Tema 03 – Programación Basada en Lenguaje de Marcas y Código Embebido

2º DAW – Desarrollo Web Entorno Servidor

Juan Carlos

CURSO 2024/2025

Tabla de Contenido

3 Programación Basada en Lenguaje de Marcas y Código Embebido

El capítulo anterior tenía como objetivo mostrar la sintaxis básica de los lenguajes de programación del entorno del servidor. Sin embargo para poder dominar la programación web, necesitamos conocer estructuras de control que nos permitan dotar de flexibilidad a las aplicaciones creadas. Para facilitar el desarrollo de aplicaciones web, necesitamos dominar además, diferentes técnicas de subprogramación basadas en la definición y utilización de procedimientos y funciones. Con dichas funciones, bien sean creadas por el programador o bien de forma predeterminada por el lenguaje, una aplicación web debe ser capaz de gestionar las peticiones del cliente. Por último un buen programador debe ser capaz de manejar con soltura estructuras de datos complejas, tales como los arrays o matrices. Además, en este capitulo estudiaremos los métodos básicos de comunicación con el cliente web y la forma de recuperar dicha información.

3.1 Sentencias Condicionales

Las sentencias condicionales son estructuras de control que permiten decidir el flujo de ejecución de un programa, es decir, el orden en el que las instrucciones de un programa se van ejecutar. Al introducir sentencias condicionales en nuestro código, este deja de ejecutar las instrucciones de manera secuencial, una detrás de otra, para poder definir caminos alternativos dependiendo de si se cumplen las condiciones establecidas por el programador.

Más tarde, definiremos la sintaxis de las distintas sentencias condicionales que existen en los lenguajes ASP, PHP y JSP, como son las sentencias *if* y *switch*.

Antes de nada veamos cómo se forman las condiciones en PHP y los operadores lógicos y de comparación:

3.1.1 Condiciones en PHP

Operadores de comparación

Se utilizan para establecer una relación entre dos valores. PHP compara estos valores entre si y esta comparación produce un resultado lógico (true o false).

Ejemplo	Nombre	Resultado
$a == $b	Igual	**TRUE** si $a es igual a $b después de la manipulación de tipos.
$a === $b	Idéntico	**TRUE** si $a es igual a $b, y son del mismo tipo.
$a != $b	Diferente	**TRUE** si $a no es igual a $b después de la manipulación de tipos.
$a <> $b	Diferente	**TRUE** si $a no es igual a $b después de la

		manipulación de tipos.
$a !== $b	No idéntico	**TRUE** si $a no es igual a $b, o si no son del mismo tipo.
$a < $b	Menor que	**TRUE** si $a es estrictamente menor que $b.
$a > $b	Mayor que	**TRUE** si $a es estrictamente mayor que $b.
$a <= $b	Menor o igual que	**TRUE** si $a es menor o igual que $b.
$a >= $b	Mayor o igual que	**TRUE** si $a es mayor o igual que $b.

Veamos el siguiente ejemplo

```
<?php
  // Teniendo las siguientes variables definidas:
  $a = 10;
  $b = '10';
  $c = 5;
  $d = 'Hola Pepe';
  $e = 'Hola Luis';
  $f = 'hola';

  // Comprobamos las expresiones:
  $a==$b;       // True son iguales
  $a===$b;      // False son iguales pero  de distinto tipo
  $a!==$b;      // True $a es de distinto  tipo que $b
  $b>$c;        // True $b es mayor que $c
  $a!=$c;       // True $a es distinto de  $c
  $a<>$c;       // True igual que la anterior
  $d==$e;       // False no son cadenas idénticas
  $d[0]==$e[0]; // True su  primer carácter es idéntico
  $d[0]==$f;    // False su primer carácter  es distinto (hay
distinción de mayúsculas y minúsculas)

  $Resultado=($a>$c)?  'Es Mayor':'Es Menor';
  // Dara como resultado Es Mayor porque $a es  mayor que $b
  echo $Resultado;
?>
```

Operadores Lógicos

También llamados operadores booleanos, se utilizan para crear condiciones compuestas en una fórmula. Al igual que los operadores condicionales devuelven un valor lógico de verdadero o falso y determinar si la expresión completa se cumple en función de las tablas de verdad.

Ejemplo	Nombre	Resultado
$a and $b	And (y)	**TRUE** si tanto *$a* como *$b* son **TRUE**.
$a or $b	Or (o inclusivo)	**TRUE** si cualquiera de *$a* o *$b* es **TRUE**.
$a xor $b	Xor (o exclusivo)	**TRUE** si *$a* o *$b* es **TRUE**, pero no ambos.
! $a	Not (no)	**TRUE** si *$a* no es **TRUE**.
$a && $b	And (y)	**TRUE** si tanto *$a* como *$b* son **TRUE**. Se evalúa antes que el operador and
$a || $b	Or (o inclusivo)	**TRUE** si cualquiera de *$a* o *$b* es **TRUE**. Se evalúa antes que el operador or

3.1.2 Sentencia If

Este tipo de estructuras de control condicionales definen dos flujos de ejecución dependiendo si se cumple o no la condición establecida por el programador. Si se cumple la condición se ejecutará una instrucción o un grupo de instrucciones. Si de lo contrario no se cumple la condición se ejecutarán otras instrucciones distintas.

Para el caso en el que solo queremos ejecutar ciertas instrucciones si se cumple la condición la sintaxis es la siguiente:

- **PHP y JSP**:

```
if (condición){
  instrucciones;
}
```

- **ASP**:

```
if (condición) then
  instrucciones
end if
```

Puede darse el caso de que necesitemos extender la funcionalidad del bloque if para controlar no solo qué instrucciones queremos que se ejecuten cuando se cumpla la condición, sino que además necesitamos especiñcar qué instrucciones queremos que se ejecuten cuando no se cumpla dicha condición. Esto se consigue haciendo uso del *else*

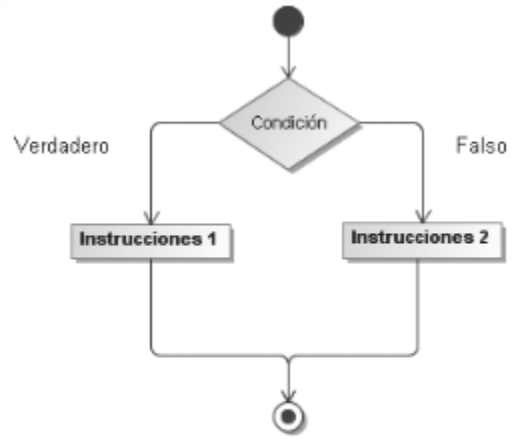

***Figura 3.1.** Diagrama de control de flujo de la sentencia `if-else`*

✓ **PHP y JSP**:

```
if (condición){
  instrucciones1;
}else{
  instrucciones2;
}
```

✓ **ASP**:

```
if (condición) then
  instrucciones1
else
  instrucciones2
end if
```

Otra de las posibilidades que nos ofrece la sentencia *if* es definir varios *if* anidados que nos permitan evaluar varias condiciones previas antes de ejecutar las instrucciones correspondientes.

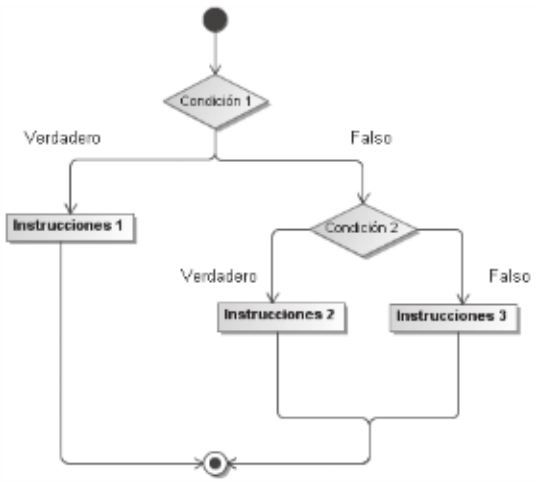

***Figura 3.2.** Diagrama de control de flujo de la sentencia `if-elseif`*

- **PHP**:
  - **Forma 1**:

```
if (condición1){
  instrucciones1;
}elseif (condición2){
  instrucciones2;
}else{
  instrucciones3;
}
```

  - **Forma 2 (Válido también para JSP)**:

```
if (condición1){
  instrucciones1;
}else if (condición2){
  instrucciones2;
}else{
  instrucciones3;
}
```

- **ASP**:
  - **Forma 1**:

```
if (condición1) then
  instrucciones1
elseif (condición2) then
  instrucciones2
else
  instrucciones3
end if
```

  - **Forma 2**:

```
if (condición1) then
  instrucciones1
else
  if (condición2) then
    instrucciones2
  else
    instrucciones3
  end if
end if
```

Con respecto a la forma de definir *if* anidados, destacar que PHP yASP permiten tanto el uso de *elseif* como *else if* mientras que en JSP solo se puede escribir separado.

3.1.2.1 If corto o abreviado

El objetivo es optimizar y minimizar el código al máximo sólo que se pierde un poco de claridad en cuanto a la estructura lógica del progrma.

Partimos de la estructura

```
if (condición){
  instrucciones1;
}else{
  instrucciones2;
}
```

Sería equivalente a

```
(condicion) ? instruccion1 : instruccion2;
```

Veamos el siguiente ejemplo

```
if ($a>$b)
{
  $resultado = "A es Mayor que B";
}
else
{
   $resultado = "B es Mayor que A";
}
```

Usando método abreviado se quedaría de la siguiente forma:

```
$resultado = ($a>$b) ? "A es Mayor que B":"B es Mayor que A";
```

3.1.2.2 Sintaxis If alternativa

Este tipo de instrucciones se usa para crear plantillas en HTML, es decir para combinar bloques de código de HTML con código PHP.

Sintaxis

```
<?php if(condicion): ?>
      Comandos HTML 1
<?php else: ?>
      Comandos HTML 2
<?php endif; ?>
```

Aunque la sintáxis anterior se puede usar de esta forma sin mezclar con código HTML

```
<?php
if (condición-1):
      bloque-instrucciones-1;
elseif (condición-2):
      bloque-instrucciones-2;
else:
      bloque-instrucciones-3;
endif;
?>
```

Ejemplos

```
<?php if ($a == 5): ?>
A es igual a 5
<?php endif; ?>
```

```
<?php if ($a == 5): ?>
      A es igual a 5
<?php else: ?>
      A no es igual a 5
<?php endif; ?>
```

```
<?php if ($a == 5): ?>
      A es igual a 5
<?php elseif ($a > 5): ?>
      A es mayor que 5
<?php else: ?>
      A es menor que 5
<?php endif; ?>
```

```
<?php
if ($a == 5):
    echo "a igual 5";
    echo "...";
elseif ($a == 6):
    echo "a igual 6";
    echo "!!!";
else:
    echo "a no es 5 ni 6";
endif;
?>
```

Como vemos el código resulta mucho más legible, sencillo y fácil de entender.

3.1.3 Sentencia SWITCH O SELECT CASE

Estas sentencias se usan cuando dependiendo del valor que toma una variable o expresión, necesitamos que se ejecute un conjunto de instrucciones distintas para cada uno de los valores que pueda tomar.

Su funcionamiento es sencillo, primero se calcula el valor de la expresión y se compara dicho valor con cada uno de los casos. Una que se encuentra el caso con el que coincide se ejecutan las instrucciones incluidas dentro del caso correspondiente. Si al evaluar todos los casos no coincide con ninguno se ejecutan las instrucciones definidas en el bloque por defecto si es que está definido, pues la declaración de este caso es opcional.

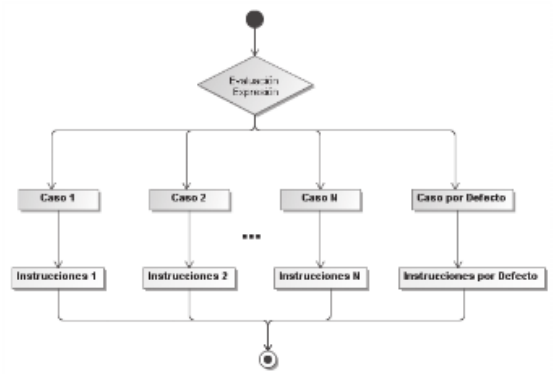

***Figura 3.3.** Diagrama de control de flujo de la sentencia `switch/select case`*

✓ **PHP y JSP**:

```
switch (expresión){
  case valor1: instrucciones1; break;
  case valor2: instrucciones2; break;
  ...
  case valorN: instruccionesN; break;
  [default: instruccionesN+1;]
}
```

✓ **ASP**:

```
select case (expresión)
  case valor1
    instrucciones1
  case valor2
    instrucciones2
  ...
  case valorN
    instruccionesN
  [case else
    instruccionesN+1]
end select
```

En ciertas ocasiones podemos utilizar los *switch* para sustituir a los if anidados lo que es altamente recomendable debido a que más eficiente computacionalmente el uso de estas sentencias, ya que ejecutan las instrucciones secuencialmente y evalúan la condición una sola vez. Además este tipo de sentencias presentan la ventaja de facilitar la lectura del código.

Con respecto a la sintaxis de estas sentencias señalar una de las diferencias más importantes que podemos apreciar entre los distintos lenguajes, que es el uso de *break* en JSP y PHP al final de la definición de cada caso. El uso de la sentencia *break* en JSP y PHP provoca que se interrumpa la evaluación de los casos siguientes una vez que se ha encontrado el caso que coincide con el valor resultante de evaluar la expresión. Otras de las diferencias que podemos encontrar es que mientras que para JSP y PHP el nombre que recibe el caso por detecto es *default*, para ASP es *case else*. Por último decir que en ASP el valor de los casos no va seguido de ":" y se indica el fin de la sentencia *select case* con la palabra reservada *end select*.

Veamos el siguiente ejemplo:

```
<?php
  $forma_pago=2;
  switch($forma_pago) {
```

```
    case 0: $pago='Contado';
    break;
    case 1: $pago='Transferencia bancaria';
    break;
    case 2: $pago='Contra reembolso';
    break;
    default: $pago='No definida';
  }
?>
```

La lista de sentencias para un caso también puede estar vacía, lo cual simplemente pasa el control a la lista de sentencias para el siguiente caso.

```
<?php
switch ($i) {
    case 0:
    case 1:
    case 2:
        echo "i es menor que 3 pero no negativo";
        break;
    case 3:
        echo "i es 3";
}
?>
```

3.1.3.1 Switch avanzado con condiciones

Existe la posibilidad de programar un *switch* para que se comporte de forma muy similar a como lo haría una cadena *de IF ELSEIF y ELSE* utilizando condiciones simples o múltiples.

Esta transformación del *switch* suele utilizarse cuando muchas condiciones contienen un código simple o cuando el valor de los *case* puede estar en un rango o varios.

Para lograr este comportamiento debemos, de alguna forma, terminar con la comparación de la variable del *switch* en PHP, para pasar a comprobar los case y sus condiciones.

Veamos un ejemplo de *switch* php con condiciones para un rango de valores, exactamente las edades posibles para la vida laboral:

```
<?php
    switch ( true ) {
    case ( $edad < 16 ):
          echo 'Edad No Laboral';
          break;
    case (  $edad >= 16 && $edad < 65 ):
          echo 'Edad Laboral';
          break;
     case ( $edad >= 65 ):
          echo 'Edad de jubilación';
          break;
      }
?>
```

3.1.3.2 Switch sintaxis alternativa

Como se dijo en el anterior apartado la sintaxis alternativa de esta estructura condicional se usa cuando queremos integrar código PHP en HTML, con el objeto de hacer lo que posteriormente conoceremos con el nombre del plantillas.

```
<?php switch ($var):

            case 1: ?>
      Bloque HTML 1
      <?php break; ?>

      <?php case 2: ?>
      Bloque HTML 2
      <?php break; ?>
      …
      <?php case N: ?>
      Bloque HTML N
      <?php break; ?>

      <?php default: ?>
      Bloque HTML DEFAULT

<?php endswitch; ?>
```

3.2 Setencias Iterativas o Bucles

Este tipo de sentencias se utilizan para ejecutar de forma reiterativa una instrucción o grupo de instrucciones.

Estas instrucciones se pueden ejecutar un número determinado o indeterminado de veces dependiendo del bucle que utilicemos para ello. En este apartado vamos a definir la sintaxis de cada uno de los bucles que existen para los lenguajes PHP, JSP y ASP, así como su comportamiento y su utilidad a la hora de programar.

Los bucles que vamos aver son los siguientes:

- While
- Do-while
- For
- Foreach

Al igual que en las sentencias condicionales, la condición puede ser simple o compuesta por varias expresiones lógicas y/o aritméticas cuyo valor resultante de evaluarlas es un valor booleano.

3.2.1 Bucle while

Este tipo de estructuras permiten ejecutar un número indeterminado de veces una instrucción o grupo de instrucciones, mientras se cumpla la condición. Como muestra la Figura 3.4— en

cada iteración del bucle se evalúa la condición y si esta es verdadera pasan a ejecutarse las instrucciones contenidas en el cuerpo del bucle. Finalmente el bucle termina cuando el resultado de evaluar la condición es falso, es decir, cuando la condición ha dejado de cumplirse.

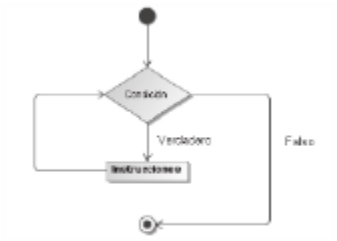

Figura 5.4. Diagrama de control de flujo del bucle while

A continuación mostramos su sintaxis:

✓ **PHP y JSP**:

```
while (condición){
  instrucciones;
}
```

✓ **ASP**:

– **Forma 1**:

```
do while (condición)
  instrucciones
loop
```

– **Forma 2**:

```
while (condición)
  instrucciones
wend
```

Como podemos observar, la Única diferencia entre la sintaxis utilizada por JSP, PHP, y la utilizada porA5P es el nombre que recibe el bucle para cada uno de ellos. Mientras que para JSP y PHP es while, para ASP es do while...loop.

Aunque ASP ofrece dos sintaxis distintas para definir este tipo de bucles, la más utilizada es la primera, ya que esta se refiere a la forma nueva de definir este bucle y la segunda es un vestigio de los inicios de Basic. Actualmente los intérpretes soportan también la segunda forma de definir este bucle, para los programadores reticentes autilizar la nueva forma pero es posible que en el futuro los intérpretes dejen de contemplarlo.

Veamos el siguente ejemplo

```
<?php

  $Contador=0;
  echo '<p>';
  while($Contador<=6) {
    echo "Contador con valor: $Contador</br>";
    $Contador++;
  }
  echo '</p>';
```

```
?>
```

Estructura While alternativa

Se puede usar sólo con código PHP

```
while (expr):
    sentencias
    ...
endwhile;
```

El siguiente ejemplo muestra los números del 1 al 10

```
<?php

      $i = 1;
      while ($i <= 10):
            echo $i;
            $i++;
      endwhile;
?>
```

O mezclando código PHP con HTML de la siguiente forma:

```
<?php while (expr): ?>
      Sentencias HTML
      ...
<?php endwhile; ?>
```

3.2.2 Bucle Do While

Este bucle es muy parecido al anterior con la salvedad de que siempre se ejecuta al menos una vez se cumpla o no la condición, debido a que la evaluación de la condición se realiza al final de cada iteración y no al principio. Como muestra la Figura 3.5, este bucle se ejecuta un número indeterminado de veces hasta que el resultado de evaluar la condición es falsa.

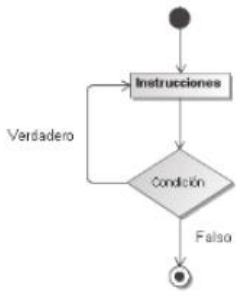

Figura 3.5. Diagrama de control de flujo del bucle `do-while`

La sintaxis para este bucle es la siguiente

✓ **PHP y JSP**:

```
do{
  instrucciones;
}while (condición);
```

✓ **ASP**:

```
do
  instrucciones
loop while (condición)
```

La sintaxis para este bucle es idéntica para PHP y JSP, mientras que para ASP lo único que cambia es el nombre del bucle.

Veamos el siguiente ejemplo:

```
<?php
      $i = 0;
      do {
            echo $i;
      } while ($i > 0);
?>
```

3.2.3 Bucle for

A diferencia del resto de bucles vistos hasta ahora este tipo de bucles se ejecutan un número determinado de veces y al igual que el resto de bucles permite ejecutar un conjunto de instrucciones de forma repetitiva.

La Figura 3.8 muestra el comportamiento del bucle for. Primero se inicializa el contador, que es el que va a controlar el número de veces que se ejecuta el bucle. Seguido de esto se evalúa la condición y si es verdadera se ejecuta el contenido del bucle y se actualiza el contador. En el momento en el que el resultado de evaluar la condición resulta falsa, el bucle iinaliza siguiendo con la ejecución de las instrucciones siguientes al bucle.

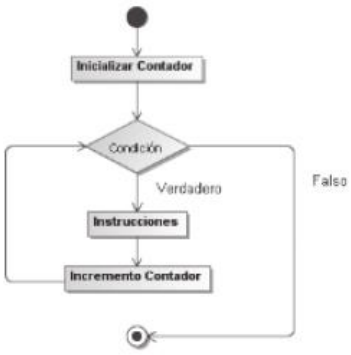

***Figura 3.8.** Diagrama de control de flujo del bucle* `for`

- **PHP y JSP**:

```
for([contador=valorInicial];[condición];[incremento]){
  instrucciones;
}
```

- **ASP**:

```
for contador=valorInicial to valorFinal [step incremento]
  instrucciones
next
```

Como podemos observar, la sintaxis del bucle for de PHP y JSP cambia un poco con respecto al for...next de ASP.

Vamos a explicar cómo se define este bucle para PHP y JSP3 y posteriormente para ASP.

En el primer caso, para definir un bucle for necesitamos crear un contador y asignarle un valor inicial. También definiremos la condición que se evaluará al principio de cada iteración del bucle para comprobar si se debe seguir ejecutando o no. Y por último indicaremos que incremento se le va a aplicar al contador en cada ejecución del bucle expresado en forma de expresión aritmética. El bucle se ejecutará mientras se cumpla la condición. Dicha condición suele ser una comparación en el que está implicado el contador, y es cuando llega éste a cierto valor cuando se sale del bucle y sigue ejecutando las instrucciones siguientes. Otra de las cosas que tenemos que tener en cuenta es que la declaración del contador es opcional, ya que lo podemos crear e inicializar fuera del bucle antes de este. Lo mismo pasa con el incremento y la ccmdiciún que no es necesario delinirla el bucle si la definimos posteriormente dentro del mismo.

Con respecto al comportamiento y la definición del bucle for...next en ASP, la primera diferencia es que no necesita deiinir ninguna condición, sino que en su lugar se crea un contador y se le da un valor inicial. Además de indicar el valor inicial que toma también se especifica el valor que puede tomar como límite, cuando el contador toma el valor definido como límite se ejecuta por última vez el bucle y se sale de este para seguir con la ejecución secuencial de las siguientes instrucciones. De forma opcional podemos definir el valor con el que vamos a incrementar el contador en cada ejecución del bucle, pero si no lo definimos por defecto se incrementara el contador en uno.

Veamos los siguientes ejemplos

```php
<?php
/* ejemplo 1 */

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

/* ejemplo 2 */

for ($i = 1; ; $i++) {
    if ($i > 10) {
        break;
    }
```

```
    echo $i;
}

/* ejemplo 3 */

$i = 1;
for (; ; ) {
    if ($i > 10) {
        break;
    }
    echo $i;
    $i++;
}
```

Sintaxis alternativa

Sólo con PHP

```
for (ini-contador;condicion;incremento):

        bloque-instrucciones;

endfor;
```

Ejemplo

```
<?php

    for ($i=1;$i<=10;$i++):
        echo $i;
    endfor;

?>
```

Mezclando PHP con HTML

```
<?php for (ini-contador;condicion;incremento):?>

        bloque-instrucciones-HTML;

<?php endfor; ?>
```

Ejemplo

```
<?php for ($i=1;$i<=10;$i++):?>

        <p>Párrafo  <?=$i?> </p>

<?php endfor; ?>
```

3.3 Tipos de Datos Complejos

En este apartado vamos a aprender a definir, inicializar y acceder a los valores de un **array** o **matriz**, que no es más que un array multidimensional.

3.3.1 Definición y Acceso

Los arrays o matrices son estructuras que permiten el almacenamiento de un conjunto de datos, son una construcción tradicional de los lenguajes de programación.

Podemos delinir un **array** o **matriz** como un conjunto ordenado de elementos identificados por un índice (la posición del elemento dentro de esta colección ordenada), de modo que en cada posición marcada por un indice el array contiene un valor. Podemos construir tantos índices como queramos, aunque el uso habitual de los arrays es en forma de matriz unidimensional.

La **longitud** del array se modifica de forma dinámica siempre que añadimos un nuevo elemento.

A continuación mostramos la sintaxis para definir un **array** en los distintos lenguajes:

- **PHP**: `nombreVariable=array(clave => valor,…);`
- **ASP**: `Dim nombreVariable(tamaño)`
- **JSP**:
  - **Forma 1**:

```
tipo[] nombreVariable  = new tipo [tamaño];
```

  - **Forma 2**:

```
tipo[] nombreVariable  = {valor1,valor2,…,valorN};
```

Una ver vista la sintaxis para declarar un array, vamos a ver un mismo ejemplo en todos los lenguajes que nos ayude a comprender como se declara e inicializa un array:

- **PHP**: `$miarray=array(0=>2,1=>4);`
- **ASP**:

```
Dim miarray(1)
miarray(0)=2
miarray(1)=4
```

- **JSP**:
  - **Forma 1**:

```
int[] miarray= new int [2];
miarray[0]=2;
miarray[1]=4;
```

  - **Forma 2**:

```
int[] miarray= {2,4};
```

Por otro lado para definir una **matriz** la sintaxis es la siguiente:

- **PHP**: `array(clave => array(),…);`
- **ASP**: `Dim nombreVariable(n° filas,n° columnas)`
- **JSP**:
  - **Forma 1**:

```
tipo[][] nombreVariable = new tipo [n°filas][n°columnas];
```

  - **Forma 2**:

```
tipo [][] nombreVariable = {valor1.1, valor1.2, …,
  valor1.N},…,{valorN.1,valorN.2,…,valorN.N}};
```

Al igual que con los arrays, vamos a ver con un mismo ejemplo para los distintos lenguajes como se **define** y se **inicializa** una matriz. Sobre estos ejemplos explicaremos la diferencia entre las distintas formas que existen para definirlos:

- **PHP**: `mimatriz(0 =>array(0=>2,1=>4),1 =>array(0=>1,1=>3));`
- **ASP**:

```
Dim mimatriz(1,1)
mimatriz(0)(0)=2
mimatriz(0)(1)=4
mimatriz(1)(0)=1
mimatriz(1)(1)=3
```

- **JSP**:
  - **Forma 1**:

```
int[][] mimatriz = new int [2][2];
mimatriz[0][0]=2;
mimatriz[0][1]=4;
mimatriz[1][0]=1;
mimatriz[1][1]=3;
```

  - **Forma 2**:

```
int [][] mimatriz = {{2,4},{1,3}};
```

Como podemos ver en los ejemplos aunque la sintaxis cambia el comportamiento es el mismo.

En **PHP** la definición de un **array** se define por el par ***clave => valor***, en el que la clave hace referencia al índice y es opcional y el valor se refiere a lo que se va almacenar en esa posición del array. Mientras que la **clave** solo puede ser de tipo entero o string (cadena de caracteres), el valor puede ser de cualquier tipo de dato.

Para definir un array en **ASP** se utiliza la palabra reservada dim seguido por el nombre de la variable y el tamaño que tendrá entre paréntesis. Con respecto al tamaño tenemos que tener en cuenta que si por ejemplo lo definimos con tamaño 3, será uno más el número de elementos que podremos almacenar, en este caso serían 4. Esto es debido a que el array empieza en 0 y toma como límite el número que le hemos pasado como tamaño.

En cambio en **JSP**, existen varias formas de definir e inicializar un array, la primera de ellas lo que hace es crear una variable de tipo array. Con la smtencia new reserva memoria, guarda la referencia en la variable y posteriormente lo inicializa, accediendo al array a través de los índices contenidos entre corchetes y no entre paréntesis como lo hace ASP. La segunda forma en cambio declara e inicializa el array en una misma línea.

El **acceso** a los valores de un array se consigue gracias a la utilización de los índices. A continuación mostramos como acceder a la pisición 0 del array creado anteriormente:

- **PHP y JSP**: `miarray[0]`
- **ASP**: `miarray(0)`

Si queremos acceder al valor de cada una de las posiciones de un array necesitamos utilizar un bucle *for* o *foreach* con el que poder iterar sobre los indices del array acceder al valor almacenado en cada posición. Para ello vamos a realizar un ejemplo en cada uno de los lenguajes en el que mostramos como recorrer un array e imprimir el valor contenido en cada una de las posiciones:

- **PHP**:

```
foreach($miarray as $valor){
  echo $valor;
}
```

- **ASP**:

```
for each valor in miarray
  response.write(valor)
next
```

- **JSP**:

```
for(i=0;i<miarray.length;i++){
  out.println(miarray[i]);
}
```

Tanto en el ejemplo de ASP como en el de PHP hemos utilizado el bucle *foreach*, ya que estos bucles han sido creados para este tipo de tareas, aunque también podriamos haber utilizado el bucle *for* que hemos explicado en el apartado anterior. También podemos apreciar que para imprimir valores por pantalla hemos utilizado la función *out.println* en JSP, *echo* en PHP y *response.write* en ASP.

Por último añadir una serie de funcionalidades que presenta PHP para el manejo de arrays:

- Borrar un array entero:
  unset($miarray);
- Borrar un elemento de un array:
  unset($miarray[0]);
- Añadir un nuevo elemento a un array:
  $miarray[]=8;

3.3.2 Tipos de arrays

3.3.2.1 Escalares

Los arrays escalares son aquellos en los que para acceder a los elementos utilizamos un índice que representa la posición del valor dentro del array comenzando desde el índice 0. También se les llama **arrays indexados**.

Para definir estos tipos de arrays tenemos dos formas posibles:

- Usandon el constructor *array()*

- Utilizando la notación de corchetes

```
$miArray = array(valor1, valor2, valor3, ...);
$miArray = [Valor1, valor2, valor3, ...];
```

Veamos el siguiente ejemplo:

```
// Array escalar de valores numéricos  utilizando el constructor
array()
   $variable = array(10, 20,  30, 40, 50);
   echo $variable[0]; -> 10
   echo $variable[3]; -> 40

// Array escalar de valores cadena  utilizando la notación de
corchetes
    $variable = ['Enero', 'Febrero', 'Marzo', 'Abril'];
    echo $variable[0]; -> 'Enero'
    echo $variable[2]; -> 'Marzo'

// Array escalar de valores combinados  utilizando la notación de
corchetes
    $variable = ['Enero', 10, 'Marzo',  20, 'Abril'];
    echo $variable[0]; -> 'Enero'
    echo $variable[3]; -> 20
```

3.3.2.2 Asociativos

Para acceder a los elementos del array usamos claves personalizadas que podrán ser bien números o string, para ello por cada elemento usamos la sintaxis clave => valor.

Si la clave es de tipo string habrá que usar las comillas en el índice. Por ejemplo $clientes[‘nombre’].

No podremos acceder a los elementos por medio de un índice numérico, como el caso de los arrays escalares, a no ser que la clave asociada al valor sea de tipo numérica.

Para definir un array asociativo usamos la siguiente sintaxis:

```
$miArray = array(clv1=>valor2,clv2=>valor2,clv3=>valor3,...);
$miArray = [clv1=>valor1,clv2=>valor2,clv3=>valor3,...];
```

Veamos el siguiente ejemplo:

```
// Array asociativo con claves string y valores  numéricos
   $Notas = ['Luis'=>6, 'Carmen'=>6, 'Pedro'=>3, 'Rosa'=>8);
   echo $Notas['Pedro']; -> 3
   $indice='Luis';
   echo $Notas[$indice]; -> 6 // Accedemos al elemento por medio de
una variable
```

```
// Array asociativo con claves string y de  valores string
  $Capitales= ['España'=>'Madrid', 'Francia'=>'Paris',
'Italia'=>'Roma', 'Alemania'=>'Berlin');
  echo $Capitales['España']; -> 'Madrid'
  echo $Capitales[0]; -> Error de índice

// Array asociativo de claves y valores  numéricos y cadena
  $Valores= ['Peso'=>65, 10=>'Diez', 'Altura'=>1.75, 20=>'Nombre');
  echo $Valores['Altura']; -> 1.75
  echo $Valores[20]; -> 'Nombre'
```

3.3.2.3 Multidimensionales

Un array multidimensional es aquel cuyos valores son otros arrays. Para acceder a sus elementos se tienen que indicar los índices de cada una de sus dimensiones, utilizando tantos pares de corchetes como dimensiones se definan en el array.

La sintaxis para definir un array de dos dimensioines son las siguientes:

```
$miArray = [
            [valor1,valor2,...],
            [valor1,valor2,...],
            …
           ];
            #  Se accede a los elementos como:
            #  [índice][índice]

$miArray = [
            [clv1=>valor1,  clv2=>valor2, ...],
            [clv1=>valor1,  clv2=>valor2, ...],
            ....
           ];
            #  Se accede a los elementos como:
            #  [índice][clave]

$miArray = [
            clv1=>[valor1, valor2,...],
            clv2=>[valor1, valor2,...],
            …
            ];
            #  Se accede a los elementos como:
            #  [clave][indice]

$miArray = [
            clv1=>[clv1=>valor1, clv2=>valor2, ...],
            clv2=>[clv1=>valor1, clv2=>valor2, ...],
            ....
            ];
            #  Se accede a los elementos como:
            #  [clave][clave]
```

Veamos los siguientes ejemplos

```
<?php
    # Multidimensional de índices escalares
    $matrizNumerica=[
        [2,4,6,8],
        [5,10,15,20],
        [10,20,30,40]
    ];
    echo $matrizNumerica[0][1]; // valor 4

    # Multidimensional de índices asociativos
    $ciudades=[
        'España'  => ['Valencia','Madrid','Barcelona'],
        'Francia' => ['Paris',  'Marsella', 'Lion'],
        'Italia' => ['Roma',  'Nápoles', 'Venecia']
    ];
    echo $ciudades['España'][1]; // Madrid

    # Multidimensional de índices escalares
    # y subíndices asociativos

    $alumnos=[
        ['Nombre'=>'Luis','Edad'=>45,'Sexo'=>'Hombre'],
        ['Nombre'=>'Carmen','Edad'=>40,'Sexo'=>'Mujer'],
        ['Nombre'=>'Pedro','Edad'=>25,'Sexo'=>'Hombre']
    ];
    echo $alumnos[2]['Edad']; // 25

    # Multidimensional de índices y subíndices asociativos
    $paises=[
        'España'  =>
['capital'=>'Madrid','idioma'=>'Español','poblacion'=>46400000],
        'Francia' =>
['capital'=>'Paris','idioma'=>'Frances','poblacion'=>66415161],
        'Italia' =>
['capital'=>'Roma','idioma'=>'Italiano','poblacion'=>59801000]];
    echo $paises['España']['idioma']; // español
?>
```

3.3.3 El constructor foreach

El constructor foreach proporciona un modo sencillo de iterar sobre arrays. *foreach* funciona sólo sobre arrays, y emitirá un error al intentar usarlo con una variable de un tipo diferente de datos o una variable no inicializada. Existen dos sintaxis:

```
foreach (expresión_array as $valor)
    sentencias
foreach (expresión_array as $clave => $valor)
    sentencias
```

La primera forma recorre el array dado por *expresión_array*. En cada iteración, el valor del elemento actual se asigna a *$valor* y el puntero interno del array avanza una posición (así en la próxima iteración se estará observando el siguiente elemento).

La segunda forma además asigna la clave del elemento actual a la variable *$clave* en cada iteración.

Ambas formas admiten la sintaxis alternativa de dos puntos (:), indicando la palabra *endforeach;* como finalización de las sentencias del bucle.

Veamos el siguiente ejemplo para la primera sintáxis

```
<?php
      $array = array(1, 2, 3, 4);
      foreach ($array as $valor) {
            echo $valor;
            echo "<BR>";
      }
?>
```

Veamos otro ejemplo para la segunda sintáxis

```
<?php
      $array = array(1, 2, 3, 4);
      foreach ($array as $key => $valor) {
            echo "Indice: ". $key . " Valor: ". $valor;
            echo "<BR>";
      }
?>
```

Veamos como el foreach recore un array bidimensional con indices y subindices asociativos

```
<?php
# Multidimensional de índices y subíndices asociativos
    $paises=[
        'España'  =>
['capital'=>'Madrid','idioma'=>'Español','poblacion'=>46400000],
        'Francia' =>
['capital'=>'Paris','idioma'=>'Frances','poblacion'=>66415161],
        'Italia' =>
['capital'=>'Roma','idioma'=>'Italiano','poblacion'=>59801000]];
    echo $paises['España']['idioma']; // español

    echo "<ul>";
    foreach ($paises as $key => $pais) {
        echo "<li>";
        echo  $key;
        echo ":";
        echo "<ul>";
        foreach ($pais as $key =>$campo) {
            echo "<li>";
```

```
            echo $key.": ".$campo;
            echo "</li>";
        }
        echo "</ul>";
        echo "</li>";
    }
    echo "</ul>";
¿>
```

El resultado sería:

- España:
  - capital: Madrid
  - idioma: Español
  - poblacion: 46400000
- Francia:
  - capital: Paris
  - idioma: Frances
  - poblacion: 66415161
- Italia:
  - capital: Roma
  - idioma: Italiano
  - poblacion: 59801000

3.3.4 Funciones para arrays

En la versión actual de PHP existen más de 70 funciones para trabajar con arrays (acceder al enlace https://www.php.net/manual/es/ref.array.php), en esta sección vamos a mostrar la lista de funciones más frecuentes.

La lista de funciones son:

- implode(): Convierte un array en una cadena de texto.
- explode(): Convierte un string en un array.
- foreach(): Función para recorrer arrays.
- count(): Cuenta el número de elementos.
- sizeof(): Alias de la función count().
- array_push(): Añade nuevos elementos.
- sort(), asort() y ksort(): Ordena los arrays con distinto creterio
- unset(): Elimina elementos.
- var_export(): Muestra el valor.
- var_dump(): Muestra el valor
- print() y print_r(): Muestra el valor.
- shuffle(): Desordena un array.
- array_merge(): Une varios arrays en uno.
- array_search(): Busca valores en un array.

- array_rand(): Devuelve una clave aleatoria.
- array_chunk(): Divide arrays en varios arrays.
- str_split(): Convierte un string en un array.
- preg_split(): Convierte un string en un array con expresiones regulares.
- array_unique: Eliminar los valores duplicados de un array
- array_keys: Devuelve en un array las claves de otro array

3.4 Principios de subprogramación

En este apartado vamos a estudiar como podemos definir y usar pequeñas funciones y/o procedimientos definidos por el usuario para incentivar la reutilización de código y la programación modular. También estudiaremos algunas de las funciones predefinidas en PHP.

3.4.1 Definición y uso

Un subprograma es un fragmento de código que tiene una funcionalidad específica. Este permite que el código sea modular y lo podamos reutilizar.

El nacimiento de los subprogramas nace de la idea de que un problema dificil resulta más fácil solucionarlo si se divide en otros más pequeños y su vez más sencillos.

Existen dos tipos de subprogramas:

- las funciones
- los procedimientos

Las **funciones** son aquellos subprogramas que devuelven un valor como resultado de su ejecución. Por otro lado los **procedimientos** son aquellos que ejecutan un conjunto de instrucciones pero sin devolver ningún tipo de valor.

Una vez que ya sabemos qué son los subprogramas, que tipos nos podemos encontrar y para qué sirven, vamos a pasar a ver su sintaxis en los distintos lenguajes.

Primero vamos a mostrar cómo se define una **función**.

✓ **PHP**:

```
function nombre($arg1,$arg2,…){
  instrucciones;
  return $valorDevuelto;
}
```

✓ **ASP**:

```
function nombre (arg1,arg2,…)
  instrucciones
  nombre=valordevuelto
end function
```

✓ **JSP**:

```
tipoDevuelto nombre (tipo1 arg1,tipo2 arg2,…){
  instrucciones;
  return valorDevuelto;
}
```

A continuación mostramos los siguientes ejemplos

✓ **PHP**:

```
function sumar ($sumando1,$sumando2){
  return $sumando1+$sumando2;
}
```

✓ **ASP**:

```
function sumar (sumando1,sumando2)
  sumar=sumando1+sumando2
end function
```

✓ **JSP**:

```
double sumar (double sumando1, double sumando2){
  return sumando1+sumando2;
}
```

Entre los distintos lenguajes podemos apreciar ciertas diferencias como por ejemplo que tanto PHP como JSP utilizan la sentencia **return** para devolver el valor resultante, mientras que ASP le asigna dicho valor al nombre de la función. Otra de las diferencias notorias es que solo en JSP se indica el tipo de dato de cada argumento que pasamos como parámetro en la función y se especiñca en la cabecera de dicha función el tipo de dato del valor devuelto.

Para finalizar esta sección vamos a ver cómo se definen los **procedimientos** en los distintos lenguajes.

✓ **PHP**:

```
function nombre($arg1,$arg2,…){
  instrucciones;
}
```

✓ **ASP**:

```
sub nombre (arg1,arg2,…)
  instrucciones
end sub
```

✓ **JSP**:

```
void nombre (tipo1 arg1,tipo2 arg2,…){
  instrucciones;
}
```

Veamos ahora los siguientes ejemplos

✓ **PHP**:

```
function mensaje ($texto){
  echo $texto;
}
```

✓ **ASP**:

```
sub mensaje (texto)
  response.write(texto)
end sub
```

✓ **JSP**:

```
void mensaje (String texto){
  out.println(texto);
}
```

Salvo ASP que tiene una sintaxis específica para definir procedimientos PHP y JSP utilizan la misma sintaxis que para definir las funciones salvo que no utilizan la sentencia return para devolver un valor. En el caso de JSP en la cabecera de la función se especifica que el tipo del valor devuelto es vacío (void).

Las funciones y procedimientos se suelen delinir en la **cabecera del documento** para asegurarnos que estén cargadas en memoria previamente antes de usarlas.

Para usar las funciones y procedimientos solamente necesitamos llamarlos por su nmnbre y pasarle los parámetros necesarios para que se puedan ejecutar.

En el caso de las funciones podemos asignar a otra variable el valor que esta devuelve para utilizarlo posteriormente.

Acontinuación mostramos la sintaxis y un ejemplo de la llamada a un procedimiento o función.

- **PHP**:

```
nombre($arg1,$arg2,...);

$texto = "Esto es una prueba.";
mensaje ($texto);
```

- **ASP**:
  - **Forma 1**:

```
nombre (arg1,arg2,...)

texto = "Esto es una prueba."
mensaje (texto)
```

  - **Forma 2** (solo procedimientos):

```
call nombre (arg1,arg2,...)

texto = "Esto es una prueba."
call mensaje |texto)
```

- **JSP**:

```
nombre (arg1,arg2,...);

texto = "Esto es una prueba.";
mensaje (texto);
```

3.4.2 Nombre de la función

Los nombres de las funciones siguen las mismas reglas que las demás etiquetas de PHP. Un nombre de función válido comienza con una letra o guion bajo, seguido de cualquier número de letras, números o guiones bajos.

Los nombres de funciones son Case-insensitive, es decir no se hace la distinción entre mayúsculas y minúsculas, aunque en la práctica siempre las escribiremos en minúsculas.

3.4.3 Argumentos

Podemos pasar cualquier información a las funciones mediante la lista de argumentos delimitadas por comas. Los argumentos son evaluados de izquierda a derecha, lo que implica que se tienen que enviar a la función en el mismo orden en el que hayan sido definidos.

También es importante tener en cuenta que los datos que le enviemos a la función, tendrán que ser del mismo tipo que espera la función recibir, es decir; si la función espera recibir dos argumentos de tipo numérico, le enviaremos dos números, si espera recibir un string y un número, le enviaremos una cadena y un número. De lo contrario el resultado puede ser inesperado a la hora de operar con los argumentos recibidos.

```
function NombreFuncion($Argumento1, $Argumento2, ...) {
  Líneas de código
}
```

3.4.3.1 Argumentos por valor y referencia

El paso de argumentos se realiza de forma predeterminada por valor, es decir se envía una copia del valor. También admite el paso de argumentos por referencia anteponiendo el símbolo '&' al nombre del argumento y listas de argumentos de longitud variable.

```
function NombreFuncion($Argumento1, &$Argumento2, ...) {
  Líneas de código
}
```

Veamos el siguiente ejemplo en el que se pasa el argumento por valor

```
function sumar_uno($x) {
   $x++;
   return $x;
}
$a = 2;
echo sumar_uno($a); // 3
echo $a; // 2
```

Ahora pasamos el argumento por referencia

```
function sumar_uno(&$x) {
    $x++;
    return $x;
}

$a = 2;
echo sumar_uno($a); // 3

// $a se pasó por referencia a suma_uno()
// El cambio dentro de la función
// se refleja en la referencia original
echo $a; // 3
```

3.4.3.2 Argumentos con valores predeterminados

Si deseamos declarar valores por defecto o predeterminados para los argumentos, los definiremos realizando la asignación del valor. Este valor será el que se utilice en el caso de no recibirse el argumento durante la llamada a la función. Cuando se emplean argumentos predeterminados, tendrán que ser declarados a la derecha de los argumentos no predeterminados, es decir; tendrán que ser siempre los últimos de la lista de argumentos de la función.

```
function NombreFuncion($Argumento1, $Argumento2=Valor,
$Argumento3=Valor,  ..) {
```

```
  Líneas de código
}
```

Observemos el siguiente ejemplo

```
<?php
      function foo($month=8,$year=2019)
      {
            return "mes $month de $year";
      }
      echo foo();           // mes 8 de 2019
      echo foo(10);         // mes 10 de 2019
      echo foo(1,1900);     // mes 1 de 1900
?>
```

3.4.3.3 Uso de arrays como argumentos predeterminados

PHP admite el uso de arrays para declarar argumentos predeterminados en nuestras funciones, lo que nos permite en un solo argumento enviar múltiples valores.

Si utilizamos arrays asociativos, donde la clave es el nombre del argumento y el valor el valor del argumento, podremos enviar los argumentos que deseemos y en cualquier orden.

Dentro de la función utilizaremos la función *array_key_exists()*, para determinar qué argumentos hemos recibido.

```
function NombreFuncion($Argumento=[]) {
  if(array_key_exists('NombreArgumento', $Argumento))
      $Arg1=$Argumento['NombreArgumento'];
      Líneas de código
}
```

Observar el siguiente ejemplo

```
function eti_html($eti=[]) {

      // Variables de trabajo para crear la etiqueta HTML
      $contenido='';
      $etiqueta='P';
      $clase='';

      // Si se reciben los argumentos esperados, asignamos a cada
      // variable su valor
      if(array_key_exists('contenido',$eti))
      $contenido=$eti['contenido'];

      if(array_key_exists('etiqueta',$eti))
      $etiqueta=$eti['etiqueta'];

      if(array_key_exists('clase',$eti))
      $clase='class="'.$eti['clase'].'"';
```

```
      // Imprime la etiqueta HTML con su contenido y la clase CSS
      echo "<$etiqueta $clase>$contenido</$etiqueta>";
}

// Imprime título de nivel 2 aplicándole las clases CSS tit-nav y
//text-center
$array['contenido']='Titulo nivel 2';
$array['etiqueta']='H2';
$array['clase']= 'tit-nav text-center';

eti_html($array);
```

3.4.4 Devolver valores

Cuando una función tiene que devolver un valor de retorno utiliza la sentencia opcional 'return' seguida del valor que desea retornar. Se puede devolver cualquier tipo de datos incluso arrays y objetos.

Esta sentencia provoca la finalización de la ejecución de las líneas de código de la función y devuelve el control a la instrucción que la referenció. Si se omite return dentro de la función el valor devuelto será siempre NULL.

```
function NombreFuncion([Argumentos]) {
    Líneas de código
    return[ValorRetorno]
}
```

3.4.5 Librerías de funciones

Si tenemos funciones que nos pueden servir para los scripts de nuestra aplicación web, lo que tenemos que hacer es agruparlas en un único archivo php, formando lo que se llama una **librería**.

Cada vez que necesitemos utilizar alguna de las funciones, sólo tendremos que incluir la librería al principio de nuestro script para poder utilizarlas. De esta manera evitaremos tener que reescribirlas cada vez que las vayamos a utilizar.

Si añadimos nuevas funciones o modificamos las características de alguna de las funciones de la librería, estos cambios estarán también presentes en cada uno de los scripts que la incluyeron en su código.

Para poder incluir un archivo de librería en nuestros scripts, tendremos utilizar alguna de las siguientes sentencias de PHP:

- Include
- include_once
- require
- require_once.

3.4.5.1 Include y require

Las sentencias *include* y *require*, incluyen en nuestro código el contenido del archivo que se le pase como argumento y lo evalúan.

```
include(RutaArchivoPHP)
require(RutaArchivoPHP)
```

Los archivos se incluirán en nuestro script con base a la ruta de acceso dada, o si no se indica ruta de acceso se buscarán en el directorio actual o en la lista de directorios en la directiva include_path de PHP.

La principal diferencia entre include y requiere, se produce en el caso de no encontrarse el archivo a incluir. Mientras que include simplemente generará un mensaje de aviso y la ejecución continua, con requiere se genera un error fatal que provoca la interrupción del código.

3.4.5.2 include_once y require_once

De funcionamiento análogo a include y require, siendo la única diferencia de que si el código del fichero ya ha sido incluido, no se volverá a incluir, e include_once y require_once devolverán TRUE. Como su nombre indica, el fichero será incluido solamente una vez.

```
<?php
      require("funciones.php");
      …
?>
```

3.5 Funciones predefinidas del lenguaje PHP

3.5.1 Funciones para string

Extraidas de php.net (https://www.php.net/manual/es/ref.strings.php)

- addcslashes — Escapa una cadena al estilo de C
- addslashes — Escapa un string con barras invertidas
- bin2hex — Convierte datos binarios en su representación hexadecimal
- chop — Alias de rtrim
- chr — Devuelve un caracter específico
- chunk_split — Divide una cadena en trozos más pequeños
- convert_cyr_string — Convierte de un juego de caracteres cirílico a otro juego de caracteres cirílico
- convert_uudecode — Descodifica una cadena codificada mediante uuencode
- convert_uuencode — Codificar mediante uuencode una cadena
- count_chars — Devuelve información sobre los caracteres usados en una cadena
- crc32 — Calcula el polinomio crc32 de una cadena
- crypt — Hash de cadenas de un sólo sentido
- echo — Muestra una o más cadenas
- explode — Divide un string en varios string
- fprintf — Escribir una cadena con formato a una secuencia

- get_html_translation_table — Devuelve la tabla de traducción utilizada por htmlspecialchars y htmlentities
- hebrev — Convierte texto hebreo lógico a texto visual
- hebrevc — Convertir texto de hebreo lógico a texto visual con conversión de línea nueva
- hex2bin — Decodifica una cadena binaria codificada hexadecimalmente
- html_entity_decode — Convierte todas las entidades HTML a sus caracteres correspondientes
- htmlentities — Convierte todos los caracteres aplicables a entidades HTML
- htmlspecialchars_decode — Convierte entidades HTML especiales de nuevo en caracteres
- htmlspecialchars — Convierte caracteres especiales en entidades HTML
- implode — Une elementos de un array en un string
- join — Alias de implode
- lcfirst — Pasa a minúscula el primer caracter de un string
- levenshtein — Cálculo de la distancia Levenshtein entre dos strings
- localeconv — Obtener información sobre el formato numérico
- ltrim — Retira espacios en blanco (u otros caracteres) del inicio de un string
- md5_file — Calcula el resumen criptográfico md5 de un archivo dado
- md5 — Calcula el 'hash' md5 de un string
- metaphone — Calcula la clave metaphone de un string
- money_format — Da formato a un número como un string de moneda
- nl_langinfo — Consulta información sobre el idioma y la configuración regional
- nl2br — Inserta saltos de línea HTML antes de todas las nuevas líneas de un string
- number_format — Formatear un número con los millares agrupados
- ord — devuelve el valor ASCII de un caracter
- parse_str — Convierte el string en variables
- print — Mostrar una cadena
- printf — Imprimir una cadena con formato
- quoted_printable_decode — Convierte un string quoted-printable en un string de 8 bits
- quoted_printable_encode — Convierte un string de 8 bits en un string quoted-printable
- quotemeta — Escapa meta caracteres
- rtrim — Retira los espacios en blanco (u otros caracteres) del final de un string
- setlocale — Establecer la información del localismo
- sha1_file — Calcula el hash sha1 de un archivo
- sha1 — Calcula el 'hash' sha1 de un string
- similar_text — Calcula la similitud entre dos strings
- soundex — Calcula la clave soundex de un string
- sprintf — Devuelve un string formateado
- sscanf — Interpreta un string de entrada de acuerdo con un formato
- str_getcsv — Convierte un string con formato CSV a un array
- str_ireplace — Versión insensible a mayúsculas y minúsculas de str_replace
- str_pad — Rellena un string hasta una longitud determinada con otro string

- str_repeat — Repite un string
- str_replace — Reemplaza todas las apariciones del string buscado con el string de reemplazo
- str_rot13 — Realizar la transformación rot13 sobre una cadena
- str_shuffle — Reordena aleatoriamente una cadena
- str_split — Convierte un string en un array
- str_word_count — Devuelve información sobre las palabras utilizadas en un string
- strcasecmp — Comparación de string segura a nivel binario e insensible a mayúsculas y minúsculas
- strchr — Alias de strstr
- strcmp — Comparación de string segura a nivel binario
- strcoll — Comparación de cadenas basada en la localidad
- strcspn — Averiguar la longitud del segmento inicial que no coincida con una máscara
- strip_tags — Retira las etiquetas HTML y PHP de un string
- stripcslashes — Desmarca la cadena marcada con addcslashes
- stripos — Encuentra la posición de la primera aparición de un substring en un string sin considerar mayúsculas ni minúsculas
- stripslashes — Quita las barras de un string con comillas escapadas
- stristr — strstr insensible a mayúsculas y minúsculas
- strlen — Obtiene la longitud de un string
- strnatcasecmp — Comparación de strings, insensible a mayúsculas y minúsculas, utilizando un algoritmo de "orden natural"
- strnatcmp — Comparación de strings utilizando un algoritmo de "orden natural"
- strncasecmp — Comparación de los primeros n caracteres de cadenas, segura con material binario e insensible a mayúsculas y minúsculas
- strncmp — Comparación segura a nivel binario de los primeros n caracteres entre strings
- strpbrk — Buscar una cadena por cualquiera de los elementos de un conjunto de caracteres
- strpos — Encuentra la posición de la primera ocurrencia de un substring en un string
- strrchr — Encuentra la última aparición de un caracter en un string
- strrev — Invierte una string
- strripos — Encuentra la posición de la última aparición de un substring insensible a mayúsculas y minúsculas en un string
- strrpos — Encuentra la posición de la última aparición de un substring en un string
- strspn — Averigua la longitud del segmento inicial de un string que consista únicamente en caracteres contenidos dentro de una máscara dada
- strstr — Encuentra la primera aparición de un string
- strtok — Tokeniza string
- strtolower — Convierte una cadena a minúsculas
- strtoupper — Convierte un string a mayúsculas
- strtr — Convierte caracteres o reemplaza substrings
- substr_compare — Comparación segura a nivel binario de dos o más strings desde un índice hasta una longitud de caracteres dada
- substr_count — Cuenta el número de apariciones del substring

- substr_replace — Reemplaza el texto dentro de una porción de un string
- substr — Devuelve parte de una cadena
- trim — Elimina espacio en blanco (u otro tipo de caracteres) del inicio y el final de la cadena
- ucfirst — Convierte el primer caracter de una cadena a mayúsculas
- ucwords — Convierte a mayúsculas el primer caracter de cada palabra de una cadena
- vfprintf — Escribe un string con formato en un flujo
- vprintf — Muestra una cadena con formato
- vsprintf — Devuelve una cadena con formato
- wordwrap — Ajusta un string hasta un número dado de caracteres

3.5.2 Funciones para array

Extraidas de php.net (https://www.php.net/manual/es/ref.array.php)

- array_change_key_case — Cambia a mayúsculas o minúsculas todas las claves en un array
- array_chunk — Divide un array en fragmentos
- array_column — Devuelve los valores de una sola columna del array de entrada
- array_combine — Crea un nuevo array, usando una matriz para las claves y otra para sus valores
- array_count_values — Cuenta todos los valores de un array
- array_diff_assoc — Calcula la diferencia entre arrays con un chequeo adicional de índices
- array_diff_key — Calcula la diferencia entre arrays empleando las claves para la comparación
- array_diff_uassoc — Calcula la diferencia entre arrays con un chequeo adicional de índices que se realiza por una función de devolución de llamada suministrada por el usuario
- array_diff_ukey — Calcula la diferencia entre arrays usando una función de devolución de llamada en las keys para comparación
- array_diff — Calcula la diferencia entre arrays
- array_fill_keys — Llena un array con valores, especificando las keys
- array_fill — Llena un array con valores
- array_filter — Filtra elementos de un array usando una función de devolución de llamada
- array_flip — Intercambia todas las claves de un array con sus valores asociados
- array_intersect_assoc — Calcula la intersección de arrays con un chequeo adicional de índices
- array_intersect_key — Calcula la intersección de arrays usando sus claves para la comparación
- array_intersect_uassoc — Calcula la intersección de arrays con una comprobación adicional de índices, los cuales se comparan con una función de retrollamada
- array_intersect_ukey — Calcula la intersección de arrays usando una función de devolución de llamada en las claves para la comparación
- array_intersect — Calcula la intersección de arrays
- array_key_exists — Verifica si el índice o clave dada existe en el array

- array_key_first — Gets the first key of an array
- array_key_last — Gets the last key of an array
- array_keys — Devuelve todas las claves de un array o un subconjunto de claves de un array
- array_map — Aplica la retrollamada a los elementos de los arrays dados
- array_merge_recursive — Une dos o más arrays recursivamente
- array_merge — Combina dos o más arrays
- array_multisort — Ordena varios arrays, o arrays multidimensionales
- array_pad — Rellena un array a la longitud especificada con un valor
- array_pop — Extrae el último elemento del final del array
- array_product — Calcula el producto de los valores de un array
- array_push — Inserta uno o más elementos al final de un array
- array_rand — Seleccionar una o más entradas aleatorias de un array
- array_reduce — Reduce iterativamente un array a un solo valor usando una función llamada de retorno
- array_replace_recursive — Reemplaza los elementos de los arrays pasados al primer array de forma recursiva
- array_replace — Reemplaza los elementos del array original con elementos de array adicionales
- array_reverse — Devuelve un array con los elementos en orden inverso
- array_search — Busca un valor determinado en un array y devuelve la primera clave correspondiente en caso de éxito
- array_shift — Quita un elemento del principio del array
- array_slice — Extraer una parte de un array
- array_splice — Elimina una porción del array y la reemplaza con otra cosa
- array_sum — Calcular la suma de los valores de un array
- array_udiff_assoc — Computa la diferencia entre arrays con una comprobación de indices adicional, compara la información mediante una función de llamada de retorno
- array_udiff_uassoc — Computa la diferencia entre arrays con una verificación de índices adicional, compara la información y los índices mediante una función de llamada de retorno
- array_udiff — Computa la diferencia entre arrays, usando una llamada de retorno para la comparación de datos
- array_uintersect_assoc — Calcula la intersección de arrays con una comprobación de índices adicional, compara la información mediante una función de retrollamada
- array_uintersect_uassoc — Calcula la intersección de arrays con una comprobación de índices adicional, compara la información y los índices mediante funciones de retrollamada por separado
- array_uintersect — Computa una intersección de arrays, compara la información mediante una función de llamada de retorno
- array_unique — Elimina valores duplicados de un array
- array_unshift — Añadir al inicio de un array uno a más elementos
- array_values — Devuelve todos los valores de un array
- array_walk_recursive — Aplicar una función de usuario recursivamente a cada miembro de un array

- array_walk — Aplicar una función proporcionada por el usuario a cada miembro de un array
- array — Crea un array
- arsort — Ordena un array en orden inverso y mantiene la asociación de índices
- asort — Ordena un array y mantiene la asociación de índices
- compact — Crear un array que contiene variables y sus valores
- count — Cuenta todos los elementos de un array o algo de un objeto
- current — Devuelve el elemento actual en un array
- each — Devolver el par clave/valor actual de un array y avanzar el cursor del array
- end — Establece el puntero interno de un array a su último elemento
- extract — Importar variables a la tabla de símbolos actual desde un array
- in_array — Comprueba si un valor existe en un array
- key_exists — Alias de array_key_exists
- key — Obtiene una clave de un array
- krsort — Ordena un array por clave en orden inverso
- ksort — Ordena un array por clave
- list — Asignar variables como si fueran un array
- natcasesort — Ordenar un array usando un algoritmo de "orden natural" insensible a mayúsculas-minúsculas
- natsort — Ordena un array usando un algoritmo de "orden natural"
- next — Avanza el puntero interno de un array
- pos — Alias de current
- prev — Rebobina el puntero interno del array
- range — Crear un array que contiene un rango de elementos
- reset — Establece el puntero interno de un array a su primer elemento
- rsort — Ordena un array en orden inverso
- shuffle — Mezcla un array
- sizeof — Alias de count
- sort — Ordena un array
- uasort — Ordena un array con una función de comparación definida por el usuario y mantiene la asociación de índices
- uksort — Ordena un array según sus claves usando una función de comparación definida por el usuario
- usort — Ordena un array según sus valores usando una función de comparación definida por el usuario

3.5.3 Funciones fecha hora

Extraidas de php.net (https://www.php.net/manual/es/ref.datetime.php)

- checkdate — Validar una fecha gregoriana
- date_add — Alias de DateTime::add
- date_create_from_format — Alias de DateTime::createFromFormat
- date_create_immutable_from_format — Alias de DateTimeImmutable::createFromFormat
- date_create_immutable — Alias de DateTimeImmutable::__construct
- date_create — Alias de DateTime::__construct

- date_date_set — Alias de DateTime::setDate
- date_default_timezone_get — Obtiene la zona horaria predeterminada usada por todas las funciones de fecha/hora en un script
- date_default_timezone_set — Establece la zona horaria predeterminada usada por todas las funciones de fecha/hora en un script
- date_diff — Alias de DateTime::diff
- date_format — Alias de DateTime::format
- date_get_last_errors — Alias de DateTime::getLastErrors
- date_interval_create_from_date_string — Alias de DateInterval::createFromDateString
- date_interval_format — Alias de DateInterval::format
- date_isodate_set — Alias de DateTime::setISODate
- date_modify — Alias de DateTime::modify
- date_offset_get — Alias de DateTime::getOffset
- date_parse_from_format — Obtiene información de una fecha dada formateada de acuerdo al formato especificado
- date_parse — Devuelve un array asociativo con información detallada acerca de una fecha dada
- date_sub — Alias de DateTime::sub
- date_sun_info — Devuelve una matriz con información sobre la puesta/salida del sol y el comienzo/final del crepúsculo
- date_sunrise — Devuelve la hora de la salida del sol de un día y ubicación dados
- date_sunset — Devuelve la hora de la puesta de sol de un día y ubicación dados
- date_time_set — Alias de DateTime::setTime
- date_timestamp_get — Alias de DateTime::getTimestamp
- date_timestamp_set — Alias de DateTime::setTimestamp
- date_timezone_get — Alias de DateTime::getTimezone
- date_timezone_set — Alias de DateTime::setTimezone
- date — Dar formato a la fecha/hora local
- getdate — Obtener información de la fecha/hora
- gettimeofday — Obtener la hora actual
- gmdate — Formatea una fecha/hora GMT/UTC
- gmmktime — Obtener la marca temporal de Unix para una fecha GMT
- gmstrftime — Formatear una fecha/hora GMT/UTC según la configuración local
- idate — Formatea una fecha/hora local como un entero
- localtime — Obtiene fecha y hora local
- microtime — Devuelve la fecha Unix actual con microsegundos
- mktime — Obtener la marca de tiempo Unix de una fecha
- strftime — Formatea una fecha/hora local según una configuración local
- strptime — Analiza una fecha/hora generada con strftime
- strtotime — Convierte una descripción de fecha/hora textual en Inglés a una fecha Unix
- time — Devuelve la fecha Unix actual
- timezone_abbreviations_list — Alias de DateTimeZone::listAbbreviations
- timezone_identifiers_list — Alias de DateTimeZone::listIdentifiers

- timezone_location_get — Alias de DateTimeZone::getLocation
- timezone_name_from_abbr — Devuelve el nombre de la zona horaria desde su abreviatura
- timezone_name_get — Alias de DateTimeZone::getName
- timezone_offset_get — Alias de DateTimeZone::getOffset
- timezone_open — Alias de DateTimeZone::__construct
- timezone_transitions_get — Alias de DateTimeZone::getTransitions
- timezone_version_get — Obtiene la versión de la base de datos timezonedb

3.5.4 Funciones matemáticas

Extraidas de php.net (https://www.php.net/manual/es/ref.math.php)

- abs — Valor absoluto
- acos — Arco coseno
- acosh — Arco coseno hiperbólico
- asin — Arco seno
- asinh — Arco seno hiperbólico
- atan2 — Arco tangente de dos variables
- atan — Arco tangente
- atanh — Arco tangente hiperbólica
- base_convert — Convertir un número entre bases arbitrarias
- bindec — Binario a decimal
- ceil — Redondear fracciones hacia arriba
- cos — Coseno
- cosh — Coseno hiperbólico
- decbin — Decimal a binario
- dechex — Decimal a hexadecimal
- decoct — Decimal a octal
- deg2rad — Convierte el número en grados a su equivalente en radianes
- exp — Calcula la exponencial de e
- expm1 — Devuelve exp(numero)-1, calculado de tal forma que no pierde precisión incluso cuando el valor del numero se aproxima a cero.
- floor — Redondear fracciones hacia abajo
- fmod — Devuelve el resto en punto flotante (módulo) de la división de los argumentos
- getrandmax — Mostrar el mayor valor aleatorio posible
- hexdec — Hexadecimal a decimal
- hypot — Calcula la longitud de la hipotenusa de un triángulo de ángulo recto
- intdiv — División entera
- is_finite — Encuentra si un valor es un número finito legal
- is_infinite — Encuentra si un valor es infinito
- is_nan — Encuentra si un valor no es un número
- lcg_value — Generador lineal congruente combinado
- log10 — Logaritmo en base 10
- log1p — Devuelve log(1 + numero), calculado de tal forma que no pierde precisión incluso cuando el valor del numero se aproxima a cero.
- log — Logaritmo natural

- max — Encontrar el valor más alto
- min — Encontrar el valor más bajo
- mt_getrandmax — Mostrar el mayor valor aleatorio posible
- mt_rand — Genera un mejor número entero aleatorio
- mt_srand — Genera el mejor número aleatorio a partir de una semilla
- octdec — Octal a decimal
- pi — Obtener valor de pi
- pow — Expresión exponencial
- rad2deg — Convierte el número en radianes a su equivalente en grados
- rand — Genera un número entero aleatorio
- round — Redondea un float
- sin — Seno
- sinh — Seno hiperbólico
- sqrt — Raíz cuadrada
- srand — Genera un número aleatorio a partir de una semilla
- tan — Tangente
- tanh — Tangente hiperbólica

3.6 Acceso a la información del cliente web

En este apartado vamos a aprender los distintos tipos de paso de parámetros, sus diferencias, como definir un formulario y como recuperar los datos enviados por el método **POST** y **GET**.

3.6.1 Método GET y POST

Tanto el método **GET** como el método **POST** no son más que métodos del protocolo **HTTP** para el intercambio de información entre el cliente y el servidor.

3.6.1.1 Método GET

El método GET envía la información codificada del usuario en el header del HTTP request, directamente en la URL. La página web y la información codificada se separan por un interrogante ?:

```
www.ejemplo.com/index.htm?key1=value1&key2=value2&key3=value3...
```

- El método GET envía la información en la propia URL, estando limitada a 2000 caracteres.
- La información es visible por lo que con este método nunca se envía información sensible.
- No se pueden enviar datos binarios (archivos, imágenes...).
- En PHP los datos se administran con el array asociativo $_GET.

El método **GET** le “pide” al servidor web que le devuelve al cliente la información identiticada en la petición URI (identificador de recursos uniformes). Lo más común es que las peticiones URI se refieran a un documento HTML o a una imagen, aunque también se puede referir a una consulta de una base de datos. En tal caso, el servidor procesa la petición y le devuelve al cliente el resultado generado.

Mientras que el método **GET** lo utilizamos para recuperar información, el método **POST** se usa para enviar informacion a un servidor web. Estos casos de “posting" pueden ser utlizados para completar un formulario de autenticación, asi como entradas de datos o especificar parámetros para algún tipo de software del servidor.

Para entender cómo se envian las variables en el método GET es necesario que entendamos las partes de una URL. La URL está formada por:

- **Protocolo**. Especifica el protocolo de comunicación que se utiliza para el intercambio de la información.
- **Nombre de dominio**. Nombre del servido donde se aloja la información.
- **Directorios**. Secuencia de directorios separados por “/” que indica laruta en la que se encuentra el recurso.
- **Fichero**. Nombre del recurso o fichero al que queremos acceder.

En la Figura 3.16 podemos ver las partes que forman una URL:

Figura 3.16. Formato URL

Unavez que sabemos cómo es la estructura de una URL es necesario conocer que al final de esta podemos concatenar el símbolo de cierre de interrogación “?”. Este simbolo se utiliza para indicar que se van a definir a continuación variables con el valor que toman *(variable=valor*). Las distintas variables se separan entre ellas con el símbolo “&”. A continuación vemos un ejemplo de cómo viajan las variables en las peticiones realizadas con el método GET:

```
http://www.example.org/file/example1.php?v1=0&v2=3
```

3.6.1.2 Método POST

Características de este método:

- El método POST no tiene límite de cantidad de información a enviar.
- La información proporcionada no es visible, por lo que se puede enviar información sensible.
- Se puede usar para enviar texto normal así como datos binarios (archivos, imágenes...).
- PHP proporciona el array asociativo $_POST para acceder a la información enviada.

Por otro lado, en el método POST la información va codificada en el cuerpo de la petición HTTP y, por tanto, viaja oculta. Por este motivo hay que utilizar este método cuando queremos ingresar datos en un formulario, ya sea para realizar la autenticación de un usuario o la inserción y/o actualización en una base de datos:

Si utilizáramos el método GET para realizar este tipo de acciones cualquier persona con mínimos conocimientos de informática podría manipular la URL, o pasarle a otra persona la URL para que acceda a una página para la que no debería tener permiso.

Esto no quiere decir que el método GET sea más inseguro que el método POST ya que este puede ser vulnerado con ataques de tipo **HTML injection** que es un ataque que permite ejecutar código scripting debido a las vulnerabilidades del sistema de validación del método POST. Estos ataques afectan al servidor que contienen la información.

3.6.1.3 Conclusión

Como conclusión decir que no hay un método más seguro que otro. Como podemos ver ambos tienen sus pros y sus contras, pero resulta recomendable que cada uno de ellos los utilicemos para la labor para la que fueron creados:

- Recuperación (GET) de información
- Envío (POST) de información

Las llamadas a través del método GET pueden ser cacheadas, indexadas por los buscadores e incluso se pueden almacenar en el navegador como favoritos. Esto a veces resulta útil, como por ejemplo cuando rellenamos un formulario para realizar algún tipo de búsqueda y nos devuelve la página con la información permitiéndonos volver atrás utilizando el boton del navegador, mientras que el método POST esto no lo permite.

3.6.2 Recuperación información con GET

Una vez que hemos enviado el formulario al servidor, este tiene que recuperar la información para poder prucesarla.

En PHP quizás esto resulte un poco más fácil que en el resto de lenguajes que estamos viendo, porque en el caso del envio de información utilizando el método GET existe una variable especial $_GET, donde se almacenan todas las variables pasadas con este método.

La forma en que lo almacena es sencilla, pues esta variable no es más que un array en el que el índice es el nombre asignado al elemento del formulario con el atributo **name** y dentro almacena el valor introducido por el usuario en el formulario.

Ejemplo de formulario utilizando el método GET:

```
<html>
  <head>
    <title>Ejemplo Formularios</title>
  </head>
  <body>
    <h1>Ejemplo de procesado de formularios</h1>
    <FORM ACTION="ejemplo1.php" METHOD="GET">
      Introduzca su nombre:<INPUT TYPE="text"
        NAME="nombre"><BR>
      Introduzca sus apellidos:<INPUT TYPE="text"
        NAME="apellidos"><BR>
      <INPUT TYPE="submit" VALUE="Enviar">
    </FORM>
  </body>
</html>
```

En el ejemplo del formulario anterior si queremos acceder al nombre y apellidos que introduce el usuario solo es necesario realizar lo siguiente:

- **Forma 1**:

```
echo $_GET['nombre'];
echo $_GET['apellidos'];
```

- **Forma 2**:

```
print_r($_GET);
```

3.6.3 Recuperación de la información con POST

Al igual que en el envio de formularios utilizando el método GET, para recuperar la información enviada al servidor utilizando el método POST en PHP se utiliza la variable $_POST.

Esta variable al igual que $_GET es un array y almacena la información de la misma forma, utilizando como índice el nombre (name) del elemento del formulario y almacenando en su interior el valor que toma.

Como hemos visto en apartados anteriores lo que cambia entre el método GET y POST es como se envia la información o más bien en que parte del mensaje HTTP que le envia el cliente a el servidor viaja.

```
<html>
  <head>
    <title>Ejemplo Formularios</title>
  </head>
  <body>
    <h1>Ejemplo de procesado de formularios</h1>
    <FORM ACTION="ejemplo2.php" METHOD="POST">
      Introduzca su nombre:<INPUT TYPE="text"
        NAME="nombre"><BR>
      Introduzca sus apellidos:<INPUT TYPE="text"
        NAME="apellidos"><BR>
      <INPUT TYPE="submit" VALUE="Enviar">
    </FORM>
  </body>
</html>
```

Utilizando el ejemplo del formulario anterior vamos aver cómo se recuperaría la información enviada por el método POST:

```
echo $_POST['nombre'];
echo $_POST['apellidos'];

print_r($_POST);
```

Finalmente decir que en PHP existe la variable $_REQUEST que contiene tanto $_POST como el de $_GET. Esta variable por tanto también se puede utilizar para recuperar la información enviada por ambos métodos.