





Tabla de contenido

Descripción material del programa	1
Mapa conceptual	3
1. Etiquetas PHP	3
Separación de instrucciones	12
Comentarios en PHP	14
Sangrías.....	16
Largo máximo de línea.....	18
Constructor echo.....	18
2. Manejo de datos en PHP	19
Variables	19
Variables predefinidas.....	25
Convenciones de nombres para variables	27
Tipos de datos:	28
Constantes	34
Convenciones de nombres para constantes	35
Operadores	35
Expresiones	38
Asignación por valor y por referencia.....	41
3. Estructuras de control.....	44
Condicionales if, else, elseif.....	44
Condiciones	46
Ciclos condicionales while y do-while	48
Ciclos for y foreach	51
Instrucciones break y continue.....	53
Sentencia Switch	54
Referencias	56





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 pueden encontrarse 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 2



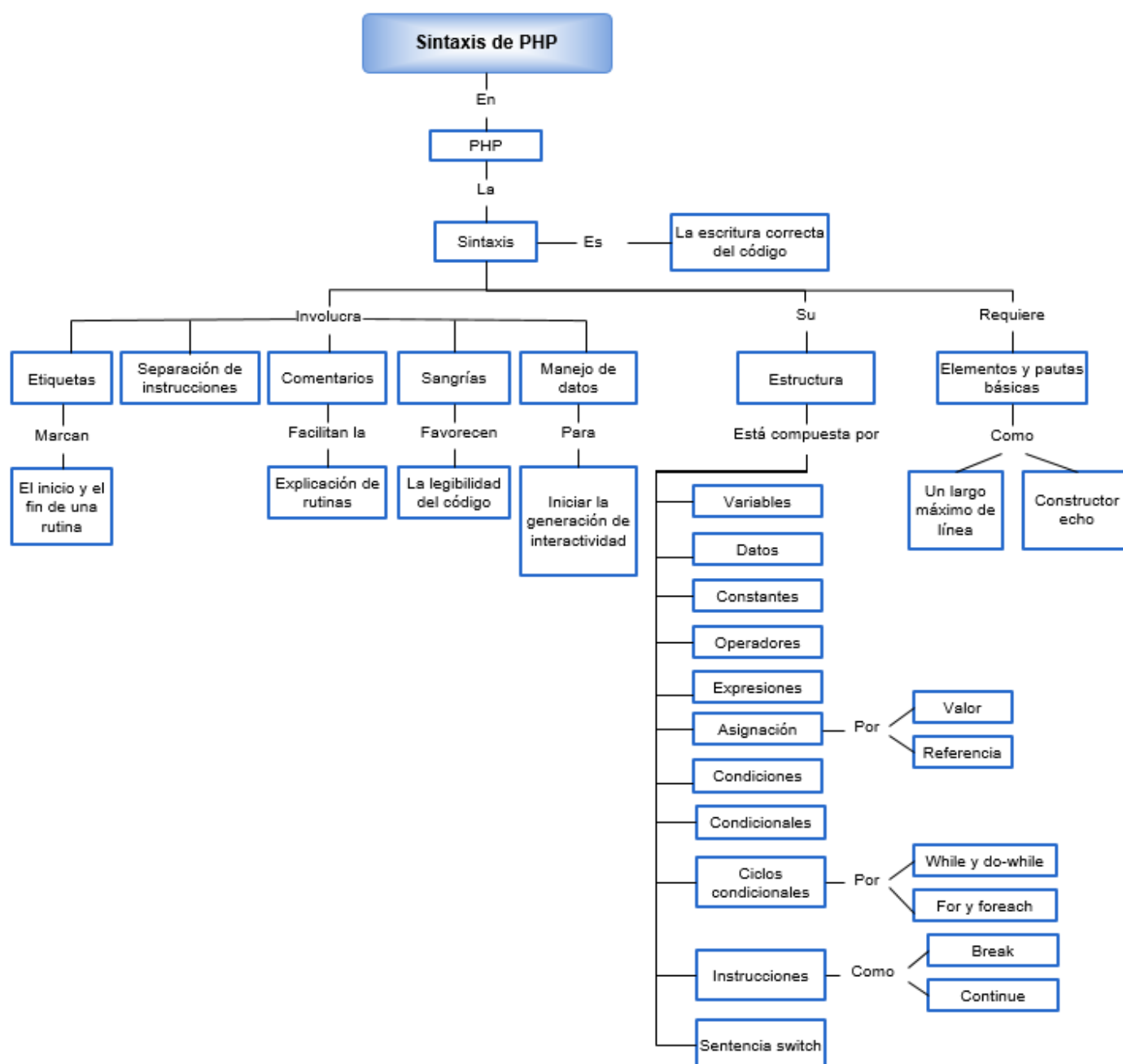


Para usar estos segmentos de código solo se necesita copiarlos y pegarlos en el editor o entorno que esté usando para el desarrollo, o en el caso de que se encuentren etiquetados con la ruta del archivo puede abrirse directamente desde el editor o entorno. Los segmentos de código están comentados (usando 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. Etiquetas PHP

Así como todo el código HTML requiere de etiquetas para delimitar cada objeto que se utiliza, PHP emplea etiquetas para delimitar las sentencias que le pertenecen y que componen una rutina, estas son: `<?php` para iniciar un segmento del programa y `?>` para cerrar un segmento del programa; tal como se muestra a continuación:

```
<?php
    Contenido PHP
?>
```

Fuente: SENA

Aunque el segmento anterior tiene las etiquetas de PHP correctamente utilizadas, según lo aprendido en la actividad de aprendizaje 1 al correr el ejemplo en el entorno de desarrollo, se obtendrá un error (se recomienda que cada segmento de código que aparece en este material de formación sea corrido en el entorno de desarrollo, aún aquellos que sean erróneos, con el fin de manipular el lenguaje tanto como sea posible para lograr un mejor entendimiento de su funcionamiento, así mismo es conveniente que se use el código para hacerle cambios y observar lo que sucede), puesto que la única “sentencia” que se tiene es el texto “Contenido PHP”, pero esta no es en absoluto una instrucción válida del lenguaje, ya que en primer lugar si desde PHP se requiere enviar un texto que deba salir al navegador se tiene que utilizar el constructor del lenguaje echo, que se explicará más adelante; toda cadena de carácter que contenga espacios deberá ser escrita entre comillas simples o dobles y toda sentencia debe terminar con un punto y coma (;).

En la Figura 1 se puede ver el error que se generaría al ejecutar el ejemplo anterior, es conveniente notar que el depurador del lenguaje indica el posible error y la línea en la que se encuentra. Es necesario ir comprendiendo los diferentes tipos de errores y a qué se refieren, por otra parte la línea que se muestra normalmente no es exactamente la que contiene el error sino la siguiente que se ve afectada por el mismo, en la Figura 1 se muestra el error en la línea 2, pero es la línea 1 donde falta el punto y coma (;), aunque este no es el único error, ya que no hay ninguna instrucción válida de PHP.



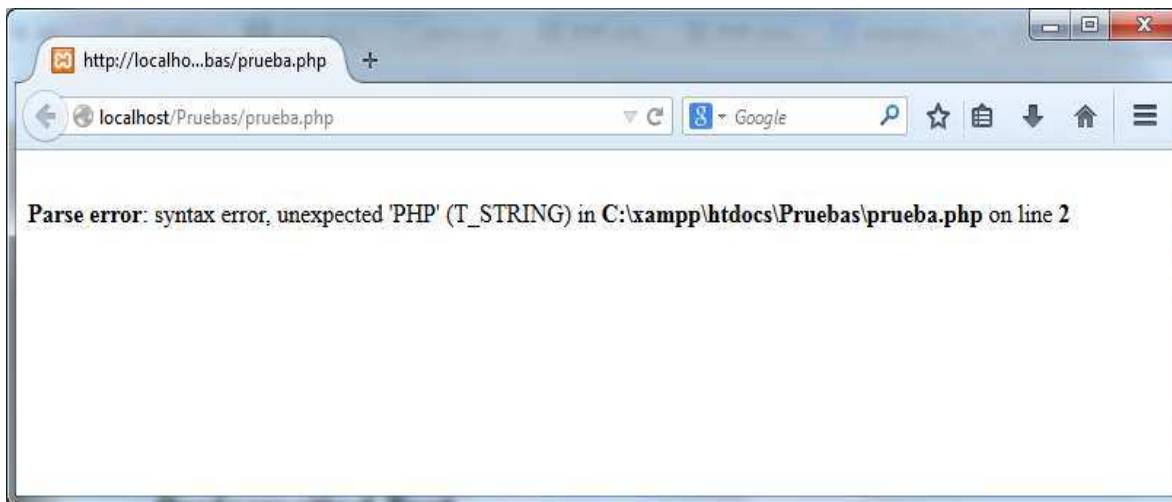


Figura 1. Error de sintaxis por una línea de código que no es una sentencia correcta de PHP

Fuente: SENA



Fuente: SENA

Se recomienda que cada segmento de código que aparece en este material de formación sea corrido en el entorno de desarrollo, aún aquellos que sean erróneos, con el fin de manipular el lenguaje tanto como sea posible para lograr un mejor entendimiento de su funcionamiento, así mismo es conveniente que se use el código para hacerle cambios y observar lo que sucede.

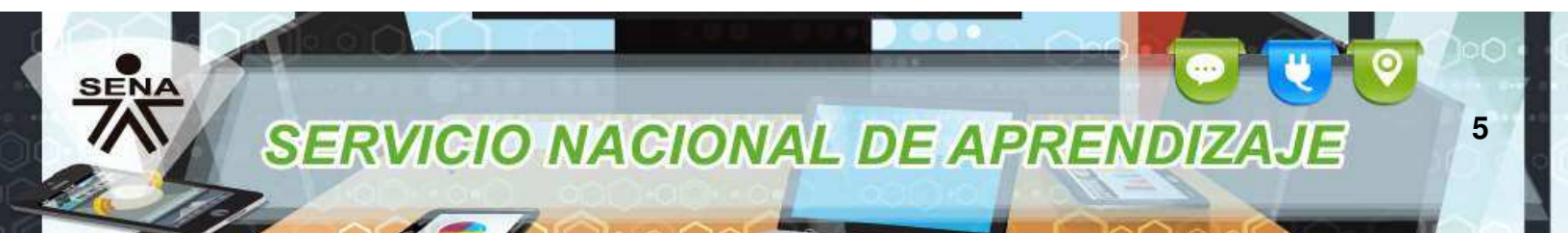
Entonces la forma correcta de escribir el código anterior sería:



```
<?php
    echo "Contenido PHP";
?>
```

Fuente: SENA

Al ejecutar este código se obtendrá un resultado exitoso como puede verse en la Figura 2.



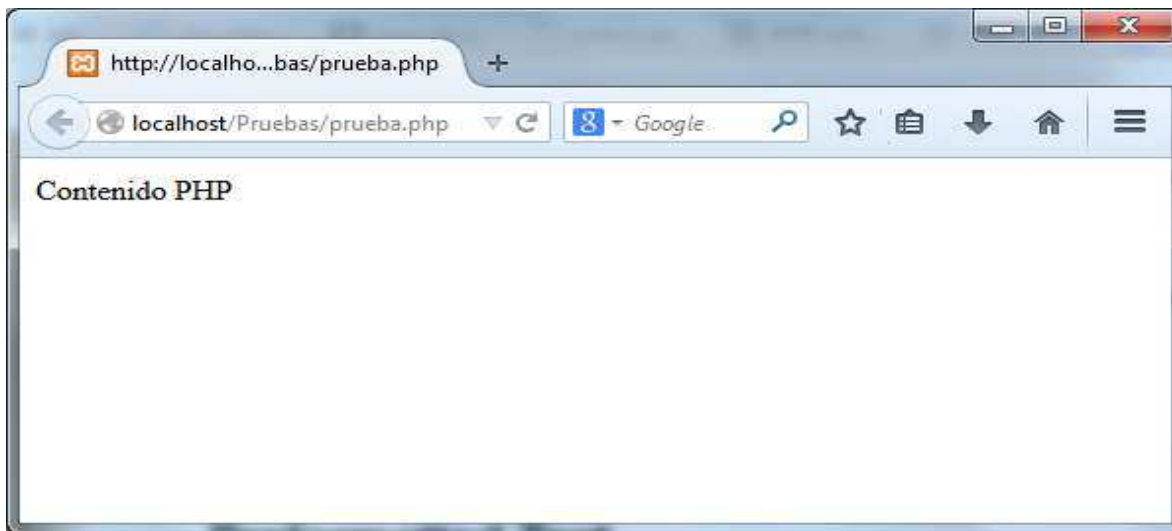
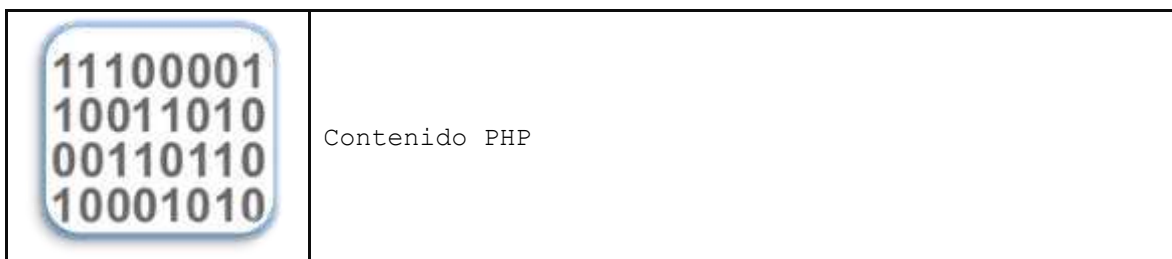


Figura 2. Resultado exitoso al ejecutar una sentencia válida de PHP
Fuente: SENA

Ahora bien, si dentro del archivo .php simplemente se escribe el texto “Contenido PHP”, el resultado será el mismo que el último ejemplo, puesto que el navegador tomaría ese texto como si fuera HTML.



Fuente: SENA

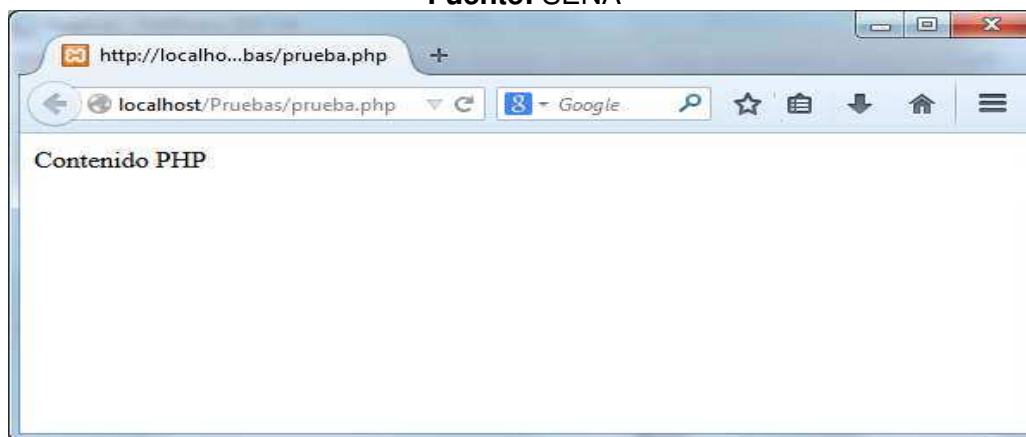



Figura 3. Resultado de ejecutar un archivo .php sin las etiquetas del lenguaje
Fuente: SENA





Debe recordarse que PHP es un lenguaje cuyas rutinas pueden escribirse dentro del código HTML de una página y más allá de esto, incluso las buenas prácticas aconsejan que todo código HTML que deba tener el archivo .php se escriba sin utilizar sentencias PHP, así mismo aunque en HTML muchas etiquetas (como las de la estructura de la página) no son obligatorias, se recomienda que se escriban como parte de las buenas prácticas del lenguaje (solo se tendrán archivos .php sin ningún tipo de código HTML en casos como los repositorios de funciones que son llamados desde otros archivos PHP y que solo contienen rutinas para procesar datos, tema que se verá en la actividad de aprendizaje 3). Por lo tanto, la forma correcta de escribir todo el archivo sería:

Ejemplo 1:

	<pre><!DOCTYPE html> <html> <head> <title>Prueba</title> <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859- 1" /> </head> <body> <?php echo "Contenido PHP"; ?> </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 2 / Ejemplo 1

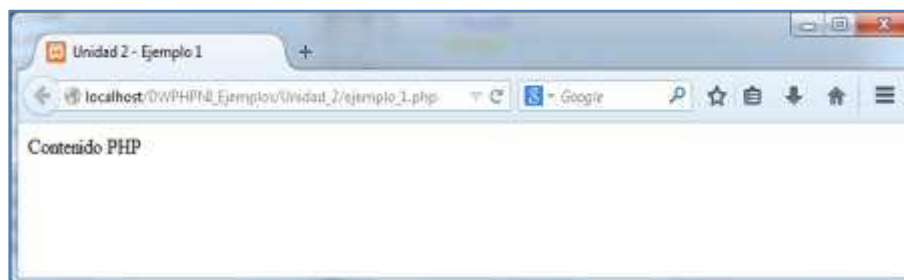
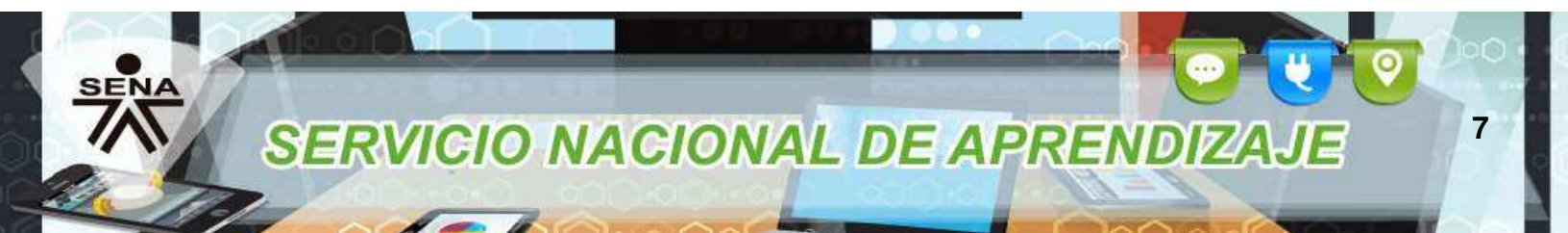


Figura 3. Ejecución del ejemplo 1

Fuente: SENA





Fuente: SENA

Las buenas prácticas aconsejan que todo código HTML que deba tener el archivo .php se escriba sin utilizar sentencias PHP, así mismo aunque en HTML muchas etiquetas (como las de la estructura de la página) son obligatorias, se recomienda que se escriban como parte de las buenas prácticas del lenguaje.

Cuando un archivo .php tiene en su interior varios segmentos de código PHP el intérprete los toma como un mismo programa.

En el siguiente ejemplo se puede ver que hay un programa dividido en 4 segmentos, cuando se trate el tema de variables se comprenderá mejor este concepto, ya que una variable inicializada en un segmento de código tendrá como ámbito todo el archivo .php a pesar de que hayan aperturas y cierres del código PHP.

Ejemplo 2:



```
<!DOCTYPE html>
<html>
  <head>
    <title>Prueba</title>
    <meta http-equiv="Content-Type"
      content="text/html; charset=ISO-8859-
1" />
  </head>
  <body>
    <h3>Nombres: </h3>
    <?php
      echo "Luis Pablo <br />"; //Primer
segmento de un programa PHP
    ?>
    <h3>Apellidos:</h3>
    <?php
      echo "Perez Jimenez <br />"; //Segundo
segmento del mismo programa
    ?>
    <h3>Edad:</h3>
    <?php
      echo "26 años"; //Tercer segmento
del mismo programa
    ?>
    <h3>Teléfono:</h3>
```



	<pre><?php echo "+57 (1) 323 12 00"; //Cuarto segmento del mismo programa ?> </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 2 / Ejemplo 2

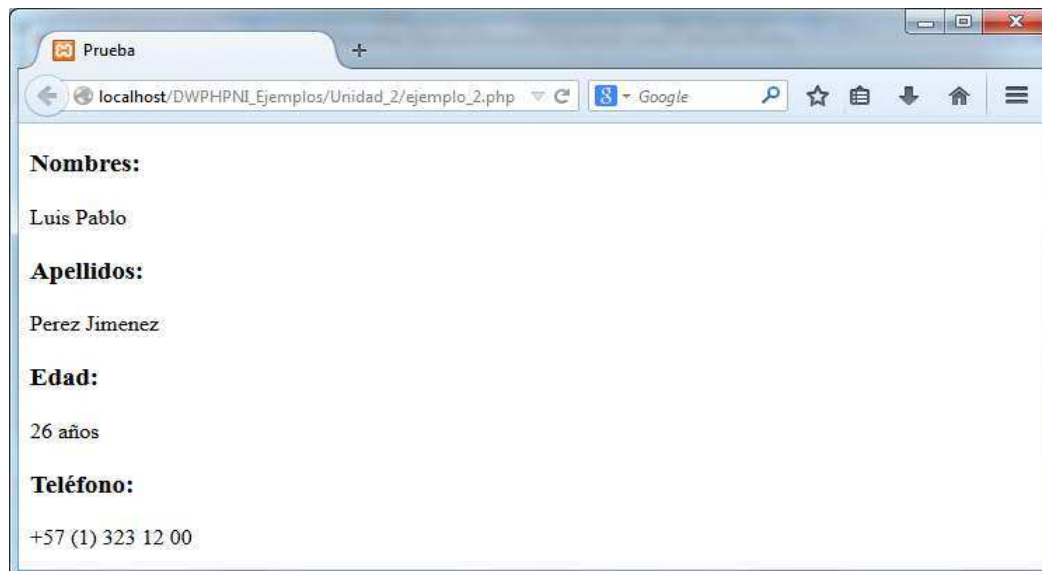


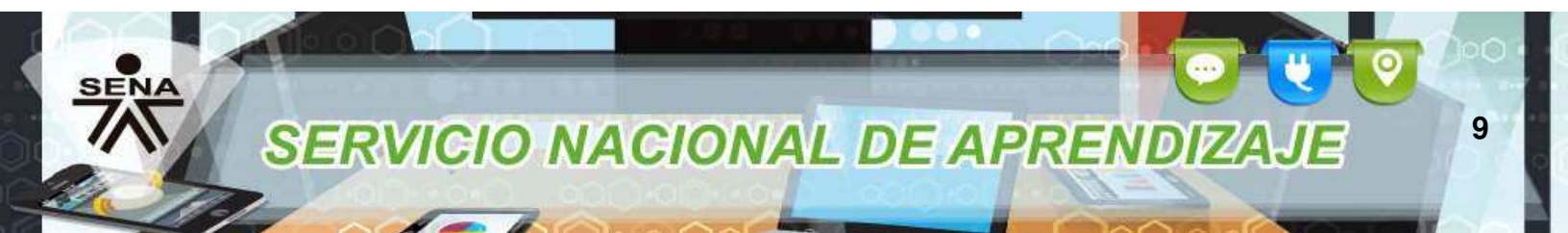
Figura 4. Ejecución de programa PHP dividido en varias secciones dentro del ejemplo 2

Fuente: SENA

Ejercicio 1:

Modifique el código anterior para adicionar la dirección de la persona, donde en la dirección se resalte con negrilla como se indica en el siguiente texto de ejemplo: **Cr. 34 No. 54 - 25**, es decir que se resalta la primera parte de la dirección (Cl, Cr., Av., Dg. o Tr.) y que toda la dirección que se ingrese quede en la misma línea, revise el código resultante y de ser necesario depúelos.

PHP soporta otras etiquetas de apertura y cierre como por ejemplo:

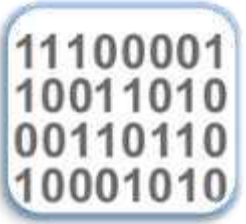




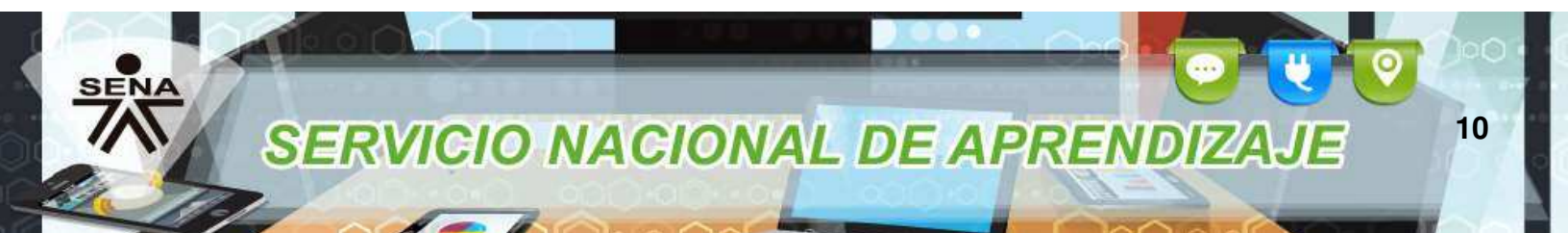
- **Etiquetas ASP:** `<% %>` (no permitido) necesita el parámetro `asp_tags` habilitado en `php.ini`, que por defecto viene deshabilitado.
- **Etiquetas cortas:** `<? ?>` (no permitido), necesita el parámetro `short_open_tag` habilitado en `php.ini`, que por defecto viene deshabilitado.
- **Etiquetas script:** (no recomendado) `<script language = "php"> </script>`.
- **Etiquetas cortas echo:** `<?= ?>`, reemplazan salidas cortas del programa que deberían escribirse: `<?php echo "Texto de salida"; ?>` y se escribiría `<?= "Texto de salida" ?>`.

Si se reescribe el código del archivo del ejemplo 2, cambiando en cada segmento las etiquetas de apertura y cierre por las que se muestran en el listado anterior, se podrá ver el comportamiento de cada opción.

Ejemplo 3:

	<pre> <!DOCTYPE html> <html> <head> <title>Prueba</title> <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859- 1" /> </head> <body> <h3>Nombres: </h3> <% echo "Luis Pablo
"; //Primer segmento de un programa PHP %> <h3>Apellidos:</h3> <? echo "Perez Jimenez
"; //Segundo segmento del mismo programa ?> <h3>Edad:</h3> <script language = "php"> echo "26 a&ntilde;os"; //Tercer segmento del mismo programa </script> <h3>Tel&eacute;fono:</h3> <?= "+57 (1) 323 12 00"; ?> </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 2 / Ejemplo 3

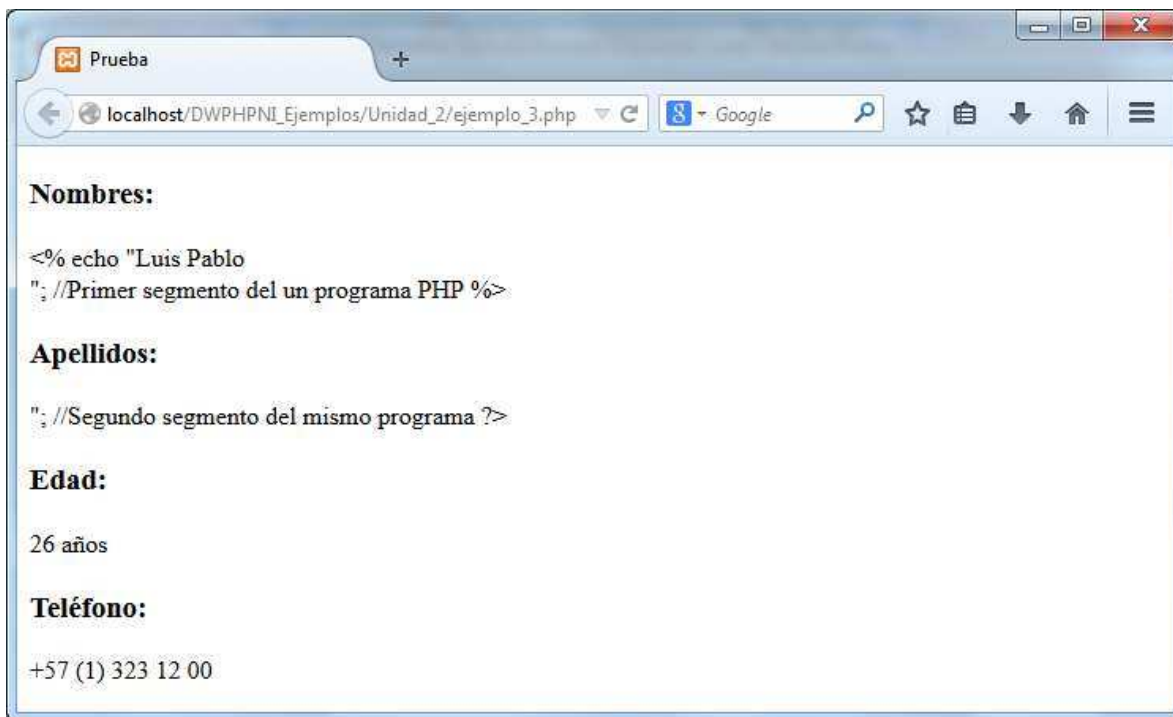


Figura 5. Ejecución de programa PHP con diferentes etiquetas de apertura y cierre del ejemplo 3
Fuente: SENA


En el primer segmento que debería mostrar el nombre, el intérprete envía todo el texto tal cual como aparece en el código fuente, esto se debe a que el parámetro `asp_tags` está deshabilitado por defecto en el archivo `php.ini`. El segundo segmento que debería presentar los apellidos también presenta un funcionamiento no deseado, pues solo muestra el texto que hay después del símbolo mayor que (`>`) del objeto `
`, lo cual se debe a que el parámetro `short_open_tag` también está deshabilitado por defecto y a que el navegador interpreta la etiqueta `<?` como XHTML. En estos dos casos el uso de estas etiquetas no solo está desaconsejado sino que no está permitido en ningún caso según los estándares de programación propuestos por Zend Technologies.

El tercer segmento que presenta la edad funciona correctamente aunque no está aconsejado. El cuarto segmento también funciona correctamente y su uso no está desaconsejado por los estándares. De cualquier forma, el único juego de etiquetas que





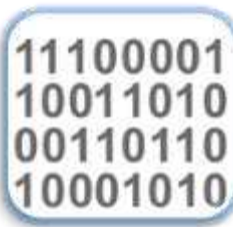
está totalmente aconsejado para utilizar en todos los casos es `<?php ?>` y muchos expertos incluso aconsejan que se use `<?php echo "Texto de salida"; ?>` en lugar de `<?= "Texto de salida"; ?>`.

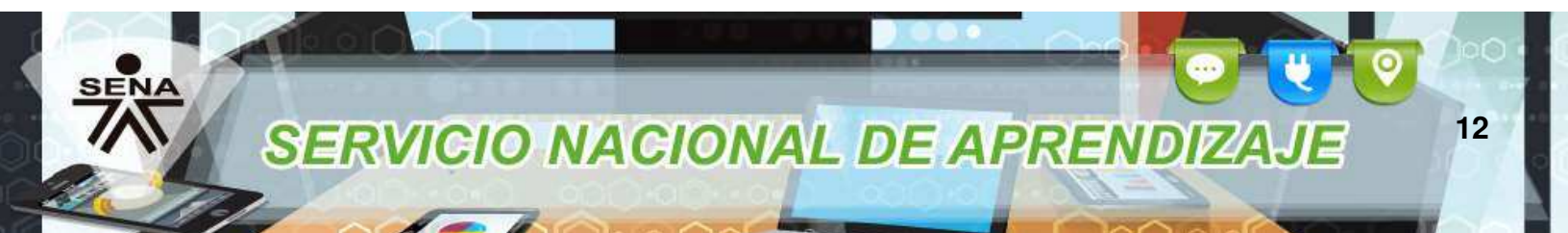
 <p>Fuente: SENA</p>	<p>Para aquellos archivos que contengan exclusivamente código PHP, la etiqueta de cierre <code>?></code> no está permitida en ningún caso, no es requerida por el intérprete PHP, y omitirla previene la inserción accidental de espacios en blanco o nuevas líneas en la respuesta al navegador, lo cual puede causar efectos no deseados. (Zend Technologies Ltd., s.f.)</p>
--	---

Separación de instrucciones

Las instrucciones en PHP deben terminar siempre con un punto y coma (;) al final de cada sentencia, esto excepto para la última línea de un segmento de código ya que la etiqueta de cierre `?>` implica la existencia del punto y coma, a pesar de ello no es recomendado dejar la última línea sin este delimitador ya que se pueden generar errores al depurar porque se adiciona nuevo código luego de la última línea y se olvida poner el punto y coma faltante.

Ejemplo 4:

	<pre> <!DOCTYPE html> <html> <head> <title>Prueba</title> <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859- 1" /> </head> <body> <h3>Nombres: </h3> <?php echo "Luis Pablo
"; //aquí se usa el punto y coma final ?> <h3>Apellidos:</h3> <?php echo "Perez Jimenez
"; //aquí se usa el punto y coma final ?> <h3>Edad:</h3> <?php echo "26 a&ntilde;os" //aquí NO se usa el </pre>
---	---





```
punto y coma final
?>
<h3>Teléfono:</h3>
<?php
    echo "+57 (1) 323 12 00" //aquí NO se
    usa el punto y coma final
    ?>
</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 2 / Ejemplo 4

En la Figura 7 puede verse como, a pesar de que los dos últimos segmentos de código no usan el punto y coma final, el programa funciona correctamente, esto es debido a que PHP no requiere el delimitador en la última línea (o en este caso única), pero lo que se aconseja es que siempre se ponga el punto y coma, aunque sea la única y/o última línea del segmento.

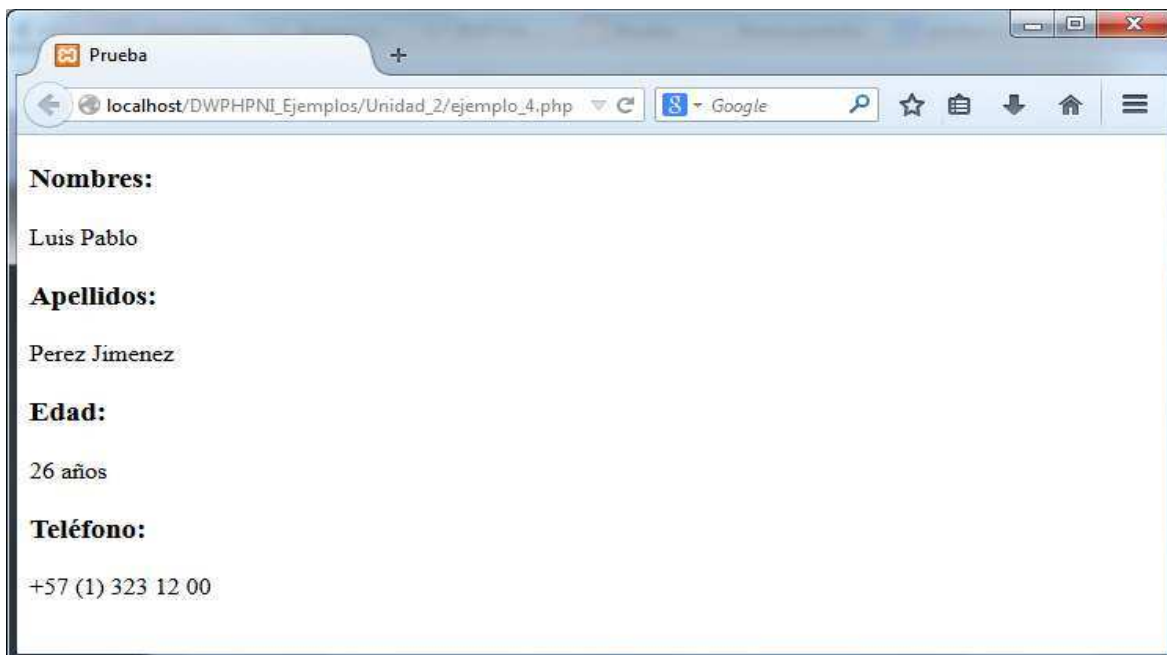
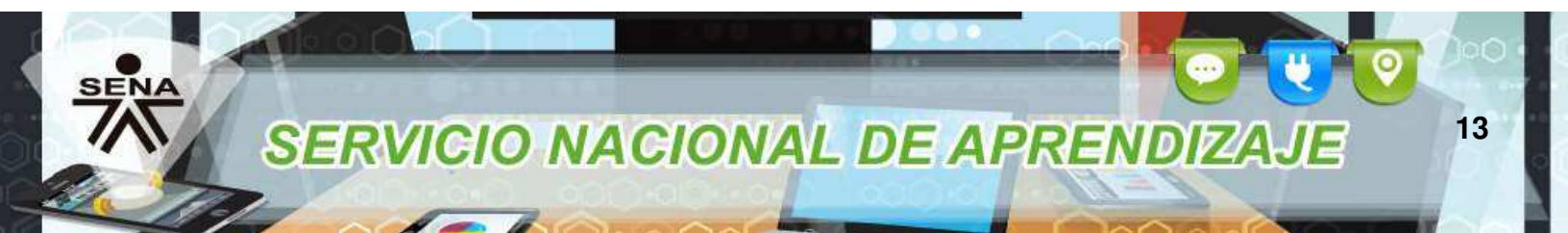


Figura 6. Ejecución de programa que no usa el punto y coma al final en algunas de sus instrucciones del ejemplo 4

Fuente: SENA





Fuente: SENA

Aunque la sintaxis indica que en la última sentencia de un script PHP puede obviarse el punto y coma (;) para finalizar la línea, se recomienda que no se haga para evitar posteriores errores en el caso de que el código se tenga que extender más allá de esa última línea.

Comentarios en PHP

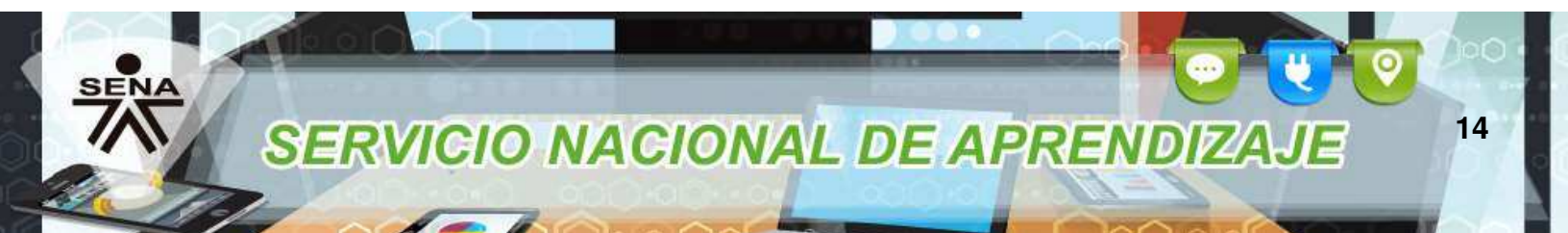
Los comentarios son una herramienta que le permite al desarrollador poner texto que solo será visto en el código fuente y que no va a salir impreso en el navegador, se usan para explicar rutinas complicadas o para dar información sobre la función de variables, lo cual es muy útil cuando un equipo de desarrolladores deben participar en un mismo proyecto o cuando un desarrollador diferente al original debe hacer modificaciones al código, también pueden usarse para generar la documentación de una aplicación. PHP permite el uso del mismo tipo de comentarios que se utilizan en C y C++: `//` para comentarios de una sola línea y `/* */` para bloques de comentario que pueden tener una o más líneas, también es válido utilizarlos de una sola línea con el estilo consola Unix `#`, en el ejemplo 5 puede verse el código de los ejemplos anteriores comentado, usando las diferentes opciones posibles.




Fuente: SENA

Los comentarios son una herramienta que permite al desarrollador poner texto que solo será visto en el código fuente y que no va a salir impreso en el navegador, se usan para explicar rutinas complicadas o para dar información sobre la función de variables.

Ejemplo 5:







```
<!DOCTYPE html>
<html>
  <head>
    <title>Unidad 2 - Ejemplo 5</title>
    <meta http-equiv="Content-Type"
          content="text/html; charset=ISO-8859-
1" />
  </head>
  <body>
    <h3>Nombres: </h3>
    <?php
      /* En este pequeño programa se muestra el
uso de los diferentes
      * estilo de comentarios que pueden
usarse en PHP
      * este es un comentario tipo bloque de
varias líneas
      */
      echo "Luis Pablo<br />"; //Esta línea
imprime el texto Luis Pablo
    ?>
    <h3>Apellidos:</h3>
    <?php
      echo "Perez Jimenez <br />"; #Esta línea
imprime el texto Perez Jimenez
    ?>
    <h3>Edad:</h3>
    <?php
      echo "26 a&ntilde;os"; //Comentario de
una línea estilo C y C++
    ?>
    <h3>Tel&eacute;fono:</h3>
    <?php
      echo "+57 (1) 323 12 00"; #Comentario de
una línea estilo consola Unix
    ?>
  </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 2 / Ejemplo 5

Los comentarios también pueden ser usados con el fin de hacer que una parte del código que está en evaluación o depuración no se ejecute, simplemente se debe encerrar dicho código en un comentario de bloque o poner un comentario de línea al inicio de cada una



de las sentencias que no quiere que se ejecuten y estas no serán tenidas en cuenta pero podrán recuperarse simplemente quitando los comentarios.

 Fuente: SENA	Se recomienda comentar toda rutina que se cree, ya que esto ayuda a futuros programadores que deban hacer mantenimiento a la aplicación pero incluso al mismo desarrollador cuando más adelante tenga que hacer mantenimiento a su propio código. Los comentarios se usan también para la documentación de la aplicación.
--	---

Ejercicio 2:

Con base en el ejemplo 5 presente en el navegador los datos básicos de una empresa y un listado de los servicios que presta, en el encabezado se debe crear un comentario de bloque que indique los datos del creador de la página, se debe poner un comentario de línea indicando dónde empiezan los datos de la empresa y dónde empiezan los servicios.

Sangrías

Con el fin de hacer mucho más legible todo código en un lenguaje de programación se utiliza la técnica de sangría que implica que cuando una línea de código está supeditada a otra (por ejemplo en las estructuras de control condicionales que se verán en esta actividad de aprendizaje, las líneas de código que solo se ejecutarán si la condición se cumple están subordinadas a la condición) el renglón de dicha línea debe empezar más a la derecha que la que está arriba.

En PHP la sangría debe ser siempre de 4 espacios en blanco exactamente por cada nivel de subordinación; es decir que el primer nivel estará 4 espacios a la derecha, el segundo nivel 8 espacios más a la derecha, el tercer nivel 12 espacios más a la derecha y así sucesivamente.

No está permitido en ningún caso el uso de tabulaciones ya que a pesar de que el uso de estas puede hacer un poco más sencillo el sangrado puede generar diferencias en el código entre plataformas, puesto que en algunas la tabulación es de 4 espacios y en otras es de 5.



Fuente: SENA

Las sangrías son muy importantes para hacer más legible el código, en PHP se usan 4 espacios por cada nivel de sangría para evitar los cambios en el código que puede producir el usar las tabulaciones (que no son del mismo número de espacios en todos los sistemas).

En la Figura 8 se muestra como el código se organiza usando las sangrías, estas también se aconsejan en el código HTML.

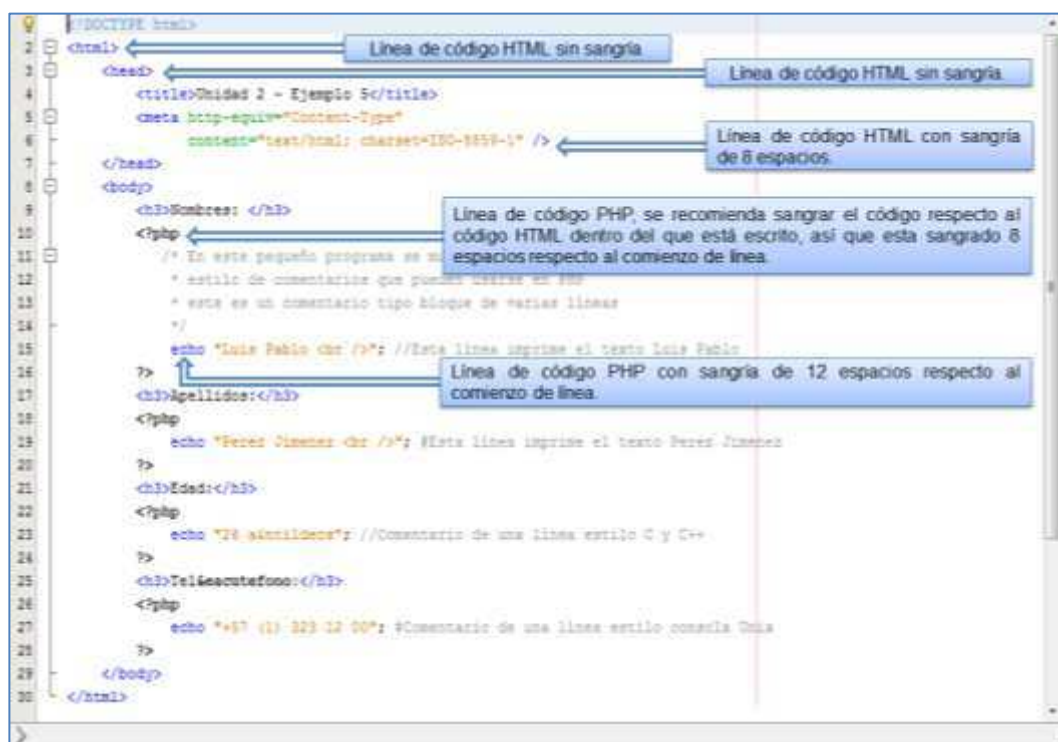


Figura 7. Código PHP y HTML con sangrías que muestran la jerarquía de las instrucciones

Fuente: SENA

Ejercicio 3:

Busque el método abreviado de NetBeans que da formato de manera automática al código para que cumpla con las indicaciones de sangría que se explicaron.





Largo máximo de línea

El objetivo debe ser tener líneas de código que no superen los 80 caracteres. Es decir que los desarrolladores que quieran mantener los estándares del Framework Zend deben esforzarse por mantener cada línea de su código por debajo de 80 caracteres siempre que sea posible y práctico hacerlo. Sin embargo, líneas más largas son aceptables en ciertas circunstancias. El largo máximo de cualquier línea de código PHP es de 120 caracteres. (Zend Technologies Ltd., s.f.)

Nota: los estándares del Framework Zend, se encuentran en el siguiente enlace: <http://framework.zend.com/manual/1.10/en/coding-standard.html>

En la Figura 8 puede verse una línea roja en la parte derecha del área de trabajo del entorno de desarrollo, esta línea muestra el punto donde se alcanzan los 80 caracteres, lo cual es muy útil para cumplir con este requisito de los estándares de desarrollo.

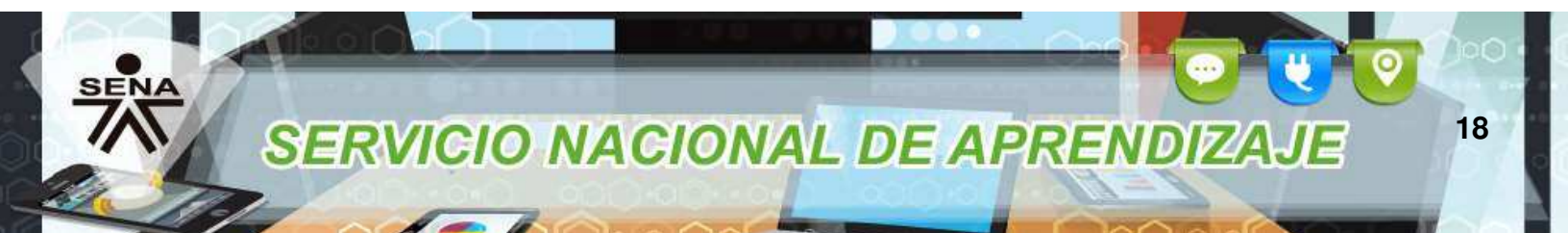


Fuente: SENA

Las líneas no deben sobrepasar 80 caracteres y solo cuando no haya otra opción se puede llegar hasta un máximo de 120 caracteres, este último límite no se debe sobrepasar en ningún caso.

Constructor echo

Se tratará el tema del constructor del lenguaje echo debido a que hace parte de los elementos básicos para escribir un programa en PHP. Este constructor tiene como función mostrar una o más cadenas de carácter, dichas cadenas pueden ser valores literales (por ejemplo: `echo 'Texto literal'`), en este caso siempre se recomienda usar las comillas simples, pero también se pueden construir cadenas usando el contenido de variables (por ejemplo: `echo "Buenos días $nombre"`), en este caso es necesario usar las comillas simples ya que de no hacerlo echo tomará `$nombre` no como una variable sino como una cadena y la mostrará literalmente en el navegador, si dentro de un segmento de comillas dobles intentan realizar operaciones con variables, se mostrarán los valores de las variables, los operadores usados y no el resultado de la operación (por ejemplo: teniendo `$a` con un valor asignado de 3 y `$b` con un valor





asignado de 4 la instrucción `echo "La suma es igual a $a + $b"` mostrará "La suma es igual 3 + 4" en el navegador).

Las cadenas que `echo` muestra pueden ser construidas mediante concatenación, el operador de concatenación de cadenas es el punto (`.`), para el último ejemplo del párrafo anterior se podría construir la sentencia así: `echo "La suma es igual a " . ($a + $b)` y en este caso en el navegador se mostraría: "La suma es igual a 7".

Cuando en una cadena que se va a mostrar `echo` se necesite insertar caracteres que podrían generar conflictos como por ejemplo las comillas dobles, entonces se debe utilizar el símbolo de barra inversa o backslash para escaparlos, no se puede escribir por ejemplo: `echo "ella dijo "buenos días" sacandosticamente"` porque esto generaría un conflicto debido a que las comillas dobles que se usan como recurso literario para indicar el sarcasmo en la frase buenos días sería tomadas por PHP como el cierre de la cadena y la nueva apertura sin que exista un operador de concatenación intercediendo, por lo tanto tendría que escribirse: `echo "ella dijo \"buenos días\" sacandosticamente"`.

2. Manejo de datos en PHP

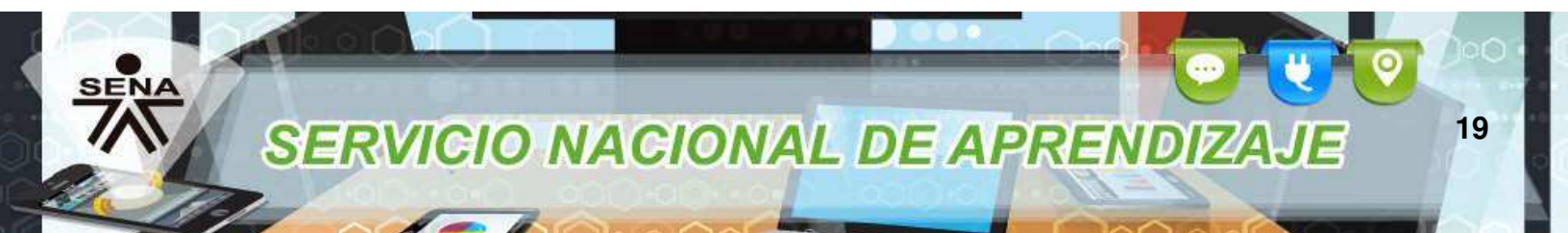
En el caso de los lenguajes de desarrollo web la capacidad de procesar y manejar datos es el paso inicial para generar interactividad en una página. Las estructuras de datos que maneja cada lenguaje deben ser conocidas a fondo con el fin de comprender cómo es que se almacena la información en las rutinas en las que se procesa.

Variables

Antes que nada se debe tener en cuenta que PHP es un lenguaje **débilmente tipado**, esto significa que a una variable que se inicializa no se le asigna un tipo de dato, sino que la variable toma el tipo de dato según el primer valor que se le asigna (más adelante conocerán los diferentes tipos de datos que puede tomar una variable).

En cuanto a la sintaxis de las variables todas deben iniciar con el símbolo dólar (`$`), seguido como primer carácter del nombre únicamente por una letra (a-z A-Z) o un carácter de subrayado (`_`) o underscore, como segundo carácter del nombre y de allí en adelante se pueden utilizar únicamente letras (a-z A-Z), caracteres de subrayado (`_`) y números (0-9), no se pueden utilizar espacios en ninguna parte del nombre de las variables ni otros caracteres especiales como la letra (ñ o Ñ) ni acentos de ningún tipo (á, é, í, ó, ú; Â, Ê, ...; ô, ü, ...; à, è, ...), a continuación se muestran ejemplos de variables correctas e incorrectas:

- `$_nombre` → Correcto.





- `$nombre` → Correcto.
- `$nombre1` → Correcto.
- `$nombre_1` → Correcto.
- `$Nombre` → Correcto.
- `Nombre` → Incorrecto porque no empieza con el símbolo dollar (\$), la forma correcta sería `$Nombre`.
- `$1nombre` → Incorrecto, hay un número (1) como primer carácter del nombre de la variable después del símbolo dólar (\$), una forma correcta sería `$nombre1`.
- `$_nombre 1` → Incorrecto, tiene un espacio entre la palabra nombre y el número 1, la forma correcta sería `$_nombre1`.
- `$nombreNiño` → Incorrecto, usa el carácter especial (ñ), la forma correcta sería `$nombreNino`.
- `$nombreSección` → Incorrecto, usa la o con acento (ó), la forma correcta sería `$nombreSeccion`.

Es importante comprender que PHP es sensible a las mayúsculas (case sensitive) en lo que a los nombres de variables respecta, es decir que las variables `$nombreSeccion` y `$nombreseccion` son totalmente diferentes.

Para inicializar una variable es necesario asignarle un valor, esto puede hacerse asignándole un valor literal (por ejemplo: `$a = 5`), asignándole el valor de otra variable (por ejemplo: `$a = $b`) o asignándole el valor resultante de una operación (por ejemplo: `$a = $b + $c`), el solo hecho de declararla no implica que quede inicializada, sino se le asigna ningún valor la variable queda con tipo NULL y este será el valor que retornará si se usa antes de inicializarla.



Fuente: SENA

Antes que nada se debe tener en cuenta que PHP es un lenguaje débilmente tipado, esto significa que a una variable que se inicializa no se le asigna un tipo de datos sino que la variable toma el tipo de dato según el primer valor que se le asigna.

En el siguiente segmento de código se puede ver una serie de variables declaradas e inicializadas con diferentes tipos de datos (más adelante se tratará el tema de los diferentes tipos de datos que maneja PHP), así como un ejemplo de lo que sucede si se declara pero no se inicializa una variable.



Ejemplo 6:



```
<!DOCTYPE html>
<html>
  <head>
    <title>Unidad 2 - Ejemplo 6</title>
    <meta http-equiv="Content-Type"
          content="text/html; charset=ISO-8859-
1" />
  </head>
  <body>
    <h3>Nombres: </h3>
    <?php
      /* Este programa se usará para tratar el
      tema de variables
      * se van a declarar e inicializar
      variables con diferentes tipos
      * de datos y su valor será impreso en el
      navegador
      */

      /* Aunque las dos variables siguientes
      tienen la misma palabra como
      * nombre, son totalmente diferentes
      pues la primera empieza con N
      * y la segunda con n, lo que demuestra
      que PHP es sensible a las
      * mayúsculas y minúsculas, ambas
      variables son inicializadas con
      * datos tipo String lo cual hace que el
      tipo que tomas es String
      */
      $Nombre = 'Luis';
      $nombre = 'Pablo';
      echo "$Nombre $nombre <br />";

    ?>
    <h3>Apellidos:</h3>
    <?php
      $apellidosUsuario_1 = 'Perez Jiménez';
//Otra variable con tipo String
      echo "$apellidosUsuario_1 <br />";

    ?>
    <h3>Edad:</h3>
    <?php
      $edad = 26; //Variable con tipo de dato
Integer
      echo "$edad <br />";

    ?>
    <h3>Estatura:</h3>
    <?php
      $estatura = 1.78; //Variable con tipo de
```



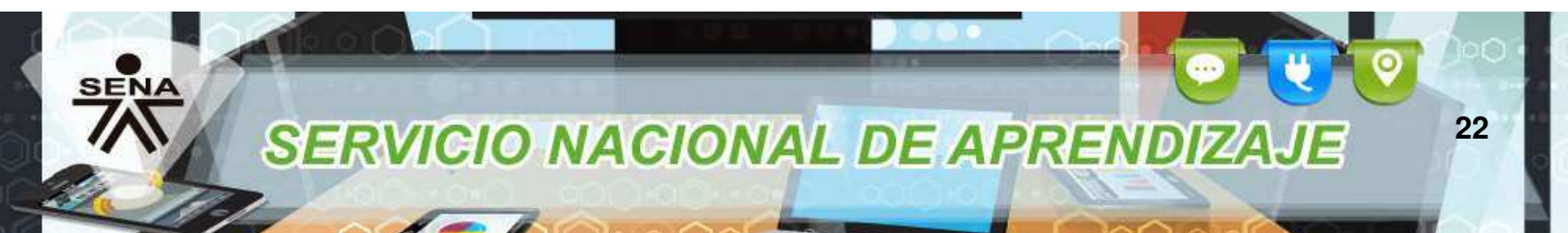


```
dato Float
    echo "$estatura <br />";
?>
<h3>Teléfono;fono:</h3>
<?php
    $telefono = '+57 (1) 323 12 00';
    echo "$telefono <br />"; #
?>
<h3>Dirección:</h3>
<?php
    /* La siguiente variable va a ser
    declarada pero no inicializada
    * el intérprete la tomará como tipo de
    dato NULL y este será
    * el valor que va a contener,
    dependiendo de cómo esté configurado
    * el intérprete puede mostrar un error
    de tipo E_NOTICE
    */
    $direccion;//Variable declarada pero no
    inicializada toma el tipo NULL
    echo "$direccion";
?>
</body>
</html>
</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 2 / Ejemplo 6



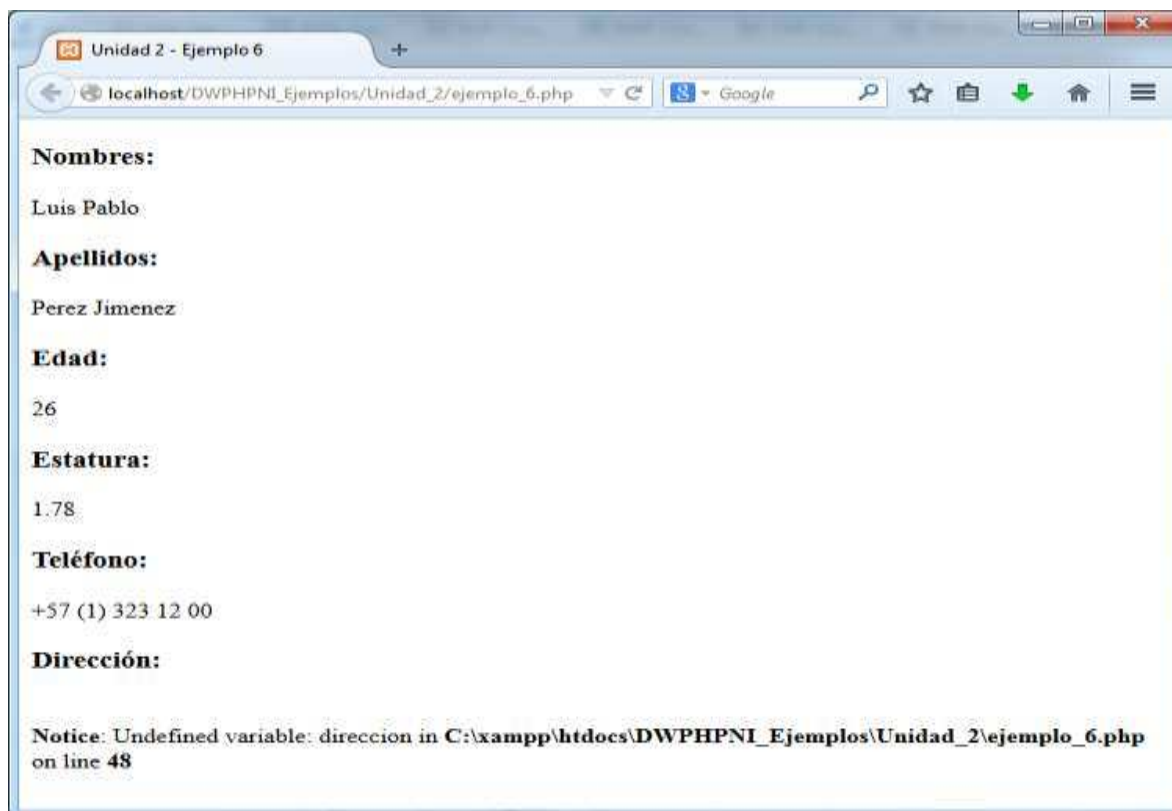
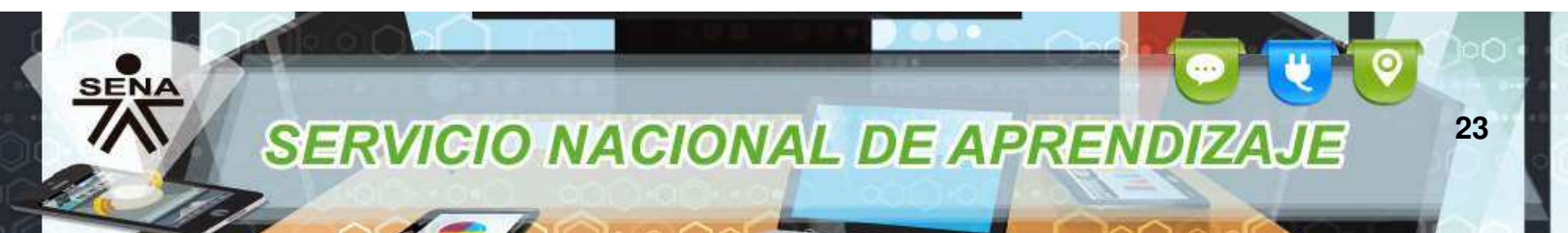


Figura 8. Ejecución del programa contenido en el ejemplo 6
Fuente: SENA

En PHP las variables tienen como ámbito el archivo dentro del que está escrito el código en el que se inicializó la variable y los archivos que sean incluidos o requeridos dentro de este (usando las funciones `include()` o `require()` que se verán más adelante).

En el ejemplo 7 puede verse como en varios segmentos se utiliza la misma variable, cuando una variable se declara dentro de una función (el tema de funciones se tratará en la actividad de aprendizaje 3) se crea un ámbito local de la variable para dicha función únicamente y por lo tanto, si otra variable se declara con el mismo nombre fuera de la función serán dos variables totalmente distintas.





Ejemplo 7:

	<pre><!DOCTYPE html> <html> <head> <title>Unidad 2 - Ejemplo 7</title> <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859- 1" /> </head> <body> <p>El valor de la compra es: \$ <?php /* En este programa se tiene la variable \$valor la cual se * utiliza en varios segmentos del mismo código pudiendo * modificar su contenido debido a que su ámbito se mantiene * dentro del mismo archivo en el que fue declarada */ \$valor = 1000; //Declaración e inicialización de \$valor echo "\$valor"; ?> </p> <p>M&aacute;s el IVA: \$ <?php \$valor = \$valor * 1.16; //Modificación de \$valor echo "\$valor"; ?> </p> <p>M&eacute;nos el descuento de 5%: \$ <?php \$valor = \$valor * 0.95; //Modificación de \$valor echo "\$valor"; ?> </p> </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 2 / Ejemplo 7

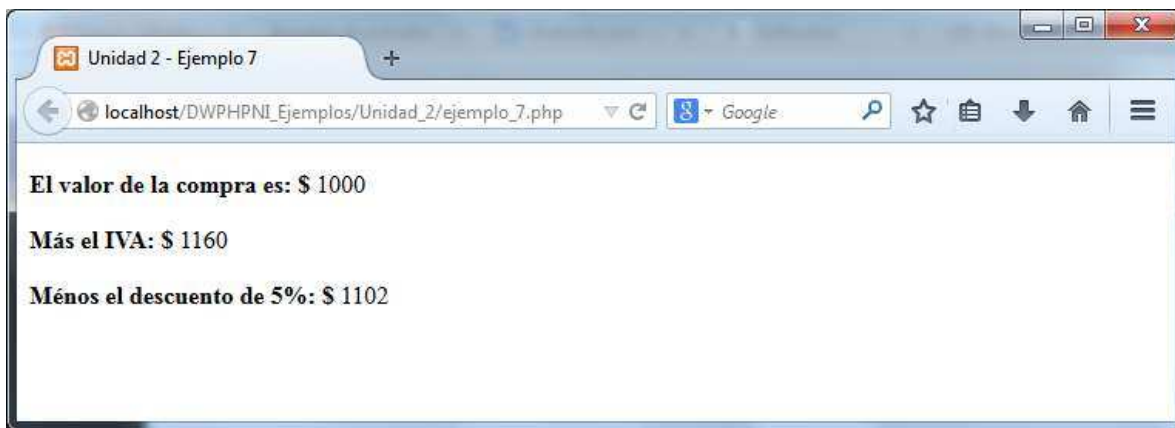


Figura 9. Ejecución del programa contenido en el ejemplo 7
Fuente: SENA

	<p>En PHP las variables tienen como ámbito el archivo dentro del que está escrito el código en el que se inicializó la variable y los archivos que sean incluidos o requeridos dentro de este.</p>
<p>Fuente: SENA</p>	

Variables predefinidas

PHP tiene una gran cantidad de variables predefinidas que están listas en cualquier momento en que se solicite la interpretación de un script y que por lo tanto pueden ser utilizadas en cualquier parte de la aplicación, estas variables manejan una gran cantidad de información útil, como por ejemplo para obtener los datos que se pasan entre una página y otra usando la URL o formularios y se usan las variables del sistema `$_GET` y `$_POST` (tema que se tratará en la actividad de aprendizaje 4).

Para obtener información sobre el entorno del servidor y de ejecución se usa la variable `$_SERVER`, para trabajo con archivos se usa la variable `$_FILES`, para obtener información sobre el último error generado por PHP se usa `$php_errormsg`. Muchas de las variables predefinidas son básicamente arreglos, así que para acceder a la información que contienen, se necesita saber cómo esta indexada dicha información y





usar el índice correcto en la variable, por ejemplo: si se requiere conocer el software que usa el servidor se debe escribir `$_SERVER['SERVER_SOFTWARE']`.

Como puede verse, la mayoría de las variables predefinidas usan caracteres de subrayado y muchas lo tienen en la parte inicial, esta es una de las razones por las que a continuación se prohíbe el uso de caracteres de subrayado en las variables declaradas por el usuario.

Para un listado y referencia completa sobre las variables predefinidas de PHP se puede acceder a: <http://www.php.net/manual/es/reserved.variables.php>

Ejemplo 7_1:

	<pre><!DOCTYPE html> <html> <head> <title>Unidad 2 - Ejemplo 7_1</title> <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859- 1" /> </head> <body> <h3>Ejemplo del uso de una variable predefinida (\$_SERVER): </h3> <p> Software del servidor: <?php /* En este segmento se muestra el ejemplo del uso de una variable * predefinida */ echo \$_SERVER['SERVER_SOFTWARE']; ?> </p> </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 2 / Ejemplo 7_1

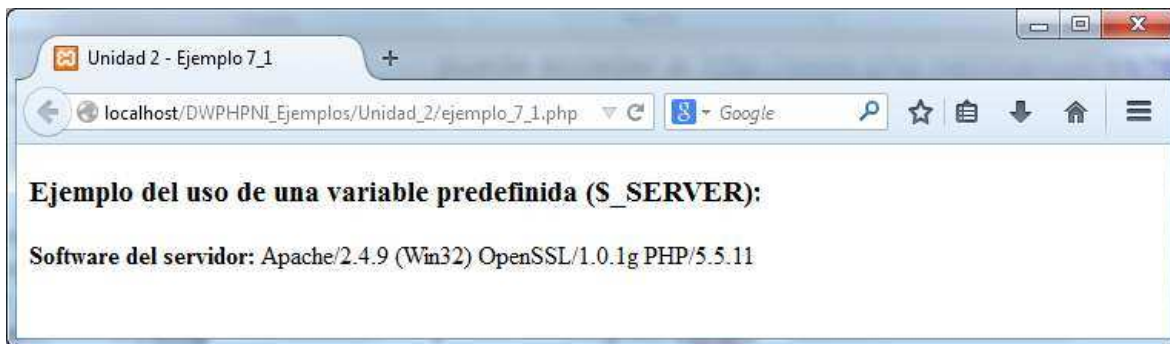


Figura 10. Ejecución del ejemplo7_1
Fuente: SENA

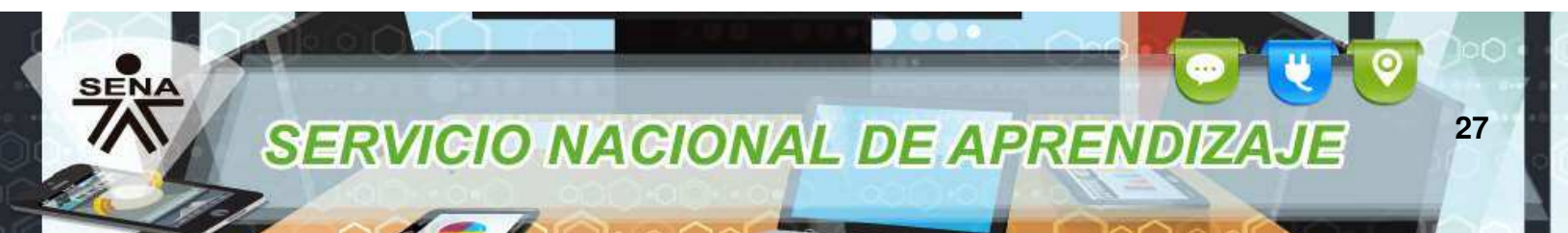
Convenciones de nombres para variables

Los nombres de variables solo deben contener caracteres alfanuméricos, los caracteres de subrayado no están permitidos y los números se permiten en los nombres de variables pero se desaconsejan en la mayoría de los casos.

Por otra parte para las variables que se declaran con el modificador “private” o “protected” (PHP Orientado a Objetos que se tratará en las siguientes actividades de aprendizaje), el primer carácter del nombre de la variable debe ser un carácter de subrayado simple; esta es la única aplicación aceptable del carácter de subrayado en un nombre de variable. Las variables declaradas como “public” nunca deben empezar con un carácter de subrayado.

Los nombres de las variables siempre deben empezar con una letra en minúscula y seguir la convención “notacionCamello”, esta indica que después de la primera palabra del nombre de una variable compuesto por varias palabras las siguientes palabras se escriben con la primera letra en mayúscula, por ejemplo si se tiene la variable `$nombrecompletotrabajador` la notación “mayusculasCamello” indica que debe escribirse `$nombreCompletoTrabajador`.

La verbosidad es aconsejable generalmente, es decir, que los nombres de variables deben ser altamente descriptivos, tanto como sea práctico y suficiente para describir los datos que el desarrollador pretende almacenar en ellas. Nombres de variables breves como `$i` y `$n` se desaconsejan para todo, excepto para contextos de ciclos cortos. Si un ciclo contiene más de 20 líneas de código, las variables que se usan para los conteos, como `$i` o `$j` deben tener nombres más descriptivos. (Zend Technologies Ltd., s.f.)





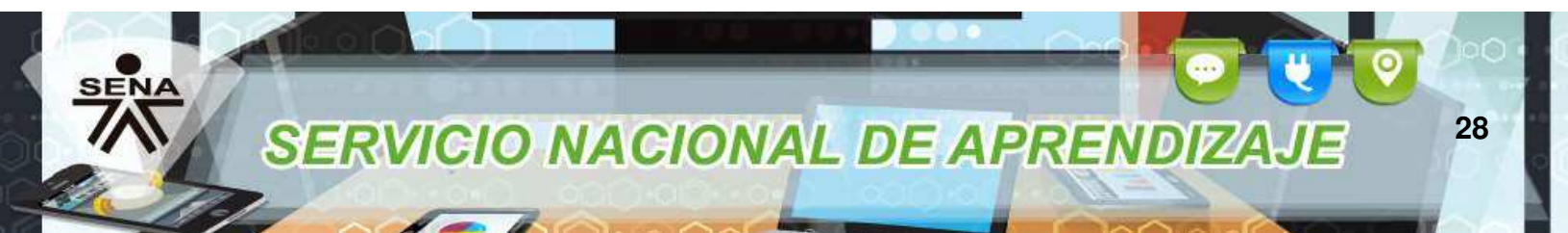
Fuente: SENA

Los nombres de variables solo deben contener caracteres alfanuméricos. Los caracteres de subrayado no están permitidos. Los números se permiten en los nombres de variables pero se desaconsejan en la mayoría de los casos.

Tipos de datos:

- **Booleanos:** en este tipo de variable almacena únicamente valores lógicos TRUE o FALSE.
- **Entero o Integer:** almacena números enteros positivos o negativos (... , -3, -2, -1, 0, 1, 2, 3, ...), pero los números no solo deben ser decimales, también puede ser binarios, octales o hexadecimales.
- **Cadenas de carácter o cadenas String:** este tipo de datos son de mucha importancia en cualquier lenguaje de programación ya que permite el almacenamiento de texto, PHP almacena cada carácter en un byte, por lo tanto solo permite el uso de 256 caracteres, no teniendo soporte para nativo Unicode (se espera que este sea uno de los grandes avances de la versión 6 del lenguaje), una variable de tipo String puede almacenar un texto de hasta 2 gigabytes, para asignar valor a una variable de tipo String se pueden usar comillas simples ('Texto') o comillas dobles ("Texto"), se recomienda que en el común de los casos se usen las comillas simples, a menos que se quieran usar caracteres de escape o introducir variables dentro de la cadena, igual como se explicó en el tema del constructor del lenguaje echo.
- **Números de punto flotante:** también se conocen como flotantes, dobles o números reales, que se refieren a números que además del componente entero tienen un valor decimal, en PHP pueden escribirse de cualquier de las siguientes formas, 3.1416, 2.3e2 (que equivaldría a 230) y 8E-3 (que equivaldría a 0.008).
- **Arreglos o Arrays:** en php los arreglos se tratan con un mapa de datos donde se asigna una llave a cada valor, para su declaración se requiere del constructor array(), su sintaxis es de la siguiente forma:

```
$arreglo = array (  
    llave => valor,  
    llave2 => valor2,  
    llave3 => valor3,
```





```
...  
);
```

Las llaves pueden ser solamente enteros o cadenas, los valores almacenados pueden ser de cualquier tipo.

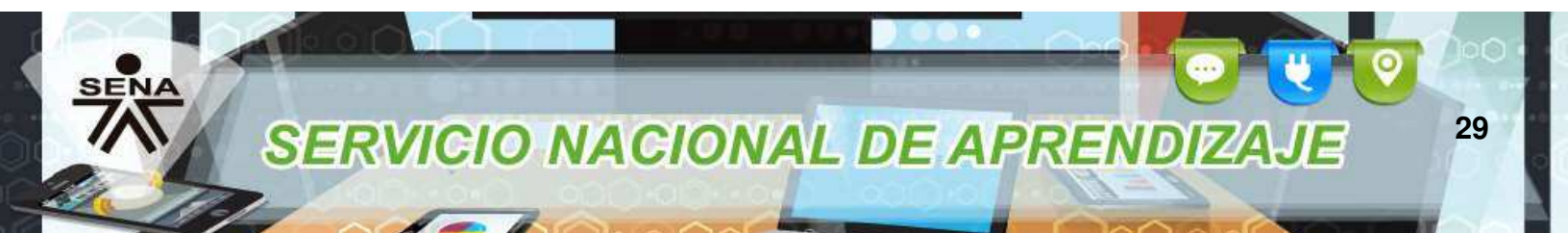
También puede crearse el arreglo obviando las llaves y el sistema pondrá los enteros del 0 en adelante como índices para cada valor así: `$arreglo = array (1, 2, 3, 4);`

En este caso el valor 1 tendrá como llave 0, el valor 2 tendrá como llave 1 y así sucesivamente.

Para referirse a un elemento individual del arreglo se debe hacer de la siguiente forma: `$arreglo[llave]`

- **Objetos:** son la forma en que PHP maneja el soporte para el paradigma Orientado a Objetos, se deben crear clases y luego instanciarlas mediante la declaración `new`, este tema se tratará a profundidad en las siguientes actividades de aprendizaje.
- **Recursos:** es un tipo de variable especial que resulta de las funciones que crean recursos externos, por ejemplo cuando se realizan conexiones a bases de datos o se cargan archivos de texto se crea un recurso externo al cual se apunta con la información contenida en una variable con este tipo de dato, se hará uso de este tipo de variables cuando se trate el tema de conexiones a bases de datos MySQL en las siguientes actividades de aprendizaje.
- **NULO o NULL:** básicamente es un valor representado por la constante NULL (el tema de constantes se tratará más adelante), una variable queda de tipo NULL cuando se declara pero no se le asigna ningún valor, es decir que no se inicializa (como se vio en ejemplo 6), cuando deliberadamente se le asigna la constante NULL o cuando se destruye el contenido de la variable con la función `unset()`.

En el ejemplo 8 se muestra el uso de algunos de los tipos de datos que se acaban de listar.





Ejemplo 8:



```
<!DOCTYPE html>
<html>
  <head>
    <title>Unidad 2 - Ejemplo 8</title>
    <meta http-equiv="Content-Type"
          content="text/html; charset=ISO-8859-
1" />
  </head>
  <body>
    <?php
      /* En este programa se declaran e
      inicializan diferentes tipos de
      * variables para dar un ejemplo de los
      diferentes tipos de datos
      * que se manejan en PHP          *
      */
      $nombreCompleto = 'Juan Pablo Casas
Casas';//tipo de dato String
      /*Como puede verse a continuación las fechas
se almacenan como tipo
      de datos string*/
      $fechaIngreso = '12/07/2005'; //Tipo de dato
String
      $edad = 26; //Tipo de dato entero
      $estatura = 1.68; //Tipo de dato coma
flotante
      $salario = 1.4e6; //Tipo de dato coma
flotante
    ?>

    <p>
      <?php
        echo "El se&ntilde;or $nombreCompleto
trabaja con la empresa "
          . "Sysdevelopment desde el
d&iacute;a $fechaIngreso, "
          . "su edad es $edad a&ntilde;os,
su estatura "
          . "es $estatura mts. y su
salario es \$$salario";
        ?>
      </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 2 / Ejemplo 8



Figura 11. Ejecución del ejemplo 8
Fuente: SENA

Se muestra a continuación el ejemplo 8 - 1 de código para la definición de arreglos, en el último ejemplo se puede ver que se repiten varias líneas de código para recorrer todo el arreglo, cuando se trate el tema de ciclos será más eficiente el proceso, ahorrando líneas de código mediante una rutina iterativa.

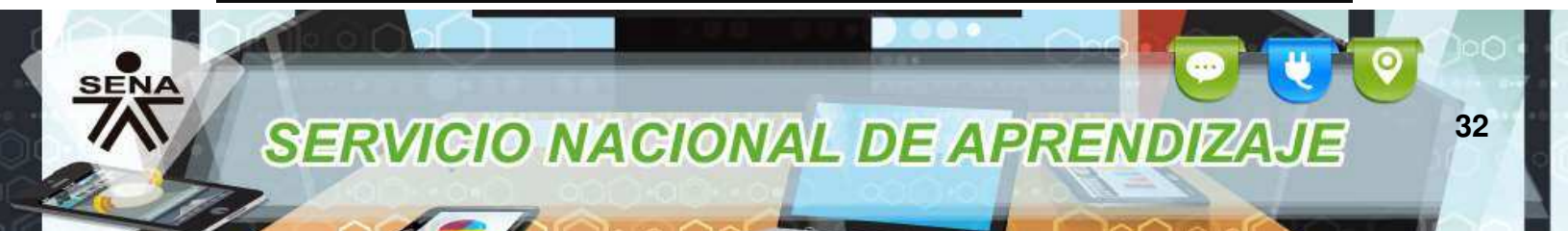
Ejemplo 8 - 1:

	<pre><!DOCTYPE html> <html> <head> <title>Unidad 2 - Ejemplo 8 - 1</title> <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" /> </head> <body> <h3>Ejemplo de arreglo con llaves num&eacute;ricas</h3> <?php /* En este programa se va a especificar la forma de * declarar arreglos y acceder a la información que contiene */ //Arreglo con llaves numéricas \$listadoProductosCodigoNombre = array(</pre>
--	---



```
1234 => "Arroz x libra",
2341 => "Frasco aceite x 1000 mls.",
2345 => "Mantequilla x 250 grs."

);
?>
<p>
    <?php
        $codigoProducto = 1234;
        echo "El producto con c&acute;digo
$codigoProducto es: "
        .
"$listadoProductosCodigoNombre[$codigoProducto]";
    ?>
</p>
<h3>Ejemplo de arreglo con llaves
alfanum&eacute;ricas</h3>
<?php
//Arreglo con llaves alfanum&eacute;ricas
$listadoDiasActividad= array(
    "lunes" => "caminar",
    "martes" => "sentadillas",
    "miercoles" => "flexiones",
    "jueves" => "abdominales",
    "viernes" => "estiramiento",
    "sabado" => "saltar lazo",
    "domingo" => "futbol"
);
?>
<p>
    <?php
        $dia = 'miercoles';
        echo "La actividad para el d&iacute;a
$dia es: "
        . "$listadoDiasActividad[$dia]";
    ?>
</p>
<?php
//Arreglo con llaves num&eacute;ricas autodefinidas
$listadoEstudiantes = array(
    "Ardila, Pablo",
    "Benitez, Nancy",
    "Cepeda, Juan",
    "Castro, Luis",
    "Soto, Carlos"
);
?>
<h3>Ejemplo de arreglo con llaves
num&eacute;ricas autodefinidas</h3>
<p>
    <?php
        echo "El primer estudiante de la lista
es: "
```





```
        . "$listadoEstudiantes[0] <br />";  
        echo "El segundo estudiante de la lista  
es: "  
  
        . "$listadoEstudiantes[1] <br />";  
        echo "El tercer estudiante de la lista  
es: "  
  
        . "$listadoEstudiantes[2] <br />";  
        echo "El cuarto estudiante de la lista  
es: "  
  
        . "$listadoEstudiantes[3] <br />";  
        echo "El quinto estudiante de la lista  
es: "  
  
        . "$listadoEstudiantes[4] <br />";  
        ?>  
    </p>  
</body>  
</html>
```

Fuente: SENA

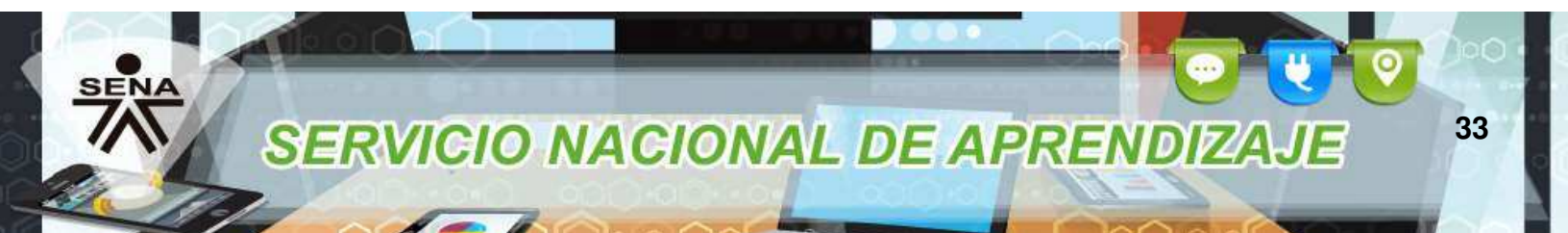
Descargue el segmento anterior del código como archivo .php del material complementario de este programa de formación en la siguiente ruta:

Materiales del programa / Materiales de apoyo / Documentos complementarios / Documentos complementarios: Actividad de aprendizaje 2 / Ejemplo 8 - 1



Figura 12. Ejecución del ejemplo 8 - 1

Fuente: SENA





Ejercicio 4:

Con base en el ejemplo anterior cree un arreglo que contenga un pequeño directorio telefónico con 3 personas con los siguientes datos: Nombre, Dirección, Teléfono y Fecha de cumpleaños. Muestre todos los datos del arreglo en el navegador.

Nota: para este ejercicio es necesario buscar información sobre cómo se crea un array multidimensional.


Constantes

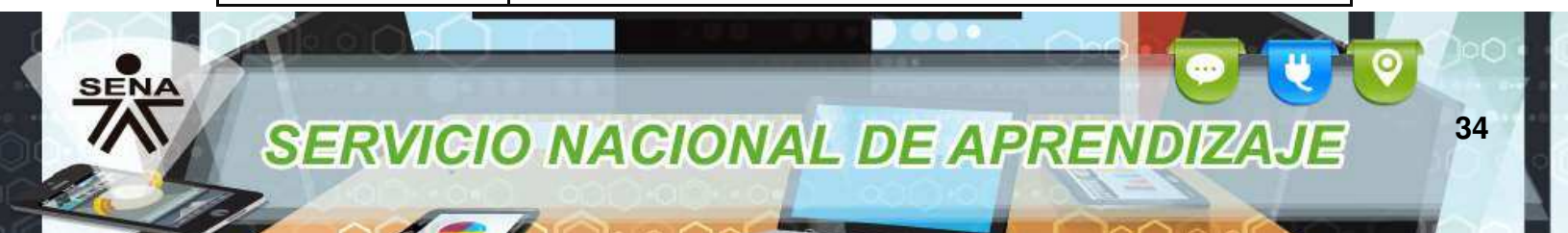
Una constante es un identificador que se define para hacer referencia a un valor simple y que como su nombre lo indica, se mantiene constante o invariable durante toda la ejecución de un programa.

Se puede definir usando la función `define("NOMBRE_CONSTANTE", "Valor constante")`, y se aconseja que el nombre de la constante siempre sea en mayúsculas, sus nombres solo pueden contener letras (A-Z), números (0-9) y caracteres de subrayado (`_`) pero el primer carácter del nombre solo puede ser una letra o un carácter de subrayado, nunca un número, una vez que la constante ha sido definida no puede ser cambiada o redefinida por ninguna otra instrucción, los tipos de datos de las constantes pueden ser booleano, entero, coma flotante o cadena.

No se aconseja que se use la forma `__NOMBRECONSTANTE__` cuando se nombra una constante puesto que esta es una convención que usa PHP para las “constantes mágicas” y es posible que en el futuro llegará a crearse una constante de este tipo en PHP con el mismo nombre que se usó y esto podría generar errores de lógica en el código escrito.

Ejemplo 9:

	<pre><!DOCTYPE html> <html> <head> <title>Unidad 2 - Ejemplo 9</title> <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859- 1" /> </head> <body> <p> <?php /* En este programa puede verse el proceso de declaración y * uso de una constante</pre>
---	---





```
*/
define('PI', 3.1416); //Declaración de
la constante
echo 'El número Pi es:' . PI;
//Uso de la constante
?>
</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 2 / Ejemplo 9

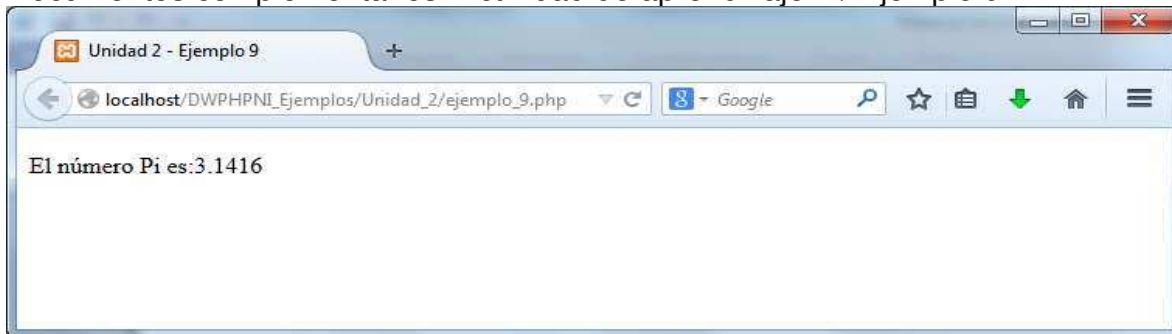


Figura 13. Ejecución del ejemplo 9

Fuente: SENA

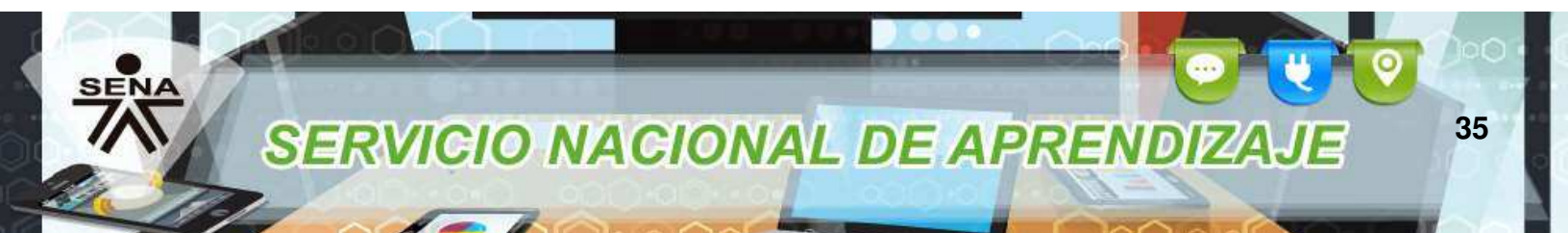
Convenciones de nombres para constantes

Las constantes pueden contener caracteres alfanuméricos y de subrayado, los números se permiten en los nombres de las constantes. Todas las letras utilizadas en el nombre de una constante deben estar en mayúsculas y las palabras separadas por caracteres de subrayado.

Por ejemplo: `PORCENTAJE_IMPUESTO_VALOR_AGREGADO` está permitido pero `PORCENTAJE_IMPUESTOVALORAGREGADO` no.

Operadores

PHP utiliza operadores para procesar diferentes tipos de datos, por ejemplo si se tienen dos valores enteros y se requiere realizar una multiplicación entre ellos se utilizará el operador aritmético para la multiplicación (*), por supuesto los aritméticos no son los





únicos operadores existentes, a continuación se mostrará una tabla con los tipos más importantes.

Si se requiere más información sobre los diferentes tipos de operadores consulte el siguiente enlace: <http://www.php.net/manual/es/language.operators.php>

Tabla 1. Operadores aritméticos

Ejemplo	Nombre	Resultado
-\$a	Negación	Opuesto de \$a.
\$a + \$b	Adición	Suma de \$a y \$b.
\$a - \$b	Sustracción	Diferencia de \$a y \$b.
\$a * \$b	Multiplicación	Producto de \$a y \$b.
\$a / \$b	División	Cociente de \$a y \$b.
\$a % \$b	Módulo	Resto de \$a dividido por \$b.

Fuente: The PHP Group (s.f.)

Tabla 2. Operadores de asignación

Ejemplo	Nombre	Resultado
\$a = \$b	Asignación	Asigna a la variable \$a el valor de \$b
\$a += \$b	Adición	Lo mismo que: \$a = \$a + \$b
\$a -= \$b	Sustracción	Lo mismo que: \$a = \$a - \$b
\$a *= \$b	Multiplicación	Lo mismo que: \$a = \$a * \$b
\$a /= \$b	División	Lo mismo que: \$a = \$a / \$b
\$a %= \$b	Módulo	Lo mismo que: \$a = \$a % \$b
\$a .= \$b	Concatenar	Lo mismo que: \$a = \$a . \$b

Fuente: The PHP Group (s.f.)

Tabla 3. Operadores de comparación

Ejemplo	Nombre	Resultado
\$a == \$b	Igual	TRUE si \$a es igual a \$b después de la





		manipulación de tipos.
\$a === \$b	Idéntico	TRUE si <i>\$a</i> es igual a <i>\$b</i> , y son del mismo tipo.
\$a !== \$b	Diferente	TRUE si <i>\$a</i> no es igual a <i>\$b</i> después de la manipulación de tipos.
\$a <> \$b	Diferente	TRUE si <i>\$a</i> no es igual a <i>\$b</i> después de la manipulación de tipos.
\$a !== \$b	No idéntico	TRUE si <i>\$a</i> no es igual a <i>\$b</i> , o si no son del mismo tipo.
\$a < \$b	Menor que	TRUE si <i>\$a</i> es estrictamente menor que <i>\$b</i> .
\$a > \$b	Mayor que	TRUE si <i>\$a</i> es estrictamente mayor que <i>\$b</i> .
\$a <= \$b	Menor o igual que	TRUE si <i>\$a</i> es menor o igual que <i>\$b</i> .
\$a >= \$b	Mayor o igual que	TRUE si <i>\$a</i> es mayor o igual que <i>\$b</i> .

Fuente: The PHP Group (s.f.)

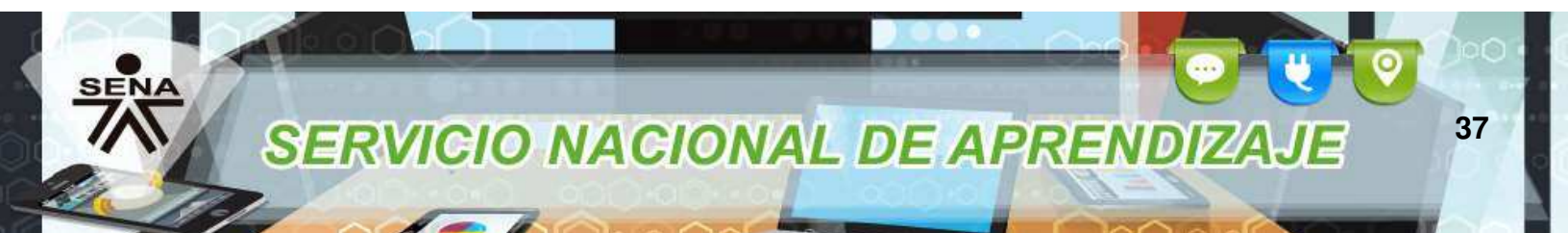
Tabla 4. Operadores de incremento/decremento

Ejemplo	Nombre	Efecto
++\$a	Pre-incremento	Incrementa <i>\$a</i> en uno, y luego retorna <i>\$a</i> .
\$a++	Post-incremento	Retorna <i>\$a</i> , y luego incrementa <i>\$a</i> en uno.
--\$a	Pre-decremento	Decrementa <i>\$a</i> en uno, luego retorna <i>\$a</i> .
\$a--	Post-decremento	Retorna <i>\$a</i> , luego decrementa <i>\$a</i> en uno.

Fuente: The PHP Group (s.f.)

Tabla 5. Operadores lógicos

Ejemplo	Nombre	Resultado
\$a and \$b	And (y)	TRUE si tanto <i>\$a</i> como <i>\$b</i> son TRUE .





\$a or \$b	Or (o inclusivo)	TRUE si cualquiera de <i>\$a</i> o <i>\$b</i> es TRUE .
\$a xor \$b	Xor (o exclusivo)	TRUE si <i>\$a</i> o <i>\$b</i> es TRUE , pero no ambos.
! \$a	Not (no)	TRUE si <i>\$a</i> no es TRUE .
\$a && \$b	And (y)	TRUE si tanto <i>\$a</i> como <i>\$b</i> son TRUE .
\$a \$b	Or (o inclusivo)	TRUE si cualquiera de <i>\$a</i> o <i>\$b</i> es TRUE .

Fuente: The PHP Group (s.f.)


Expresiones

Una expresión es una construcción sintáctica de PHP en la que básicamente existe algún valor, ya sea numérico, lógico o una cadena de carácter; toda rutina está escrita por expresiones que conforman las sentencias, estas últimas están delimitadas por el punto y coma (;).

Son ejemplos válidos de expresiones:

- `$a = 5 + 4`
- `$a = $b + 4`
- `$a = $b + $c`
- `$a = 4 > 3`
- `$a = $b > $c`
- `$a = $b || $c`
- `$a += 5`
- `$a++`
- `--$a`

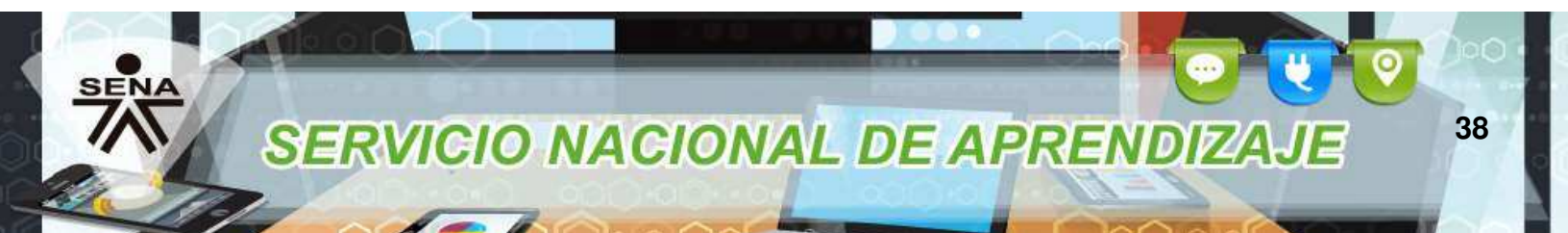
Ejemplo 10:



```

<!DOCTYPE html>
<html>
  <head>
    <title>Unidad 2 - Ejemplo 10</title>
    <meta http-equiv="Content-Type"
      content="text/html; charset=ISO-8859-
1" />
  </head>
  <body>
    <h3>Ejemplo de expresi&ocirc;n con operador
aritm&eacute;tico:</h3>

```





	<pre> <p> <?php /* En este programa se presenta el uso de diferentes tipos de * operadores en expresiones correctamente formuladas */ \$nombreTrabajador = 'Jose Pablo Caro Ospina'; //Se usa el operador de asignación = para darle valor a \$salario \$salario = 1.2e6; \$porcentajeReteFuente = 0.011; //Se usa el operador aritmético * para multiplicar \$reteFuente = \$salario * \$porcentajeReteFuente; \$totalDevengado = \$salario * (1- \$porcentajeReteFuente); echo "El trabajador \$nombreTrabajador tiene un salario de: \\$\$salario" . ", se le hace un descuento por retenci&ocute;n en la fuente de:" . "\\$\$reteFuente, el salario total que devenga el trabajador" . "es de: \\$\$totalDevengado"; ?> </p> <h3>Ejemplo de expresi&ocute;n con operador de asignaci&ocute;n: </h3> <p> <?php \$manzanas = 3; \$regalo = 4; /* Se usa el operador de asignación += para sumar el valor de una * variable a otra */ echo "Pablo tiene \$manzanas manzanas, Mar&iacute;a le regala \$regalo " . "manzanas, por lo tanto ahora Pablo tiene " . (\$manzanas += \$regalo) . " manzanas"; ?> </p> <h3>Ejemplo de expresi&ocute;n con operador de incremento: </h3> <p> Usando pre-incremento: <?php </pre>
--	--





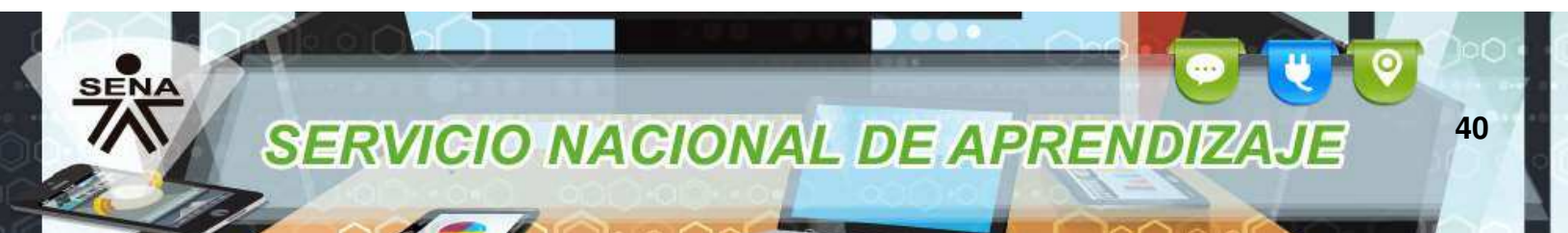
```
$edad = 21;
// Se usa el operador de pre-incremento
++
    echo "El a&ntilde;o pasado Nancy
Ten&iacute;a $edad a&ntilde;os, "
        . "puesto que este a&ntilde;o
ella ya cumpli&oacute; ahora "
        . "tiene ". ++$edad . "
a&ntilde;os";
?>
<b>Usando post-incremento: </b>
<?php
    $edad = 21;
    echo "El a&ntilde;o pasado Nancy
Ten&iacute;a ". $edad++
        . " a&ntilde;os, puesto que este
a&ntilde;o ella ya "
        . "cumpli&oacute; ahora tiene
$edad a&ntilde;os";

?>
</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 2 / Ejemplo 10



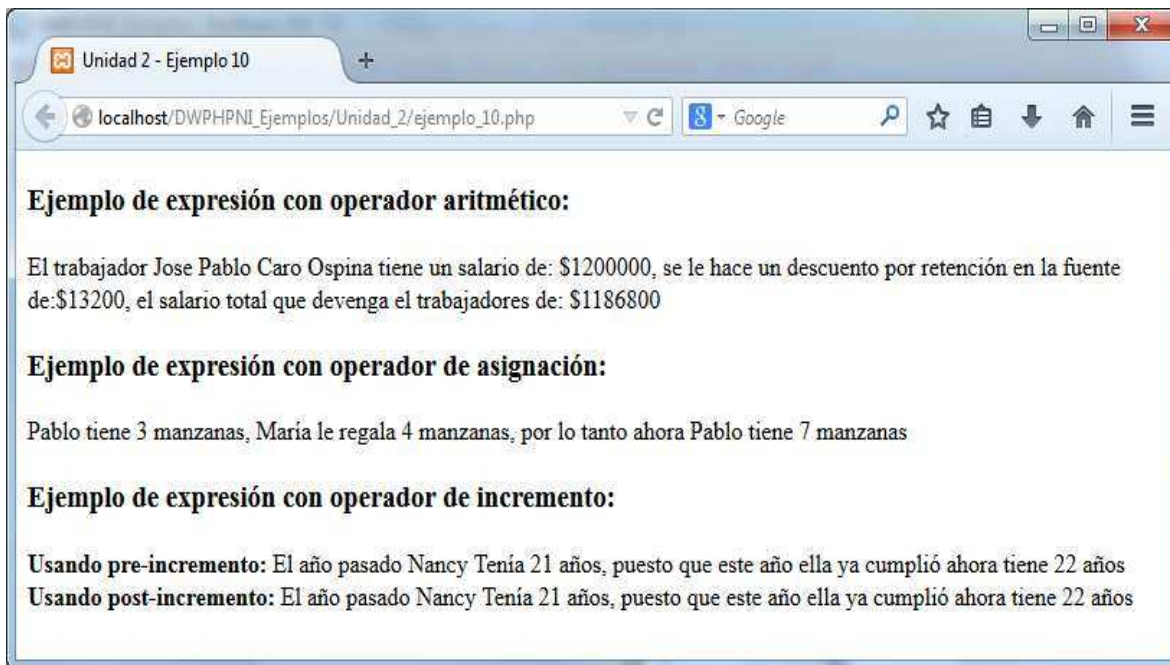
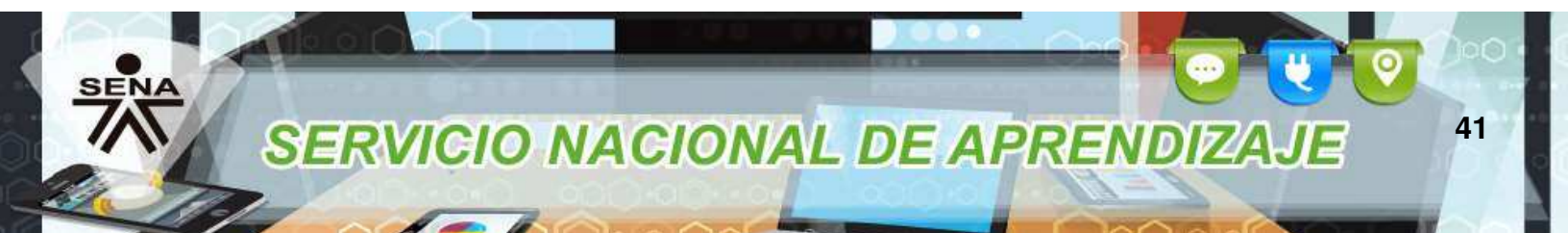


Figura 14. Ejecución del ejemplo 10
Fuente: SENA

Asignación por valor y por referencia

Por defecto cuando existe una expresión de asignación entre dos variables se está haciendo una asignación por valor, es decir que en la expresión `$a = $b`, lo que se le está indicando a PHP es que exactamente el mismo dato o valor que contiene `$b` se le va a asignar a la variable `$a`, por lo tanto si `$b` tiene un valor de 5 luego de la asignación tanto `$a` como `$b` tendrá el valor 5.

Por otra parte, en PHP se cuenta con otra forma de asignar los valores de una variable a otra y es por referencia, en este caso la expresión se escribe anteponiéndole un signo ampersand a la variable cuya referencia se va a asignar en otra, la expresión se vería así: `$a = &$b`, en esta expresión se almacena dentro de la variable `$a` una referencia a la variable `$b` y lo que sucede es que ahora la variable `$a` va a “apuntar” (no como en los apuntadores que se usan en otros lenguajes sino simplemente como un nombre alternativo) a la variable `$b`, lo conveniente de esto (ya que puede servir para solucionar problemas de lógica en una rutina) es que si en cualquier momento se modifica la variable `$a` automáticamente se va a modificar también el contenido de `$b`.





Por supuesto, sólo se dan asignaciones por referencia utilizando variables y no literales u otros elementos, por lo tanto la expresión: `$a = &5`, no guardaría una referencia sino la cadena “&5” y nada más.

Ejemplo 11:

	<pre> <!DOCTYPE html> <html> <head> <title>Unidad 2 - Ejemplo 11</title> <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859- 1" /> </head> <body> <h3>Ejemplo de asignaci&ocute;n de variables por valor:</h3> <p> <?php /* En este programa se muestra como al asignar el contenido de * una variable a otra por valor las modificaciones de una * variable no afectan a la otra */ \$edad = 6; \$edadFutura = \$edad; //Se asigna el valor de \$edad a \$edadFutura \$edadFutura += 4; //Se modifica el valor de \$edadFutura echo "En este momento Pablo tiene \$edad a&ntilde;os, cuando pasen" . " 4 a&ntilde;os Pablo tendr&aacute; \$edadFutura a&ntilde;os"; ?> </p> <h3>Ejemplo de asignaci&ocute;n de variables por referencia:</h3> <p> <?php /* En este caso se muestra como al asignar el contenido de * una variable a otra por referencia las modificaciones de una * variable si afectan a la otra */ \$edad = 6; \$edadFutura = &\$edad; //Se asigna la REFERENCIA a \$edad en \$edadFutura echo "- En este momento Pablo tiene </pre>
--	--



```
$edad a&ntilde;os <br /> ";
/* A continuación se le va a asignar a
$edadFutura el valor de lo
* que esta contenga más 4, pero como en
$edadFutura no hay un valor
* sino una referencia lo que va a pasar
entonces es que el valor
* de $edad se va a modificar sumándole
4
*/
$edadFutura += 4;
/* Como puede verse en la siguiente
línea no se utiliza la
* variable $edadFutura sino que se usa
$edad y aunque esta
* no se ha modificado directamente sino
por referencia el nuevo
* valor será el que contenía antes + 4
*/
echo "- Cuando pasen 4 a&ntilde;os Pablo
tendr&aacute;"; $edad a&ntilde;os";
?>
</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 2 / Ejemplo 11

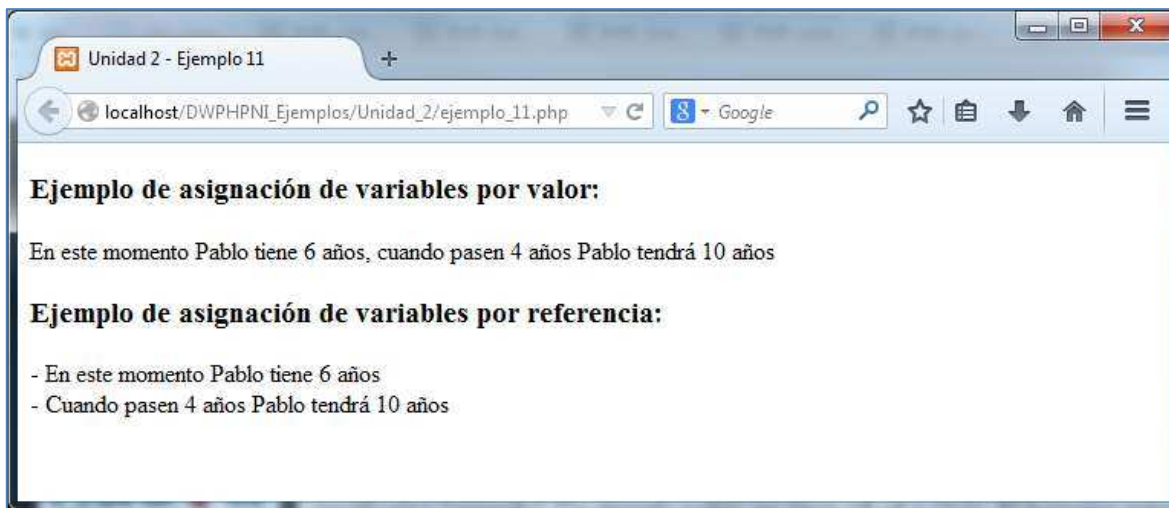




Figura 16. Ejecución del ejemplo 11
Fuente: SENA

3. Estructuras de control

Hasta el momento se ha trabajado con sentencias simples que se ejecutan siempre de manera lineal y cada una se ejecuta solo una vez, pero este tipo de programación lineal permite muy poco al desarrollar una rutina compleja, para poder avanzar en las posibilidades del lenguaje es necesario tratar el tema de las estructuras de control que permiten al desarrollador hacer que solo se ejecuten ciertas sentencias cuando sea necesario o que algunas se repitan para hacer más eficiente el código.

Condicionales if, else, elseif

Las estructuras condicionales permiten modificar el flujo de un programa para que ya no se ejecuten todas las instrucciones de forma lineal, sino que solo se ejecuten las que se necesitan, siempre y cuando se cumplan o no unas condiciones específicas (de allí su nombre de condicionales); básicamente su estructura se fundamenta en la evaluación de una expresión o condición y en el agrupamiento de las sentencias que se van a ejecutar cuando la condición se cumpla y en el momento que esta no lo haga.

Una estructura `if` simple evalúa la condición y permite ejecutar las sentencias que tenga agrupadas con la siguiente estructura:

```
if (condición) {  
    sentencias a ejecutar  
    ...  
}
```

Aunque en la estructura anterior se pueden obviar los corchetes cuando sólo se tiene una sentencia que va a estar condicionada, no se recomienda que se obvien los corchetes ni siquiera en este caso, puesto que si en el futuro se requiere adicionar una línea más para que esté condicionada y se le olvida adicionar los corchetes, se van a generar errores de lógica en el programa.

Se pueden anidar estructuras condicionales de forma infinita, esto implica que una estructura este dentro del ámbito de la otra y se hace de la siguiente forma:

```
if (condición) {  
    sentencias a ejecutar  
    ...  
}
```





```
if (condición) {  
    if (condición) {  
        sentencias a ejecutar  
        ...  
    }  
}  
sentencias a ejecutar  
...  
}
```

En la estructura anterior se muestra que en el primer nivel del condicional hay sentencias antes y después del segundo nivel; lo cual no es obligatorio, solo se muestra para que se comprenda que es posible hacerlo.

Además de tener un grupo de sentencias que se ejecutarán cuando la condición se cumple, se requiere también tener un grupo de sentencias que se ejecutarán solo cuando la condición no se cumple, se usa la estructura adicional `else`, que quiere decir algo así como “de lo contrario” y se hace de la siguiente forma:

```
if (condición) {  
    sentencias a ejecutar si la condición se cumple  
    ...  
} else {  
    sentencias a ejecutar si la condición NO se cumple  
    ...  
}
```

Finalmente, si se requiere evaluar más de una condición dentro de la misma estructura y en cada caso se ejecutará sentencias distintas, y por la lógica que se aplica esto no se puede hacer mediante la anidación de estructuras `if`, se debe usar la estructura adicional `elseif`, con esta opción se pueden tener tantas condiciones como sea necesario; la estructura es la siguiente:

```
if (condición) {  
    sentencias a ejecutar si la primera condición se cumple  
    ...  
} elseif (condición) {  
    sentencias a ejecutar si la primera condición NO se  
    cumple pero si la segunda  
    ...  
} elseif (condición) {  
    sentencias a ejecutar si NO se cumple ni la primera ni la  
    segunda condición pero si la tercera
```





```

...
} else {
    sentencias a ejecutar si ninguna de las condiciones se
    cumple
...
}

```

Condiciones

Las condiciones que se utilizan en diferentes estructuras pueden estar construidas como expresiones que normalmente usan operadores de comparación y operadores lógicos, por ejemplo: `if ($a<=$b)` que solo sería cierta cuando la variable `$a` tenga un valor que pueda ser evaluado como “menor o igual que” `$b`, o `if (($a<=$b) && ($b<$c))` que solo será cierta si el contenido de la variable `$a` puede ser evaluado como “menor o igual que” `$b` “y si y solo si” el contenido de `$b` puede ser evaluado como “menor que” `$c`, por lo tanto `$a` debe ser “menor que” `$c`. Por otra parte también pueden utilizarse los valores lógicos retornados por funciones como en el ejemplo: `if (isset($a))` donde se usa la función `isset()` que evalúa si la variable `$a` ya fue inicializada y de ser así devuelve un valor `TRUE` de lo contrario devuelve un valor `FALSE`.

Ejemplo 12:

	<pre> <!DOCTYPE html> <html> <head> <title>Unidad 2 - Ejemplo 12</title> <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859- 1" /> </head> <body> <h3>Ejemplo de condicionales anidados:</h3> <p> <?php /* En este programa se muestra cómo se pueden usar las estructuras * condicionales del PHP */ \$a = 3; \$b = 4; if (\$a >= 3) { echo "La variable \\$a es mayor o igual que 3
"; if (\$b >= 4) { echo "La variable \\$b es mayor o </pre>
--	--



```
igual que 4";
    }
    }
    ?>
</p>
<h3>Ejemplo de condicionales multiples:</h3>
<p>
    <?php
    $a = 3;
    if ($a > 3) {
        echo "La variable \$a es mayor que 3
<br />";
    } elseif ($a < 3) {
        "La variable \$a es menor que 3 <br
/>";
    } else {
        echo "La variable \$a es igual a 3
<br />";
    }
    ?>
</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 2 / Ejemplo 12

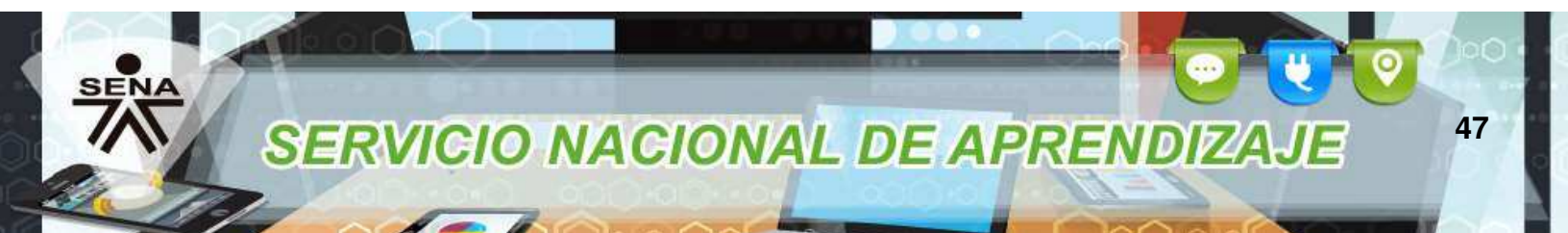




Figura 17. Ejecución del ejemplo 12
Fuente: SENA

Ciclos condicionales while y do-while

En muchas ocasiones el solo hecho de cambiar el flujo del programa según se cumpla o no una condición no es suficiente para lograr la solución de un problema de programación, puede requerirse además que el grupo de sentencias se ejecuten varias veces hasta que la condición original ya no se cumpla, es allí donde entran los ciclos condicionales `while` y `do-while` (la traducción literal de estas instrucciones sería: mientras y haga-mientras, lo cual indica que el ciclo se repetirá “mientras” la condición siga cumpliéndose), no solo evalúan una condición sino que además repiten la ejecución de las mismas sentencias hasta que los cambios hechos dentro de dichas sentencias en la variable evaluada hagan que la condición ya no se cumpla.

En ciclo condicional `while` evalúa primero la condición y si se cumple ejecuta las sentencias agrupadas (de lo contrario no las ejecuta ni siquiera una vez) y se repite hasta que la condición ya no se cumpla; se usa de la siguiente forma:

```
while (condición) {
    sentencias a ejecutar repetitivamente mientras la
    condición se cumpla
    sentencia que modifica la variable evaluada
    ...
}
```



Fuente: SENA

En la anterior estructura es muy importante la línea que dice “sentencia que modifica la variable evaluada”, puesto que dentro de las sentencias agrupadas por el `while` no hay nada que modifique la variable que se evalúa en la condición, esta seguirá cumpliéndose siempre y por lo tanto se generará un bucle infinito.

El ciclo condicional `do-while` funciona un poco diferente al ciclo anterior, ya que está diseñado para que las sentencias agrupadas por este se ejecuten siempre al menos una vez y luego si se evalúa la condición, lo cual implica que las sentencias podrían ejecutarse solo una vez si la condición del ciclo no se cumple pasando el control del programa a las sentencias que siguen después de este.





Se usa de la siguiente forma:

```
do {
    sentencias que se ejecutarán al menos una vez antes de
    evaluar la condición y seguirán ejecutándose mientras la
    condición se cumpla
    sentencia que modifica la variable evaluada
    ...
} while (condición);
```

Es conveniente hacer notar que en esta estructura se debe poner un punto y coma después de la sentencia `while (condición)`, puesto que de lo contrario el analizador del lenguaje verá esta línea como una sentencia sin terminar.



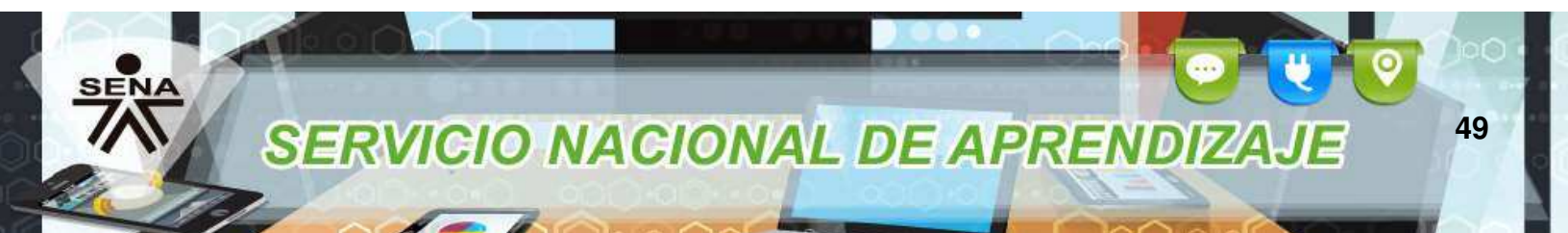
Fuente: SENA

Es muy importante tener cuidado en la lógica que se usa con los ciclos condicionales, ya que fácilmente pueden crearse ciclos infinitos que pueden desbordar los recursos del sistema y bloquearlo, por lo tanto debe evaluarse muy bien que la condición llegue a cumplirse en cualquier caso y que exista dentro de las sentencias agrupadas, algo que modifique el valor de la variable evaluada en la condición.

Ejemplo 13:



```
<!DOCTYPE html>
<html>
  <head>
    <title>Unidad 2 - Ejemplo 13</title>
    <meta http-equiv="Content-Type"
      content="text/html; charset=ISO-8859-
1" />
  </head>
  <body>
    <h3>Ciclo while</h3>
    <p>
      <?php
        /* En este programa se muestra cómo se
        pueden los ciclos
          * while y do-while
          */
        $a = 1; //Se inicializa una variable de
        control
        while ($a <= 5) { //Se pone la condición
        de finalización
```





	<pre> echo "\$a
"; //Se ejecuta una rutina \$a++; //Se modifica la variable de control } ?> </p> <h3>Ciclo do-while</h3> <p> <?php /* Este programa solo mostrará el número 6 ya que * no se devuelve la variable \$a a un valor menor que 5 por lo tanto * se muestra el ultimo valor luego de la ejecución del ciclo * anterior, se evalua la condición y puesto que no se cumple no se * itera de nuevo */ do { rutina echo "\$a
"; //Se ejecuta una \$a++; //Se modifica la variable de control } while (\$a <= 5) //Se pone la condición de finalización ?> </p> </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 2 / Ejemplo 13



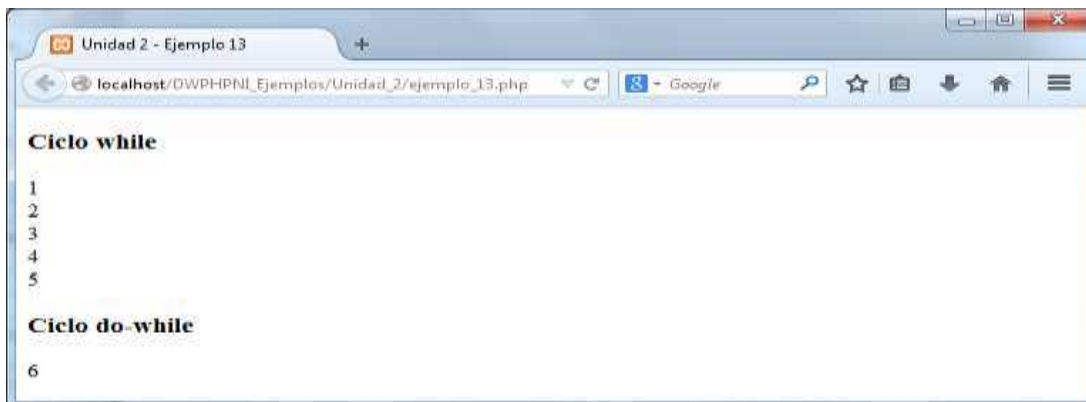


Figura 15. Ejecución del ejemplo 13
Fuente: SENA

Ejercicio 5:

Modifique el ejemplo anterior para que el ciclo while muestre los mismos números en orden inverso.

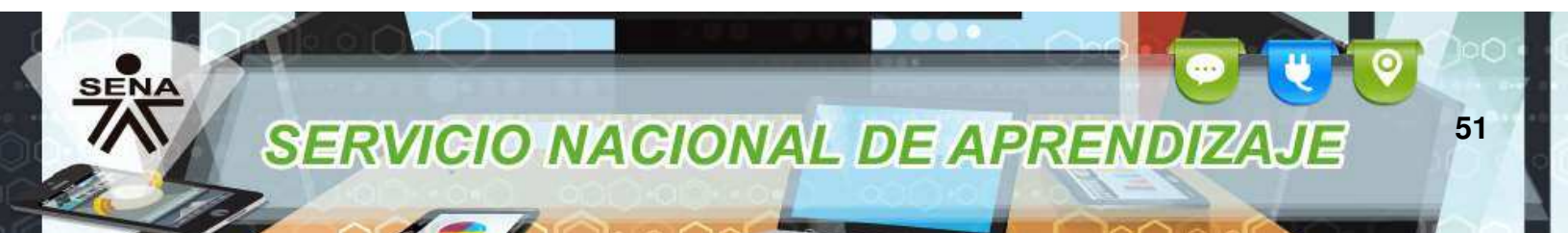
Ciclos for y foreach

El ciclo `for` es útil cuando se requiere ejecutar un grupo de sentencias por un número determinado de veces y además se necesita un índice que puede ser utilizado dentro de la lógica de las sentencias, se usa de la siguiente manera:

```
for (inicialización del índice; condición de terminación;  
modificación del índice) {  
    sentencias a ejecutar repetitivamente  
    ...  
}
```

El ciclo `foreach` fue diseñado con el fin de recorrer arreglos, es bastante útil pues recorrer arreglos es una tarea que se requiere reiterativamente en la programación, este ciclo solo funciona sobre los arreglos y los objetos. Se usa de las siguientes formas:

```
foreach ($arreglo as $valor) {  
    sentencias a ejecutar mientras haya elementos del arreglo  
    por recorrer  
    ...  
}  
foreach ($arreglo as $llave => $valor) {  
    sentencias a ejecutar mientras haya elementos del arreglo  
    por recorrer
```





}

En ambos casos el ciclo inicia siempre en el primer índice del arreglo, no es necesario usar ninguna instrucción para mover el puntero del arreglo a la primera posición. En el primer caso en cada iteración del ciclo se almacena solo el valor de cada posición del arreglo en la variable `$valor`, en el segundo caso se almacena la llave de cada posición en la variable `$llave` y el valor en la variable `$valor`.

Ejemplo 14:

	<pre><!DOCTYPE html> <html> <head> <title>Unidad 2 - Ejemplo 14</title> <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859- 1" /> </head> <body> <h3>Ciclo for</h3> <p> <?php /* En este programa se muestra cómo se pueden los ciclos * for y foreach */ for (\$i = 1; \$i <= 5; \$i++) { //Se ponen todas las condiciones del ciclo echo "\$i
"; //Se ejecuta una rutina } ?> </p> <h3>Ciclo foreach</h3> <p> <?php /* Puesto que el ciclo foreach fue creado para recorrer arreglos * lo primero que se requiere es tener un arreglo con información */ \$arregloAnimales = array("Perro", "Gato", "Conejo", "Liebre", "Tortuga"</pre>
--	--



```
);  
foreach ($arregloAnimales as $animal) {  
    echo "$animal <br />";  
}  
?>  
</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 2 / Ejemplo 14

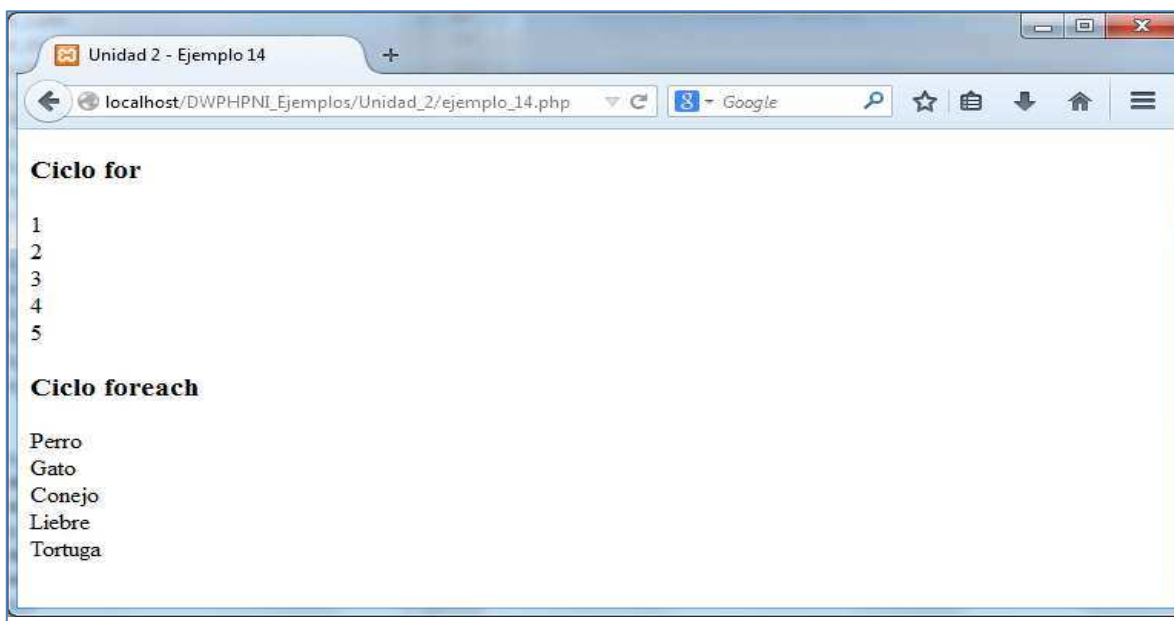
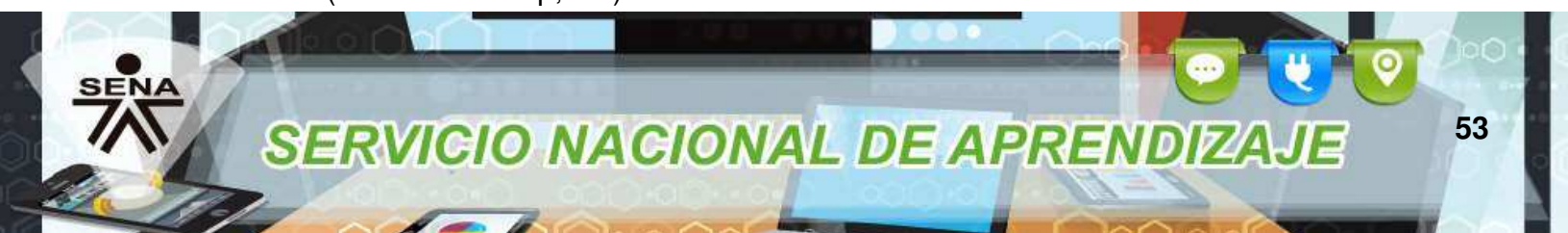


Figura 16. Ejecución del ejemplo 14

Fuente: SENA

Instrucciones break y continue

Ambas se usan para modificar el funcionamiento de las estructuras `for`, `foreach`, `while`, `do-while`, `switch` (que se explicará más adelante). En el caso de `break` se usa para salir totalmente de la estructura de control terminando su ejecución (literalmente `break` es romper en español) y `continue` por el contrario solo detiene la iteración actual devolviendo el control al punto en el que se evalúa la condición del ciclo. Ambas se usan de la mano con la estructura `if` ya que se requiere de evaluar alguna condición previo a su uso. (The PHP Group, s.f.)





Estas instrucciones aceptan como argumento opcional un número entero que se usa en las estructuras cuando están anidadas para indicar hasta que nivel del anidamiento se debe salir. A continuación se muestra su sintaxis:

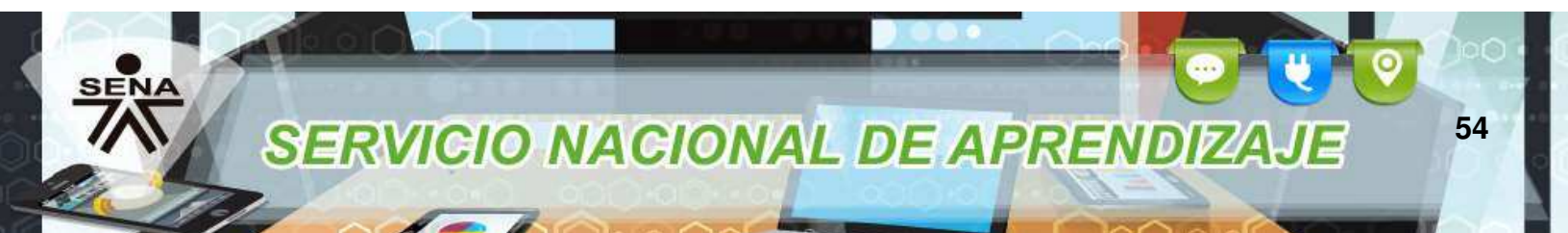
```
while (condición) {  
    sentencias  
    if (condición) {  
        sentencias  
        break n; //n es opcional en este caso sería 1 pues  
                //solo hay un nivel de anidación  
    }  
    sentencias  
    ...  
}
```

```
while (condición) {  
    sentencias  
    if (condición) {  
        sentencias  
        continue n; //n es opcional en este caso sería 1 pues  
                   //solo hay un nivel de anidación  
    }  
    sentencias  
    ...  
}
```

Sentencia Switch

Esta estructura se usa como una solución alternativa a la necesidad de evaluar varios valores de una misma variable, para no tener que generar un numero extenso de estructuras if. Su estructura es la siguiente:

```
switch ($i) {  
    case 0:  
        sentencias a ejecutar si la variable tiene el valor 0  
        break;  
    case 1:  
        sentencias a ejecutar si la variable tiene el valor 0  
        break;  
    case 2:  
        sentencias a ejecutar si la variable tiene el valor 2
```





```
        break;
    case ...
        break;
    default:
        sentencias a ejecutar si la variable tiene un valor
        diferente a todos los evaluados
}
```

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 2



Referencias

- 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.). *Variables predefinidas*. Consultado el 30 de junio de 2015, en <http://www.php.net/manual/es/reserved.variables.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