MEDAC
Instituto Oficial de Formación Profesional

DESARROLLO WEB EN ENTORNO SERVIDOR
TÉCNICO EN DESARROLLO DE APLICACIONES WEB

Programación basada en lenguaje de marcas con código embebido II

04

ÍNDICE

OBJETIVOS

Trabajar con funciones ya existentes.

Crear y utilizar nuevas funciones con y sin paso de parámetros.

Utilizar formularios web para interactuar con el usuario del navegador web.

Emplear métodos para recuperar la información introducida en un formulario.

Añadir comentarios al código.

/ 1. Introducción y contextualización práctica

En los temas anteriores, hemos realizado una programación menos estructurada. Sí que hemos utilizado algunas funciones ya implementadas en librerías de PHP, pero no hemos creado las nuestras; por lo tanto, si era necesario reutilizar código, volvíamos a escribirlo.

Con la creación de funciones, se reducen los costes en desarrollo de software. Además, el resultado siempre será más legible, se disminuirá la posibilidad de cometer algún error al escribirlo y, en el caso de hacerlo, también será más cómodo depurarlo. Por todas estas razones, conviene evitar que nuestro programa contenga código repetitivo.

En este tema, también comenzaremos con el diseño de formularios y el modelado con los métodos GET y POST, que permiten enviar al servidor toda la información necesaria para que la petición solicitada se muestre en el navegador del cliente.

Escucha el siguiente audio donde planteamos la contextualización práctica de este tema. Encontrarás su resolución en el apartado «Resumen y resolución del caso práctico de la unidad».

Fig.1. Diseño de formulario.

Audio intro. "¿Son necesarias las funciones?"

https://bit.ly/2PKJ8NZ

/ 2. Funciones

Las funciones, en programación, son **subrutinas o subprogramas** que forman parte del programa principal. Cada una de estas funciones recibe un nombre que la diferencia del resto de subrutinas y que permite que sea invocada desde cualquier parte del código total del programa. Opcionalmente, puede recibir valores, se ejecuta, y puede devolver un valor. La existencia de las funciones permite obtener un programa más ordenado.

Para comprender mejor este concepto, veamos el siguiente ejemplo. Si dentro un programa se solicita a un usuario que se introduzca un determinado valor por teclado:

- **¿Es necesario realizar la función descrita en la imagen 2, cada vez que se reciba una variable?**
- **¿Bastará con invocar a esa subrutina?**

Al diseñar una función que implemente la funcionalidad descrita, bastará con invocarla cada vez que se requiera de su presencia.

Al crear funciones, se reducen los costes de desarrollo de software; además, el resultado del código siempre será más legible, se disminuirá la posibilidad de cometer algún error al escribirlo, y, en el caso de hacerlo, también será más cómodo depurarlo.

Por todas estas razones, conviene evitar que nuestro programa contenga código repetitivo.

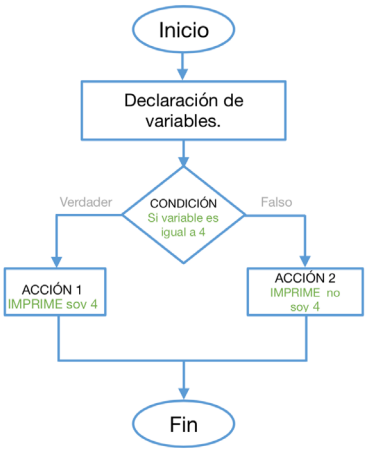

Fig.2. Diagrama de una función.

/ 3. Principales funciones en PHP

PHP incorpora un amplio repositorio de funciones útiles para el desarrollo de este lenguaje de programación. A continuación, se muestran algunas de las más relevantes, atendiendo al tipo de elemento sobre el que actúan.

A. **Funciones de hora y fecha**

Esta función permite generar, de forma automática, la fecha o la hora actual. Recibe por parámetro, entre comillas dobles, el formato de la fecha o la hora a mostrar. A continuación, se muestra tanto su sintaxis como algunos de los indicadores de formato más comunes.

```
date(string $format [, int $timestamp = time() ] ) ;
```

Código 1. Generación de fecha.

INDICADOR	DESCRIPCIÓN
d	Día en dígito (01-31)
D	Día textual (Mon-Sun)
m	Mes (01-12)
M	Mes (Jan-December)
y	Año (4 dígitos)
H	Hora en 24 horas (0-23)
h	Hora en 12 horas (1-12)
i	Minutos con 0 iniciales
s	Segundos con 0 iniciales

Tabla 1. Propiedades de formato función fecha en PHP.

B. **Números aleatorios:** Esta función permite generar, de forma automática, un número aleatorio. Puede recibir como parámetro los valores mínimos y máximos entre los que se desea generar el número.

```
rand(); //Número aleatorio sin límite
rand($valorMinimo, $valorMaximo);
```

Código 2. Función números aleatorios en PHP.

C. **Directorios:** La función getcwd indica el directorio exacto en el que se está trabajando. La ejecución de esta función devuelve la ruta completa desde la raíz del sistema operativo.

```
getcwd();
```

Código 3. Función directorios en PHP.

D. **Imprimir**

Son varias las funciones que permite la impresión de una cadena de texto en PHP. Una de las más utilizadas es *echo*, pero existen otras como *print* y algunas variaciones de esta.

```
echo "Hola";
print ("Hola");
print "Hola";
```

Código 4. Funciones para imprimir en PHP.

E. **Buscar ficheros**

En ocasiones, no recordamos el nombre de un fichero PHP. Para realizar una búsqueda completa, se utiliza la función *glob*, que recibe por parámetro el nombre del fichero que queremos buscar.

```
glob('fichero.php);
```

Código 5. Función búsqueda de ficheros en PHP.

Si no se recuerda el nombre exacto, obtendremos un listado de todos aquellos ficheros de tipo PHP. La función *print_r* permite imprimir todo el contenido de un *array*. Es una función muy útil.

```
$nombreficheros = glob('*.php);
print_r($nombreFicheros);
```

Código 6. Ejemplo completo con bucle for en PHP.

F. **Creación de ficheros**

Para la creación de nuevos ficheros desde PHP, se utiliza la función ***fopen()***. Esta función crea un nuevo fichero con el nombre indicado por parámetro, si no existe. Por el contrario, si existiera, lo abre y queda vinculado con la variable sobre la que se asigna su valor y que recibe el nombre de manejador.

```
$archivo = fopen ('ruta_nombreFichero');
```

Código 7. Bucle while en PHP.

Tras utilizar el fichero, hay que cerrar el vínculo entre la variable y el fichero. Para ello, se utiliza la función **fclose($archivo)**. Esta función recibe como parámetro el nombre de la variable, no el del fichero.

La función *fopen()* también puede recibir como segundo parámetro el modo de creación o apertura de un fichero. Esto es muy importante porque, en ocasiones, nos puede interesar solo mostrar un documento, pero que no se permite ninguna modificación sobre él.

MODO	DESCRIPCIÓN
a	Modo escritura, el nuevo texto se añade al anterior, no se reemplaza
w	Modo escritura, se borra lo anterior
w+	Igual que el anterior pero además permite la lectura
r	Modo de lectura
r+	Modo de lectura y escritura
a+	Modo que permite añadir texto al ya existente y además permite la lectura

Tabla 2. Tipos de apertura de ficheros en PHP.

/ 4. Creación de una función en PHP

La creación de funciones constituye el eje central de la programación modular y la reutilización del código. Estos dos aspectos son claves en la actualidad, puesto que permiten utilizar librerías, componentes o cualquier otro tipo de elemento que ya está diseñado, probado y verificado.

Gracias a la creación de nuevas funciones, será posible reducir el tiempo empleado en el desarrollo repetitivo de instrucciones, que pueden ser agrupadas bajo una misma función. De esta forma, la implementación de nuevas aplicaciones podrá centrarse en las nuevas especificaciones, tanto en su desarrollo, como en la posterior prueba.

En PHP será posible utilizar funciones ya definidas o crear nuevas en cualquier programa. La sintaxis utilizada se basa en el uso de la palabra ***function*** con las siguientes especificaciones:

```
function nombreFuncion($argumentos...)
{
        //acciones
  return $valor;
}
```

Código 8. Sintaxis creación de una función en PHP.

- Se indica el nombre que se le asigna a la función y con el que podrá ser invocado desde el resto del programa. Se coloca a continuación de la palabra ***function***.
- Entre **paréntesis, se indican los argumentos que recibe**, pero esto es opcional.
- Además, a través de la palabra ***return,*** se devuelve el valor resultado de la ejecución de la función. Esta devolución es opcional.

```
<?php
 function saludar($nombre)
 {
            echo “hola $nombre”
             }
 $nombre=“Pepe”;
             saludar($nombre);
?>
```

Código 9. Ejemplo Hola Mundo en PHP.

En este ejemplo, se invoca a la función *saludar,* que ha sido definida para que concatene una cadena al valor que recibe por parámetro. Como se puede ver, para llamar a una función, basta con utilizar su nombre e indicarle, entre paréntesis, los parámetros que espera recibir; de lo contrario, no funcionará.

4.1. Paso de parámetros

Como se visto en el apartado anterior, las funciones permiten diferentes tipos de implementaciones. Distinguimos entre las que reciben valores por parámetro y las que no.

Las funciones que **no reciben ningún valor por parámetro** se configuran de la siguiente forma:

```
function nombreFuncion(){
            //acciones
}
```

Código 10. Sintaxis creación de una función sin parámetros en PHP.

Mientras que las que **reciben parámetros** se definen indicando entre paréntesis los parámetros que espera recibir, si se introduce más de uno, se utilizan la coma (,) para separarlos.

```
function nombreFuncion($arg1, $arg2 ...){
            //acciones
}
```

Código 11. Sintaxis creación de una función con parámetros en PHP.

Ahora bien, la ubicación en el paso de parámetros a estas funciones debe cumplir la misma posición que se indica en su definición como función.

Por ejemplo, si el siguiente caso se reciben dos valores, donde el segundo se concatena al primero, cuando se llame a esta función, el paso de parámetros deberá respetar el orden definido en la creación de la función; de lo contrario, el resultado no sería el esperado.

```
<?php
 function saludar($nombre, $apellido){
                        echo “hola $nombre
$apellido”;
        }
 $nombre=“Pepe”;
 $apellido=“García”;
        saludar($nombre, $apellido);
 // hola Pepe García
 saludar($apellido, $nombre);
        // hola García Pepe
?>
```

Código 12. Ejemplo creación de una función con parámetros en PHP.

Es posible que una función reciba menos parámetros que los indicados en su creación. En estos casos, tenemos que recordar que la función tomará los valores pasados por parámetro de izquierda a derecha, dejando sin asignar los últimos.

4.2. Retorno de valores en funciones. Return

De la misma forma que las funciones pueden recibir o no parámetros de entrada, también será opcional que estas devuelvan un valor o no.

Las funciones que **no devuelven ningún valor** simplemente no utilizan ninguna palabra de retorno, son de la forma habitual:

```
function nombreFuncion(con/sin argumentos){
        //acciones
}
```

Código 13. Sintaxis creación de una función sin retorno en PHP.

Por otro lado, para que una función sí devuelva un valor como resultado, se utiliza la palabra **return** seguida del nombre de la variable donde queda contenida la respuesta.

```
function nombreFuncion(con/sin argumentos){
        //acciones
 return $variable;
}
```

Código 14. Sintaxis creación de una función con retorno en PHP.

Tras la llamada a una función que devuelve un valor, es importante tener en cuenta que, si este se desea guardar en una variable, se debe completar el proceso de asignación habitual.

Vídeo 1. "Creación y utilización de funciones"

https://bit.ly/2DDf193

Por ejemplo, el siguiente código llama de nuevo a la función *saludar_sinsalida($nombre, $apellido)*, pero no devuelve ningún valor; por lo tanto, tras esta llamada, no podemos almacenar ningún valor en el programa principal. En cambio, en el caso de función *saludar_consalida($nombre, $apellido),* la cadena resultante se puede almacenar en una variable en el programa principal para poder seguir trabajando sobre ella.

```
<?php
 function saludar_sinsalida($nombre, $apellido)
{
        echo "hola $nombre $apellido";
}
 function saludar_consalida($nombre,
$apellido){
        $resultado="hola".$nombre.$apellido;
        echo $resultado;
 return $resultado;
}
 $nombre="Pepe";
 $apellido="García";
        saludar_sinsalida($nombre, $apellido);
 // hola Pepe García
 $var=saludar_consalida($apellido, $nombre);
        // hola García Pepe
?>
```

Código 15. Ejemplo creación de una función con retorno en PHP.

/ 5. Método GET

Hasta ahora, hemos definido todas las características esenciales de la programación en un lenguaje de desarrollo en entorno servidor. Ahora bien, será necesario el uso de unos métodos concretos que permitan el envío y recepción de datos. Hablamos de los métodos GET y POST.

Fig.3. Métodos GET y POST para protocolo HTTP

El **método GET** se utiliza para enviar una **petición de información** desde el navegador al servidor. Es el método utilizado con más frecuencia.

Este método modela una petición, que recibe el nombre de URI y es enviada al servidor PHP, que la procesa y devuelve el resultado. El método GET envía la información de forma visible al usuario.

GET construye y envía la petición a través de la propia URL, toda la información queda a la vista del usuario. Por esta razón, no se recomienda para el envío de información confidencial. La URL está formada por el nombre del destino seguido de los datos. Para definir estos datos se envía el nombre de la variable y el contenido. Si se realiza un envío con más de un dato se utiliza el símbolo '&' como separador.

```
URL: PROTOCOLO + DOMINIO + DIRECTORIO + FICHEROS + DATOS
http://www.dominio.com/directorio/file/ficheroPHP.php?var1=0&var2=0
```

Código 16. Modelado de la URL de petición.

Tras la URL, se colocan los datos a enviar, como se ha indicado anteriormente, teniendo en cuenta las directrices:

- **?**: Tras el nombre del fichero .php se coloca este símbolo.
- Se concatena el nombre de la variable y el valor.
- Entre cada dato se utiliza **&**.

/ 6. Método POST

El método POST se utiliza para **enviar información al servidor**, puesto que, en este caso, la información no se muestra en la URL visible en el navegador. El envío de los datos en POST se realiza a través de la variable STDIO.

Una de las ventajas del envío con POST, a diferencia de GET, es que el primero sí permite enviar ficheros.

El método POST, tras el modelado de la petición, la envía al servidor PHP, que la procesa y devuelve el resultado. El método POST envía la información de forma **invisible** al usuario.

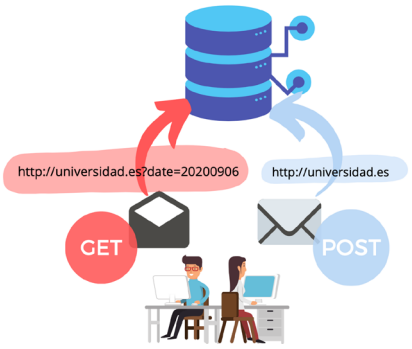

Fig.4. Diagrama GET y POST.

POST también modela el envío de los datos, por lo que necesita que estos estén claramente identificados con un nombre (**name**) en cada una de las etiquetas de elementos de los formularios. La diferencia es que ahora este envío no queda a la vista del usuario.

```
URL: PROTOCOLO + DOMINIO + DIRECTORIO + FICHEROS
```

Código 17. Modelado de la URL de petición POST.

De nuevo, la URL queda formada por el nombre del fichero PHP destino.

La diferencia con respecto al método GET es que, tras la URL, no aparecen los datos a enviar, un ejemplo sería el siguiente:

```
http://www.dominio.com/directorio/file/ficheroPHP.php
```

Código 18. Ejemplo URL de petición POST.

Es posible implementar varios tipos de métodos en un mismo programa. En estos casos, cada uno de los formularios que se desarrollaran podrán utilizar GET o POST en función de la conveniencia del diseño.

Audio 1. "Protocolo HTTP"
https://bit.ly/3aj5iAp

/ 7. Formularios con método GET

El modelado de las peticiones se realiza utilizando formularios, los cuales permiten tomar diferente tipo de información completada por un cliente web y realizar el envío de esta al servidor PHP.

Veremos cómo se realiza este modelado con el método GET y con el método POST.

En el **método GET**, para indicar el tipo de envío que se está generando, se utiliza el atributo **METHOD** y, a continuación, se añade el método a utilizar.

El funcionamiento se basa en el envío al archivo PHP indicado bajo el atributo **ACTION,** de forma visible, puesto que estamos en el método GET, la información de todos los atributos que se describen dentro de las etiquetas de apertura y cierre del formulario.

```
<FORM ACTION="destino.php" METHOD="GET">
        //acciones
</FORM>
```

Código 19. Diseño formulario con método GET.

A continuación, se realiza el modelado habitual de un formulario con HTML. Es IMPRESCINDIBLE que, para la creación de cada uno de los campos que conforman los formularios, se les asocie un nombre (atributo **NAME**), puesto que esta será la forma de identificarlos en el envío de datos y en su posterior recuperación.

```
<FORM ACTION="destino.php" METHOD="GET">
Nombre: <input type="text" name="nombre"></br>
Asunto: <input type="text" name="asunto"></br>
Mensaje:<input type="text" name="mensaje"></br>
<input type="submit" value="OK" />
</FORM>
```

Código 20. Ejemplo diseño formulario con método GET.

El formulario quedaría de la forma:

Fig.5. Formulario salida código 20.

Finalmente, si vemos lo que se está enviando en la barra del navegador, vemos que GET envía por aquí todos los datos.

Fig.6. URL generada envío GET código 20.

/ 8. Formularios con método POST

El comportamiento con este tipo de método es muy similar al anterior. Ahora, en el atributo **METHOD,** se añade la palabra POST. De nuevo, se envía toda la información contenida en el formulario al archivo PHP indicado bajo el atributo **ACTION, pero ahora se hace de forma NO VISIBLE para el usuario.**

```
<FORM ACTION="destino.php" METHOD="POST">
        //acciones
</FORM>
```

Código 21. Diseño formulario con método POST.

El ejemplo visto en el apartado anterior quedará fácilmente adaptado a la nueva situación, modificando el valor del atributo METHOD como se ha descrito anteriormente.

```
<FORM ACTION="serv.php" METHOD="POST">
Nombre: <input type="text" name="nombre"></br>
Asunto: <input type="text" name="asunto"></br>
Mensaje:<input type="text" name="mensaje"></br>
<input type="submit" value="OK" />
</FORM>
```

Código 22. Ejemplo diseño formulario con método POST.

El formulario quedaría de la misma forma que en el caso anterior, el método POST no varía en nada la salida de este:

Fig.7. Formulario salida código 22.

Ahora bien, la diferencia es lo que aparece en la barra del navegador cuando se pulsa el botón de envío del formulario. Vemos que POST no envía por aquí los datos, lo único que aparece es el nombre del fichero PHP destino:

Fig.8. URL generada envío GET código 22.

/ 9. Recuperación y utilización de la información incluida en el formulario

Tras el envío de la información a través de los formularios descritos, utilizando un método u otro, los datos son recibidos por un fichero PHP en el servidor, indicado bajo el atributo **ACTION**. Para obtener el valor de estos datos, es importante tener en cuenta el nombre que se le ha asignado a cada elemento en el formulario, utilizando el atributo ***name***.

En el caso del método **GET,** la sintaxis será la siguiente:

```
<?php
$_GET['nombreDato'];
...
?>
```

Código 23. Recepción datos con método GET.

Y en el caso del método **POST**, la sintaxis será la siguiente:

```
<?php
$_POST['nombreDato'];
...
?>
```

Código 24. Recepción datos con método POST.

Fig.9. Proceso completo envío de formulario, URL y salida servidor.

Cada uno de los datos del cliente que se reciben pueden almacenarse en una variable que permitirá ser utilizada en el resto del fichero PHP. Por ejemplo, para el modelado de una consulta a una posible base de datos.

Vídeo 2. "Desarrollo-Web-Entorno-Servidor-tema4-punto10.video2"
https://bit.ly/2DC6Yt6

/ 10. Caso práctico 1: "Funciones en PHP"

Planteamiento: Utilizando como base el Caso Práctico 1 del Tema 3, ahora vamos a construir una función que reciba por parámetro dos valores, el mínimo y el máximo, utilizando el mismo código ya implementado con las sentencias condicionales y los bucles.

Realizaremos un programa en PHP que recoja en dos cajas de texto los valores y, al pulsar el botón OK, se devuelven todos los valores comprendidos entre un número mínimo y otro máximo.

Nudo: Para implementar este programa, en primer lugar, se crea un formulario que envía los datos recogidos en las cajas de texto. Una vez son recogidos, utilizando el método GET, estos se pasan como parámetro a la función *minmax*().

```
<html>
<body>
<FORM action =”Tema3Practico1.php”
METHOD=”GET”>
      Valor mínimo: <input type=”text”
name=”minimo”></br>
      Valor máximo: <input type=”text”
name=”maximo”></br>
      <input type=”submit” value=”OK” />
</FORM>
<?php
 $valorMinimo=$_GET[“minimo”];
 $valorMaximo=$_GET[“maximo”];
 minmax($valorMinimo, $valorMaximo);
 function minmax($valorMin, $valorMax){
                while ( $valorMin <= $valorMax ){
                        echo $valorMin . “</br>”;
                        $valorMin++;
                }
      }
?>
</body>
</html>
```

Código 25. Código del Caso Práctico 1 funciones en PHP.

Desenlace: En la barra del navegador accedemos a la URL del servidor en la que se haya almacenado el fichero.

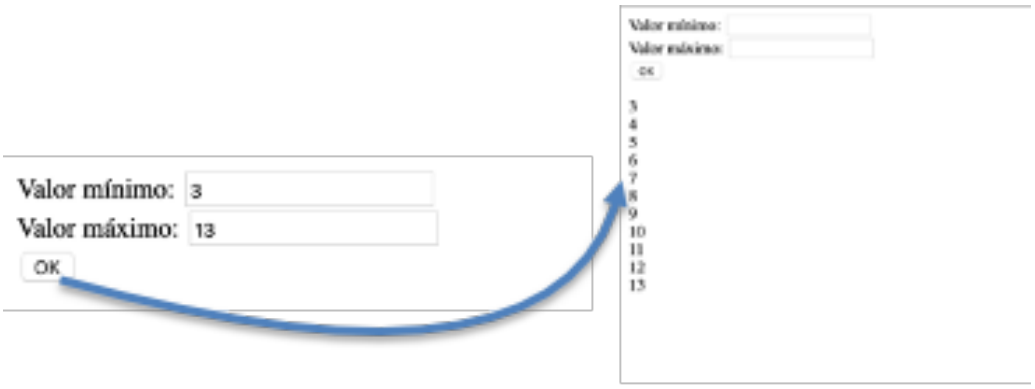

Fig.10. Código del Caso Práctico 1 funciones en PHP.

/ 11. Caso práctico 2: "Formulario de autenticación"

Planteamiento: Necesitamos desarrollar un nuevo formulario de acceso a una aplicación web que solicite al cliente su usuario y contraseña. ¿Qué tipo de diseño se propondría para este tipo de formularios?

Cuando se verifique la identidad del usuario en el servidor, se mostrará el mensaje de identidad autenticada.

Fig.11. Código del Caso Práctico 1 funciones en PHP.

Nudo: Para el envío de datos sensibles, se recomienda utilizar el método POST, puesto que los datos se envían de manera invisible al usuario, es decir, no aparecen en el navegador. Por la tanto, el formulario en HTML quedará de la forma:

```
<html>
<body>
<FORM action =”login.php” METHOD=”POST”>
      Login: <input type=”text” name=”login”></br>
      Password: <input type=”text”
name=”password”></br>
      <input type=”submit” value=”Validar” />
</FORM>
</body>
</html>
```

Código 26. Código del Caso Práctico 2 formulario con POST.

Desenlace: El código PHP utiliza algunos *if-else* para comprobar las credenciales del usuario, recogidos en una función llamada *validar*, que recibe por parámetro las variables recibidas de usuario y *password*:

```
<?php
$usuario=$_POST[“login”];
$password=$_POST[“password”];
validar($usuario, $password);
function validar($usuario, $password){
      if ($usuario == “user1” ){
       if ($password == “1234” ){
                echo “Correcto, puedes pasar!”;
       }else{
                echo “Contraseña incorrecta”;
       }
      }else{
                echo “Usuario incorrecto”;
      }
}
?>
```

Código 27. Código del Caso Práctico 2 tratamiento en PHP.

/ 12. Procesamiento de la información incluida en el formulario

En este apartado, veremos de manera íntegra el envío, procesado y utilización de la información desde el cliente web hasta el servidor PHP. Se trata de un formulario que recibe 2 valores contenidos en dos cajas de texto y, al pulsar el botón *Aceptar*, envía la petición a un servidor PHP que, en función del valor, devuelve un resultado.

En primer lugar, se modela un formulario como el siguiente. Es importante indicar correctamente el tipo de envío (en este caso, escogeremos GET), el nombre de cada uno de los elementos y, además, añadir un botón de tipo *Submit*, que acciona el envío cuando es pulsado.

```
<FORM ACTION=“calcula.php” METHOD=“GET”>
 Introduce dos valores y pulsar Aceptar
Valor 1: <input type=“text” name=“valor1”>
Valor 2: <input type=“text” name=“valor2”>
<input type=“submit” value=“Aceptar” />
</FORM>
```

Código 28. Diseño de formulario completo con GET.

Cuando esta petición se ejecuta y envía al servidor en el navegador, podemos observar algo similar a esto si se está utilizando el entorno XAMP.

```
http://www.dominio.com/directorio/file/ficheroPHP.php?var1=0&var2=0&var3=0
```

Código 29. Petición enviada con datos y método GET.

El fichero PHP se modela utilizando $_GET['valor1'] y $_GET['valor2']. De esta forma, se extrae el valor recibido en el formulario y se almacena en una variable con el mismo nombre.

```
<?php
 $valor1= $_GET['valor1'];
 $valor2= $_GET['valor2'];
 $resultado=$valor1+$valor2;
 echo $resultado
?>
```

Código 30. Código PHP extracción datos envío formulario GET código 26.

El resultado final se mostrará en la pantalla.

Fig.12. Salida código 27 y 28.

/ 13. Resumen y resolución del caso práctico de la unidad

A lo largo del este tema, hemos visto la importancia de las funciones en cualquier lenguaje de programación, puesto que, gracias a la reutilización del código, es posible agilizar el proceso de desarrollo, poniendo el foco de atención en los nuevos hitos que se deben implementar, aprovechando elementos previamente creados y probados convenientemente.

En PHP, es posible utilizar un amplio catálogo de funciones ya existentes en sus librerías, como ***rand***, ***array***, ***date*** o ***getcwd***. Pero, además, es posible que las creemos nosotros mismos.

La creación de las funciones en PHP se basa en el uso de la palabra ***function***, seguido del nombre de la función y, a continuación, entre paréntesis, se indican los parámetros que recibe por parámetro, si es que estos son requeridos.

Ahora bien, uno de los puntos de este tema, y de la asignatura en general, es el modelado de las peticiones que serán enviadas desde el cliente hasta el servidor a través del protocolo HTTP. Se distinguen dos métodos: ***GET*** y ***POST***. Ambos tienen fines similares, pero la diferencia más importante radica en que GET envía los datos en la URL de forma visible al usuario, mientras que POST no los muestra.

El envío de la petición y los datos asociados a esta se realiza en formularios que, gracias a los atributos ***ACTION*** y ***METHOD***, permiten definir el fichero PHP destino y el método de envío.

Resolución del caso práctico de la unidad

Como se ha visto a lo largo del tema, la creación de funciones permite crear elementos de código reutilizables. La implementación de funciones presenta múltiples ventajas, como la reducción de costes. Otra de las ventajas más importantes es la posibilidad de centrar la atención del desarrollo en las nuevas especificaciones de cualquier proyecto.

Si se desea diseñar un programa para evaluar que los números introducidos en algunas cajas de texto de un formulario son primos, sería conveniente implementar una función que realice esta acción y que será invocada cada vez que se introduce un nuevo número; de esta forma, se optimiza el desarrollo de una aplicación. Como podemos observar, el domino en la creación de funciones es clave para cualquier desarrollador.

Fig.13. Creación de funciones.

/ 14. Bibliografía

Ganzabal, X. (2019). *Desarrollo web en entorno servidor.* (1.ª ed.). Madrid: Síntesis.

Vara, J. M. (2012). *Desarrollo en entorno Servidor.* (1.ª ed.). Madrid: Rama.

14.1. Webgrafía

PHP. Recuperado de: https://www.php.net/manual/es/intro-whatis.php

Programación Lado Servidor. Recuperado de: https://developer.mozilla.org/es/docs/Learn/Server-side