



Funciones de PHP





Tabla de contenido

Descripción material del programa	1
Mapa conceptual	3
1. Las funciones de PHP.....	4
Funciones definidas por el usuario.....	4
Nombres de las funciones.....	5
Llamar una función	6
2. Argumentos de funciones	8
Paso de argumentos por valor	9
Paso de argumentos por referencia.....	9
Argumentos con valores predeterminados.....	10
3. Devolviendo valores	13
4. Bibliotecas de funciones propias	16
5. Funciones internas (incluidas)	24
Referencias	28





Descripción material del programa

Este material está diseñado para facilitar el proceso de aprendizaje, por esta razón, los contenidos buscan que el aprendiz se apropie del conocimiento que realmente necesita para desarrollar sus habilidades y que lo haga de una forma sencilla y organizada; además de la lectura general cuenta con algunos apartes que contienen: frases o datos para recordar, segmentos de código, consejos y advertencias, estos elementos se destacan por las siguientes convenciones gráficas:

Ícono	Elemento importante
	Frases o datos para recordar: son extraídas de la lectura previa.
	Segmentos de código: pueden tomarse como base para los ejercicios propuestos.
	Consejos: buenas prácticas para el proceso de desarrollo.
	Advertencias: lo que no se recomienda hacer dentro de los procesos de desarrollo.

Fuente de imágenes: SENA





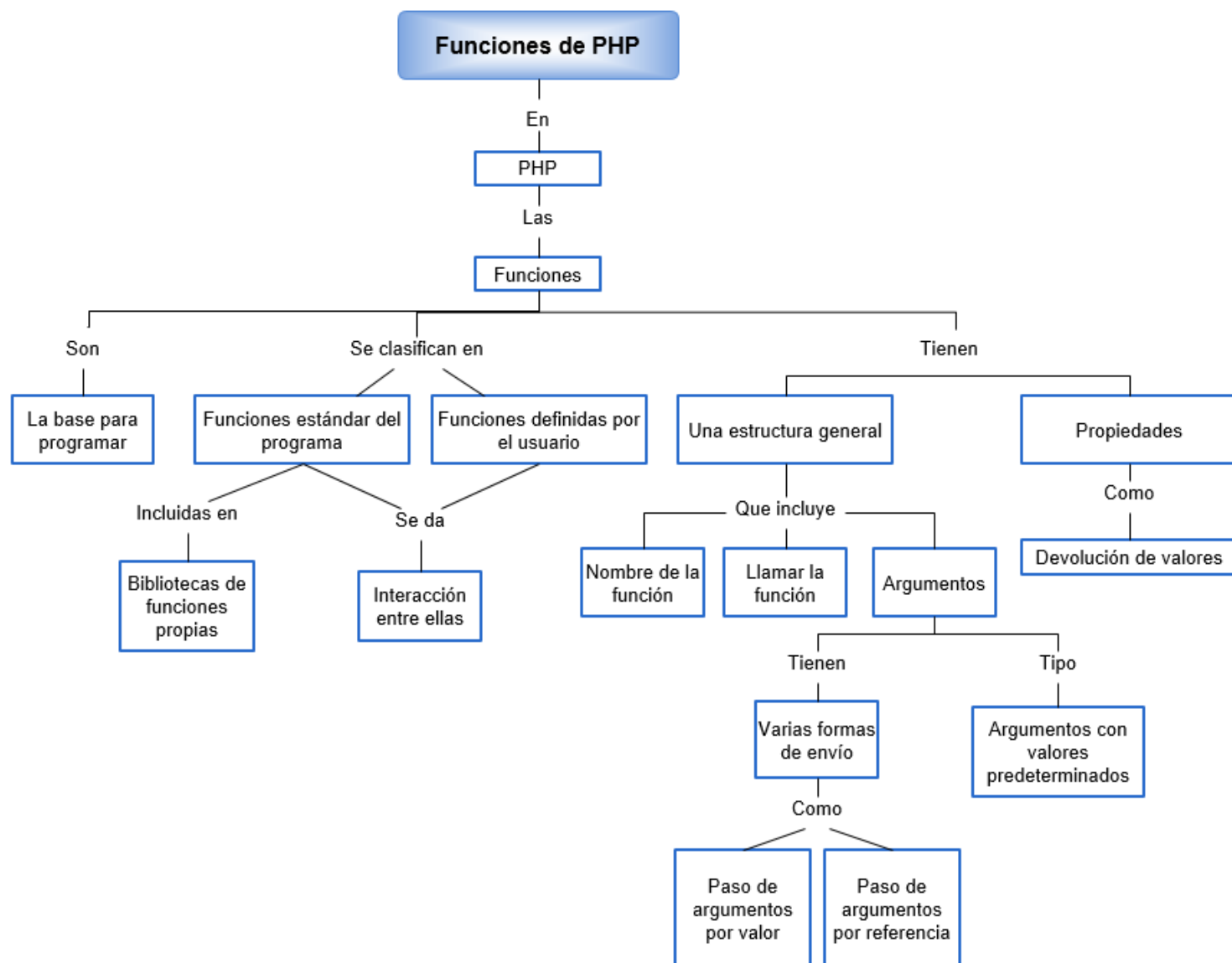
La mayor parte de los segmentos de código que aparecen en este material de formación los encuentran como archivos .php que se pueden descargar del material complementario en la siguiente ruta: Materiales del programa / Materiales de apoyo / Documentos complementarios / Documentos complementarios: Actividad de aprendizaje 3

Para usar estos segmentos de código solo se necesita copiarlos y pegarlos en el editor o entorno que esté usando para el desarrollo, o en el caso de que se encuentren etiquetados con la ruta del archivo puede abrirse directamente desde el editor o entorno. Los segmentos de código están comentados (usando los comentarios de cada lenguaje: HTML y PHP) para facilitar su comprensión y uso, dichos comentarios pueden ser modificados o retirados de ser necesario.



Mapa conceptual

En el mapa conceptual que se comparte a continuación, se evidencia la interrelación temática del contenido que se plantea en este material de formación:





1. Las funciones de PHP

PHP fue originalmente diseñado para programar de forma estructurada, aunque las versiones actuales tienen un excelente soporte para la Programación Orientada a Objetos (OOP). Las funciones de PHP siguen siendo su pilar principal y su uso para desarrollar de forma estructurada es aún muy popular; es por ello que se puede afirmar que es un lenguaje multiparadigma. En esta actividad de aprendizaje se tratará el tema de funciones empezando desde la posibilidad que se tiene como desarrollador de crear funciones propias para dar solución a las necesidades de la aplicación y luego se dará un vistazo al tema de las funciones de la biblioteca de PHP que hacen más fácil el proceso de desarrollo, ya que en programación existen gran cantidad de soluciones a problemas comunes.

Funciones definidas por el usuario

Las funciones no son otra cosa que un segmento de código que se programa de forma tal que pueda ser usado varias veces dentro de la aplicación que se está desarrollando, para esto es necesario utilizar una lógica que adapte cada una de estas rutinas a las diferentes necesidades que se tienen de ellas, es allí donde entran en juego los argumentos de una función (las funciones pueden ser creadas con o sin argumentos, pero normalmente una buena función que sea altamente reutilizable requiere argumentos), puesto que permiten que la función trabaje de formas distintas según la información que reciba. La sintaxis básica de una función es la siguiente:

```
function    nombreFuncion($argumento_1,    $argumento_2,    ...,
$argumento_n)
{
    sentencias a ejecutar cada vez que la función sea llamada
    ...
    return $valorDevuelto;
}
```

Se puede ver una estructura de control especial que es la sentencia `return`, básicamente tiene dos funciones, devuelve el control del programa al punto desde el que se hizo la llamada a la función y devuelve un dato que fue procesado por la función que puede ser almacenado en una variable, esta sentencia así como la forma en que pueden aprovecharse los argumentos son temas que se tratarán más adelante.





Dentro de una función se pueden escribir todas las sentencias válidas de PHP, expresiones, condicionales, ciclos e incluso dentro de una función se pueden declarar otras.


Las funciones pueden ser escritas dentro de un código en cualquier lugar sin importar si se hace en un punto antes o después de aquel en el que se le llama, esto es cierto para todas las funciones excepto para aquellas que se escriben dentro de una estructura condicional, caso en el cual dicha función solo estará disponible cuando la condición se cumpla y el flujo del programa llegue al punto donde está declarada la función.

Nombres de las funciones

Los nombres de las funciones pueden contener caracteres alfanuméricos y de subrayado, el primer carácter del nombre solo puede ser una letra (a-z A-Z) o un carácter de subrayado (_) nunca un número, seguido de cualquier cantidad de caracteres alfanuméricos y de subrayado como se necesiten.

Aunque es posible utilizar los caracteres de subrayado en los nombres de las funciones, los estándares de codificación de Zend Framework no los permiten y los números si son permitidos pero no se aconsejan en la mayoría de los casos.

Los nombres de las funciones cuando son de una sola palabra, solo deben usar letras minúsculas, cuando consisten en más de una palabra las siguientes después de la primera deben empezar con una letra mayúscula, lo cual se conoce como “notacionCamello”. (Zend Technologies Ltd., s.f.)

	<p>“La verbosidad es generalmente aconsejada, esto quiere decir que se deben tener nombres de variables que identifiquen lo más completamente posible su propósito y comportamiento, y pueden ser tan largos como se necesite hasta donde sea práctico claro está” (Zend Technologies Ltd., s.f.).</p>
Fuente: SENA	

Algunos nombres de funciones válidos serían:

```
sumaEnterosPostivos()  
eliminaEspaciosCadenas()  
cuentaLetrasMayusculasCadenas()
```





Fuente: SENA

Los nombres de las funciones cuando son de una sola palabra, solo deben usar letras minúsculas, cuando consisten en más de una palabra las siguientes después de la primera deben empezar con una letra mayúscula, lo cual se conoce como “notacionCamello”.

Llamar una función

Nombrar y programar una función no sirve de nada si no se la invoca en algún punto del programa, para que el flujo de control ejecute cualquier función esta debe ser llamada, cuando la función no retorna ningún tipo de valor, el llamado puede hacerse simplemente poniendo el nombre y los argumentos de las funciones (si la función no tiene argumentos simplemente se ponen los paréntesis frente al nombre de la función), a continuación se muestra un ejemplo:

```
sentencias antes de la función
...
nombreFuncion ($arg1, $arg2, ..., $argn);
sentencias después de la función
...
```

Por otro lado, si la función retorna un valor, normalmente se genera una expresión que implica declarar una variable y/o inicializarla asignándole el valor de la función así:

```
sentencias antes de la función
...
$variableQueRecibeValorDevuelto = nombreFuncion ($arg1,
$arg2, ..., $argn);
sentencias después de la función
...
```

También en este último caso se puede simplemente imprimir el valor retornado por la función así:

```
sentencias antes de la función
...
```





echo nombreFuncion (\$arg1, \$arg2, ..., \$argn);
sentencias después de la función
...

Ejemplo 1:



```
<!DOCTYPE html>
<html>
  <head>
    <title>Unidad 3 - Ejemplo 1</title>
    <meta http-equiv="Content-Type"
      content="text/html; charset=ISO-8859-1" />
  </head>
  <body>

    <?php
    /* En este programa se presenta el funcionamiento de una
    función
    * simple, creada por el usuario
    */
    /* Intencionalmente se escribió primero el llamado a la
    función antes que
    * su definición con el fin de demostrar que no importa
    donde se haga
    * el llamado de una función, si antes o después de si
    declaración
    */
    imprimeTabla();
    /* La función imprimeTabla() genera una tabla HTML de 3
    filas por 4
    * columnas con una fila de encabezado que contiene los
    títulos de
    * las columnas, la función no recibe argumentos ni
    devuelve valores
    */
    function imprimeTabla() {
    ?>
      <table border ="1">
        <thead>
          <td>Nombre</td>
          <td>Direcci&oacute;n</td>
          <td>Tel&eacute;fono</td>
          <td>Fecha de Nacimiento</td>
        </thead>
        <tr>
          <td>&nbsp;</td>
          <td>&nbsp;</td>
          <td>&nbsp;</td>
          <td>&nbsp;</td>
        </tr>
        <tr>
          <td>&nbsp;</td>
          <td>&nbsp;</td>
          <td>&nbsp;</td>
          <td>&nbsp;</td>
        </tr>
        <tr>
          <td>&nbsp;</td>
          <td>&nbsp;</td>
          <td>&nbsp;</td>
          <td>&nbsp;</td>
        </tr>
      </table>
    }
```



```
<td>&emsp;</td>
<td>&emsp;</td>
</tr>
</table>
<?php
}
?>
</body>
</html>
```

Fuente: SENA

Descargue el segmento anterior del código como archivo .php del material complementario de este programa de formación en la siguiente ruta:

Materiales del programa / Materiales de apoyo / Documentos complementarios / Documentos complementarios: Actividad de aprendizaje 3 / Ejemplo 1

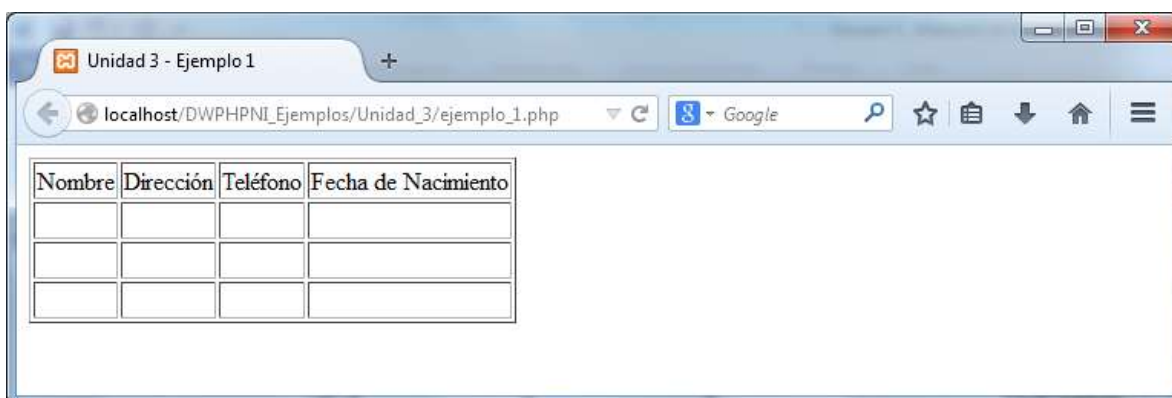


Figura 1. Ejecución del ejemplo 1

Fuente: SENA

Ejercicio 1:

Modifique el código del ejemplo anterior escribiendo un nuevo llamado a la función después de su declaración, esto hará que la función se ejecute dos veces y por lo tanto aparecerán dos tablas en el navegador.

2. Argumentos de funciones

En PHP las funciones tienen argumentos, como ya se dijo antes, son los argumentos los que permiten generar funciones con altas alternativas de reutilizables que se aplican para solucionar diferentes posibilidades de un mismo problema. Los argumentos son información y en PHP las funciones pueden recibir uno o varios argumentos de cualquier tipo, en el caso de tener varios argumentos estos deben ir separados por comas y se evaluarán desde la izquierda hacia la



derecha, es decir que el primer argumento que se lee es el que está a la izquierda y el último el que está a la derecha.

Como ya se dijo, cuando se declara una función se definen todos los argumentos que la función va a recibir, estas definiciones se convierten en los contenedores de información de la función y serán automáticamente variables que se pueden utilizar dentro del ámbito de la función; cuando se hace el llamado de la función será necesario enviarle los datos que van a llenar dichos contenedores (de lo contrario se generará un error ya que la función no puede ser invocada sino se le envía la información que requiere para ejecutarse), estos datos pueden ser enviados de varias formas:

Paso de argumentos por valor: esta es la forma por defecto en que se pasan los datos, simplemente para cada argumento solicitado por la función, se puede poner un valor literal o una variable que contiene el dato así:

Declaración de la función:

```
function  almacenaNombreEdadDepartamento  ($nombre,  $edad,
$departamento)
{
    sentencias de la función
    ...
}
```

Llamado de la función:

```
almacenaNombreEdadDepartamento ($nombre, 34, "Contabilidad");
```

Paso de argumentos por referencia: como se analizó en la actividad de aprendizaje 2, cuando se hace una asignación de variables por referencia y se hace una modificación a la variable que contiene la referencia, automáticamente se modifica el valor de la variable referida, esto es útil cuando se trabaja con funciones ya que se puede usar en los argumentos de forma tal que en ellos se reciba no un valor sino la referencia a una variable cuyo valor se quiere modificar mediante la función de la siguiente manera:

Declaración de la función:

```
function modificaEdad (&$edad)
{
```





```
    sentencias de la función que modifican $edad
    ...
}
```

Llamado de la función:

Sentencias antes de la función

```
...
$edadCliente
modificaEdad ($edadCliente);
```

Argumentos con valores predeterminados: en algunos casos puede ser útil dejar valores predeterminados en las funciones, de forma tal que al llamarlas, estos argumentos puedan dejarse en blanco y aun así la función se ejecuta correctamente usando los valores por defecto, para lograr esto se debe dar un valor predeterminado a estos argumentos en el momento en el que se declara la función, se debe tener en cuenta que estos argumentos tienen que estar a la derecha de los que no tengan valores predeterminados, pues de lo contrario no podrían obviarse al llamar la función.

Declaración de la función:

```
function almacenaDatos ($nombre, $edad, $ciudad = "Bogotá",
$casado = FALSE)
{
    sentencias de la función
    ...
}
```

Llamados posibles de la función:

```
almacenaDatos ("Juan Pérez", 35);
almacenaDatos ("Juan Pérez", 35, "Cali");
almacenaDatos ("Juan Pérez", 35, "Cali", TRUE);
```





Ejemplo 2:

11100001
10011010
00110110
10001010

```

<!DOCTYPE html>
<html>
  <head>
    <title>Unidad 3 - Ejemplo 2</title>
    <meta http-equiv="Content-Type"
          content="text/html; charset=ISO-8859-1" />
  </head>
  <body>
    <?php
    /* En este programa se presenta el funcionamiento de una
función
    * que recibe argumentos
    */
    /* La función imprimeTabla() genera una tabla HTML de n
filas donde
    * la cantidad de filas está definida por el argumento
$filas
    * pero el argumento tiene un valor predefinido, así que
si el
    * usuario de la función no da un valor se generará una
tabla con 3
    * filas por 4 columnas con una fila de encabezado que
contiene los títulos de
    * las columnas, la función no recibe argumentos ni
devuelve valores
    */
function imprimeTabla($filas = 3) {
    ?>
    <table border ="1">
      <thead>
        <td>Nombre</td>
        <td>Direcci&oacute;n</td>
        <td>Tel&eacute;fono</td>
        <td>Fecha de Nacimiento</td>
      </thead>
    <?php
    /* Mediante un ciclo for se pueden crear la cantidad
de filas
    * que se requieran con base en el argumento $filas,
como puede
    * verse esto hace el código más eficiente ya que se
requieren menos
    * líneas de código que en el ejemplo anterior que
hacia algo similar
    */
    for ($i = 1; $i <= $filas; $i++) {
      ?>
      <tr>
        <td>&nbsp;</td>
        <td>&nbsp;</td>
        <td>&nbsp;</td>
        <td>&nbsp;</td>
      </tr>
    <?php
    }
    ?>
  </table>
<?php

```



```
}
/* Se llama la función sin poner argumento por lo que se
tomará el valor
* predefinido 3 que será el número de filas de la tabla
*/
imprimeTabla();
/* Se llama la función poniendo como argumento 5 que sepa el
número de
* filas de la tabla
*/
imprimeTabla(5);
?>
</body>
</html>
```

Fuente: SENA

Descargue el segmento anterior del código como archivo .php del material complementario de este programa de formación en la siguiente ruta:

Materiales del programa / Materiales de apoyo / Documentos complementarios / Documentos complementarios: Actividad de aprendizaje 3 / Ejemplo 2

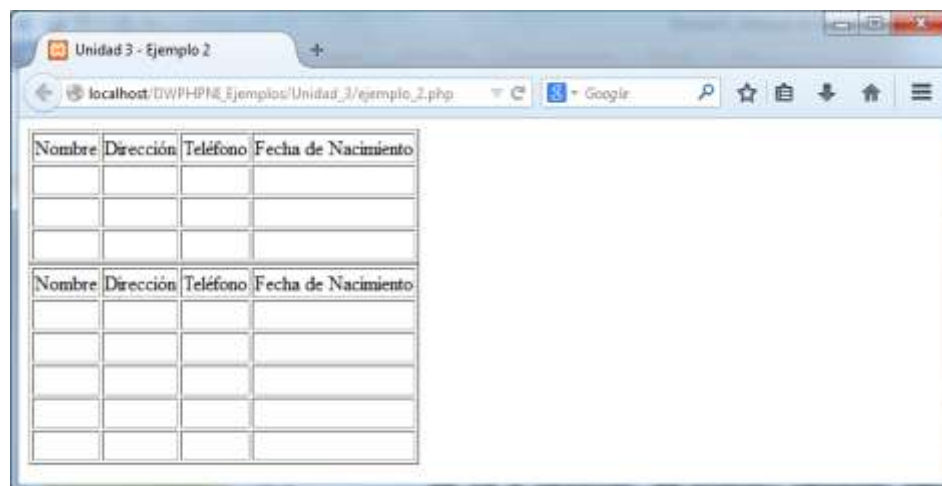


Figura 2. Ejecución del ejemplo 2

Fuente: SENA

Ejercicio 2:

Cree y utilice una función que reciba un argumento por referencia y otro con valor predeterminado, recuerde que el parámetro con valor predeterminado debe ir a la derecha en la declaración de la función.





Fuente: SENA

Los argumentos opcionales no deben ponerse nunca a la izquierda de los demás, ya que no tendrían ningún sentido pues nunca podrían dejarse en blanco, no se puede hacer un llamado de una función de la forma: `funcion(, , $a, $b);`

3. Devolviendo valores

Las funciones pueden devolver valores luego de realizar el procesamiento de los datos, para esto se utiliza la instrucción `return`, que debe ir precedida del valor que se va a devolver, se pueden devolver valores literales o contenidos en variables y cualquier tipo de dato, incluidos los arreglos; las funciones solo pueden devolver un dato, cuando es literal o si está contenido en una variable simple, pero si se quiere simular la devolución de varios datos se pueden utilizar los array, se almacenan en este todos los datos debidamente indexados y luego se retorna el array. Solo puede haber una ejecución de una instrucción `return` por función, lo cual quiere decir que si se necesita tener varias instrucciones `return` pueden tenerse condicionadas de forma tal que al final solo se ejecute una, si no se tienen condicionadas sino puestas linealmente dentro del código se ejecutará solo la primera instrucción ya que luego de `return` la función termina su ejecución y devuelve el control al punto de llamado de la función. Las instrucciones `return` pueden usarse así:

Declaración de la función:

```
function ordenaDatos ($datos)
{
    sentencias que ordenan lo contenido en $datos
    ...
    return $datosOrdenadosDentroDeFuncion; //Este puede ser
    un arreglo
}
```

Llamado de la función:

```
$datosOrdenados = ordenaDatos ($datosAOrdenar);
```





Varias instrucciones return condicionadas

Declaración de la función:


```
function ordenaDatos ($datos, $orden = "ASC")
{
    if ($orden == "ASC") {
        sentencias que ordenan Ascendentemente
        ...
        return $datosOrdenadosDentroDeFuncion;

    } elseif ($orden == "DESC") {
        sentencias que ordenan Descendentemente
        ...
        return $datosOrdenadosDentroDeFuncion;
    }
}
```

Llamado de la función:

```
$datosOrdenados = ordenaDatos ($datosAOrdenar, "DESC");
```

Ejemplo 3:

	<pre><!DOCTYPE html> <html> <head> <title>Unidad 3 - Ejemplo 3</title> <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" /> </head> <body> <?php /* En este programa se presenta el funcionamiento de una función * que devuelve datos luego de su ejecución */ /* encuentraValorArreglo() es una función que recibe un arreglo * y un valor para buscar el valor dentro del arreglo enviado * Esta función ya está programada en la biblioteca de PHP pero * la codificamos aquí a modo de ejemplo de una función que retorna * valores, al función retorna TRUE si encuentra el valor en el arreglo * y FALSE si no lo encuentra */ function encuentraValorArreglo(\$arreglo, \$valorBuscado) {</pre>
---	--



```
        foreach ($arreglo as $valor) {
            if ($valor == $valorBuscado) {
                return TRUE;
            }
        }
        return FALSE;
    }
    // Se declara un arreglo con datos para poder buscar en el
    $arregloAnimales = array(
        "Perro",
        "Gato",
        "Liebre",
        "Conejo",
        "Vaca",
        "Lobo",
        "Abeja",
        "Oveja",
        "Pollo"
    );
    //Se declaran dos variables con datos a ser buscados en el
    arreglo
    $animalQueEsta = "Vaca";
    $animalQueNoEsta = "Leon";
    /* Se usa el valor retornado por la variable como
    expresión para
    * un condicional en este caso el valor a buscar si está
    dentro del
    * arreglo por lo tanto la función retornará TRUE lo que
    hará que
    * la condición se cumpla
    */
    if (encuentraValorArreglo($arregloAnimales,
    $animalQueEsta)) {
        echo "$animalQueEsta si se encuentra en el arreglo <br
    />";
    }
    /* En este caso el valor a buscar no estará en el arreglo,
    por ello
    * debe utilizarse una negación pues la función retornará
    FALSE
    * que al ser negado se convierte en TRUE y hace que la
    condición
    * se cumpla
    */
    if (!encuentraValorArreglo($arregloAnimales,
    $animalQueNoEsta)) {
        echo "$animalQueNoEsta no se encuentra en el arreglo";
    }
    ?>
</body>
</html>
```

Fuente: SENA

Descargue el segmento anterior del código como archivo .php del material complementario de este programa de formación en la siguiente ruta:
Materiales del programa / Materiales de apoyo / Documentos complementarios / Documentos complementarios: Actividad de aprendizaje 3 / Ejemplo 3



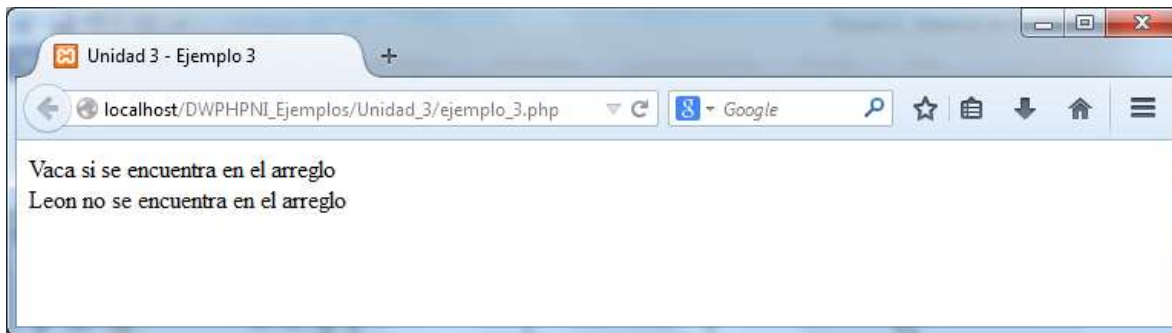


Figura 3. Ejecución del ejemplo 3

Fuente: SENA

4. Bibliotecas de funciones propias

Cuando se está creando una aplicación que requiere cierta complejidad normalmente se necesita la repetición de rutinas en diferentes puntos del sistema, y para ello se crean funciones que puedan facilitar este trabajo, pero en PHP las funciones que se crean solo son llamadas dentro del archivo .php, lo cual implicaría que a pesar de que la función pueda servir para otros módulos del sistema desarrollados en archivos diferentes, se tendría que reescribir el código en cada archivo donde se necesite, esto no haría nada eficiente el código y se perdería la funcionalidad de escribir funciones. Para ello PHP cuenta con la posibilidad de incluir o requerir un archivo .php dentro de otro, de esta manera se pueden crear archivos .php que contengan únicamente funciones que van a ser vinculadas con los archivos .php donde se necesiten las funciones y se podrán utilizar como si estuvieran escritas allí mismo.

Los archivos que contengan las bibliotecas de funciones no necesitan código HTML, para cuestiones de organización se pueden separar las funciones en diferentes archivos de acuerdo con su utilidad, por ejemplo poner en un archivo las funciones para tratamiento de cadenas, en otro las funciones para conexiones con bases de datos, en otro las que se usen para procesar arreglos, entre otros, esto evita que se cargue un archivo con una inmensa cantidad de funciones en todos los demás archivos del aplicativo, lo cual podría bajar la eficiencia del servidor.

Un ejemplo de un archivo .php con funciones sería:

```
funcionesArreglos.php
<?php
funcionUno ($arg1, $arg2, ..., $argn)
{
```





```
    sentencias de la funcionUno
    ...
}
funcionDos ($arg1, $arg2, ..., $argn)
{
    sentencias de la funcionDos
    ...
}
funcionTres ($arg1, $arg2, ..., $argn)
{
    sentencias de la funcionTres
    ...
}
funcionN ($arg1, $arg2, ..., $argn)
{
    sentencias de la funcionN
    ...
}
...
```

En el ejemplo anterior se obvió la etiqueta de cierre de php `¿>`, esto es una buena práctica de desarrollo que impide al momento de vincular este archivo con otro, se incluyan posibles espacios en blanco en la respuesta.



Fuente: SENA

Los archivos que contengan las bibliotecas de funciones no necesitan código HTML, para cuestiones de organización se pueden separar las funciones en diferentes archivos de acuerdo con su utilidad.

Las funciones del sistema que permiten vincular archivos .php son:

include(): la sentencia `include` incluye y evalúa el archivo especificado.

Si una ruta es definida — ya sea absoluta (comenzando con una letra de unidad o `\` en Windows o `/` en sistemas Unix/Linux) o relativa al directorio





actual (comenzando con `.` o `..`) — el `include_path` será ignorado por completo. Por ejemplo, si un nombre de archivo comienza con `../`, el intérprete buscará en el directorio padre para encontrar el archivo solicitado.

Cuando se incluye un archivo, el código que contiene hereda el ámbito de las variables de la línea en la cual ocurre la inclusión. Cualquier variable disponible en esa línea del archivo que hace el llamado, estará disponible en el archivo llamado, desde ese punto en adelante. Sin embargo, todas las funciones y clases definidas en el archivo incluido tienen el ámbito global.

`include_once()`: incluye y evalúa el fichero especificado durante la ejecución del script. Es un comportamiento similar al de la sentencia `include`, siendo la única diferencia que si el código del fichero ya ha sido incluido, no se volverá a incluir. Como su nombre lo indica, será incluido solo una vez.

`Include_once` puede ser usado en casos donde el mismo fichero podría ser incluido y evaluado más de una vez durante una ejecución particular de un script, así que en este caso, puede ayudar a evitar problemas como la redefinición de funciones, reasignación de valores de variables, etc.

`require()`: es idéntico a `include` excepto que en caso de fallo producirá un error fatal de nivel `E_COMPILE_ERROR`. En otras palabras, éste detiene el script mientras que `include` solo emitirá una advertencia (`E_WARNING`) lo cual permite continuar el script.

`require_once()`: es idéntica a `require` excepto que PHP verificará si el archivo ya ha sido incluido y si es así, no se incluye (`require`) de nuevo. (The PHP Group, s.f.)





En cuanto al uso de las sentencias `include` o `require` hay diferentes opiniones, algunos expertos creen que lo mejor es utilizar siempre la opción `require`, ya que si el código que se está escribiendo depende absolutamente de lo que se contiene en el archivo a vincular será referible que el script se detenga (así sea abruptamente), a que continúe y genere errores lógicos, pero otros dicen que un error fatal de terminación del programa como el que podría generar `require` al no encontrar el archivo vinculado podría crear peores desastres en el funcionamiento del aplicativo en general, así que es necesario que en la lógica del sistema que se está desarrollando se evalúe muy bien cuál de las dos opciones es la mejor. En lo que sí están de acuerdo la mayoría, es que son mejores las versiones `include_once` y `require_once`, teniendo en cuenta que evitan la instanciación múltiple de los archivos a vincular, cuando se necesitan en varios puntos de la aplicación mejorando un poco la eficiencia del servidor.

Para el caso del archivo de funciones de ejemplo `funcionesArreglos.php`, suponiendo que se encuentra en la misma carpeta que el archivo que lo está llamando, la forma de incluirlo sería cualquiera de las siguientes:

```
include(funcionesArreglos.php);  
include_once(funcionesArreglos.php);  
require(funcionesArreglos.php);  
require_once(funcionesArreglos.php);
```



Fuente: SENA

Son mejores las versiones `include_once` y `require_once`, teniendo en cuenta que evitan la instanciación múltiple de los archivos a vincular, cuando se necesitan en varios puntos de la aplicación mejorando un poco la eficiencia del servidor.



Ejemplo 4:



```
<!DOCTYPE html>
<html>
  <head>
    <title>Unidad 3 - Ejemplo 4</title>
    <meta http-equiv="Content-Type"
          content="text/html; charset=ISO-8859-1" />
  </head>
  <body>
    <?php
    /* En este programa se muestra cómo se pueden usar
    bibliotecas de
      * funciones creadas por el usuario, solo hace falta
    recopilar
      * las funciones en un archivo .php y luego incluirlo o
    requerirlo
      * dentro del archivo donde se van a requerir las
    funciones
      */

    /* La función require_once() permite llamar el archivo en
    este caso
      * puesto que está dentro de la misma carpeta se hace de
    la siguiente
      * manera
      */
    require_once './ejemplo_4_Biblioteca.php';

    /* Vamos a inicializar un arreglo que contiene los datos
    de un listado
      * de personas
      */
    $listadoAmigos = array(
      array(
        "nombre" => "Juan Perez",
        "direccion" => "Clle. 3 # 25 - 40",
        "telefono" => "2345674",
        "fechaNacimiento" => "12/03/2000",
        "colorFavorito" => "Azul"
      ),
      array(
        "nombre" => "Lola Fuentes",
        "direccion" => "Cra. 4 # 12 - 18",
        "telefono" => "2345674",
        "fechaNacimiento" => "07/12/1980",
        "colorFavorito" => "Verde"
      ),
      array(
        "nombre" => "Pablo Reyes",
        "direccion" => "Cra. 16 # 125 - 15",
        "telefono" => "3456271",
        "fechaNacimiento" => "03/07/1987",
        "colorFavorito" => "Amarillo"
      ),
    );

    /* Lo que se requiere es mostrar todos los datos del
    arreglo
      * ordenados en una tabla pero que además el color
    favorito se busque
```




	<pre> * en un arreglo que contenga el color su significado y que en * una columna de la tabla se ponga el significado del color * la siguiente función hace todo eso, pero no está dentro de este * mismo archivo sino en el archivo ejemplo_4_Biblioteca.php * que fue requerido al inicio */ muestraListadoTabla(\$listadoAmigos); ?> </body> </html> </pre>
--	---

Fuente: SENA

Descargue el segmento anterior del código como archivo .php del material complementario de este programa de formación en la siguiente ruta:

Materiales del programa / Materiales de apoyo / Documentos complementarios / Documentos complementarios: Actividad de aprendizaje 3 / Ejemplo 4

Ejemplo 4 - Biblioteca:

	<pre> <?php /* Esta es una biblioteca de funciones que están diseñadas para mostrar * los datos de un arreglo que contiene un listado de personas y ciertos datos * específicos, y permite buscar uno de los datos que es el color favorito * en un arreglo que contiene los colores y su significado mostrando en la tabla * el significado del color favorito de cada persona del listado */ /* La función encuentraSignificado() busca el color que recibe como parámetro * en un arreglo inicializado dentro de la misma función que contiene * los colores y sus significados y devuelve el significado del color recibido */ function encuentraSignificadoColor(\$colorBuscado) { /* Inicialización del arreglo que contiene los colores y los significados * respectivos de cada color */ \$coloresSignificado = array(array("color" => "Amarillo", "significado" => "Alegria, riqueza"), array("color" => "Verde", </pre>
---	--





```

        "significado" => "Esperanza"
    )
);

/* Se recorre el arreglo con un ciclo foreach y si se
encuentra el color que
* se recibió como parámetro se devuelve el dato del
significado
*/
foreach ($coloresSignificado as $colorSignificado) {
    if ($colorSignificado['color'] == $colorBuscado) {
        return $colorSignificado['significado'];
    }
}
/* Si después de recorrer todo el arreglo no se encuentra el
color recibido
* se devuelve el mensaje "No se encuentra el significado"
*/
return "No se encuentra el significado";
}

/* La función muestraListadoTabla() imprime una tabla HTML en la
que muestra
* todos los datos del arreglo que recibe como parámetro, esta no
es una función
* muy flexible ni reutilizable, ya que solo muestra los datos de
un
* arreglo que tenga una estructura muy específica
*/

function muestraListadoTabla($listado) {
    ?>
    <table border ="1">
        <thead>
            <td>Nombre</td>
            <td>Direcci&oacute;n</td>
            <td>Tel&eacute;fono</td>
            <td>Fecha de Nacimiento</td>
            <td>Color Favorito</td>
            <td>Significado</td>
        </thead>
        <?php
        /* Mediante un ciclo for se pueden crear la cantidad de filas
        * que se requieran con base en el argumento $filas, como
puede
        * verse esto hace el código más eficiente ya que se requieren
menos
        * líneas de código que en el ejemplo anterior que hacia algo
similar
        */
        foreach ($listado as $registro) {
            ?>
            <tr>
                <td><?php echo $registro['nombre']; ?></td>
                <td><?php echo $registro['direccion']; ?></td>
                <td><?php echo $registro['telefono']; ?></td>
                <td><?php echo $registro['fechaNacimiento']; ?></td>
                <td><?php echo $registro['colorFavorito']; ?></td>
                <td>
                    <?php
                    /* En este punto se hace el llamado a la función
                    * encuentraSignificadoColor() que está en esta

```





```

        misma
        * biblioteca, y que retorna el significado del
color
        * favorito o un mensaje en caso de no encontrarlo
        */
    echo
encuentraSignificadoColor($registro['colorFavorito']);
    ?>
</td>
</tr>
<?php
}
?>
</table>
<?php
}

```

Fuente: SENA

Descargue el segmento anterior del código como archivo .php del material complementario de este programa de formación en la siguiente ruta:

Materiales del programa / Materiales de apoyo / Documentos complementarios / Documentos complementarios: Actividad de aprendizaje 3 / Ejemplo 4 - Biblioteca

Nombre	Dirección	Teléfono	Fecha de Nacimiento	Color Favorito	Significado
Juan Perez	Clle. 3 # 25 - 40	2345674	12/03/2000	Azul	No se encuentra el significado
Lola Fuentes	Cra. 4 # 12 - 18	2345674	07/12/1980	Verde	Esperanza
Pablo Reyes	Cra. 16 # 125 - 15	3456271	03/07/1987	Amarillo	Alegria, riqueza

Figura 4. Ejecución del ejemplo 4

Fuente: SENA

Ejercicio 3:

Con base en el ejemplo anterior, cree una función que muestre en una tabla cualquier arreglo de tipo matriz, sin importar el número de filas o columnas que tenga, en este caso no se requiere que la tabla tenga una fila en la que estén los



títulos de cada columna, solo que se muestren los datos ordenadamente. Al igual que en el ejemplo la función está en un archivo aparte.

5. Funciones internas (incluidas)

PHP se estandariza con muchas funciones y construcciones. También existen funciones que necesitan extensiones específicas de PHP compiladas, si no, aparecerán errores fatales "undefined function" ("función no definida"). Por ejemplo, para usar las funciones de `image` tales como `imagecreatetruecolor()`, PHP debe ser compilado con soporte para GD. O para usar `mysql_connect()`, PHP debe ser compilado con soporte para MySQL. Hay muchas funciones de núcleo que están incluidas en cada versión de PHP, tales como las funciones de string y de variable. Una llamada a `phpinfo()` o `get_loaded_extensions()` mostrará las extensiones que están cargadas en PHP. (The PHP Group, s.f.)

El manual oficial de PHP disponible en el siguiente enlace <http://php.net/manual/es/functions.internal.php>, complementa conceptos en relación a la descripción y uso de funciones PHP. En la sección **Cómo interpretar la definición de una función**, se explica la forma correcta de comprender un prototipo de una función. En este sentido, es importante comprender lo que devuelve una función o si una función trabaja directamente con un valor pasado. Por ejemplo, `str_replace()` devolverá la cadena modificada mientras que `usort()` funciona con la variable actual pasada.

Cada página del manual también tiene información específica para cada función, como información sobre parámetros de funciones, cambios de comportamiento, valores devueltos en caso de éxito o fallo, e información de disponibilidad. Conocer estas importantes diferencias (a menudo imperceptibles) es crucial para escribir código de PHP correcto. (The PHP Group, s.f.)





Ejemplo 5:

11100001
10011010
00110110
10001010

```
<!DOCTYPE html>
<html>
  <head>
    <title>Unidad 3 - Ejemplo 5</title>
    <meta http-equiv="Content-Type"
          content="text/html; charset=ISO-8859-1" />
  </head>
  <body>
    <p>
      <h3>Ejemplo de la función date()</h3>
      <?php
        /* En este programa se muestra el uso de algunas funciones
         * de la biblioteca de PHP
         */
        /* La función date() devuelve la fecha y hora del sistema
         * requiere un parámetro de tipo cadena que indica la
         forma en la
         * que se debe mostrar la información devuelta
         */
        echo date('D, d M Y H:i:s');
      ?>
    </p>
    <p>
      <h3>Ejemplo de la función print_r()</h3>
      <?php
        /* Se inicializa un arreglo para poder utilizar algunas
         funciones
         * que nos permiten procesarlos
         */
        $arregloAnimales = array(
          "Perro",
          "Gato",
          "Liebre",
          "Conejo",
          "Vaca",
          "Lobo",
          "Abeja",
          "Oveja",
          "Pollo"
        );
        /* La función print_r() muestra en el navegador todo el
         * contenido del arreglo
         */
        print_r($arregloAnimales);
      ?>
    </p>
    <p>
      <h3>Ejemplo de la función asort()</h3>
      <?php
        /* La función asort() ordena el arreglo que se pasa como parámetro
         * pero no usa la instrucción return, ya que recibe el arreglo
         como
         * parámetro por referencia, es decir, que el arreglo ordenado
         * queda almacenado en el mismo arreglo que se envió, por esa
         razón si
         * volvemos a imprimir el mismo arreglo se muestra ahora ordenado,
         * pero podremos notar que la función solo reordena las posiciones
         * de los elementos en el arreglo sin modificar los índices
         */
      ?>
    </p>
  </body>
</html>
```





```

asort($arregloAnimales);
print_r($arregloAnimales);
?>
</p>
<p>
<h3>Ejemplo de la función isset()</h3>
<?php
/* La función isset() evalúa si una variable ya ha sido
 * inicializada y devuelve un valor lógico, es supremamente
 * útil para evitar errores lógicos
 */
$a = "Si estoy inicializada";
if (isset($a)) {
    echo "La variable \$a si esta inicializada <br />";
}
if (!isset($b)) {
    echo "La variable \$b no está inicializada";
}
?>
</p>
</body>
</html>

```

Fuente: SENA

Descargue el segmento anterior del código como archivo .php del material complementario de este programa de formación en la siguiente ruta:

Materiales del programa / Materiales de apoyo / Documentos complementarios / Documentos complementarios: Actividad de aprendizaje 3 / Ejemplo 5



Figura 5. Ejecución del ejemplo 5

Fuente: SENA



Ejercicio 4:

Busque y utilice en un ejemplo desarrollado por usted, una función de la biblioteca PHP que busque un valor dado dentro de un arreglo cualquiera y de encontrarlo, devuelva la clave correspondiente, recuerde que para acceder a la sección de referencia de funciones del manual PHP en la dirección <http://www.php.net/manual/es/funcref.php> puede buscar por nombres de función utilizando la herramienta de búsqueda disponible en la parte superior derecha de la página.

Con el propósito de poner en práctica los conocimientos adquiridos a través de este material de formación, consulte la guía de aprendizaje y realice todas las evidencias propuestas en ella.

Para acceder a la guía y a las evidencias diríjase al botón: Actividades / Actividad de aprendizaje 3



Referencias

- The PHP Group. (s.f.). *include*. Consultado el 30 de junio de 2015, en <http://php.net/manual/es/function.include.php>
- The PHP Group. (s.f.). *include_once*. Consultado el 30 de junio de 2015, en <http://php.net/manual/es/function.include-once.php>
- The PHP Group. (s.f.). *Manual de PHP*. Consultado el 30 de junio de 2015, en <http://www.php.net/manual/es/index.php>
- The PHP Group. (s.f.). *require*. Consultado el 30 de junio de 2015, en <http://php.net/manual/es/function.require.php>
- The PHP Group. (s.f.). *require_once*. Consultado el 30 de junio de 2015, en <http://php.net/manual/es/function.require-once.php>
- Zend Technologies Ltd. (s.f.). *Zend Framework Coding Standard for PHP*. Consultado el 30 de junio de 2015, en <http://framework.zend.com/manual/1.10/en/coding-standard.html>

Control del documento

	Nombre	Cargo	Dependencia	Fecha
Autor	Jorge Luis Ballesteros Vargas	Instructor	Centro Metalmecánico Regional Distrito Capital	Diciembre de 2014
Adaptación	Paola Andrea Bobadilla Gutiérrez	Guionista - Línea de producción	Centro Agroindustrial Regional Quindío	Junio de 2015

