# UD 2. El lenguaje PHP

## Índice de contenidos

# 1. Elementos del lenguaje PHP

PHP es un lenguaje de guiones (o lenguaje script) diseñado para el desarrollo de páginas web dinámicas utilizando código embebido dentro del lenguaje de marcas, aunque puede ser utilizado para desarrollar otro tipo de aplicaciones. Su sintaxis está basada en la de C/C++, y por lo tanto es muy similar a la de Java.

Hay tres formas de delimitar código PHP:

- **`<?php ... ?>`**. Son los delimitadores recomendados para incluir código PHP dentro de una página.
- **`<? ... ?>`**. El delimitador `<?` se podía utilizar antiguamente como delimitador de bloques PHP, pero desde hace años se desaconseja completamente su uso porque provoca conflictos, entre otros, con la instrucción de procesamiento xml (`<?xml`) de los documentos XML. En su lugar, se debe utilizar el delimitador anterior.

  (*Mediante la directiva short_open_tag (etiqueta de apertura abreviada) se puede permitir el uso del delimitador <? además del delimitador <?php.* )
- **`<?= ... ?>`**. El delimitador `<?=` es una abreviatura de la expresión `<?php echo` que como se verá más adelante permite una lista de expresiones.

  (*Antes de PHP 5.4, publicado en marzo de 2012, para poder usar este delimitador se necesitaba activar la directiva short_open_tag. El problema era que, como se comenta en el apartado anterior, desde hace años se aconseja no activar esa directiva para impedir el uso del delimitador <?. A partir de PHP 5.4 se resolvió este problema desvinculado el delimitador <?= de la directiva short_open_tag. por tanto, desde PHP 5.4, el delimitador <?= se puede utilizar sin limitaciones*.)

## *¡Ojo!*

*Frecuentemente, aunque no es recomendable, se omite el delimitador de cierre cuando el documento termina con código PHP y no HTML.*

## *Para saber más...*

*Una buena práctica como desarrollador es seguir un estilo de código. De esta manera nuestro código estará más ordenado y será más familiar para otros desarrolladores que sigan el mismo estilo.*

*En este módulo seguiremos el estilo de código PSR-2, para lo cual es recomendable que instalemos la extensión **PHP Intelephense** en **Visual Studio Code**, que nos permite formatear tanto el código PHP como el HTML, incluso al mismo nivel ambos cuando coexisten.*

## 1.1. Comentarios

A medida que vayas escribiendo código en PHP, puede ser útil que introduzcas en el mismo algunos comentarios que ayuden a revisarlo cuando lo necesites (a no ser que sigas las recomendaciones de nombres descriptivos de Clean Code).

En una página web los comentarios al HTML van entre los delimitadores de la forma `<!-- Aquí puedes escribir tu comentario -->`. En cambio, dentro del código PHP, hay tres formas de poner comentarios:

- **Comentarios de una línea utilizando `//`**. Son comentarios al estilo del lenguaje C. Cuando una línea comienza por los símbolos `//`, toda ella se considera que contiene un comentario, hasta la siguiente línea.
- **Comentarios de una línea utilizando `#`**. Son similares a los anteriores, pero utilizando la sintaxis de los scripts de Linux.
- **Comentarios de varias líneas**. También iguales a los del lenguaje C. Cuando en una línea aparezcan los caracteres `/*`, se considera que ahí comienza un comentario. El comentario puede extenderse varias líneas, y acabará cuando escribas la combinación de caracteres opuesta: `*/`.

```
// Esto es un comentario

# Esto es otro comentario

/* Esto es un
   comentario de
   varias líneas */
```

Recuerda que cuando pongas comentarios al código PHP, éstos no figurarán en ningún caso en la página web que se envía al cliente (justo al contrario de lo que sucede con los comentarios HTML).

## 1.2. Variables y tipos de datos en PHP

Como en todos los lenguajes de programación, en PHP puedes crear variables para almacenar valores. En el caso de PHP, las variables deben comenzar por el signo `$`, seguido de una letra o el carácter `_`. El resto del nombre de la variable puede ser cualquier número de letras, números y caracteres de subrayado.

Al contrario que en muchos otros lenguajes, en PHP no es necesario declarar una variable ni especificarle un tipo (entero, cadena...) concreto. ***PHP es un lenguaje débilmente tipado, al contrato que Java que es fuertemente tipado***. Para empezar a usar una variable, simplemente asígnale un valor utilizando el operador `=`.

```
$mi_variable = 7;
```

Dependiendo del valor que se le asigne, a la variable se le aplica un tipo de datos, que puede cambiar si cambia su contenido. Esto es, el tipo de la variable se decide en función del contexto en que se emplee.

```
// Al asignarle el valor 7, la variable es de tipo “entero”
$mi_variable = 7;

// Si le cambiamos el contenido
$mi_variable = "siete";

// La variable puede cambiar de tipo
// En este caso pasa a ser de tipo “cadena”
```

Los tipos de datos más simples en PHP son:

- **boolean (lógico)**. Sus posibles valores son true y false. Además, cualquier número entero se considera como true, salvo el 0 que es false.
- **integer (entero)**. Cualquier número sin decimales. Se pueden representar en formato decimal, octal (comenzando por un 0), o hexadecimal (comenzando por 0x).
- **float (real)**. Cualquier número con decimales. Se pueden representar también en notación científica.
- **string (cadena)**. Conjuntos de caracteres delimitados por comillas simples o dobles.
- **NULL (nulo)**. Es un tipo de datos especial, que se usa para indicar que la variable no tiene valor.

Por ejemplo:

```
$mi_booleano = false;
$mi_entero = 25;
$mi_entero = 0x2A; // 42 en Decimal
$mi_real = 7.3e-1;
$mi_cadena = "texto";
$mi_variable = NULL;
```

Si realizas una operación con variables de distintos tipos, ambas se convierten primero a un tipo común. Por ejemplo, si sumas un entero con un real, el entero se convierte a real antes de realizar la suma:

```
$mi_entero = 3;
$mi_real = 2.3;
$resultado = $mi_entero + $mi_real;

// La variable $resultado es de tipo real
```

Estas conversiones de tipo, que en el ejemplo anterior se lleva a cabo de forma automática, también se pueden realizar de forma forzada:

```
$mi_entero = 3;
$mi_real = 2.3;
$resultado = $mi_entero + (int) $mi_real;
```

```
// La variable $mi_real se convierte a entero (valor 2) antes de
sumarse.
// La variable $resultado es de tipo entero (valor 5)
```

*Para saber más ...*

*En la documentación de PHP se especifican las conversiones de tipo posibles y los resultados obtenidos con cada una.*

## 1.3 Expresiones y operadores

Como en muchos otros lenguajes, en PHP se utilizan las expresiones para realizar acciones dentro de un programa. Todas las expresiones deben contener al menos un operando y un operador. Por ejemplo:

```
$mi_variable = 7;
$a = $b + $c;
$valor++;
$x += 5;
```

Los operadores que puedes usar en PHP son similares a los de muchos otros lenguajes como Java.

### 1.3.1. Operaciones aritméticas

- Suma: `+`
- Resta: `-`
- Multiplicación: `*`
- División: `/`
- Módulo: `%`
- Potencia: `**` (PHP 5.6 en adelante).
- Incremento y decremento: `++` y `--`. Si se utilizan junto a una expresión de asignación, modifican el operando antes o después de la asignación en función de su posición (antes o después) con respecto al operando.

```
$a = 5;
$b = ++$a;
// Antes se le suma uno a $a (pasa a tener valor 6)
// y después se asigna a $b (que también acaba con valor 6).

$a = 5;
$b = $a++;
// Antes se asigna a $b el valor de $a (5).
// y después se le suma uno a $a (pasa a tener valor 6)
```

### 1.3.2. Asignaciones

Además del operador =, existen operadores con los que realizar operaciones y asignaciones en un único paso (`+=`, `-=`...) añadiendo el símbolo de la operación antes del igual.

### 1.3.3. Comparaciones

Además de los que nos podemos encontrar en otros lenguajes (`>`, `>=`...), en PHP tenemos dos operadores para comprobar igualdad (`==`, `===`) y tres para comprobar diferencia (`<>`, `!=` y `!==`).

- Los operadores `<>` y `!=` son equivalentes. Comparan los valores de los operandos.
- El operador `===` devuelve verdadero (true) sólo si los operandos son del mismo tipo y además tienen el mismo valor. El operador `!==` devuelve verdadero (true) si los valores de los operandos son distintos o bien si éstos no son del mismo tipo. Por ejemplo:

```
$x = 0;
// La expresión $x == false es cierta (devuelve true).
// Sin embargo, $x === false no es cierta (devuelve false) pues $x es
de tipo entero, no booleano.
```

### 1.3.4. Combinación de condiciones

Tratan a los operandos como variables booleanas (true o false):

- **Y lógico** (operadores `and` o `&&`)
- **O lógico** (operadores `or` o `||`)
- **No lógico** (operador `!`)
- **O lógico exclusivo** (operador `xor`).

### 1.3.5. Realizar operaciones binarias

- AND booleano (`&`)
- OR booleano (`|`)
- XOR booleano (`^`)
- NOT booleano (`~`)
- Desplazamiento a izquierda (`<<`)
- Desplazamiento a derecha (`>>`)

## *Para saber más ...*

*En la documentación de PHP se explican en detalle todos los operadores disponibles en el lenguaje y la forma en que se utilizan.*

## 1.4. Ámbito de utilización de las variables

En PHP puedes utilizar variables en cualquier lugar de un programa. Si esa variable aún no existe, la primera vez que se utiliza se reserva espacio para ella. En ese momento, dependiendo del lugar del código en que aparezca, se decide desde qué partes del programa se podrá utilizar esa variable. A esto se le llama ***visibilidad de la variable***.

Si la variable aparece por primera vez dentro de una función, se dice que esa ***variable es local*** a la función. Si aparece una asignación fuera de la función, se le considerará una variable distinta. Por ejemplo:

```
$a = 1;
function prueba()
{
    // Dentro de la función no se tiene acceso a la variable $a
anterior
    $b = $a;
    // Por tanto, la variable $a usada en la asignación anterior es
una variable nueva
    // que no tiene valor asignado (su valor es null)
}
```

Si en la función anterior quisieras utilizar la variable $a externa, podrías hacerlo utilizando la palabra ***global***. De esta forma le dices a PHP que no cree una nueva variable local, sino que utilice la ya existente.

```
$a = 1;
function prueba()
{
    global $a;
    $b = $a;
    // En este caso se le asigna a $b el valor 1
}
```

Las variables locales a una función desparecen cuando acaba la función y su valor se pierde. Si quisieras ***mantener el valor de una variable local entre distintas llamadas a la función***, deberás declarar la variable como estática utilizando la palabra ***static***.

```
function contador()
{
    static $a=0;
    $a++;
     // Cada vez que se ejecuta la función, se incrementa el valor de
$a
}
```

Las variables estáticas deben inicializarse en la misma sentencia en que se declaran como estáticas. De esta forma, se inicializan sólo la primera vez que se llama a la función.

*Para saber más ...*

*Puedes consultar más ejemplos sobre los ámbitos de las variables en la documentación de PHP.*

## 1.5. Bloques de código

Los programas escritos en PHP, además de encontrarse estructurados normalmente en varias páginas (ya veremos más adelante cómo se pueden comunicar datos de unas páginas a otras), suelen incluir en una misma página varios bloques de código.

Cada bloque de código debe ir entre delimitadores, y en caso de que genere alguna salida, ésta se introduce en el código HTML en el mismo punto en el que figuran las instrucciones en PHP.

Por ejemplo, en las siguientes líneas tenemos dos bloques de código en PHP:

```
<body>
    <?php $a=1; ?>
    <p>Página de prueba</p>
    <?php $b=$a; ?>
```

Aunque no se utilice el valor de las variables, en el segundo bloque de código la variable $a mantiene el valor 1 que se le ha asignado anteriormente.

## 1.6. Funciones de impresión

Existen varias formas incluir contenido en la página web a partir del resultado de la ejecución de código PHP. La forma más sencilla es usando `echo`, que no devuelve nada `(void)`, y genera como salida el texto de los parámetros que recibe. Ejemplo:

```
void echo (string $arg1, ...);
```

Otra posibilidad es `print`, que funciona de forma similar. La diferencia más importante entre `print` y `echo`, es que `print` sólo puede recibir un parámetro y devuelve siempre 1.

```
int print (string $arg);
```

Tanto `print` como `echo` no son realmente funciones, por lo que no es obligatorio que pongas paréntesis cuando las utilices.

`printf()` es otra opción para generar una salida desde PHP. Puede recibir varios parámetros, el primero de los cuales es siempre una cadena de texto que indica el formato que se ha de aplicar. Esa cadena debe contener un especificador de conversión por cada uno de los demás parámetros que se le pasen a la función, y en el mismo orden en que figuran en la lista de parámetros. Por ejemplo:

```
$ciclo="DAW";
$modulo="DWES";
print "<p>";
printf("%s es un módulo de %d curso de %s", $modulo, 2, $ciclo);
print "</p>";
```

Cada especificador de conversión va precedido del carácter % y se compone de las siguientes partes:

- **signo (opcional)**. Indica si se pone signo a los número negativos (por defecto) o también a los positivos (se indica con un signo +).
- **relleno (opcional)**. Indica que carácter se usará para ajustar el tamaño de una cadena. Las opciones son el carácter 0 o el carácter espacio (por defecto se usa el espacio).
- **alineación (opcional)**. Indica que tipo de alineación se usará para generar la salida: justificación derecha (por defecto) o izquierda (se indica con el carácter -).
- **ancho (opcional)**. Indica el mínimo número de caracteres de salida para un parámetro dado.
- **precisión (opcional)**. Indica el número de dígitos decimales que se mostrarán para un número real. Se escribe como un dígito precedido por un punto.
- **tipo (obligatorio)**. Indica cómo se debe tratar el valor del parámetro correspondiente. En la siguiente tabla puedes ver una lista con todos los especificadores de tipo. Especificadores de tipo para las funciones printf y sprintf.

| Especificador | Significado |
|---|---|
| b | el argumento es tratado como un entero y presentado como un número binario. |
| c | el argumento es tratado como un entero, y presentado como el carácter con dicho valor ASCII. |
| d | el argumento es tratado como un entero y presentado como un número decimal. |
| u | el argumento es tratado como un entero y presentado como un número decimal sin signo. |
| o | el argumento es tratado como un entero y presentado como un número octal. |
| x | el argumento es tratado como un entero y presentado como un número hexadecimal (con minúsculas). |
| X | el argumento es tratado como un entero y presentado como un número hexadecimal (con mayúsculas). |
| ff | el argumento es tratado como un doble y presentado como un número de coma flotante (teniendo en cuenta la localidad). |
| F | el argumento es tratado como un doble y presentado como un número de coma flotante (sin tener en cuenta la localidad). |
| e | el argumento es presentado en notación científica, utilizando la e minúscula (por ejemplo, 1.2e+3). |
| E | el argumento es presentado en notación científica, utilizando la e mayúscula (por ejemplo, 1.2E+3). |
| gg | se usa la forma más corta entre %f y %e. |
| G | se usa la forma más corta entre %f y %E. |
| s | el argumento es tratado como una cadena y es presentado como tal. |
| % | se muestra el carácter %. No necesita argumento.. |

Por ejemplo, al ejecutar la línea siguiente, en la salida el número π se obtiene con signo y sólo con dos decimales:

```
printf("El número π vale %+.2f", 3.1416); // +3.14
```

Existe una función similar a **`printf()`** pero en vez de generar una salida con la cadena obtenida, permite guardarla en una variable de tipo string: **`sprintf()`**.

```
$txt_pi = sprintf("El número π vale %+.2f", 3.1416);
```

## 1.7. Cadenas de texto

En PHP las cadenas de texto pueden usar tanto comillas simples como comillas dobles. Sin embargo hay una diferencia importante entre usar unas u otras: **cuando se pone una variable dentro de unas comillas dobles, se procesa y se sustituye por su valor**.

Así, los siguientes ejemplos serían equivalentes:

```
$modulo="DWES";
echo "<p>Módulo:";
echo $modulo;
echo "</p>";

$modulo="DWES";
echo "<p>Módulo: $modulo</p>";
```

La variable `$modulo` se reconoce dentro de las comillas dobles, y se sustituye por el valor "DWES" antes de generar la salida. Si esto mismo lo hubieras hecho utilizando comillas simples, no se realizaría sustitución alguna.

**Para que PHP distinga correctamente el texto que forma la cadena del nombre de la variable, a veces es necesario rodearla entre llaves { }**.

```
echo "<p>Módulo: {$modulo}</p>";
```

El siguiente código muestra una situación en que es estrictamente necesario para poder "escapar" las comillas dobles:

```
for($i=0; $i<=count($arrElementos); $i++) {
    echo "Este es el elemento '{$arrElementos[$i]["nombre"]}'";
}
```

### 1.7.1. Secuencias de escape

Cuando se usan comillas simples, sólo se realizan dos sustituciones dentro de la cadena:

- Cuando se encuentra la secuencia de caracteres `\'`, se muestra en la salida una comilla simple.
- Cuando se encuentra la secuencia `\\`, se muestra en la salida una barra invertida.

Estas secuencias se conocen como **secuencias de escape**.

En las cadenas que usan comillas dobles, además de la secuencia `\\`, se pueden usar algunas más, pero no la secuencia `\'`.

En esta tabla puedes ver las secuencias de escape que se pueden utilizar, y cuál es su resultado:

| Secuencia | Resultado |
|---|---|
| `\\` | se muestra una barra invertida. |
| `\'` | se muestra una comilla simple. |
| `\"` | se muestra una comilla doble. |
| \n | se muestra un avance de línea (LF o 0x0A (10) en ASCII). |
| \r | se muestra un retorno de carro (CR o 0x0D (13) en ASCII). |
| \t | se muestra un tabulador horizontal (HT o 0x09 (9) en ASCII). |
| \v | se muestra un tabulador vertical (VT o 0x0B (11) en ASCII). |
| \f | se muestra un avance de página (FF o 0x0C (12) en ASCII). |
| `\$` | se muestra un signo del dólar. |

### 1.7.2. Operadores

En PHP tienes dos operadores exclusivos para trabajar con cadenas de texto:

- El operador de concatenación punto (`.`) devuelve unidas la cadena de su izquierda y la de su derecha.
- El operador de asignación y concatenación (`.=`) concatena al argumento del lado izquierdo la cadena del lado derecho.

```
$a = "Módulo ";
$b = $a . "DWES"; // ahora $b contiene "Módulo DWES"
$a .= "DWES"; // ahora $a también contiene "Módulo DWES"

$nombre = "Laura";
echo "El usuario " . $nombre . " ha accedido al sistema";
// Resultado: El usuario Laura ha accedido al sistema.
```

## *Para saber más ...*

*En PHP tienes otra alternativa para crear cadenas: la sintaxis **heredoc**.*

*Consiste en poner el operador <<< seguido de un identificador de tu elección, y a continuación y empezando en la línea siguiente la cadena de texto, sin utilizar comillas. La cadena finaliza cuando escribes ese mismo identificador en una nueva línea. Esta línea de cierre no debe llevar más caracteres, ni siquiera espacios o sangría, salvo quizás un punto y coma después del identificador.*

```
$a = <<<MICADENA
Desarrollo de Aplicaciones Web<br />
Desarrollo Web en Entorno Servidor
MICADENA;

echo $a;
```

## 1.8. Funciones relacionadas con los tipos de datos

### 1.8.1. Comprobación y conversión

En PHP existen funciones específicas para comprobar y establecer el tipo de datos de una variable, **`gettype()`** obtiene el tipo de la variable que se le pasa como parámetro y devuelve una cadena de texto, que puede ser `array`, `boolean`, `double`, `integer`, `object`, `string`, `NULL`, `resource` o `unknowntype`.

También podemos comprobar si la variable es de un tipo concreto utilizando una de las siguientes funciones: `is_array()`, `is_bool()`, `is_float()`, `is_integer()`, `is_null()`, `is_numeric()`, `is_object()`, `is_resource()`, `is_scalar()` e `is_string()`. Devuelven *true* si la variable es del tipo indicado.

Análogamente, para establecer el tipo de una variable utilizamos la función **`settype()`** pasándole como parámetros la variable a convertir, y una de las siguientes cadenas: `boolean`, `integer`, `float`, `string`, `array`, `object` o `null`. La función **`settype()`** devuelve *true* si la conversión se realizó correctamente, o *false* en caso contrario.

```
$a = $b = "3.1416"; // asignamos a las dos variables la misma cadena de texto
settype($b, "float"); // y cambiamos $b a tipo float
echo "\$a vale $a y es de tipo ".gettype($a);
echo "<br />";
echo "\$b vale $b y es de tipo ".gettype($b);
```

El resultado del código anterior es:

```
$a vale 3.1416 y es de tipo string
$b vale 3.1416 y es de tipo double
```

### 1.8.2. Definición de variables

Si lo único que te interesa es saber si una variable está definida y no es null, puedes usar la función **`isset()`**. La función `unset` destruye la variable o variables que se le pasa como parámetro.

```
$a = "3.1416";
if (isset($a)) { // si la variable $a está definida...
    unset($a); //ahora ya no está definida
}
```

Si quieres comprobar si no está definida o, si lo está pero está vacía (vale "", 0, null, false, array sin elementos...) puedes usar la función `empty()`.

Existe también en PHP una función llamada **`define()`** con la que puedes definir **constantes**, esto es, identificadores a los que se les asigna un valor que no cambia durante la ejecución del programa.

```
bool define (string $identificador, mixed $valor);
```

Los identificadores de constantes **no van precedidos por el signo $** y **suelen escribirse en mayúsculas.**

```
define ("PI", 3.1416);
echo "El valor de PI es " . PI;
```

Sólo se permiten los siguientes tipos de valores para las constantes: array, integer, float, string, boolean y null.

### 1.8.3. Fecha y hora

En PHP no existe un tipo de datos específico para trabajar con fechas y horas. La información de fecha y hora se almacena internamente como un número entero (Tiempo Unix). Sin embargo, dentro de las funciones de PHP tienes a tu disposición unas cuantas para trabajar con ese tipo de datos.

Una de las más útiles es quizás la función **`date()`**, que te permite obtener una cadena de texto a partir de una fecha y hora, con el formato que tú elijas. La función recibe dos parámetros, la descripción del formato y el número entero que identifica la fecha, y devuelve una cadena de texto formateada.

```
string date (string $formato [, int $fechahora]);
```

El formato lo debes componer utilizando como base una serie de caracteres de los que figuran en la siguiente tabla.

*Función date: caracteres de formato para fechas y horas*

| Carácter | Resultado |
|---|---|
| d | día del mes con dos dígitos. |
| j | día del mes con uno o dos dígitos ( sin ceros iniciales ). |
| z | día del año, comenzando por el cero ( 0 = 1 de enero ). |
| N | día de la semana ( 1 = lunes, ..., 7 = domingo ). |
| w | día de la semana ( 0 = domingo, ..., 6 = sábado ). |
| l | texto del día de la semana, en inglés ( Monday, ..., Sunday ). |
| D | texto del día de la semana, solo tres letras, en inglés ( Mon, ..., Sun ). |
| W | número de la semana del año. |
| m | número del mes con dos dígitos. |
| n | número del mes con uno o dos dígitos ( sin ceros iniciales ). |
| t | número de días que tiene el mes. |
| F | texto del día del mes, en inglés ( January, ..., December ). |
| M | texto del día del mes, solo tres letras, en inglés ( Jan, ..., Dec ). |
| Y | número del año. |

| Carácter | Resultado |
|---|---|
| y | dos últimos dígitos del número del año. |
| L | 1 si el año es bisiesto, 0 si no lo es. |
| h | formato de 12 horas, siempre con dos dígitos. |
| H | formato de 24 horas, siempre con dos dígitos. |
| g | formato de 12 horas, con uno o dos dígitos ( sin ceros iniciales ). |
| G | formato de 24 horas, con uno o dos dígitos ( sin ceros iniciales ). |
| i | minutos, siempre con dos dígitos. |
| s | segundos, siempre con dos dígitos. |
| u | microsegundos. |
| a | am o pm, en minúsculas. |
| A | AM o PM, en mayúsculas. |
| r | fecha entera con formato RFC 2822. |

Además, el segundo parámetro es opcional. Si no se indica, se utilizará la hora actual para crear la cadena de texto.

Por ejemplo, si usamos...

```
echo date('l jS \of F Y h:i:s A');
```

...tendremos algo como...

```
Thursday 5th of October 2022 10:47:56 AM
```

La vista en español la tendremos si lanzamos:

```
setlocale(LC_ALL,"es_ES");
echo strftime("%A %d de %B del %Y");
```

## *¡Ojo!*

*Los formatos para strftime() son distintos a los de date(). Puedes consultarlos aqui.*

Para que el sistema pueda darte información sobre tu fecha y hora debe conocer la zona horaria. Esta se establece de forma global en el archivo php.ini (parámetro `date.timezone`), pero también puedes usar la función **`date_default_timezone_set()`** para establecer la zona horaria de tu script. Ejemplo:

```
date_default_timezone_set('Europe/Madrid');
```

Si utilizas alguna función de fecha y hora sin haber establecido tu zona horaria, lo más probable es que recibas un error o mensaje de advertencia de PHP indicándolo.

### *Para saber más ...*

*En la documentación de PHP puedes consultar las distintas zonas horarias que se pueden indicar.*

*Otras funciones como `getdate()` devuelven un array con información sobre la fecha y hora actual.*

### *Para saber más ...*

*En la documentación de PHP puedes consultar todas las funciones para gestionar fechas y horas.*

### *Actividad 2.1*

*Haz una página web que muestre la fecha y hora actual con un formato similar al siguiente: "25-12-2022 17:13:23".*

## 1.9. Variables especiales de PHP

En la unidad anterior ya aprendiste qué eran y cómo se utilizaban las variables globales. PHP incluye unas cuantas variables internas predefinidas que pueden usarse desde cualquier ámbito, por lo que reciben el nombre de **variables superglobales**. Ni siquiera es necesario que uses global para acceder a ellas.

Cada una de estas variables es un array que contiene un conjunto de valores (en esta unidad veremos más adelante cómo se pueden utilizar los arrays). Las variables superglobales disponibles en PHP son las siguientes:

- $_SERVER. Contiene información sobre el entorno del servidor web y de ejecución. Entre la información que nos ofrece esta variable, tenemos:

| Valor | Contenido |
|---|---|
| $_SERVER['PHP_SELF'] | script que se está ejecutando actualmente. |
| $_SERVER['SERVER_ADDR'] | dirección IP del servidor web. |
| $_SERVER['SERVER_NAME'] | nombre del servidor web. |
| $_SERVER['DOCUMENT_ROOT'] | directorio raíz bajo el que se ejecuta el script actual. |
| $_SERVER['REMOTE_ADDR'] | dirección IP desde la que el usuario está viendo la página. |
| $_SERVER['REQUEST_METHOD'] | método utilizado para acceder a la página ('GET', 'HEAD', 'POST' o 'PUT') |

## *Para saber más ...*

*En la documentación de PHP puedes consultar toda la información que ofrece `$_SERVER`:*

- `$_GET`, `$_POST` y `$_COOKIE` contienen las variables que se han pasado al script actual utilizando respectivamente los métodos GET (parámetros en la URL), HTTP POST y además las cookies HTTP.
- `$_REQUEST` junta en uno solo el contenido de los tres arrays anteriores, `$_GET`, `$_POST` y `$_COOKIE`.
- `$_ENV` contiene las variables que se puedan haber pasado a PHP desde el entorno en que se ejecuta.
- `$_FILES` contiene los ficheros que se puedan haber subido al servidor utilizando el método POST.
- `$_SESSION` contiene las variables de sesión disponibles para el guion actual.

En posteriores unidades iremos trabajando con estas variables, pero conviene tener a mano la información sobre estas variables superglobales disponible en el manual de PHP.

## *Actividad 2.2*

*Crea un script en el que imprima los campos de la variable `$_SERVER` mencionada anteriormente. Accede desde tu equipo al servidor de un compañero para comprobar cómo cambia la información.*

# 2. Estructuras de control

En PHP, los scripts se construyen en base a sentencias. Utilizando llaves, puedes agrupar las sentencias en conjuntos, que se comportan como si fueran una única sentencia.

Para definir el flujo de un script en PHP, al igual que en la mayoría de lenguajes de programación, hay sentencias para dos tipos de estructuras de control:

- **Sentencias condicionales**, que permiten definir las condiciones bajo las que debe ejecutarse una sentencia o un bloque de sentencias
- **Sentencias de bucle**, con las que puedes definir si una sentencia o conjunto de sentencias se repite o no, y bajo qué condiciones.

*Para saber más ...*

*Puedes usar también (**aunque no es recomendable**) la sentencia **goto**, que te permite saltar directamente a otro punto del script que indiques mediante una etiqueta.*

```
$a = 1;
goto salto;
$a++;  //esta sentencia no se ejecuta
salto:
echo $a;  // el valor obtenido es 1
```

## 2.1. Condicionales

### 2.1.1. if / elseif / else

La sentencia `if` permite definir una expresión para ejecutar o no la sentencia o conjunto de sentencias siguiente. Si la expresión se evalúa a *true* (verdadero), la sentencia se ejecuta. Si se evalúa a *false* (falso), no se ejecutará.

Cuando el resultado de la expresión sea *false*, puedes utilizar `else` para indicar una sentencia o grupo de sentencias a ejecutar en ese caso. Otra alternativa a `else` es utilizar `elseif` y escribir una nueva expresión que comenzará un nuevo condicional.

```
if ($a < $b) {
    echo "a es menor que b";
} elseif ($a > $b) {
    echo "a es mayor que b";
} else {
    echo "a es igual a b";
}
```

Cuando, como sucede en el ejemplo, la sentencia `if`, `elseif` o `else` actúe sobre una única sentencia, no será necesario usar llaves:

```
if ($a < $b)
    echo "a es menor que b";
elseif ($a > $b)
    echo "a es mayor que b";
else
    echo "a es igual a b";
```

¡Ojo!

La abreviación **`elseif`** no debe confundirse con la clausula **ELSIF** de PL/SQL

### 2.1.2. switch

La sentencia **`switch`** es similar a enlazar varias sentencias if comparando una misma variable con diferentes valores. Cada valor va en una sentencia case. Cuando se encuentra una coincidencia, comienzan a ejecutarse las sentencias siguientes hasta que acaba el bloque switch, o hasta que se encuentra una sentencia break. Si no existe coincidencia con el valor de ningún case, se ejecutan las sentencias del bloque default, en caso de que exista.

```
switch ($a) {
    case 0:
        echo "a vale 0";
        break;
    case 1:
        echo "a vale 1";
        break;
    default:
        echo "a no vale 0 ni 1";
}
```

*Actividad 2.3*

*Crea un script en el que tengamos almacenado el año en una variable y nos imprima por pantalla si el año es bisiesto o no.*

*Un año es bisiesto si es divisible por 4 y, si es divisible por 100, debe serlo también por 400.*

## 2.2. Bucles

### 2.2.1. Bucle while

Usando **`while`** puedes definir un bucle que se ejecuta mientras se cumpla una expresión. La expresión se evalúa antes de comenzar cada ejecución del bucle.

```
$a = 1;
while ($a < 8) {
   $a += 3;
}
echo $a; // el valor obtenido es 10
```

### 2.2.2. Bucle do / while

Es un bucle similar al anterior, pero la expresión se evalúa al final, con lo cual se asegura que la sentencia o conjunto de sentencias del bucle se ejecutan al menos una vez.

```
$a = 5;
do {
    $a -= 3;
} while ($a > 10);
echo $a; // el bucle se ejecuta una sola vez, el valor obtenido es 2
```

### 2.2.3. Bucle for

Son los bucles más complejos de PHP. Al igual que los del lenguaje Java, se componen de tres expresiones:

```
for (expr1; expr2; expr3) {
    # sentencia o conjunto de sentencias;
}
```

- La primera expresión, *expr1*, se ejecuta solo una vez al comienzo del bucle.
- La segunda expresión, *expr2*, se evalúa para saber si se debe ejecutar o no la sentencia o conjunto de sentencias. Si el resultado el false, el bucle termina.
- Si el resultado es true, se ejecutan las sentencias y al finalizar se ejecuta la tercera expresión, *expr3*, y se vuelve a evaluar *expr2* para decidir si se vuelve a ejecutar o no el bucle.

Un ejemplo es:

```
for ($a = 5; $a<10; $a+=3) {
    echo $a; // Se muestran los valores 5 y 8
    echo "<br />";
}
```

Puedes anidar cualquiera de los bucles anteriores en varios niveles. También puedes usar, aunque no se recomienda, las sentencias **`break`**, para salir del bucle, y **`continue`**, para omitir la ejecución de las sentencias restantes y volver a la comprobación de la expresión respectivamente.

## *Actividad 2.4*

*Genera una pirámide rellena de asteriscos mediante el uso de bucles.*

## *Actividad 2.5*

*Escribe un código en el que, teniendo en variables el número de filas y columnas de una tabla, se genere mediante bucles dicha tabla.*

# 3. Funciones

Cuando quieres repetir la ejecución de un bloque de código, puedes utilizar un bucle. Las funciones tienen una utilidad similar: nos permiten asociar una etiqueta (el nombre de la función) con un bloque de código a ejecutar.

Al usar funciones estamos ayudando a estructurar mejor el código, ya que el código dentro de una función está aislado del exterior, puede ser reutilizado y, por tanto, evita repeticiones innecesarias.

## 3.1. Ejecución y creación de funciones

Ya sabes que para hacer una llamada a una función, basta con poner su nombre y unos paréntesis.

```
phpinfo();
```

Para crear tus propias funciones, deberás usar la palabra **`function`**.

```
function da_la_bienvenida() {
   echo "Bienvenido a mi web";
}

da_la_bienvenida();
```

En PHP por regla general no es necesario que definas una función antes de utilizarla ...

```
da_la_bienvenida();

function da_la_bienvenida() {
   echo "Bienvenido a mi web";
}
```

...excepto cuando está condicionalmente definida como se muestra en el siguiente ejemplo:

```
$primeraVisita = true;
da_la_bienvenida(); // Da error, aún no está definida la función
if ($primeraVisita) {
    function da_la_bienvenida() {
        echo "Bienvenido a mi web";
    }
}
da_la_bienvenida(); // Aquí ya no da error
```

Cuando una función está definida de una forma condicional sus definiciones deben ser procesadas antes de ser llamadas. Por tanto, la definición de la función debe estar antes de cualquier llamada.

## 3.2. Argumentos

A veces necesitamos pasar a la función valores. Podemos hacerlo con variables globales, pero no es una buena práctica. Siempre es mejor utilizar argumentos o parámetros al hacer la llamada.

Los argumentos se indican en la definición de la función como una lista de variables separada por comas:

```
function precio_con_iva($precio) {
    echo $precio * 1.21;
}

function precio_con_descuento($precio, $descuento) {
    echo $precio * (100 - $descuento) / 100;
}
```

Las funciones además pueden devolver un resultado usando la sentencia `return`:

```
function precio_con_iva($precio) {
    return $precio * 1.21;
}

$precio = 10;

$precio_iva = precio_con_iva($precio);

echo "El precio con IVA es ".$precio_iva
```

Si una función no tiene sentencia `return` devuelve null.

Normalmente **es preferible que una función devuelva un resultado frente a que lo imprima**, ya que de esa manera es el programador que utiliza la función el que decide qué hacer con el resultado.

### *¡Ojo!*

*Cuando en una función se alcanza la sentencia `return`, la función termina. Cualquier sentencia de la función que se escriba después no se ejecuta.*

```
function calculaCuadrado($numero) {
    return $numero * $numero;
    echo "hola"; // Esta línea no se ejecuta
}
```

Al definir la función, puedes indicar valores por defecto para los argumentos, de forma que cuando hagas una llamada a la función puedes no indicar el valor de un argumento; en este caso se toma el valor por defecto indicado.

```
function precio_con_descuento($precio, $descuento = 10)
{
    return $precio * (100 - $descuento) / 100;
}
$precio = 10;

$precio_rebajado = precio_con_descuento($precio); // Rebaja del 10%
echo "El precio con descuento estándar es ".$precio_rebajado;

$precio_rebajado = precio_con_descuento($precio, 25); // Rebaja del
25%
echo "El precio con 25% de descuento es ".$precio_rebajado;
```

Puede haber valores por defecto definidos para varios argumentos, pero en la lista de argumentos de la función todos ellos deben estar a la derecha de cualquier otro argumento sin valor por defecto.

En los ejemplos anteriores los argumentos se pasaban por valor. Esto es, cualquier cambio que se haga dentro de la función a los valores de los argumento no se reflejará fuera de la función. Si quieres que esto ocurra debes definir el parámetro para que su **valor se pase por referencia**, añadiendo el símbolo **&** antes de su nombre.

```
function precio_con_iva(&$precio, $iva=0.21) {
    $precio *= (1 + $iva);
}
$precio = 10;
precio_con_iva($precio);
echo "El precio con IVA es ".$precio
```

A partir de PHP 7 **se puede indicar el tipo de dato de cada argumento**, indicando el tipo antes del nombre del argumento. También **se puede indicar el tipo de dato de retorno**, indicando el tipo precedido de **:** entre el paréntesis que cierra los argumentos y la llave que abre la función:

```
function precio_con_descuento(float $precio, int $descuento) : float
{
    return $precio * (100 - $descuento) / 100;
}
```

Esta mejora hace más fácil detectar posibles errores:

```
function precio_con_descuento($precio, $descuento)
{
    return $precio * (100 - $descuento) / 100;
}

function precio_con_descuento_tipado(float $precio, int $descuento) :
float
{
    return $precio * (100 - $descuento) / 100;
}
```

```
echo precio_con_descuento(5.99, "baratillo");
// Asume que el descuento es 0 (imprime 5.99) y da warning

echo precio_con_descuento_tipado(5.99, "baratillo");
// Da error y detiene el script
```

*Actividad 2.6*

*Crea una función llamada `es_familia_numerosa()` que reciba como parámetro el número de hermanos de una familia en forma numérica y **devuelva** true si se trata de una familia numerosa (3 o más hermanos) o false en caso contrario.*

*Crea una versión tipada y otra sin tipar, y comprueba qué ocurre si un programador despistado introduce "tres" como número de hermanos.*

## 3.3. Inclusión de ficheros externos

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

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

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

  ```
  <?php
  //definiciones.php
  $modulo = 'DWES';
  $ciclo = 'DAW';
  ```

  ```
  <?php
  // script.php
  echo "Módulo $modulo del ciclo $ciclo<br />"; //Solo muestra "Modulo
  del ciclo"
  include 'definiciones.php';
  echo " Módulo $modulo del ciclo $ciclo<br />"; // muestra "Modulo
  DWES del ciclo DAW"
  ```

  Cuando se comienza a evaluar el contenido del fichero externo, se abandona de forma automática el modo PHP y su contenido se trata en principio como etiquetas HTML. Por este motivo, es necesario delimitar el código PHP que contenga nuestro archivo externo utilizando dentro del mismo los delimitadores .

- **include_once**: Si por equivocación incluyes más de una vez un mismo fichero, lo normal es que obtengas algún tipo de error (por ejemplo, al repetir una definición de una función). La función `include_once` funciona exactamente igual que include, pero solo incluye aquellos ficheros que aún no se hayan incluido.
- **require**: Si el fichero que queremos incluir no se encuentra, `include` da un aviso y continua la ejecución del script. La diferencia más importante al usar `require` es que en ese caso, cuando no se puede incluir el fichero, da error y se detiene la ejecución del script..
- **require_once**. Es la combinación de las dos anteriores. Asegura la inclusión del fichero indicado solo una vez, y genera un error si no se puede llevar a cabo.

Muchos programadores utilizan la doble extensión `.inc.php` para aquellos ficheros en lenguaje PHP cuyo destino es ser incluidos dentro de otros, y nunca han de ejecutarse por sí mismos.

### *Actividad 2.7*

*Crea tres archivos php: header.inc.php, footer.inc.php e inicio.php. El contenido debe ser el siguiente:*

- ***header.inc.php**: HTML5 del principio de la página (etiqueta de apertura html, sección head y etiqueta de apertura body).*
- ***footer.inc.php**: HTML del final de la página (sección footer con mensaje de copyright y cierre de etiqueta body y html). En el mensaje de copyright de la página debe incluirse el año actual (generado con PHP).*
- ***inicio.php**: Utiliza los otros dos ficheros para crear un documento web completa en la que se muestre un mensaje de bienvenida con la etiqueta h1.*

## 3.4. PECL (PHP Extension Community Library)

Como programador puedes aprovecharte de la gran cantidad de funciones disponibles en PHP. De éstas, muchas están incluidas en el núcleo de PHP y se pueden usar directamente. Otras muchas se encuentran disponibles en forma de extensiones, y se pueden incorporar al lenguaje cuando se necesitan.

Con la distribución de PHP se incluyen varias extensiones. Para poder usar las funciones de una extensión, tienes que asegurarte de activarla mediante el uso de una directiva `extension=ARCHIVO_EXTENSION.so`

Dependiendo de la distribución de GNU/Linux que se use, se puede añadir la directiva al fichero `php.ini` directamente o mediante la creación de un archivo con extensión `.ini` en la carpeta `conf.d/` que contenga la directiva.

Muchas otras extensiones no se incluyen con PHP y, para poder usarse, la extensión se debe descargar, compilar, instalar y cargar.

Para obtener extensiones para el lenguaje PHP puedes utilizar PECL. PECL es un repositorio de extensiones para PHP. Estas extensiones se pueden obtener de la página web de PECL, pero la forma más cómoda es usando el comando `pecl`:

```
pecl install <extension>
```

Esto descargará el código fuente de `<extension>`, lo compilará, e instalará el archivo `<extension.so>` en su directorio de extensiones. En ese momento se podrá añadir la directiva `extension=<extension.so>` a la configuración de PHP.

## *¡Atención!*

*Al construir un módulo PHP, es importante contar con las herramientas requeridas para compilar (autoconf, automake, libtool, etc.).*

## *Actividad 2.8*

*Instala la librería mcrypt en tu instalación de PHP siguiendo los siguientes pasos:*

```
sudo apt-get -y install gcc make autoconf libc-dev pkg-config
sudo apt-get -y install libmcrypt-dev
sudo apt-get -y install php-pear php7.2-dev
sudo pecl install mcrypt-1.0.1
```

*Durante el proceso nos aparecerá la dirección donde se genera el módulo:*

```
Installing '/usr/lib/php/XXX/mcrypt.so'
```

*Fíjate en el valor de **XXX** y, una vez instalada, ejecuta las siguientes líneas (sustituyendo **XXX** por lo que corresponda):*

```
sudo bash -c "echo extension=/usr/lib/php/XXX/mcrypt.so >
/etc/php/7.2/cli/conf.d/mcrypt.ini"
sudo bash -c "echo extension=/usr/lib/php/XXX/mcrypt.so >
/etc/php/7.2/apache2/conf.d/mcrypt.ini"
```

*Esto es equivalente a añadir la línea a los archivos php.ini. Reinicia apache y comprueba con phpinfo que la se ha cargado correctamente. Puedes hacerlo también con el siguiente comando:*

```
php-config
```

# 4. Tipos de datos compuestos

Un tipo de datos compuesto es aquel que te permite almacenar más de un valor. En PHP puedes utilizar dos tipos de datos compuestos: el **array** y el **objeto**. Los objetos los veremos más adelante; vamos a empezar con los arrays.

Un array es un tipo de datos que nos permite almacenar varios valores. Cada miembro del array se almacena en una posición a la que se hace referencia utilizando un valor clave. **Las claves pueden ser numéricas o asociativas**.

```
// array numérico
$modulos1 = array
(
      0 => "Programación",
      1 => "Bases de datos",
       ...,
      9 => "Desarrollo web en entorno servidor"
);


// array asociativo
$modulos2 = array(
 "PR" => "Programación",
 "BD" => "Bases de datos",
 ...,
 "DWES" => "Desarrollo web en entorno servidor"
);
```

En PHP existe la función **`print_r()`**, que nos muestra todo el contenido del array que le pasamos. Es muy útil para tareas de depuración.

### *Actividad 2.9*

*Utiliza `print_r` con `$_SERVER` para ver toda la información disponible. Comprueba cómo se muestra la información si encierras la salida de `print_r` entre las etiquetas `<pre>` y `</pre>`*

Para hacer referencia a los elementos almacenados en un array, tienes que utilizar el valor clave entre corchetes:

```
echo $modulos1[9];
echo $modulos2["DWES"];
```

Los arrays anteriores son **vectores**, esto es, arrays unidimensionales. En PHP puedes crear también arrays de varias dimensiones almacenando otro array en cada uno de los elementos de un array.

```
// array bidimensional
$ciclos = array
(
      "DAW" => array
             (
                   "PR" => "Programación",
                   "BD" => "Bases de datos", ...,
                   "DWES" => "Desarrollo web en entorno servidor"
             ),
      "DAM" => array
             (
                   "PR" => "Programación",
                   "BD" => "Bases de datos", ...,
                   "PMDM" => "Programación multimedia y de dispositivos
móviles"
             )
);
```

Para hacer referencia a los elementos almacenados en un array multidimensional, debes indicar las claves para cada una de las dimensiones:

```
echo $ciclos["DAW"]["DWES"];
```

Para iniciar un array indicando sus elementos puedes utilizar la función **`array()`** o la notación de corchetes:

```
// array numérico
$lista = array(1,2,3,4)
$lista2 = [1,2,3,4];
```

En PHP no es necesario que indiques el tamaño del array antes de crearlo. Ni siquiera es necesario indicar que una variable concreta es de tipo array. Simplemente puedes comenzar a asignarle valores:

```
// array numérico
$modulos1[0] = "Programación";
$modulos1[1] = "Bases de datos";
...
$modulos1[9] = "Desarrollo web en entorno servidor";

// array asociativo
$modulos2["PR"] = "Programación";
$modulos2["BD"] = "Bases de datos";
...
$modulos2["DWES"] = "Desarrollo web en entorno servidor";
```

Ni siquiera es necesario que especifiques el valor de la clave. Si la omites, el array se irá llenando a partir de la última clave numérica existente, o de la posición 0 si no existe ninguna:

```
$modulos1[] = "Programación";
$modulos1[] = "Bases de datos";
...
$modulos1[] = "Desarrollo web en entorno servidor";
```

## 4.1. Recorrer arrays

Las cadenas de texto o strings se pueden tratar como arrays en los que se almacena una letra en cada posición, siendo 0 el índice correspondiente a la primera letra, 1 el de la segunda, etc.

```
// cadena de texto
$modulo = "Desarrollo web en entorno servidor";
// $modulo[3] == "a";
```

Para recorrer los elementos de un array, en PHP puedes usar un bucle específico: **`foreach`**. Utiliza una variable temporal para asignarle en cada iteración el valor de cada uno de los elementos del array. Puedes usarlo de dos formas. Recorriendo sólo los valores:

```
$modulos = array("PR" => "Programación", "BD" => "Bases de datos", ...,
"DWES" => "Desarrollo web en entorno servidor");

foreach ($modulos as $modulo) {
    echo "Módulo: ".$modulo. "<br />"
}
```

O recorriendo los valores y sus claves de forma simultánea:

```
$modulos = array("PR" => "Programación", "BD" => "Bases de datos", ...,
"DWES" => "Desarrollo web en entorno servidor");

foreach ($modulos as $codigo => $modulo) {
    echo "El código del módulo ".$modulo." es ".$codigo."<br />"
}
```

### *Actividad 2.10*

*Haz una página PHP que utilice foreach para mostrar todos los valores del array $_SERVER en una tabla con dos columnas. La primera columna debe contener la clave, y la segunda su valor.*

### *Para saber más ...*

*En PHP también hay otra forma de recorrer los valores de un array: **usando el puntero interno del array**. Utilizando funciones específicas, podemos avanzar, retroceder o inicializar el puntero, así como recuperar los valores del elemento (o de la pareja clave / elemento) al que apunta el puntero en cada momento. Algunas de estas funciones son:*

| ***Función*** | ***Resultado*** |
|---|---|
| *reset* | *Sitúa el puntero interno al comienzo del array.* |
| *next* | *Avanza el puntero interno una posición.* |
| *prev* | *Mueve el puntero interno una posición hacia atrás.* |

| ***Función*** | ***Resultado*** |
|---|---|
| *end* | *Sitúa el puntero interno al final del array.* |
| *current* | *Devuelve el elemento de la posición actual.* |
| *key* | *Devuelve la clave de la posición actual.* |
| *each* | *Devuelve un array con la clave y el elemento de la posición actual. Además, avanza el puntero interno una posición.* |

*Ejemplo:*

```
$transporte = array('pie', 'bici', 'coche', 'moto');
$modo = current($transporte); // $modo = 'pie';
$modo = next($transporte);    // $modo = 'bici';
$modo = next($transporte);    // $modo = 'coche';
$modo = prev($transporte);    // $modo = 'bici';
$modo = end($transporte);     // $modo = 'moto';
```

## 4.2. Funciones relacionadas con los tipos de datos compuestos

### `array()`

Además de asignando valores directamente, la función `array` permite crear un array con una sola línea de código, tal y como vimos anteriormente. Esta función recibe un conjunto de parámetros, y crea un array a partir de los valores que se le pasan. Si en los parámetros no se indica el valor de la clave, crea un array numérico (con base 0). Si no se le pasa ningún parámetro, crea un array vacío.

```
$a = array();  // array vacío
$modulos = array("Programación", "Bases de datos", ..., "Desarrollo web en
entorno servidor");   // array numérico
```

### `unset()`

Una vez definido un array puedes añadir nuevos elementos (no definiendo el índice, o utilizando un índice nuevo) y modificar los ya existentes (utilizando el índice del elemento a modificar). También se pueden eliminar elementos de un array utilizando la función `unset`.

```
unset ($modulos [0]);
// El primer elemento pasa a ser $modulos [1] == "Bases de datos";
```

*¡Ojo!*

*En el caso de los arrays numéricos, eliminar un elemento significa que las claves del mismo ya no serán consecutivas.*

### array_values()

La función `array_values` recibe un array como parámetro, y devuelve un nuevo array con los mismos elementos y con índices numéricos consecutivos empezando en 0.

```
$nuevo_array = array_values($viejo_array);
```

### is_array()

Para comprobar si una variable es de tipo array, utiliza la función `is_array`.

```
$variable = array(1,2,3,4,5);
if (is_array($variable)) {
    echo "La variable es un array";
}
```

### count()

Para obtener el número de elementos que contiene un array, tienes la función `count`.

```
$array = array(1,2,3,4,5);
echo "El array contiene " . count($array) . " elementos";
```

### in_array()

Si quieres buscar un elemento concreto dentro de un array, puedes utilizar la función `in_array`. Recibe como parámetros el valor del elemento a buscar y la variable de tipo array en la que buscar, y devuelve true si encontró el valor o false en caso contrario.

```
$modulos = array("Programación", "Bases de datos", "Desarrollo web en
entorno servidor");
$modulo = "Bases de datos";
if (in_array($modulo, $modulos)) {
    echo "Existe el módulo de nombre ".$modulo;
}
```

### array_search()

Otra posibilidad es la función `array_search`, que recibe los mismos parámetros pero devuelve la clave correspondiente al elemento, o false si no lo encuentra.

```
$modulos = array("Programación", "Bases de datos", "Desarrollo web en
entorno servidor");
$modulo = "Bases de datos";
$posicion = array_search($modulo, $modulos)
if ($posicion !== false) {
    echo "El módulo ".$modulo . " esta en la posición " . $posicion;
}
```

### array_key_exists()

Y si lo que quieres buscar es una clave en un array, tienes la función `array_key_exists`, que devuelve true o false.

```
$modulos = array("PR" => "Programación",
```

```
                "BD" => "Bases de datos",
                "DWES" => "Desarrollo web en entorno servidor");
$clave = "DWES";
if (array_key_exists($clave, $modulos)) {
    echo "Existe la clave ".$clave;
}
```

## *Para saber más ...*

*Puedes consultar la lista completa de funciones relacionadas con arrays en el manual online de PHP.*

## *Actividad 2.11*

*Crea un array numérico y prueba a borrar el tercer elemento. Comprueba que los índices numéricos ya no están consecutivos. Utiliza `array_values` para crear una copia del array y comprueba que sus índices son consecutivos.*

## *Actividad 2.12*

*Usando un array asociativo, crea un diccionario inglés - español en el que la clave sea la palabra en inglés y el valor la palabra en español. Rellénalo con unas cuantas palabras y, usando funciones para arrays:*

- *Muestra el número de palabras que contiene el diccionario.*
- *Comprueba si el diccionario contiene la palabra inglesa cuya traducción es "pelota".*
- *Comprueba si la palabra "car" viene en el diccionario.*
- *Comprueba si el diccionario contiene la traducción de la palabra "fresa" y, si la tiene, muestrala.*

# 5. Envío de datos a la aplicación web

En ocasiones es necesario que el visitante nos indique qué producto quiere comprar, cuál es su nombre, su edad... en resumen, que nos envíe información. Para ello las opciones más comunes son hacerlo a través de la propia **URL** y a través de **formularios**.

## 5.1. Envío de datos a través de la URL

Normalmente el navegador hace peticiones tipo GET para acceder a la página web. Aunque este tipo de peticiones está diseñado para recibir un recurso, es posible enviar a su vez información incluyéndola en la URL de la petición.

Para ello al final de la URL se añade una interrogación (`?`) seguida del nombre de la variable, un signo igual (`=`) y su valor:

```
http://periodico.com/noticias.php?categoria=sucesos
```

Si se necesitan varias variables, estas se separan con un ampersand (`&`):

```
http://periodico.com/noticias.php?categoria=sucesos&pagina=3
```

Este método no es apropiado para enviar grandes cantidades de información (está limitado a unos 2000 caracteres) pero permite compartir la URL y que otro visitante haga exactamente la misma petición. Este método se utiliza principalmente cuando la información que se necesita es para hacer consultas (cuando no se van a hacer cambios) y es con el que construimos la red de enlaces de nuestra aplicación web.

### *Para saber más...*

*El SEO es muy importante en internet, y una de las prácticas más comunes es el uso de **URLs amigables**. Son URLs intuitivas en las que se procura que haya palabras clave. Para ello se configura el servidor de forma que si la URL recibida encaja con un patrón, la transforme en la auténtica de forma transparente al usuario.*

*Por ejemplo, podemos fijar que el patrón `/categoria/([A-Za-z]+)/pagina/([0-9]+)` se corresponda con la url `/noticias.php?categoria=$1&pagina=$2`, de forma que si un visitante utiliza la URL `http://periodico.com/categoria/deportes/pagina/5` internamente se usaría la URL `http://periodico.com/noticias.php?categoria=deportes&pagina=5`.*

Para acceder a estos valores desde PHP utilizamos principalmente la variable `$_GET`, la cual es un array asociativo cuyas etiquetas son los nombres de las variables de la URL. Por

ejemplo, en el caso de la URL `http://periodico.com/noticias.php?categoria=sucesos&pagina=3`, podremos acceder a los datos de la siguiente manera:

```
echo $_GET['categoria']; // Imprime "sucesos"
$pagina = $_GET['pagina']; // Almacena "3" en la variable $pagina
```

Es una buena práctica comprobar si las variables están definidas, ya que el visitante puede no haberlas puesto en la URL:

```
if (isset($_GET['categoria'])) {
    echo $_GET['categoria'];
}

if (isset($_GET['pagina'])) {
    $pagina = $_GET['pagina'];
} else {
    $pagina = 1; // valor por defecto
}
```

A partir de PHP 7 está disponible el ***operador de fusión de null*** `??` (null coalescing operator), el cual asigna un valor en caso de que la variable no esté definida o sea nula:

```
// Si $_GET['pagina'] no está definido asigna 1 a la variable.
$pagina = $_GET['pagina'] ?? 1;
```

Como se mencionó en el punto sobre las variables especiales, es posible usar `$_REQUEST` en lugar de `$_GET`.

### *Para saber más …*

*En las nuevas características de PHP 7 aparece el operador de fusión de null.*

### *Actividad 2.13*

*Crea un script llamado `area_triangulo.php` que reciba por GET los parámetros base y altura y muestre un mensaje indicando el valor de cada uno y el área del correspondiente triángulo.*

*Si alguno de los parámetros no se ha recibido, se utilizará un 5 en su lugar.*

## 5.2. Envío de datos a través de formularios

La otra forma natural para hacer llegar a la aplicación web los datos del usuario desde un navegador es utilizar formularios HTML.

Los formularios HTML van encerrados siempre entre las etiquetas `<form> </form>`. Dentro de un formulario se incluyen los elementos sobre los que puede actuar el usuario, principalmente usando las etiquetas `<input>`, `<select>`, `<textarea>` y `<button>`.

El atributo `action` del elemento `<form>` indica la URL a la que se le enviarán los datos del formulario al pulsar en botón de envío. En nuestro caso se tratará de un script PHP.

Por su parte, el atributo `method` especifica el método usado para enviar la información. Este atributo puede tener dos valores:

- **GET**: con este método los datos del formulario se agregan a la URL objetivo automáticamente, como se ha visto en el punto anterior. La información que se envía desde un formulario con el método `GET` es visible para todo el mundo (todos los nombres de variables y sus valores se muestran en la URL) y esto no es elegante de cara al usuario. No es una opción muy usada.
- **POST**: con este método los datos se envían utilizando una petición HTTP o HTTPS tipo POST. Es el método más utilizado en formularios, sobretodo cuando se desea hacer cambios (añadir un comentario, enviar un mensaje, modificar un pedido...). Es la forma más elegante de enviar datos, pues estos **se incluyen en el cuerpo de la petición**, de forma que no se le muestran al usuario variables ni valores. Como contrapartida, el usuario no puede compartir la URL y que otro visitante vea la misma página, ya que esta no contiene la información. Ejemplo de petición POST:

```
POST /contacto.php HTTPS/1.1
Host: mipagina.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 49
Accept-Language: es-es

(encriptado)email=maria97@gmail.com&nombre=Maria&mensaje=Hol
a
```

## 5.2.1 Procesamiento de la información devuelta por un formulario web

Los datos enviados con un formulario se recogen en la página destino utilizando `$_GET` o `$_POST`, según corresponda, o `$_REQUEST` en ambos casos, usando como clave del array el valor `name` del campo correspondiente.

Dado el siguiente formulario:

```
<form name="input" action="procesa.php" method="post">
    Nombre: <input type="text" name="nombre" /><br />
    Email: <input type="text" name="correo" /><br />
    <input type="submit" value="Enviar" name="enviar"/>
</form>
```

Los datos se recogerían en el script `procesa.php`, en las variables `$_POST['nombre']` y `$_POST['correo']`:

```
// fragmento de procesa.php
echo "Nombre: " . $_POST['nombre'] . "<br />";
echo "Correo: " . $_POST['correo'] . "<br />";
```

Se supone que `procesa.php` es un script al que se accede desde el formulario, pero puede haber visitantes que intenten acceder directamente, de modo que no podemos dar por sentado que se haya rellenado el formulario.

Una forma muy común de comprobar si el formulario ha sido enviado es ponerle un nombre al botón de enviar y comprobar si el botón de enviar está definido en `$_POST` (el botón se comporta como cualquier otro campo del formulario):

```
// fragmento de procesa.php
if (isset($_POST['enviar'])) {
    echo "Nombre: " . $_POST['nombre'] . "<br />";
    echo "Correo: " . $_POST['correo'] . "<br />";
}
```

Además, si la página tiene varios formularios, esta técnica nos permite saber cual de ellos se ha usado.

Si en un formulario web tienes que enviar alguna variable en la que sea posible almacenar más de un valor tendrás que ponerle corchetes [] al "name" para indicar que se trata de un array.

```
<p>Hay que comprar:</p>
<input type="checkbox" name="cosas[]" value="Patatas" />
    Patatas<br />
<input type="checkbox" name="cosas[]" value="Pimientos" />
    Pimientos<br />
```

En el script php, `$_POST['cosas']` será un array que contenga los *value* de los cuadros marcados.

También puede haber casos en los que necesitemos incluir información en el formulario que el usuario no tenga por qué ver, como la fecha y hora o la IP del usuario. Para estos casos se pueden usar campos ocultos de tipo hidden:

```
<input type="hidden" name="timestamp" value="<?php echo date('d/m/Y h:i:s');?>" />
```

Este es un ejemplo de formulario en el que un alumno indica su nombre y los módulos de los que se matricula:

```
<!-- matricula.php -->
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Formulario de matriculación</title>
    </head>
    <body>
        <form action="procesar_matricula.php" method="post">
            <label for="nombre">Nombre:</label><br />
```

```
            <input type="text" name="nombre"><br />
            <p>Módulos que cursa:</p>
                        <label><input  type="checkbox"  name="modulos[]"  value="DWES"
/>DWES</label><br />
                        <label><input  type="checkbox"  name="modulos[]"  value="DWEC"
/>DWEC</label><br />
                         <label><input  type="checkbox"  name="modulos[]"  value="DIW"
/>DIW</label><br />
                         <label><input  type="checkbox"  name="modulos[]"  value="DAW"
/>DAW</label><br />
                         <label><input  type="checkbox"  name="modulos[]"  value="EIE"
/>EIE</label><br />
            <br />
            <input type="submit" value="Enviar" name="enviar"/>
                <input type="hidden" name="timestamp" value="<?php echo date('d/m/Y
h:i:s');?>" />
        </form>
    </body>
</html>
```

Y el correspondiente script en el que se procesa la información recibida:

```
<!-- procesar_matricula.php -->
<!DOCTYPE html>
<html>
   <head>
      <meta charset="UTF-8">
      <title>Formulario de matriculación</title>
   </head>
   <body>
<?php if (isset($_POST['enviar'])) { ?>
      <p>Nombre del alumno: <?php echo $_POST['nombre']; ?></p>
      <p>Módulos:</p>
      <ul>
      <?php
      foreach ($_POST['modulos'] as $modulo) {
         echo "<li>".$modulo."</li>";
      }
      ?>
      </ul>
      <p>Fecha: <?php echo $_POST['timestamp']; ?></p>
<?php } ?>
   </body>
</html>
```

## *Actividad 2.14*

*Crea un script PHP con un formulario POST que permita introducir dos números y un select con las operaciones de sumar o restar. Incluye también como dato oculto la IP del visitante.*

*Crea otro script PHP en el que se procese el formulario y se muestre el valor de la operación. Debes comprobar que el botón haya sido pulsado.*

### 5.2.2. Validación de los datos introducidos

Si queremos que una aplicación de cualquier tipo sea robusta, una de las tareas que debemos realizar es comprobar cada vez que un usuario introduce datos que estos son válidos. Por ejemplo que no deje en blanco un campo obligatorio, o no introduzca un número negativo como edad.

En nuestro caso, en el código de procesamiento del formulario tendríamos que comprobar si está todo correcto y, en caso contrario, notificar al usuario de su error.

Esto puede afectar terriblemente a la experiencia del usuario, ya que dependiendo del navegador probablemente tenga que rellenar el formulario de nuevo para corregir el error.

Por tanto, es muy recomendable hacer una **validación del lado del cliente**, es decir, hacer una validación **antes de enviar los datos**, de forma que **el propio navegador realice las comprobaciones**. De esta manera, si hay errores se le indican al usuario antes para que los corrija y el servidor recibe menos peticiones erróneas. Esto se hace principalmente con código JavaScript, aunque a partir de HTML 5 hay etiquetas especiales (como `<input type="email" />`) que el navegador valida automáticamente.

*¡Ojo!*

*No se puede confiar en la navegación del lado cliente, ya que el visitante puede desactivar o manipular el código JavaScript y su navegador puede no soportar la validación HTML.* ***La validación de lado cliente es recomendable, pero la de lado servidor es obligatoria****.*

#### Procesado en una sola página

Del lado servidor, una opción para mejorar la experiencia de usuario es **procesar el formulario en la misma página**, es decir, usar la misma página que muestra el formulario como destino de los datos:

- Si tras comprobar los datos, estos son correctos, se muestra un mensaje de confirmación o se reenvía a otra página.
- Si tras comprobar los datos, estos son incorrectos, se vuelve a mostrar el formulario relleno con los datos recibidos y se indican cuáles son incorrectos y por qué.

Para hacerlo de este modo, hay que comprobar si la página recibe datos (hay que mostrarlos y no generar el formulario), o si no recibe datos (hay que mostrar el formulario). En el siguiente código de ejemplo se muestra cómo hacerlo:

```
<!-- matricula.php -->
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Formulario de matriculación</title>
    </head>
    <body>
<?php if (isset($_POST['enviar'])) { ?>
```

```
        <p>Nombre del alumno: <?php echo $_POST['nombre']; ?></p>
        <p>Módulos:</p>
        <ul>
        <?php
        foreach ($_POST['modulos'] as $modulo) {
            echo "<li>".$modulo."</li>";
        }
        ?>
        </ul>
        <p>Fecha: <?php echo $_POST['timestamp']; ?></p>
<?php } else { ?>
        <form action="matricula.php" method="post">
            <label for="nombre">Nombre:</label><br />
            <input type="text" name="nombre"><br />
            <p>Módulos que cursa:</p>
            <label><input type="checkbox" name="modulos[]" value="DWES"
/>DWES</label><br />
            <label><input type="checkbox" name="modulos[]" value="DWEC"
/>DWEC</label><br />
            <label><input type="checkbox" name="modulos[]" value="DIW"
/>DIW</label><br />
            <label><input type="checkbox" name="modulos[]" value="DAW"
/>DAW</label><br />
            <label><input type="checkbox" name="modulos[]" value="EIE"
/>EIE</label><br />
            <br />
            <input type="submit" value="Enviar" name="enviar"/>
            <input type="hidden" name="timestamp" value="<?php echo date('d/m/Y
h:i:s');?>" />
        </form>
<?php } ?>
    </body>
</html>
```

Fíjate en la forma de englobar el formulario dentro de una sentencia else para que sólo se genere si no se reciben datos en la página.

Además, para enviar los datos a la misma página que contiene el formulario podemos usar `$_SERVER['PHP_SELF']` para obtener su nombre; esto hace que no se produzca un error aunque la página se cambie de nombre:

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

### Comprobación de los datos

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

Para ello podemos sustituir la condición que comprueba si el botón ha sido pulsado por la siguiente:

```
<?php if (!empty($_POST['modulos']) && !empty($_POST['nombre']))
{ ?>
```

```
      // Aquí se incluye el código a ejecutar cuando los datos son
correctos
<?php } else { ?>
    // Aquí generamos el formulario, indicando los datos incorrectos
    //   y rellenando los valores correctamente introducidos
<?php } ?>
```

Esta condición comprueba que *módulos* y *nombre* no estén vacíos, es decir, que estén definidos (lo que indica que el formulario ha sido enviado) y su valor no sea null, un array vacío o una cadena vacía.

### Rellenado del formulario

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

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

Para las casillas utilizamos el atributo checked:

```
<label><input type="checkbox" name="modulos[]" value="DWES"
<?php
if (isset($_POST['modulos']) && in_array("DWES",$_POST['modulos'])) {
    echo ' checked="checked"';     //  Observa  el  intercambio  de
comillas
}
?>
/>DWES</label><br />
```

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

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

```
<label for="nombre">Nombre:</label><br />
<input type="text" name="nombre" value="
<?php
if (isset ($_POST['nombre']))
      echo $_POST['nombre'];?>
" />
<?php
if (isset($_POST['enviar']) && empty($_POST['nombre'])) {
    echo " <span class='error'>Debe introducir un nombre</span>";
}
?>
<br />
```

Revisa el documento con el ejemplo completo, fijándote en las partes que hemos comentado anteriormente.

```
<!-- matricula.php -->
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Formulario de matriculación</title>
    </head>
    <body>
<?php if (isset($_POST['enviar'])) { ?>
        <p>Nombre del alumno: <?php echo $_POST['nombre']; ?></p>
        <p>Módulos:</p>
        <ul>
        <?php
        foreach ($_POST['modulos'] as $modulo) {
            echo "<li>".$modulo."</li>";
        }
        ?>
        </ul>
        <p>Fecha: <?php echo $_POST['timestamp']; ?></p>
<?php } else { ?>
        <form action="matricula.php" method="post">
            <label for="nombre">Nombre:</label><br />
            <input type="text" name="nombre"><br />
            <p>Módulos que cursa:</p>
             <label><input type="checkbox" name="modulos[]" value="DWES"
/>DWES</label><br />
             <label><input type="checkbox" name="modulos[]" value="DWEC"
/>DWEC</label><br />
             <label><input type="checkbox" name="modulos[]" value="DIW"
/>DIW</label><br />
             <label><input type="checkbox" name="modulos[]" value="DAW"
/>DAW</label><br />
             <label><input type="checkbox" name="modulos[]" value="EIE"
/>EIE</label><br />
            <br />
            <input type="submit" value="Enviar" name="enviar"/>
            <input type="hidden" name="timestamp" value="<?php echo
date('d/m/Y h:i:s');?>" />
        </form>
<?php } ?>
    </body>
</html>
```

## *Actividad 2.15*

*Adapta el formulario anterior para que se procese en la misma página y además compruebe que ninguno de los dos números se ha quedado en blanco. Si no se ha hecho correctamente se debe mostrar un error y los campos que estuvieran rellenos deben mantenerse.*