PHP

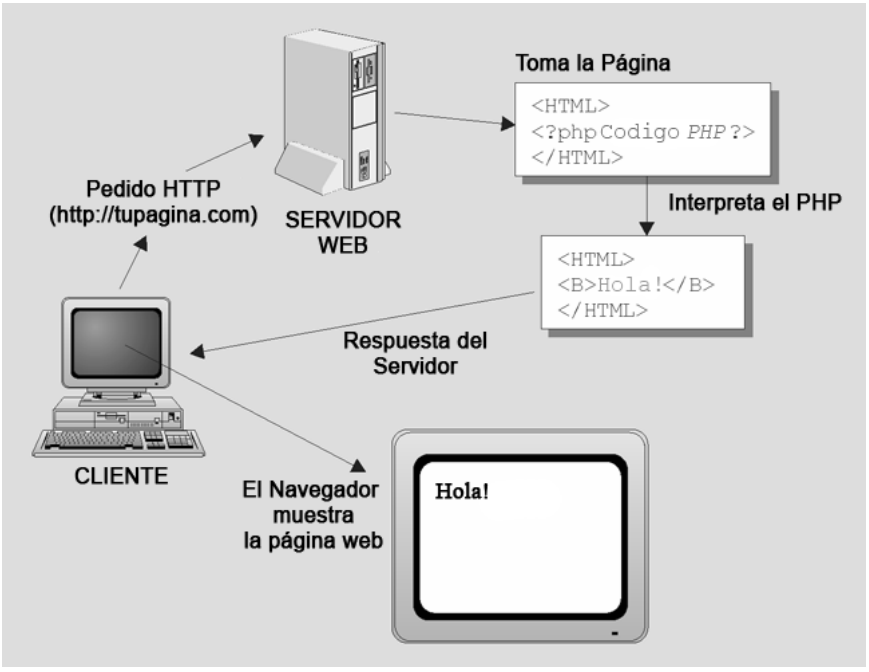
Conceptos Básicos.

¿Qué es PHP?

Las siglas **"PHP"** significan **"PHP: Hypertext Processor"** o también **"Pre-Hypertext Processor"** (Procesador Previo al Hipertexto). Es un lenguaje de programación que sirve para hacer páginas de Internet dinámicas, con la posibilidad de combinarlo con bases de datos. Los *script* de PHP suelen intercalarse en el código HTML de una página, y el encargado de procesar esos *scripts* es el intérprete del servidor web.
El fin de un *script* PHP es generar una salida (que puede ser texto o código HTML, entre otras cosas), que puede ser diferente según el caso, ya que la gracia de incluir scripts PHP en una página es obtener resultados dinámicos.

Funcionamiento.

PHP es un **lenguaje de lado de servidor**. ¿Qué significa esto? Que lo que el programador codifique en PHP dentro de una página de Internet, no va a ser interpretado por el cliente (navegador), si no por un servidor que procese el código y devuelva al cliente el código HTML resultante.
Cuando nosotros le pedimos al navegador que nos lleve a una página en Internet, el navegador le pide al servidor el código de esta página, lo interpreta y lo muestra en pantalla. Si en el código de la página a la que queremos acceder hay *scripts* PHP (y por supuesto, el documento tiene **extensión .php**), antes de que el servidor le entregue al cliente el código de la página, estos *scripts* son interpretados por el servidor, y en vez de devolverle código PHP al cliente, le devuelve el código HTML resultante de procesar esos *scripts* PHP.



Instalación

Para poder ver los resultados de los *scripts* PHP, es necesario contar con un servidor que pueda interpretarlos. Un servidor que permite esto es el **Apache**. Cuando uno quiere hacer un sitio web con PHP, al momento de ponerlo online debe asegurarse que el hosting a contratar cuenta con soporte para PHP y corre el servidor Apache.
Si se van a usar bases de datos (generalmente **MySQL**) también hay que asegurarse de que el hosting lo soporte.
Para trabajar localmente sin depender de estar subiendo constantemente los archivos al hosting para poder verlos en funcionamiento, podemos instalar el servidor Apache y MySQL en nuestra computadora. Un programa que facilita esta instalación es el **Easy-PHP 1.8**.
Para poder visualizar los documentos php, estos deberán estar guardados en la carpeta **www** de la carpeta de instalación del Easy-PHP, y para verlos en el explorador habrá que acceder a ellos mediante la url **http://localhost/nombredearchivo.php** (teniendo el programa Easy-PHP abierto y corriendo).

Alcances

Usando PHP en un sitio web, se puede lograr un mayor dinamismo del contenido y, sobre todo combinándolo con la gestión de bases de datos, lograr hacer funcionar algunos de los siguientes ejemplos:

- Foros
- Guestbooks
- Blogs
- Calendarios
- Sistemas de Carrito de Compras
- Restricción de acceso a sitios web mediante user/password
- Registro de estadísticas
- Proceso de Formularios de Email

Intercalando PHP con HTML

Un documento PHP puede tener íntegramente código PHP así como intercalaciones de código PHP y HTML.
Muchas veces, el contenido de una página no es fijo. Ejemplos de esto hay muchísimos, desde algo trivial como mostrar la hora actual en el mensaje de bienvenida de un sitio web, hasta mostrar una lista de productos que un visitante del sitio haya ido agregando a un carrito de compras, los cuales se van listando a medida que el visitante los selecciona. En estos casos, el HTML “nos queda corto” y necesitamos usar scripts PHP para generar o administrar un

contenido generado dinámicamente, el cual seguramente será sólo una porción de la página y no toda entera. Entonces, el contenido **estático** de la página estará *codificado* en HTML (diseño, estructura, textos fijos, etc), y el contenido **dinámico** estará *programado* en PHP.

Para comenzar a introducirnos en la sintaxis de PHP y en cómo intercalar PHP con HTML, se hará uso de ejemplos que no reflejan el "dinamismo" que se puede obtener con PHP, pero ayudarán a dar el primer paso a tomar una idea de cómo se puede llegar a eso.

Tenemos el siguiente documento holamundo.php, con HTML y PHP intercalados:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Hola mundo</title>
</head>

<body>

<?php
echo "Hola mundo";
?>

</body>
</html>
```

Dentro del body de esa página, nos encontramos con la sentencia **echo "Hola mundo";** que muestra por pantalla la frase "Hola mundo". Sería exactamente lo mismo que figure esa sentencia PHP allí a que sólo figuren las palabras "Hola mundo", que sería un texto común de HTML.
Los elementos que encontramos en este *script* son, por un lado, los **delimitadores** de PHP y por otro la **sentencia**. Los delimitadores son **<?php** y **?>**, y sirven para que cuando el servidor reciba la petición de holamundo.php, sepa cuándo tiene que procesar código PHP y cuándo debe dejar de hacerlo. Al encontrarse con el delimitador de apertura **<?php**, el servidor sabe que allí tendrá que empezar a interpretar el código PHP para "convertirlo" en su salida correspondiente en HTML, y al encontrarse con el delimitador de cierre **?>**, sabe que debe dejar de interpretar PHP. De esta manera, usando los delimitadores de código PHP, podemos intercalar libremente *scripts* PHP con código HTML.

Por otro lado, tenemos la sentencia PHP que el servidor interpretará. **echo** es una instrucción del lenguaje PHP que toma como argumento un string (cadena de caracteres) que aparezca a su derecha, y lo muestra por pantalla. En realidad, a ese argumento que recibe lo imprime en el código HTML, en el lugar donde figura el código PHP que lo ejecuta. Si nosotros ejecutamos la página holamundo.php en un navegador y después miramos su código fuente desde el mismo, el resultado será este:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Hola mundo</title>
</head>

<body>

Hola mundo
</body>
</html>
```

Se verá que el servidor interpretó el *script* PHP y en su lugar "imprimió" su salida correspondiente, que en este caso era simplemente las palabras "Hola mundo".
Volviendo a la sentencia **echo "Hola mundo";** hay un par de cosas más para notar. La frase "Hola mundo" está entre **dobles comillas** por ser un **string**. También puede ir entre comillas simples, aunque guardan diferencias que se verán más adelante. El otro punto muy importante es el **punto y coma** (**;**). Todas las sentencias de PHP terminan con un punto y coma. Es la manera que tiene el intérprete de saber que el programador allí quiso terminar una sentencia. Mientras terminemos todas las sentencias con un punto y coma, no es siquiera necesario que estén separadas por renglones, podrían estar una detrás de la otra en el mismo renglón, y por otro lado, tampoco es necesario que una misma sentencia la expresemos en un solo renglón, podría estar repartida en varios que hasta que no haya un punto y coma en el código, el intérprete de PHP sabe que todavía no se terminó la sentencia. NUNCA debemos olvidar poner un punto y coma al final de cada sentencia.

Ejemplos:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
```

```
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Hola mundo</title>
</head>

<body>

<?php
echo "Este es

el

clásico "; echo "Hola mundo";
?>

</body>
</html>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Hola mundo</title>
</head>

<body>

<?php
echo "Este es el clásico "; echo "Hola mundo";
?>

</body>
</html>
```

Ambos códigos mostrarán por pantalla el mismo mensaje:
“Este es el clásico Hola Mundo”

Así como hasta ahora se ha usado texto con la instrucción echo, se puede pensar que si ese texto queda impreso en el código fuente de la página como HTML cuando ya fue procesado

por el intérprete PHP del servidor y devuelto al navegador, también podríamos imprimir directamente código HTML.

Por ejemplo, en vez del *script* anteriormente usado, podríamos reemplazarlo por algo como:

```
<?
echo "<p>Hola mundo</p>";
?>
```

y si ejecutamos holamundo.php en el navegador y vemos su código fuente, nos encontraremos con:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Hola mundo</title>
</head>

<body>

<p>Hola mundo</p>
</body>
</html>
```

Sintaxis de PHP

Ya se adelantaron en la sección anterior un par de temas de sintaxis de PHP. Por un lado, las sentencias siempre finalizan con un punto y coma. Por otro, el código PHP siempre debe estar encerrado por sus delimitadores, que pueden ser **<?php** y **?>**, como también **<?** y **?>**.
Otras cuestiones sobre la sintaxis son los comentarios. Los **comentarios de una sola** línea en PHP pueden hacerse empezando la línea con **//** o con **#.** Los **comentarios de múltiples líneas** están encerrados entre **/*** y ***/** . Por supuesto, sólo podrán usarse dentro de las delimitaciones de un script PHP.
Por otro lado, tenemos la sintaxis de las variables, que cuenta con las siguientes características:

- Las variables **SIEMPRE llevan el signo $ (pesos) adelante**.
- Los únicos caracteres que aceptan los nombres de las variables son **letras, números y el guión bajo** (_).
- Los nombres de las variables NUNCA pueden llevar espacios ni empezar con un número.
- Los nombres de las variables pueden empezar con letras o con un guión bajo.
- Los nombres de las variables **son sensibles a mayúsculas y minúsculas**.

Veamos algunos ejemplos:

```
<?php

$Cadena = "Hola Mundo"; //Esto es correcto
$cadena = "Hola Mundo"; /*      Esto también es correcto pero
                                        NO es lo mismo que la línea anterior.
                                        $Cadena y $cadena son dos variables
                                        distintas */

$_Cadena = "Hola Mundo"; //Esto es correcto

$1Cadena = "Hola Mundo"; //Esto NO es correcto
$Una Cadena = "Hola Mundo"; //Esto TAMPOCO es correcto.

?>
```

Una observación interesante sobre la sintaxis de PHP, es que sólo es sensible a mayúsculas y minúsculas cuando se trata de nombres de variables y no cuando se trata de funciones o instrucciones del lenguaje. Esto quiere decir, que si en los scripts anteriores escribíamos *ECHO* en vez de *echo*, el script seguiría funcionando.

Variables

Una variable en PHP es un espacio de memoria (en el caso de un lenguaje de lado de servidor, un espacio en la memoria del servidor) que guarda un valor, y tiene un identificador (el nombre de la variable) y un tipo de dato (enteros, reales, cadenas, arrays, objetos). No es necesario indicar de qué tipo de dato es una variable, el intérprete de PHP se da cuenta solo según su contenido, y en muchos casos también "convierte" el tipo de dato de una variable a otro si es conveniente según el contexto. ¿Para qué usamos variables? Para guardar información en ellas que luego usaremos varias veces a lo largo del código de la página. Veamos algunos ejemplos de uso de variables:

```
<?php
/* Caso 1 */
$Cadena = "Hola Mundo"; //Asignación de una variable tipo string

/* Caso 2 */
echo $Cadena."<br/>"; //Muestra por pantalla el valor de $Cadena

/* Caso 3 */
$NumeroUno = 1; //Asignación de una variable de tipo int.

/* Caso 4 */
$NumeroDos = "2"; //Asignación de una variable tipo string.

/* Caso 5 */
echo $NumeroUno + $NumeroDos; //Muestra por pantalla '3'.

/* Caso 6 */
echo $NumeroDos + $Cadena; //Muestra por pantalla '2', no realiza la suma.

?>
```

En el Caso 1 vemos un ejemplo simple de asignación de una variable con el operador de asignación =.
En el Caso 2, vemos un ejemplo de uso de la instrucción echo con una variable en vez de texto y otra cosa a destacar: **concatenación de variables.** La concatenación de variables sirve para unir el valor de una variable con el valor de otra mediante el operador **”.”** y que el resultado de sus uniones sea una cadena formada por sus valores. En el Caso 2, echo estaría mostrando “Hola Mundo
”, provocando que después de mostrar “Hola Mundo” por pantalla, haya un salto de línea.
En el Caso 3 se muestra un ejemplo de asignación de un número entero a una variable.
El Caso 5 y el Caso 6 muestran que el intérprete de PHP se da cuenta de cuándo puede operar entre números y strings y cuándo no.

Observaciones sobre Cadenas

Hay algunos puntos a tener en cuenta en el tratamiento de cadenas.
Supongamos que tenemos este script que muestra por pantalla “Mi nombre es Juan”:

```
<?php
```

```
$Cadena = "Juan";
echo "Mi nombre es ".$Cadena;
?>
```

También puede escribirse de la siguiente manera sin tener que concatenar variables:

```
<?php
$Cadena = "Juan";
echo "Mi nombre es $Cadena";
?>
```

Pero esto nos trae algunos inconvenientes. ¿Qué pasa si quiero que en una cadena aparezca el símbolo $, o la comilla doble, por ejemplo?
Cuando queremos que en una cadena aparezca un caracter que ser evaluado por el intérprete y reemplazado por otro valor, le anteponemos una **contrabarra** al caracter o cuestión: \. De este modo, el interprete PHP se da cuenta que *sólo el caracter inmediatamente siguiente* a la contrabarra debe ser tomado como texto.

Ejemplos:

```
<?php
$Cadena = "Juan";
echo "Mi nombre es ".$Cadena; //Muestra por pantalla 'Mi nombre es Juan'

echo "Mi nombre es \$Cadena"; //Muestra por pantalla 'Mi nombre es $Cadena'

echo "Mi nombre es \"$Cadena\""; //Muestra por pantalla 'Mi nombre es "Juan"'

?>
```

También hay algunos caracteres especiales que producen efectos en las cadenas:

- \t produce una tabulación.
- \n produce un salto de línea.
- \r produce un retorno de carro.

Operando con Variables

Operadores aritméticos

+ Suma

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

Operadores de comparación

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

Operadores lógicos

And	Y
Or	O
!	No

Operadores de incremento

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

Operadores combinados

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

Algunos ejemplos:

```
<?php
$Var1=1;
$Var2=2;
$Var3=3;
$VarUno="1";

echo "2x3= ".($Var2*$Var3)."<br>"; //Muestra '2x3= 6'
echo "2+1 es ".($Var2+$VarUno)."<br>"; //Muestra '2+1 es 3'
echo "¿3 es igual a 2? ".($Var3==$Var2)."<br>"; //No muestra nada como respuesta
echo "1 es igual a \"1\"? ".($Var1==$VarUno)."<br>"; //Muestra '1' como respuesta
```

```
?>
```

Cabe destacar que en los últimos dos casos de estos ejemplos, en el primero $Var3==$Var2 no devuelve un resultado porque es falso (en realidad devuelve 0). En el último caso, devuelve 1 porque es verdadero.

Variables de tipo Array

Array indexado

Las variables de tipo Array son como variables con muchos "compartimientos" que pueden almacenar varios valores, a los que se puede acceder mediante un índice.
En la sintaxis de Arrays en PHP, el índice (o clave) se indica entre corchetes.
Hay dos maneras de definir un array.

a) Dándole un valor a cada posición accediendo invididualmente por su índice:

```
<?
$familia[0]="Padre";
$familia[1]="Madre";
$familia[2]="Hijo";

echo $familia[0]; //Muestra 'Padre'
echo $familia[1]; //Muestra 'Madre'
?>
```

b) Definiendo los valores iniciales separados por comas dentro de array( ):

```
$familia = array("Padre","Madre","Hijo");

echo $familia[0]; //Muestra 'Padre'
echo $familia[1]; //Muestra 'Madre'
?>
```

Los índices de los array siempre empiezan desde 0, así que hay que tener en cuenta que si se quiere acceder al segundo elemento contenido en un array, habrá que acceder a él mediante el índice 1, por ejemplo.

Array asociativo

La diferencia de los Arrays asociativos con los indexados, es que en vez de acceder al contenido de un "compartimiento" del array por el índice de su posición, podemos definir las CLAVES con cadenas que identifiquen mejor el contenido de cada uno de esos compartimientos.
Al tener cadenas en vez de un número como clave, es mucho más fácil identificar el valor de cada elemento del array:

```
<?
$precios["TV"]=700;
$precios["Telefono"]=150;
$precios["Computadora"]=1900;
echo $precios["Computadora"]; //Muestra '1900'
?>
```

Otra manera de definir un array asociativo es:

```
<?
$precios=array("TV"=>1500, "Telefono"=>150, "Computadora"=> 1900);
echo $precios["Computadora"]; //Muestra '1900'
?>
```

Observación: una función muy práctica para usar con arrays es count, que devuelve la cantidad de elementos de un array.

```
<?
$un_array=array(4,6,3,6,7,23,1);
echo count($un_array); //Muestra '7'
?>
```

Estructuras de Control

Condicionales:

Son estructuras que evaluando una condición dada, "deciden" si ejecutar cierto bloque de código u otro.

if:

La sintaxis es:

```
if (condición) {
//Código a ejecutar si la condición es verdadera
}
else {
//Otro código a ejecutar en caso contrario
}
```

Ejemplo:

```
<?php
$x=4;
if ($x>5) {
echo "El valor de x es mayor a 5";
}
else {
echo "El valor de x es menor o igual a 5";
}
?>
```

Salida:

```
El valor de x es menor o igual a 5
```

switch:

Un switch es una estructura que evalúa el valor de una variable o expresión y depende de su valor, ejecuta cierto bloque de código.

Sintaxis:

```
switch(expresión) {
  case "a": //Código a ejecutar;
                        break;
  case "b": //Código a ejecutar;
                        break;
  case "c": //Código a ejecutar;
                        break;
```

```
        .
        .
        .
 default: //Código a ejecutar por default.
}
```

Un switch busca dentro de los “case” el valor de la variable o expresión evaluada (generalmente se evalúan variables). Si lo encuentra, ejecuta el código correspondiente. Si no lo encuentra, ejecuta el código por default.

Observaciones: El case por default es opcional.
Es importante poner “break; “ al final de cada bloque de código dentro de cada case para que el switch no siga comparando al valor de la variable con los case que le siguen al correcto.

Ejemplo:

```
<?php
$examen="Bueno";
switch($examen) {
case "Excelente": echo "Su nota fue 10. ";
                                  break;
case "Muy bien": echo "Su nota fue 8 o 9. ";
                                  break;
case "Bueno": echo "Su nota fue 6 o 7";
                              break;
case "Regular": echo "Su nota fue 4 o 5";
                                 break;
default: echo "Su nota fue menor o igual a 3. ";
}
?>
```

Salida:

```
Su nota fue 6 o 7
```

Ciclos (o loops):

Hay 3 ciclos fundamentales:

While

El while ejecutará cierto bloque de código mientras una condición dada a evaluar resulte siempre verdadera.
Primero evalúa si es verdadera la expresión dada como condición, luego ejecuta el código, y cuando termina de ejecutar la última línea del mismo vuelve a evaluar la condición. Si sigue siendo verdadera, vuelve a ejecutar el código, si no, deja de ejecutarlo, sale del ciclo y sigue con el código que siga debajo.

Sintaxis:

```
while (expresión) {
//Código a ejecutar
}
```

Observación: Hay que tener cuidado de no caer en un ciclo infinito, ya que provoca problemas en la computadora (y además no tendría sentido). Para eso, tras cada vuelta del while hay que asegurarse que en la condición no se esté evaluando lo mismo que la vuelta anterior, cambiando algo de ella a través del código del mismo while.

do... while:
Ejecuta una vez cierto bloque de código por primera vez, y luego actúa igual que el while.
Sintaxis:

```
do {
//Código a ejecutar
} while (condición);
```

For:
El for es muy similar al while, pero se asegura de que los valores evaluados en la condición siempre cambien, y se usa cuando se quiere hacer un ciclo de determinado número fijo de veces.

Sintaxis:

```
for (inicio; condición; incremento) {
//Código a ejecutar
```

```
}
```

Ejemplo:

```
<?php
$amigos=array(“Laura”,”Pablo”,”Daniela”);
for ($i=0; $i<count($amigos); $i++) {
Echo $amigos[$i]. "<br />";
}
?>
```

Salida en pantalla:

```
Laura
Pablo
Daniela
```

En este ejemplo, usamos este for para recorrer todos los valores de un array y mostrarlos por pantalla.