8. Lenguaje de programación PHP.

INTRODUCCIÓN A LA PROGRAMACIÓN EN PHP

PHP es uno de los lenguajes de lado del servidor más extendidos en la red. Nacido en 1994, se trata de un lenguaje de creación relativamente reciente que ha tenido una gran aceptación debido sobre todo a su potencia y simplicidad.

PHP nos permite embeber los fragmentos de su código dentro de páginas HTML y realizar determinadas acciones de una forma fácil y eficaz sin tener que generar programas programados íntegramente en un lenguaje distinto al HTML. Por otra parte, y es aquí donde reside su mayor interés con respecto a lenguajes pensados para los CGI, PHP ofrece un sinfín de funciones para la explotación de bases de datos de una manera sencilla.

Otro conocido lenguaje de programación es ASP. Si bien PHP y ASP son lenguajes parecidos en cuanto a potencia y dificultad, su sintaxis difiere sensiblemente. Algunas de sus diferencias principales son:

- PHP, aunque multiplataforma, ha sido concebido inicialmente para entornos UNÍS y es en este sistema operativo donde se pueden aprovechar mejor sus prestaciones. ASP, siendo una tecnología Microsoft, está orientado hacia sistemas Windows, especialmente NT.
- Las tareas principales que puede realizar directamente el lenguaje son definidas en PHP como funciones mientras que ASP invoca más frecuentemente los objetos. Por supuesto, esto no es más que una simple cuestión de forma ya que ambos lenguajes soportan igualmente ambos procedimientos.
- ASP realiza numerosas tareas sirviéndose de componentes (objetos) que deben ser comprados a empresas especializadas o programados por el servidor. PHP presenta una filosofía totalmente diferente, es progresivamente construido por los usuarios que implementan nuevas funciones en nuevas versiones del lenguaje.

BREVE HISTORIA DE PHP

PHP es un lenguaje creado por una gran comunidad de personas. El sistema fue desarrollado originalmente en el año 1994 por Rasmus Lerdorf como un CGI en C que permitía la interpretación de un número limitado de comandos. El sistema fue denominado Personal Home Page Tools y adquirió relativo éxito gracias a que otras personas pidieron a Rasmus poder utilizar los programas que desarrolló en sus propias páginas. Dada la aceptación del primer PHP y de manera adicional, su creador diseñó un sistema para procesar formularios al que le atribuyó el nombre de FI (Form Interpreter) y el conjunto de estas dos herramientas sería la primera versión compacta del lenguaje PHP/FI.

La siguiente contribución al lenguaje se realizó a mediados del 97 cuando se volvió a programar el analizador sintáctico, se incluyeron nuevas funcionalidades como el soporte a nuevos protocolos de Internet y el soporte a la gran mayoría de las bases de datos comerciales. Todas estas mejoras sentaron las bases de PHP versión 3. Actualmente PHP se encuentra en su versión 4, que utiliza el motor Zend, desarrollado con mayor meditación para cubrir las necesidades actuales y

solucionar algunos inconvenientes de la anterior versión. Algunas mejoras de esta nueva versión son su rapidez, gracias a que primero se compila y luego se ejecuta mientras que antes se ejecutaba mientras se interpretaba el código, su mayor independencia del servidor web, creando versiones de PHP nativas para más plataformas, y un API más elaborado y con más funciones.

En los últimos años el número de servidores que utilizan PHP se ha disparado logrando situarse por encima de 5 millones de sitios y más de 800.000 direcciones IP. El hecho de que PHP se haya convertido en una tecnología popular se debe, entre otras razones, a que es el complemento ideal para que el tándem Linux-Apache sea compatible con la programación del lado del servidor de sitios web. Gracias a la aceptación lograda y a los esfuerzos realizados por una creciente comunidad de colaboradores para implementarlo de la manera más óptima, podemos asegurar que este lenguaje se convertirá en un estándar de éxito entre los sistemas desarrollados en código abierto.

PHP: LENGUAJE DEL LADO DEL SERVIDOR

El navegador es una espacie de aplicación capaz de interpretar las órdenes recibidas en forma de código HTML, fundamentalmente, y convertirlas en páginas que son el resultado de dicha orden.

Al pinchar sobre un enlace hipertexto lo que pasa es que establecemos una petición de un archivo HTML residente en el servidor (un ordenador que se encuentra continuamente conectado a la red) el cual es enviado e interpretado por nuestro navegador (el cliente).

Si la página que pedimos no es un archivo HTML, el navegador es incapaz de interpretarla y lo único que es capaz es de salvarla en forma de archivo. Es por ello que si queremos emplear lenguajes accesorios para realizar un sitio web, es absolutamente necesario que sea el propio servidor quien los ejecute e interprete para luego enviarlos al cliente en forma de archivo HTML totalmente legible por él.

Así pues tenemos lenguajes de lado servidor, que son aquellos reconocidos, ejecutados e interpretados por el propio servidor y que se envían al cliente en un formato comprensible por él , y lenguajes de lado cliente (entre los que no sólo se encuentra HTML sino también Java y JavaScript los cuales son simplemente incluidos en el código HTML), que son directamente "digeridos" por el navegador y no necesitan un pretratamiento.

Cada tipo tiene sus ventajas e inconvenientes. Así un lenguaje de lado cliente es totalmente independiente del servidor lo cual permite, por ejemplo, que la página pueda ser albergada en cualquier sitio sin necesidad de pagar más ya que, por regla general, los servidores que aceptan páginas con scripts de lado servidor son en su mayoría de pago o sus prestaciones son muy limitadas. Inversamente, un lenguaje de lado servidor es independiente del cliente por lo que es mucho menos rígido respecto al cambio de un navegador a otro o respecto a las versiones del mismo. Por otra parte, los scripts son almacenados en el servidor de quien los ejecuta y traduce a HTML por lo que permanecen ocultos para el cliente. Este hecho puede resultar una forma legítima de proteger el trabajo intelectual realizado.

En el dominio de la red, los lenguajes de lado servidor más ampliamente utilizados para el desarrollo de páginas dinámicas son el ASP, PHP y PERL.

PERL es el lenguaje más rápido y potente por lo que obviamente requiere un aprendizaje más largo quedando más reservado para personas ya familiarizadas con la programación.

El ASP (Active Server Pages) es un lenguaje derivado del Visual Basic desarrollado por Microsoft. Evidentemente su empleo se realiza sobre plataformas funcionando bajo sistema Windows NT.

El PHP podría ser considerado como un lenguaje análogo al ASP utilizado en plataformas Unix y Linux.

Estos dos lenguajes resultan bastantes útiles para la explotación de bases de datos y su aprendizaje es accesible para una persona profana en programación. Cualquiera de ellos resultaría una buena opción a la hora de hacer evolucionar un sitio web realizado en HTML.

TAREAS PRINCIPALES DEL PHP

En un principio diseñado para realizar un poco más que un contador y un libro de visitas, PHP ha experimentado en poco tiempo una verdadera revolución y, a partir de sus funciones, en estos momentos permite realizar una multitud de tareas útiles para el desarrollo web:

- **Gestión de bases de datos**

Resulta difícil concebir un sitio actual, potente y rico en contenido que no sea gestionado por una base de datos. El lenguaje PHP ofrece interfaces para el acceso a la mayoría de las bases de datos comerciales y por ODBC a todas las bases de datos posibles en sistemas Microsoft, a partir de las cuales podremos editar el contenido de nuestro sitio con absoluta sencillez.

Efectivamente uno de los puntos fuertes de las páginas en PHP es la posibilidad de explotar bases de datos mediante funciones de una simplicidad y potencia muy agradecidas. Estas bases de datos pueden servir a nuestro sitio para almacenar contenidos de una forma sistemática que nos permita clasificarlos, buscarlos y editarlos de una forma fácil y rápida. La base de datos más difundida con el tandem UNIX- Apache es sin duda MySQL, existiendo una versión disponible para Windows.

- **Gestión de archivos**

Crear, borrar, mover, modificar,...cualquier tipo de operación más o menos razonable que se nos pueda ocurrir puede ser realizada a partir de una amplia librería de funciones para la gestión de archivos por PHP. También podemos transferir archivos por FTP a partir de sentencias en nuestro código, protocolo para el cual PHP ha previsto también gran cantidad de funciones.

- **Tratamiento de imágenes**

Evidentemente resulta mucho más sencillo utilizar Photoshop para el tratamiento de imágenes pero PHP es útil, por ejemplo, cuando tenemos que tratar gran cantidad de imágenes enviadas por nuestros internautas. Puede resultar muy tedioso uniformar en tamaño y formato miles de imágenes recibidas día tras día pero con PHP podemos automatizarlo eficazmente.

Funciones de correo electrónico

Podemos, con gran facilidad, enviar un e-mail a una persona o lista parametrizando toda una serie de aspectos tales como el e-mail de procedencia, asunto, persona a responder,...

Muchas otras funciones pensadas para Internet (tratamiento de cookies, accesos restringidos, comercio electrónico,...) o para propósito general (funciones matemáticas, explotación de cadenas, de fechas, corrección ortográfica, comprensión de archivos, botones dinámicos,...) son realizadas por este lenguaje. A esta inmensa librería cabe ahora añadir todas las funciones personales que uno va creando por necesidades propias y que luego son reutilizadas en otros sitios y todas ellas intercambiadas u obtenidas en foros o sitios especializados.

INSTALACIÓN DE PHP EN NUESTRO SERVIDOR

Como todo leguaje del lado del servidor, PHP requiere de la instalación de un servidor en nuestro PC para poder trabajar en local. Este modo de trabajo es evidentemente más práctico que colgar los archivos por FTP en el servidor y ejecutarlos desde nuestro navegador.

Así pues, antes de crear programas en PHP, necesitamos:

- Convertir nuestro ordenador en un servidor. Esto se hace instalando uno de los varios servidores disponibles para el sistema operativo de nuestra máquina.
- Introducir en nuestro servidor los archivos que le permitirán la comprensión del PHP.

Si queremos trabajar con PHP una posibilidad para los usuarios de Windows es instalar Apache como servidor web lo cual resulta más ventajoso con respecto al uso del PWS ya que PHP está especialmente diseñado para correr en este servidor. Esto quiere decir que, aunque en principio todo debería funcionar correctamente en ambos servidores, es posible que algún bug no corregido haga fallar uno de nuestros scripts si trabajamos para un servidor cuyas actualizaciones son menos frecuentes y detalladas.

Aunque las mejores prestaciones de este lenguaje son obtenidas trabajando en entorno UNIX con un servidor Apache, para fines de desarrollo local podemos contentarnos con emplear la versión Windows.

SINTAXIS DE PHP

PHP se escribe dentro de la propia página web, junto con el código HTML y, como para cualquier otro tipo de lenguaje incluido en un código HTML, en PHP necesitamos especificar cuáles son las partes constitutivas del código escritas en este lenguaje. Esto se hace, como en otros casos, delimitando nuestro código por etiquetas. Podemos utilizar distintos modelos de etiquetas:

```
<? y ?>
<% y %>
<?php y ?>
<script languaje="php">
```

El ultimo modo está principalmente aconsejado para trabajar con Front Page ya que, usando cualquier otro tipo de etiqueta, corremos el riesgo de que la aplicación nos la borre sin más debido a que se trata de un código incomprensible para ella.

El modo de funcionamiento de una página PHP, a grandes rasgos, no difiere del clásico para una página dinámica de lado servidor: El servidor va a reconocer la extensión correspondiente a la página PHP (phtml, php, php4,...) y antes de enviarla al navegador va a encargarse de interpretar y ejecutar todo aquello que se encuentre entre las etiquetas correspondientes al lenguaje PHP. El resto, lo enviara sin más ya que, asumirá que se trata de código HTML absolutamente comprensible por el navegador.

Otra característica general de los scripts en PHP es la forma de separar las distintas instrucciones. Para hacerlo, hay que acabar cada instrucción con un punto y coma ";". Para la ultima expresión, la que va antes del cierre de etiqueta, este formalismo no es necesario.

Un elemento importante son los comentarios. Un comentario es una frase o palabra que nosotros incluimos en el código para comprenderlo más fácilmente al volverlo a leer un tiempo después y que, por supuesto, el ordenador tiene que ignorar ya que no va dirigido a él sino a nosotros mismos. La forma de incluir estos comentarios es variable dependiendo si queremos escribir una línea o más. Veamos esto con un primer ejemplo de script:

```
<?
$mensaje="hola!!"; //Comentario de una linea
echo $mensaje; # Este comentario también es de una linea
/* En este caso
mi comentario ocupa
varias lineas */
?>
```

Si usamos doble barra // o el símbolo # podemos introducir comentarios de una línea. Mediante /* y */ creamos comentarios multilínea. Para comprender el script adelantamos que las variables en PHP se definen anteponiendo un símbolo de dólar ($) y que la instrucción *echo* sirve para sacar en pantalla lo que hay escrito a continuación.

Variables en PHP

Para PHP, las variables son definidas anteponiendo el símbolo dólar ($) al nombre de la variable que estábamos definiendo.

Dependiendo de la información que contenga, una variable puede ser considerada de uno u otro tipo:

- Variables numéricas. Almacenan números, p.ej.:
  - Enteros. Números sin decimales: `$entero=2003`.
  - Real. Números con o sin decimales: `$real=3.1416`.
- Variables alfanuméricas. Almacenan textos compuestos de números y/o cifras.
  - Cadenas. Almacenan variables alfanuméricas: `$cadena="hola,¿qué tal?"`.
- Tablas. Almacenan series de información numéricas y/o alfanuméricas.
  - Arrays. Guardan tablas: `$sentido(1)="oído";`
    `$sentido(2)="vista";`

```
$sentido(3)= "olfato";
$sentido(4)= "gusto";
$sentido(5)= "tacto";
```

- Objetos. Se trata de un conjunto de variables y funciones asociadas.

PHP posee una gran flexibilidad a la hora de operar con variables. En efecto, cuando definimos una variable asignándole un valor, el ordenador le atribuye un tipo, p.ej.: `$cadena= "5"; //esto es una cadena`, `$entero= 5; //esto es un entero`. Sin embargo hay que tener cuidado en no cambiar mayúsculas por minúsculas ya que, en este sentido, PHP es sensible.

Variables de sistema

Dada su naturaleza de lenguaje de lado servidor, PHP es capaz de darnos acceso a toda una serie de variables que nos informan sobre nuestro servidor y sobre el cliente. La información de estas variables es atribuida por el servidor y en ningún caso nos es posible modificar sus valores directamente mediante el script. Para hacerlo es necesario influir directamente sobre la propiedad que definen.
Existen multitud de variables de este tipo. Veamos algunas de estas variables y la información que nos aportan:

- `$HTTP_USER_AGENT`. Nos informa principalmente sobre el sistema operativo y tipo y versión de navegador utilizado por el internauta.
- `$HTTP_ACCEPT_LANGUAGE`. Nos devuelve la o las abreviaciones de la lengua considerada como principal por el navegador.
- `$HTTP_REFERER`. Nos indica la URL desde la cual el internauta ha tenido acceso a la página.
- `$PHP_SELF`. Nos devuelve una cadena con la URL del script que está siendo ejecutada.
- `$HTTP_COOKIES_VARS`. Se trata de un array que almacena los nombres y contenidos de las cookies. Veremos qué son más adelante.
- `$REMOTE_ADDR`. Muestra la dirección IP del visitante.
- `$PHPSESSID`. Guarda el identificador de sesión del usuario. Veremos más adelante en qué consisten las sesiones.

Tablas en PHP

Un array es una variable que está compuesta de varios elementos cada uno de ellos catalogado dentro de ella misma por medio de una clave. El ejemplo visto antes era:

```
$sentido(1)= "oído";
$sentido(2)= "vista";
$sentido(3)= "olfato";
$sentido(4)= "gusto";
$sentido(5)= "tacto";
```

En este caso el array cataloga sus elementos, comunmente llamados valores, por números. Los números del 1 al 5 son por lo tanto las claves y los sentidos son los valores asociados. Nada nos impide emplear nombres (cadenas) para clasificarlos. Lo único que deberemos hacer es entrecomillarlos:

```
<?
$moneda("espana")="Madrid";
$moneda("francia")="París";
$moneda("italia")="Roma";
?>
```

Otra forma de definir idénticamente este mismo array y que nos puede ayudar para la creación de arrays más complejos es la siguiente sintaxis:

```
<?
$moneda=array("espana"=> "Madrid","francia" => "París",
"italia" => "Roma");
?>
```

Una forma muy práctica de almacenar datos es mediante la creación de arrays multidimensionales (tablas). Veamos el ejemplo siguiente: queremos almacenar dentro de una misma tabla el nombre, capital y lengua hablada en cada país. Para hacerlo podemos emplear un array llamado país que vendrá definido por estas tres características (claves). Para crearlo, deberíamos escribir una expresión del mismo tipo que la vista anteriormente en la que meteremos una array dentro del otro. Este proceso de incluir una instrucción dentro de otra se llama anidar y es muy corriente en programación:

```
<?
$pais=array
(
"espana" =>array
  (
  "nombre"=>"España",
  "lengua"=>"Castellano",
  "capital"=>"Madrid"
  ),
"francia" =>array
  (
  "nombre"=>"Francia",
  "lengua"=>"Francés",
  "capital"=>"París"
  )
);
echo $pais("espana")("capital") //Saca en pantalla: "Madrid"
?>
```

Como puede verse, en esta secuencia de script, no hemos introducido punto y coma ";" al final de cada línea. Esto es simplemente debido a que lo que hemos escrito puede ser considerado como una sola instrucción.

Este programa nos permite almacenar tablas y, a partir de una simple petición, visualizar un determinado valor en pantalla. Pero la utilidad de los arrays no acaba aquí, sino que también podemos utilizar toda una serie de funciones creadas para ordenarlos por orden alfabético directo o inverso, por claves, contar el número de elementos que componen el array además de poder movernos por dentro de él hacia delante o atrás. Algunas de estas funciones son:

- ARRAY_VALUES (MI_ARRAY). Lista los valores contenidos en mi_array.
- ASORT(MI_ARRAY) Y ARSORT(MI_ARRAY). Ordena por orden alfabético directo o inverso en función de los valores.
- COUNT(MI_ARRAY). Nos da el número de elementos de nuestro array.
- KSORT(MI_ARRAY) Y KRSORT(MI_ARRAY). Ordena por orden alfabético directo o inverso en función de las claves.
- LIST ($VARIABLE1, $VARIABLE2...)=MI_ARRAY. Asigna cada una variable a cada uno de los valores del array.
- NEXT(MI_ARRAY), PREV(MI_ARRAY), RESET(MI_ARRAY) y END(MI_ARRAY). Nos permiten movernos por dentro del array con un puntero hacia delante, atrás y al principio y al final.
- EACH(MI_ARRAY). Nos da el valor y la clave del elemento en el que nos encontramos y mueve al puntero al siguiente elemento.

De gran utilidad es también el bucle FOREACH que recorre de forma secuencial el array de principio a fin.

Cadenas

Las cadenas son información de carácter no numérico (textos, por ejemplo). Para asignar a una variable un contenido de este tipo, lo escribiremos entre comillas dando lugar a declaraciones de este tipo:

```
$cadena="Esta es la información de mi variable"
```

Podemos yuxtaponer o concatenar varias cadenas poniendo para ello un punto entre ellas:

```
<?
$cadena1="Hola";
$cadena2=" ¿Qué tal?";
$cadena3=$cadena1.$cadena2;
echo $cadena3 //El resultado es: "Hola ¿Qué tal?"
?>
```

También podemos introducir variables dentro de nuestra cadena:

```
<?
$a=55;
$mensaje="Tengo $a años";
echo $mensaje //El resultado es: "Tengo 55 años"
?>
```

En este caso el símbolo $ define una variable pero puede interesarnos meterlo por razones comerciales significando dólar. Pues bien, para meter éste y otros caracteres utilizados por el lenguaje dentro de las cadenas y no confundirlos, lo que hay que hacer es escribir una contrabarra delante:

- \$. Escribe dólar en la cadena.
- \". Escribe comillas en la cadena.
- \\. Escribe contrabarra en la cadena.

- \8/2. Escribe 8/2 y no 4 en la cadena.

Además, existen otras utilidades de esta contrabarra que nos permiten introducir en nuestro documento HTML determinados eventos:

- \t. Introduce una tabulación en nuestro texto.
- \n. Cambiamos de línea.
- \r. Retorno de carro.

Estos cambios de línea y tabulaciones tienen únicamente efecto en el código y no en el texto ejecutado por el navegador. Así si queremos que nuestro texto ejecutado cambie de línea hemos de introducir "*
" y no "\n*" ya que este ultimo sólo cambia de línea en el archivo HTML creado y enviado al navegador cuando la página es ejecutada en el servidor.

Las cadenas pueden asimismo ser tratadas por medio de funciones de todo tipo, existen muchas posibles acciones que podemos realizar sobre ellas: dividirlas en palabras, eliminar espacios sobrantes, localizar secuencias, remplazar caracteres especiales por su correspondiente en HTML o incluso extraer las etiquetas META de una página web.

Funciones

Una función puede ser definida como un conjunto de instrucciones que explotan ciertas variables para realizar una tarea más o menos elemental.

PHP basa su eficacia principalmente en este tipo de elemento. Una gran librería que crece constantemente, a medida que nuevas versiones van surgiendo, es complementada con las funciones de propia cosecha dando como resultado un sinfín de recursos que son aplicados por una simple llamada.

Las funciones integradas en PHP son muy fáciles de utilizar. Tan sólo hemos de realizar la llamada de la forma apropiada y especificar los parámetros y/o variables necesarios para que la función realice su tarea.

Lo que puede parecer ligeramente más complicado, pero que resulta sin lugar a dudas muy práctico, es crear nuestras propias funciones. De una forma general, podríamos crear nuestras propias funciones para conectarnos a una base de datos o crear los encabezados o etiquetas meta de un documento HTML. Para una aplicación de comercio electrónico podríamos crear por ejemplo funciones de cambio de una moneda a otra o de calculo de los impuestos a añadir al precio de articulo. En definitiva, es interesante crear funciones para la mayoría de acciones más o menos sistemáticas que realizamos en nuestros programas.

Veamos por ejemplo la creación de una función que, llamada al comienzo de nuestro script, nos crea el encabezado de nuestro documento HTML y coloca el titulo que queremos a la página:

```
<?
Function hacer_encabezado($titulo)
{
$encabezado="<html> \n<head> \n\t<title>$titulo</title> \n<
/head> \n";
echo $encabezado;
}
?>
```

Esta función podría ser llamada al principio de todas nuestras páginas de la siguiente forma:

```
$titulo="Mi web";
hacer_encabezado($titulo);
```

De esta forma automatizamos el proceso de creación de nuestro documento.

Por supuesto, estas funciones ha de ser definidas dentro del script ya que no se encuentran integradas en PHP sino que las hemos creado nosotros. Esto en realidad no pone ninguna pega ya que pueden ser incluidas desde un archivo en el que iremos almacenando las definiciones de las funciones que vayamos creando o recopilando.

Estos archivos en los que se guardan las funciones se llaman librerías. La forma de incluirlos en nuestro script es a partir de la instrucción *require* o *include*:

```
require("libreria.php") o include("libreria.php")
```

Control del flujo en PHP

La programación exige en muchas ocasiones la repetición de acciones sucesivas o la elección de una determinada secuencia y no de otra dependiendo de las condiciones específicas de la ejecución.

- **Condiciones IF**

Cuando queremos que el programa, llegado a un cierto punto, tome un camino concreto en determinados casos y otro diferente si las condiciones de ejecución difieren, nos servimos del conjunto de instrucciones *if, else* y *elseif*. La estructura de base de este tipo de instrucciones es la siguiente:

```
if (condición)
{
  Instrucción 1;
  Instrucción 2;
  ...
}
else
{
  Instrucción A;
  Instrucción B;
  ...
}
```

Llegados a este punto, el programa verificará el cumplimiento o no de la condición. Si la condición es cierta las instrucciones 1 y 2 serán ejecutadas. De lo contrario (*else*), las instrucciones A y B serán llevadas a cabo.

Otras condiciones pueden plantearse dentro de la condición principal. Hablamos entonces de condiciones anidadas que tendrían una estructura del siguiente tipo:

```
if (condición1)
{
  Instrucción 1;
```

```
    Instrucción 2;
    ...
}
else
{
    if (condición2)
    {
        Instrucción A;
        Instrucción B;
        ...
    }
    else
    {
        Instrucción X
        ...
    }
```

De este modo podríamos introducir tantas condiciones como queramos dentro de una condición principal.

De gran ayuda es la instrucción *elseif* que permite en una sola línea introducir una condición adicional. Este tipo de instrucción simplifica ligeramente la sintaxis que acabamos de ver:

```
if (condición1)
{
    Instrucción 1;
    Instrucción 2;
    ...
}
elseif (condición2)
{
    Instrucción A;
    Instrucción B;
    ...
}
else
{
    Instrucción X
    ...
}
```

- **Bucles**

Sirven para realizar una determinada secuencia de instrucciones un cierto número de veces.

- **Bucle while.** Es el bucle más utilizado y el más sencillo. Se usa para ejecutar las instrucciones contenidas en su interior siempre y cuando la condición definida sea verdadera. La estructura sintáctica es la siguiente:

```
while (condición)
{
    instruccion1;
    instruccion2;
    ...
}
```

Un ejemplo sencillo es este bucle que aumenta el tamaño de la fuente en una unidad a cada nueva vuelta por el bucle:

```
<?
$size=1;
While ($size<=6)
{
  echo"<font size=$size>Tamaño
$size</font><br>\n";
  $size++;
}
?>
```

En primer lugar se define el valor de la variable que vamos a evaluar en la condición. En este caso le hemos atribuido un valor de 1 que corresponde a la letra más pequeña. A continuación se crea el bucle en el que imponemos la condición de que la variable no exceda el valor de 6. La instrucción a ejecutar es imprimir en nuestro documento un código HTML en el que la etiqueta *font* y el mensaje que contiene varían a medida que *$size* cambia su valor. Finalmente se incrementa en una unidad el valor de $size.

- **Bucle do/while**. La diferencia con respecto a los bucles *while* es que este tipo de bucle evalúa la condición al final con lo que, incluso siendo falsa desde el principio, éste se ejecuta al menos una vez. La sintaxis es la siguiente:

```
do
{
  instruccion1;
  instruccion2;
  ...
}
while (condición)
```

- **Bucle for**. La diferencia con los anteriores radica en cómo se plantea la condición de finalización del bucle. Para aclarar su funcionamiento veamos el ejemplo de bucle *while* en forma de bucle *for*:

```
<?
For ($size=1;$size<=6;$size++)
{
  echo"<font size=$size>Tamaño
$size</font><br>\n";
}
?>
```

Las expresiones dentro del paréntesis definen respectivamente: inicialización de la variable (valida para la primera vuelta del bucle), condición de evaluación a cada vuelta (si es cierta, el bucle continua) y acción a realizar al final de cada vuelta de bucle.

- **Bucle foreach**. Este bucle, implementado en las versiones de PHP4, nos ayuda a recorrer los valores de un array lo cual puede resultar muy útil por ejemplo para efectuar una lectura rápida del mismo.

Recordamos que un array es una variable que guarda un conjunto de elementos (valores) catalogados por clave. La estructura general es la siguiente:

```
Foreach ($array as $clave=>$valor)
{
  instruccion1;
  instruccion2;
  ...;
}
```

- **Operadores**

Las variables, como base de información de un lenguaje, pueden ser creadas, modificadas y comparadas con otras por medio de los llamados operadores.

- **Operadores aritméticos**. Nos permiten realizar operaciones numéricas con nuestras variables:

+	Suma
-	Resta
*****	Multiplicación
/	División
%	Devuelve el resto de la división

- **Operadores de comparación**. Se utilizan principalmente en las condiciones para comparar dos variables y verificar si cumple o no la propiedad del operador.

==	Igualdad
!=	Desigual
<	Menor que
<=	Menor igual que
>	Mayor que
>=	Mayor igual que

- **Operadores lógicos**. Se usan en combinación con los operadores de comparación cuando la expresión de la condición lo requiere.

And	Y
Or	O
!	No

- **Operadores de incremento**. Sirven para aumentar o disminuir de una unidad el valor de una variable.

++$variable	Aumenta de 1 el valor de $variable
--$variable	Reduce de uno el valor de $variable

- **Operadores combinados**. Una forma habitual de modificar el valor de las variables es mediante los operadores combinados:

$variable **+=** 10	Suma 10 a $variable
$variable **-=** 10	Resta 10 a $variable
$variable **.=** "añado"	Concatena las cadenas $variable y "añado"

Este tipo de expresiones no son más que abreviaciones de otras formas más clásicas: $variable+=10 es lo mismo que: $variable = $variable+10

Pasar variables por la URL

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

Las variables de un script tienen una validez exclusiva para dicho script y resulta imposible conservar su valor cuando ejecutamos otro archivo distinto aunque ambos estén enlazados. Para que la página destino reconozca las variables de otra página origen es necesario introducir dichas variables dentro del enlace hipertexto de la página destino. La sintaxis en la página origen sería la siguiente:

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

Como vemos estas variables no poseen el símbolo $ delante. Esto es debido a que en realidad este modo de pasar variables no es específico de PHP sino que es utilizado por otros lenguajes.

Procesar variables de formularios

Este tipo de transferencia es de gran utilidad ya que nos permite interaccionar directamente con el usuario.

El proceso es similar al explicado para las URLs. Primeramente, presentamos una primera página con el formulario clásico a rellenar y las variables son recogidas en una segunda página que las procesa:

Primera página:

```
<HTML>
<HEAD>
<TITLE>formulario.html</TITLE>
</HEAD>
<BODY>
```

```
<FORM METHOD="POST" ACTION="destino2.php">
Nombre<br>
<INPUT TYPE="TEXT" NAME="nombre"><br>
Apellidos<br>
<INPUT TYPE="TEXT" NAME="apellidos"><br>
<INPUT TYPE="SUBMIT">
</FORM>
</BODY>
</HTML>
```

Segunda página:

```
<HTML>
<HEAD>
<TITLE>destino2.php</TITLE>
</HEAD>
<BODY>
<?
echo "Variable \$nombre: $nombre <br> \n";
echo "Variable \$apellidos: $apellidos <br> \n"
?>
</BODY>
</HTML>
```

Utilización de las cookies

Las cookies son unas informaciones almacenadas por un sitio web en el disco duro del usuario. Esta información es almacenada en un archivo tipo texto que se guarda cuando el navegador accede al sitio web.

La utilidad principal de las cookies es la de poder identificar al navegador una vez éste visita el sitio por segunda vez y así, en función del perfil del cliente dado en su primera visita, el sitio puede adaptarse dinámicamente a sus preferencias (lengua utilizada, colores de pantalla, formularios rellenados total o parcialmente, redirección a determinadas páginas...).

Para crear un archivo cookies, modificar o generar una nueva cookie lo podemos hacer a partir de la función SetCookie:

```
setcookie("nombre_de_la_cookie",valor,expiracion);
```

Es importante que la creación de la cookie sea previa a la apertura del documento HTML. En otras palabras, las llamadas a la función setcookie() deben ser colocadas antes de la etiqueta HTML.

Por otra parte, es interesante señalar que el hecho de que definir una cookie ya existente implica el borrado de la antigua. Del mismo modo, el crear una primera cookie conlleva la generación automática del archivo texto.

Para utilizar el valor de la cookie en nuestros scripts tan sólo tendremos que llamar la variable que define la cookie. Hay que tener cuidado sin embargo de no definir variables en nuestro script con el mismo nombre que las cookies puesto que PHP privilegiará el contenido de la variable local con respecto a la cookie y no dará un mensaje de error.

Sesiones

Hasta ahora hemos utilizado variables que sólo existían en el archivo que era ejecutado. Cuando cargábamos otra página distinta, los valores de estas variables se perdían a menos que nos tomásemos la molestia de pasarlos por la URL o inscribirlos en las cookies o en un formulario para su posterior explotación. Estos métodos, aunque útiles, no son todo lo prácticos que podrían en determinados casos en los que la variable que queremos conservar ha de ser utilizada en varios scripts diferentes y distantes los unos de los otros.

Podríamos pensar que ese problema puede quedar resuelto con las cookies ya que se trata de variables que pueden ser invocadas en cualquier momento. El problema es que las cookies no son aceptadas ni por la totalidad de los usuarios ni por la totalidad de los navegadores lo cual implica que una aplicación que se sirviera de las cookies para pasar variables de un archivo a otro no sería 100% infalible. Es importante a veces pensar en "la inmensa minoría", sobre todo en aplicaciones de comercio electrónico donde debemos captar la mayor cantidad de clientes posibles y nuestros scripts deben estar preparados ante cualquier eventual deficiencia del navegador del cliente. Nos resulta pues necesario el poder declarar ciertas variables que puedan ser reutilizadas tantas veces como queramos dentro de una misma sesión.

Pensemos en un carrito de la compra de una tienda virtual donde el cliente va navegando por las páginas del sitio y añadiendo los artículos que quiere comprar a un carrito. Este carrito podría ser perfectamente una variable de tipo array (tabla) que almacena para cada referencia la cantidad de artículos contenidos en el carrito. Esta variable debería ser obviamente conservada continuamente a lo largo de todos los scripts.

Este tipo de situaciones son solventadas a partir de las variables de sesión. Una sesión es considerada como el intervalo de tiempo empleado por un usuario en recorrer nuestras páginas hasta que abandona nuestro sitio o deja de actuar sobre él durante un tiempo prolongado o bien, sencillamente, cierra el navegador.

PHP nos permite almacenar variables llamadas de sesión que, una vez definidas, podrán ser utilizadas durante este lapso de tiempo por cualquiera de los scripts de nuestro sitio. Estas variables serán especificas del usuario de modo que varias variables sesión del mismo tipo, con distintos valores, pueden estar coexistiendo para cada una de las sesiones que están teniendo lugar simultáneamente. Estas sesiones tienen además su propio identificador de sesión que será único y específico.

Las variables de sesión se diferencian de las variables clásicas en que éstas residen en el servidor, son especificas de un solo usuario definido por un identificador y pueden ser utilizadas en la globalidad de nuestras páginas.

Para iniciar una sesión podemos hacerlo de dos formas distintas:

- Declaramos abiertamente la apertura de sesión por medio de la función *session_start()*. Esta función crea una nueva sesión para un nuevo visitante o bien recupera la que está siendo llevada a cabo.
- Declaramos una variable de sesión por medio de la función *session_register('variable')*. Esta función, además de crear o recuperar la sesión para la página en la que se incluye también sirve para introducir una nueva variable de tipo sesión.

Las sesiones han de ser iniciadas al principio de nuestro script. Antes de abrir cualquier etiqueta o de imprimir cualquier cosa. En caso contrario recibiremos un error.

Un ejemplo de utilización de una sesión es un contador que deberá aumentar en una unidad cada vez que recargamos la página o pulsemos el enlace:

```
<?
session_register('contador');
?>
<HTML>
<HEAD>
<TITLE>contador.php</TITLE>
</HEAD>
<BODY>
<?
If (isset($contador)==0)
{ $contador=0;}
++$contador;
echo "<a href=\"contador.php\">Has recargado esta página
$contador veces</a>";
?>
</BODY>
</HTML>
```

La condición *if* tiene en cuenta la posibilidad de que la variable *$contador* no haya sido todavía inicializada. La función *isset* se encarga de dar un valor cero cuando una variable no ha sido inicializada

TRABAJAR CON BASES DE DATOS EN PHP

Una de las principales ventajas que presenta el trabajar con páginas dinámicas es el poder almacenar los contenidos en bases de datos. De esta forma, podemos organizarlos, actualizarlos y buscarlos de una manera mucho más simple.

El lenguaje PHP, ya hemos dicho, ofrece interfaces para el acceso a la mayoría de las bases de datos comerciales y por ODBC a todas las bases de datos posibles en sistemas Microsoft, a partir de las cuales podremos editar el contenido de nuestro sitio con absoluta sencillez.

Esta interacción se realiza, por un lado, a partir de las funciones que PHP nos propone para cada tipo de base de datos y, por otro estableciendo un diálogo a partir de un idioma universal: SQL (Structured Query Language) el cual es común a todas las bases de datos.

La base de datos más extendida en combinación con PHP es MySQL debido sobre todo a su gratuidad, eficiencia y simplicidad.

Una vez instalado MySQL será necesario llevar a cabo las siguientes operaciones:

- Introducidos dentro de MySQL, crearemos la base de datos ejemplo con la siguiente sentencia: `create database ejemplo;`

- Seleccionaremos la base ejemplo como la base a utilizar: `use ejemplo;`

- Crearemos a continuación la tabla clientes a partir de la siguiente sentencia:

```
create table clientes (
nombre varchar(100),
telefono varchar(100)
);
```

Ahora ya disponemos de una tabla vacía a la que podremos hacerle las siguientes operaciones:

- **Introducción de nuevos registros.**

  Una vez creada la tabla *clientes* en nuestra base de datos *ejemplo*, el paso siguiente sea llenarla con registros. Los datos del registro pueden ser recogidos, por ejemplo, a partir de un formulario.
  Lo primero que habrá que hacer es establecer un vínculo entre el programa y la base de datos. Esta conexión se lleva a cabo con la función *mysql_connect*. A continuación, deberemos generar una orden de inserción del registro en lenguaje SQL. Esta orden será ejecutada por medio de la función *mysql_db_query*. En esta función especificaremos primeramente la base de datos sobre la que queremos actuar y a continuación introduciremos la sentencia SQL.

```
<?
//Conexion con la base
mysql_connect("localhost","tu_user","tu_password");
//Ejecucion de la sentencia SQL
mysql_db_query("ejemplo","insert into clientes
(nombre,telefono) values (' $nombre' ,' $telefono' )");
?>
```

  Los parametros user y password son definidos por el creador de la base.

- **Selección y lectura de registros**.

  Dentro de una base de datos, organizada por tablas, la selección de una tabla entera o de un cierto numero de registros resulta una operación rutinaria.
  Los pasos a realizar son, en un principio, los vistos para la inserción de un registro: Conexión a la base y ejecución de la sentencia. Supongamos que la información de dicha ejecución es almacenada en la variable (*$result*).
  El siguiente paso será plasmar en pantalla la información recogida en $result. Esto lo haremos mediante la función *mysql_fetch_array* que devuelve una variable array con los contenidos de un registro a la vez que se posiciona sobre el siguiente.
  La función *mysql_free_result* se encarga de liberar la memoria utilizada para llevar a cabo la consulta.

- **Actualización de un registro.**

  Supongamos que queremos cambiar el numero de teléfono de las distintas personas presentes en nuestra base. El nombre de estas personas, así como el nuevo numero de teléfono, serán recogidos por medio de un formulario.
  El archivo del formulario puede ser un script PHP en el que efectuaremos una llamada a nuestra base de datos para construir un menú desplegable donde aparezcan todos los nombres.

La manera de operar para construir el menú desplegable es la misma que para visualizar la tabla, por ejemplo mediante un bucle *while* en combinación con la función *mysql_fetch_array*, y el script de actualización será equivalente al de inserción.

- **Borrado de un registro con PHP**

  Otra de las operaciones elementales que se pueden realizar sobre una base de datos es borrar un registro. Para hacerlo, SQL nos propone sentencias del tipo *Delete*. Una opción sería crear un menú desplegable dinámico, como para el caso de las actualizaciones, y a continuación hacer efectiva la operación a partir de la ejecución de la sentencia SQL que construimos a partir de los datos un formulario.