

PHP

PARTE I:

Programación: instrucciones básicas

Parte I: Programación

- Sintaxis
- Operadores y variables
- Estructuras de control:
 - De condición
 - De iteración

INTRODUCCIÓN I

php (Hypertext preprocessor) es un lenguaje interpretado del lado del servidor y que se incrusta en las páginas HTML para lograr que éstas tengan contenido dinámico



XAMPP (*X* , *Apache*, *MySQL*, *PHP*, *Perl*.)

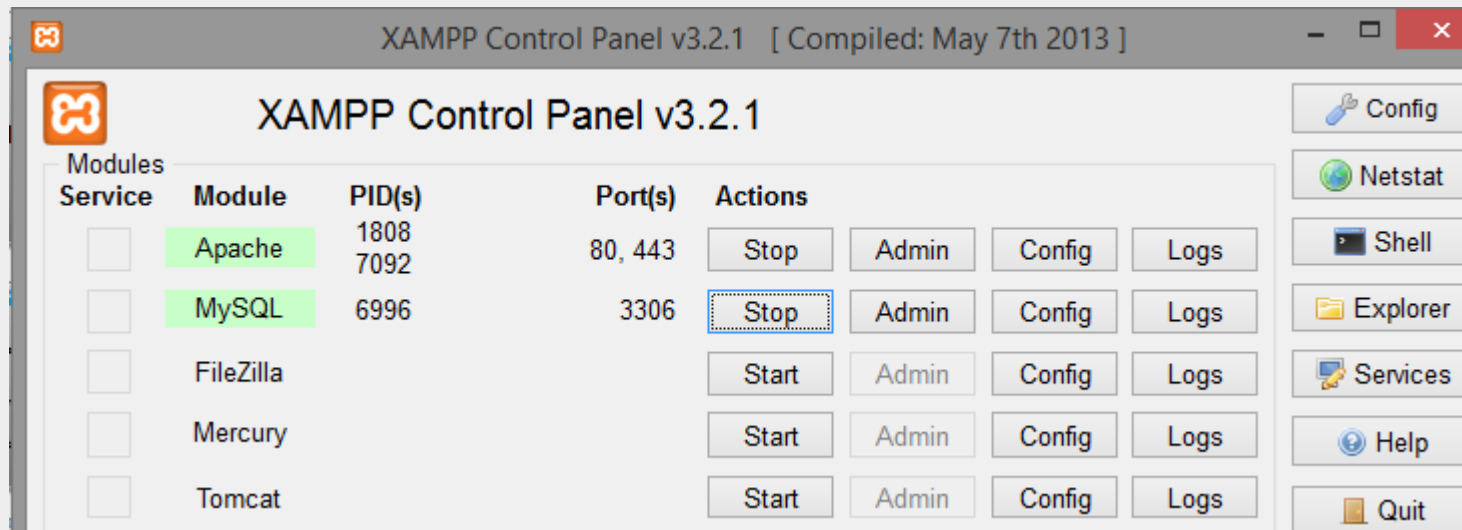
Para ejecutar php necesitaremos tener instalado un software de servidor

XAMPP es una plataforma independiente que contiene un servidor Apache en local, un sistema de gestión de base de datos MySQL y los intérpretes para los lenguajes PHP y Perl.

Lo podemos descargar en:

<https://www.apachefriends.org/es/index.html>

Panel de control de XAMPP



Una vez instalado tendremos que arrancar el servidor Apache y el gestor de base de datos MySQL

ERROR XAMPP I

Si aparece el mensaje “*Apache shutdown unexpectedly*”

Para solucionar este problema debemos cambiar los puertos que usan por defecto tanto Apache y como MySQL, editando dos archivos de configuración, que son el “httpd.conf”, ubicado en la siguiente ruta:

En Windows: “[C:\Xampp\Apache\Conf\httpd.conf](#)”

En Linux: “[opt/lampp/etc/httpd.conf](#)”

Ahora realizaremos el siguiente cambio.

ServerName localhost:80

ServerName localhost:8080

Listen 80

Listen 8080

ERROR XAMPP (cont)

Ahora vamos al archivo “httpd-ssl.conf” que se encuentra en la ruta:

En Windows: “C:\Xampp\Apache\Conf\Extra\httpd-ssl.conf”

En Linux: “opt/lampp/etc/extra/httpd-ssl.conf”

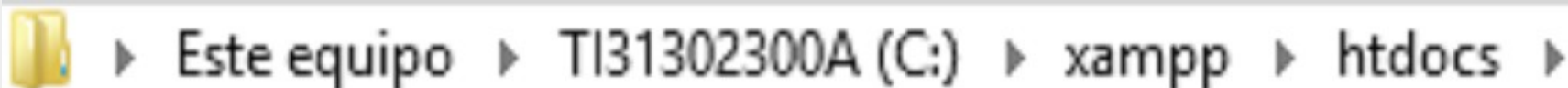
Una vez dentro del archivo, editamos las siguientes líneas:

```
<VirtualHost _default_:443>  
<VirtualHost _default_:4430>  
ServerName localhost:443  
ServerName localhost:4430  
Listen 443  
Listen 4430
```

Ahora reiniciamos el servidor Apache y ya debería funcionar, independientemente del programa que utilicemos y que use el puerto 80.

Nota: Recordad que acabamos de cambiar los puertos del servidor apache, así que también ha cambiado la forma en la que llamamos a nuestro servidor local. Para que funcione basta con escribir en el navegador “localhost:8080”.

DONDE GUARDAR LOS PROYECTOS



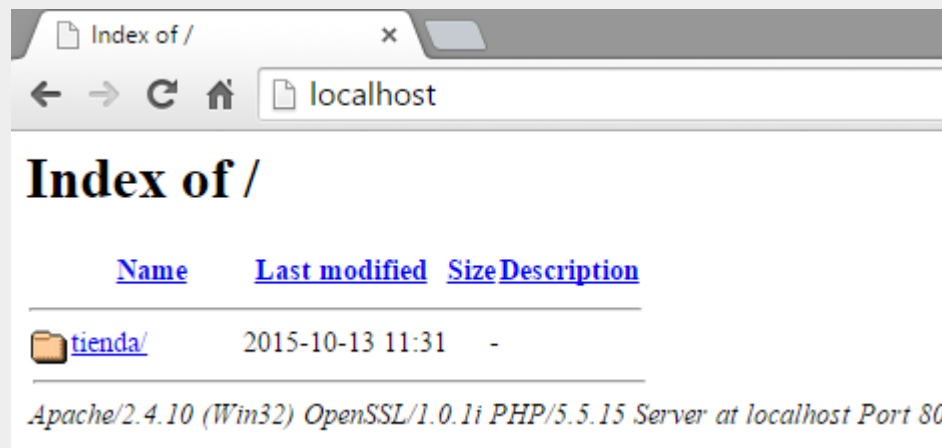
Este equipo > TI31302300A (C:) > xampp > htdocs >

Dentro de la carpeta xampp → htdocs crearemos las carpetas de nuestro proyecto.

Ej: si tenemos una página llamada 'tienda' podríamos crear una carpeta 'tienda' dentro de 'htdocs'

COMO VISUALIZAR LOS PROYECTOS

- Desde el navegador solo hay que teclear como URL: localhost y nos aparecerán todas las carpetas creadas:



COMO LANZAR PROYECTOS

- Dentro de la carpeta 'tienda' guardaríamos todos los archivos de nuestra página web: archivos .html, .php, .js, .css, etc
- Si uno de los archivos es el index.html (o index.php) será el primero que se lance al abrir la carpeta desde localhost, sin necesidad de ponerlo de forma expresa en la url
- En el ejemplo de abajo tenemos un archivo index.php con el siguiente contenido:

```
<?php phpinfo(); ?>
```

ECHO Y HOLA MUNDO

La instrucción **echo** devuelve el valor o cadena que se especifica detrás de ella.

Y vemos el esperado Hola Mundo en php

```
<?php  
    echo "Hola mundo";  
?>
```

Las instrucciones php siempre irán encerradas con las etiquetas **<?php ... ?>** y pueden situarse en cualquier lugar del documento.

COMENTARIOS

Es importante comentar el código de la siguiente manera:

```
<?php
```

```
    //comentario de una línea
```

```
    /*comentario de más de  
    una línea */
```

```
?>
```

NOTA: En php es obligatorio finalizar todas las instrucciones con punto y coma

Podemos mezclar html y php

```
<!DOCTYPE html>
```

```
<html>
```

```
  <head>
```

```
  </head>
```

```
  <body>
```

```
    <p><?php echo "<b>Texto en negrita</b>"; ?></p>
```

```
  </body>
```

```
</html>
```

Y asociar contenido dinámico con CSS

```
<html>
  <head>
    <style type="text/css">
      div {background-color: red;width: 300px;height: 300px}
    </style>
  </head>
  <body>
    <?php echo "<div></div>"; ?>
  </body>
</html>
```



VARIABLES I

Las variables son zonas de memoria a las que asignamos un nombre concreto (normalmente de acuerdo a la naturaleza de los datos a guardar en ellas) y que nos servirán para almacenar información que puede variar durante la ejecución del programa.

Se suelen utilizar las minúsculas o el signo _

Nunca acentos, espacios en blanco ni caracteres especiales

Se pueden usar números pero nunca al principio

TODAS LAS VARIABLES EN PHP COMIENZAN POR \$

VARIABLES II

Las variables se declaran únicamente nombrándolas.

Ejemplo:

```
<?php
    $variable_1 = 'soy un texto' //variable de texto
    $variable_2 = 34 //variable numérica
?>
```


EJERCICIO

Que se muestre un mensaje que diga:

“Buenas tardes *nombre*”, dónde *nombre* será el valor de una variable llamada **\$nombre**

\$nombre=...
echo “Buenas..

NOTA: Para concatenar en php se utiliza el punto .

VARIABLES III

De texto:

```
<?php
    $texto="soy una cadena de texto";
    echo $texto; //se muestra el contenido de la variable
    echo "<br/>";
    echo '$texto'; //se muestra la palabra texto
    echo "<br/>";
    echo "$texto"; //se muestra el contenido de la variable
?>
```

NOTA 1: Podemos ver como las etiquetas **HTML** se interpretan como tales.

NOTA 2: php evalúa el contenido que se encuentre entre comillas dobles y muestra el valor de las variables.

VARIABLES IV

Existen algunos caracteres especiales que requieren una notación distinta:

'\n' – nueva línea.

'\r' – retorno de carro.

'\t' – tab.

'\"' – comillas simples.

'\\' – barra invertida.

'\"' – comillas dobles.

VARIABLES V

Numéricas:

```
<?php
    $i = 5;
    $j = 3;
    echo $i;
    echo "<br/>";
    echo $j;
?>
```

Si quisiéramos mostrar la suma de las dos variables podríamos hacer

```
echo $i + $j; //resultado 8
```

VARIABLES VI

Concatenar variables de texto:

```
<?php
    $texto1="Hola, ";
    $texto2="soy texto1";
    echo $texto1.$texto2; //resultado: Hola, soy texto1
?>
```

Concatenar variables numéricas:

```
<?php
    $num1=3;
    $num2=8;
    echo $num1.$num2; //resultado 38
?>
```

VARIABLES VII

Variables booleanas (variables que solo pueden valer verdadero -true- o falso -false-)

```
<?php
    $var1=true;
    $var2=false;
    echo $var1; //resultado 1
    echo '<br>';
    echo $var2; //resultado null
?>
```

VARIABLES VIII

Asignación:

Podemos asignar a una variable el contenido de otra simplemente indicando ésta última en la expresión:

```
<?php
    $var1=34;
    $var2=$var1;
    echo $var1;
    echo '<br>';
    echo $var2;
?>
```

VARIABLES IX

Suma de variables de distinto tipo:

ATENCIÓN porque php no da ningún error si intentamos sumar variables de distinto tipo:

```
<?php
    $var1=3;
    $var2='7';
    $var3='b';
    echo $var1+$var2; //resultado 10
    echo '<br>';
    echo $var1+$var3; //resultado 3
?>
```


CONSTANTES

Podemos definir también constantes que no podrán ser modificadas en ninguna parte del programa

```
<?php
    const PI = 3.1415;
    echo PI;
?>
```

O bien:

```
<?php
    define("PI", 3.1415);
    echo PI;
?>
```

EJERCICIO

Qué resultado obtendremos de estas expresiones:

1.- $\$a = 4; \$b = '5' \rightarrow \$a + \$b = ?$

2.- $\$a = 5; \$b = 4; \$c = '1' \rightarrow \$a + \$b + \$c = ?$

3.- $\$a = 'b'; \$b = 4; \$c = 'a' \rightarrow \$a + \$b + \$c = ?$

4.- $\$a = 3; \$b = 2; \$c = \$a \rightarrow \$a + \$b + \$c = ?$

5.- $\$a = 3; \$b = '2'; \$c = \$a+1 \rightarrow \$a + \$b + \$c = ?$

OPERADORES I

Los operadores son símbolos especiales que comúnmente se utilizan en expresiones.

Se pueden clasificar en:

- De concatenación
- Operadores aritméticos
- Operadores de asignación
- Operadores relacionales
- Operadores lógicos

OPERADORES II

Operadores de concatenación:

Son operadores binarios (requieren siempre dos operandos) que concatenan dos o más expresiones:

```
<?php  
    echo "texto1 " . "texto2 " . "texto3";  
?>
```

Resultado en pantalla:

texto1 texto2 texto3

OPERADORES III

Operadores aritméticos:

Son operadores binarios (requieren siempre dos operandos) que realizan las operaciones aritméticas habituales:

suma (+)

resta (-)

multiplicación (*)

división (/)

resto de la división (%)

OPERADORES III

Operadores aritméticos:

```
<?php
    echo "<br/> adición ". (7+2); //resultado 9
    echo "<br/> sustracción ". (7-2); //resultado 5
    echo "<br/> multiplicación ". (7*2); //resultado 14
    echo "<br/> división ". (7/2); //resultado 3.5
    echo "<br/> resto ". (7%2); //resultado 1
?>
```

OPERADORES IV

Operadores de asignación:

Los operadores de asignación permiten asignar un valor a una variable.

El operador de asignación es el igual (=).

La forma general de las sentencias de asignación con este operador es:

`$variable = expresión;`

OPERADORES VI

php dispone de otros operadores de asignación. Se trata de versiones abreviadas del operador (=) que realizan operaciones “acumulativas” sobre una variable.

Operador	Utilización	Expresión equivalente
<code>+=</code>	<code>op1 += op2</code>	<code>op1 = op1 + op2</code>
<code>-=</code>	<code>op1 -= op2</code>	<code>op1 = op1 - op2</code>
<code>*=</code>	<code>op1 *= op2</code>	<code>op1 = op1 * op2</code>
<code>/=</code>	<code>op1 /= op2</code>	<code>op1 = op1 / op2</code>
<code>%=</code>	<code>op1 %= op2</code>	<code>op1 = op1 % op2</code>

OPERADORES VII

Operadores unarios:

Los operadores más (+) y menos (-) unarios sirven para mantener o cambiar el signo de una variable, constante o expresión numérica.

`$a = 2;`

`$c = -$a // resultado $c = -2;` equivale a multiplicar por -1

`$b=-2`

`$c = -$b // resultado $c = 2;`

OPERADORES VII

Operadores unarios (continuación):

Los operadores (++) y (--) unarios sirven para incrementar o decrementar una unidad respectivamente.

Pueden prefijar o seguir a la variable.

```
<?php
    $a = 10; $b=10;
    $a++; //equivale a $a=$a+1
    $b--; //equivale a $b=$b-1
    echo "<br/> incremento 1 ". $a; //resultado 11
    echo "<br/> decremento 1 ". $b; //resultado 9
?>
```

OPERADORES VII

Operadores unarios (efecto colateral):

Operadores (++) y (—)

Observemos esta expresión:

```
$a=0; $b=0;  
$b=$a++;
```

Si ahora visualizamos las variables a y b obtendríamos:

$\$a \rightarrow 1$ (esperado ya que incrementamos en 1 la variable a)

$\$b \rightarrow 0$ (en principio esperaríamos que b adquiriera el valor de a, es decir 1)

La expresión $\$b=\$a++$ PRIMERO asigna el valor de 'a' a 'b' y DESPUÉS incrementa en 1 la variable 'a'

```
b = a  
a = a + 1
```

OPERADORES VII

Operadores unarios (efecto colateral):

Operadores (++) y (—)

Observemos ahora esta expresión:

```
$a=0; $b=0;  
$b=++$a;
```

Si ahora visualizamos las variables a y b obtendríamos:

$\$a \rightarrow 1$ (esperado ya que incrementamos en 1 la variable a)

$\$b \rightarrow 1$ (ahora si vemos que b toma el valor de a después del incremento)

La expresión $\$b=++\a PRIMERO incrementa en 1 la variable 'a' y DESPUÉS asigna el valor de 'a' a 'b'

```
a = a + 1  
b = a
```

OPERADORES VIII

Operadores relacionales:

Sirven para realizar comparaciones de igualdad, desigualdad y relación de menor o mayor.

El resultado de estos operadores es siempre un valor **booleano** (**true** o **false**) según se cumpla o no la relación considerada.

OPERADORES VIII

Operadores relacionales:

Operador	Utilización	El resultado es true
>	$op1 > op2$	si $op1$ es mayor que $op2$
>=	$op1 \geq op2$	si $op1$ es mayor o igual que $op2$
<	$op1 < op2$	si $op1$ es menor que $op2$
<=	$op1 \leq op2$	si $op1$ es menor o igual que $op2$
==	$op1 == op2$	si $op1$ y $op2$ son iguales
!=	$op1 \neq op2$	si $op1$ y $op2$ son diferentes

OPERADORES VIII

Operadores relacionales:

```
<?php
    echo "<br/> Mayor: ". (7>2);
    echo "<br/> Menor: ". (7<2);
    echo "<br/> Mayor o igual: ". (7>=2);
    echo "<br/> Menor o igual: ". (7<=2);
    echo "<br/> Igual: ". (7==2);
    echo "<br/> Distinto: ". (7!=2);
    echo "<br/> Igual y del mismo tipo: ". (7===7);
    echo "<br/> Igual y del mismo tipo: ". (7==='7');
?>
```

OPERADORES IX

Operadores lógicos:

Se utilizan para construir *expresiones lógicas*, combinando valores lógicos.

Operador	Nombre	Utilización	Resultado
&&	AND	op1 && op2	true si op1 y op2 son true. Si op1 es false ya no se evalúa op2
	OR	op1 op2	true si op1 u op2 son true. Si op1 es true ya no se evalúa op2
!	NOT	! op	true si op es false y false si op es true
&	AND	op1 & op2	true si op1 y op2 son true. Siempre se evalúa op2 Esto se hace así porque op2 podría ser una expresión que tuviera un efecto lateral sobre op1 haciendo que éste último sea cierto. Digamos que hace la evaluación después de procesar los dos operandos de la expresión. NO SE RECOMIENDA
	OR	op1 op2	true si op1 u op2 son true. Siempre se evalúa op2
^	XOR	op1 ^ op2	true si un operando es true y el otro false, esto es, si son diferentes

OPERADORES IX

Operadores lógicos:

```
<?php
    $op1=true; $op2=false;
    echo "<br/> op1: ". $op1 ." op2: ".$op2;
    echo "<br/> op1 && op2: ". ($op1 && $op2);
    echo "<br/> op1 || op2: ". ($op1 || $op2);
    echo "<br/> !op1: ". !$op1;
?>
```

PRECEDENCIA DE OPERADORES

En aritmética conocemos la regla que nos obliga a calcular la multiplicación antes de una suma. Esto también se cumple en PHP.

```
$j = 1 + 3 * 4; // resultado $j = 13
```

Si queremos cambiar esta prioridad utilizaremos los parentesis

```
$j = (1 + 3) * 4; // resultado $j = 16
```

Si todos los operadores tienen un nivel idéntico de precedencia se evalúa la expresión de izquierda a derecha.

```
$j = 1 + 3 - 4; // resultado $j= 0;
```

PETICIONES AL SERVIDOR: ENVIO

Cuando utilizamos un formulario para enviar datos al servidor podemos utilizar dos métodos:

GET → Los datos del formulario se envían por la URL

POST → Se envía el cuerpo del formulario entero

```
<form method="get" action="procesaFormulario.php">
```

ENVÍO DESDE FORMULARIO CON GET

Dado el siguiente formulario:

```
<form method="get" action="procesaFormulario.php">  
  <input type="text" name="nombre" placeholder="nombre"/>  
  <input type="number" name="edad" placeholder="edad"/>  
  <input type="text" name="telefono" placeholder="telefono"/><br><br>  
  <input type="submit" value='Enviar'>  
</form>
```

Al clicar sobre 'submit' estamos indicando al servidor que ejecute el archivo 'procesaFormulario.php' pasando por la url (método GET) los datos del mismo (atributos 'name')

url: <procesaFormulario.php?nombre=david&edad=44&telefono=55555555>

ENVÍO DESDE FORMULARIO CON POST

Dado el siguiente formulario:

```
<form method="post" action="procesaFormulario.php">  
  <input type="text" name="nombre" placeholder="nombre"/>  
  <input type="number" name="edad" placeholder="edad"/>  
  <input type="text" name="telefono" placeholder="telefono"/><br><br>  
  <input type="submit" value='Enviar'/>  
</form>
```

Al clicar sobre 'submit' estamos indicando al servidor que ejecute el archivo 'procesaFormulario.php' pasando en bloque todo el formulario (método POST)

url: [procesaFormulario.php](#)

PETICIONES AL SERVIDOR: RECEPCIÓN

- Con PHP podemos recuperar datos que nos llegan desde un formulario de nuestro documento web de la siguiente forma:

`$_GET['atributoName']` → si enviamos por método GET

`$_POST['atributoName']` → si enviamos por método POST

- Los datos se pueden enviar desde la página web ya sea directamente desde el formulario, desde javascript con AJAX o utilizando XML

RECEPCIÓN CON GET

La función procesaFormulario para el envío con GET es la siguiente:

```
<?php
    $nombre = $_GET['nombre'];
    $edad = $_GET['edad'];
    $telefono = $_GET['telefono'];
    echo $nombre." ".$edad." ".$telefono;
?>
```

El parámetro recogido en `$_GET` se corresponde con el atributo `'name'` del formulario y su valor es el que introducimos en el campo de formulario

RECEPCIÓN CON POST

La función procesaFormulario para el envío con POST es la siguiente:

```
<?php
    $nombre = $_POST['nombre'];
    $edad = $_POST['edad'];
    $telefono = $_POST['telefono'];
    echo $nombre." ".$edad." ".$telefono;
?>
```

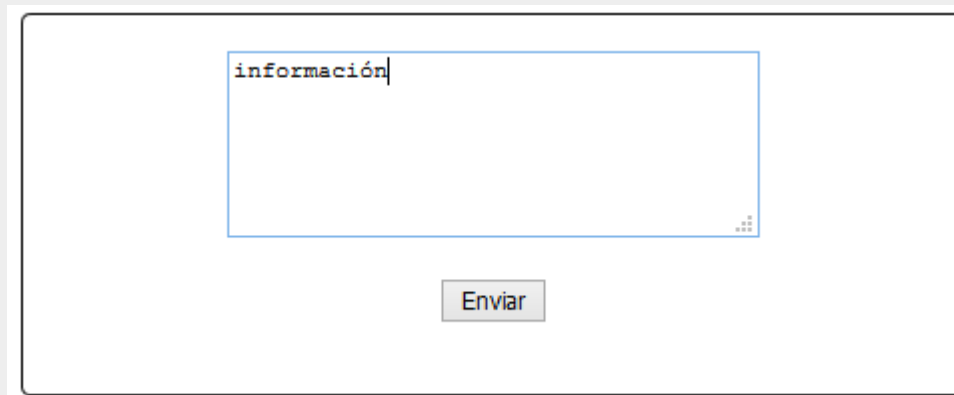
El parámetro recogido en **\$_POST** se corresponde con el atributo **'name'** del formulario y su valor es el que introducimos en el campo de formulario

GET vs POST

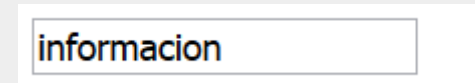
- GET es más rápido que POST.
- El número de caracteres que puedes enviar utilizando GET es limitado y depende del servidor.
- La información enviada por GET se puede visualizar en la URL, y por tanto no podríamos enviar información delicada (passwords, etc.)
- No es posible almacenar en favoritos la URL de una petición POST, ya que una petición POST no modifica la URL.

EJERCICIO

La información que introduzcamos en un **textarea** de un formulario aparecerá dentro de un campo de texto en una página distinta



A screenshot of a web form. It features a large rectangular text area with a blue border. Inside the text area, the word "información" is written in a monospaced font, and a cursor is positioned at the end of the word. Below the text area is a small, light gray button with the text "Enviar" in a dark gray font.



A screenshot of a web form. It features a single rectangular text input field with a light gray border. Inside the field, the word "informacion" is written in a monospaced font.

NOTA: Recordad que en la instrucción **echo** las etiquetas **html** se interpretarán como tales (por ejemplo si escribimos **echo "<input>"** en pantalla se mostrará un campo de input)

\$_REQUEST

Es idéntico a **\$_GET** y **\$_POST** pero no discrimina si los datos se han enviado desde el formulario html con el método **get** o **post**:

```
<?php
    $nombre = $_REQUEST['nombre'];
    $edad = $_REQUEST['edad'];
    $telefono = $_REQUEST['telefono'];
    echo $nombre." ".$edad." ".$telefono;
?>
```

NAME Y VALUE

- El **name** identifica a cada elemento; el **value** es la información que contiene.
- Con **\$_POST/\$_GET** recuperamos el **value** del elemento cuyo **name** pasamos como parámetro.
- Todas las personas tenemos un nombre y un valor, para conocer el valor de una persona en concreto, deberemos conocer su nombre, para poder identificarla.

Recojer información con acentos

Si tenemos algún problema en la recuperación de información con acentos (se nos muestra %C3%B3 en lugar de ó), podemos usar:

```
<?php  
    header('Content-Type: text/html; charset=UTF-8');  
?>
```

RECOGER RADIO BUTTONS

Podemos recoger los valores de un radio button de la siguiente forma:

```
<input type='radio' name='radio' value='radio 1'>  
<input type='radio' name='radio' value='radio 2'>
```

```
<?php  
    if (isset($_REQUEST['radio'])) {  
        $radio = $_REQUEST['radio'];  
        echo $radio;  
    }  
?>
```

NOTA: Para recuperar check boxes necesitaremos arrays

ESTRUCTURAS DE PROGRAMACION

ESTRUCTURAS DE PROGRAMACION

Las ***estructuras de programación*** o ***estructuras de control*** permiten tomar decisiones y realizar un proceso repetidas veces.

Son los denominados ***bifurcaciones o control de flujo*** y ***bucles o de iteración***.

En la mayoría de los lenguajes de programación, este tipo de estructuras son comunes en cuanto a *concepto*, aunque su *sintaxis* varía de un lenguaje a otro.

ESTRUCTURAS DE CONTROL

ESTRUCTURAS DE DECISION

Permiten ejecutar una de entre varias acciones en función del valor de una expresión lógica (cierto/falso) o relacional (igual, diferente, mayor, etc).

Son muy importantes ya que son las encargadas de controlar el flujo de ejecución de un programa.

Existen tres bifurcaciones diferentes:

- IF
- ELSE IF
- SWITCH

IF

Esta estructura permite ejecutar un conjunto de sentencias en función del valor que tenga la expresión de comparación (se ejecuta si la expresión de comparación tiene valor **true**).

Las **llaves** {} sirven para agrupar en un **bloque** las sentencias que se han de ejecutar, y no son necesarias si sólo hay una sentencia dentro del **if**.

Tiene la siguiente forma:

```
if (ExpresionLogica) {  
    sentencias;  
}
```

IF (EJEMPLO)

En esta expresión comparamos si un número es menor que otro para mostrar un mensaje en pantalla:

```
<?php
    $num1=10;
    $num2=12;
    if ($num1 < $num2) {
        echo "num1 es menor que num2";
    }
?>
```

EJERCICIO

Modifica el símbolo en rojo para que se imprima el mensaje.

```
<?php
    $dato1 = 10; $dato2 = 10;
    if($dato1 > $dato2){
        echo "exito";
    }
?>
```

IF ELSE

Análoga a la anterior, de la cual es una ampliación.

Las sentencias incluidas en el **else** se ejecutan en el caso de no cumplirse la expresión de comparación (**false**).

Tiene la siguiente forma:

```
if (ExpresionLogica) {  
    sentencias1;  
} else {  
    sentencias2;  
}
```

IF ELSE (EJEMPLO)

En esta expresión comparamos si un número es menor que otro para mostrar el mensaje que corresponda en pantalla

```
<?php
    $num1=10;  $num2=12;
    if ($num1 > $num2) {
        echo "num1 es menor que num2";
    } else {
        echo "num1 es mayor o igual que num2";
    }
?>
```

EJERCICIO

Informar dos variables con dos números.

Evaluarlas con un IF de forma que:

- Si el primer número es mayor que el segundo mostrar el mensaje “el primer número es el mayor”
- En caso contrario mostrar el mensaje “el segundo número es el mayor”

IF ELSE IF

Pero, ¿qué pasa si ambos números son iguales?. Pues que tendremos que introducir una tercera expresión para evaluar esta condición

```
<?php
    $num1=10;
    $num2=10;
    if ($num1 > $num2) {
        echo "el primer número es el mayor";
    } else if ($num1 < $num2) {
        echo "el segundo número es el mayor";
    } else {
        echo "los dos números son iguales";
    }
?>
```

IF ELSE IF

Si la primera condición no se cumple, se compara la segunda y así sucesivamente.

En el caso de que no se cumpla ninguna de las comparaciones se ejecutan las sentencias correspondientes al else.

```
if (ExpresionLogica1) {  
    sentencias1;  
} else if (ExpresionLogica2) {  
    sentencias2;  
} else if (ExpresionLogica3) {  
    sentencias3;  
} else {  
    sentencias4;  
}
```

EJERCICIO

Corrije los errores de este código:

```
<?php
    if {10 = 10} (
        echo "Verdadero"
    ) else (
        echo "Falso";
    )
?>
```

COMPROBAR CAMPO NUMÉRICO

Para comprobar en php si un campo es numérico:

```
if (is_numeric($num)) {echo "campo numérico"}  
if (!is_numeric($num)) {echo "campo no numérico"}
```

EJERCICIO

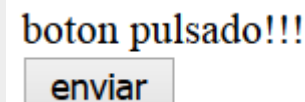
Utilizar un formulario para introducir la nota de un examen (de 0 a 10).

Al pulsar enviar se llamará a un fichero php con GET que realizará las siguientes validaciones:

- Si la nota es mayor que 5 mostrar el texto “Aprobado”
- Si la nota es menor que 5 mostrar “Suspenso”
- Si la nota es igual a 5 mostrar “Por los pelos”
- Si la nota es igual a 10 mostrar el texto 'Matrícula de Honor'

Procesar el formulario en la propia página

```
<?php
    if (isset($_POST['botonEnviar'])) {
        echo "boton pulsado!!!";
    }
?>
```



boton pulsado!!!

enviar

```
<form method="post" action="#">
    <input type="submit" name="botonEnviar" value="enviar">
</form>
```

EJERCICIO

Utilizar un formulario para introducir la nota de un examen (de 0 a 10).

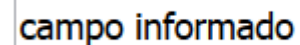
Al pulsar enviar tratar el formulario dentro de la propia página de forma que:

- Si la nota es mayor que 5 mostrar el texto “Aprobado”
- Si la nota es menor que 5 mostrar “Suspenso”
- Si la nota es igual a 5 mostrar “Por los pelos”
- Si la nota es igual a 10 mostrar el texto 'Matrícula de Honor'

INFORMAR UN CAMPO DE FORMULARIO CON PHP

Con php podemos informar un campo de formulario de la siguiente forma:

```
<?php  
    $valor='campo informado'  
?>
```



```
<input type='text' value='<?php echo $valor ?>'>
```

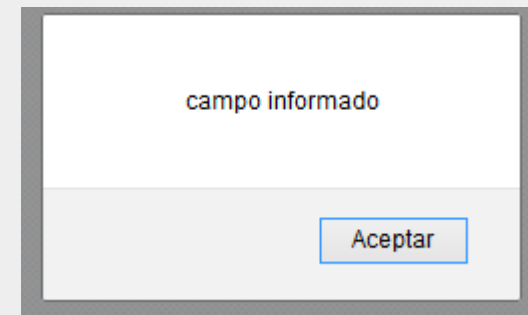
NOTA: Sintaxis reducida de la expresión php anterior: `<?=$valor?>`

INFORMAR VARIABLE JAVASCRIPT CON PHP

Con php también podemos informar una variable de javascript de la siguiente forma:

```
<?php  
    $varphp='campo informado'  
?>
```

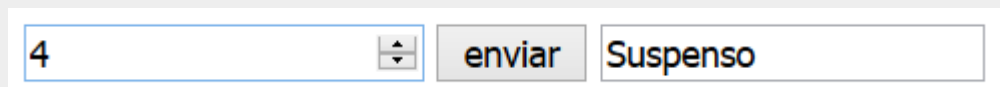
```
<script>  
    varjs = '<?=$varphp?>';  
    alert(varjs)  
</script>
```



EJERCICIO

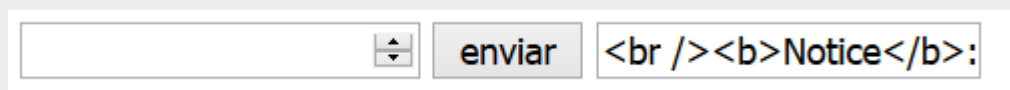
Repetir el ejercicio de las notas del exámen pero ahora:

- El resultado aparecerá en un campo de input situado al lado del botón de enviar.
- No se borrará la nota del examen



A form with a numeric input field containing the value '4', a button labeled 'enviar', and a text field containing the word 'Suspenso'.

NOTA: La variable a utilizar para mostrar la nota ya debe existir antes de mostrar la página. De lo contrario obtendríamos el siguiente error:



A form with an empty input field, a button labeled 'enviar', and a text field containing the error message: `
Notice:`

Si el campo de input al que dirigimos el resultado es numérico NO se visualizará el error aunque éste sigue existiendo

OPERADOR AND (&&)

Nos permite concatenar condiciones en un IF de forma que han de cumplirse todas ellas para que se ejecute el código:

```
<?php
    $num1=10; $num2=12;
    if ($num1>1 && $num2>10) {
        echo "Las dos condiciones son ciertas";
    } else {
        echo "Alguna de las condiciones no es cierta";
    }
?>
```

OPERADOR OR (||)

Nos permite concatenar condiciones en un IF de forma que solo con cumplirse una de ellas se ejecutará el código:

```
<?php
    $num1=0; $num2=12;
    if ($num1>1 || $num2>10) {
        echo "Al menos una de las condiciones es cierta";
    } else {
        echo "Ninguna de las condiciones es cierta";
    }
?>
```

EJERCICIO

Utilizando un formulario recoger dos números:

- Si los dos números son pares mostrar el mensaje: “los dos números son pares”
- Si al menos uno es par mostrar el mensaje: “uno de los números es par”
- Si ninguno es par mostrar el mensaje “los dos números son impares”
- El resultado se mostrará en un alert javascript

NOTA: Para saber si es par: $\text{numero} \% 2 = 0$

Número 1:



Número 2:



enviar

USO SIMULTANEO DE AND Y OR

El operador AND siempre se evalúa antes que el OR.

Consideremos la siguiente expresión:

```
$a=0; $b=5; $c=10; $d=15;
```

```
echo ($a<$b || $b>$c && $c>$d);
```

Si la ejecutamos nos devolverá: true

Mientras que la siguiente:

```
echo (($a<$b || $b>$c) && $c>$d);
```

nos devolverá: false

EJERCICIO

Determinar mentalmente el resultado de las siguientes operaciones o sentencias.

- A) `true || false && true`
- B) `(false || false) && (!true)`
- C) `!true && !false`
- D) `!(5 > 7 || 8 > 0 && 1 == 2)`

Realizar la prueba y ejecución de este código para verificar los resultados obtenidos.

SWITCH I

Se trata de una alternativa a la bifurcación if elseif cuando se compara la misma expresión con distintos valores. Su forma general es la siguiente:

```
switch(variable){  
    case valor1:  
        sentencias;  
        break;  
    case valor2:  
        sentencias;  
        break;  
    ...  
    default:  
        sentencias;  
}
```


SWITCH II

Switch no permite evaluar expresiones utilizando operadores lógicos. Es decir, la expresión **switch(\$num1>\$num2)** no es válida.

```
$ciudad = "Barcelona";  
switch ($ciudad) {  
    case "Oviedo":  
        echo "Nos vamos a Oviedo";  
        break;  
    case "Barcelona":  
        echo "Nos vamos a Barcelona";  
        break;  
    default:  
        echo "No nos vamos a ningún sitio";  
}
```

SWITCH III

La instrucción **break** es obligatoria para evitar que, una vez se cumpla uno de los '**case**' se sigan ejecutando los demás

Es conveniente utilizar **default** al final del bloque para tratar aquellos valores que no entren en ningún case

EJERCICIO

Hacer una calculadora que haga uso de los operadores aritméticos $+$, $-$, $*$, $/$, y determine la operación a realizar por medio de una estructura de control **else if** o **switch**.

El resultado se mostrará en un campo de input situado al lado del botón de enviar

<input type="text" value="7"/>	<input type="text" value="+"/>	<input type="text" value="5"/>	<input type="button" value="Enviar consulta"/>	<input type="text" value="12"/>
--------------------------------	--------------------------------	--------------------------------	--	---------------------------------

EJERCICIO

Modificar el ejercicio anterior de la calculadora para que, una vez realizado el cálculo, los campos de input deben recordar los valores utilizados antes de hacer el submit

7 + 5 12

EJERCICIO

Ahora la calculadora recordará también la opción seleccionada en la combo antes de hacer el submit

Pista!: Una opción de una combo está seleccionada si tiene un atributo **selected**

```
<?php if ($oper=='+') {echo 'selected';} ?>
```

ESTRUCTURAS DE ITERACIÓN

BUCLES

Un **bucle** se utiliza para realizar un proceso repetidas veces.

Se denomina también **estructura repetitiva** o **loop**.

El código incluido entre las **llaves** {} (opcionales si el proceso repetitivo consta de una sola línea), se ejecutará tantas veces como se indique en una variable o mientras se cumplan unas determinadas condiciones.

BUCLES

Hay que prestar especial atención a los bucles infinitos, hecho que ocurre cuando la condición de finalizar el bucle (***ExpresionLogica***) no se llega a cumplir nunca.

Tipos de bucles:

- FOR
- WHILE
- DO WHILE
- FOREACH (exclusiva para ARRAYS)

BUCLE FOR

Permite ejecutar un fragmento de código tantas veces como se especifique en los parámetros del bucle.

```
for (inicializacion; ExpresionLogica; incremento) {  
    sentencias;  
}
```

La **inicialización** se ejecuta una única vez al comienzo del **for** para inicializar la variable que se utilizará como el contador de iteraciones (variable de control).

El **incremento** se ejecuta *n* veces después de ejecutarse las **sentencias**.

La **ExpresionLogica** se evalúa al comienzo de cada iteración incluyendo la primera.

El bucle termina cuando la expresión de comparación toma el valor **false**.

BUCLE FOR (EJEMPLO)

```
<?php
    for($i=0; $i<10; $i++) { //$i++ equivale a $i=$i+1
        echo $i."<br/>";
    }
?>
```

Desde que \$i vale 0 **mientras** \$i sea menor que 10 ejecuta el código que se encuentre dentro del bucle y, **al final de cada iteración** suma 1 a \$i

EJERCICIO I

Imprimir los números pares que hay entre el 0 y un número que introducimos en pantalla mediante formulario.

NOTA: si queremos incrementar en más de 1 el contador del bucle podemos abreviar la expresión con `$i+=n` (donde n es el valor que queremos incrementar el contador)

NOTA: recordad que el operador % nos da el resto de dividir un número entre otro (ej: $7\%2=1$)

EJERCICIO II

Imprimir los números del 1 a otro número introducido mediante formulario:

- Para números divisibles por 3, imprimir “Fizz”.
- Para números divisibles por 5, imprimir “Buzz”.
- Para números divisibles por 3 y 5, imprimir “FizzBuzz”.
- En cualquier otro caso, imprimir el número.

BUCLE WHILE

Las sentencias que están dentro del bloque {} se ejecutan mientras la expresión lógica sea true.

```
while (ExpresionLogica) {  
    sentencias;  
}
```

Se utiliza cuando NO se sabe a priori cuantas veces se va a ejecutar el código contenido en él porque esto depende de una condición externa al flujo de programa (ej: decisión del usuario)

BUCLE WHILE

En este ejemplo se ejecuta el bucle mientras el número aleatorio generado no sea divisible entre 7

```
<?php
    $continuar=true;
    while($continuar) { //no hace falta preguntar por $continuar=true
        $num=rand(1,100); //generar número aleatorio de 1 al 100
        echo $num.'<br>';
        if ($num%7==0) {$continuar=false};
    }
?>
```

El código que se encuentra dentro del bucle WHILE se ejecutará siempre que se cumpla la condición que se especifica como parámetro

EJERCICIO

Tabla de multiplicar.

En un formulario se introducirá un número del 1 al 10.

Comprobar que el valor es numérico (`is_numeric()`)

Mostrar por pantalla la tabla de multiplicar de dicho número.

Tabla del: 7



Enviar consulta

7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
7 x 10 = 70

EJERCICIO

Tabla de multiplicar (continuación).

Validar que no se introduzca un número fuera del rango solicitado (1 a 10). En caso contrario mostrar el mensaje “número fuera de rango”

Tabla del:

número fuera de rango

EJERCICIO

Tablas de multiplicar

Hacer un nuevo ejercicio en donde se muestren por pantalla las tablas de multiplicar de los números del 1 al 10

1 x 1 = 1
1 x 2 = 2
1 x 3 = 3
1 x 4 = 4
1 x 5 = 5
1 x 6 = 6
1 x 7 = 7
1 x 8 = 8
1 x 9 = 9
1 x 10 = 10

2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
2 x 4 = 8
2 x 5 = 10
2 x 6 = 12
2 x 7 = 14
2 x 8 = 16
2 x 9 = 18
2 x 10 = 20

3 x 1 = 3
3 x 2 = 6
3 x 3 = 9
3 x 4 = 12
3 x 5 = 15
3 x 6 = 18
3 x 7 = 21
3 x 8 = 24
3 x 9 = 27
3 x 10 = 30

4 x 1 = 4
4 x 2 = 8
4 x 3 = 12
4 x 4 = 16
4 x 5 = 20
4 x 6 = 24
4 x 7 = 28
4 x 8 = 32
4 x 9 = 36
4 x 10 = 40

-->

9 x 1 = 9
9 x 2 = 18
9 x 3 = 27
9 x 4 = 36
9 x 5 = 45
9 x 6 = 54
9 x 7 = 63
9 x 8 = 72
9 x 9 = 81
9 x 10 = 90

10 x 1 = 10
10 x 2 = 20
10 x 3 = 30
10 x 4 = 40
10 x 5 = 50
10 x 6 = 60
10 x 7 = 70
10 x 8 = 80
10 x 9 = 90
10 x 10 = 100

SUPER EJERCICIO

Tablas de multiplicar (continuación)

Ya sabemos que con `echo` podemos incorporar etiquetas `html` y que éstas son interpretadas como tales por el navegador.

Utilizando esta característica repetir el ejercicio anterior pero, esta vez, la salida será en forma de tabla utilizando la etiqueta `<table></table>`

`table:`

`border:2px solid black`
`display:inline-block`

`td:`

`border:1px solid black`

1 x 1 = 1	2 x 1 = 2	3 x 1 = 3	4 x 1 = 4	5 x 1 = 5	6 x 1 = 6	7 x 1 = 7	8 x 1 = 8	9 x 1 = 9	10 x 1 = 10
1 x 2 = 2	2 x 2 = 4	3 x 2 = 6	4 x 2 = 8	5 x 2 = 10	6 x 2 = 12	7 x 2 = 14	8 x 2 = 16	9 x 2 = 18	10 x 2 = 20
1 x 3 = 3	2 x 3 = 6	3 x 3 = 9	4 x 3 = 12	5 x 3 = 15	6 x 3 = 18	7 x 3 = 21	8 x 3 = 24	9 x 3 = 27	10 x 3 = 30
1 x 4 = 4	2 x 4 = 8	3 x 4 = 12	4 x 4 = 16	5 x 4 = 20	6 x 4 = 24	7 x 4 = 28	8 x 4 = 32	9 x 4 = 36	10 x 4 = 40
1 x 5 = 5	2 x 5 = 10	3 x 5 = 15	4 x 5 = 20	5 x 5 = 25	6 x 5 = 30	7 x 5 = 35	8 x 5 = 40	9 x 5 = 45	10 x 5 = 50
1 x 6 = 6	2 x 6 = 12	3 x 6 = 18	4 x 6 = 24	5 x 6 = 30	6 x 6 = 36	7 x 6 = 42	8 x 6 = 48	9 x 6 = 54	10 x 6 = 60
1 x 7 = 7	2 x 7 = 14	3 x 7 = 21	4 x 7 = 28	5 x 7 = 35	6 x 7 = 42	7 x 7 = 49	8 x 7 = 56	9 x 7 = 63	10 x 7 = 70
1 x 8 = 8	2 x 8 = 16	3 x 8 = 24	4 x 8 = 32	5 x 8 = 40	6 x 8 = 48	7 x 8 = 56	8 x 8 = 64	9 x 8 = 72	10 x 8 = 80
1 x 9 = 9	2 x 9 = 18	3 x 9 = 27	4 x 9 = 36	5 x 9 = 45	6 x 9 = 54	7 x 9 = 63	8 x 9 = 72	9 x 9 = 81	10 x 9 = 90
1 x 10 = 10	2 x 10 = 20	3 x 10 = 30	4 x 10 = 40	5 x 10 = 50	6 x 10 = 60	7 x 10 = 70	8 x 10 = 80	9 x 10 = 90	10 x 10 = 100

SENTENCIAS BREAK Y CONTINUE

La sentencia **break** es válida tanto para las bifurcaciones como para los bucles (aunque se utiliza más en las de iteración). Hace que se salga inmediatamente de la repetitiva o bloque que se está ejecutando sin finalizar el resto de las sentencias.

La sentencia **continue** se utiliza sólo en los bucles (no en bifurcaciones). Finaliza la iteración que en ese momento se está ejecutando (no ejecuta el resto de sentencias que hubiera hasta el final del bucle). Vuelve al comienzo del bucle y comienza la siguiente iteración.

BUCLE DO WHILE

Es una variante de WHILE pero en este caso el código que se encuentra dentro del bucle se ejecuta una vez (antes de chequear si la condición es verdadera), posteriormente se repite mientras esta condición sea verdadera

```
<?php
    $i=5;
    do {
        echo "Contador: " . $i . "<br>";
        $i++;
    } while ($i < 5);
?>
```

FINALIZAR EJECUCION PHP

Si queremos finalizar la ejecución de un programa php cuando se produce un error controlado podemos utilizar la función nativa:

```
<?php
    $i='a';
    if (!is_numeric($i)) {
        die('dato no numérico');
    }
    echo $i;
?>
```