

3.1 Programación basada en lenguajes de marcas con código embebido

28 Septiembre 2022



Educar en la Verdad
para ser libres



Necesidades del Módulo 3: PROGRAMACIÓN BASADA EN LENGUAJES DE MARCAS CON CÓDIGO EMBEBIDO

3.1 Tomas de decisión

3.2 Bucles

3.3 Comentarios de cliente y de servidor

3.4 Tipos de datos compuestos

3.5 Arrays

3.6 Funciones

3.7 Paso de parámetros. Devolución de valores

3.8 Recuperación y utilización de información proveniente del cliente web

3.9 Interacción con el usuario: formularios

3.10 Procesamiento de la información introducida en un formulario



0. PROGRAMACIÓN BASADA EN LENGUAJES DE MARCAS CON CÓDIGO EMBEBIDO

En esta unidad vamos a profundizar en el ámbito de la programación. Programar es trasladar la solución a un problema o una necesidad a un código escrito según las reglas de un cierto lenguaje. El propósito de la programación es crear programas que exhiban un comportamiento deseado.

Los lenguajes de programación tienen unas reglas estrictas en la escritura de los programas, pero casi todas estas reglas se parecen entre lenguajes del mismo tipo y generación.

Los lenguajes de marcas son lenguajes imperativos con estructuras de objetos.

Esto significa que el código indica exactamente las acciones que hay que ejecutar y los objetos son, entre otras funcionalidades, los almacenes de la información.

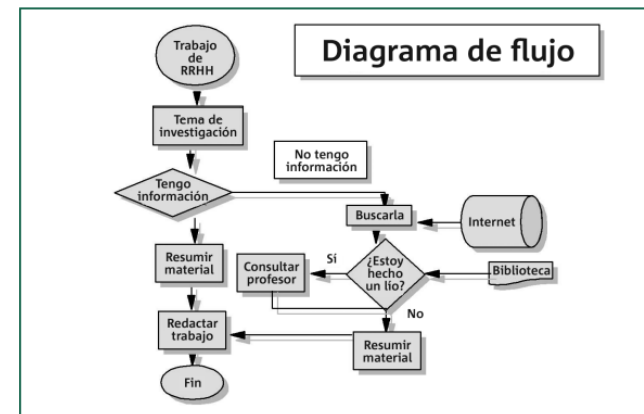
3.1 Tomas de decisión

Ya hemos visto en la unidad anterior que la ejecución del código se realiza de forma secuencial, línea a línea, sentencia a sentencia.

Pero no solamente hay que realizar acciones una detrás de otra; sino que, en ciertos momentos, deberemos tomar una decisión. Esta decisión hará que, en lugar de tener un camino a recorrer en ese instante, haya dos posibilidades y se deba tomar una posibilidad o la otra.

La toma de decisiones es fundamental en la programación, puesto que nos permiten generar código genérico que sea capaz de realizar muchas posibles acciones según las condiciones en las que se encuentre (Figura 3.1).

Figura 3.1
Ejemplo de diagrama de flujo
que ilustra el proceso de toma
de decisiones.



3.1 Tomas de decisión

Si no existieran las tomas de decisión, la realización de programas sería, cuando menos, extremadamente difícil sino imposible, puesto que se deberían realizar muchos programas en lugar de uno solo y además para los usuarios también sería muy complicado la elección de uno u otro programa porque tendrían que conocer de antemano todas las condiciones iniciales de cada uno de esos programas para poder escogerlos.

La toma de decisión fundamental es la instrucción if:

- **PHP, JSP, JavaScript.** Los paréntesis de la condición son necesarios. Las llaves indican dónde empieza y acaba el bloque de código:

```
if (condicion) {  
    sentencias_de_condicion_cierta  
} else {  
    sentencias_de_condicion_falsa  
}
```

3.1 Tomas de decisión

- VBScript:

```
if condicion then
sentencias_de_condicion_cierta
else
sentencias_de_condicion_falsa
end if
```

Cuando la condición resulta ser cierta se ejecutan las sentencias de la condición cierta, en cambio cuando la condición resulta falsa se ejecutan las sentencias de la condición falsa. Escribimos el código pensando en la posibilidad de ejecución de ambos grupos de sentencias, pero solamente una de ellas se ejecutará para ese trozo de código según el resultado de la condición. Veis que la condición es determinante para la ejecución del código, pero veamos que es realmente una condición. Una condición es una expresión que devuelve un resultado de tipo booleano: cierto o falso.



3.1 Tomas de decisión

Esta expresión puede ser:

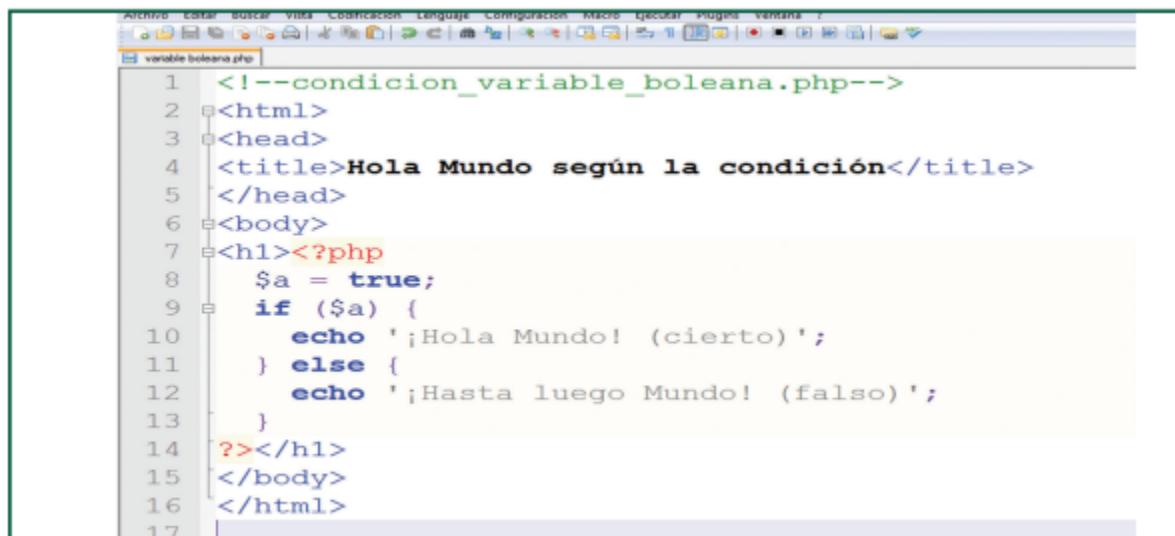
- Una variable booleana.
- Una comparación entre dos expresiones.
- Una función que retorne un resultado booleano.
- Cualquier combinación de las anteriores expresiones u otras expresiones que devuelvan un resultado final de tipo booleano.

Al hablar de comparación entre dos expresiones, nos referimos a las comparaciones igual, diferente, mayor que, menor que, mayor o igual que y menor o igual que. Estas dos últimas se escriben “tal y como se leen” por tanto mayor o igual que se escribe \geq y menor o igual que se escribe \leq . En caso de escribirlas al revés estaríamos cometiendo un error en la escritura de la comparación.



3.1 Tomas de decisión

Veamos un ejemplo (Figura 3.2):

A screenshot of a code editor window titled 'variable_boleana.php'. The code is written in PHP and HTML. It starts with a comment line: <!--condicion_variable_boleana.php-->. Then it opens an HTML document with <html>, <head>, and <title>Hola Mundo según la condición</title>. The body contains an <h1> tag with a PHP block. Inside the PHP block, a variable \$a is set to true, followed by an if statement. The if statement checks if \$a is true. If true, it echoes '¡Hola Mundo! (cierto)'. If false, it echoes '¡Hasta luego Mundo! (falso)'. The code ends with closing tags for the h1, body, and html.

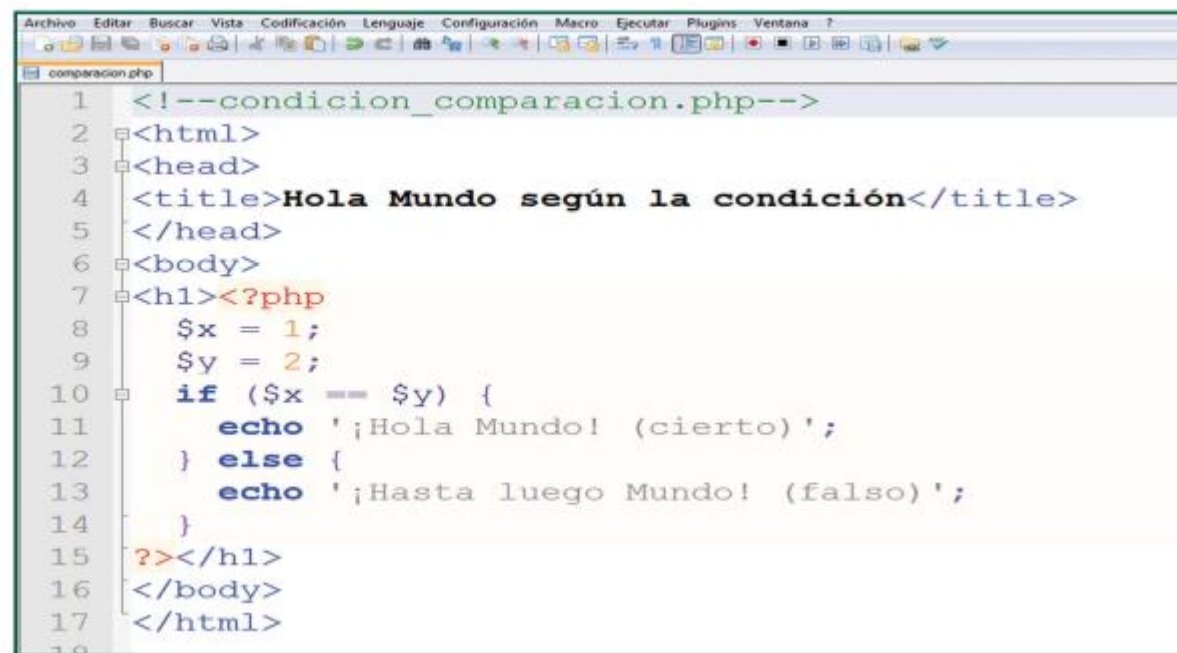
```
1 <!--condicion_variable_boleana.php-->
2 <html>
3 <head>
4 <title>Hola Mundo según la condición</title>
5 </head>
6 <body>
7 <h1><?php
8     $a = true;
9     if ($a) {
10         echo '¡Hola Mundo! (cierto)';
11     } else {
12         echo '¡Hasta luego Mundo! (falso)';
13     }
14     ?></h1>
15 </body>
16 </html>
17
```

Figura 3.2

Condición de variable booleana.

3.1 Tomas de decisión

En este caso, la condición es simplemente una variable booleana que producirá que la ejecución de la página retorne ¡Hola Mundo! (cierto); pero, si realizamos un cambio en la página y ponemos $\$a = \text{false}$, la página retornará ¡Hasta luego Mundo! (falso). Veamos otro ejemplo en la figura 3.3.

A screenshot of a code editor window titled 'comparacion.php'. The code is as follows:

```
1 <!--condicion_comparacion.php-->
2 <html>
3 <head>
4 <title>Hola Mundo según la condición</title>
5 </head>
6 <body>
7 <h1><?php
8     $x = 1;
9     $y = 2;
10    if ($x == $y) {
11        echo '¡Hola Mundo! (cierto)';
12    } else {
13        echo '¡Hasta luego Mundo! (falso)';
14    }
15    ?></h1>
16 </body>
17 </html>
```

Figura 3.3

Condición de comparación
de variables.



3.1 Tomas de decisión

En este caso, la ejecución de la página retornará ¡Hasta luego Mundo! (falso), pero si realizamos un cambio en la página y ponemos $x = 2$, entonces la página retornará ¡Hola Mundo! (cierto).

Esta segunda condición es una comparación entre variables.

Puedes observar que hay trozos de código que están colocados en una columna más a la derecha. Esta forma de escribir el código se llama indentación y se utiliza para clarificar la lectura de los programas, indicando que hay ciertos trozos de código que dependen del código inmediatamente superior

3.2 Bucles

Supongamos que necesitamos ejecutar dos veces una cierta acción, pero esta acción es exactamente la misma. Podemos programar de forma que esté escrita esa acción dos veces, una debajo de la otra.

Si se trata de dos posiblemente no tengamos ningún problema al programarlo, pero en caso que no sean dos veces sino más, entonces sí que existe un problema.

Es más, habrá veces en que se deban repetir esas acciones varias veces, pero además no se pueda saber la cantidad exacta de veces.

Cuando necesitamos realizar las mismas acciones varias veces de forma repetitiva estamos ante un caso claro del uso de un bucle. Hay dos tipos de bucles: condicionales y de cantidad fija de vueltas.



3.2.1 Bucles condicionales

Se realiza el bucle mientras la condición del bucle se cumpla. La condición booleana se define exactamente igual como en el apartado anterior.

Este es el bucle while:

- PHP, JSP, JavaScript :

```
while (condicion) {  
    sentencias_de_bucle  
}
```

- VBScript:

```
while condicion  
    sentencias_de_condicion_cierta  
wend
```

3.2.2 Bucles de cantidad fija de vueltas

Se realiza el bucle una cantidad determinada de veces. Es necesario saber de antemano la cantidad de veces que debe repetirse el bucle.

Este es el bucle for:

- PHP, JSP, JavaScript:

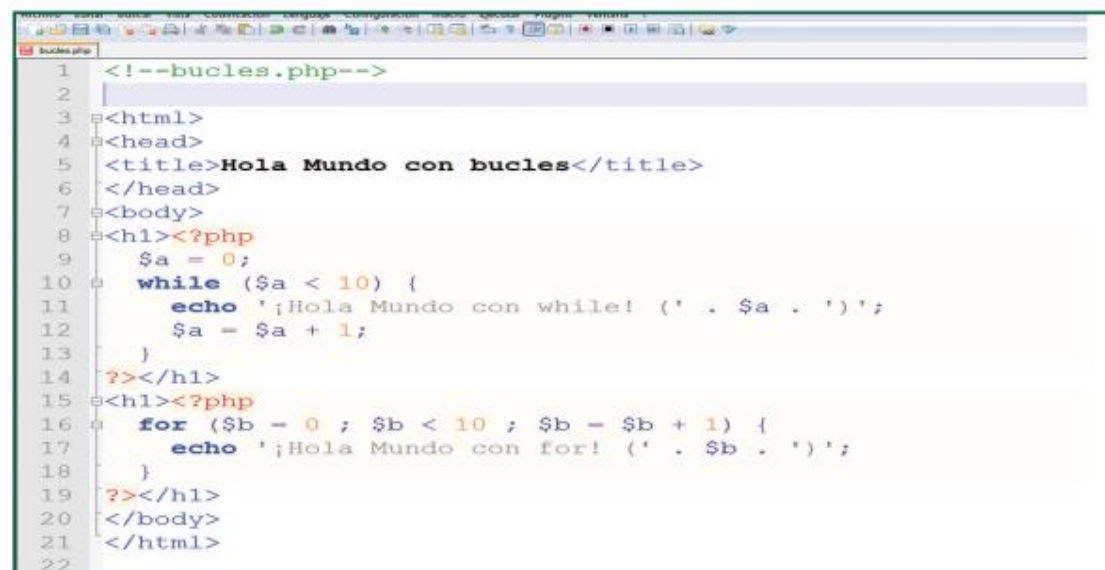
```
for (asignacion_de_variable ; condicion_de_variable ; siguiente_valor_de_variable) {  
    sentencias_de_bucle  
}
```

- VBScript:

```
for variable = valor_inicial to valor_final step salto do  
    sentencias_de_bucle  
next
```

3.2.2 Bucles de cantidad fija de vueltas

En este ejemplo (Figura 3.4) se puede observar que los dos tipos de *bucle while y for* se comportan exactamente de la misma forma aunque se escriban diferente



```
1 <!--bucles.php-->
2
3 <html>
4 <head>
5 <title>Hola Mundo con bucles</title>
6 </head>
7 <body>
8 <h1><?php
9     $a = 0;
10     while ($a < 10) {
11         echo '¡Hola Mundo con while! (' . $a . ')';
12         $a = $a + 1;
13     }
14     ?></h1>
15 <h1><?php
16     for ($b = 0 ; $b < 10 ; $b = $b + 1) {
17         echo '¡Hola Mundo con for! (' . $b . ')';
18     }
19     ?></h1>
20 </body>
21 </html>
22
```

Figura 3.4

Equivalencia entre bucle while
y bucle for.

La instrucción $\$a = \$a + 1$; debe leerse de esta forma: a la variable $\$a$ le modifico el valor leyendo el valor actual de $\$a$ y le sumo 1.

3.3 Comentarios de cliente y de servidor

Cuando queremos explicar de qué forma estamos escribiendo el código, o queremos indicar las decisiones que hemos tomado para escribir el código de esta forma y no de otra, usaremos los comentarios.

Los comentarios son textos que una vez reconocidos como comentarios simplemente se dejan pasar sin realizar ninguna acción. El objetivo principal de los comentarios es realizar indicaciones al respecto del código y esas indicaciones son totalmente libres sin que exista ninguna restricción del lenguaje.

Sabemos que el lenguaje HTML tiene también comentarios que son:

```
<!-- Todas las líneas escritas aquí dentro son comentarios HTML.-->
```

3.3 Comentarios de cliente y de servidor

Pero dentro del código de cliente y de servidor también podemos escribir comentarios y únicamente dependen del lenguaje que estemos utilizando:

- PHP, JSP, JavaScript:

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

- VBScript:

```
' Esto es un comentario de una sola línea
```




3.3 Comentarios de cliente y de servidor

De todos modos, existen ciertas herramientas de programación que utilizan los comentarios para realizar indicaciones al respecto de datos de la misma herramienta.

En estos casos los comentarios sí están regulados por esa herramienta y se aconseja a los programadores no tocar ninguno de esos comentarios porque pueden modificar el comportamiento de la herramienta.

Estas herramientas suelen estar vinculadas al lenguaje Java.

3.4 Tipos de datos compuestos

En los lenguajes de programación clásicos, un tipo de datos compuesto se genera usando dos o más tipos de datos simples o compuestos.

En los lenguajes de programación web, los tipos de datos compuestos se reducen a dos:

- **Objetos.** Elementos compuestos de atributos y métodos.
- **Arrays.** Listas de datos accesibles por su índice numérico o de texto.

La creación de objetos se realiza usando los elementos de programación llamados **clases**.

3.4 Tipos de datos compuestos

- PHP, JSP:

```
class nombre_de_clase {  
    //declaracion de atributos  
    private int id;  
    private int age;  
    //declaracion de métodos  
    public function calculo(entrada){  
        return entrada;  
    }  
}
```

- JavaScript:

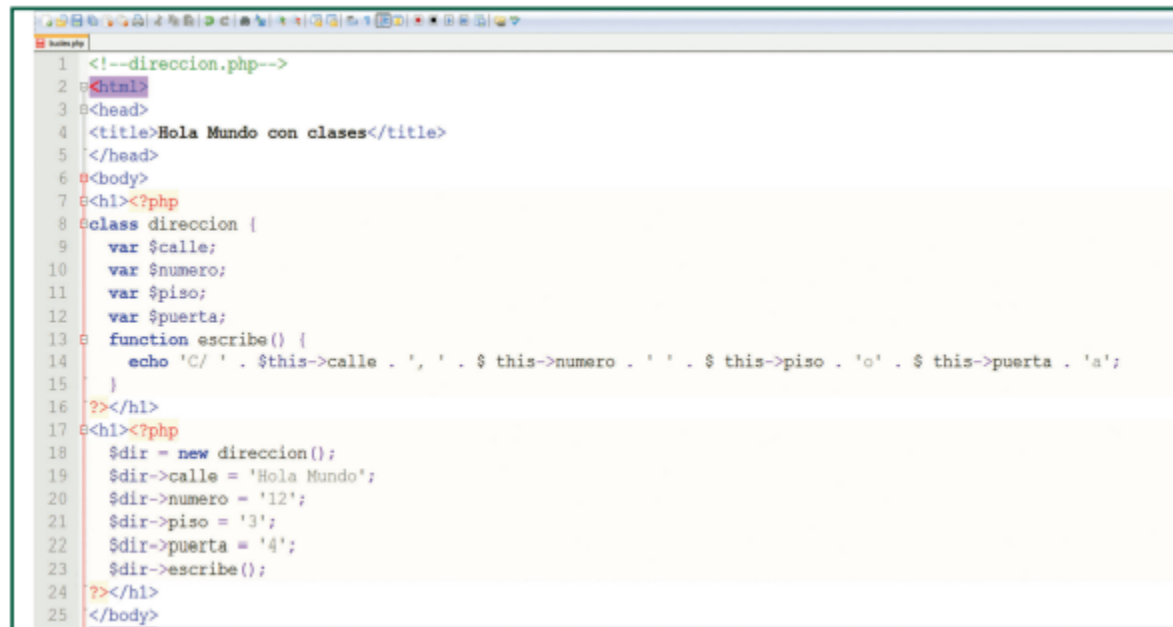
```
class nombre_de_clase {  
    //declaracion de atributos  
    int id = 0;  
    int age = 0;  
    //declaracion de métodos  
    void calculo(int entrada) {  
        return entrada;  
    }  
}
```

- VBScript:

```
class nombre_de_clase  
    //declaracion de atributos  
    private int id;  
    private int age;  
    //declaracion de métodos  
    public property calculo(int entrada)  
        return entrada;  
    end property  
end class
```

3.4 Tipos de datos compuestos

Recordemos el ejemplo de la dirección (Figura 3.5), el resultado de la página será C/ Hola Mundo, 12 3o 4a. Cuando escribimos programas, lo que estamos haciendo es encapsular una serie de acciones de forma que se ejecutan como si fuera una sola. Con la aparición de los objetos se amplía el encapsulamiento también a los datos. Esos datos se denominan atributos y las acciones se denominan métodos.



```
1 <!--direccion.php-->
2 <html>
3 <head>
4 <title>Hola Mundo con clases</title>
5 </head>
6 <body>
7 <h1><?php
8 class direccion {
9     var $calle;
10    var $numero;
11    var $piso;
12    var $puerta;
13    function escribe() {
14        echo 'C/ ' . $this->calle . ', ' . $this->numero . ' ' . $this->piso . 'o' . $this->puerta . 'a';
15    }
16    ?></h1>
17 <h1><?php
18     $dir = new direccion();
19     $dir->calle = 'Hola Mundo';
20     $dir->numero = '12';
21     $dir->piso = '3';
22     $dir->puerta = '4';
23     $dir->escribe();
24    ?></h1>
25 </body>
```

Figura 3.5
Declaración y uso de la
clase dirección.



3.4 Tipos de datos compuestos

Los atributos son las características propias del objeto y se definen como variables locales del objeto. Los métodos son las funciones que el objeto ofrece como interfaz para comunicarse con el exterior. Todos los lenguajes modernos aceptan la programación orientada a objetos (object oriented programming) siendo en este caso Java el lenguaje que, de todos los que estamos utilizando en este módulo, mejor se adapta a ellos puesto que Java es, desde sus inicios, un lenguaje orientado a objetos.

3.5 Arrays

Cuando tenemos necesidad de almacenar en variables una cantidad de valores del mismo tipo y del mismo concepto podemos usar un array (tabla).

La creación y el uso de los arrays depende de cada lenguaje:

- PHP:

```
nombre_de_variable = array ( key => value, ... )  
// donde key puede ser un integer o string  
// y value puede ser cualquier valor  
// ejemplo: $a = array( 1=> "uno", 2=> 2, "tres" => 3)
```

- JSP:

```
tipo_de_variable nombre_de_variable = new tipo_de_  
variable[cantidad];  
// ejemplo: int i = new int[5];
```

- JavaScript:

```
var nombre_de_variable = new Array(lista_de_valores);  
// ejemplo: var s = new Array("hola", "mundo");
```

3.5 Arrays

- VBScript:

```
dim nombre_de_variable(cantidad);  
// ejemplo: dim datos(15)
```

Los arrays son la primera forma y la más sencilla de almacenar un conjunto de elementos siendo la base de otros objetos más complejos que nos proporcionarán capacidades de almacenamiento temporal diferentes de las habituales.

Aparte de poder crear las estructuras que se requieran en cada momento, lo habitual es la utilización de estructuras estandarizadas como son los conjuntos, las listas, las colas, las pilas, los árboles o los grafos.

Cada una de estas estructuras suelen estar realizadas como objetos proporcionándonos sus atributos y métodos para usarlas, aunque internamente estarán implementadas utilizando arrays o estructuras de almacenamiento similares a los arrays.



3.6 Funciones

Un método o una función es un conjunto de instrucciones que se ejecutan como un grupo y puede ser llamada en cualquier momento desde otro bloque de código de la página. Hay muchas maneras de agrupar las funciones. Podemos hacerlo

usando diferentes conceptos como son:

- Por retorno. Las funciones son las que retornan un valor y métodos los que no retornan nada.
- Por ámbito. Hay públicas, semipúblicas o privadas.
- Por tipo. Pueden ser funciones de texto, de fecha, numéricas, de base de datos, de acceso a ficheros, etc.
- Por definición. Serán predefinidas si son propias del lenguaje o definidas por el usuario si no forman parte del lenguaje y las hemos escrito nosotros u otra persona



3.6 Funciones

Los métodos y las funciones se deben definir antes de utilizarse, puesto que, en caso contrario, al ejecutarse se producirá un error.

La diferencia fundamental entre método y función es que dan respuesta a dos conceptos diferentes:

- Método. Conjunto de instrucciones que realizan una acción.
- Función. Conjunto de instrucciones que retornan un resultado.

Veamos ahora cómo definir los métodos:

- PHP, JavaScript:

```
function nombre_de_metodo (lista_de_parametros) {  
    instrucciones_del_metodo;  
}
```

3.6 Funciones

- JSP:

```
void nombre_de_metodo (lista_de_parametros) {  
    instrucciones_del_metodo;  
}
```

- VBScript:

```
sub nombre_de_metodo (lista_de_parametros)  
    instrucciones_del_metodo  
end sub
```

Y las funciones se definen de este modo:

- PHP, JavaScript:

```
function nombre_de_funcion (lista_de_parametros) {  
    instrucciones_de_la_funcion;  
return valor_de_retorno;  
}
```

3.6 Funciones

- JSP:

```
tipo_de_datos nombre_de_funcion (lista_de_parametros) {  
    instrucciones_de_la_funcion;  
    return valor_de_retorno;  
}
```

- VBScript:

```
function nombre_de_funcion (lista_de_parametros) as tipo_de_re-  
torno  
    instrucciones_de_la_funcion  
    nombre_de_funcion = valor_de_retorno  
end function
```

3.7 Paso de parámetros. Devolución de valores

Cualquier proceso informático se define de esta manera (Figura 3.6), con independencia del lenguaje utilizado:

- Entrada
- Proceso
- Almacenamiento
- Salida

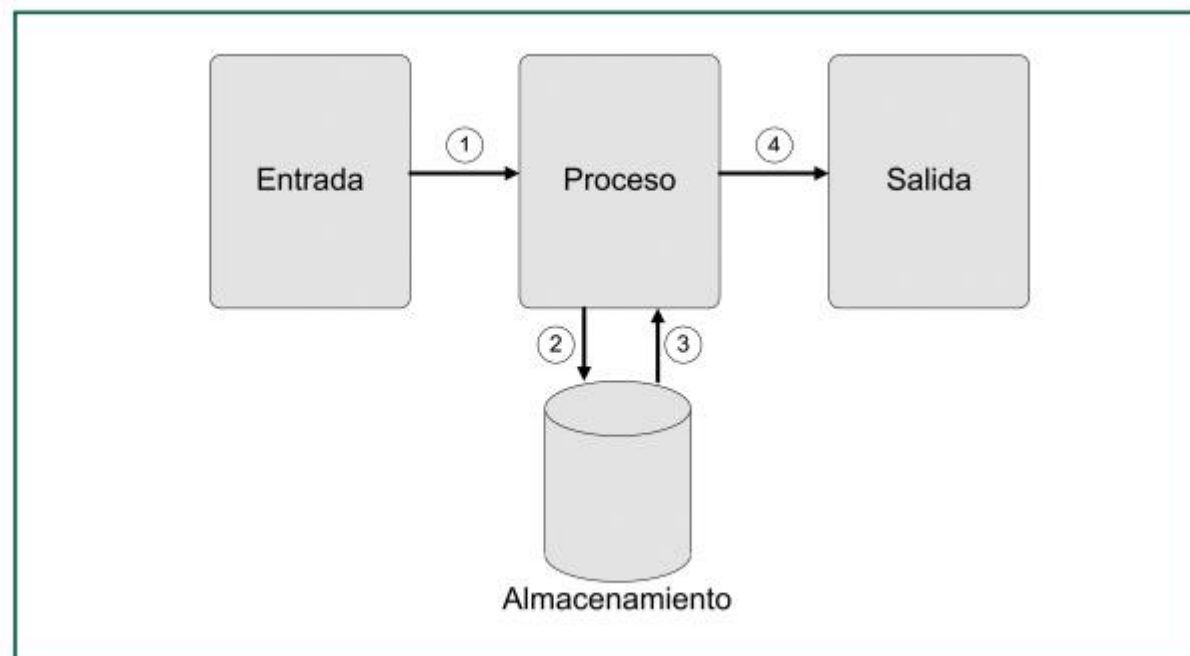


Figura 3.6
Esquema de entrada
– proceso – salida.

3.7 Paso de parámetros. Devolución de valores

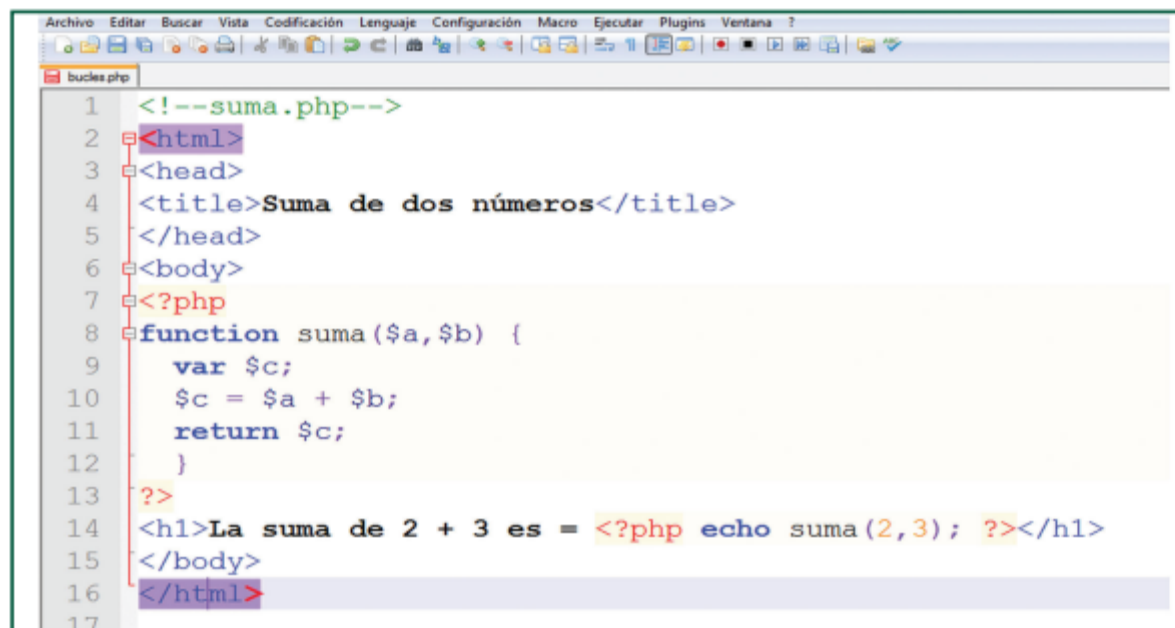
Los valores de entrada son los datos que se necesitan para poder realizar las acciones encomendadas. Cuando hablamos de métodos y funciones los valores de entrada se denominan parámetros de entrada.

El proceso es el bloque de código escrito para realizar una serie de acciones. El almacenamiento es opcional y se refiere al almacenamiento permanente de información.

La salida son los datos que se ofrecen como resultado de la ejecución del bloque de código. Cuando hablamos de métodos y funciones, la salida se denomina **retorno de valores**.

3.7 Paso de parámetros. Devolución de valores

Cualquier sistema informático funciona de esta manera, como por ejemplo las páginas web. Y no solamente las páginas web, sino cualquier método o función interna de las páginas web. Veámoslo con un ejemplo (Figura 3.7):



```
1  <!--suma.php-->
2  <html>
3  <head>
4  <title>Suma de dos números</title>
5  </head>
6  <body>
7  <?php
8  function suma($a,$b) {
9      var $c;
10     $c = $a + $b;
11     return $c;
12 }
13 ?>
14 <h1>La suma de 2 + 3 es = <?php echo suma(2,3); ?></h1>
15 </body>
16 </html>
17
```

Figura 3.7
Declaración y uso de la
clase dirección.

3.7 Paso de parámetros. Devolución de valores

La función suma tiene dos parámetros de entrada: \$a y \$b. Realiza la operación de sumar esos valores y los almacena temporalmente en la variable \$c. Finalmente retorna el valor obtenido en la suma.

Podemos cambiar los valores 2 y 3 por otros valores y comprobaremos que la operación se realiza correctamente. En este caso los valores 2 y 3 se pasan como parámetros de entrada a la función suma, y dentro de la función suma se han asignado a las variables \$a y \$b respectivamente. La instrucción echo nos escribe en la página un texto o un número y en este caso escribirá el resultado que la función suma(2,3) devuelva, evidentemente 5.

3.8 Recuperación y utilización de información proveniente del cliente web

Las páginas web estáticas se utilizan para obtener información fija del servidor web como cierta información, imágenes o vídeos. En este caso el cliente web únicamente realiza la petición de una página, imagen o vídeo. No se necesita más información. Pero en el caso de páginas web dinámicas o programas web se necesita que el usuario aporte más información al servidor de aplicaciones para poder generar el contenido dinámico correctamente (Figura 3.8).

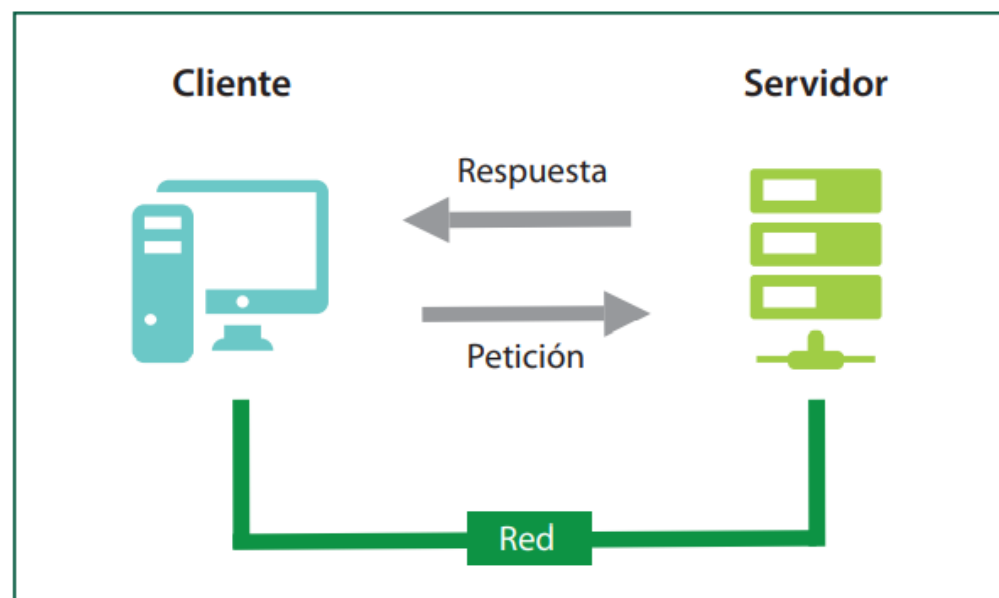


Figura 3.8
La información se envía del
cliente al servidor web.

3.8 Recuperación y utilización de información proveniente del cliente web

Si adaptamos el concepto anterior de cualquier proceso informático, podemos decir que:

- Entrada: parametro1=valor1 y parametro2=valor2
- Proceso: ejecución de la página web
- Salida: envío de la página web al cliente

Este método de envío de datos por parte del cliente se denomina método get.

Hay un límite en cuanto a la longitud máxima que puede tener una URL, pero estos límites son más por el navegador y el servidor que por la URL en sí misma. El límite está variando y se ha fijado en 1.024 caracteres de longitud en navegadores y servidores antiguos y en 2.048 en los modernos. Probablemente vaya aumentando en el futuro.



3.8 Recuperación y utilización de información proveniente del cliente web

Aun así, este límite es muy pequeño para enviar según qué tipo de información del cliente al servidor de aplicaciones. Para solucionar este problema tenemos el método post. Este método post nos permite enviar sin límite de longitud información del cliente al servidor.

Además, el método post realiza el envío de la información sin utilizar la URL, sino que utiliza la cabecera http para colocar los datos y enviarlos. Si queremos utilizar el método post deberemos generar una página web con formularios para que el cliente pueda enviarnos esa información.

Recuerda

Utiliza GET para acciones seguras y POST para las inseguras.

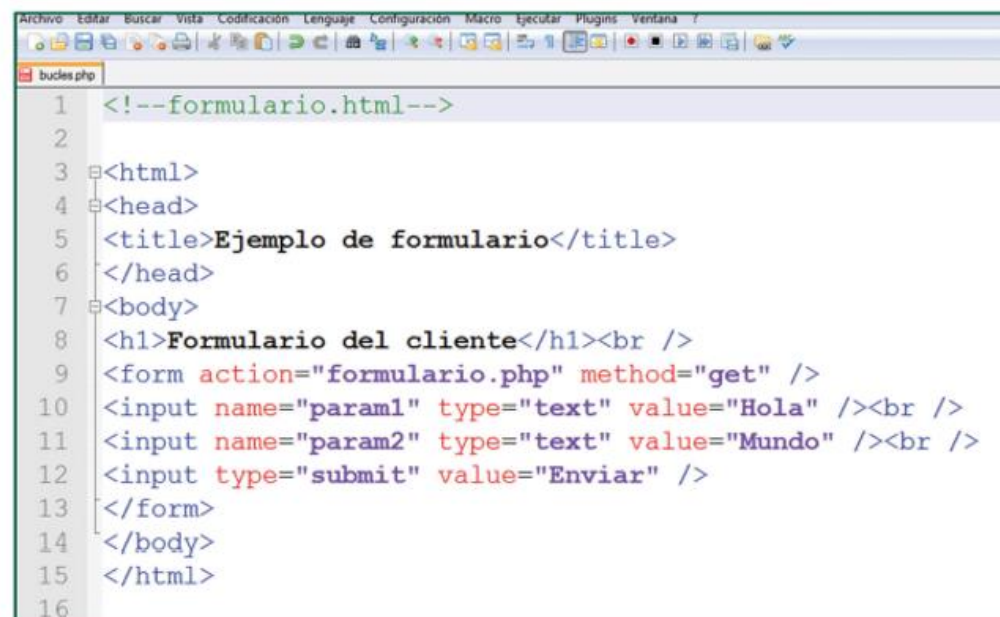
Utiliza POST si envías datos privados (passwords, números de tarjetas de crédito, etc.).

Utiliza POST cuando envíes mucha información (más de 2.024 caracteres).



3.9 Interacción con el usuario: formularios

Un formulario es un objeto html que utiliza el navegador del cliente para enviar información al servidor de aplicaciones. Los formularios se ejecutan en el navegador del cliente y son los encargados de recoger la información que el usuario desea enviar al servidor. Veamos un ejemplo en las figuras 3.9 y 3.10

A screenshot of a code editor window titled 'bucles.php'. The editor displays HTML code for a form. The code is as follows:

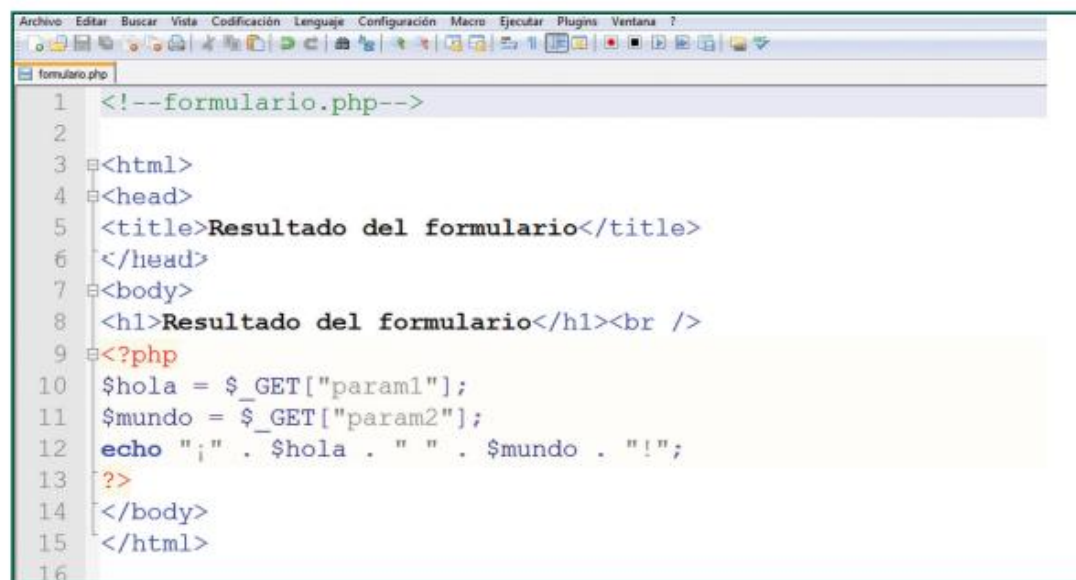
```
1 <!--formulario.html-->
2
3 <html>
4 <head>
5   <title>Ejemplo de formulario</title>
6 </head>
7 <body>
8   <h1>Formulario del cliente</h1><br />
9   <form action="formulario.php" method="get" />
10  <input name="param1" type="text" value="Hola" /><br />
11  <input name="param2" type="text" value="Mundo" /><br />
12  <input type="submit" value="Enviar" />
13 </form>
14 </body>
15 </html>
16
```

Figura 3.9
Ejemplo de formulario html.

3.9 Interacción con el usuario: formularios

Estos dos ficheros deben estar en el servidor web y el navegador del usuario debe llamar a la página html:

<http://miServidorWeb/formulario.html>

A screenshot of a code editor window titled 'formulario.php'. The editor shows PHP code that outputs an HTML page. The code includes a comment, HTML tags for head and body, and PHP code that retrieves 'param1' and 'param2' from the GET request and echoes them separated by a space.

```
1 <!--formulario.php-->
2
3 <html>
4 <head>
5   <title>Resultado del formulario</title>
6 </head>
7 <body>
8   <h1>Resultado del formulario</h1><br />
9 <?php
10 $hola = $_GET["param1"];
11 $mundo = $_GET["param2"];
12 echo "¡" . $hola . " " . $mundo . "!" ;
13 ?>
14 </body>
15 </html>
16
```

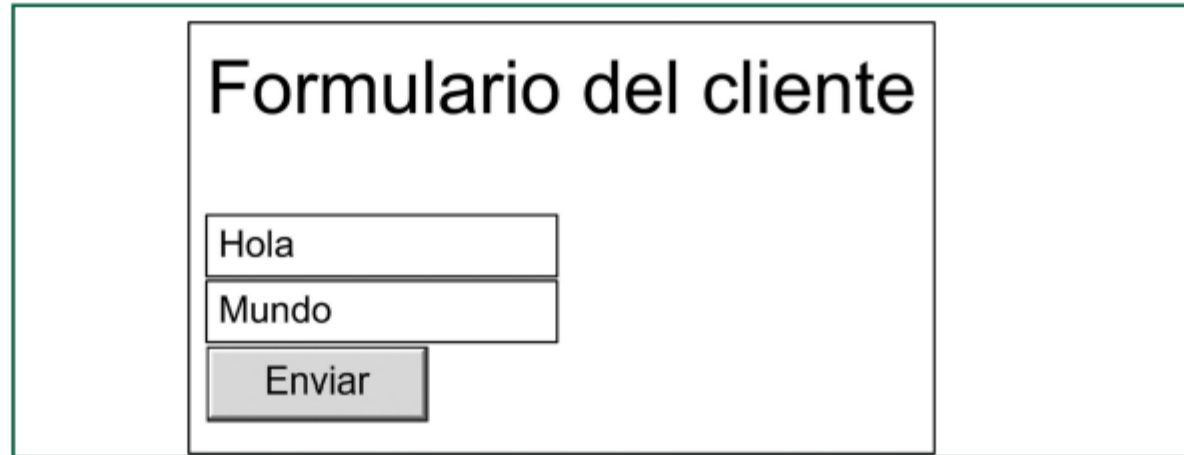
Figura 3.10

Ejemplo de formulario html.

3.9 Interacción con el usuario: formularios

El resultado puede verse en la figura 3.11.

Figura 3.11
Visualización del formulario en
un navegador.

A screenshot of a web browser displaying a form titled "Formulario del cliente". The form contains two text input fields. The first field contains the text "Hola" and the second field contains the text "Mundo". Below these fields is a button labeled "Enviar". The entire form is enclosed in a rectangular border.

Al pulsar el botón Enviar se está realizando esta petición:

`http://miServidorWeb/formulario.php?param1=Hola¶m2=Mundo`

El navegador del cliente realiza una petición GET con dos parámetros param1 y param2 a la página formulario.php.

3.9 Interacción con el usuario: formularios

Este funcionamiento es común a todos los navegadores y a cualquier lenguaje de programación de código embebido.

La petición que hemos descrito se realiza usando el método get, pero podemos modificar la página formulario.html cambiando el método get y poniendo el método post:

```
<form action="formulario.php" method="post" />
```

Para el usuario, la página seguirá siendo la misma y funcionará exactamente igual, pero al enviar ya no veremos los parámetros en la URL sino que solamente veremos esto:

<http://miServidorWeb/formulario.php>

Al llegar al servidor de aplicaciones esta nueva solicitud, el proceso que se debe realizar es exactamente el mismo que antes, solo cambia la forma de obtener los parámetros, que en este caso se han recibido usando el método post



3.10 Procesamiento de la información introducida en un formulario

Una vez se ha realizado la petición, el servidor de aplicaciones debe procesarla y obtener los parámetros que le ha enviado el usuario y reconocer si los hemos recibido usando el método get. El servidor de aplicaciones informará a unas variables de ámbito global de los parámetros que se han recibido. Estas variables globales son dos arrays diferentes: un array para los parámetros get y otro para los post.

Estos arrays son diferentes para cada lenguaje:

PHP: `$_GET["nombre_del_parametro"]` o `$_POST["nombre_del_parametro"]`

JSP: Los parámetros get y post se informan en el mismo array: **`request.getParameter("nombre_del_parametro")`**

ASP: Los parámetros get y post también se agrupan en el array: **`request.form("nombre_del_parametro")`**



3.10 Procesamiento de la información introducida en un formulario

Durante el tiempo de ejecución de los bloques de código, estas variables globales están siempre informadas y pueden ser accedidas como si fueran variables ordinarias. Estas variables globales que almacenan los parámetros de entrada se refieren a los parámetros que se han recibido en la última llamada a la página web.

Por tanto son volátiles. Esto significa que los datos almacenados en estas variables solamente podrán ser utilizados por la página web que las ha recibido y en ese instante del tiempo.

Una segunda ejecución procesará otros parámetros de entrada que no serán los mismos que los anteriormente recibidos.

En caso de necesitar almacenar de forma más permanente esos datos se deberán utilizar otras variables globales especialmente adecuadas para ese cometido.



3.10 Procesamiento de la información introducida en un formulario

Para almacenar la información:

- **PHP:** `$_SESSION["nombre_de_la_variable"] = valor;`
- **JSP:** `session.setAttribute("nombre_de_la_variable", valor);`
- **ASP:** `Session("nombre_de_la_variable ") = valor`

Para recuperar la información:

- **PHP:** `variable = $_SESSION["nombre_de_la_variable"];`
- **JSP:** `variable = session.getAttribute("nombre_de_la_variable");`
- **ASP:** `variable = Session("nombre_de_la_variable ")`





RESUMEN

Programar es trasladar la solución a un problema o una necesidad a un código escrito según las reglas de un cierto lenguaje. El propósito de la programación es crear programas que exhiban un comportamiento deseado.

Los lenguajes de programación tienen unas reglas estrictas en la escritura de los programas pero casi todas estas reglas se parecen entre lenguajes del mismo tipo y generación. Los programas no se ejecutan “en línea recta” sino que tienen elementos de decisión y estructuras iterativas (bucles) que les hacen “tomar algunas curvas” según sean las condiciones que en ese momento se plantean en el hilo de ejecución de nuestro programa.

Los lenguajes de marcas son lenguajes imperativos con estructuras de objetos. Esto significa que el código indica exactamente las acciones que hay que ejecutar y los objetos son, entre otras funcionalidades, los almacenes de la información.





RESUMEN

Tenemos diferentes formas en las que podemos almacenar la información tanto de forma simple como de forma compleja usando los tipos de datos compuestos como son, por ejemplo, los objetos con sus atributos y métodos o los arrays como listas de elementos.

Para poder interactuar con el usuario tenemos que utilizar el formulario. Un formulario es un objeto html que utiliza el navegador del cliente para enviar información al servidor de aplicaciones. Los formularios se utilizan para recoger la información que los usuarios introducen y poder enviarla a las páginas web encargadas de gestionar esa información



Bibliografía

- Vicente, J. (2013) “Desarrollo Web en entorno servidor”, editorial: Ibergaceta publicaciones, S.L. ISBN:978-84-1545-292-8
- Comesaña, L. (2017) “Desarrollo Web en entorno servidor”
- Desarrollo web en entorno servidor. Realización: ITACA (Interactive Training Advanced Computer Applications, S.L.) Colaboradores: Departamento de Producto de Centro de Estudios CEAC © Planeta DeAgostini Formación, S.L.U. Barcelona (España), 2016 ISBN: 978-84-9063-804-0 (Obra completa) ISBN: 978-84-9128-797-1 (Desarrollo web en entorno servidor)