `header()`, PHP generará un aviso de error que es también parte de la salida del programa y eso **provocaría el fallo de la función `header()` posterior**.

Redirecciones: `header("Location:...")`

- ☐ La función `header()` se puede utilizar para **redirigir automáticamente a otra página**, enviando como **argumento** la **cadena `Location:` seguida de la dirección absoluta o relativa de la página** a la que queremos redirigir.
- ☐

Ejemplos,

Redireccionamiento tras un formulario: *Un usuario llena un formulario de contacto y al enviar, se le redirige a una página de agradecimiento.*

Si el formulario se envía (método POST), el usuario es redirigido a la página `gracias.php`. Esto evita que el usuario vuelva a enviar el formulario accidentalmente al recargar la página.

```
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    // Procesar los datos del formulario
    // ...

    header("Location: gracias.php");
    exit;
}
```

Redireccionamiento después de un proceso: *Un usuario inicia sesión y es redirigido al panel de control.*

Si las credenciales son correctas, el usuario es redirigido al panel de control.

```
if (// Verificar credenciales) {
    // Iniciar sesión
    $_SESSION['usuario'] = $usuario;
    header("Location: panel_control.php");
    exit;
} else {
    // Mostrar mensaje de error
```

```
}
```

3. Redireccionamiento 301 para SEO: Se ha cambiado la URL de una página y se quiere redirigir permanentemente el tráfico antiguo a la nueva URL.

Explicación: El código 301 indica a los motores de búsqueda que la página se ha movido de forma permanente, ayudando a mantener el SEO.

```
header("HTTP/1.1 301 Moved Permanently");  
header("Location: https://nuevo-dominio.com/nueva-pagina");  
exit;
```

Redireccionamiento condicional: Redirigir a usuarios no registrados a una página de registro.

Si el usuario no está autenticado, se le redirige a la página de registro.

```
if (!isset($_SESSION['usuario'])) {  
    header("Location: registro.php");  
    exit;  
}
```

Redireccionamiento basado en el navegador; Este código redirige a los usuarios móviles a una versión optimizada para móviles de la página.

Obtiene el agente de usuario del navegador del visitante. Y lo redirige a la página optimizada para su navegación.

```
$user_agent = $_SERVER['HTTP_USER_AGENT'];  
if (strpos($user_agent, 'Mobile') !== false) {  
    header('Location: version_movil.php');  
    exit;  
} else {  
    header('Location: version_escritorio.php');  
    exit;  
}
```

- ☐ Si además de redirigir a una página, se quieren enviar controles a dicha página, se pueden añadir a la cadena separando los controles con el carácter &.

[cabeceras-header-1-2.php](#)

```
<?php  
  
header('Location: cabeceras-header-1-3.php?busqueda=coches&categoria=deportivos&pagina=2');  
print "<p>Esta es la página 2</p>";  
print "<p>La redirección <strong>NO</strong> se ha
```

```

realizado</p>";
print "<p><a href=\"cabeceras-header-1-1.php\">Volver a la página
1</a></p>";
?>

```

cabeceras-header-1-3.php

```

<?<?php
print_r($_REQUEST);
print_r(nl2br("\n"));
print_r($_GET);
echo "<p>Esta es la página 3.</p>
<p>La redirección <strong>SÍ</strong> se ha realizado.</p>";
header('cabeceras-header-1-1.php');
echo "<p>Volver a la página 1</p>";
?>

```

- ☐ Si el valor a enviar contiene espacios, se pueden escribir caracteres + o espacios para separar las palabras:

```

<?php
header("Location: cabeceras-header-1-3.php?nombre=Pepe+Sol&edad=25
");
?>

```

Resto del programa tras la redirección

- ☐ La ejecución de un programa no se detiene al encontrar una redirección, sino que PHP ejecuta el programa hasta el final. **Dependiendo de las instrucciones que haya tras la dirección, el resultado puede ser relevante o no.**
- ☐ Si se escriben varias redirecciones, se aplicará la última. En el ejemplo siguiente, la página redirigirá a “cabeceras-header-1-4.php”.

cabeceras-header-1-2.php

```

<?php
header("Location: cabeceras-header-1-3.php");
header("Location: cabeceras-header-1-4.php");
print "<p>Esta es la página 2</p>";
print "<p>La redirección <strong>NO</strong> se ha
realizado</p>";
print "<p><a href=\"cabeceras-header-1-1.php\">Volver a la página
1</a></p>";
?>

```

cabeceras-header-1-4.php

```
<?php
print_r($_REQUEST);
print_r(nl2br("\n"));
print_r($_GET);
echo "<p>Esta es la página 4.</p>
<p>La redirección <strong>Sí</strong> se ha realizado.</p>";
//header('Location: cabeceras-header-1-1.php');
?>
<a href="cabeceras-header-1-1.php"> ir a otra página </a>
```

- ☐ Si después de la redirección se genera texto, ese texto **no** llegará al usuario, puesto que la redirección le llevará antes a otra página
- ☐ Si el programa realiza acciones como **modificar registros en una base de datos, borrar archivos, etc.**, esas acciones **sí** que se realizan.
- ☐ Si tras una redirección no queremos que se ejecute el resto del programa, tenemos dos opciones:
 - ☐ Podemos utilizar una expresión if ... else ...
 - ☐ Podemos utilizar la instrucción exit, que detiene el programa en ese punto
 - ☐ **exit no es una función**, sino una palabra reservada del lenguaje, **pero también se puede utilizar** seguida de paréntesis **como una función**: exit().
 - ☐ Se puede incluso incluir un parámetro que puede ser un entero entre 0 y 255 o una cadena:
 - ☐ Si el parámetro es un valor numérico, este valor se utilizará como estado de salida y no se imprimirá. Los estados de salida deben estar en el rango de 0 a 254, el estado de salida 255 está reservado por PHP y no se utilizará. El estado 0 se utiliza para finalizar el programa con éxito.
 - ☐ Si el valor es una cadena, se imprime su valor antes de terminar.
- ☐ Pero **hay que tener en cuenta que exit detiene no sólo la ejecución del programa, sino también la de los programas que hubieran llamado a esos programas**

EJEMPLO CON IF

```
<?php
    if (condicion) {
        header("Location: https://www.google.com");
    } else {
        ...
    }
?>
```


EJEMPLO CON EXIT

(cabeceras-header-1-2.php)

```
<?php
    header("Location: cabeceras-header-1-3.php");
    exit;
    header("Location: cabeceras-header-1-4.php");
    print "<p>Esta es la página 2</p>";
    print "<p>La redirección <strong>NO</strong> se ha
    realizado</p>";
    print "<p><a href='\"cabeceras-header-1-1.php\"'>Volver a la página
    1</a></p>";
?>
```

Crear con PHP otros tipos de archivos

- ☐ Con instrucciones print, PHP puede generar archivos con cualquier contenido (archivos de texto o incluso binarios), pero **para que el navegador acepte esos archivos** y los interprete correctamente, en **las cabeceras se debe incluir el tipo MIME correspondiente**.
- ☐ Existen muchos tipos MIME (véase lista de tipos MIME en MDN)

extensión del archivo	tipo de archivo	tipo MIME
.html	página web HTML	text/html
.css	hoja de estilo CSS	text/css
.js	JavaScript	application/js
.svg	SVG	image/svg+xml
.json	JSON	application/json

- ☐ El **tipo MIME se indica en la cabecera**, por lo que un programa PHP puede declarar el tipo MIME **mediante la función header()**, enviando como **argumento la cadena Content-type: seguida del tipo MIME correspondiente**.

Ejemplo

Cuando se ejecuta este script, el navegador recibirá un archivo de texto llamado "ejemplo.txt" con el contenido especificado. El navegador interpretará el contenido como texto plano y lo mostrará o permitirá al usuario guardarlo.

```
<?php
// Generamos un archivo de texto simple
$texto = "Este es el contenido de un archivo de texto generado por
PHP.";

// Definimos el tipo MIME como texto plano
header('Content-Type: text/plain');

// Indicamos al navegador que descargue el archivo con un nombre
específico
header('Content-Disposition: attachment; filename="ejemplo.txt"');

// Imprimimos el contenido del archivo
print $texto;
?>
```

Crear hojas de estilo CSS

- ☐ Para que el navegador aplique la hoja de estilo, el **programa debe generar una cabecera que declare el tipo MIME de hoja de estilos text/css**, como muestra el siguiente ejemplo. CUIDADO *el servidor asigna por defecto el tipo MIME text/html a los ficheros generados por un programa PHP*

```
<link rel="stylesheet" href="mime-css.php" title="Color">
```

mime-css.php

```
<?php
header("Content-type: text/css");
$c1 = rand(0, 255);
$c2 = rand(0, 255);
$c3 = rand(0, 255);
print "body{";
print " background-color: rgb($c1, $c2, $c3);";
print " color: rgb($c2, $c3, $c1);";
print "}";
?>
```

Crear imágenes SVG

- ☐ El programa siguiente crea las etiquetas correspondientes a una imagen SVG, concretamente a un cuadrado de color aleatorio. Si escribimos directamente la url de ese programa en el navegador, el navegador muestra el contenido esperado.

```
<?php
$c1 = rand(0, 255);
$c2 = rand(0, 255);
$c3 = rand(0, 255);
print "<svg version=\"1.1\"
xmlns=\"http://www.w3.org/2000/svg\" \" \" . \" width=\"100\"
height=\"100\" viewBox=\"-5 -5 100 100\">";
print " <rect fill=\"rgb($c1, $c2, $c3)\" x=\"0\" y=\"0\"
width=\"90\" height=\"90\" />";
print "</svg>";
?>
```

- ☐ Aunque el navegador consiga mostrar el cuadrado, realmente no estamos haciendo bien las cosas, en otras circunstancias, este mismo programa no daría el resultado esperado.

- ☐ Como muestra el siguiente ejemplo, si el programa se llama en una etiqueta, el navegador no puede mostrar la imagen. El problema es que en una etiqueta el navegador tiene que recibir obligatoriamente algún tipo MIME de imagen (jpg, png, svg, etc.).



```
<p></p>
```

```
<?php
$c1 = rand(0, 255);
$c2 = rand(0, 255);
$c3 = rand(0, 255);
print "<svg version=\"1.1\"
xmlns=\"http://www.w3.org/2000/svg\" \" \" . \" width=\"100\"
height=\"100\" viewBox=\"-5 -5 100 100\">";
print " <rect fill=\"rgb($c1, $c2, $c3)\" x=\"0\" y=\"0\"
width=\"90\" height=\"90\" />";
print "</svg>";
?>
```

- ☐ Una imagen SVG debería entregarse al navegador con el tipo MIME image/svg+xml.

```
<p></p>
```

```
<?php
```

```

header("Content-type: image/svg+xml");
$c1 = rand(0, 255);
$c2 = rand(0, 255);
$c3 = rand(0, 255);
print "<svg version=\"1.1\"
xmlns=\"http://www.w3.org/2000/svg\" " . " width=\"100\"
height=\"100\" viewBox=\"-5 -5 100 100\">";
print " <rect fill=\"rgb($c1, $c2, $c3)\" x=\"0\" y=\"0\"
width=\"90\" height=\"90\" />";
print "</svg>";
?>

```

SESIONES

MOTIVACIÓN

- ☐ Una de las **limitaciones de las páginas web** es que **cada página web es un documento independiente**. Eso hace que **dos programas PHP no puedan, en principio, compartir información**. (Por compartir información nos estamos refiriendo a utilizar variables comunes en programas distintos, sin que la información salga del servidor.)
- ☐ Podemos **enviar información de una página a otra**:
 - ☐ **A través de un formulario**→ se envía de la información que ha introducido el usuario en ese formulario. Y adicionalmente, podemos enviar información mediante controles ocultos
 - ☐ Podemos "imitar" a un formulario **redirigiendo a otra página enviando información en la dirección con la función header**
- ☐ **Ninguno de los dos métodos se puede considerarse compartir información**, puesto que la página que recibe la información no puede saber si la información ha sido manipulada por el camino ni quién se la envía.

programa.php

```

<?php
    $nombre = "Antonio";
    print "<p>El nombre es $nombre</p>";
?>

```

programa2.php

```

<p>El nombre es Antonio</p>
<?php
    print "<p>El nombre es $nombre</p>";
?>

```

SALIDA

Notice: Undefined variable: nombre in ejemplo.php on line 2

El nombre es

- ☐ En PHP se puede superar esta limitación mediante las sesiones. **Las sesiones permiten que páginas distintas** (en el mismo servidor) **puedan acceder a una variable común, la matriz \$_SESSION.**



programa1.php

```
<?php
```

```
    session_start();
```

```
    $_SESSION["nombre"] = "Antonio";
```

```
    print "<p>El nombre es $_SESSION[nombre]</p>";
```

```
?>
```

salida

```
<p>El nombre es Antonio</p>
```

programa2.php

```
<?php
```

```
    session_start();
```

```
    print "<p>El nombre es $_SESSION[nombre]</p>";
```

```
?>
```

salida

```
<p>El nombre es Antonio</p>
```

- ☐ **Las sesiones no deben confundirse con las cookies.** Las cookies es un método que permite **guardar información en el ordenador del cliente** para recuperarla en el futuro, mientras que en las sesiones la información **se mantiene en el servidor** hasta que se cierra la sesión (por intervención del usuario o por tiempo).

Partes del trabajo con sesiones

- ☐ Creación o apertura de la sesión

- ☐ Cuando alguna *página crea una sesión* utilizando la función correspondiente, el servidor asocia al navegador del usuario un identificador de usuario único. El identificador se *guarda en el usuario en forma de cookie* o, si el navegador del usuario no permite la creación de cookies, *añadiendo el identificador en la dirección de la página.*

- ☐ Las **sesiones se crean** mediante la función **session_start()**.

- ☐ Si la sesión no existía, esta función crea la sesión y le asocia un identificador de sesión único.

- ☐ Si la sesión ya existía, esta función permite que la página tenga acceso a la información vinculada a la sesión. Es decir, **todas las páginas que quieran guardar datos en \$_SESSION o leer datos de \$_SESSION deben comenzar con la función session_start().**

☐ Consideraciones importantes:

☐ Seguridad:

- ☐ Los SID deben ser generados aleatoriamente y ser lo suficientemente largos para evitar ataques de fuerza bruta.
- ☐ Los datos de sesión deben ser almacenados de forma segura en el servidor.
- ☐ Nunca almacenar información sensible directamente en cookies.

☐ Duración de la sesión:

- ☐ Las sesiones pueden tener una duración predeterminada (por ejemplo, 30 minutos) o pueden ser configuradas para persistir hasta que el usuario cierre el navegador.

☐ Manejo de múltiples dispositivos:

- ☐ Si un usuario accede a un sitio web desde diferentes dispositivos, es necesario manejar las sesiones de forma adecuada para evitar conflictos.

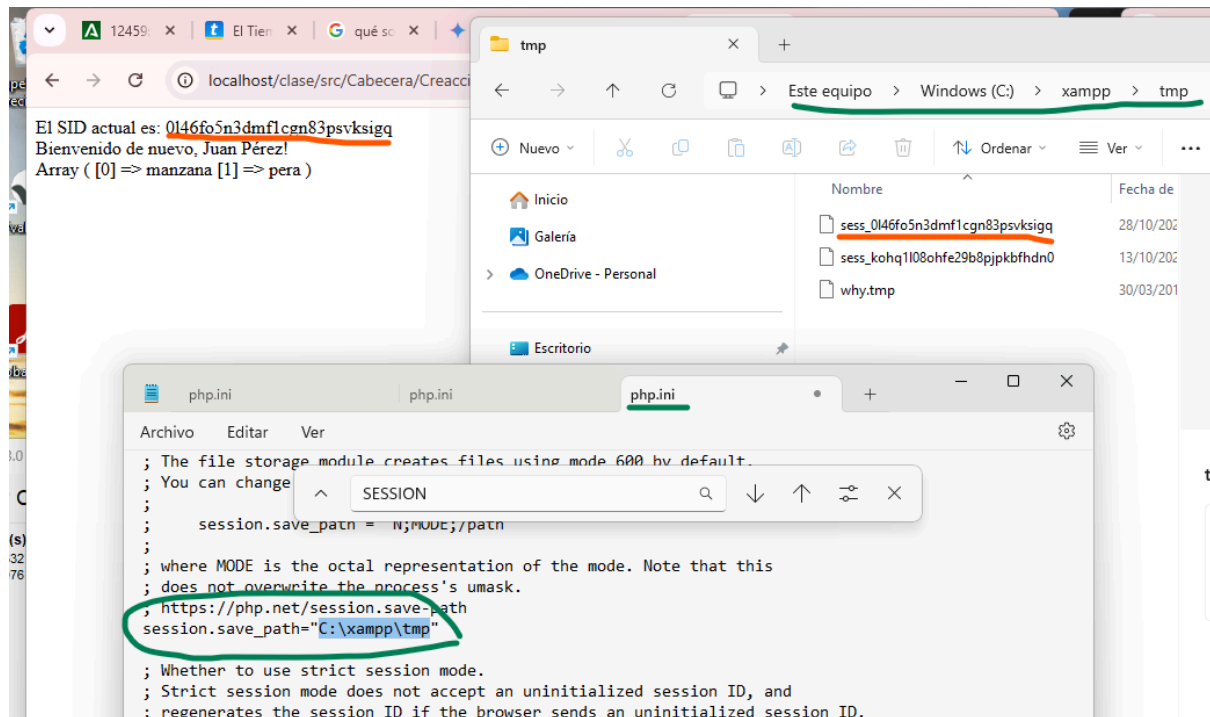
```
<?php
// Iniciar una sesión
session_start();
echo "El SID actual es: " . session_id();

// Comprobar si la sesión ya existe
if (!isset($_SESSION['usuario'])) {
    // Si no existe, crearla y establecer variables de sesión
    $_SESSION['usuario'] = 'Juan Pérez';
    $_SESSION['carrito'] = array('manzana', 'pera');
    print_r (nl2br("\n Creo la sesión de usuario, " .
$_SESSION['usuario']. " \n")) ;
} else {
    // Si existe, recuperar los datos de la sesión
    print_r(nl2br( "\n Bienvenido de nuevo, " . $_SESSION['usuario'] .
"! \n"));
    print_r($_SESSION['carrito']);
}
?>
```

☐ El SID (Session ID) no se muestra explícitamente, pero está implícito en la función `session_start()`.

- ☐ **Genera (o recupera) un SID único:** Este SID es una cadena de caracteres aleatoria que identifica de forma única la sesión del usuario.
- ☐ **Almacena el SID:** Lo guarda en una cookie en el navegador del cliente o en la URL (si las cookies están deshabilitadas).

- ☐ **Crea un espacio de almacenamiento de variables de sesión:** Este espacio es un array asociativo (en PHP) donde se almacenan los datos de la sesión.
- ☐ ¿Dónde está físicamente el SID?
 - ☐ **Archivo de sesión:** PHP almacena los datos de la sesión (incluyendo el SID y su información asociada) en un archivo en el servidor. La ubicación de estos archivos está configurada en el archivo **php.ini**.
 - ☐ **Base de datos:** En algunas configuraciones, los datos de sesión pueden almacenarse en una base de datos.



- ☐ **Utilización de la sesión**
 - ☐ Si ya se ha creado la sesión, las páginas solicitadas por el mismo navegador pueden guardar y recuperar información en el servidor, información que se asocia al identificador de usuario, por lo que no es accesible a otros usuarios. La información se conserva hasta que el usuario o el servidor destruyan la sesión.
 - ☐ Cuando una página ha creado una sesión o ha accedido a una sesión ya existente mediante `session_start()`, la página tiene acceso a la matriz `$_SESSION` que contiene las variables de esa sesión.
 - ☐ La matriz `$_SESSION` es una **matriz asociativa en la que se pueden definir valores como en cualquier otra matriz**. La diferencia es que `$_SESSION` es **accesible desde páginas diferentes** (siempre que esas páginas tengan asociada la misma sesión), *manteniéndose los valores de una página a otra*.

```
<?php
    session_start();
    $_SESSION["nombre"] = "Antonio";
    print "<p>El nombre es $_SESSION['nombre']</p>";
```

```
?>
```

```
<p>El nombre es Antonio</p>
```

```
<?php
    session_start();
    print "<p>El nombre es $_SESSION['nombre']</p>";
?>
```

```
<p>El nombre es Antonio</p>
```

- ☐ Los nombres de los **primeros índices de la matriz \$_SESSION** tienen que **cumplir las mismas reglas que los nombres de las variables**, es decir, que el **primer carácter debe ser una letra o un guión bajo** (_). En particular, no deben ser números ni contener caracteres no alfanuméricos.

```
<?php
    session_start();
    $_SESSION[] = "Antonio";
    print "<p>Se ha guardado su nombre.</p>";
    print "<pre>";
    print_r($_SESSION);
    print "</pre>";
```

```
?>
```

```
<p>Se ha guardado su nombre</p>
```

Notice: Unknown: Skipping numeric key 0 in Unknown on line 0

☐ IMPORTANTE

- ☐ **Seguridad: Siempre debes validar y sanear los datos** que se **almacenan en las sesiones** para evitar vulnerabilidades como la inyección de código.

```
<?php
session_start();

// Función para sanear una cadena
function sanear_cadena($cadena) {
    // Eliminar etiquetas HTML
    $cadena = strip_tags($cadena);
    // Convertir caracteres especiales a entidades HTML
    $cadena = htmlspecialchars($cadena, ENT_QUOTES);
    // Eliminar espacios en blanco al principio y al final
    $cadena = trim($cadena);

    return $cadena;
}

// Ejemplo de recepción y almacenamiento de datos
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
```



```

$nombre = sanear_cadena($_POST['nombre']);
$edad = filter_var($_POST['edad'], FILTER_SANITIZE_NUMBER_INT);

// Validación adicional (opcional)
if (!is_numeric($edad) || $edad < 0) {
    echo "La edad debe ser un número positivo.";
} else {
    $_SESSION['usuario'] = [
        'nombre' => $nombre,
        'edad' => $edad
    ];
    echo "Datos almacenados en la sesión.";
}
}

```

- ☐ **Duración de la sesión:** La duración de una sesión **puede configurarse en el archivo `php.ini` o mediante funciones como `session_set_cookie_params()`.**

- ☐ **`session.gc_maxlifetime`;** establece el tiempo máximo de vida de un archivo de sesión en segundos.

Cuando se supera este tiempo, el archivo de sesión se considera "basura" y es **candidato a ser eliminado** por el garbage collector de PHP. Sin embargo, **la eliminación no es inmediata y depende de otros factores como la probabilidad de ejecución del garbage collector.**

Afecta al tiempo de vida del archivo de sesión en el servidor.

- ☐ **`session.cookie_lifetime`;** define el tiempo de vida de la cookie de sesión en segundos.

Si esta cookie expira, el cliente (navegador) ya no enviará el SID (Session ID) al servidor, lo que equivale a finalizar la sesión desde el punto de vista del cliente.

Puedes utilizar la función `session_set_cookie_params()` para establecer el tiempo de vida de la cookie de sesión desde tu código PHP.

Afecta al tiempo de vida de la cookie de sesión en el cliente.

PTE DE EJMEPLO

```
<
```

- ☐ **Cierre automático:** Puedes configurar la sesión para que se cierre automáticamente después de un período de inactividad.

```

<?php
ini_set('session.gc_maxlifetime', 5);

```

```

session_start();

// Configurar el tiempo de vida de la sesión en segundos (por ejemplo,
5 SEGUNDOS)

// Marcar el momento en que el usuario estuvo activo por última vez
if (!isset($_SESSION['last_activity'])) {
    $_SESSION['last_activity'] = time();
}

// Comprobar si ha transcurrido el tiempo de inactividad
$max_idle_time = 5; // 5 segundos
if (isset($_SESSION['last_activity']) && (time() -
$_SESSION['last_activity'] > $max_idle_time) {
    // Destruir la sesión si ha transcurrido el tiempo de inactividad
    session_unset();
    session_destroy();
    echo "Tu sesión ha expirado por inactividad.";
} else {
    // Actualizar el momento de la última actividad
    $_SESSION['last_activity'] = time();
}
?>

```

☐ Destrucción o cierre de la sesión

- ☐ Cerrar una sesión **es destruir la matriz \$_SESSION y el identificador de la sesión (SID).**
- ☐ Tanto el usuario como el servidor pueden cerrar la sesión. El usuario puede destruir la sesión cerrando el navegador. El servidor puede destruir la sesión cuando alguna página utilice la función correspondiente o al cabo de un tiempo determinado (definido mediante la función `session_set_cookie_params()`).

```

<?php
session_start();
echo "El SID actual es: " . session_id();
session_destroy();
print_r (nl2br("\n Sesión cerrada correctamente \n"));
?>

```

- ☐ Las sesiones se pueden cerrar de varias maneras:

- ☐ El usuario puede cerrar la sesión simplemente **cerrando el navegador** (no basta con cerrar las pestañas).
- ☐ Un programa puede cerrar la sesión **mediante la función `session_destroy()`**.
- ☐ En general, las cookies tienen una duración establecida en la directiva **`session.cookie_lifetime`** (y el servidor puede borrar la información cuando ha pasado el tiempo indicado en segundos en la directiva **`session.gc_maxlifetime`**), pero la duración de una sesión en particular puede establecerse en el momento de su creación mediante la función **`session_set_cookie_params()`** (tiempo que se puede modificar posteriormente).
- ☐ Cuando se destruye una sesión, el **programa que ha destruido la sesión sigue teniendo acceso a los valores de `$_SESSION` creados antes de la destrucción** de la sesión, pero **las páginas siguientes no**.

```
<?php
session_start();
$_SESSION["nombre"] = "Antonio";
session_destroy();
if (isset($_SESSION["nombre"])) {
    print "<p>Su nombre es $_SESSION[nombre].</p>";
} else {
    print "<p>No sé su nombre.</p>";
}
```

?>

<p>Su nombre es Antonio</p>

```
<?php
session_start();
if (isset($_SESSION["nombre"])) {
    print "<p>Su nombre es $_SESSION[nombre].</p>";
} else {
    print "<p>No sé su nombre.</p>";
}
```

?>

<p>No sé su nombre.</p>

Nombre de sesión

- ☐ Cuando el navegador se conecta a un servidor, la sesión es única, es decir, todas las páginas del mismo dominio compartirán la misma matriz `$_SESSION`.
- ☐ La función **`session_name()`** permite **establecer un nombre de sesión específico**, de manera que **todas las páginas que declaren el mismo nombre de sesión accederán a la misma matriz `$_SESSION`**.
- ☐ El nombre de sesión **distingue entre minúsculas y mayúsculas**.

- ☐ El nombre de la sesión **no puede contener** únicamente números, ni tampoco puede contener los caracteres espacio (), punto (.), ampersand (&), más (+), corchete izquierdo ([]) ni almohadilla (#).

```
<?php
    session_name("ejemplo1");
    session_start();
    $_SESSION["nombre"] = "Antonio";
    print "<p>El nombre es $_SESSION[nombre]</p>";
```

```
?>
```

<p>El nombre es Antonio</p>

```
<?php
    session_name("ejemplo2");
    session_start();
    $_SESSION["nombre"] = "Juan";
    print "<p>El nombre es $_SESSION[nombre]</p>";
```

```
?>
```

<p>El nombre es Juan</p>

EJEMPLO

```
<?php
// Página de inicio de la tienda
session_name('tienda');
session_name('foro');
session_start();

// Almacenar un producto en el carrito de la tienda
$_SESSION['carrito'][] = 'Manzana';
if (isset($_SESSION['carrito'])) {
    echo "Contenido del carrito de la tienda: ";
    print_r($_SESSION['carrito']);
}

// Almacenar un mensaje en el foro
$_SESSION['mensajes'][] = 'Hola, soy nuevo en el foro';

// Mostrar los mensajes del foro
echo "<br>Último mensaje en el foro: " . $_SESSION['mensajes'][0];

?>
```

Borrar elementos de la sesión

- ☐ Los valores de `$_SESSION` se borran **como en cualquier otra matriz** mediante la función **`unset()`**.

```
<?php
    session_start();
    $_SESSION["nombre"] = "Antonio";
    print "<p>Su nombre es $_SESSION[nombre]</p>";
    unset($_SESSION["nombre"]);
if (isset($_SESSION["nombres"])) {
    print "<p>Su nombre es $_SESSION[nombre].</p>";
} else {
    print "<p>No sé su nombre.</p>";
}
?>
<p>Su nombre es Antonio.</p>
<p>No sé su nombre.</p>
```

- ☐ Para borrar todos los valores de `$_SESSION` se pueden borrar uno a uno o utilizar la función **`session_unset()`**, pero normalmente **no se debe utilizar `unset()`**:
 - ☐ **`unset($_SESSION)`** impide que el resto de la página escriba o lea valores en `$_SESSION`, pero la sesión conserva los valores, por lo que otras páginas seguirán viendo esos valores.

```
<?php
    session_start();
    $_SESSION["nombre"] = "Antonio";
    $_SESSION["apellidos"] = "López";
    $a = &$_SESSION;
    unset($_SESSION);
    $a["apellidos"] = "García"; // $a hacer referencia a $_SESSION original
    $_SESSION["apellidos"] = "Pérez";
    header("Location: pagina.php");
?>
```

[pagina.php](#)

```
<?php
    session_start();
    $_SESSION["saludo"] = "Hola";
    print "<pre>";
    print_r($_SESSION);
    print "</pre>";
?>
<pre>
Array
(
    [nombre] => Antonio
    [apellidos] => García
```

```
[saludo] => Hola
)
</pre>
```

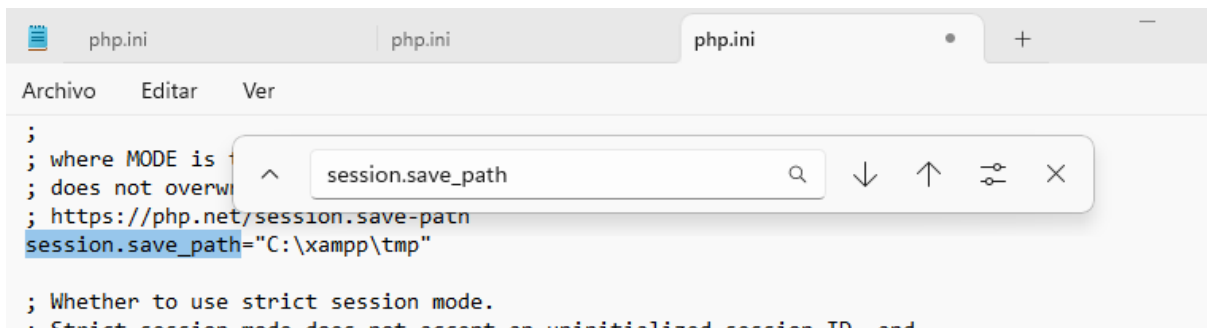
- ☐ **session_unset()** borra todos los valores pero **permite que el resto de la página (y otras páginas) escriba o lea valores en \$_SESSION.**

Directiva session.save_handler

- ☐ Por defecto, PHP almacena los datos de sesión en archivos en el sistema de archivos del servidor. Esto significa que cada sesión se guarda como un archivo individual en un directorio específico, configurado a través de la directiva **session.save_path**.
- ☐ Para utilizar sesiones mediante el mecanismo propio de PHP, sin necesidad de crear funciones propias, la directiva **session.save_handler** del **archivo de configuración php.ini** debe tener el valor **files**.
- ☐ Esta configuración es la más habitual, pero **algunos gestores de contenidos (CMS)** tienen sus propias funciones de gestión de sesiones y **requieren que esta directiva tome el valor user**.
- ☐ Si se ha necesitado modificar php.ini, pero queremos ejecutar otros programas que no incluyen sus propias funciones de gestión de sesiones, se puede incluir en las páginas PHP la función ini_set() antes de abrir la sesión. Ese cambio sólo afectará a la página que incluya la llamada a la función.

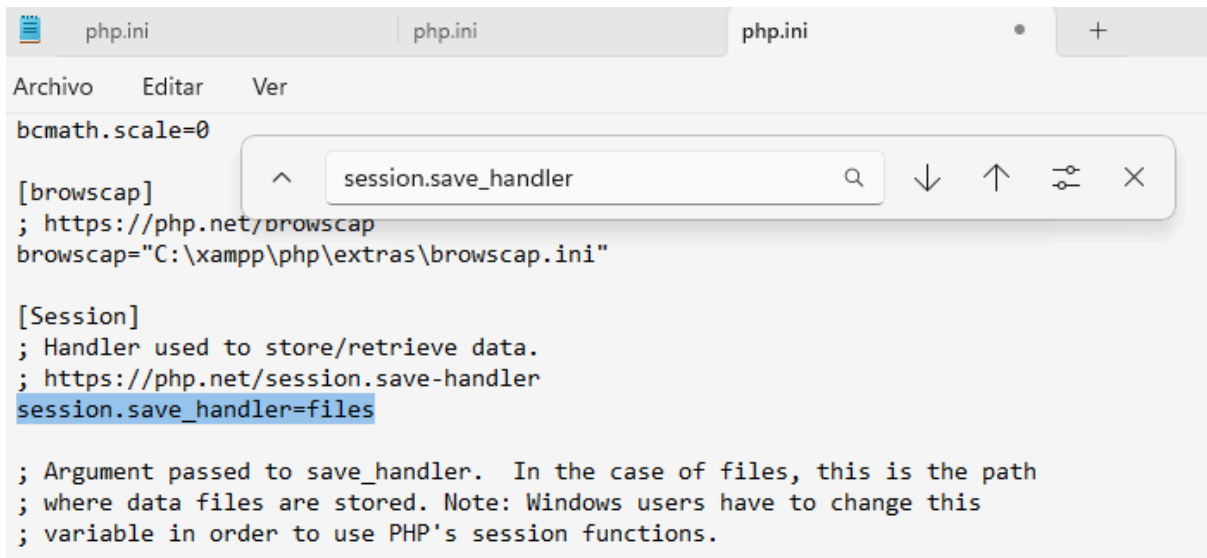
```
<?php
ini_set("session.save_handler", "files"); // Necesario únicamente cuando
session.save_handler = user en php.ini
session_start();
?>
```

- ☐ **vENTAJAS**
 - ☐ **Flexibilidad**: Puedes elegir **almacenar los datos de sesión en bases de datos, memcached, Redis o cualquier otro** sistema de almacenamiento que prefieras, dependiendo de tus necesidades y del rendimiento requerido.
 - ☐ **Escalabilidad**: Para aplicaciones de alto tráfico, almacenar las sesiones en una base de datos puede ofrecer mejor rendimiento y escalabilidad.
 - ☐ **Seguridad**: Almacena los datos de sesión en un lugar más seguro, como una base de datos con acceso restringido.
 - ☐ **Persistencia**: Puedes personalizar la forma en que se almacenan y recuperan los datos de sesión para satisfacer requisitos específicos de tu aplicación.



```
; where MODE is
; does not overw
; https://php.net/session.save-path
session.save_path="C:\xampp\tmp"

; Whether to use strict session mode.
; Strict session mode does not accept an uninitialized session ID, and
```



```
bcmath.scale=0

[browscap]
; https://php.net/browscap
browscap="C:\xampp\php\extras\browscap.ini"

[Session]
; Handler used to store/retrieve data.
; https://php.net/session.save-handler
session.save_handler=files

; Argument passed to save_handler. In the case of files, this is the path
; where data files are stored. Note: Windows users have to change this
; variable in order to use PHP's session functions.
```

```
<?php
$servername = "localhost";
$username = "tu_usuario";
$password = "tu_contraseña";
$dbname = "tu_base_de_datos";

// Función para abrir la conexión a la base de datos
function abrir_sesion() {
    global $servername, $username, $password, $dbname;

    //$conn = new mysqli($servername, $username, $password, $dbname);
    if ($conn->connect_error) {
        die("Connection failed: " . $conn->connect_error);
        return $conn;
    }
}

// Función para cerrar la conexión a la base de datos
function cerrar_sesion($conn) {
    $conn->close();
}
```

```

}

// Función para leer los datos de la sesión desde la base de datos
function leer_sesion($id_sesion) {
    $conn = abrir_sesion();
    $sql = "SELECT datos_sesion FROM sesiones WHERE id = '$id_sesion'";
    $result = $conn->query($sql);

    if ($result->num_rows > 0) {
        $row = $result->fetch_assoc();
        return unserialize($row["datos_sesion"]);
    } else {
        return "";
    }
    cerrar_sesion($conn);
}

// Función para escribir los datos de la sesión en la base de datos
function escribir_sesion($id_sesion, $datos) {
    $conn = abrir_sesion();
    $datos_serializados = serialize($datos);
    $sql = "INSERT INTO sesiones (id, datos_sesion) VALUES
('$id_sesion', '$datos_serializados')
ON DUPLICATE KEY UPDATE datos_sesion =
'$datos_serializados'";
    $conn->query($sql);
    cerrar_sesion($conn);
}

// Función para destruir los datos de la sesión
function destruir_sesion($id_sesion) {
    $conn = abrir_sesion();
    $sql = "DELETE FROM sesiones WHERE id = '$id_sesion'";
    $conn->query($sql);
    cerrar_sesion($conn);
}

// Función para limpiar las sesiones caducadas
function limpiar_sesion($maxlifetime) {
    $conn = abrir_sesion();
    $tiempo_actual = time();
    $sql = "DELETE FROM sesiones WHERE timestamp < $tiempo_actual -
$maxlifetime";

```



```

$conn->query($sql);
cerrar_sesion($conn);
}
// Configurar el manejador de sesiones para utilizar una base de datos
session_set_save_handler(
    'abrir_sesion', // Función para abrir la conexión a la base de
datos
    'cerrar_sesion', // Función para cerrar la conexión a la base de
datos
    'leer_sesion',   // Función para leer los datos de la sesión desde
la base de datos
    'escribir_sesion', // Función para escribir los datos de la sesión
en la base de datos
    'destruir_sesion', // Función para destruir los datos de la sesión
    'limpiar_sesion' // Función para limpiar las sesiones caducadas
);
session_start();
// Ejemplo de uso
$_SESSION['nombre'] = "Juan";
$_SESSION['edad'] = 30;
?>

```

Seguridad en las sesiones

- ☐ Algunas técnicas sencillas para evitar riesgos con las sesiones:

Sesión Time-Outs.

- ☐ Establecer un tiempo de vida a las sesiones. Es algo importante para lidiar con las sesiones de usuarios en aplicaciones. Si **un usuario inicia sesión en un lugar y se olvida de cerrarla, otros pueden continuar con la misma**, por eso es mejor establecer un tiempo máximo de sesión:

```

<?php
    session_start();
// Establecer tiempo de vida de la sesión en segundos
    $inactividad = 60;
// Comprobar si $_SESSION["timeout"] está establecida
    if(isset($_SESSION["timeout"])){
// Calcular el tiempo de vida de la sesión (TTL = Time To Live)
        $sessionTTL = time() - $_SESSION["timeout"];
if($sessionTTL > $inactividad){
            session_destroy();
            header("Location: fin.php");
        }
}

```

```

}
// La siguiente clave se crea cuando se inicia sesión (llamada a la página)
$_SESSION["timeout"] = time();
?>

```

Regenerar el Session ID.

- ☐ La función **session_regenerate_id()** crea un nuevo ID único para representar la sesión actual del usuario. Esto se debe realizar cuando se realizan acciones importantes como logearse o modificar los datos del usuario. Darle a las sesiones un nuevo ID reduce la probabilidad de ataques session hijacking

Cuándo regenerar el ID de sesión:

- ☐ **Después de que el usuario inicie sesión:** Esto es fundamental para evitar ataques de fijación de sesión.

```

<?php
session_start();

// ... (tu código existente)

// Regenerar el ID de sesión después de que el usuario inicie sesión
if (isset($_POST['login']) && $_POST['username'] === 'usuario' &&
$_POST['password'] === 'contraseña') {
    // ... (lógica de autenticación)

    // Regenerar el ID de sesión, preservando los datos existentes
    session_regenerate_id(true);

    // Redirigir al usuario a la página de inicio
    header('Location: index.php');
    exit;
}

```

- ☐ **Después de cambiar la contraseña del usuario:** Para invalidar cualquier sesión existente que pudiera haber sido comprometida.

```

<?php
session_start();

// ... (resto de tu código)

if (isset($_POST['nueva_contraseña']) && $_POST['nueva_contraseña'] ===
$_POST['confirmar_contraseña']) {
    // Verificar si la nueva contraseña cumple con los requisitos de
seguridad

```

```
// Actualizar la contraseña en la base de datos
// ... (código para actualizar la contraseña en la base de datos)

// Regenerar el ID de sesión
session_regenerate_id(true);

// Mostrar un mensaje de éxito al usuario
echo "Contraseña actualizada correctamente.";
}
```

- ☐ **Periódicamente:** Puedes regenerar el ID de sesión cada cierto tiempo para aumentar aún más la seguridad.

```
<?php
// Regenerar el ID de sesión cada 30 minutos
if (isset($_SESSION['last_regenerated']) && time() -
$_SESSION['last_regenerated'] > 1800) {
    session_regenerate_id(true);
    $_SESSION['last_regenerated'] = time();
}
```

Destruir la sesión.

- ☐ Es importante utilizar `session_destroy()` cuando ya no se vaya a hacer uso de la sesión

Utilizar almacenamiento permanente.

- ☐ **Utilizar una base de datos para almacenar los datos** que se cree que serán persistentes. Dejarlos en la sesión por demasiado tiempo abre de nuevo puertas a ataques. Depende del desarrollador decidir qué datos serán almacenados o se mantendrán en `$_SESSION`.