PHP

1. ÁMBITO DE LAS VARIABLES

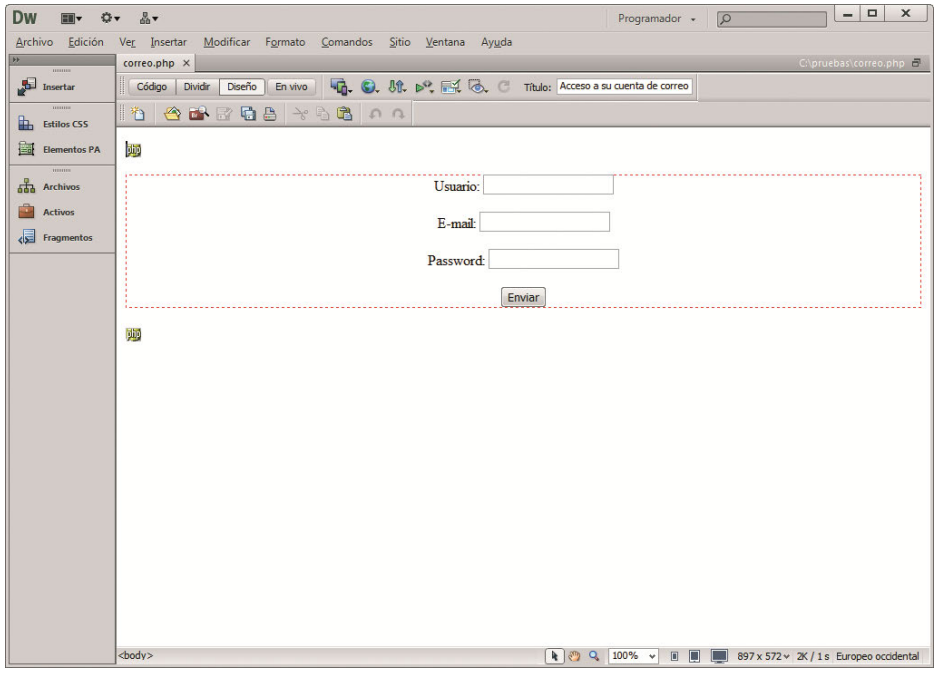
En esta lección continuaremos viendo detalles interesantes a la hora de trabajar con funciones y variables.

Empezamos estudiando lo que se conoce como *"ámbito de una variable"*. Este concepto hace referencia a dónde se puede utilizar o es conocida una variable. Principalmente se pueden encontrar dos tipos de variables en función de su ámbito:

- Variables **globales**: son todas aquellas que se definen fuera de las funciones, por lo que están accesibles en cualquier lugar del código php.
- Variables **locales**: son aquellas que se definen en el interior de una función. Solo están disponibles en esa función y desaparecen al finalizar la ejecución de la función.

Veámoslo con un ejemplo. Fíjate en la página web de la imagen siguiente.

Se trata de un formulario para recoger tres detalles del usuario: su nombre de usuario, dirección de correo electrónico y contraseña.



A continuación puedes ver parte de su código:

```
function ValidarEmail()
{
     if (strpos($email, "@") === FALSE)
          return FALSE;
     else
          return TRUE;
}

function ValidarPassword()
{
     if (strlen($password) < 6)
          return FALSE;
     else
          return TRUE;
}

$usuario = $_POST["Usuario"];
$password = $_POST["Password"];
$email = $_POST["Email"];
if (ValidarEmail() && ValidarPassword())
     echo "<h1>Formato correcto</h1>";
else
     echo "<h1>Formato incorrecto</h1>";

}
```

Se definen tres variables ($usuario, $password y $email), que como están fuera de una función, tendrán un ámbito **global**.

Esto quiere decir que deberíamos poder utilizarlas en cualquier sitio del código PHP de esta página. Su propósito es el de recoger la información enviada por el usuario en el formulario.

Seguidamente se llaman a dos funciones para validar el formato de la dirección de correo electrónico y de la contraseña: **ValidarEmail** y **ValidarPassword**, respectivamente.

Por ejemplo, la función **ValidarEmail** utiliza la función propia del lenguaje PHP **strpos** para comprobar si en el valor introducido por el usuario aparece el carácter arroba (**@**).

Como una primera aproximación, indicaremos que el formato no es correcto si no aparece ese carácter en la dirección de correo suministrada por el usuario.

En este caso se utiliza el operador con tres signos igual (**===**), que comprueba si los valores y el tipo de datos de ambos operandos son iguales.

Esto es necesario porque la función **strpos** devuelve el valor 0 si el carácter @ está en el primer lugar, lo que se podría confundir con el valor FALSE.

Fíjate que se utiliza la variable **$email** en el interior de la función. Ya que se trata de una variable global, esto no debería representar ningún problema porque, además, no importa el orden en que aparezcan las funciones y el resto del código.

Algo parecido ocurre en la función **ValidarPassword**, donde se comprueba si la contraseña introducida por el usuario tiene al menos 6 caracteres de longitud. Para ello, se utiliza la función **strlen** del PHP. Aquí utilizamos la variable global **$password** para ello.

Finalmente, se llama a estas dos funciones y, si el resultado es **TRUE** en ambos casos, entonces se indica que el formato es correcto; en otro caso, se indica que es incorrecto.

Sin embargo, si compruebas el resultado de este código, verás que siempre se indica que el formato de los datos introducidos es incorrecto.

El problema está en que las funciones utilizan las variables **$email** y **$password** en su interior, por lo que las toman como locales y no recogen el valor de las variables globales definidas fuera de las funciones.

Vemos, por lo tanto, que las variables globales no se pueden utilizar directamente en el interior de las funciones, ya que, al utilizar el mismo nombre, PHP entiende que estás definiendo una variable local en la función.

Esto puede solucionarse con la palabra **global** antes de la variable. Por ejemplo, en la función **ValidarEmail**:

```
function ValidarEmail()
{
      global $email;
      if (strpos($email, "@") === FALSE)
            return FALSE;
      else
            return TRUE;
}
```

Lo mismo haríamos con la variable **$password** en la función **ValidarPassword**.

De esta forma sí que se utilizará la variable global en el interior de cada función, con lo que se obtendrá el valor introducido por el usuario en el formulario.

Finalmente, indicarte que si quieres eliminar manualmente una variable, puedes hacerlo utilizando la función **unset()** y pasando como parámetro el nombre de la variable.

2. VARIABLES ESTÁTICAS

Sabemos que las variables locales, que se definen en el interior de una función, solo pueden utilizarse en dicha función y, además, desaparecen al finalizar esta.

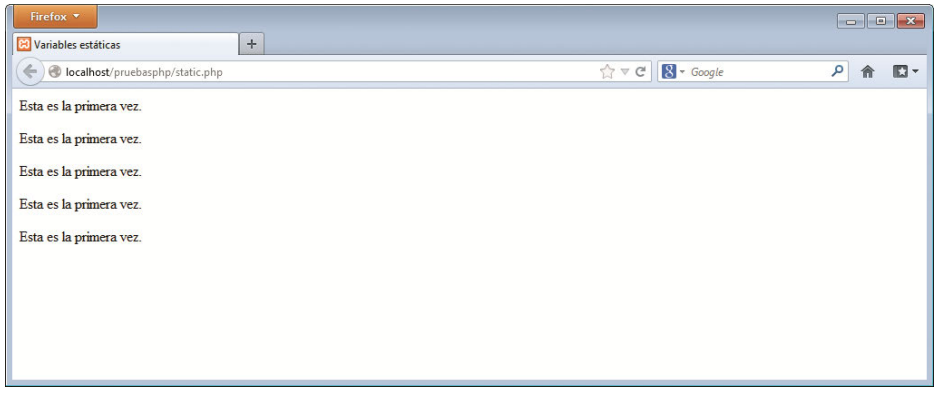
Esto es muy fácil de comprobar. Fíjate en la función **contar** siguiente.

```
function contar()
{
      $cuenta = 1;

      if ($cuenta == 1)
            echo "<p>Esta es la primera vez.</p>";
      else
            echo "<p>Esta es la vez número $cuenta.</p>";
      $cuenta++;
}

for ($i = 1; $i <= 5; $i++)
{
      echo contar();
}
```

Se utiliza una variable llamada **$cuenta** a la que se le da el valor inicial **1**, seguidamente se imprime uno u otro mensaje en función de su valor y se incrementa en una unidad.

La página simplemente llama **5** veces a la función anteriormente mencionada, para lo que se utiliza un bucle **for**.

Si compruebas el resultado en un navegador, verás el primer mensaje impreso 5 veces porque la variable recibe el valor **1** cada vez que se ejecuta la función, por lo que no guarda el valor incrementado al final de la misma.

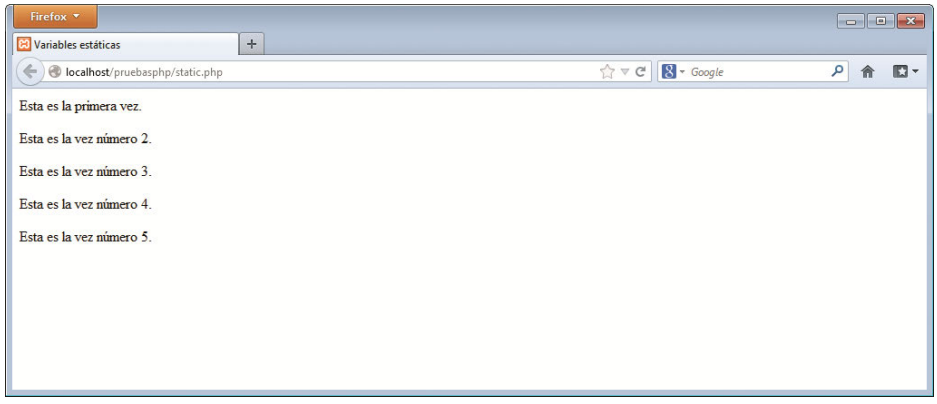


Podemos hacer, sin embargo, que la variable local guarde su valor entre las distintas llamadas a la función. Para ello, se utiliza la palabra **static**.

```
function contar()
{
      static $cuenta = 1;

      if ($cuenta == 1)
            echo "<p>Esta es la primera vez.</p>";
      else
            echo "<p>Esta es la vez número $cuenta.</p>";
      $cuenta++;
}
```

Como ahora sí que se guarda el valor que va teniendo la variable entre cada llamada a la función, se imprimen los distintos mensajes.

Recuerda que esto no es el funcionamiento normal de las variables locales, ya que desaparecen al finalizar la ejecución de la función.



3. USO DE INCLUDE Y REQUIRE

Cuando empieces a crear tus propias funciones, comprobarás que te interesa utilizarlas en varias de las páginas de tu aplicación.

En lugar de escribir el mismo código en una y otra página, es mejor escribir este código *"reutilizable"* en un archivo independiente e incluirlo en las páginas donde se necesite.

Para ello, se utilizan las instrucciones **include** o **require**. Ambas funcionan prácticamente igual, pero mientras que la primera puede provocar una advertencia si no se encuentra el archivo incluido, **require** provoca un error fatal, finalizando la ejecución de la página.

Vamos a poner el código de las funciones **ValidarEmail** y **ValidarPassword** en un archivo independiente y después lo utilizaremos aquí. Es lógico pensar que estas funciones las podamos utilizar en más de una página de nuestra aplicación.

Un aspecto importante es que el código php que escribamos en archivos independientes tiene que estar incluido entre las etiquetas de inicio y cierre de php.

Archivo **validator.inc.php**:

```
<?php
function ValidarEmail()
{
     global $email;
     if (strpos($email, "@") === FALSE)
           return FALSE;
     else
           return TRUE;
}

function ValidarPassword()
{
     global $password;
     if (strlen($password) < 6)
           return FALSE;
     else
           return TRUE;
}
?>
```

Otro aspecto a tener en cuenta es qué nombre asignar a los archivos que posteriormente serán incluidos.

Podemos utilizar cualquier extensión para este tipo de archivos, ya que, en teoría, solo deberán ser utilizados cuando se incluyan en otras páginas php.

Por ello, es una costumbre utilizar la extensión **.inc** (de incluido), aunque es mejor, como después demostraremos, utilizar la misma extensión **.php** que para el resto de páginas dinámicas.

Nosotros utilizaremos una combinación de ambas para diferenciar estos archivos respecto de las páginas ejecutables. Vamos a asignar la extensión **.inc.php**.

Ahora incluiremos este archivo para poder utilizar las funciones que validan los datos introducidos por el usuario.

```
{
 include_once("validator.inc.php");

 $usuario = $_POST["Usuario"];
 $password = $_POST["Password"];
 $email = $_POST["Email"];
 if (ValidarEmail() && ValidarPassword())
     echo "<h1>Formato correcto</h1>";
 else
     echo "<h1>Formato incorrecto</h1>";

}
```

Si escribes una instrucción include o require dentro de una estructura condicional como if – else, inclúyela siempre entre llaves { } para que todo el código incluido se escriba en la parte correcta de la estructura condicional.

Al utilizar la instrucción **include** o **require**, el intérprete PHP incluirá el código que se encuentra en el archivo indicado, exactamente igual que si hubieses escrito este código en la posición de la línea **include**, pero ahora tenemos la ventaja de incluir las funciones allá donde las necesitemos sin necesidad de volver a escribir todo su código, sino simplemente con una línea.

Debes entender que al incluir (**include**) o requerir (**require**) el código de uno o más archivos, estás haciendo que ese código se incruste en la posición donde aparecen estas líneas.

Esto quiere decir que incluso en dicho código se podría utilizar cualquier variable, constante o función que apareciese en el código de la página que los utiliza.

También quiere decir que en esos archivos podríamos escribir otro tipo de código, como el mismo código HTML que queremos incluir en más de una ocasión. Por ejemplo, esto serviría para reproducir el mismo aspecto en las páginas de la aplicación con un encabezado o pie de página incluidos en cada una de ellas.

4. INCLUIR UNA SOLA VEZ

Una versión un poco diferente de la instrucción include (o require) es **include_once** (o **require_once**).

Se utiliza para no incluir en más de una ocasión el mismo archivo independiente, ya que esto no sería correcto porque el código aparecería repetido, dando errores de duplicación de identificadores de función.

Veámoslo con un ejemplo para entenderlo mejor.

Archivo **svalidator.inc.php**:

```
<?php
include("validator.inc.php");
function SecurePassword($user, $newpassword,
                        $oldpassword)
{
      if (ValidarPassword())
      {
            if ($newpassword == $oldpassword)
                  return FALSE;
            elseif (stristr($newpassword, $user) != "")
                  return FALSE;
            else
                  return TRUE;
      }
      else
            return FALSE;
}
?>
```

En este archivo aparece una única función llamada **SecurePassword**, que es una versión un poco más *"exigente"* en cuanto a las características que tiene que cumplir la contraseña del usuario.

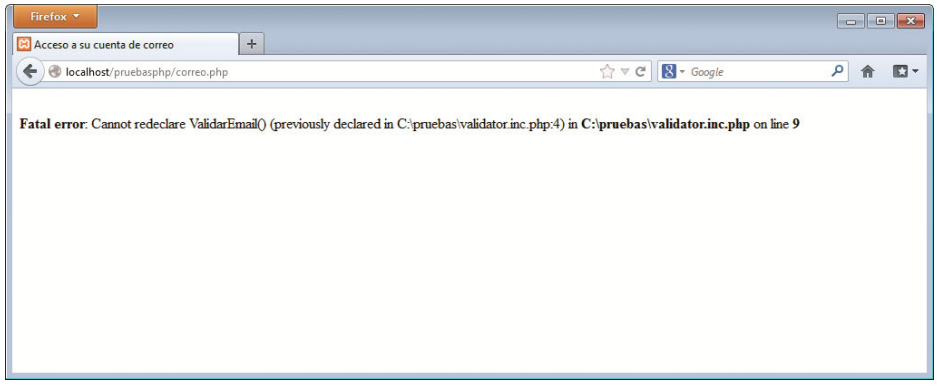
Fíjate que como se utiliza la función **ValidarPassword**, hemos incluido también el archivo **validator.inc.php**.

Vemos, por lo tanto, que si incluimos el archivo **svalidator.inc.php** estaremos, de forma indirecta, incluyendo también el archivo **validator.inc.php**.

¿Qué ocurre entonces si incluimos ambos archivos en la misma página web?

```
include("svalidator.inc.php");
include("validator.inc.php");
```

Muy fácil: que realmente el archivo **validator.inc.php** se incluirá dos veces.



Vemos que se ha producido el error que estábamos esperando. Se nos indica que no podemos volver a definir la función **ValidarEmail** porque ya está definida con anterioridad. Esto se debe a que hemos incluido dos veces el mismo archivo **validator.inc.php**.

Utilizando **include_once** en lugar de **include**, el archivo especificado solo se incluirá si no se ha hecho con anterioridad. De la misma forma funciona la versión **require_once**.

```
include("svalidator.inc.php");
include_once("validator.inc.php");
```

Debes tener en cuenta que utilizando **include_once** o **require_once**, solo se insertará el código del archivo una vez, aunque aparezcan varias líneas indicándolo.

De ahí que es necesario disponer también de la instrucción **include** o **require** para poder incluir en más de una ocasión el mismo archivo. Piensa, por ejemplo, si quieres incluir en varios lugares de la página web el mismo fragmento de código para incorporar el mismo texto o imágenes.

5. SEGURIDAD EN LOS ARCHIVOS INCLUIDOS

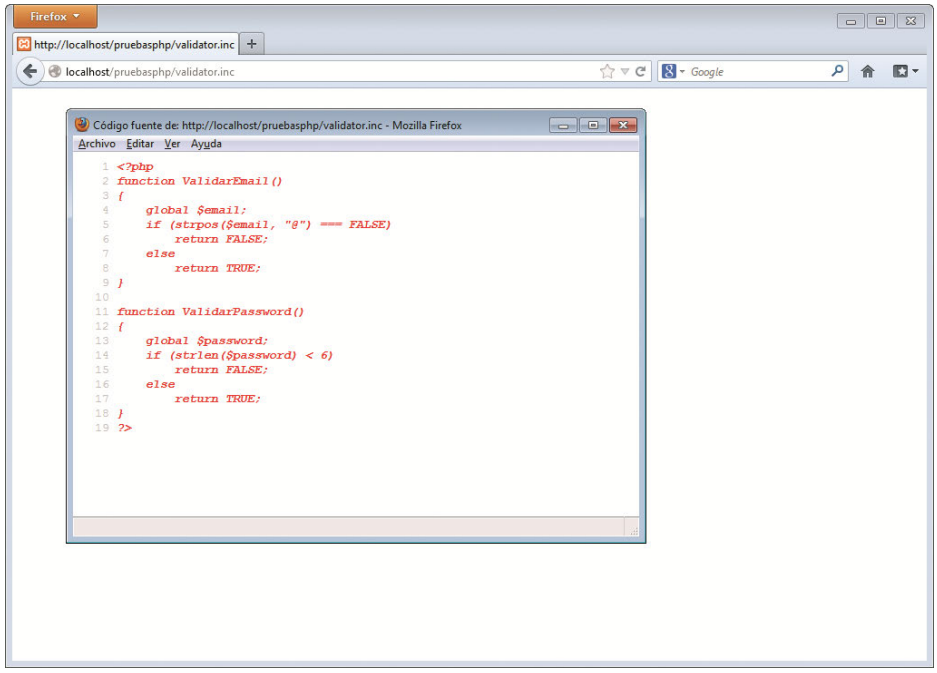
Dónde guardar los archivos independientes con código que después se incluirán es un asunto a estudiar detenidamente.

En esos archivos podemos encontrar código realmente importante que el usuario no debería ver de ninguna forma, como contraseñas, funciones internas, etc.

De ahí que sea importante la extensión que les asignemos. Como se ha indicado, es mejor utilizar la extensión **.php** ¿Por qué? Ahora lo verás.

Imagínate que incluiste el código de las dos funciones anteriores en un archivo de nombre **validator.inc**. Si el usuario supiera que existe dicho archivo en el directorio donde se almacenan las páginas web a las que tiene acceso, no tendría más que escribir el nombre del archivo en el campo de direcciones del navegador para poder acceder a su contenido. Deberías hacerlo visualizando el código fuente de la página.

Fíjate en la imagen de la página siguiente.



¿Por qué ocurre esto? Como ahora la extensión del archivo no coincide con ninguna de las que tiene que interpretar el servidor web, sí que podemos ver el código PHP, ya que no ha sido procesado por el servidor.

Esto permite al usuario ver este código, que no debería ver en ningún caso. De ahí que sea importante utilizar la extensión **.php**, ya que en lugar de mostrar el código, simplemente se ejecutaría.

En este caso, el resultado de la página enviada al navegador sería totalmente en blanco, así como su código fuente, ya que ese sería el resultado de procesar el código PHP.

Aunque esta es una buena medida, no es suficiente. Debes asegurarte de que el usuario no pueda acceder a los archivos independientes.

Para ello, lo mejor es guardarlos en una carpeta o directorio fuera del espacio web al que tiene acceso el usuario, es decir, fuera de la jerarquía de documentos, alias, etc. del servidor web Apache.

Por ejemplo, podríamos crear un directorio en nuestro disco duro llamado **funciones** y guardar estos archivos allí.

En este caso, deberíamos indicar la ubicación completa de los archivos en las líneas **include**, pero, a cambio, nadie podría acceder a dicho código utilizando su navegador, ya que estaría fuera del espacio web al que tiene acceso a través del servidor web Apache.

Otra posibilidad que existe en cuanto a asegurar los archivos incluidos es utilizar una extensión específica, como **.inc**, e impedir que se pueda acceder a este tipo de archivo.

Esto último lo indicaríamos en el archivo de configuración de Apache **httpd.conf**.

En el ejemplo planteado podríamos incluir el siguiente fragmento de código para proteger los archivos **.inc**:

```
<Files "*.inc">
    Order deny,allow
    Deny from all
</Files>
```

El problema que tiene esto es que tenemos que acceder al archivo de configuración de Apache, lo que no siempre es posible (piensa, por ejemplo, si la aplicación está siendo hospedada por una empresa de servicios de Internet o ISP).

En este último caso, todavía podríamos aplicar esta regla creando un archivo llamado **.htaccess** en el directorio de nuestra aplicación e incluyéndola.

En resumen:

Tenemos que proteger los archivos incluidos para que nadie pueda acceder a su contenido. La forma más fácil de hacerlo es guardarlos fuera de la jerarquía de documentos del sitio web y con la extensión **.php**