INDICE

[**Algunas funciones de PHP** 3](#_Toc7026126)

[**Funciones básicas** 3](#_Toc7026127)

[**Funciones de vectores** 3](#_Toc7026128)

[**Funciones de cadenas** 5](#_Toc7026129)

[**Funciones de variables** 6](#_Toc7026130)

[HTML 5 7](#_Toc7026131)

[2. Esquema mínimo de un documento HTML 8](#_Toc7026132)

[3. El texto básico de un documento HTML 9](#_Toc7026133)

[3.1. La etiqueta básica 10](#_Toc7026134)

[3.2. Etiquetas de formato de caracteres 12](#_Toc7026135)

[3.3. Otras etiquetas de caracteres 13](#_Toc7026136)

[**Escribiendo hojas de estilo** 14](#_Toc7026137)

[Elementos modificables con CSS 14](#_Toc7026138)

[**Estilo del texto** 14](#_Toc7026139)

[**Bordes de un bloque** 18](#_Toc7026140)

[**Márgenes de los bloques** 20](#_Toc7026141)

[**Dimensiones de los bloques** 21](#_Toc7026142)

[COMANDOS BÁSICOS DE POSTGRESQL 22](#_Toc7026143)

[Función PostgreSQL STRING\_AGG 25](#_Toc7026144)

[PostgreSQL ISNULL 25](#_Toc7026145)

[Estructuras de control en PHP 26](#_Toc7026146)

[**1. if** 26](#_Toc7026147)

[**2. else** 26](#_Toc7026148)

[**3. elseif/else if** 26](#_Toc7026149)

[**4. while** 26](#_Toc7026150)

[**5. do-while** 27](#_Toc7026151)

[**6. for** 27](#_Toc7026152)

[**7. foreach** 29](#_Toc7026153)

[**8. break** 30](#_Toc7026154)

[**9. continue** 31](#_Toc7026155)

[**10. switch** 32](#_Toc7026156)

[**11. declare** 33](#_Toc7026157)

[**12. return** 35](#_Toc7026158)

[**13. include/include\_once** 35](#_Toc7026159)

[**14. require/require\_once** 37](#_Toc7026160)

[**15. goto** 37](#_Toc7026161)

[**16. Sintaxis alternativas** 38](#_Toc7026162)

[WAMP 39](#_Toc7026163)

[Bibliografía 40](#_Toc7026164)

**Algunas funciones de PHP**

En este capítulo no van a incluirse todas y cada una de las funciones que soportan PHP y sus módulos. Se trata de crear una guía general de las funciones más comunes del lenguaje.

**Funciones básicas**

echo y print

A lo largo de todo el texto hemos utilizado las funciones *echo* y *print*, sin preocuparnos de explicar su significado y sintaxis. Dichas funciones, como ya habrá supuesto el lector, imprimen en la salida estándar el argumento que se les pasa. Hay que hacer notar que su sintaxis es muy flexible, siendo válidas las cuatro siguientes llamadas:

$valor = 52;

echo "El valor es $valor";

echo ("El valor es $valor");

print "El valor es $valor";

print ("El valor es $valor");

**Funciones de vectores**

Ya hemos visto con anterioridad el manejo de vectores, así como su inicialización. En este apartado vamos a estudiar una serie de funciones que van a simplificar el manejo de vectores. Además, prestaremos especial atención a una función de inicialización que aparece en esta sección.

array()

Esta función devuelve un vector que forma en base a los parámetros que se le pasan. A los parámetros se les puede dar un índice asociativo por medio del símbolo **=>**. Vamos a ver un ejemplo donde creamos un vector de dos dimensiones:

$varios = array(

"colores" => array("a"=>"amarillo, "b"=>"rojo"),

"sentidos" => array("tacto", "gusto", "olfato", "oido", "vista")

);

El resto de funciones

|  |  |  |
| --- | --- | --- |
| **nombre** | **parámetros** | **descripción** |
| **array\_count\_values()** | vector A | Devuelve un vector usando los valores del vector de entrada como índices asociativos y su frecuencia como valor |
| **array\_flip()** | vector A | Devuelve el vector inverso a A |
| **array\_keys()** | vector A | Devuelve un vector con todos los valores de clave de A |
| **array\_merge()** | vectores A; B, ... | Devuelve el vector concatenación de los vectores A, B, ... |
| **array\_pop()** | vector V | Devuelve y extrae el valor del último elemento de V |
| **array\_push** | vector V; gen A, B, ... | Añade al final de V los elementos A, B, ... devolviendo el número de elementos añadidos |
| **array\_reverse()** | vector V | Devuelve un vector con los elementos de V en orden inverso |
| **array\_shift** | vector V | Devuelve y extrae el valor del primer elemento de V |
| **array\_slice()** | vector V; ent A, B | Extrae de un vector los elementos que van desde la posición A hasta A+B, devolviendo un nuevo vector con dichos elementos |
| **array\_splice()** | vector V; ent A, B; vector W | Elimina los elementos de V desde A hasta A+B e inserta en su lugar los elementos del vector W |
| **array\_unshift()** | vector V; gen A, B, ... | Inserta los elementos A, B, ... al principio de V y devuelve la cantidad de elementos insertados |
| **array\_values()** | vector V | Devuelve un vector con todos los valores de V |
| **array\_walk()** | vector V; cad f; gen A, B, ... | Aplica la función f a todos los elementos del vector V añadiendo los parámetros A, B, ... en cada llamada |
| **arsort()** | vector V | Ordena un vector de forma inversa manteniendo la asociación clave-valor |
| **asort()** | vector V | Ordena un vector manteniendo la asociación clave-valor |
| **count()** | gen V | Devuelve el número de elementos de una variable |
| **current()** | vector V | Devuelve el elemento actual de V |
| **each()** | vector V | Devuelve la próxima pareja clave-valor de V |
| **end()** | vector V | Lleva el puntero interno del vector a su último elemento |
| **in\_array()** | gen A; vector V | Devuelve verdadero si A se encuentra en V; Falso en otro caso |
| **krsort()** | vector V | Ordena V de forma inversa en base a la clave |
| **ksort()** | vector V | Ordena V en base a la clave |
| **list()** | gen A, B... | Asigna variables como si fueran un vector |
| **next()** | vector V | Avanza el puntero interno de un vector a la siguiente posición |
| **pos()** | vector V | Devuelve el elemento señalado por el puntero interno de V |
| **prev()** | vector V | Retrocede el puntero interno de un vector V a la posición anterior |
| **reset()** | vector V | Coloca el puntero interno de V en la primera posición |
| **rsort()** | vector V | Ordena V de forma inversa |
| **shuffle()** | vector V | "Desordena" los elementos de V |
| **sizeof()** | vector V | Devuelve el número de elementos de V |
| **sort** | vector V | Ordena el vector V |
| **uasort()** | vector V | Ordena un vector en base a una función de comparación definida por el usuario y manteniendo la asociación clave-valor |
| **uksort()** | vector V | Ordena un vector en base a una función de comparación definida por el usuario sobre las claves |
| **usort()** | vector V | Ordena un vector en base a una función de comparación definida por el usuario sobre los valores |

**Funciones de cadenas**

PHP ofrece multitud de funciones para el manejo de cadenas. En esta sección mostramos sólo las más utilizadas.

|  |  |  |
| --- | --- | --- |
| **nombre** | **parámetros** | **descripción** |
| **print()** | cad A | Imprime A en la salida estándar |
| **printf()** | cad A | Imprime A en la salida estándar aplicándole un formato (estilo C) |
| **sprint()** | cad F; gen A, B, ... | Devuelve una cadena formateada según F con las variables A, B, ... |
| **strstr() / strchr()** | cad A, B | Busca B dentro de A y devuelve desde su primera aparición hasta el final de A. |
| **strcmp** | cad A, B | Devuelve un valor menor que 0 si A es menor que B; mayor que 1 si B es menor que A; y 0 si son idénticas |
| **stristr()** | cad A, B | Como strstr pero insensible a mayúsculas/minúsculas |
| **strlen()** | cad A | Devuelve la longitud de A |
| **strpos()** | cad A, B | Devuelve la posición de la primera aparición de B en A |
| **strrchr()** | cad A, B | Análoga a strchr() pero empezando por el final de la cadena |
| **strrev()** | cad A | Invierte una cadena |
| **strrpos()** | cad A, B | Análoga a strpos() pero empezando por el final de la cadena |
| **strrchr()** | cad A, B | Análoga a strchr() pero empezando por el final de la cadena |
| **strtok()** | cad A, B | Extrae un token de A en base a B |
| **strtolower() / strtoupper()** | cad A | Pone una cadena en minúsculas / mayúsculas |
| **str\_replace()** | cad A, B, C | Reemplaza por B las apariciones de A en C |
| **substr()** | cad A; ent B, C | Devuelve la subcadena de A que va desde la posición B hasta la B+C. |
| **substr\_replace()** | cad A, B; ent C, D | Sustituye en A la subcadena desde C hasta C+D por la subcadena B |

**Funciones de variables**

|  |  |  |
| --- | --- | --- |
| **nombre** | **parámetros** | **descripción** |
| **doubleval()** | esc A | Devuelve el valor *double* de A |
| **empty()** | gen A | Devuelve falso si la variable tiene valor vacío o cero |
| **gettype()** | gen A | Devuelve el tipo de la variable en forma de cadena: "integer", "double", "string", "array", "object", "unknown type" |
| **intval()** | est A; ent B | Devuelve el valor entero de A usando B como base numérica. Por defecto, B = 10 |
| **is\_array()** | gen A | Devuelve verdadero si A es un vector. |
| **is\_float() / is\_double() / is\_real()** | gen A | Devuelve verdadero si A es un real. |
| **is\_int() / is\_integer() / is\_long()** | gen A | Devuelve verdadero si A es un entero. |
| **is\_string()** | gen A | Devuelve verdadero si A es una cadena. |
| **is\_object()** | gen A | Devuelve verdadero si A es un objeto. |
| **isset()** | gen A | Devuelve verdadero si existe A |
| **settype()** | gen A; cad B | Fuerza la variable A al tipo B |
| **strval()** | gen A | Convierte el valor de A a una cadena |
| **unset()** | gen A | Destruye A |

# HTML 5

* HTML es un lenguaje universal, que funciona en cualquier plataforma (Windows, Macintosh, Unix, OS/2, etc.) y con cualquier navegador o browser (Netscape, Internet Explorer, Mozilla Firefox, etc.). El precio que paga por su universalidad es su poca sofisticación, puesto que no es otra cosa que el viejo formato ASCII[2](http://paginaspersonales.deusto.es/airibar/Ed_digital/HTML/HTML_1.html" \l "n2) (y con 7 bits en lugar de ocho, por lo que ni tan siquiera tiene acentos ni otros caracteres especiales).
* Por tanto, para que un documento HTML sea algo más que simples caracteres básicos, debe contener, además de dicho texto, una serie de instrucciones para el browser que lo va a reproducir: estas instrucciones se denominan etiquetas o tags y se distinguen del texto porque van entre guiones (< >). Estas etiquetas contienen todo el resto de la información de la página web.
* Por tanto, si un documento HTML no es más que texto ASCII (parte sin guiones y parte entre guiones: las etiquetas), cualquier documento web, hasta el más sofisticado, puede escribirse directamente desde un sencillo programa básico de texto, como el Cuaderno de Notas (*Notepad*) de Windows, por ejemplo.
* Sin embargo, escribir un documento complejo de esta manera exige un conocimiento exhaustivo de las numerosísimas etiquetas existentes y sus normas de utilización; para evitar esta dificultad, hay una serie de programas denominados comúnmente editores de HTML (desde el Composer al Dreamweaver, pasando por tantos otros), que permiten al usuario componer una página (es decir, generar etiquetas HTML) desde un interfaz más o menos parecido al de un procesador de textos.
* A la hora de crear una página web, y aunque utilicemos un editor de HTML, es casi imprescindible entender la disposición del etiquetado de un documento y manejar siquiera las etiquetas más elementales. Son varias las razones que pueden aducirse para ello:
  1. Los editores de HTML no son siempre herramientas perfectas, sobre todo cuando ha habido muchas modificaciones durante el proceso de creación. Con bastante frecuencia se hará necesario consultar el código completo, con las etiquetas HTML (lo que se suele denominar **código fuente** o **código madre**) para corregir las disfunciones.
  2. Tal vez la mejor manera de aprender a diseñar páginas web es observar las páginas de los demás, cuyos códigos fuente están siempre disponibles para ser imitados (o incluso copiados, técnicamente, al menos). Pero, para ello, es necesario entenderlos.
  3. Los mismos editores HTML manejan términos propios del lenguaje HTML (*heading*, *cellspacing*, etc.), que es preciso comprender.
  4. Es necesario un buen conocimiento de HTML para insertar otros códigos más potentes, como los de Javascript, por ejemplo.
* El objetivo de este documento es aprender a componer una sencilla página web desde el más básico de los programas de texto disponibles en Windows: el Cuaderno de Notas.[4](http://paginaspersonales.deusto.es/airibar/Ed_digital/HTML/HTML_1.html" \l "n4) De esta manera, seremos capaces de entender y manejar los códigos fuente de las páginas más complejas.

|  |
| --- |
|  |

# 2. Esquema mínimo de un documento HTML

* Un documento HTML comienza siempre con la etiqueta <HTML>[5](http://paginaspersonales.deusto.es/airibar/Ed_digital/HTML/HTML_1.html" \l "n5), que indica que el documento en cuestión está construido con dicho lenguaje.
* La mayoría de las etiquetas son pareadas, es decir, <...> corresponde al principio de la acción y </...> indica el fin de dicha acción.
* Por tanto, una página web estará siempre contenida entre las etiquetas <HTML> y </HTML>.
* Por otra parte, todo documento HTML consta de dos partes: la cabecera (**head**) y el cuerpo del documento (**body**).
  1. La cabecera contiene básicamente información destinada al browser (o navegador), que queda oculta al usuario. Su etiqueta (pareada) es <HEAD>.
  2. El cuerpo es el documento que ve el usuario. Su etiqueta (pareada) es <BODY>.
* Ya estamos en condiciones de componer nuestra primera página web. Basta con escribir las siguientes etiquetas en nuestro editor de texto y salvar el resultado con la extensión **\*.HTML** o **\*.HTM**:

|  |
| --- |
| **EJEMPLO 1** |
| <HTML> |
| <HEAD> |
| </HEAD> |
| <BODY> |
| </BODY> |
| </HTML> |

* He introducido una pequeña tabulación a la hora de escribir las etiquetas de cabecera y cuerpo, simplemente para facilitar su compresión gráfica, puesto que esta cuestión es totalmente indiferente. El resultado no variaría de haber colocado las etiquetas de cualquier otra manera. Por ejemplo:

|  |
| --- |
| **EJEMPLO 1: Variantes de escritura** |
| <HTML> |
| <HEAD> |
| </HEAD> |
| <BODY> |
| </BODY> |
| </HTML> |
|  |
| <HTML><HEAD></HEAD><BODY></BODY></HTML> |
|  |
| <HTML> <HEAD></HEAD> <BODY></BODY> </HTML> |

* Esta primera página web está, lógicamente, vacía, pero es perfectamente operativa. Para modificarla, basta con modificar sus códigos en el editor de texto. No hace falta cerrar para ello el navegador: una vez modificado –y salvado– el fichero, basta con utilizar el comando **Actualizar** (o **Refresh**) para ver los cambios.
* Al visualizar los resultados del [Ejemplo 1](javascript:openPopWin(%22elementos/Ejemplo_01.html%22,%20400,%20200,%20%22scrollbars%22,%20%22cen%22,%20%22cen%22)), en la barra de título de la ventana correspondiente sólo aparece el nombre del navegador (con Netscape) o la trayectoria completa del archivo (con Explorer). Esto sucede porque nuestro documento no posee un título propio. Para crearlo, debemos introducir una nueva etiqueta (pareada) en la cabecera: <TITLE>. Nuestra página web podría quedar, por tanto, así:

|  |
| --- |
| **EJEMPLO 2** |
| <HTML> |
| <HEAD> |
| <TITLE>Ejemplo 2</TITLE> |
| </HEAD> |
| <BODY> |
| </BODY> |
| </HTML> |

* Ya sabemos crear el esqueleto de una página web. La siguiente tarea será aprender a introducir texto (entre los tags <BODY> y </BODY>) y controlar sus etiquetas elementales.

|  |
| --- |
|  |

# 3. El texto básico de un documento HTML

* El texto básico de un documento HTML puede escribirse sin etiquetas. Cada navegador lo visualiza entonces con el tipo y tamaño de caracteres escogidos en su configuración por defecto, e introduce un salto de línea cuando los caracteres alcanzan el borde de la ventana.
* Para dominar las cuestiones de formato (de caracteres, de párrafo, etc.) es necesario utilizar las etiquetas correspondientes.

## 3.1. La etiqueta básica

* La etiqueta básica, que controla el tipo de fuente utilizado, es <FONT>.
* <FONT> es una etiqueta pareada. Por tanto, afecta a los caracteres introducidos entre <FONT> y </FONT>. Pero sin más especificaciones, la etiqueta aún no sirve para nada.
* Se puede especificar **el tipo de letra** (es decir, la fuente de caracteres) añadiéndolo a la etiqueta, de la siguiente manera (en este caso, la fuente escogida es Arial): <FONT FACE="Arial">...</FONT>.
* Se puede especificar **el color de la fuente** con la etiqueta <FONT COLOR="?">...</FONT>. Para especificar los colores, conviene saber al menos lo siguiente:
  1. Cada color posee su correspondiente código.
  2. Hay dos códigos: hexadecimal –lo más habitual– o RGB. En ambos casos se trata de expresar las proporciones de tres colores básicos: rojo, verde y azul.
  3. Los valores hexadecimales de los tres colores se expresan de la siguiente manera:
     1. Rojo: color:#ff0000;
     2. Verde: color:#00ff00;
     3. Azul: color:#0000ff;
  4. Los mismos colores, expresados con los valores RGB (red, green, blue), son los siguientes:
     1. Rojo: color:rgb(255,0,0);
     2. Verde: color:rgb(0,255,0);
     3. Azul: color:rgb(0,0,255);
  5. Todos los colores se expresan mediante combinaciones de estos tres colores básicos. Existen muchos programas que proporcionan los valores de todos los colores que el usuario puede componer. También en internet existen muchas páginas con los [Códigos de colores](http://html-color-codes.info/codigos-de-colores-hexadecimales/).
  6. Los colores más básicos pueden especificarse sin código, simplemente escribiendo la palabra correspondiente (por supuesto en inglés). Por ejemplo: <FONT COLOR="blue">...</FONT>, <FONT COLOR="red">...</FONT>, etc.
* Se puede especificar **el tamaño de la fuente** con la etiqueta <FONT SIZE="?">...</FONT>. El valor del tamaño ("?") deseado puede suministrarse de manera absoluta (un número del 1 al 7) o relativa (+1, +2, -1, -2, etc., teniendo en cuenta que el valor por defecto es 3).[6](http://paginaspersonales.deusto.es/airibar/Ed_digital/HTML/HTML_1.html" \l "n6)
* Todas estas etiquetas se pueden combinar, de modo que con una sola se controla, por ejemplo, el tipo, el tamaño y el color de los caracteres: <FONT FACE="Arial" COLOR="#0000FF" SIZE="5">...</FONT>.

Con esta etiqueta hemos visto ya la sintaxis básica de las etiquetas HTML, que podíamos formalizar así: <ETIQUETA [ATRIBUTO="valor"]>.

Lo anterior quiere decir que una etiqueta HTML (por ejemplo, FONT) puede llevar uno o varios atributos (FACE, COLOR, SIZE), cuyos valores se colocan entrecomillados tras el signo igual (="arial", ="#0000FF", etc.).

Esto será válido para todas las etiquetas HTML que veamos a partir de este momento.

* Volvamos a nuestro documento web y practiquemos con las posibilidades de la etiqueta <FONT>. Por ejemplo:

|  |
| --- |
| **EJEMPLO 3** |
| <HTML> |
| <HEAD> |
| <TITLE>Ejemplo 3</TITLE> |
| </HEAD> |
| <BODY> |
| Esto es texto simple: cada navegador lo visualizará según su configuración por defecto. |
| <FONT SIZE="1">Este texto es tamaño 1.</FONT> |
| <FONT SIZE="2">Este texto es tamaño 2.</FONT> |
| <FONT SIZE="4">Este texto es tamaño 4.</FONT> |
| <FONT SIZE="+1">Este texto es tamaño +1 (que es lo mismo que tamaño 4).</FONT> |
| <FONT FACE="Arial" SIZE="5" COLOR="FFFF00">Este texto posee varias especificaciones de formato.</FONT> |
| </BODY> |
| </HTML> |

* Nuestra página web, tal y como está diseñada en el Ejemplo 3, no posee ninguna especificación de líneas. A pesar de que las etiquetas están ordenadas en varias líneas, el navegador coloca todos los caracteres seguidos (sólo tiene en cuenta un único espacio entre palabras). Para corregir este problema podemos utilizar la etiqueta <BR>, que introduce un salto de línea. Nuestra página quedaría entonces así:

|  |
| --- |
| **EJEMPLO 4** |
| <HTML> |
| <HEAD> |
| <TITLE>Ejemplo 4</TITLE> |
| </HEAD> |
| <BODY> |
| Esto es texto simple: cada navegador lo visualizará según su configuración por defecto. <BR> |
| <FONT SIZE="1">Este texto es tamaño 1.</FONT><BR> |
| <FONT SIZE="2">Este texto es tamaño 2.</FONT><BR> |
| <FONT SIZE="4">Este texto es tamaño 4.</FONT><BR> |
| <FONT SIZE="+1">Este texto es tamaño +1 (que es lo mismo que tamaño 4).</FONT><BR> |
| <FONT FACE="Arial" SIZE="5" COLOR="FFFF00">Este texto posee varias especificaciones de formato.</FONT> |
| </BODY> |
| </HTML> |

Hay otra cuestión importante que debe saberse sobre la etiqueta <FONT>: es la etiqueta más básica, y conviene empezar a aprender por ella, pero está en desuso y desaparecerá en las versiones futuras de HTML. En su lugar, existen otras etiquetas más potentes, que iremos viendo en su momento.

## 3.2. Etiquetas de formato de caracteres

* A partir de este momento, se trata simplemente de añadir más etiquetas a nuestra aún pequeña colección, de modo que vayamos teniendo cada vez más control sobre el formato de los caracteres. La tabla siguiente muestra las más comunes:

|  |  |
| --- | --- |
| Código | Función |
| <B>...</B> | Negrita |
| <I>...</I> | Cursiva |
| <U>...</U> | Subrayado |
| <SUB>...</SUB> | Subíndice |
| <SUP>...</SUP> | Superíndice |

* Hay que realizar algunos comentarios sobre las etiquetas precedentes:
  1. Existe otra etiqueta similar a <B>, menos utilizada porque depende de las opciones de configuración del navegador: <STRONG>...</STRONG>.
  2. Existe otra etiqueta similar a <I>, menos utilizada porque depende de las opciones de configuración del navegador: <EM>...</EM>.
  3. Conviene tener cuidado a la hora de manejar el subrayado, puesto que éste se utiliza convencionalmente para indicar los hiperenlaces, y podría por tanto resultar confuso para el usuario.
* Es normal utilizar varias etiquetas para un mismo elemento de texto. En ese caso, hay que tener cuidado para encajarlas correctamente. Por ejemplo:  
  <FONT FACE="Courier"><B><I>...</I></B></FONT> es correcto, pero  
  <FONT FACE="Courier"><B><I>...</FONT><B></I> es incorrecto.

## 3.3. Otras etiquetas de caracteres

* Hay muchas otras etiquetas que afectan a los caracteres. Vamos a presentar varias más, que deben manejarse con cierta precaución:
  1. La etiqueta <PRE>...</PRE> indica que el texto afectado está preformateado, de modo que el navegador sí tiene en cuenta todos los espacios y saltos de línea definidos en la pantalla.
  2. La etiqueta <BLINK>...</BLINK> indica texto parpadeante. Sólo funciona con Netscape.
  3. La etiqueta <S>...</S> genera texto tachado.
  4. La etiqueta <TT>...</TT> genera texto similar al de una máquina de escribir (fuente courier de paso fijo).
  5. La etiqueta <BIG>...</BIG> aumenta el tamaño de letra (equivale a <FONT SIZE="+1">).
  6. La etiqueta <SMALL>...</SMALL> disminuye el tamaño de letra (equivale a <FONT SIZE="-1">).

## **Escribiendo hojas de estilo**

### Elementos modificables con CSS

O, mejor dicho, atributos de elementos que pueden modificarse con CSS. Son muchos (casi todos) los aspectos que pueden modificarse con CSS. Dado el carácter general de esta guía, aquí veremos solamente los más utilizados.

**Estilo del texto**

Veamos los parámetros disponibles para dar estilo al texto. Como ya sabes, las unidades de medida aplicables a todos ellos son varias, pero para mayor claridad, en todos los ejemplos utilizaremos solamente el píxel: **px**. Sea por ejemplo la siguiente página:

<HTML>

<HEAD><TITLE>Estilos</TITLE>

<LINK REL="stylesheet" TYPE="text/css" HREF="**estilo.css**">

</HEAD>

<BODY>

Texto fuera de párrafo.<BR>

Segunda línea fuera de párrafo.

<P>texto de párrafo. <BR> segunda línea de párrafo.</P>

</BODY>

</HTML>

Y si escribimos la siguiente hoja de estilo **estilo.css**:

BODY {color:green }

P {color:red;

font-size:20px;

font-family:Courier;

font-weight:bold;

font-style:italic;

line-height:30px;

letter-spacing:5px;

text-decoration:underline;

text-transform:capitalize;

text-align:left;

text-indent:30px;

}

Se obtiene:

Texto fuera de párrafo.  
Segunda línea fuera de párrafo.

***Texto De Párrafo.   
Segunda Línea De Párrafo***

Fíjate en la gran diferencia de estilo que hay entre las dos primeras líneas y las dos siguientes. Analicemos cómo funciona: Las dos primeras líneas de texto, en color verde, tienen todos sus valores por defecto, excepto el color que lo reciben de la etiqueta **BODY** definida en la css. Las dos siguientes, en rojo, reciben todas sus características de la etiqueta **P**, también definida en la css. Como puedes ver, hay una gran cantidad de atributos que actúan sobre el estilo de ese texto.

**color:**red;    Este ya lo conocemos. Define el color del texto. El color deseado puede escribirse directamente, en inglés, o puede utilizarse el sistema RGB en hexadecimal. Por ejemplo, este mismo color rojo en RGB sería: #FF0000;     Ciertos valores, como el rojo, pueden escribirse de forma abreviada: #F00;     Otra forma de declarar el color es mediante la función rgb() que puede parametrizarse en decimal o en tantos por ciento. He aquí un ejemplo con las cuatro formas del color rojo:

color: #ff0000

color: #f00

color: rgb(255,0,0)

color: rgb(100%, 0%, 0%)

Recuerda que en el índice tienes unas tablas con los [nombres](https://www.uv.es/jac/guia/texcolej.htm) y [códigos](https://www.uv.es/jac/guia/colorind.htm) de los colores.

**font-size:**20px;    También muy conocido: define el tamaño de los caracteres. Además del tamaño definido por el usuario, expresado en cualquiera de las unidades de medida conocidas, puede tener alguno de los siguientes valores absolutos: **xx-large   x-large   large   medium   small   x-small   xx-small**.

**font-family:**Courier;    Indica el nombre (o nombres) del tipo de letra que se va a emplear. En el ejemplo se ha utilizado el tipo "Courier", pero pueden escribirse varios separados por comas, lo que indica al navegador que si no existe en la máquina el primer tipo, utilice el segundo, y si tampoco, el tercero, etc. Por ejemplo: **font-family:**Courier,Verdana,Arial;      
Cuando definas tipos de letra, recuerda que muchos programas y algunos drivers de impresora instalan sus propias fuentes en el sistema sin avisarte, y puede que en tu máquina haya tipos que no son estándar en Windows. No se deben utilizar tipos extraños que difícilmente se encontrarán en las máquinas de los clientes, es decir, el tipo de fuente indicado en la hoja simplemente le indica al navegador qué fuente debe usar, NO se la sirve. En la versión 3 de CSS parece que se podrá indicar al cliente dónde obtener una fuente que no tenga instalada.

**font-weight:**bold;    Aquí se especifica el peso o grosor de la fuente. Pueden emplearse literales como **lighter**, **normal** (por defecto) o **bold**. También se pueden utilizar números entre **100** y **900**, escritos de 100 en 100 (no sirve 190, pero sí 200). Los pesos no tienen los mismos resultados en todas las máquinas, ya que depende mucho de la calidad de su pantalla, de su resolución, del navegador que emplea... En cualquier caso, no deben hacerse combinaciones extrañas, como definir un size muy pequeño y darle peso 900, ya que lo único que conseguirás es un borrón perfectamente ilegible.

**font-style:**italic;    Solamente tiene dos posibilidades: **normal** (por defecto) o **italic** que es el empleado en el ejemplo que estamos analizando, y que hace que el texto tenga cierta inclinación.

**line-height:**30px;    Sirve para establecer la distancia entre líneas consecutivas de un mismo párrafo. Además de en píxels o alguna de las unidades de medida que ya conocemos, puede indicarse con un simple número que viene a indicar, aproximadamente, cuantas anchuras de línea se tomarán como unidad de medida para separar una línea de otra. Así, por ejemplo, si pones **0** la segunda línea se suporpone a la primera.

**letter-spacing:**5px;    Establece la separación entre los caracteres de la línea.

**word-spacing:**5px;    Establece la separación entre las palabras de la línea.

**text-decoration:**underline;    Con este parámetro, que no influye en el tamaño, puedes acompañar al texto de una delgada línea decorativa que puede estar en tres posiciones distintas, como **underline** (el típico subrayado debajo de la línea), **through** (en el centro de la línea -tachado-) o **overline** (encima del texto). Para que no aparezca se utiliza **none** (por defecto). El valor none se puede utilizar para eliminar el efecto de subrayado que ponen otros elementos, como ocurre en los links.

**text-transform:**capitalize;    Curioso efecto que provee de cuatro posibilidades: **capitalize** que convierte en mayúscula la primera letra de cada palabra del texto, como puedes ver en el ejemplo, donde el texto original no es así. **uppercase** para convertir todas las letras a mayúsculas y **lowercase** para lo contrario, es decir, convertir todas las letras a minúsculas. El valor por defecto es **none**, que como ya habrás supuesto, deja el texto tal como está escrito.

**text-align:**left;    Alinea el texto según convenga. Con **left** a la izquierda (por defecto), **right** a la derecha, **center** centrado, y **justify** justificado, es decir, igualando todas las líneas en longitud a izquierda o derecha.

**text-indent:**30px    Produce que la primera línea del párrafo se escriba adentrada (indentada) un cierto espacio hacia la derecha o hacia la izquierda, dependiendo de la alineación activa.

Este sería un resumen sobre la declaración **font**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Declaración** | font-size | font-family | font-weight | font-style |
| **Valor por defecto** | medium | Definido por el usuario | normal | normal |
| **Valores posibles** | Absoluto, relativo, porcentaje, unidad de tamaño | Familia de fuente Windows | normal, lighter, bolder,  un valor entre 100 y 900 | normal, bold, italic |
| **Valor porcentual** | Se calcula respecto al padre | No aplicable | No aplicable | No aplicable |
| **Se aplica a** | Todos | Todos | Todos | Todos |
| **¿Se hereda?** | Sí | Sí | Sí | Sí |

Y este sobre el resto:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Declaración** | line-height | text-decoration | text-transform | text-align | text-indent | word-spacing | letter-spacing | white-space |
| **Valor por defecto** | normal | none | none | left | 0 | normal | normal | normal |
| **Valores posibles** | número, longitud, porcentaje, normal | none, underline, line-through, blink | capitalize, uppercase, lowercase, none, inherit | left, right, center, justify, none | longitud, porcentaje | normal, inherit, medida | normal, inherit, medida | normal, pre, nowrap, pre-wrap, pre-line, inherit |
| **Valor porcentual** | Se calcula respecto al tamaño de la fuente | No aplicable | No aplicable | No aplicable | Se calcula respecto al ancho del elemento padre | No aplicable | No aplicable | No aplicable |
| **Se aplica a** | Elementos a nivel de bloque | Todos | Todos | Elementos a nivel de bloque | Elementos a nivel de bloque | Todos | Todos | Todos |
| **Se hereda?** | Sí | No | Sí | Sí | Sí | Sí | Sí | Sí |

Tenemos también otras instrucciones que pueden modificar el aspecto de determinadas zonas en lugar de actuar sobre todo el bloque.   
Son los pseudo-elementos. Por ejemplo:

p :first-letter{

color:red;

}

p :first-line{

color:green;

}

p :after{

content:"Hola";

}

p :before{

content:"Por ejemplo:";

}

Como puedes ver, es fácilmente deducible lo que hace cada uno:   
**:first-letter** actuará sobre la primera letra del párrafo.   
**:first-line** lo hará sobre la primera línea.   
**:after content:"Hola";** actuará sobre el texto que haya después de la palabra "Hola" y  
**:before content:"Por ejemplo:";** antes de la frase "Por ejemplo:"

**Bordes de un bloque**

Con css se pueden definir los bordes de un bloque, que por defecto son invisibles. Por tanto, lo primero que hay que definir es el estilo del borde. Además del estilo, se puede definir su color y grosor, y todo ello puede hacerse globalmente, sobre los cuatro lados del bloque, o cada uno por separado. Las unidades de medida y los nombres o códigos de colores que pueden utilizarse, son los mismos que ya se han visto anteriormente para los textos. Para definir el grosor de los bordes tenemos:

* **border-left-width:**unidad   borde izquierdo
* **border-right-width:**unidad   borde derecho
* **border-top-width:**unidad   borde superior
* **border-bottom-width:**unidad   borde inferior
* **border-width:**unidad   los cuatros bordes

Donde unidad es una de las ya conocidas, por ejemplo 1px. También pueden utilizarse constantes tales como **thin** para fino (2px), **medium** para medio (4px), y **thick** para grueso (6px).

Ningun estilo de bordes (excepto **solid**) se lleva bien con el parámetro de grosor, que implícitamente establece su dimensionado más conveniente. Tenemos:

* **border-left-style:**estilo   borde izquierdo
* **border-right-style:**estilo   borde derecho
* **border-top-style:**estilo   borde superior
* **border-bottom-style:**estilo   borde inferior
* **border-style:**estilo   los cuatros bordes

Donde estilo puede ser:

* **none** (por defecto)
* **solid**
* **double**
* **ridge**
* **groove**
* **inset**
* **outset**
* **dotted**
* **dashed**

Por último, los parámetros de color de los bordes:

* **border-left-color:**color   borde izquierdo
* **border-right-color:**color   borde derecho
* **border-top-color:**color   borde superior
* **border-bottom-color:**color   borde inferior
* **border-color:**color   los cuatros bordes

Donde color puede ser un nombre de color, en inglés, o su código RGB en hexadecimal, como ya sabes.

Las combinaciones de todos estos parámetros son infinitas, y lo mejor es ir haciendo pruebas. Recuerda que cada navegador interpreta todo esto a su manera, y algunos no lo interpretan en absoluto. He aquí aplicado al ejemplo anterior sobre texto:

Si escribimos la siguiente hoja de estilo **estilo.css**:

BODY {color:green }

P {color:red;

font-size:20px;

font-family:Courier;

font-weight:bold;

font-style:italic;

line-height:30px;

letter-spacing:5px;

text-decoration:underline;

text-transform:capitalize;

text-align:left;

text-indent:30px;

border-width:2px;

border-color:blue;

border-style:solid;

}

Se obtiene:

Texto fuera de párrafo.  
Segunda línea fuera de párrafo.

***Texto De Párrafo.   
Segunda Línea De Párrafo***

Y este sería un resumen de su utilización:

|  |  |  |  |
| --- | --- | --- | --- |
| **Declaración** | border-left-width border-right-width  border-top-width border-bottom-width border-width | border-style | border-color  border-left-color  border-right-color border-top-color border-bottom-color |
| **Valor por defecto** | 0 | none | none |
| **Valores posibles** | Unidad de longitud,  thin, medium, thick | none, solid, double, inset,  outset, groove, ridge | none, color |
| **Valor porcentual** | No aplicable | No aplicable | No aplicable |
| **Se aplica a** | Todos | Todos | Todos |
| **¿Se hereda?** | No | No | No |

**Márgenes de los bloques**

Los bloques, por defecto, se escriben en la ventana activa comenzando en el ángulo superior izquierdo de la misma, y su contenido, en el interior del bloque, comienza en el mismo sitio. Esto es posible modificarlo:

Para los márgenes del bloque respecto a la ventana activa:

* **margin-left:**unidad   margen izquierdo
* **margin-right:**unidad   margen derecho
* **margin-top:**unidad   margen superior
* **margin-bottom:**unidad   margen inferior
* **margin:**unidad   los cuatros márgenes

Donde unidad es una de las ya conocidas.

Para los márgenes **dentro** del bloque, es decir, respecto al borde del bloque:

* **padding-left:**unidad   margen interno izquierdo
* **padding-right:**unidad   margen interno derecho
* **padding-top:**unidad   margen interno superior
* **padding-bottom:**unidad   margen interno inferior
* **padding:**unidad   los cuatros márgenes internos

En resumen:

|  |  |  |
| --- | --- | --- |
| **Declaración** | margin margin-left margin-right margin-top margin-bottom | padding padding-left padding-right padding-top padding-bottom |
| **Valor por defecto** | 0 | 0 |
| **Valores posibles** | Longitud, porcentaje, auto | Longitud, porcentaje |
| **Valor porcentual** | Se calcula respecto al ancho del padre | Se calcula respecto al ancho del padre |
| **Se aplica a** | Todos | Todos |
| **Se hereda?** | No | No |

Existe otra propiedad que consigue efectos muy parecidos a los vistos en los bordes: **outline** que sirve para crear contornos alrededor de los objetos, tales como botones, campos activos de los formularios, etc., a fin de resaltarlos, pero no se trata de bordes, sino de contornos, es decir, están ligeramente más hacia adentro, y por tanto no ocupan lugar extra en la página. La mala noticia es que no funciona en todos los navegadores.

**Dimensiones de los bloques**

Como ya se ha dicho más arriba, por defecto, y en función de su contenido, el bloque ocupará todo el ancho de la ventana y el alto que precise. Al componer la página puede que esto no interese y se quiera limitar el espacio ocupado por el bloque. Para ello disponemos de dos declaraciones básicas para indicar el ancho y el alto exactos que se desea, así como otras para definir el mínimo y máximo tamaño que deben alcanzar:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Declaración** | width | min-width | max-width | height | min-height | max-height |
| **Valor por defecto** | auto | Incierto | none | auto | 0 | none |
| **Valores posibles** | Longitud, porcentaje, auto, inherit | Longitud, porcentaje, inherit | Longitud, porcentaje, none, inherit | Longitud, porcentaje, auto, inherit | Longitud, porcentaje, inherit | Longitud, porcentaje, none, inherit |
| **Valor porcentual** | Se calcula respecto al ancho del padre | Se calcula respecto al ancho del padre | Se calcula respecto al ancho del padre | Se calcula respecto al alto del padre o auto | Se calcula respecto al alto del padre | Se calcula respecto al alto del padre |
| **Se aplica a** | Todos | Todos | Todos | Todos | Todos | Todos |
| **¿Se hereda?** | No | No | No | No | No | No |

Desafortunadamente, cada navegador interpreta las cosas de una forma, y es necesario probar con cuantos se posible. Sea, por ejemplo, la siguiente hoja de estilo **estilo.css**:

P {color:red;

font-size:20px;

font-family:Courier;

border:2px;

border-color:blue;

border-style:solid;

width:100px;

height:100px;

}

# COMANDOS BÁSICOS DE POSTGRESQL

A continuación, un ejemplo de comandos básicos para la administración de PostgreSQL. Siempre estará disponible el aplicativo Pgadmin, para una administración gráfica.

**Lista de Comandos:**

1. Listar usuarios:

postgres=# SELECT \* FROM pg\_user;

2. Crear usuario BD:

operador@equipo:/$ sudo createuser -s -U postgres nuevo\_usuario

3. Cambiar contraseña:

postgres=# ALTER USER postgres WITH PASSWORD '\*\*\*\*\*';

ALTER ROLE

4. Renombrar usuario:

postgres=# ALTER USER pedro RENAME TO admin;

ALTER ROLE

5. Crear BD:

postgres=# CREATE DATABASE "nombre\_bd" WITH ENCODING='UTF8' OWNER=usuario CONNECTION LIMIT=-1;

6. Lista bases de datos del sistema:

postgres=# SELECT datname FROM pg\_database ;

7. Respaldo BD:

operador@equipo:/$ sudo pg\_dump -Uusuario -dnombre\_bd -f /carpeta/destino/archivo.sql

8. Respaldo TABLA:

operador@equipo:/$ sudo pg\_dump --table table\_name -Uusuario -dnombre\_bd -f only\_table.sql

9. Restaurar TABLA respaldada:

operador@equipo:/$ sudo psql -f only\_table.sql -dnombre\_bd

10. Montar BD:

operador@equipo:/$ sudo psql -Uusuario -dnombre\_bd -f /carpeta/destino/archivo.sql

11. Dar todos los privilegios de una BD a un usuario:

postgres=# GRANT ALL PRIVILEGES ON DATABASE nombre\_bd to usuario;

12. Cambiar dueño de BD:

postgres=# ALTER DATABASE nombre\_bd OWNER TO usuario;

13. Respaldo BD remota:

operador@equipo:/$ sudo pg\_dump -O -Uusuario -p 5470 -h127.0.0.1 nmbre\_bd > /carpeta/destino/archivo.sql

**donde:**  
-O = –no-owner  
-U = usuario que establece la conexión a la BD  
-p = puerto de conexión BD  
-h = host donde se encuentra la BD

14. Restaurar respaldo de base de datos (.dump) remota con asignación de core de procesamiento:

operador@equipo:/$ pg\_restore -i -hlocalhost -p5432 -Upostgres -j 25 -d nombre\_bd -v -Fc '/ubicación/respaldo.dump'

**donde:**  
j -> es la cantida de core de procesamiento

15. Restaurar respaldo base de datos (.backup) remota

operador@equipo:/$ pg\_restore -i -h localhost -p 5432 -U usuario -d nombre\_bd -v "/ruta/archivo.backup"

16. Restaurar respaldo base de datos (.dump) local en la BD que el script del respaldo espacifica

operador@equipo:/$ pg\_restore -U usuario -C -d postgres respaldo.dump

**Nota: Crea la base de datos postgres (por la opción -C) pero siempre restaura los datos en la base que el script de respaldo especifica**

17. Restaura respaldo base de datos (.dump) local en una BD específica

operador@equipo:/$ pg\_restore -U usuario -d nombre\_bd respaldo.dump

18. Pasar un respaldo. dump a texto plano:

operador@equipo:/$ sudo pg\_restore respaldo.dump > respaldo.sql

**Nota: El archivo respaldo.sql puede ser leido con cualquier editor de texto**

19. Renombrar BD:

postgres=# ALTER DATABASE nombre\_anterior RENAME TO nuevo\_nombre;

20. Verificar los query o procesos dentro de postgres:

postgres=# SELECT \* FROM pg\_stat\_activity;

En consola SO:

operador@server:# psql -Upostgres -c "select \* from pg\_stat\_activity"

21. Detener un proceso en específico:

postgres=# SELECT pg\_cancel\_backend(procpid);

22. Detener una conexión de Base de datos:

postgres=# SELECT pg\_terminate\_backend(procpid);

23. Listar parámetro de log’s de postgres:

postgres=# select name,setting,unit from pg\_settings where name like '%log%';

24. Lista los Wal almacenados:

postgres=# show checkpoint\_segments;

# Función PostgreSQL STRING\_AGG

La STRING\_AGG()función PostgreSQL es una [función agregada](http://www.postgresqltutorial.com/postgresql-aggregate-functions/) que concatena una lista de cadenas y coloca un separador entre ellas. La función no agrega el separador al final de la cadena.

A continuación se muestra la sintaxis de la STRING\_AGG()función:

|  |  |
| --- | --- |
| 1 | STRING\_AGG ( expression, separator [order\_by\_clause] ) |

La STRING\_AGG()función acepta dos argumentos y una ORDER BYcláusula opcional .

* expressiones cualquier expresión válida que pueda resolverse en una cadena de caracteres. Si utiliza otros tipos que no sean el tipo de cadena de caracteres, debe [convertir](http://www.postgresqltutorial.com/postgresql-cast/) explícitamente estos valores de ese tipo al tipo de cadena de caracteres.
* separator Es el separador para cuerdas concatenadas.

El order\_by\_clausees una cláusula opcional que especifica el orden de los resultados concatenados. Tiene la siguiente forma:

EJEMPLO

SELECT

    f.title,

    STRING\_AGG (

a.first\_name || ' ' || a.last\_name,

        ','

       ORDER BY

        a.first\_name,

        a.last\_name

    ) actors

FROM

    film f

INNER JOIN film\_actor fa USING (film\_id)

INNER JOIN actor a USING (actor\_id)

GROUP BY

    f.title;

# PostgreSQL ISNULL

PostgreSQL no tiene la ISNULL función. Sin embargo, puede utilizar la [COALESCE](http://www.postgresqltutorial.com/postgresql-coalesce/) función que proporciona una funcionalidad similar. Tenga en cuenta que la COALESCE función devuelve el primer argumento no nulo, por lo que la siguiente sintaxis tiene el mismo efecto que la ISNULL función anterior:

|  |  |
| --- | --- |
| 1 | COALESCE(expression,replacement) |

Para el COALESCEejemplo, echa un vistazo a la [COALESCE](http://www.postgresqltutorial.com/postgresql-coalesce/) función tutorial.

Además de la COALESCEfunción, puede utilizar la [CASE](http://www.postgresqltutorial.com/postgresql-case/) expresión:

|  |  |
| --- | --- |
| 1  2  3  4  5 | SELECT      CASE WHEN expression IS NULL              THEN replacement              ELSE expression      END AS column\_alias; |

# Estructuras de control en PHP

**1. if**

La estructura de control **if** permite la ejecución condicional de fragmentos de código.

<?php

if ($x > $y) {

echo "$x es mayor que $y";

}

Las sentencias if se pueden incluir unas dentro de otras indefinidamente.

**2. else**

Sirve para ejecutar una sentencia cuando otra no se cumple. ***else*** extiende una sentencia ***if***, y se ejecuta cuando ésta es **false**. Siguiendo el ejemplo anterior:

<?php

if ($x > $y) {

echo "$x es mayor que $y";

} else {

echo "$y es mayor que $x";

}

**3. elseif/else if**

Es una combinación entre ***if*** y ***else***. Se ejecuta cuando if es **false**, pero sólo si la expresión condicional del ***elseif*** es **true**.

<?php

if ($x > $y) {

echo "$x es mayor que $y";

} elseif ($x == $y) {

echo "$x es igual que $y";

} else {

echo "$y es mayor que $x";

}

Puede haber varios ***elseif*** dentro de una sentencia ***if*** (aunque en ese caso suele ser más recomendable usar ***switch***). No hay diferencias de comportamiento entre ***elseif*** y ***else if***.

***elseif*** sólo se ejecuta si el ***if*** precedente o cualquier ***elseif*** anterior son evaluadas como **false**.

**4. while**

Es el tipo más sencillo de **loop** en **PHP**. Se ejecutan las sentencias dentro del ***while*** siempre y cuando se evalúen como **true**. El valor de la expresión se comprueba cada vez al inicio del **loop**, y la ejecución no se detendrá hasta que finalice la **iteración** (cada vez que PHP ejecuta las sentencias en un loop es una iteración). Si la expresión ***while*** se evalúa como **false**, las sentencias no se ejecutarán ni siquiera una vez.

También es posible agrupar varias instrucciones ***while*** dentro de una.

<?php

$i = 1;

while($i <= 10){

echo $i;

$i++;

}

**5. do-while**

Muy similares a los loops ***while***, simplemente aquí la expresión para el loop se verifica al final en lugar de al principio, esto garantiza que el código se ejecute por lo menos la primera vez.

<?php

$i = 0;

do {

echo $i;

} while ($i > 0);

Este loop se ejecutaría sólo una vez, ya que después no cumple la condición.

**6. for**

Los loops ***for*** son los más complejos en PHP.

<?php

for ($i = 1; $i <= 10; $i++) {

echo $i;

} // Devuelve 123456789

* Las expresiones o conjunto de expresiones van separadas por punto y coma; y sólo hay 3.
* La primera expresión, **$i = 1**, se ejecuta una vez incondicionalmente al comienzo del bucle.
* La segunda expresión, **$i <= 10**, es una **condición**, si es true, se ejecutará la tercera expresión.
* La tercera expresión, **$i++**, es la acción a realizar si se cumple la segunda expresión.

Cada una de las expresiones pueden estar **vacías** o contener **múltiples expresiones**, lo que resulta útil en ciertas ocasiones. Si la expresión 2 está vacía, el loop será definido como **true**.

<?php

// Todos los siguientes ejemplos son válidos y devuelven lo mismo, 123456789

// EJEMPLO 1

for($i = 1; $i <= 10; $i++) {

echo $i;

}

// EJEMPLO 2

for ($i = 1; ; $i++){

if($i > 10) {

break;

}

echo $i;

}

// EJEMPLO 3

$i = 1;

for( ; ; ){

if($i > 10){

break;

}

echo $i;

$i++;

}

// EJEMPLO 4

for ($i = 1, $j = 0; $i <= 10; $j += $i, print $i, $i++);

Dependiendo de donde se coloque el operador **incremento** o **decremento**, dará un resultado distinto:

for ($i = 1; $i <= 10; print $i++); // 12345678910

for ($i = 1; $i <= 10; print ++$i); // 234567891011

Se pueden hacer **iteraciones por medio de arrays**:

$gente = array(

array('nombre' => 'Carlos', 'salt' => 123),

array('nombre' => 'Maria', 'salt' => 124));

// Numero de gente

$size = count($gente);

// Loop que da un salt de 4 dígitos aleatorio

for ($i=0; $i < $size; ++$i) {

$gente[$i]['salt'] = mt\_rand(0000, 9999);

}

var\_dump($gente);

/\*

\* array (size=2)

0 =>

array (size=2)

'nombre' => string 'Carlos' (length=6)

'salt' => int 2029

1 =>

array (size=2)

'nombre' => string 'Maria' (length=5)

'salt' => int 9724

\*/

**7. foreach**

***foreach*** permite una forma fácil de iterar sobre **arrays** u **objetos**.

Cuando ***foreach*** inicia su ejecución, el puntero apunta directamente al primer elemento del **array**, por lo que no es necesario llamar a la función *reset()* antes que un **loop** *foreach*. Es recomendable no cambiar el puntero dentro del loop.

Se puede iterar de las siguientes dos formas:

<?php

// Devuelve directamente el value de cada key, comenzando desde el primero

foreach ($array as $value) {

// Lo que se desee mostrar

}

// Devuelve cada key con cada value, para poder trabajar con cualquiera de los dos

foreach ($array as $key => $value){

// Lo que se desee mostrar

}

Se pueden modificar los elementos del array dentro del loop, anteponiendo **&** a *$value* (asignándolo por referencia).

<?php

$array = array(1, 2, 3, 4);

foreach ($array as &$value){

$value = $value \* 2;

}

// cada valor del array vale ahora : 2, 4, 6, 8

unset($value);

La función *unset()* elimina la referencia del último elemento, pero el array sigue valiendo 2, 4, 6, 8. Es recomendable hacer *unset()*porque la referencia del *$valor* y el último elemento del array permanecen aún después del loop foreach.

[*foreach*](http://php.net/manual/es/control-structures.foreach.php) y *[list](http://php.net/manual/es/function.list.php)* pueden devolver los mismos resultados de la siguiente forma:

<?php

$array = array("uno", "dos", "tres");

reset($array);

while (list($clave, $valor) = each($array)){

echo "Clave: $clave; Valor: $valor </br>";

}

foreach ($array as $clave => $valor){

echo "Clave: $clave; Valor: $valor </br>";

}

La función *list()* coge un *array* y lo convierte en **variables** individuales.

*each()* coge un *array* y devuelve el key y value actuales, es decir, donde está apuntando el cursor en el array. Es necesario hacer un *[reset()](http://php.net/manual/es/function.reset.php)* del *array* para asegurarse que el cursor comienza desde el principio (cosa que no es necesaria con el foreach).

Suele ser más óptimo y legible utilizar el ***foreach*** para iterar, aunque ambos se utilizan más o menos con la misma frecuencia.

Con **arrays multidimensionales** se puede hacer un **doble foreach**:

<?php

$x = array();

$x[0][0] = "a";

$x[0][1] = "b";

$x[1][0] = "y";

$x[1][1] = "z";

foreach ($x as $primero){

foreach ($primero as $segundo){

echo "$segundo" . "</br>";

}

}

En el ejemplo anterior, si no se hace doble foreach con *$x* y se hace sólo un foreach surgirá un *Notice: Array to string conversion*.

Desde PHP 5.5 se puede recorrer un **array de arrays** y utilizar el segundo array para asignar variables:

<?php

$array = array(

array('azul', 'rojo'),

array('verde', 'amarillo')

);

foreach ($array as list($a, $b)) {

echo "A: $a; B: $b" . "</br>";

}

// Devuelve:

**8. break**

***break*** termina la ejecución de las siguientes estructuras: ***for***, ***foreach***, ***while***, ***do-while*** y ***switch***.

Se puede añadir un **argumento numérico opcional** que indica de cuántas estructuras debe salir. El valor por defecto es 1:

// Ejemplo sin valor numérico

$array = array('uno', 'dos', 'parar', 'tres');

while(list(, $valor) = each($array)){

if($valor == 'parar'){

break;

}

echo "$valor</br>";

}

// Ejemplo con valor numérico

$i = 0;

while (++$i){

switch($i){

case 5:

echo "He llegado a 5 </br>";

break 1; // Aquí sólo saldría del switch

case 10:

echo "He llegado a 10 </br>";

break 2; // Sale del switch y del while

default:

break;

}

}

**9. continue**

Se utiliza **dentro de las estructuras iterativas** para **saltar** el resto de la iteración actual del loop y **continuar a la siguiente iteración** para su ejecución:

<?php

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

if($i % 2 == 0)

continue;

print "$i ";

} // Devuelve 1 3 5 7 9

El código anterior hace que cuando el número sea par, ***continue*** se ejecute y no muestre el print de después, sino que vuelva a empezar con la siguiente iteración. ***print*** sólo se ejecutará cuando *$i* sea impar.

Al igual que con ***break***, se puede añadir un número a ***continue*** que indica el número de niveles de loops debe saltar:

<?php

$i = 0;

while ($i++ < 5){

echo "Uno </br>";

while (1) {

echo "Dos </br>";

while (1) {

echo "Tres </br></br>";

continue 3;

}

echo "Esto no aparece.";

}

echo "Esto tampoco aparece.";

}

En este ejemplo, cuando llega a ***continue 3***, comienza de nuevo la iteración, y se comprueba otra vez la condición ($i++ < 5). Si se pone, por ejemplo, ***continue 2***, salta al segundo while, cuya condición siempre se cumple, y se produce un **loop infinito** que imprime "Dos Tres" continuamente.

**10. switch**

***switch*** es como una serie de sentencias ***if***. Es útil para comparar una misma **variable o expresión** con valores diferentes y ejecutar un código diferente a otro dependiendo de esos valores.

<?php

switch ($i) {

case "perro":

echo "\$i es un perro";

break;

case "gato":

echo "\$i es un gato";

break;

case "avestruz":

echo "\$i es un avestruz";

break;

}

Cuando una sentencia ***case*** coincide con el valor de la sentencia ***switch***, PHP ejecuta el código dentro del *case*. PHP sigue ejecutando las sentencias hasta el final o hasta que choca con un ***break***, que entonces finaliza la iteración. Si se omite *break*, swith ejecutará todos los *cases* restantes cuando encuentra uno que cumpla con la condición:

<?php

switch ($i) {

case 0:

echo "i es igual a 0";

case 1:

echo "i es igual a 1";

case 2:

echo "i es igual a 2";

}

En el ejemplo anterior, si se define a *$i* como 0, se mostrarán todos los ***echo*** restantes porque no hay ningún ***break***. Si se define a *$i* como 1, se mostrarán los *echo* del *case* 1 y *case* 2. Si se define a *$i* como 2, sólo se mostrará el *case* 2.

Con ***switch*** la condición sólo se evalúa una vez, y su valor es comparado con cada uno de los *case*, a diferencia de lo que ocurre con ***elseif***, que la condión se va evaluando continuamente con el loop.

***case*** puede no ejecutar ningún código, pero hace que se activen los cases posteriores hasta que se encuentre con un break:

<?php

switch ($i) {

case 0:

case 1:

case 2:

echo "i es menor que 3 no negativo";

break;

case 3:

echo "i es 3";

}

En caso de que no haya ningún *case* válido, puede utilizarse ***default***, para ejecutar algo cuando no se cumplen los case:

<?php

switch($i) {

case 0:

echo "i es igual a 0";

break;

case 1:

echo "i es igual a 1";

break;

default:

echo "i no es ni 0 ni 1";

}

* En los cases sólo se permiten **tipos simples**: *int*, *float* y *string*. Los **arrays** y **objetos** pueden utilizarse si se muestran como un tipo simple.
* Es posible escribir punto y coma ";" en lugar de dos puntos ":" después de un *case*.

**11. declare**

***declare*** sirve para fijar directivas de ejecución para un bloque de código.

declare (directive)

statement

La sección directive permite configurar el comportamiento de declare. Sólo hay dos directivas: ***ticks*** y ***encoding***.

* No se pueden emplear variables ni constantes para las directivas.

Se pueden usar corchetes para fijar el statement o si no se aplican afecta a todo el archivo:

<?php

declare(ticks=1){

// Código afectado

}

declare(ticks=1);

// Código afectado

En este segundo ejemplo si el archivo con el declare ha sido incluído, no afectará al archivo padre.

**Ticks**

La ejecución de un **script PHP** puede representarse como una ejecución de muchas sentencias. La mayoría de las sentencias provocan un **tick** (aunque no todas). Por ejemplo en el siguiente ejemplo habría 3 ticks:

<?php

$x = 1;

$y = 2;

$z = 3;

Con **declare(ticks=N)** y \_[register\_tick\_function()](http://php.net/manual/es/function.register-tick-function.php)\_ se puede ejecutar código entre los **ticks**. La función \_register\_tick*function*especifica la función a llamar cuando un tick ocurre. declare establece cuantos ticks han de pasar hasta que un evento tick se ejecuta.

* Con declare(ticks=1) y \_register\_tick*function('miFuncion')*; llamará a *miFuncion()* después de cada tick.
* Con declare(ticks=3), *miFuncion()* se ejecutará cada 3 ticks.

<?php

function handler(){

echo "Hola";

}

register\_tick\_function("handler");

$i = 0;

declare(ticks = 3) {

while($i < 9)

echo ++$i;

} // Devuelve 123Hola456Hola789Hola

**Encoding**

La codificación puede ser especificada para cada script con encoding:

<?php

declare(encoding='ISO-8859-1');

// Código a ejecutar

**12. return**

***return*** devuelve el control del programa al módulo que lo invoca. La ejecución vuelve a la siguiente declaración después del módulo que lo invoca.

* Si se usa en una **función**, return hace que la función termine, devolviendo los argumentos que le sigan como valor de la llamada a la función.
* Si se llama **globalmente**, finaliza la ejecución del script actual.
* Si el archivo actual fue incluido o requerido, el control regresa al archivo que llama al ***include*** o ***require***.
* Si el archivo está incluído con ***include***, los argumentos del return se devolverán como valor de la llamada ***include***.
* Si return se usa en el **archivo principal del script**, termina la ejecución del script.
* También termina la ejecución de una sentencia ***eval()***.

Es recomendable **no** usar paréntesis después de return.

**13. include/include\_once**

***include*** incluye y ejecuta un archivo.

Los archivos se incluyen en base a la **ruta de acceso dada**, y si no se proporciona ninguna, se utiliza el [include\_path](http://php.net/manual/es/ini.core.php" \l "ini.include-path). Si el archivo tampoco se encuentra en el **include\_path** se mirará en el **propio directorio** desde donde se hace la llamada, antes de devolver un **mensaje warning**. Es en el tipo de mensaje donde difiere con require, que devuelve un **error fatal**.

* Si se define una ruta absoluta (en Linux comenzando por /) o relativa al directorio actual (comenzando con . o ..) el **include\_path**será ignorado.

Las variables del archivo del include estarán disponibles en el archivo desde el que se solicita:

<?php

// archivo1.php

$color = 'azul';

// archivo2.php

echo $color; // Devuelve un Notice: Undefined variable: color

include 'archivo1.php';

echo $color; // Devuelve azul

* Si la inclusión se hace dentro de una **función**, el contenido del archivo es como si estuviera dentro de esa función, por tanto su contenido tendrá el mismo ámbito.
* Cuando se incluye un archivo, el intérprete abandona el modo PHP e ingresa al modo HTML al comienzo del archivo incluído, y se reanuda de nuevo al final. Es por ello que cualquier código que tenga que ser interpretado como PHP debe incluir las etiquetas válidas de comienzo y terminación (**<?php ?>**).

Si están activadas las **envolturas URL include**, se puede incluir un archivo a través de una URL (mediante HTTP u [otro protocolo](http://php.net/manual/es/wrappers.php)). Si el servidor objetivo interpreta el archivo como PHP, las variables se pueden pasar al archivo usando un string de petición como con HTTP GET. El resultado no es lo mismo que en local, pues el archivo se ejecuta en el servidor remoto y el resultado se incluye en el local.

<?php

include 'http://www.ejemplo.com/archivo.php?var=1';

$var = 1;

Por **seguridad** es mejor que el archivo remoto se procese en el servidor remoto y se reciba sólo la salida, con *[readfile()](http://php.net/manual/es/function.readfile.php)*.

Es posible devolver valores desde los archivos include mediante **return**:

// archivo1.php con return

<?php

$var = 'PHP';

return $bar;

?>

// archivo2.php sin return

<?php

$var = 'PHP';

?>

//archivo3.php

<?php

$foo = include 'archivo1.php';

echo $foo; // devuelve PHP

$bar = include 'archivo2.php';

echo $bar; // Devuelve 1 porque el include ha sido exitoso, pero no tiene valor return. Si no hubiera funcionado devolvería false y un E\_WARNING.

También se pueden incluir archivos PHP en variables con un buffering de salida:

<?php

$string = get\_include\_contents('archivo.php');

function get\_include\_contents($filename) {

if(is\_file($filename)) {

ob\_start();

include $filename;

return ob\_get\_clean();

}

return false;

}

Para incluir archivos automáticamente en scripts ver [auto\_preprend\_file](http://php.net/manual/es/ini.core.php" \l "ini.auto-prepend-file) y [auto\_append\_file](http://php.net/manual/es/ini.core.php" \l "ini.auto-append-file) de *php.ini*.

\_**include\_once**\_ incluye el archivo especificado sólo una vez, si se incluye más veces tan sólo devuelve true. Es útil en casos donde el mismo fichero se podría incluir y evaluar más de una vez, para evitar así redefinir funciones o reasignar valores a variables.

**14. require/require\_once**

***require*** hace lo mismo que ***include*** pero en caso de fallar devuelve un **error fatal** de nivel **E\_COMPILE\_ERROR**, por lo que no puede continuar el script. *include* sólo emite un **E\_WARNING** que permite continuar el script.

\_**require\_once***es igual que***require**\_ pero PHP comprobará si el archivo ya ha sido incluído, y si es así no se incluirá otra vez.

**15. goto**

***goto*** se utiliza para saltar a otra sección del script. La **etiqueta de destino** debe estar dentro del mismo fichero y ámbito.

<?php

goto x;

echo 'Hola!';

x:

echo 'Adios!'; // sólo se imprimirá Adios!

También puede utilizarse en un **loop** en lugar de ***break***:

<?php

for($i=0, $j=50; $i<100; $i++) {

while($j--) {

if($j==17) goto end;

}

}

echo "i = $i";

end:

echo 'j llegó a 17';

**16. Sintaxis alternativas**

PHP ofrece una **sintaxis alternativa** para *if*, *while*, *for*, *foreach* y *switch*. Se cambia el corchete de apertura por dos puntos ":" y el corchete de cierre por ***endif***, ***endwhile***, ***endfor***, ***endforeach***, o ***endswitch***.

<?php if ($x == 3): ?>

X es 3

<?php endif; ?>

"X es 3" es un bloque HTML que se mostraría sólo si se cumple la condición.

Con **else** y **elseif**:

<?php

if($x == 3):

echo "x igual a 3";

elseif ($x == 4):

echo "x igual a 4";

else:

echo "x no es ni 3 ni 4";

endif;

WAMP

**WAMP** es el [acrónimo](https://es.wikipedia.org/wiki/Acr%C3%B3nimo) usado para describir un sistema de infraestructura de Internet que usa las siguientes [herramientas](https://es.wikipedia.org/wiki/Herramienta):

* [**Windows**](https://es.wikipedia.org/wiki/Microsoft_Windows) como [sistema operativo](https://es.wikipedia.org/wiki/Sistema_operativo);
* [**A**pache](https://es.wikipedia.org/wiki/Servidor_HTTP_Apache), como [servidor web](https://es.wikipedia.org/wiki/Servidor_web);
* [**M**ySQL](https://es.wikipedia.org/wiki/MySQL), como [gestor de bases de datos](https://es.wikipedia.org/wiki/Gestor_de_bases_de_datos);
* [**P**HP](https://es.wikipedia.org/wiki/PHP) (generalmente), [**P**erl](https://es.wikipedia.org/wiki/Perl), o [**P**ython](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_Python), como [lenguajes de programación](https://es.wikipedia.org/wiki/Lenguajes_de_programaci%C3%B3n).

El uso de un WAMP permite subir páginas [html](https://es.wikipedia.org/wiki/Html" \o "Html) a [internet](https://es.wikipedia.org/wiki/Internet), además de poder gestionar datos en ellas. Al mismo tiempo, un WAMP proporciona lenguajes de programación para desarrollar aplicaciones [web](https://es.wikipedia.org/wiki/Web).

[LAMP](https://es.wikipedia.org/wiki/LAMP) es el sistema análogo que corre bajo ambiente [Linux](https://es.wikipedia.org/wiki/Linux)

[MAMP](https://es.wikipedia.org/wiki/MAMP) es el sistema análogo que corre bajo ambiente [Macintosh](https://es.wikipedia.org/wiki/Macintosh)

**Paquetes gráficos:**

GD (Graphics Draw): la librería de dibujo de gráficos.

libpng: la librería oficial de referencia de PNG.

libpeg: la librería oficial de referencia de JPEG.

ncurses: la librería de gráficos de caracteres.

# Bibliografía

WAMP

https://es.wikipedia.org/wiki/WAMP

PHP

https://diego.com.es/estructuras-de-control-en-php

http://www.pablin.com.ar/computer/cursos/php/func.html

HTML

http://paginaspersonales.deusto.es/airibar/Ed\_digital/HTML/HTML\_1.html

CSS

https://www.uv.es/jac/guia/estilo2.htm

POSTGRESSQL

https://juantrucupei.wordpress.com/2015/03/24/comandos-basicos-de-postgresql/

http://www.postgresqltutorial.com/postgresql-aggregate-functions/postgresql-string\_agg-function/

http://www.postgresqltutorial.com/postgresql-isnull/

http://www.postgresqltutorial.com/postgresql-aggregate-functions/postgresql-array\_agg-function/