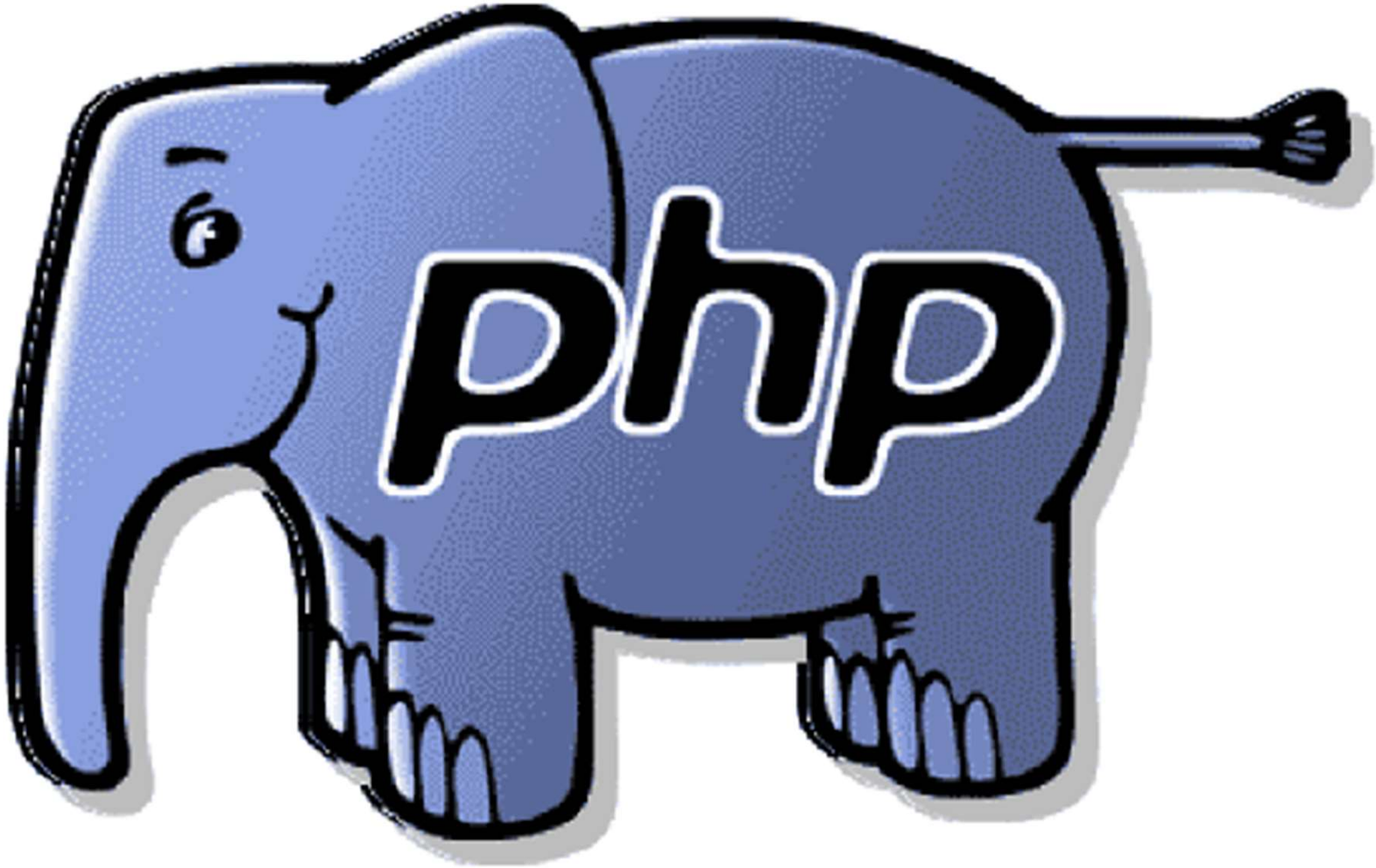
Programación basada en lenguajes de marcas con código embebido



Desarrollo Web en Entorno Servidor

Contenido

1. Estructuras de control

En PHP los guiones se construyen a base de **sentencias**.

Para definir el flujo de ejecución de un programa hay sentencias para dos tipos de **estructuras de control**:

- **Sentencias condicionales:** permiten definir las condiciones bajo las que debe ejecutarse una sentencia o un bloque de sentencias.
- **Sentencias de bucle**: con ellas se puede definir si una sentencia o conjunto de sentencias se repite o no, y bajo qué condiciones.

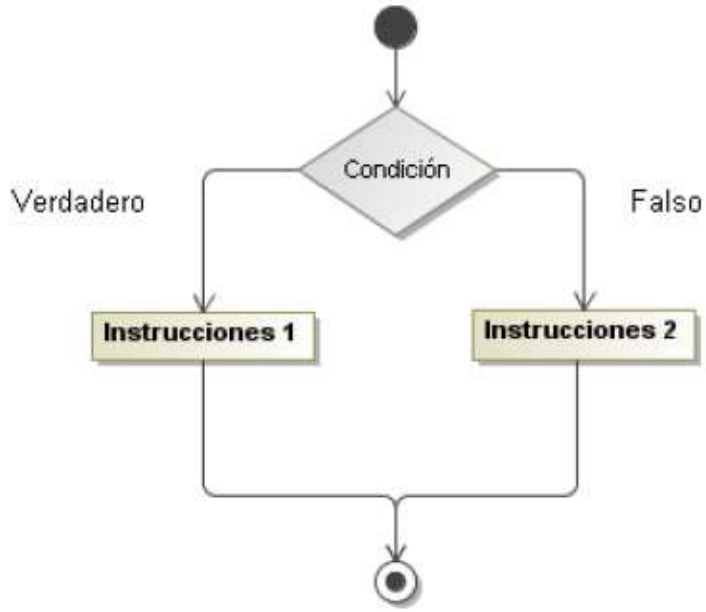
➢ **Sentencias condicionales:** permiten decidir el flujo de ejecución de un programa, es decir, el orden en el que las instrucciones de un programa se van a ejecutar.

Tipos:

- Sentencias **IF**
- Sentencias **SWITCH**

1. Estructuras de control

- **Sentencias IF:** definen dos flujos de ejecución dependiendo de si se cumple o no la condición establecida por el programador.



Sintaxis **if**:

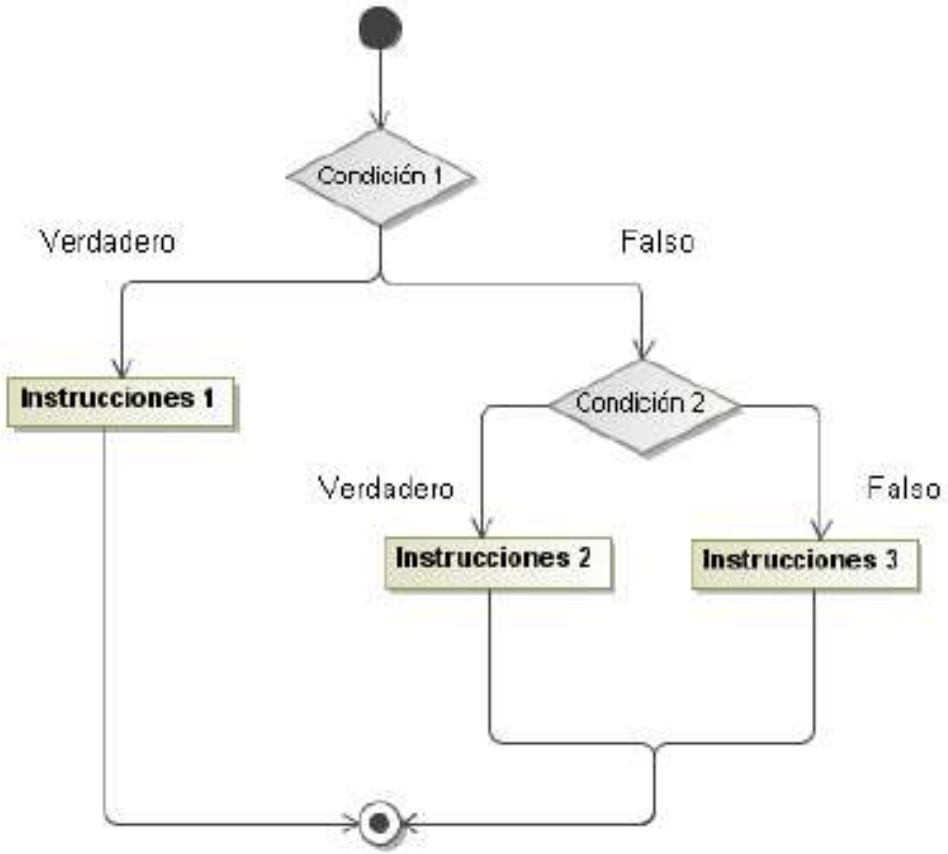
```
if (condición){
        instrucciones;
}
```

Sintaxis **if-else**:

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

1. Estructuras de control

Los **if anidados** permiten evaluar varias condiciones previas antes de ejecutar las instrucciones correspondientes.

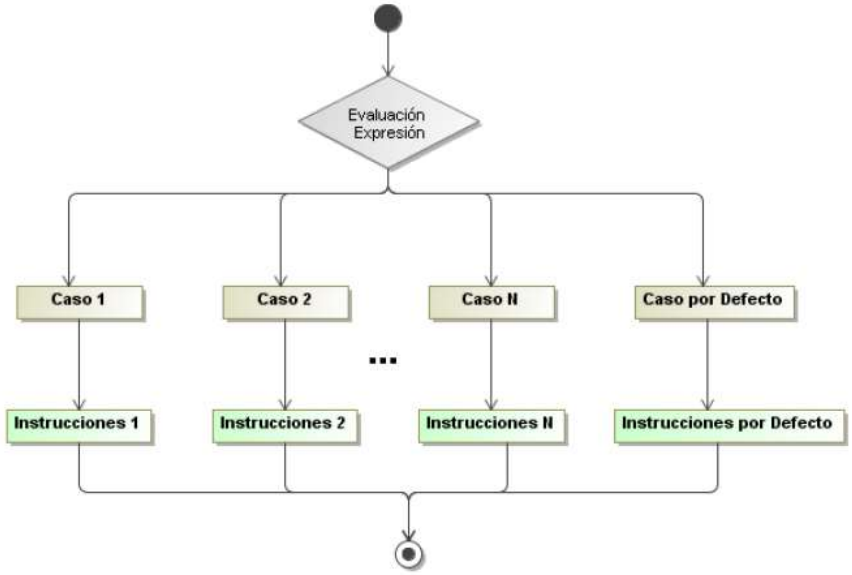


Sintaxis **if-elseif-else**:

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

1. Estructuras de control

- **Sentencias SWITCH:** se usan cuando dependiendo del valor que toma una variable o expresión, se necesita que se ejecute un conjunto de instrucciones distintas para cada uno de los valores que pueda tomar.
  1. Se calcula el valor de la expresión.
  2. Se compara dicho valor con cada uno de los casos.
     1. Si coincide con un caso se ejecutan las instrucciones contenidas dentro del mismo.
     2. Si no coincide con ningún caso se ejecutan las instrucciones definidas en el caso por defecto si es que está definido (el caso por defecto es opcional).



1. Estructuras de control

Sintaxis **switch**:

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

1. Estructuras de control

Ejemplo switch:

```
<?php
    print "<b><u>Sentencia switch</u></b><br>";
    $i=2;
    switch ($i) {
        case 0:
            print "i es igual a 0<br>";
        case 1:
            print "i es igual a 0 o 1<br>";
        case 2:
            print "i es igual a 0, 1 o 2<br>";
        default:
            print "Sólo un break impediría que se imprima esta línea<br>";
    }
?>
```

1. Estructuras de control

Ejemplo switch:

```
<?php
	print "<b><u>Ejemplo 2 switch</u></b><br>";
	// Este switch es similar a un if/elseif
	$i=2;
	switch ($i) {
		case 0:
			print "i es igual a 0<br>";
			break;
		case 1:
			print "i es igual a 1<br>";
			break;
		default:
			print "Sólo un break impediría que se imprima esta línea<br>";
	}
?>
```

1. Estructuras de control

Ejemplo if elseif else:

```
<?php
        print "<br>Ejemplo if identico a switch<br>";
        // Este if/elseif es idéntico funcionalmente al switch anterior
        $i=2;
        if ($i==0) {
                print "i es igual a 0<br>";
        }
        elseif ($i==1) {
                print "i es igual a 1<br>";
        }
        else {
                print "Ambas expresiones fueron falsas<br>";
        }
        print "Fin del ejemplo<br>";
?>
```

1. Estructuras de control

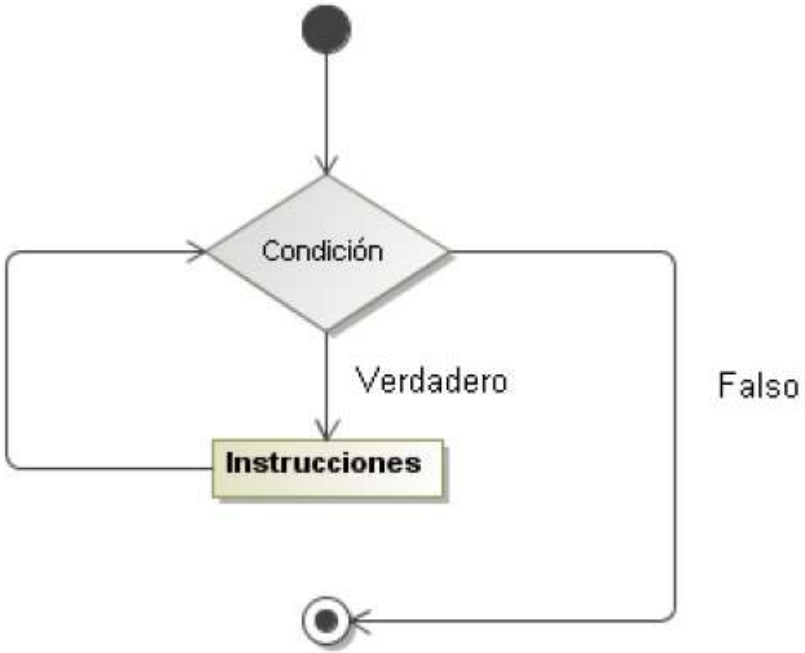
- **Bucles:** se utilizan para ejecutar de forma reiterativa una instrucción o grupo de instrucciones.

  Tipos:

  - **WHILE**
  - **DO/WHILE**
  - **FOR**
  - **FOREACH**

1. Estructuras de control

- **WHILE:** permite ejecutar un número indeterminado de veces una instrucción o grupo de instrucciones, mientras se cumpla la condición. La expresión se evalúa antes de comenzar cada ejecución del bucle.



Sintaxis **while**:

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

1. Estructuras de control

- **DO/WHILE:** este bucle se ejecuta un número indeterminado de veces hasta que el resultado de evaluar la condición es falsa. La expresión se evalúa al final, con lo cual se asegura que la sentencia o sentencias del bucle se ejecutan al menos una vez.

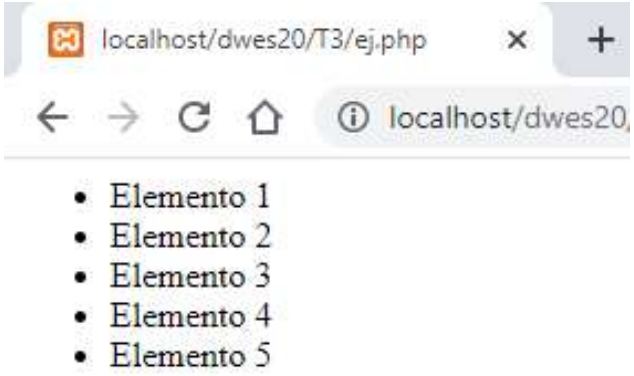
Sintaxis **do/while**:

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

1. Estructuras de control

Ejemplo while:

```
<?php
        print ("<ul>\n");
        $i=1;
        while ($i <= 5)
        {
                print "<li>Elemento $i</li>\n";
                $i++;
        }
        print ("</ul>\n");
?>
```



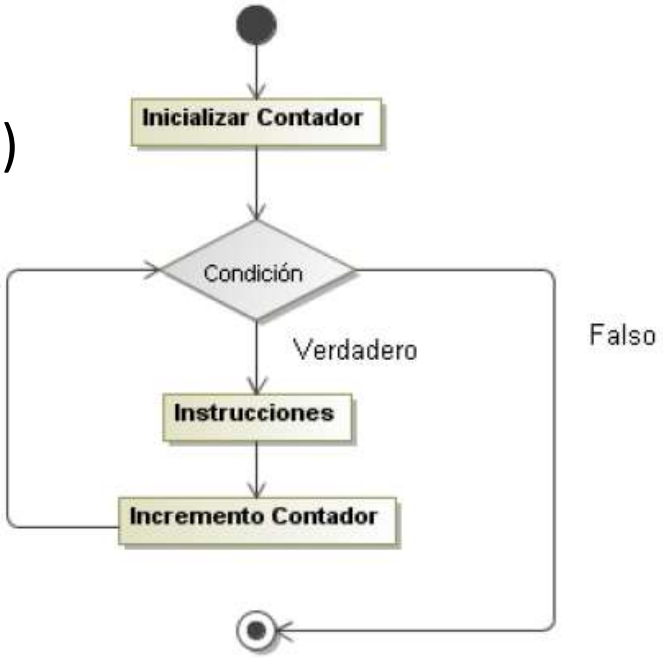
1. Estructuras de control

- **FOR:** permite ejecutar un conjunto de instrucciones un número determinado de veces. La expresión se evalúa antes de comenzar cada ejecución del bucle.

  1. Se inicializa el contador, que controla el número de veces que se ejecuta el bucle.
  2. Se evalúa la condición:
     - Si es verdadera se ejecuta el contenido del bucle y se actualiza el contador.
     - Si es falsa, el bucle finaliza siguiendo con la ejecución de las instrucciones siguientes al bucle.

Sintaxis **for**:

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



1. Estructuras de control

- **FOREACH:** proporciona una forma sencilla de iterar sobre los elementos de un array o matriz.

  Sintaxis **foreach**:

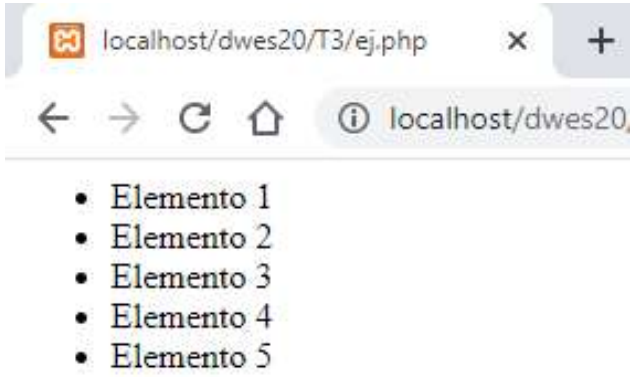
```
foreach(variableArray as variableValor){
        instrucciones;
}
```

```
foreach(variableArray as variableIndice => variableValor){
        instrucciones;
}
```

1. Estructuras de control

Ejemplo for:

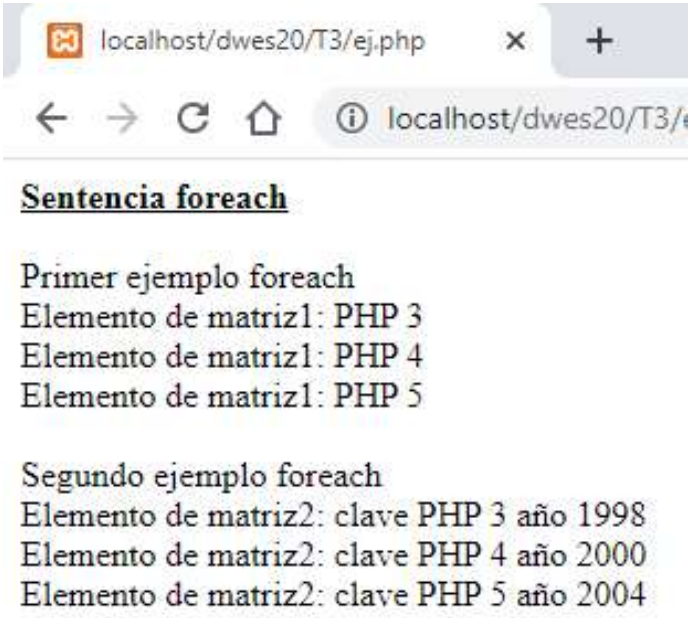
```
<?php
        print ("<ul>\n");
        for ($i=1; $i<=5; $i++)
                print "<li>Elemento $i</li>\n";
        print ("</ul>\n");
?>
```



1. Estructuras de control

Ejemplo foreach:

```
<?php
        print "<b><u>Sentencia foreach</u></b><br>";
        print "<br>Primer ejemplo foreach<br>";
        $matriz1 = array("PHP 3", "PHP 4", "PHP 5");
        foreach ($matriz1 as $var) {
                print "Elemento de matriz1: $var<br>";
        }
        print "<br>Segundo ejemplo foreach<br>";
        $matriz2["PHP 3"] = 1998;
        $matriz2["PHP 4"] = 2000;
        $matriz2["PHP 5"] = 2004;
        foreach ($matriz2 as $clave => $var1) {
                print "Elemento de matriz2: clave $clave año $var1<br>";
        }
?>
```



1. Estructuras de control

- ✓ Sentencia **Break:** se utiliza para interrumpir la ejecución de un bucle.

  Sintaxis **break**:

  break [número];

- ✓ Sentencia **Continue:** se utiliza para saltar el resto de la iteración de un bucle y continuar con la evaluación de la condición y la siguiente iteración.

  Sintaxis **continue**:

  continue[número];

2. Funciones

Un **subprograma** es un fragmento de código que tiene una funcionalidad específica. Permiten asociar una etiqueta a un bloque de código a ejecutar. Esto permite que el código sea modular y lo podamos reutilizar.

Tipos de subprogramas:

- **Funciones**: devuelven un valor como resultado de su ejecución.
- **Procedimientos**: se ejecutan sin devolver ningún tipo de valor.

Hay gran cantidad de funciones disponibles en PHP:

- incluidas en el núcleo de PHP, y se pueden utilizar directamente; ó
- se encuentran disponibles en forma de extensiones, y se pueden incorporar cuando se necesiten.

2. Funciones

- **Inclusión de ficheros externos**

En ocasiones resulta útil agrupar grupos de funciones o bloques de código, y ponerlos en un fichero aparte. Posteriormente, se puede hacer referencia a ellos para que PHP incluya su contenido como parte del programa actual.

- **include:** Evalúa el contenido del fichero que se indica y lo incluye como parte del fichero actual en el mismo punto en que se realiza la llamada. La ubicación del fichero puede especificarse con una ruta absoluta o una ruta relativa.
- **include_once:** Si por equivocación se incluye más de una vez un mismo fichero, lo normal es que se obtenga algún tipo de error (p.ej., al repetir una definición de una función). include_once funciona exactamente igual que include, pero solo incluye aquellos ficheros que aún no se hayan incluido.
- **require:** Si el fichero que queremos incluir no se encuentra, include da un aviso y continua la ejecución del guión. Al usar require, en ese caso, cuando no se puede incluir el fichero se detiene la ejecución del guión.
- **require_once**. Es la combinación de las dos anteriores. Asegura la inclusión del fichero indicado solo una vez, y genera un error si no se puede llevar a cabo.

2. Funciones

Cuando se evalúa el fichero externo, se abandona de forma automática el modo PHP y su contenido se trata como etiquetas HTML. Por este motivo, es necesario delimitar el código PHP que contenga nuestro archivo externo utilizando dentro del mismo los delimitadores <?php y ?>.

Ejemplo:

definiciones.php

```
<?php
   $modulo = 'DWES';
   $ciclo = 'DAW';
?>
```

programa.php

```
<?php
   print "Módulo $modulo del ciclo $ciclo<br />"; //Solo muestra "Modulo del ciclo"
   include 'definiciones.php';
   print " Módulo $modulo del ciclo $ciclo<br />"; // muestra "Modulo DWES del ciclo DAW"
?>
```

2. Funciones

- **Creación y ejecución de funciones**

Sintaxis de creación de funciones:

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

Sintaxis de creación de procedimientos:

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

Sintaxis de invocación de funciones y procedimientos:

```
nombre($arg1,$arg2,…);
```

2. Funciones

Ejemplo:

```
<?php
    function precio_con_iva() {
        global $precio;
        $precio_iva = $precio * 1.21;
        print "El precio con IVA es ".$precio_iva;
    }
    $precio = 10;
    precio_con_iva();
?>
```

2. Funciones

- **Argumentos**

Es mejor utilizar argumentos o parámetros que utilizar variables globales, en la llamada a una función. Las **funciones** devuelven un **valor** usando la sentencia **return**. Los **procedimientos** devuelven **null** al finalizar su procesamiento.

Ejemplo: utilización de argumentos.

```
<?php
    function precio_con_iva($precio) {
        return $precio * 1.21;
    }
    $precio = 10;
    $precio_iva = precio_con_iva($precio);
    print "El precio con IVA es
    ".$precio_iva;
?>
```

Ejemplo: valores por defecto en argumentos.

```
<?php
    function precio_con_iva($precio,
    $iva=0.21) {
        return $precio * (1+ $iva);
    }
    $precio = 10;
    $precio_iva = precio_con_iva($precio);
    print "El precio con IVA es
    ".$precio_iva;
?>
```

Si al llamar a la función no se usa el argumento, se toma el valor por defecto.

2. Funciones

En los ejemplos anteriores los argumentos se pasaban **por valor**. Esto es, cualquier cambio que se haga dentro de la función a los valores de los argumentos no se reflejará fuera de la función.

Si se quiere que esto ocurra se debe definir el parámetro para que su valor se pase **por referencia**, añadiendo el símbolo **&** antes de su nombre.

Ejemplo: paso de parámetros por referencia.

```
<?php
    function precio_con_iva(&$precio, $iva=0.21) {
    $precio *= (1 + $iva);
}
$precio = 10;
precio_con_iva($precio);
print "El precio con IVA es ".$precio;
?>
```

3. Tipos de datos compuestos

Un tipo de datos compuesto es aquel que permite almacenar más de un valor. En PHP hay dos tipos de datos compuestos: el **array** y el **objeto**.

Un **array** es un tipo de datos que nos permite almacenar varios valores. Cada miembro del **array** se almacena en una posición a la que se hace referencia utilizando un valor clave. Las claves pueden ser **numéricas** o **asociativas**.

Sintaxis (numérico): nombreVariable=array(valor,…);

Sintaxis (asociativo): nombreVariable=array(clave => valor,…);

Ejemplo:

```
// array numérico
$modulos1 = array(0 => "Programación", 1 => "Bases de datos", ..., 9 => "Desarrollo web en entorno servidor");
// array asociativo
$modulos2 = array("PR" => "Programación", "BD" => "Bases de datos", ..., "DWES" => "Desarrollo web en entorno servidor");
```

La función **print_r** muestra todo el contenido del array que se le pasa.

3. Tipos de datos compuestos

Para hacer referencia a los **elementos** almacenados **en un array**, hay que utilizar el valor clave entre corchetes:

$modulos1 [9]

$modulos2 ["DWES"]

En PHP se pueden crear **arrays de varias dimensiones** almacenando otro array en cada uno de los elementos de un array.

```
// array bidimensional
$ciclos = array(
"DAW" => array ("PR" => "Programación", "BD" => "Bases de datos", ..., "DWES" =>
"Desarrollo web en entorno servidor"),
"DAM" => array ("PR" => "Programación", "BD" => "Bases de datos", ..., "PMDM" =>
"Programación multimedia y de dispositivos móviles")
);
```

Para hacer referencia a los elementos de un **array multidimensional**, se deben indicar las claves para cada una de las dimensiones:

- $ciclos ["DAW"] ["DWES"]

3. Tipos de datos compuestos

En PHP no es necesario indicar el tamaño del array antes de crearlo. Ni siquiera es necesario indicar que una variable concreta es de tipo array.

```
// array numérico
$modulos1 [0] = "Programación";
$modulos1 [1] = "Bases de datos";
...
$modulos1 [9] = "Desarrollo web en entorno servidor";
// array asociativo
$modulos2 ["PR"] = "Programación";
$modulos2 ["BD"] = "Bases de datos";
...
$modulos2 ["DWES"] = "Desarrollo web en entorno servidor";
```

Tampoco es necesario especificar el valor de la clave. Si se omite, el array se irá llenando a partir de la última clave existente, o de la posición 0 si no existe ninguna:

```
$modulos1 [ ] = "Programación";
...
$modulos1 [ ] = "Desarrollo web en entorno servidor";
```

3. Tipos de datos compuestos

- **Recorrer arrays**

Para recorrer los elementos de un array, se puede usar el bucle: **foreach**.

Utiliza una variable temporal para asignarle en cada iteración el valor de cada uno de los elementos del array. Se puede usar de dos formas.

Recorriendo sólo los **elementos**:

```
$modulos = array("PR" => "Programación", "BD" => "Bases de datos", ..., "DWES" =>
                        "Desarrollo web en entorno servidor");
foreach ($modulos as $modulo) {
        print "Módulo: ".$modulo. "<br />"
}
```

O recorriendo los **elementos y** sus **valores clave** de forma simultánea:

```
$modulos = array("PR" => "Programación", "BD" => "Bases de datos", ..., "DWES" =>
                        "Desarrollo web en entorno servidor");
foreach ($modulos as $codigo => $modulo) {
        print "El código del módulo ".$modulo." es ".$codigo."<br />"
}
```

3. Tipos de datos compuestos

Hay otra forma de recorrer los valores de un array. Cada array mantiene un **puntero** interno. Utilizando funciones específicas, podemos avanzar, retroceder o inicializar el puntero, así como recuperar los valores del elemento (o de la pareja clave / elemento) al que apunta el puntero en cada momento.

Función	Resultado
reset	Sitúa el puntero interno al comienzo del array.
next	Avanza el puntero interno una posición.
prev	Mueve el puntero interno una posición hacia atrás.
end	Sitúa el puntero interno al final del array.
current	Devuelve el elemento de la posición actual.
key	Devuelve la clave de la posición actual.
each	Devuelve un array con la clave y el elemento de la posición actual. Además, avanza el puntero interno una posición.

Las funciones **reset**, **next**, **prev** y **end**, además de mover el puntero interno devuelven, al igual que **current**, el valor del nuevo elemento en que se posiciona. Si al mover el puntero se sale de los límites del array (p.ej., si está en el último elemento y hacemos **next**), cualquiera de ellas devuelve **false**. (Cuidado: no se distinguirá si se ha salido de los límites del array, o si estamos en una posición válida que contiene el valor "false".

3. Tipos de datos compuestos

key devuelve **null** si el puntero interno está fuera de los límites del array.

each devuelve un array con cuatro elementos. Los elementos **0** y **'key'** almacenan el valor de la clave en la posición actual del puntero interno. Los elementos **1** y **'value'** devuelven el valor almacenado.

Si el puntero interno se ha pasado de los límites del array, **each** devuelve false, por lo que se puede usar para recorrer el array de la siguiente forma:

```
while ($modulo = each($modulos)) {
        print "El código del módulo ".$modulo[1]." es ".$modulo[0]."<br />"
}
```

3. Tipos de datos compuestos

- **Funciones relacionadas con arrays**

Además de asignando valores directamente, **la función array permite crear un array con una sola línea de código**. Esta función recibe un conjunto de parámetros, y crea un array a partir de los valores que se le pasan.

- Si no se indica el valor de la clave, crea un array numérico (con base 0).
- Si no se le pasa ningún parámetro, crea un array vacío.

```
$a = array(); // array vacío
$modulos = array("Programación", "Bases de datos", ..., "Desarrollo web en entorno servidor"); // array numérico
```

Una vez definido un array se puede:

- añadir nuevos elementos (no definiendo el índice, o utilizando un índice nuevo)
- modificar los ya existentes (utilizando el índice del elemento a modificar).
- eliminar elementos de un array, utilizando la función **unset**.

  En caso de arrays numéricos, las claves del mismo ya no estarán consecutivas.

```
unset ($modulos [0]);
// El primer elemento pasa a ser $modulos [1] == "Bases de datos";
```

3. Tipos de datos compuestos

La función **array_values** recibe un array como parámetro, y devuelve un nuevo array con los mismos elementos y con índices numéricos consecutivos con base 0.

Para comprobar si una variable es de tipo array, se utiliza la función **is_array**. Para obtener el número de elementos que contiene un array, existe la función **count**.

Si se quiere buscar un elemento concreto dentro de un array, se puede utilizar la función **in_array**. Recibe como parámetros el elemento a buscar y la variable de tipo array en la que buscar, y devuelve true si encontró el elemento o false en caso contrario.

```
$modulos = array("Programación", "Bases de datos", "Desarrollo web en entorno servidor");
$modulo = "Bases de datos";
if (in_array($modulo, $modulos)) printf "Existe el módulo de nombre ".$modulo;
```

Otra posibilidad es la función **array_search**, que recibe los mismos parámetros pero devuelve la clave correspondiente al elemento, o false si no lo encuentra.

Y si lo que se quiere buscar es una clave en un array, está la función **array_key_exists**, que devuelve true o false.

4. Formularios web

Los **formularios HTML** se utilizan para enviar los datos del usuario a la aplicación web.

Van encerrados siempre entre las etiquetas **<FORM> </FORM>**. Dentro, se incluyen los elementos sobre los que puede actuar el usuario, usando etiquetas: **<INPUT>**, **<SELECT>**, **<TEXTAREA>, <BUTTON>...**

El atributo **action** del elemento **FORM** indica la página a la que se le enviarán los datos del formulario. En nuestro caso se tratará de una página PHP.

El atributo **method** especifica el método usado para enviar la información. Valores:

- **get**: los datos del formulario se agregan a la URL utilizando un signo de interrogación "?" como separador.
- **post**: los datos se incluyen en el cuerpo del formulario y se envían utilizando el protocolo HTTP.

Los datos se recogerán de distinta forma dependiendo de cómo se envíen.

En la página a la que se envía el formulario los parámetros estarán disponibles en el array superglobal **$_POST** o **$_GET**. La clave de cada argumento dentro del array es el atributo name del elemento correspondiente en el formulario.

$_REQUEST contiene tanto el contenido de $_POST como de $_GET.

4. Formularios web

```
<!DOCTYPE HTML>
<!-- Desarrollo Web en Entorno Servidor. - Ejemplo: Formulario web -->
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>Formulario web</title>
   </head>
   <body>
      <form name="input" action="procesa.php" method="post">
         Nombre del alumno: <input type="text" name="nombre" /><br />
         <p>Módulos que cursa:</p>
         <input type="checkbox" name="modulos[]" value="DWES" /> Desarrollo web en entorno servidor<br/>
         <input type="checkbox" name="modulos[]" value="DWEC" /> Desarrollo web en entorno cliente<br/>
         <br/>
         <input type="submit" value="Enviar" />
      </form>
   </body>
</html>
```

4. Formularios web

Procesamiento de la información devuelta por un formulario web. **Método POST**.

```
<!DOCTYPE HTML>
<!-- Desarrollo Web en Entorno Servidor - Ejemplo: Procesar datos post  -->
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>Desarrollo Web</title>
   </head>
   <body>
      <?php
         $nombre = $_POST['nombre'];
         $modulos = $_POST['modulos'];
         print "Nombre: ".$nombre."<br />";
         foreach ($modulos as $modulo)
            print "Modulo: ".$modulo."<br />";
      ?>
   </body>
</html>
```

4. Formularios web

Procesamiento de la información devuelta por un formulario web. **Método GET**.

```
<!DOCTYPE HTML>
<!-- Desarrollo Web en Entorno Servidor - Ejemplo: Procesar datos get -->
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>Desarrollo Web</title>
   </head>
   <body>
      <?php
         $nombre = $_GET['nombre'];
         $modulos = $_GET['modulos'];
         print "Nombre: ".$nombre."<br />";
         foreach ($modulos as $modulo)
            print "Modulo: ".$modulo."<br />";
      ?>
   </body>
</html>
```

4. Formularios web

Siempre que sea posible, es preferible **validar los datos que se introducen en el navegador antes de enviarlos**. Ello se puede hacer usando código **Javascript**.

Si hay datos que se tengan que **validar en el servidor**, por ejemplo, para comprobar que los datos no existan ya en la base de datos antes de introducirlos, será necesario hacerlo con **código PHP** en la página a la que se envía el formulario, (la que figura en el atributo **action)**.

Una posibilidad a tener en cuenta es usar la **misma página** que muestra el formulario **como destino de los datos**. Si tras comprobar los datos éstos son correctos, se procesan. Si son incorrectos, se mantienen los datos correctos en el formulario y se indican cuáles son incorrectos y por qué.

Para hacerlo de este modo, hay que comprobar si la página recibe datos (hay que procesarlos y no generar el formulario), o si no recibe datos (hay que mostrar el formulario). Esto se puede hacer utilizando la función **isset** con una variable de las que se deben recibir (por ejemplo, poniéndole un nombre al botón de enviar y comprobando sobre él).

En el siguiente código de ejemplo se muestra cómo hacerlo:

4. Formularios web

Procesar los datos en la misma página que el formulario.

```
<!DOCTYPE HTML>
<!-- Desarrollo Web en Entorno Servidor - Ej: Procesar datos en la misma página del formulario -->
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>Desarrollo Web</title>
   </head>
   <body>
      <?php
         if (isset($_POST['enviar'])) {
            $nombre = $_POST['nombre'];
            $modulos = $_POST['modulos'];
            print "Nombre: ".$nombre."<br />";
            foreach ($modulos as $modulo) {
               print "Modulo: ".$modulo."<br />";
            }
         }
```

4. Formularios web

```
    else {
  ?>
  <form name="input" action="<?php echo $_SERVER['PHP_SELF'];?>" method="post">
    Nombre del alumno: <input type="text" name="nombre" /><br />
    <p>Módulos que cursa:</p>
    <input type="checkbox" name="modulos[]" value="DWES" /> Desarrollo web en entorno
    servidor<br />
    <input type="checkbox" name="modulos[]" value="DWEC" /> Desarrollo web en entorno
    cliente<br />
    <br />
    <input type="submit" value="Enviar" name="enviar"/>
  </form>
  <?php
    }
  ?>
 </body>
</html>
```

Se engloba el formulario dentro de un **else** para que sólo se genere si no se reciben datos en la página. Para enviar los datos a la misma página que contiene el formulario se utiliza **$_SERVER['PHP_SELF']**.

4. Formularios web

Generación de formularios web en PHP.

En el ejemplo anterior, antes de mostrar los datos habría que **comprobar** que el **nombre no** esté **vacío**, y que se haya seleccionado como mínimo uno de los módulos.

Además, en el caso de que falte algún dato, se deberá generar el formulario rellenando aquellos datos que el usuario haya introducido correctamente.

Para hacer la **validación de los datos**. En el ejemplo propuesto, será algo así:

```
if (!empty($_POST['modulos']) && !empty($_POST['nombre'])) {
    // Aquí se incluye el código a ejecutar cuando los datos son correctos
}
else {
    // Aquí generamos el formulario, indicando los datos incorrectos
    // y rellenando los valores correctamente introducidos
}
```

4. Formularios web

Para **no perder los datos introducidos correctamente**, se utiliza el atributo **value** en las entradas de texto:

```
Nombre del alumno:
<input type="text" name="nombre" value="<?php echo $_POST['nombre'];?>" />
```

Y el atributo **checked** en las casillas de verificación:

```
<input type="checkbox" name="modulos[]" value="DWES"
   <?php
      if(in_array("DWES",$_POST['modulos'])) // in_array busca un elemento en un array
         echo 'checked="checked"';
   ?>
/>
```

Para indicar al usuario los datos que no ha rellenado (o que ha rellenado de forma incorrecta), se deberá **comprobar si es la primera vez que se visualiza el formulario**, o si ya se ha enviado. Se puede hacer de la siguiente forma:

```
Nombre del alumno:
<input type="text" name="nombre" value="<?php echo $_POST['nombre'];?>" />
<?php
if (isset($_POST['enviar']) && empty($_POST['nombre']))
echo "<span style='{color:red}'> <-- Debe introducir un nombre!!</span>"
?><br />
```

4. Formularios web

Ejemplo de validación

```
<!DOCTYPE HTML>
<!-- Desarrollo Web en Entorno Servidor – Ej: Validar datos en la misma página que el formulario -->
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>Desarrollo Web</title>
   </head>
   <body>
      <?php
         if (!empty($_POST['modulos']) && !empty($_POST['nombre'])) {
            $nombre = $_POST['nombre'];
            $modulos = $_POST['modulos'];
            print "Nombre: ".$nombre."<br />";
            foreach ($modulos as $modulo) {
               print "Modulo: ".$modulo."<br />";
            }
         } else {
      ?>
```

4. Formularios web

```
<form name="input" action="<?php echo $_SERVER['PHP_SELF'];?>" method="post">
   Nombre del alumno:
   <input type="text" name="nombre" value="<?php echo $_POST['nombre'];?>" />
   <?php
      if (isset($_POST['enviar']) && empty($_POST['nombre']))
         echo "<span style='color:red'> &lt;-- Debe introducir un nombre!!</span>"
   ?><br />
   <p>Módulos que cursa:
   <?php
      if (isset($_POST['enviar']) && empty($_POST['modulos']))
         echo "<span style='color:red'> &lt;-- Debe escoger al menos uno!!</span>"
   ?>
   </p>
   <input type="checkbox" name="modulos[]" value="DWES"
   <?php
      if(isset($_POST['modulos']) && in_array("DWES",$_POST['modulos']))
         echo 'checked="checked"';
   ?>
   />
   Desarrollo web en entorno servidor
   <br />
```

4. Formularios web

```
        <input type="checkbox" name="modulos[]" value="DWEC"
        <?php
            if(isset($_POST['modulos']) && in_array("DWEC",$_POST['modulos']))
                echo 'checked="checked"';
        ?>
        />
        Desarrollo web en entorno cliente<br />
        <br />
        <input type="submit" value="Enviar" name="enviar"/>
    </form>
    <?php
        }
    ?>
  </body>
</html>
```