

Diplomado en:

Programación en PHP

Guía didáctica N° 2



Formación Virtual

.....educación sin límites

GUÍA DIDÁCTICA N°2

M2-DV59-GU01

MÓDULO 2: CONDICIONALES - CICLOS – ARREGLOS

DIPLOMADO EN PROGRAMACIÓN EN PHP

© DERECHOS RESERVADOS - POLITÉCNICO DE COLOMBIA, 2019
Medellín, Colombia

Proceso: Gestión Académica

Realización del texto: Diego Palacio, Docente

Revisión del texto: - Jehison Posada, Asesor Gramatical

Diseño: Cristian Quintero, Diseñador Gráfico

Editado por el Politécnico de Colombia

ÍNDICE

PRESENTACIÓN	4
COMPETENCIAS.....	5
TEMA 1 Condicionales.....	6
TEMA 3 Switch Case	31
TEMA 3 Ciclos.....	42
TEMA 4 Arrays.....	68
ASPECTOS CLAVES.....	99

PRESENTACIÓN

La Guía Didáctica N°2 del MÓDULO 2: CONDICIONALES - CICLOS – ARREGLOS, es un material que ha sido desarrollado para el apoyo y orientación del participante en el *Diplomado en Programación en PHP*, especialmente, está orientada a la consolidación y/o desarrollo de las habilidades y destrezas necesarias para generar unas adecuadas bases en lo que concierne a la programación en PHP.

Como bien conoces, el objetivo principal de este módulo número 2 es comprender algunos de los conceptos básicos del lenguaje de PHP en cuanto a: arrays, condicionales y ciclos principalmente.

Para ello, se ha organizado en esta guía cinco (4) contenidos temáticos, entre ellos: (a) condicionales, (b) switch case, (c) ciclos, y (e) arrays, que ayudarán a formar unas adecuadas bases.

COMPETENCIAS

Se espera que con los temas abordados en la Guía Didáctica N°1 del MÓDULO 1: CONDICIONALES – CICLOS – ARREGLOS, el estudiante logre la siguiente competencia:



Comprender algunos de los conceptos básicos del lenguaje de PHP en cuanto a: condicionales, ciclos y arrays.

Indicadores de logro:

- Comprende la sintaxis y adecuada estructura de los tipos de ciclos.
- Entiende el correcto funcionamiento y aplicación de las condicionales y sentencia de control Switch Case.
- Comprende el funcionamiento y aplicación de los arrays.

TEMA 1

Condicionales

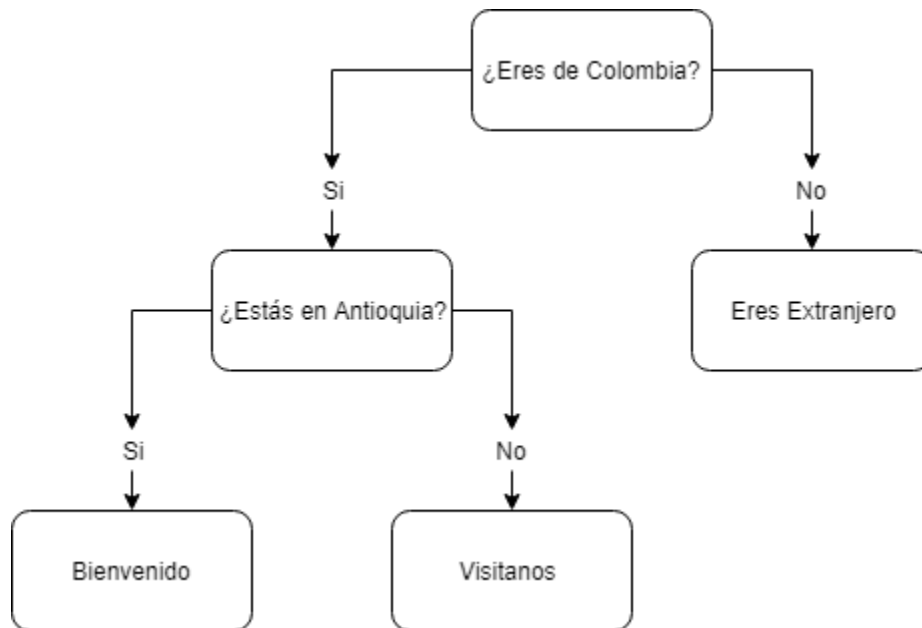
Las declaraciones condicionales se utilizan para realizar diferentes acciones basadas en diferentes condiciones.

Muy a menudo cuando escribe código, se desea realizar diferentes acciones a partir de diferentes condiciones. Puede usar declaraciones condicionales en su código para hacer esto.

Además, en estas condicionales se hace uso de algunos de los operadores vistos en el tema número 1.

PHP tiene las siguientes declaraciones condicionales:

Instrucción	Describiendo
If	Ejecuta algún código si una condición es verdadera
Else	Ejecuta un código si una condición es verdadera y otro código si esa condición es falsa
elseif	Ejecuta diferentes códigos para más de dos condiciones
anidadadas	Grandes bloques de código con X números de condicionales
switch case	Selecciona uno de los muchos bloques de código condicional.

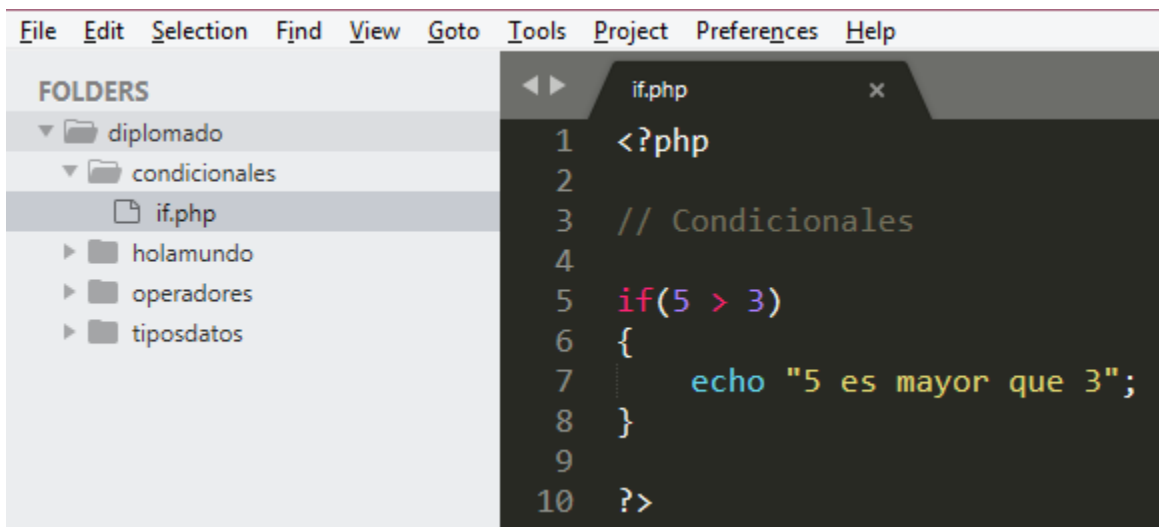


Sentencia if

La instrucción if ejecuta algún código si una condición es verdadera.

La estructura condicional más simple en PHP es el if, ésta se encarga de evaluar una condición, que para el correcto funcionamiento retornará un valor booleano, en caso de que se cumpla (true) se ejecuta el contenido entre las llaves "{}".

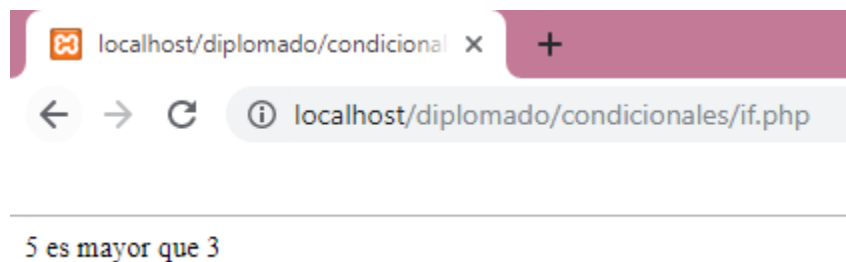
```
if (condition) {  
    code to be executed if condition is true;  
}
```



```

1  <?php
2
3  // Condicionales
4
5  if(5 > 3)
6  {
7      echo "5 es mayor que 3";
8  }
9
10 ?>

```



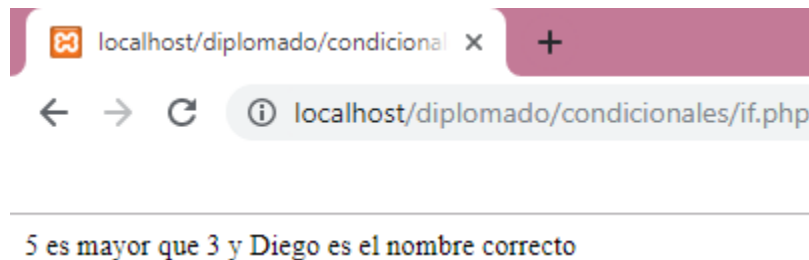
Operadores lógicos

Antes de entrar en mayor detalle de las condicionales, es importante tener en cuenta las características que tienen en relación con las condicionales, los operadores lógicos. Los operadores lógicos en PHP y en otros lenguajes se usan para combinar declaraciones condicionales y obtener un resultado verdadero o falso a raíz de estas combinaciones.

Ejemplo	Nombre	Resultado
\$a and \$b	and (y)	TRUE si tanto \$a como \$b son TRUE.
\$a or \$b	or	TRUE si cualquiera de \$a o \$b es TRUE.
\$a xor \$b	xor	TRUE si \$a o \$b es TRUE, pero no ambos.
!\$a	not	TRUE si \$a es FALSE.
\$a && \$b	and (y)	TRUE si tanto \$a como \$b son TRUE.
\$a \$b	or	TRUE si cualquiera de \$a o \$b es TRUE.

Un ejemplo con esta tabla de operadores lógicos y en especial el operador “&&” sería el siguiente:

```
1 <?php
2
3 // Condicionales
4
5 if(5 > 3 && "Diego" == "Diego")
6 {
7     echo "5 es mayor que 3 y Diego es el nombre correcto";
8 }
9
10 ?>
```



localhost/diplomado/condicional x +

← → ↻ ⓘ localhost/diplomado/condicionales/if.php

5 es mayor que 3 y Diego es el nombre correcto

En la condicional se encuentran 2 sentencias a evaluar: (1) $5 > 3$ y (2) "Diego" == "Diego" separadas por el operador lógico "&&" que se encarga de evaluar que ambas sentencias (1) y (2) sean verdaderas para el cumplimiento del "if".

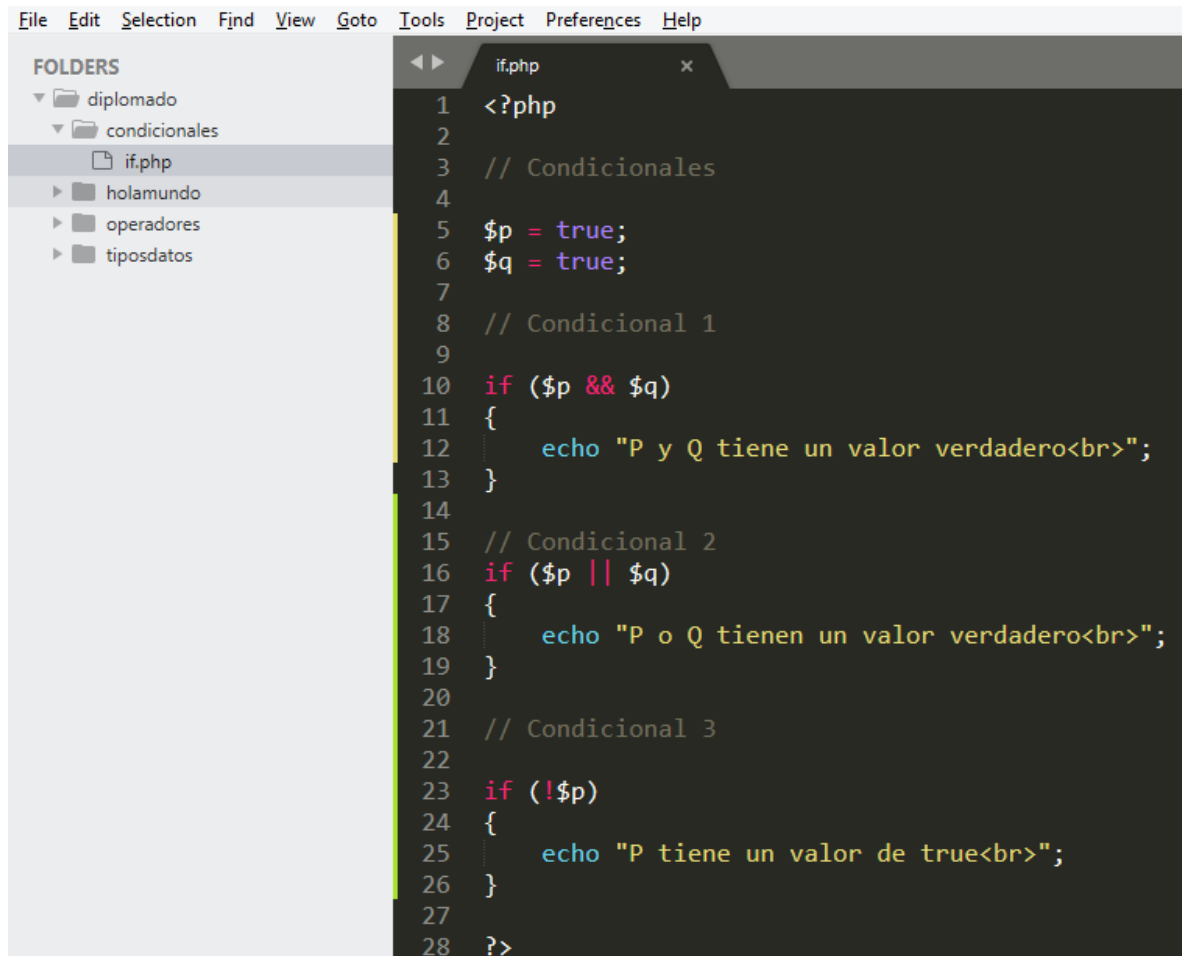
- (1) Retornará verdadero debido a que 5 sí es mayor que 3.
- (2) Retornará verdadero debido a que "Diego" es igual a "Diego".

Dado que ambas sentencias de condición retornan verdadero, la condición del if se cumple. Observe la siguiente tabla de verdad de algunos de los operadores lógicos puesto aprueba con algunos resultados para dar mayor claridad.

P	Q	P && Q	P Q	!P
True	True	True	True	False
True	False	False	True	False
False	True	False	True	True
False	False	False	False	True

En la tabla se encuentran 2 variables o valores booleanos (1) P y (2) Q, con un valor de inicialización (true o false) y 3 series de casos dónde se pone a prueba la variación a la que se pueden enfrentar, hay que tener presente la tabla de operadores lógicos para entender el siguiente ejercicio.

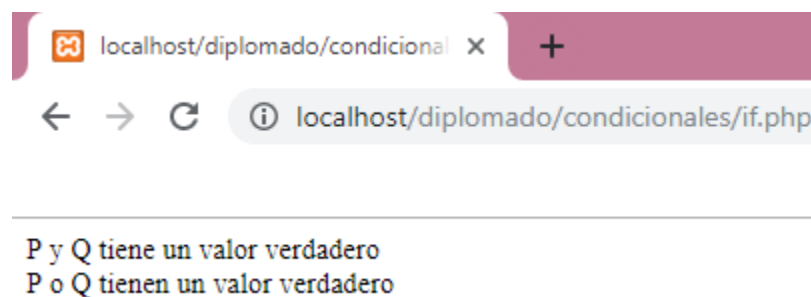
Evaluación de la primera fila



```
File Edit Selection Find View Goto Tools Project Preferences Help

FOLDERS
└─ diplomado
  └─ condicionales
    └─ if.php
  └─ holamundo
  └─ operadores
  └─ tiposdatos

if.php
1 <?php
2
3 // Condicionales
4
5 $p = true;
6 $q = true;
7
8 // Condicional 1
9
10 if ($p && $q)
11 {
12     echo "P y Q tiene un valor verdadero<br>";
13 }
14
15 // Condicional 2
16 if ($p || $q)
17 {
18     echo "P o Q tienen un valor verdadero<br>";
19 }
20
21 // Condicional 3
22
23 if (!$p)
24 {
25     echo "P tiene un valor de true<br>";
26 }
27
28 ?>
```



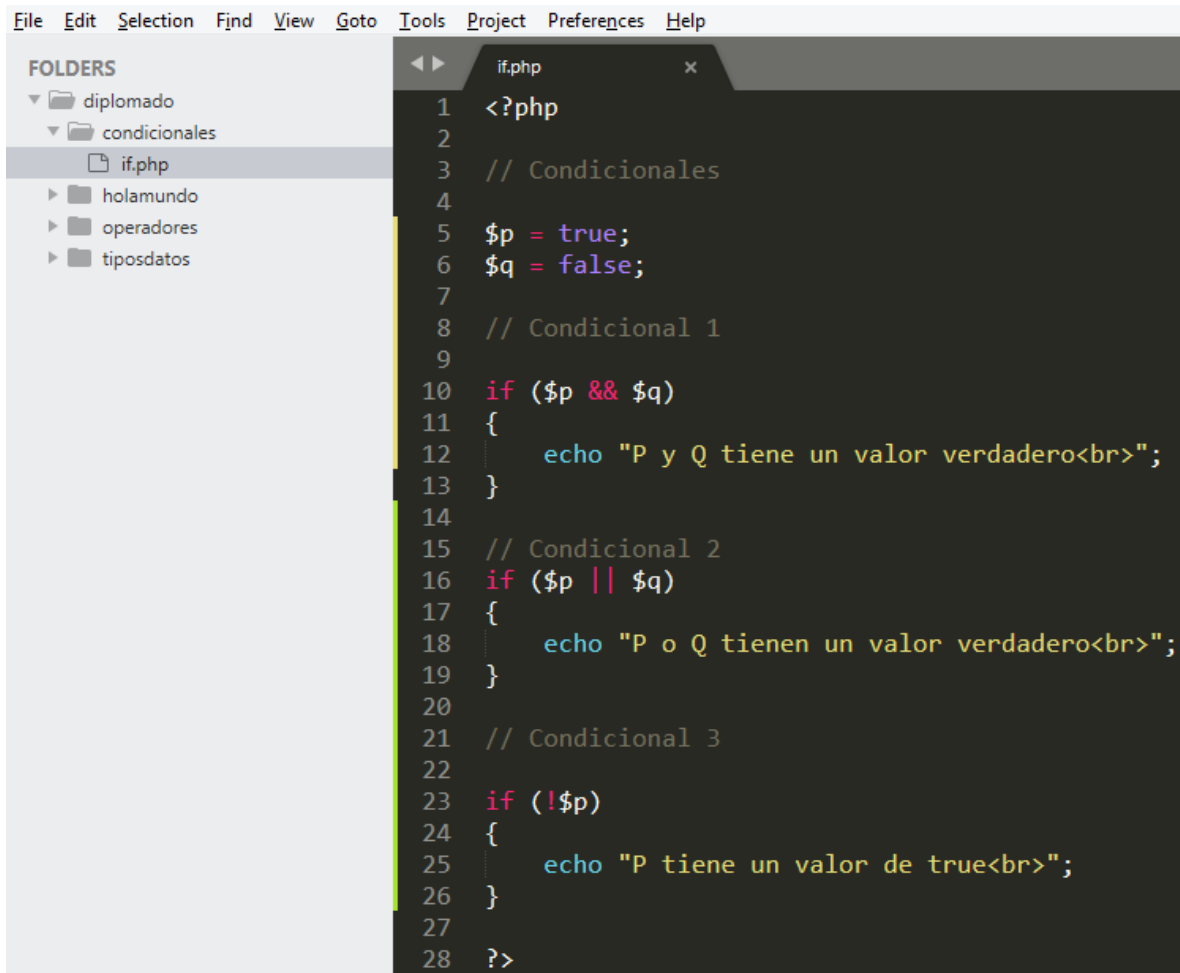
Condicional 1: P y Q tienen que ser true para cumplirse el if correctamente.

Condicional 2: P o Q tienen que ser true para cumplirse el if correctamente.

Condicional 3: P al ser true y agregar el operador de negación “!”, niega que el valor de P sea true y lo convierte a false (sólo en la ejecución de la condición, el valor de P sigue siendo true).

En la fila número 1 el resultado es el esperado, en la serie número 1 con el operador lógico “&&” al cumplirse que (1) y (2) son true, el resultado será verdadero. En la serie número 2, con el operador lógico “||” al cumplirse que (1) o (2) son al menos uno true, el resultado de la condicional será verdadero. En la serie número 3, con el operador de negación “!” al no cumplirse que (1) es true, por la negación, el resultado de la condicional será falso.

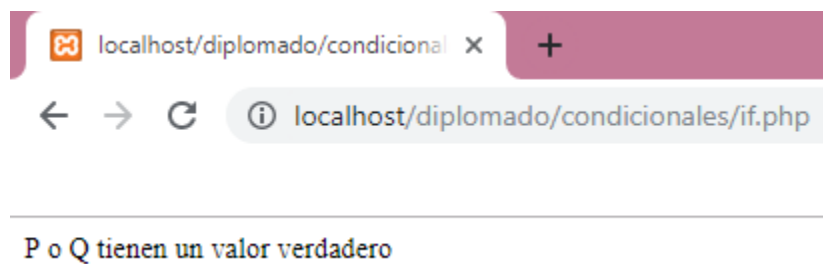
Evaluación de la segunda fila



```
File Edit Selection Find View Goto Tools Project Preferences Help

FOLDERS
└─ diplomado
  └─ condicionales
    └─ if.php
      └─ holamundo
      └─ operadores
      └─ tiposdatos

1 <?php
2
3 // Condicionales
4
5 $p = true;
6 $q = false;
7
8 // Condicional 1
9
10 if ($p && $q)
11 {
12     echo "P y Q tiene un valor verdadero<br>";
13 }
14
15 // Condicional 2
16 if ($p || $q)
17 {
18     echo "P o Q tienen un valor verdadero<br>";
19 }
20
21 // Condicional 3
22
23 if (!$p)
24 {
25     echo "P tiene un valor de true<br>";
26 }
27
28 ?>
```



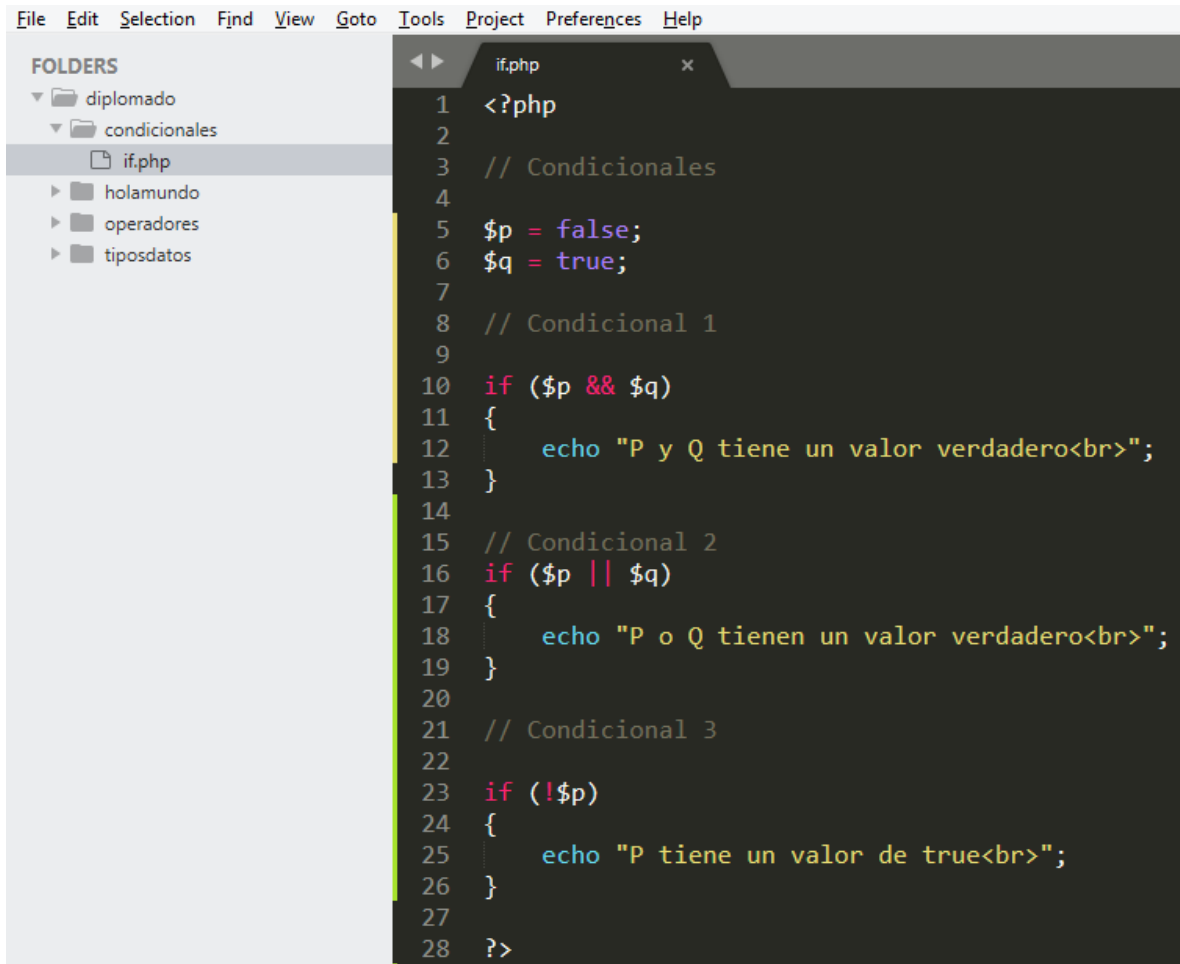
Condicional 1: P y Q tienen que ser true para cumplirse el if correctamente.

Condicional 2: P o Q tienen que ser true para cumplirse el if correctamente.

Condicional 3: P al ser true y agregar el operador de negación “!”, niega que el valor de P sea true y lo convierte a false (sólo en la ejecución de la condición, el valor de P sigue siendo true).

En la fila número 2 el resultado es el esperado, en la serie número 1 con el operador lógico “&&” al no cumplirse que (1) y (2) son true, el resultado de la condicional no será falso. En la serie número 2, con el operador lógica “||” al cumplirse que (1) o (2) son al menos uno true, el resultado de la condicional será verdadero. En la serie número 3, con el operador lógico “!” al no cumplirse que (1) es true, por la negación, el resultado de la condicional será falso.

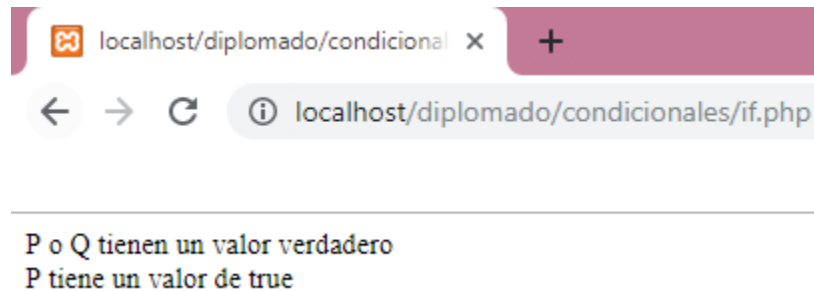
Evaluación de la tercera fila



```
File Edit Selection Find View Goto Tools Project Preferences Help

FOLDERS
└─ diplomado
  └─ condicionales
    └─ if.php
      └─ holamundo
      └─ operadores
      └─ tiposdatos

if.php
1 <?php
2
3 // Condicionales
4
5 $p = false;
6 $q = true;
7
8 // Condicional 1
9
10 if ($p && $q)
11 {
12     echo "P y Q tiene un valor verdadero<br>";
13 }
14
15 // Condicional 2
16 if ($p || $q)
17 {
18     echo "P o Q tienen un valor verdadero<br>";
19 }
20
21 // Condicional 3
22
23 if (!$p)
24 {
25     echo "P tiene un valor de true<br>";
26 }
27
28 ?>
```



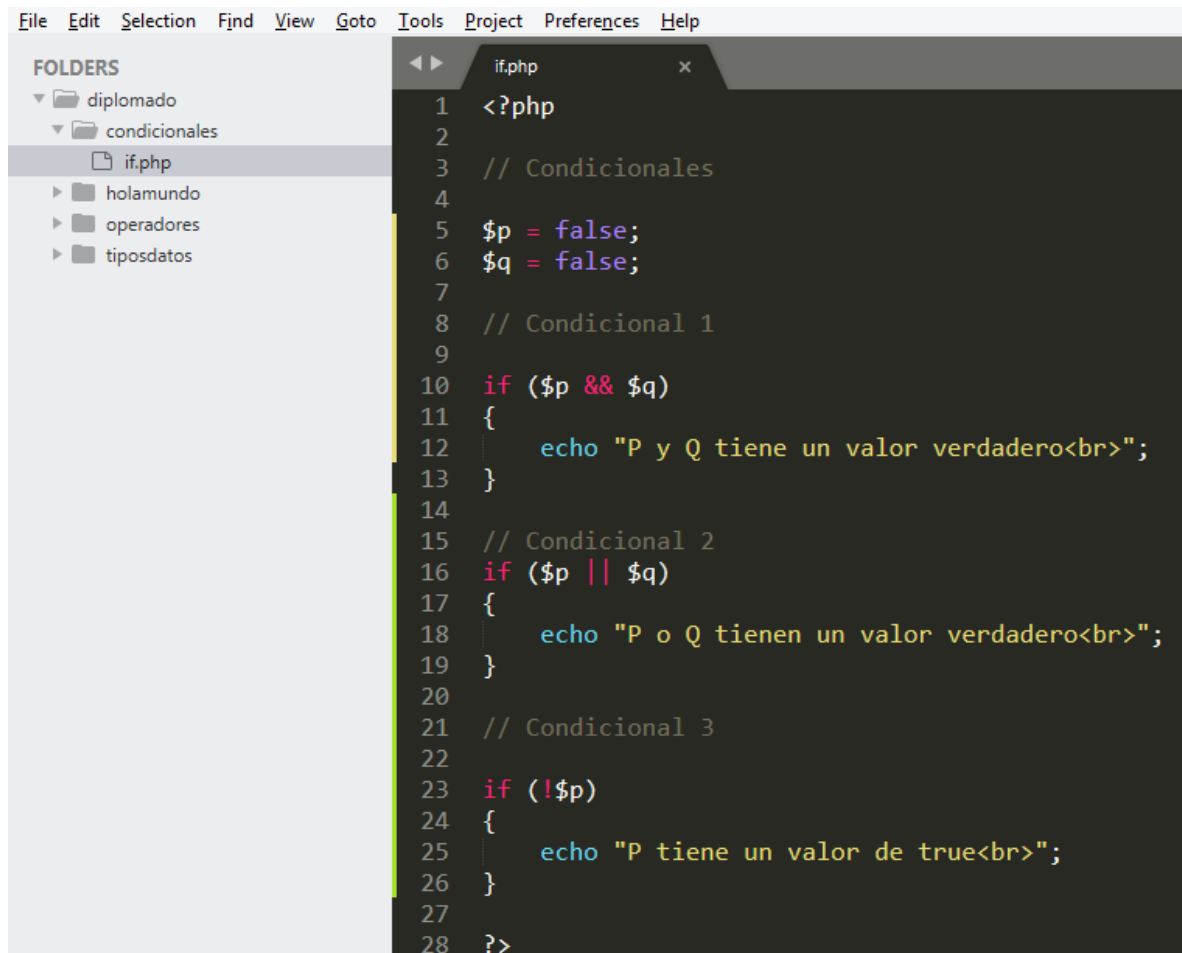
Condicional 1: P y Q tienen que ser true para cumplirse el if correctamente.

Condicional 2: P o Q tienen que ser true para cumplirse el if correctamente.

Condicional 3: P al ser true y agregar el operador de negación “!”, niega que el valor de P sea true y lo convierte a false (sólo en la ejecución de la condición, el valor de P sigue siendo true).

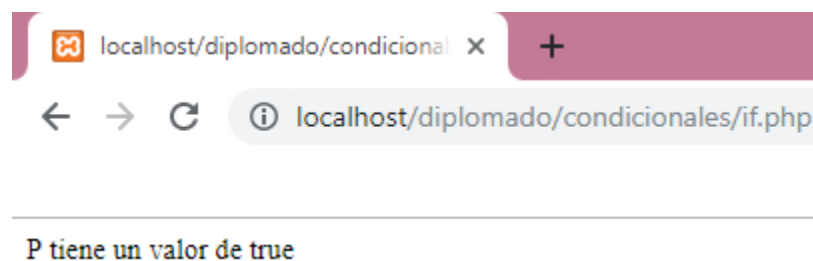
En la fila número 3 el resultado es el esperado, en la serie número 1 con el operador lógico “&&” al no cumplirse que (1) y (2) son true, el resultado de la condicional será falso. En la serie número 2, con el operador lógico “||” al cumplirse que (1) o (2) son al menos uno true, el resultado de la condicional será verdadero. En la serie número 3, con el operador lógico “!” al cumplirse que (1) es true, por la negación, el resultado de la condicional será verdadero.

Evaluación de la cuarta fila



```
File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
└─ diplomado
  └─ condicionales
    └─ if.php
      └─ holamundo
      └─ operadores
      └─ tiposdatos

1 <?php
2
3 // Condicionales
4
5 $p = false;
6 $q = false;
7
8 // Condicional 1
9
10 if ($p && $q)
11 {
12     echo "P y Q tiene un valor verdadero<br>";
13 }
14
15 // Condicional 2
16 if ($p || $q)
17 {
18     echo "P o Q tienen un valor verdadero<br>";
19 }
20
21 // Condicional 3
22
23 if (!$p)
24 {
25     echo "P tiene un valor de true<br>";
26 }
27
28 ?>
```



Condicional 1: P y Q tienen que ser true para cumplirse el if correctamente.

Condicional 2: P o Q tienen que ser true para cumplirse el if correctamente.

Condicional 3: P al ser true y agregar el operador de negación “!”, niega que el valor de P sea true y lo convierte a false (sólo en la ejecución de la condición, el valor de P sigue siendo true).

En la fila número 4, el resultado es el esperado, en la serie número 1 con el operador lógico “&&” al no cumplirse que (1) y (2) son true, el resultado de la condicional será falso. En la serie número 2, con el operador lógico “||” al no cumplirse que (1) o (2) son al menos uno true, el resultado de la condicional será falso. En la serie número 3, con el operador lógico “!” al cumplirse que (1) es true, por la negación, el resultado de la condicional será verdadero.

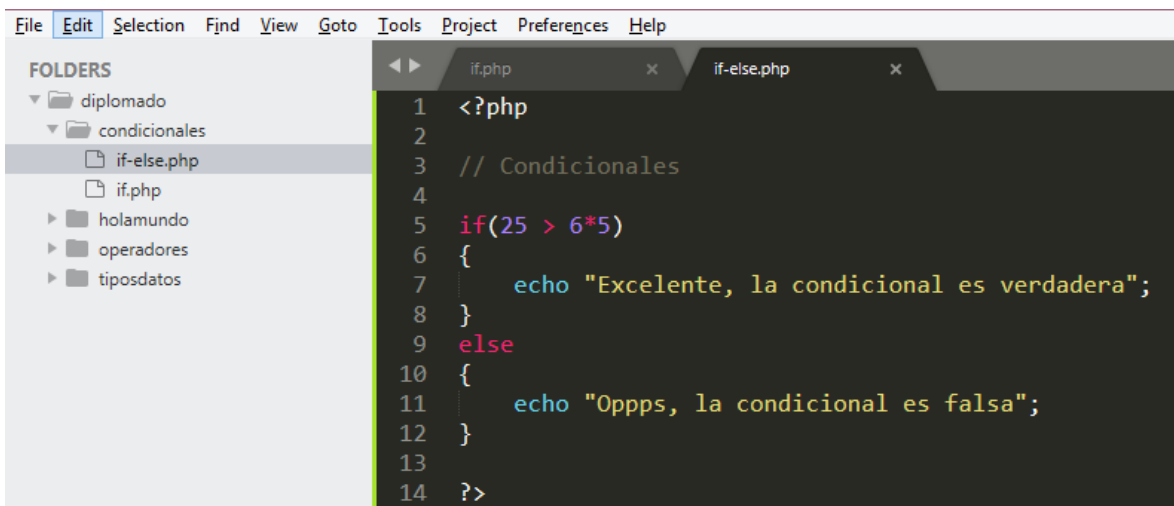
Sentencia if – else

Hasta el momento la estructura condicional if ha servido para ilustrar los casos en los que una sentencia encerrada en la condicional al momento de retornar un valor booleano true, permite ejecutar código a parte a raíz del mismo resultado, pero existen casos en qué se necesita un complemento al if, en casos de que no se cumpla la condicional, ¿qué hará el programa?, ahí entra en juego la estructura condicional else que permite dentro del código, dictaminar una secuencia de instrucciones en caso de que el if no retorne un valor true y encerrarlas dentro de “{ }” para su posterior ejecución.

La instrucción else ejecuta algún código si una condición es falsa.

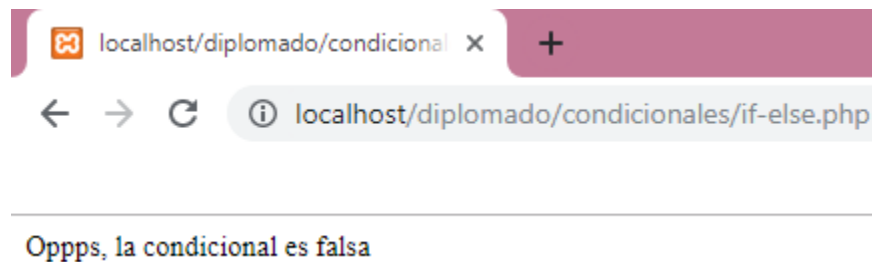
```
if (condition) {  
    code to be executed if condition is true;  
} else {  
    code to be executed if condition is false;  
}
```

Hay una característica con el else, y es que él, se ejecutará, sólo y sólo si el resultado de la condición es if arroja un resultado falso.

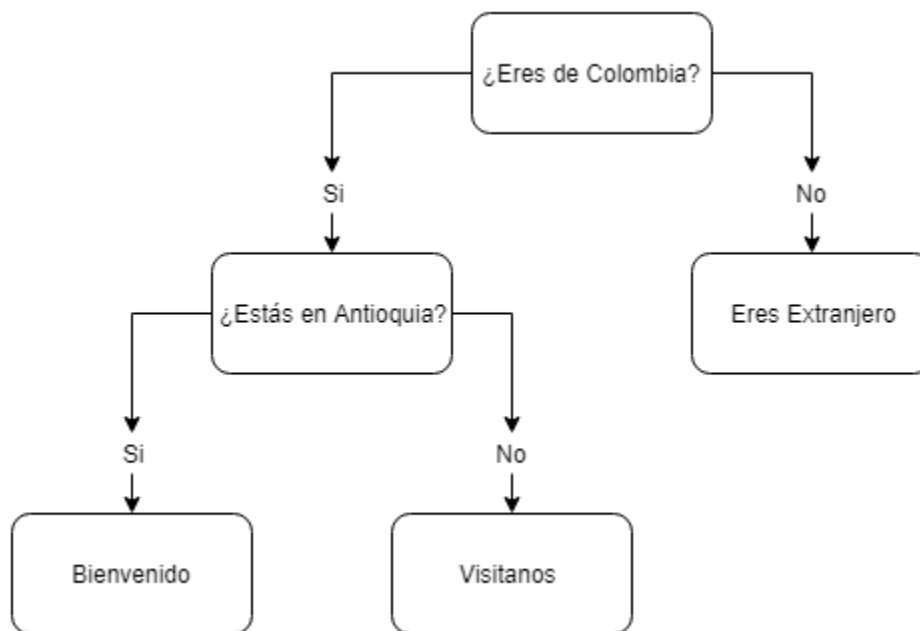


The screenshot shows a code editor with a menu bar (File, Edit, Selection, Find, View, Goto, Tools, Project, Preferences, Help) and a sidebar showing a file explorer. The main editor area displays the following PHP code:

```
1 <?php  
2  
3 // Condicionales  
4  
5 if(25 > 6*5)  
6 {  
7     echo "Excelente, la condicional es verdadera";  
8 }  
9 else  
10 {  
11     echo "Oppps, la condicional es falsa";  
12 }  
13  
14 ?>
```



El ejemplo más claro es el de ilustración del inicio de condicionales, donde se determinan una serie de casos a raíz de los resultados que la condicional va arrojando.



- Primero está la pregunta/condicional ¿Eres de Colombia?

Existen dos posibles resultados (1) sí – true y (2) no – false.

- Si el resultado es (1) es decir true, se entra de nuevo en una condicional con la siguiente pregunta/condicional.
- ¿Estás en Antioquia?

Existen nuevamente dos resultados (3) sí – true y (4) no – false.

- Si el resultado es (3) es decir true, se entra a ejecutar el mensaje “Bienvenido”.
- Si el resultado es (4) es decir false, se entra a ejecutar el mensaje “Visítanos”.
- Sí el resultado de ¿Eres de Colombia? Es (2) es decir false, se entra a ejecutar el mensaje “Eres Extranjero”.

Es simple de entender el funcionamiento del else que juega el papel de la negación, es decir el false y permite fraccionar el código para los casos particulares en que se necesita segmentar el código.

El ejemplo anterior aplicando su sintaxis en código sería de la siguiente forma.

```
if.php x if-else.php x
1 <?php
2
3 // Condicionales
4
5 $Pregunta1 = "Si";
6 $Pregunta2 = "No";
7
8 echo "¿Eres de Colombia? </br>";
9 echo "Respuesta: " . $Pregunta1 . "</br>";
10
11 if ($Pregunta1 == "Si")
12 {
13     echo "¿Eres de Antioquia?</br>";
14     echo "Respuesta: " . $Pregunta2 . "</br>";
15
16     if($Pregunta2 == "Si")
17     {
18         echo "Bienvenido</br>";
19     }
20     else
21     {
22         echo "Vísínatos</br>";
23     }
24 }
25 else
26 {
27     echo "Eres Extranjero</br>";
28 }
29
30 ?>
```

localhost/diplomado/condicional x +

localhost/diplomado/condicionales/if-else.php

¿Eres de Colombia?
Respuesta: Si
¿Eres de Antioquia?
Respuesta: No
Vísínatos

En el ejemplo anterior se observa una particularidad muy especial, el hecho que los bloques de códigos dentro de las sentencias `if` y `else`, en sus respectivas llaves “{}”, permiten la inserción de nuevas condicionales.

Sentencia `else if`

En base a las dos sentencias condicionales vistas, existe una derivación de ambas, que su utilidad se ve en el momento en el que las condicionales estructuradas como **`if`** y **`else`** ocupan una validación adicional, véase de la siguiente forma “sí no se cumple la condición, haga lo siguiente siempre y cuando se cumpla una nueva condición”.

```
if (condition) {  
    code to be executed if this condition is true;  
} elseif (condition) {  
    code to be executed if first condition is false and this condition is true;  
} else {  
    code to be executed if all conditions are false;  
}
```

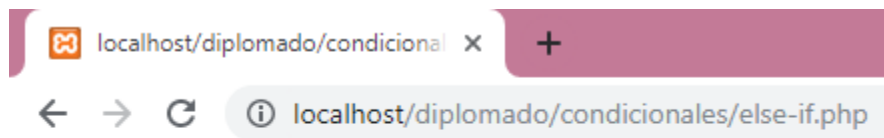
La estructura `else if` es de gran importancia cuando se ocupa para revalidar una condición, como la función de un cortafuego, que puede ir de sentencia en sentencia evaluándose hasta encontrar el retorno correcto. Para ejemplificar de mejor forma lo anterior, analícese el siguiente ejercicio:

En Colombia hay medidas que regulan la velocidad de los automoviles según las zonas donde se encuentre:

- 30KM/H – Zonas Escolares
- 60KM/H – Vías Urbanas
- 80KM/H – Vías Rurales
- 100KM/H – Rutas Nacionales

Realice un algoritmo que permita determinar según una velocidad X, a qué grupo de límites pertenezco.

```
2
3 // Condicionales
4
5 $velocidad = 89;
6
7 if($velocidad >= 0 and $velocidad <= 30)
8 {
9     echo "Zonas escolares";
10 }
11 else if($velocidad > 30 and $velocidad <= 60)
12 {
13     echo "Vías urbanas";
14 }
15 else if($velocidad > 60 and $velocidad <= 80)
16 {
17     echo "Vías rurales";
18 }
19 else if($velocidad > 80 and $velocidad <= 100)
20 {
21     echo "Rutas nacionales";
22 }
23 else
24 {
25     echo "Estás infringiendo los límites de velocidad";
26 }
27
28 ?>
```

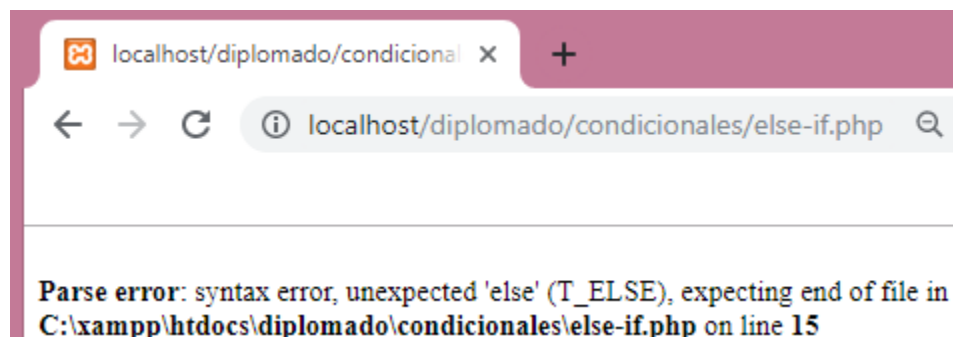



Rutas nacionales

En el ejercicio anterior se observa la aplicación donde else if cumple la función de evaluar otra sentencia previa a la ejecución del else, en caso de no estar este else if, la condicional se quedaría estancada en únicamente 2 posibles resultados (no exactos para el caso)

No se saldría de la sentencia número 1 o el resultado equívoco de la misma. Además, el uso de else's repetidamente es erróneo en el lenguaje por la sintaxis, dado que no puede contener dos o más posibles resultados para un solo caso erróneo:

```
2
3 // Condicionales
4
5 $velocidad = 89;
6
7 if($velocidad >= 0 and $velocidad <= 30)
8 {
9     echo "Zonas escolares";
10 }
11 else
12 {
13     echo "Vías urbanas";
14 }
15 else
16 {
17     echo "Vías rurales";
18 }
19 else
20 {
21     echo "Rutas nacionales";
22 }
23 else
24 {
25     echo "Estás infringiendo los límites de velocidad";
26 }
27
28 ?>
```



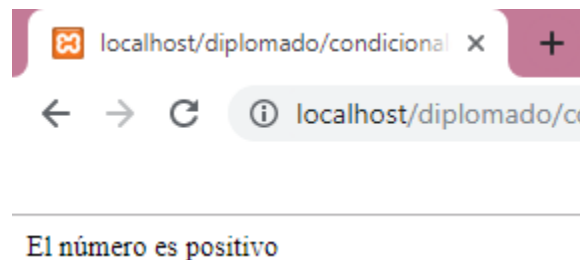
localhost/diplomado/condicional x +

← → ↻ ⓘ localhost/diplomado/condicionales/else-if.php 🔍

Parse error: syntax error, unexpected 'else' (T_ELSE), expecting end of file in C:\xampp\htdocs\diplomado\condicionales\else-if.php on line 15

Hay casos en que no aplica necesariamente el uso del else if, y son aquellos donde sólo ocupamos dos resultados un sí o un no únicamente:

```
2
3 // Condicionales
4
5 $numero = 8;
6
7 if($numero >= 0)
8 {
9     echo "El número es positivo";
10 }
11 else
12 {
13     echo "El número es negativo";
14 }
15
16 ?>
```



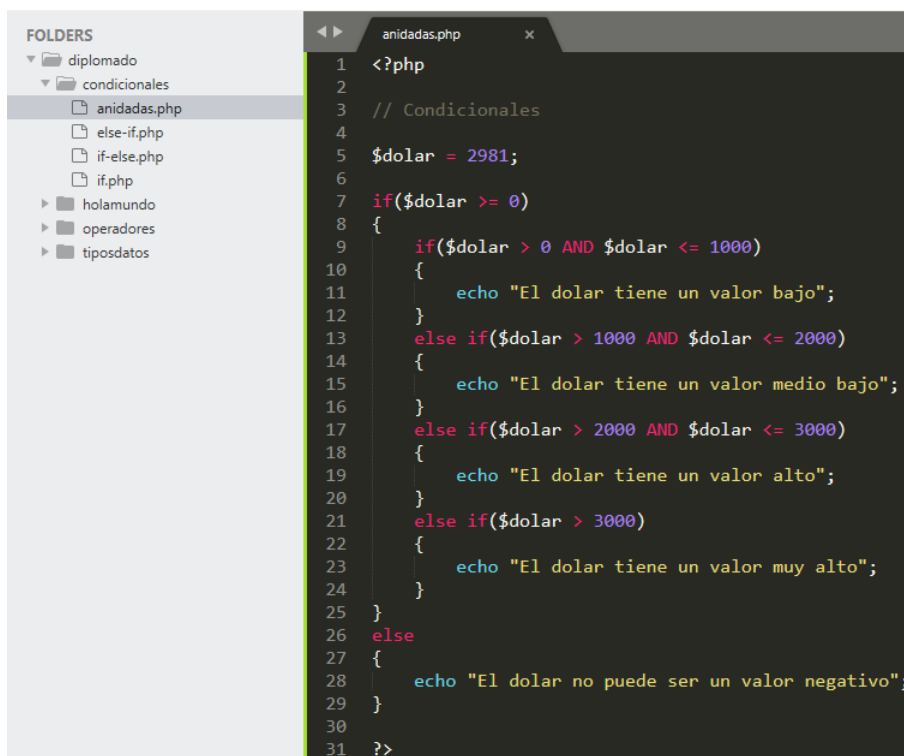
Por definición se sabe que sí el número no es mayor o igual a 0, va a ser negativo. Aunque el funcionamiento no se vería afectado en caso de agregarse un else if. Véase:

```
else-if.php x
2
3 // Condicionales
4
5 $numero = 8;
6
7 if($numero >= 0)
8 {
9     echo "El número es positivo";
10 }
11 else if($numero < 0)
12 {
13     echo "El número es negativo";
14 }
15
16 ?>
```

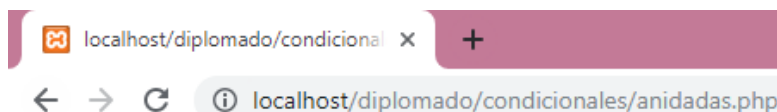
La segunda sentencia ($\$numero < 0$) puede sobrar en caso de que se quiera, en este ejemplo es como un segundo seguro de que se está validando que efectivamente el número sea menor que 0 (negativo) pese a que el else lo hace implícitamente.

Estructuras condicionales anidadas

Un paso más allá de las estructuras pasadas, están las condicionales anidadas que permiten ejecutar una sentencia a partir de una condición en forma de ciclo hasta encontrar el retorno esperado (Cumplimiento de la condición), si ésta condición se cumple, entonces se ejecuta la sentencia en el programa, en caso de no cumplirse dicha condición, se puede hacer otra condición en el programa para que se cicle, marque error y vuelva a solicitar la información hasta que se cumpla la condición, de no validarse la condición al igual que en los casos anteriores, se puede definir un caso base donde retornará un valor que se espera.



```
1 <?php
2
3 // Condicionales
4
5 $dolar = 2981;
6
7 if($dolar >= 0)
8 {
9     if($dolar > 0 AND $dolar <= 1000)
10     {
11         echo "El dolar tiene un valor bajo";
12     }
13     else if($dolar > 1000 AND $dolar <= 2000)
14     {
15         echo "El dolar tiene un valor medio bajo";
16     }
17     else if($dolar > 2000 AND $dolar <= 3000)
18     {
19         echo "El dolar tiene un valor alto";
20     }
21     else if($dolar > 3000)
22     {
23         echo "El dolar tiene un valor muy alto";
24     }
25 }
26 else
27 {
28     echo "El dolar no puede ser un valor negativo";
29 }
30
31 ?>
```



El dolar tiene un valor alto

A diferencia de las estructuras pasadas, las condicionales anidadas proveen un abanico mucho más amplio de posibilidades y combinaciones booleanas para determinar más casos y resultados, hay varios aspectos a recordar cuando se utilicen condicionales anidadas.

- Utilizar únicamente las condicionales necesarias.
- Evitar redundancias en las sentencias.
- Siempre se debe tratar de tener un retorno en caso de que ninguna condicional se aplique.
- Tener muy presente el uso correcto de las llaves y estar dentro de las condicionales correctas debido al volumen de las mismas.

Recursos disponibles para el aprendizaje



Las condicionales son un concepto muy importante en la programación con PHP, en la documentación del lenguaje hay información muy importante y que te puede ser útil, visítala, Disponible en: <https://www.php.net/manual/es/control-structures.if.php>

Ejercicio

¿Deseas profundizar en la temática de las Condicionales? Entonces te sugiero realizar los siguientes ejercicios que pondrán a prueba los conocimientos adquiridos.
(MÓDULO 2 – EJERCICIOS DE CONDICIONALES)
¡Inténtalo! 👍



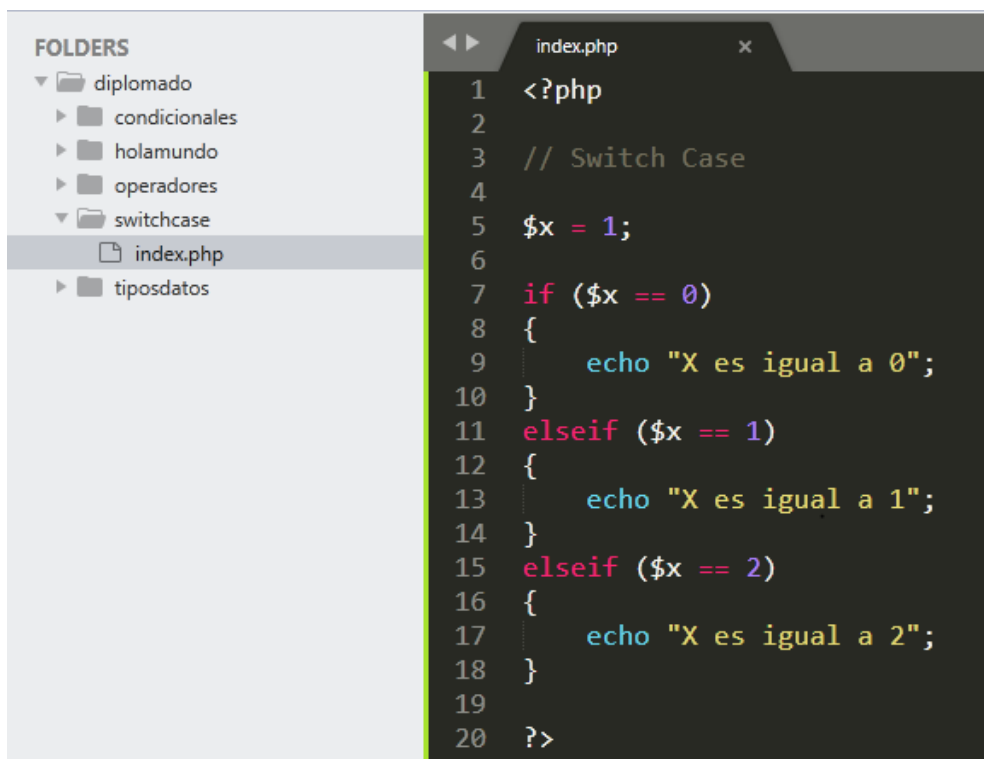
TEMA 3

Switch Case

La sentencia switch case se utiliza para realizar diferentes acciones en función de diferentes condiciones.

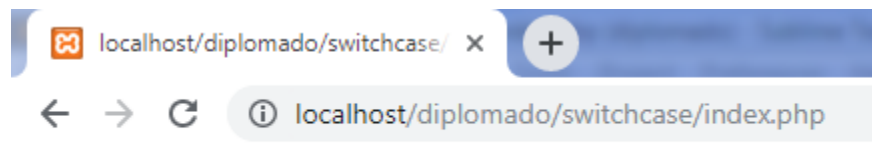
La sentencia switch es similar a una serie de sentencias **IF** en la misma expresión. En muchas ocasiones, es posible que se quiera comparar la misma variable (o expresión) con muchos valores diferentes, y ejecutar una parte de código distinta dependiendo de a qué valor es igual. Para esto es exactamente la expresión switch. (php, 2019).

Los dos ejemplos siguientes son dos formas diferentes de escribir lo mismo, uno con una serie de sentencias if y elseif, y el otro usando la sentencia switch case:

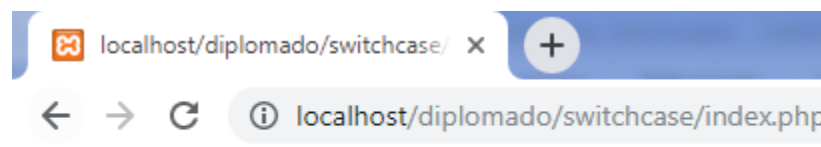
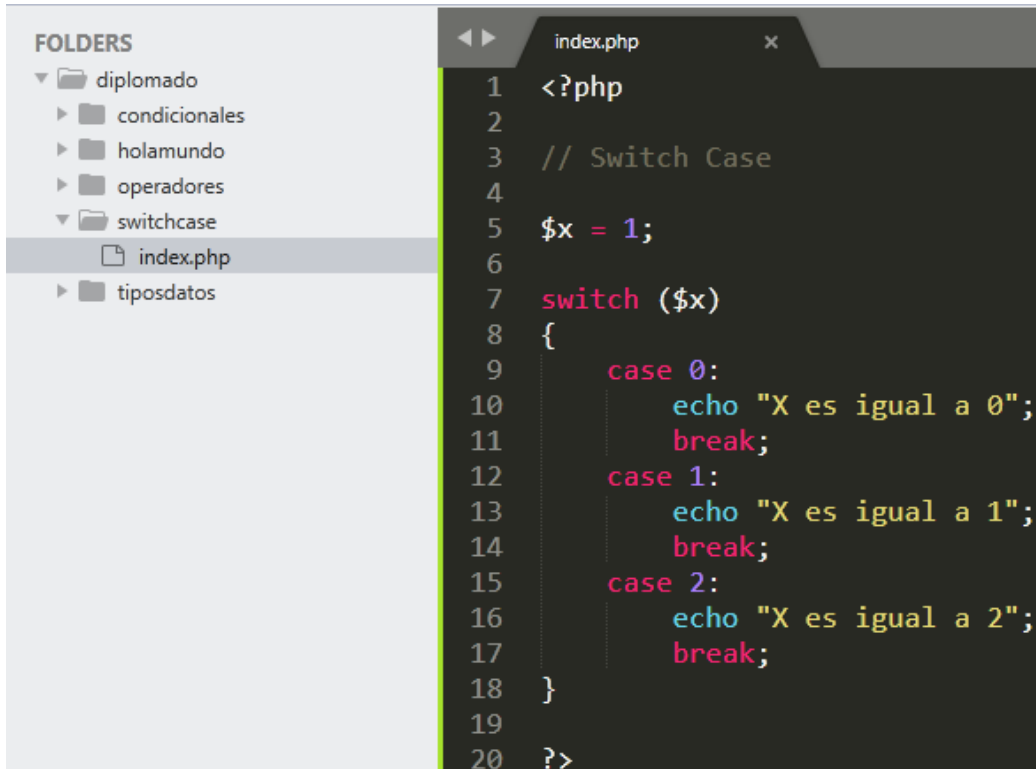


The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a folder structure: 'diplomado' containing 'condicionales', 'holamundo', 'operadores', 'switchcase', and 'tiposdatos'. The 'switchcase' folder is expanded, showing 'index.php'. The code editor shows the following PHP code:

```
1 <?php
2
3 // Switch Case
4
5 $x = 1;
6
7 if ($x == 0)
8 {
9     echo "X es igual a 0";
10 }
11 elseif ($x == 1)
12 {
13     echo "X es igual a 1";
14 }
15 elseif ($x == 2)
16 {
17     echo "X es igual a 2";
18 }
19
20 ?>
```



X es igual a 1



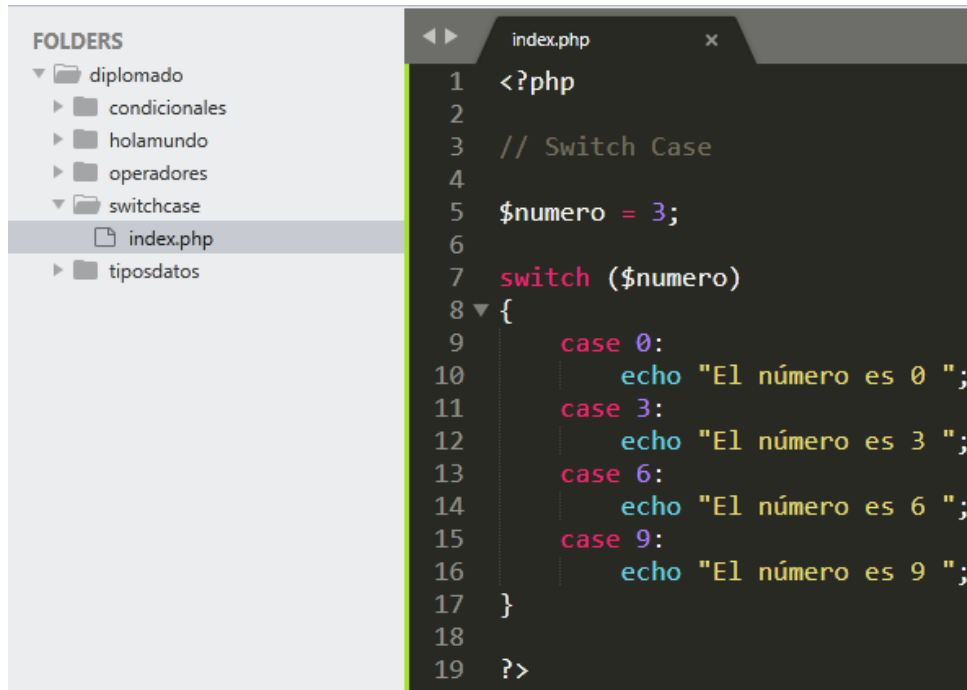
X es igual a 1

En el ejemplo anterior, se da el mismo resultado a partir de dos métodos de condición diferentes, switch case y la estructura if y else if.

Switch case por medio de sus **case** (El equivalente a las sentencias if o else if) realiza las comparaciones a partir de la variable a evaluar **switch (\$x)** hasta encontrar la condición / case que satisface el resultado esperado.

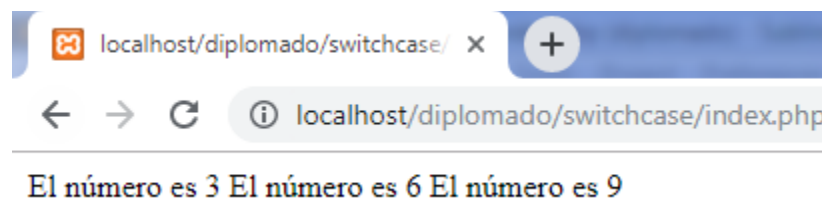
En código de cada **case** representa el delimitador de las llaves **{}** hasta encontrar un **break** que indica la terminación de ejecución del bloque de código.

Es importante entender cómo la sentencia **switch** es ejecutada con el fin de evitar errores. La sentencia **switch** ejecuta línea por línea (en realidad, sentencia por sentencia). Al principio, ningún código es ejecutado. Solo cuando se encuentra una sentencia **case** cuya expresión se evalúa a un valor que coincida con el valor de la expresión **switch**, PHP comienza a ejecutar la sentencias. PHP continúa ejecutando las sentencias hasta el final del bloque **switch**, o hasta la primera vez que vea una sentencia **break**. Si no se escribe una sentencia **break** al final de la lista de sentencias de un **case**, PHP seguirá ejecutando las sentencias del **caso** siguiente. Por ejemplo:

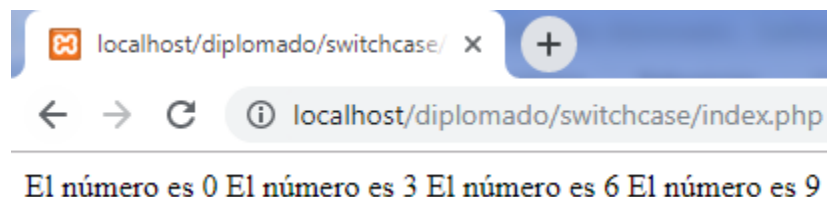


```
1 <?php
2
3 // Switch Case
4
5 $numero = 3;
6
7 switch ($numero)
8 {
9     case 0:
10         echo "El número es 0 ";
11     case 3:
12         echo "El número es 3 ";
13     case 6:
14         echo "El número es 6 ";
15     case 9:
16         echo "El número es 9 ";
17 }
18
19 ?>
```

Aquí, si \$numero es igual a 3 PHP ejecutaría todas las sentencias echo a partir del caso 2 (**case 3**) es decir, mostrar los 3 últimos echos donde se evalúa 3 – 6 y 9.



Si \$numero es igual a 0, PHP ejecutará todas las sentencias.

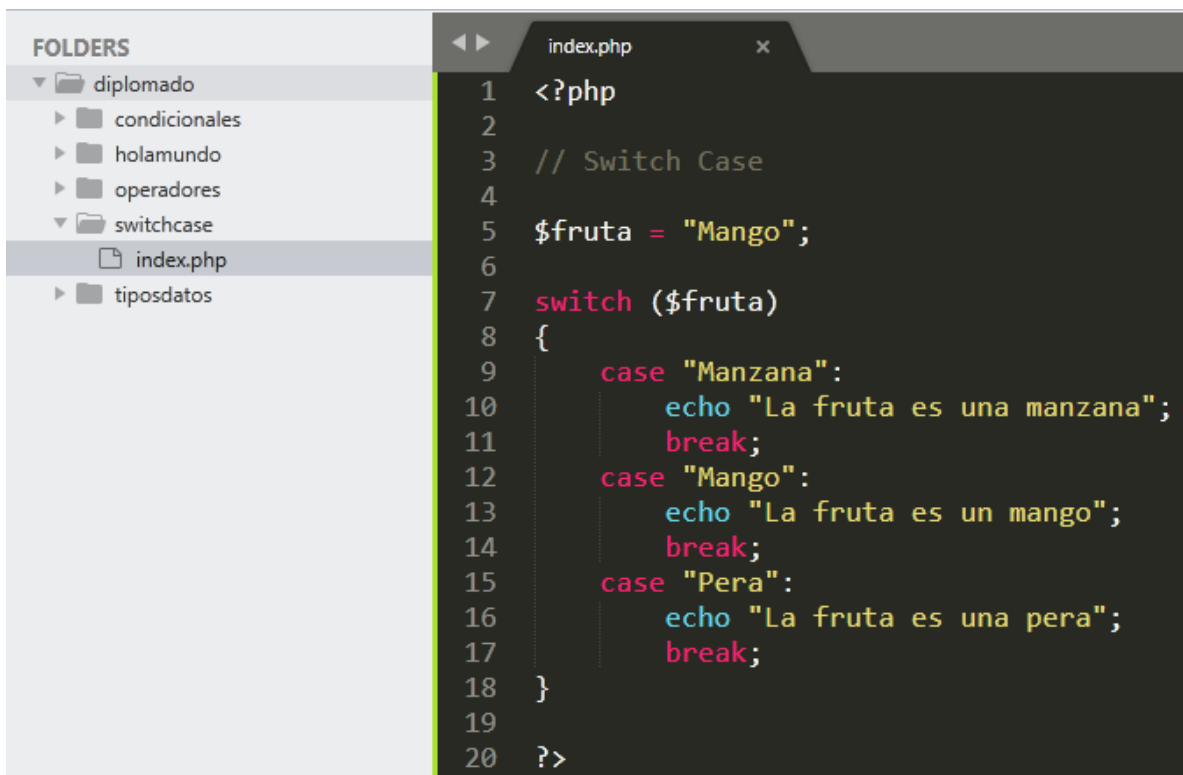


Por lo tanto, es importante no olvidar las sentencias **break** (aunque es posible que se desee evitar proporcionarlas a propósito bajo determinadas circunstancias).

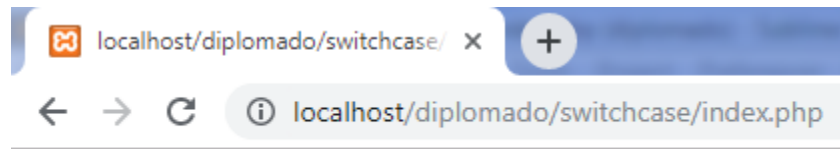
En una sentencia **switch**, la condición es evaluada sólo una vez y el resultado es comparado con cada una de las sentencias **case**. En una sentencia **else if**, la condición es evaluada otra vez. Si la condición es más complicada que una simple comparación y/o está en un ciclo (próximo tema), un **switch** puede ser más rápido.

Hay varias características a tener en cuenta dentro de la sentencia **switch** en PHP entre las que destacan las siguientes:

- La estructura **switch case** permite el uso de strings en sus **case**.

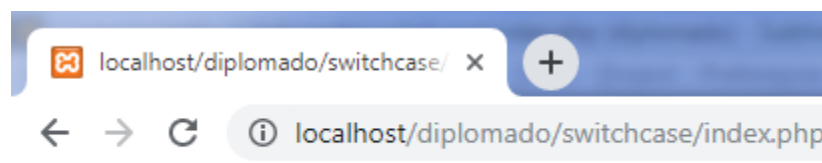
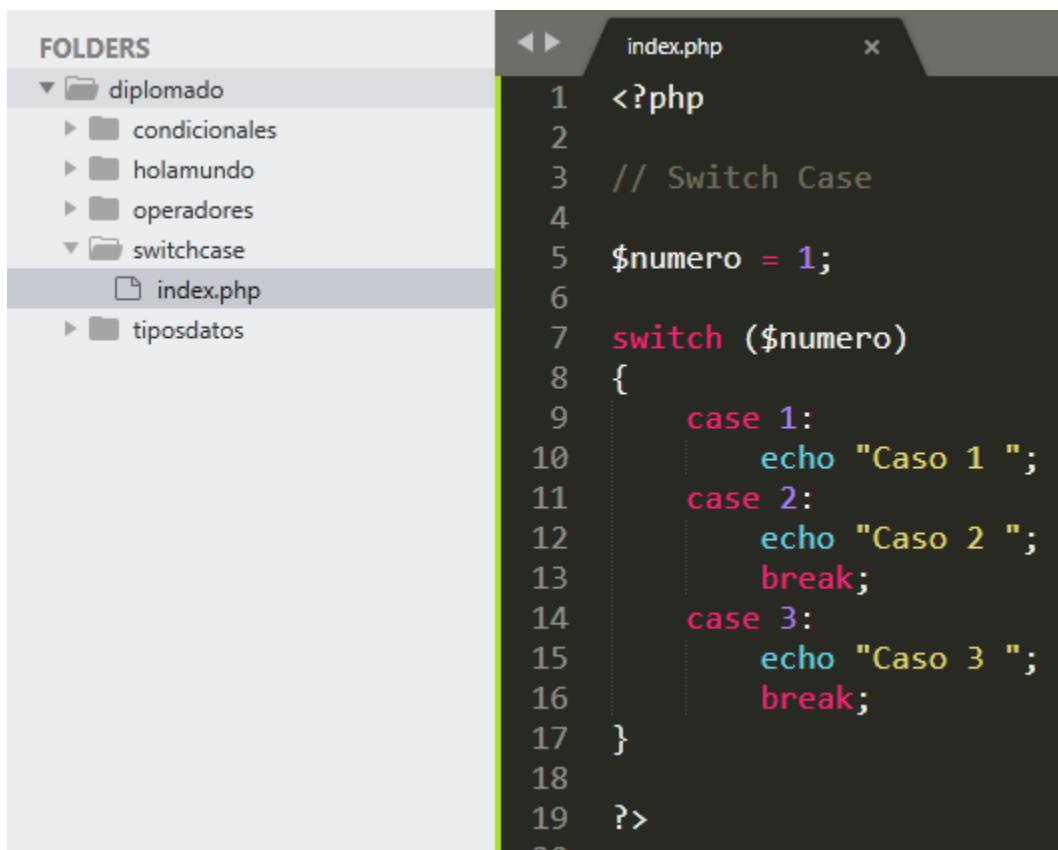


```
1 <?php
2
3 // Switch Case
4
5 $fruta = "Mango";
6
7 switch ($fruta)
8 {
9     case "Manzana":
10         echo "La fruta es una manzana";
11         break;
12     case "Mango":
13         echo "La fruta es un mango";
14         break;
15     case "Pera":
16         echo "La fruta es una pera";
17         break;
18 }
19
20 ?>
```

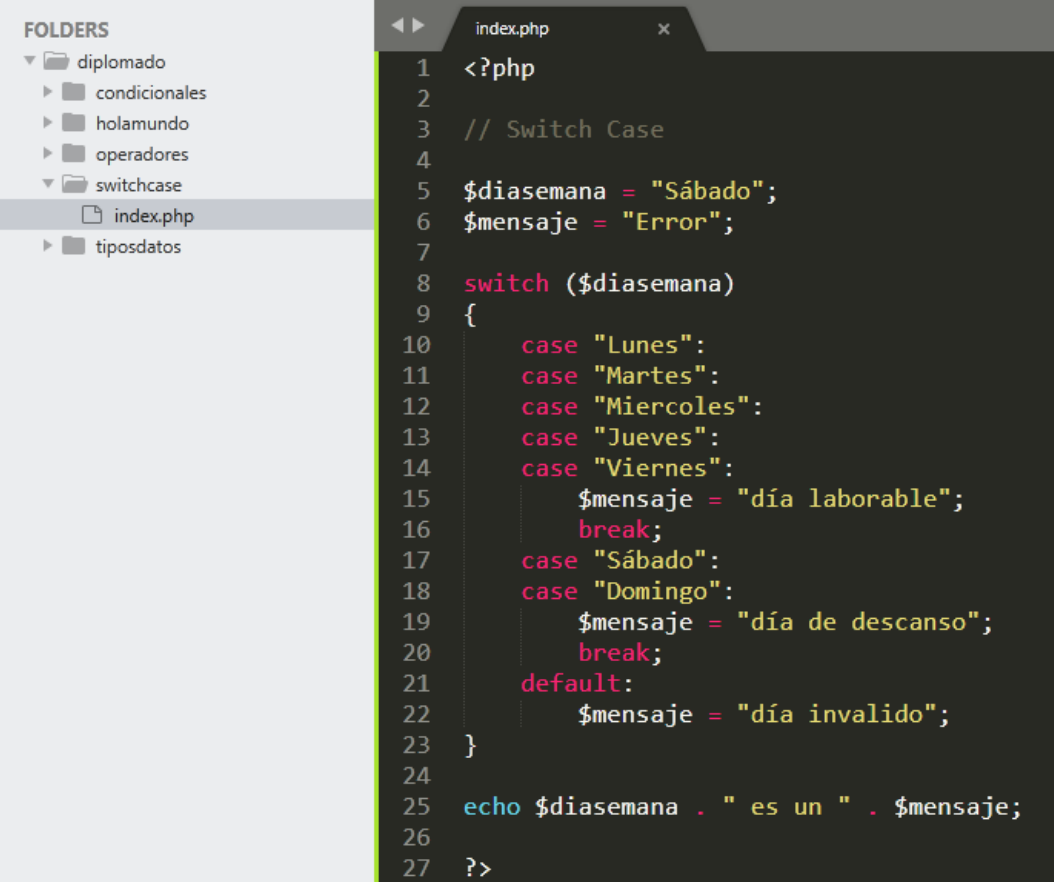


La fruta es un mango

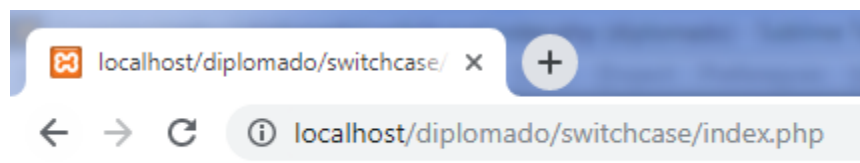
- La declaración `break` es opcional. Si se omite, la ejecución continuará en el siguiente caso.



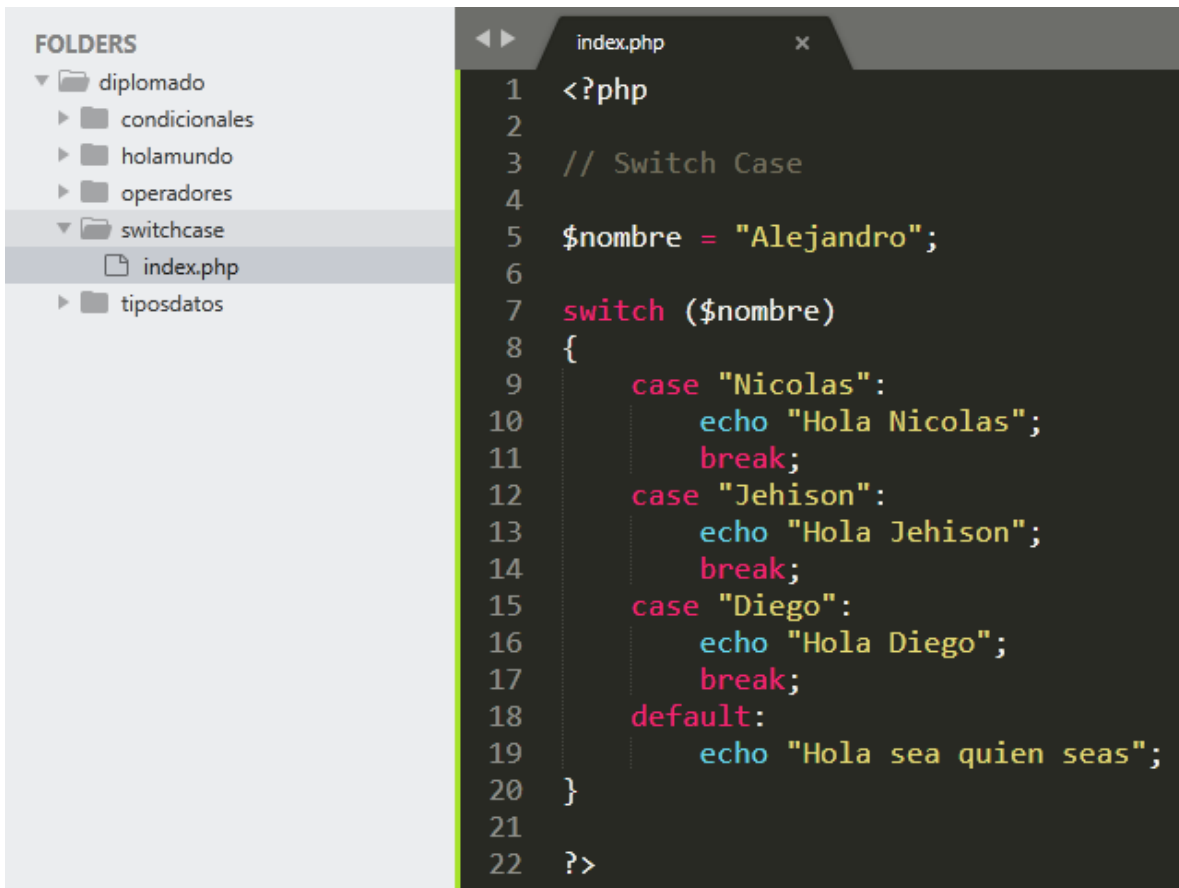
Caso 1 Caso 2



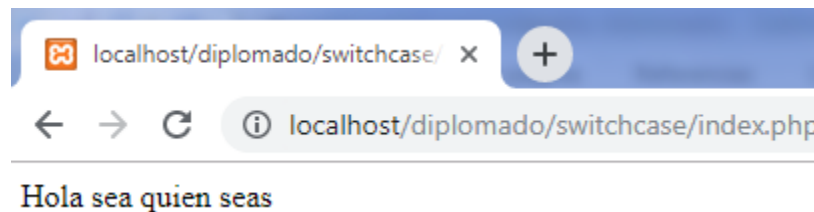
```
1 <?php
2
3 // Switch Case
4
5 $diasemana = "Sábado";
6 $mensaje = "Error";
7
8 switch ($diasemana)
9 {
10     case "Lunes":
11     case "Martes":
12     case "Miercoles":
13     case "Jueves":
14     case "Viernes":
15         $mensaje = "día laborable";
16         break;
17     case "Sábado":
18     case "Domingo":
19         $mensaje = "día de descanso";
20         break;
21     default:
22         $mensaje = "día invalido";
23 }
24
25 echo $diasemana . " es un " . $mensaje;
26
27 ?>
```



- Existe un caso especial y este es el default. El default aplica en el caso donde ningún case satisface la sentencia switch y se ejecuta por defecto (default).

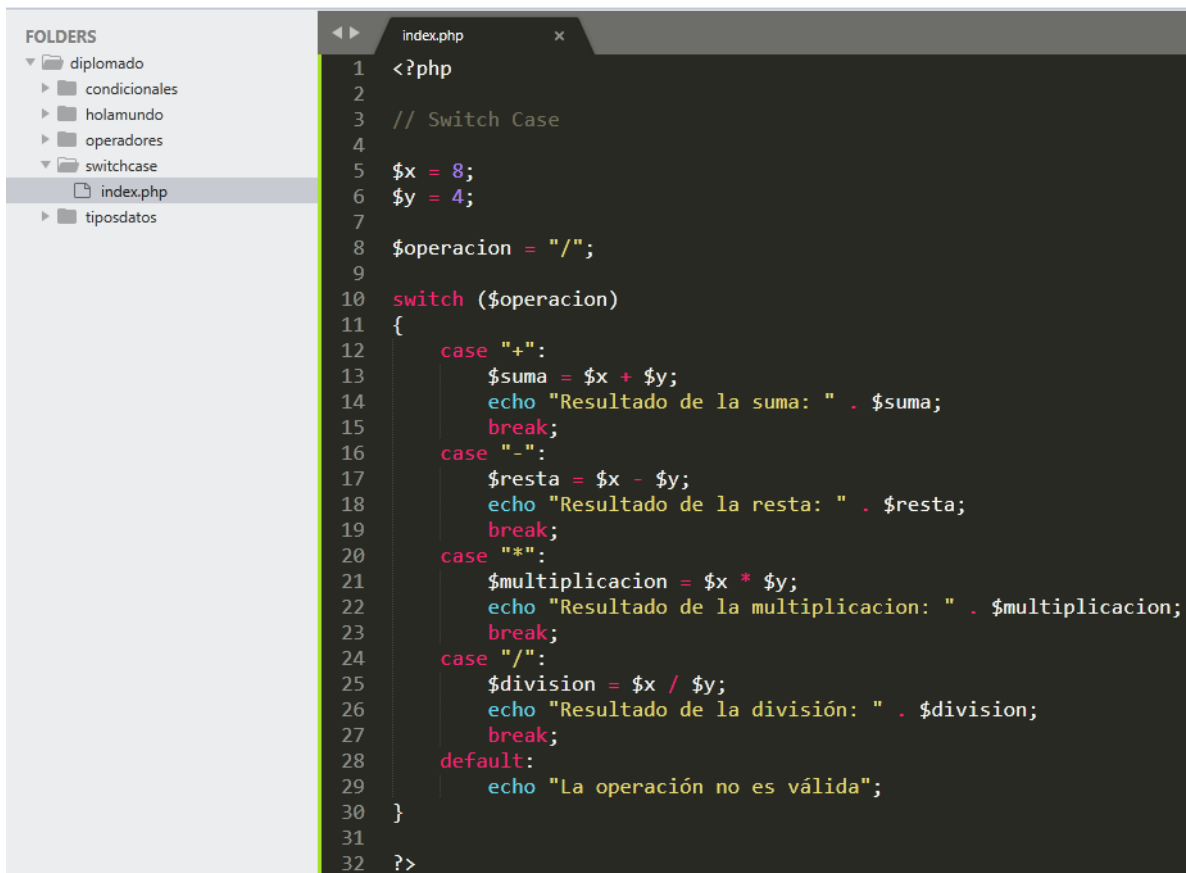


```
1 <?php
2
3 // Switch Case
4
5 $nombre = "Alejandro";
6
7 switch ($nombre)
8 {
9     case "Nicolas":
10         echo "Hola Nicolas";
11         break;
12     case "Jehison":
13         echo "Hola Jehison";
14         break;
15     case "Diego":
16         echo "Hola Diego";
17         break;
18     default:
19         echo "Hola sea quien seas";
20 }
21
22 ?>
```

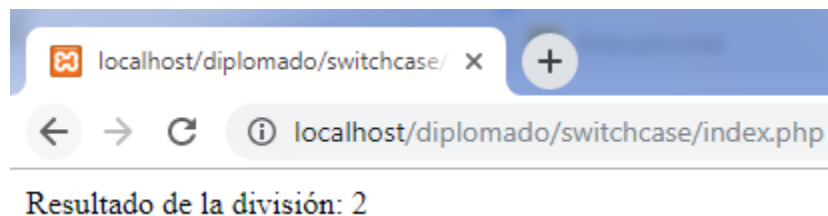


La instrucción default es opcional, y debe aparecer al final del switch como último caso de cumpliendo con similitud a la función del else.

- Se puede realizar cualquier tipo de operación o bloque de código dentro de la declaración del case.

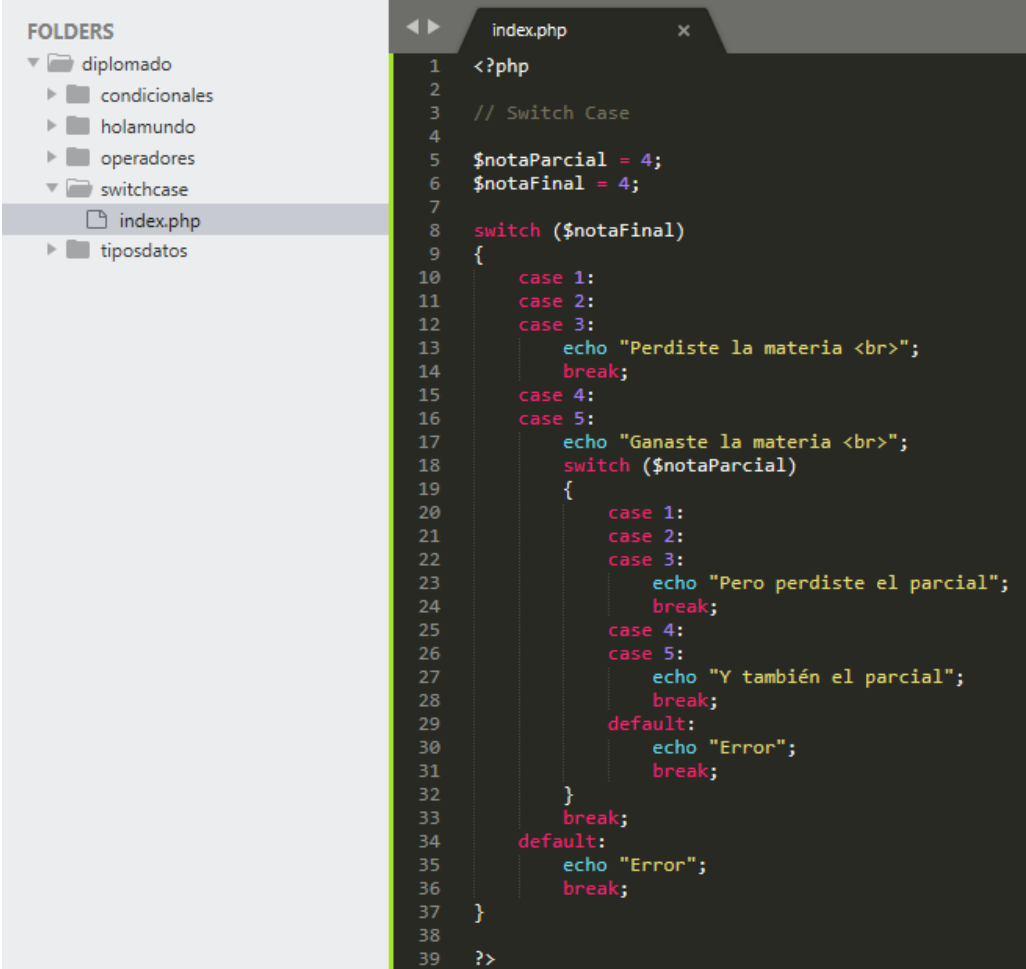


```
1 <?php
2
3 // Switch Case
4
5 $x = 8;
6 $y = 4;
7
8 $operacion = "/";
9
10 switch ($operacion)
11 {
12     case "+":
13         $suma = $x + $y;
14         echo "Resultado de la suma: " . $suma;
15         break;
16     case "-":
17         $resta = $x - $y;
18         echo "Resultado de la resta: " . $resta;
19         break;
20     case "*":
21         $multiplicacion = $x * $y;
22         echo "Resultado de la multiplicacion: " . $multiplicacion;
23         break;
24     case "/":
25         $division = $x / $y;
26         echo "Resultado de la división: " . $division;
27         break;
28     default:
29         echo "La operación no es válida";
30 }
31
32 ?>
```

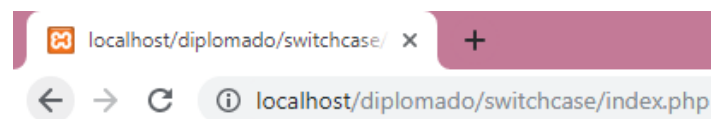


Switch Case Anidados

Se puede usar un Switch como parte de la secuencia de la declaración de un Switch externo. Esto se llama un Switch anidado.



```
1 <?php
2
3 // Switch Case
4
5 $notaParcial = 4;
6 $notaFinal = 4;
7
8 switch ($notaFinal)
9 {
10     case 1:
11     case 2:
12     case 3:
13         echo "Perdiste la materia <br>";
14         break;
15     case 4:
16     case 5:
17         echo "Ganaste la materia <br>";
18         switch ($notaParcial)
19         {
20             case 1:
21             case 2:
22             case 3:
23                 echo "Pero perdiste el parcial";
24                 break;
25             case 4:
26             case 5:
27                 echo "Y también el parcial";
28                 break;
29             default:
30                 echo "Error";
31                 break;
32         }
33         break;
34     default:
35         echo "Error";
36         break;
37 }
38
39 ?>
```



Ganaste la materia
Y también el parcial

Recursos disponibles para el aprendizaje



La estructura condicional Switch Case es un concepto muy importante y en especial en la programación con PHP, en la documentación del lenguaje hay información muy importante y que te puede ser útil, visítala, Disponible en: <https://www.php.net/manual/en/control-structures.switch.php>

Ejercicio

¿Deseas profundizar en la temática de switch case? Entonces te sugiero realizar los siguientes ejercicios que pondrán a prueba los conocimientos adquiridos.
(MÓDULO 2 – EJERCICIOS DE SWITCH CASE)
¡Inténtalo! 👍



TEMA 3

Ciclos

Los ciclos son una estructura de control de total importancia para el desarrollo en PHP. PHP y prácticamente todos los lenguajes más populares de la actualidad permiten hacer uso de estas estructuras.

Un ciclo en PHP permite repetir una o varias instrucciones cuantas veces sea necesario, por ejemplo, si se desean escribir los números del uno al diez no tendría sentido escribir cien líneas de código mostrando un número en cada una de estas, para eso y para varias acciones más es útil un ciclo. Un ciclo ayuda a llevar a cabo una tarea repetitiva en una cantidad de líneas muy pequeña y de forma prácticamente automática. Existen diferentes tipos de ciclos o bucles en PHP, cada uno tiene una utilidad para casos específicos. Se tiene en PHP las siguientes estructuras:

- **Ciclos For**

```
for (init counter; test counter; increment counter) {  
    code to be executed;  
}
```

- **Ciclos While**

```
while (condition is true) {  
    code to be executed;  
}
```

- **Ciclos do While**

```
do {  
    code to be executed;  
} while (condition is true);
```

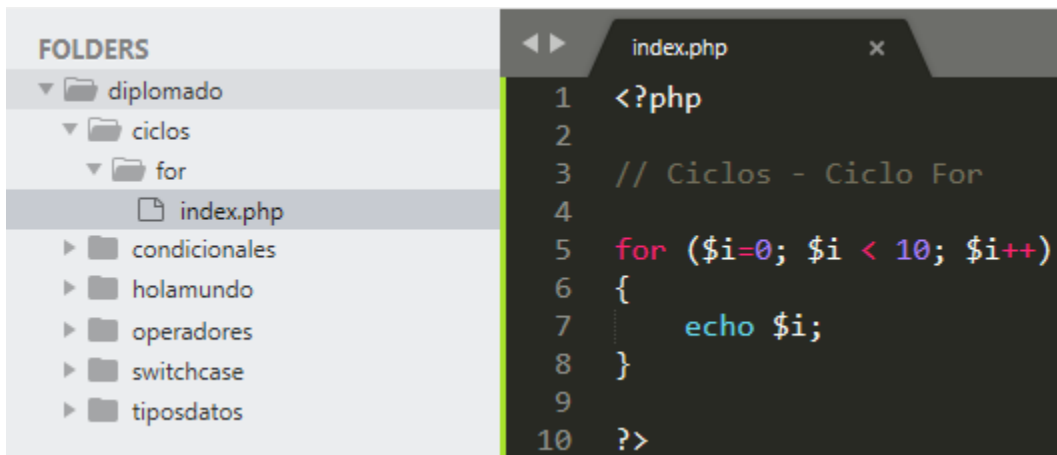
Ciclo For

Los ciclos For (o ciclos para) son una estructura de control cíclica, que permite ejecutar una o varias líneas de código de forma iterativa (o repetitiva), pero teniendo control y conocimiento sobre las iteraciones.

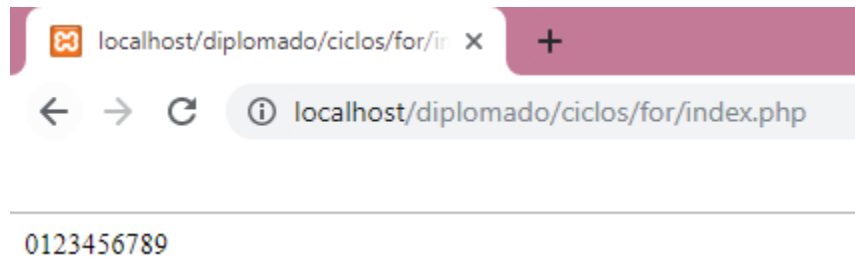
En el ciclo For, es necesario tener un valor inicial, una condición y valor final, y opcionalmente hacer uso del tamaño del "paso" entre cada iteración del ciclo.

Es decir, un ciclo For es una estructura iterativa para ejecutar un mismo segmento de código una cantidad de veces deseada; conociendo previamente un valor de inicio, un tamaño de paso (incremento) y un valor final para el ciclo.

```
for (init counter; test counter; increment counter) {  
    code to be executed;  
}
```



```
1 <?php  
2  
3 // Ciclos - Ciclo For  
4  
5 for ($i=0; $i < 10; $i++)  
6 {  
7     echo $i;  
8 }  
9  
10 ?>
```



Estructura del ciclo For

La estructura del ciclo For se componen de 3 elementos fundamentales:

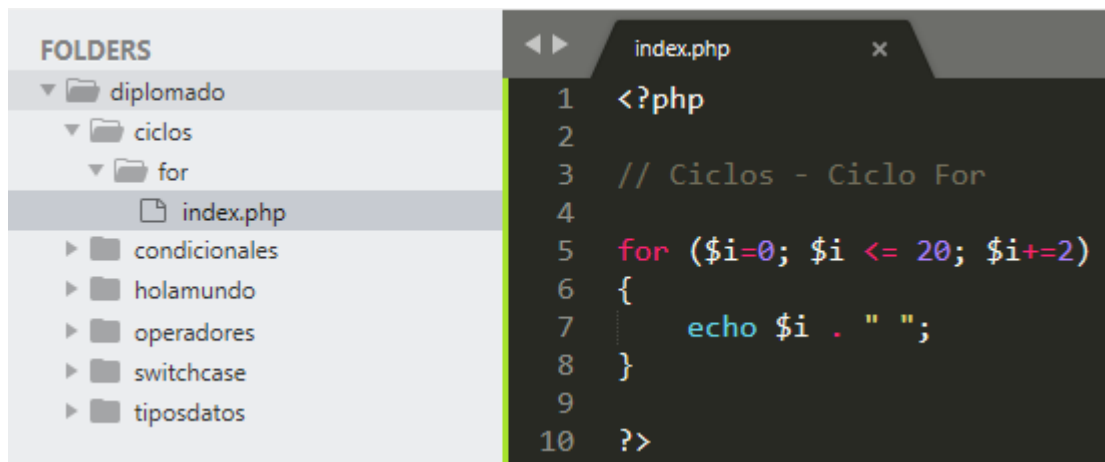
- Valor de inicio: determina a partir de qué momento y de qué valor el ciclo inicia las iteraciones.
- Condicional – Valor de parada: evalúa los casos en que el ciclo realizará las iteraciones y el momento en qué terminarán.
- Valor de incremento - decremento: realiza un incremento o decremento al valor de inicio para realizar las iteraciones.
-

```
for ($i=0; $i < 10; $i++)  
{  
    // $i = 0 : Valor de inicio  
    // $ < 10 : Condicional - Valor de parada  
    // $++    : Valor de incremento (De 1 en 1)  
}
```

Para entender mejor el funcionamiento y aplicación del ciclo para observa los siguientes ejercicios:

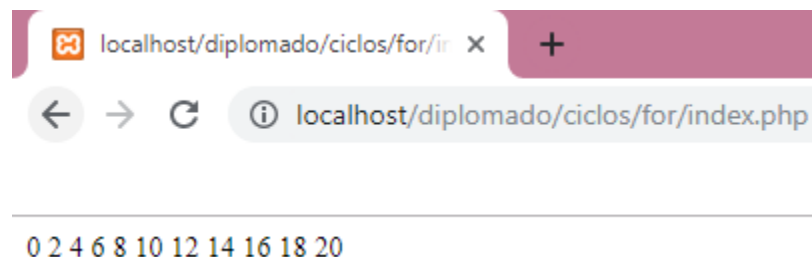
Ejercicios

- Desarrollar un programa que muestre los números pares entre 0 – 20 de forma ascendente.

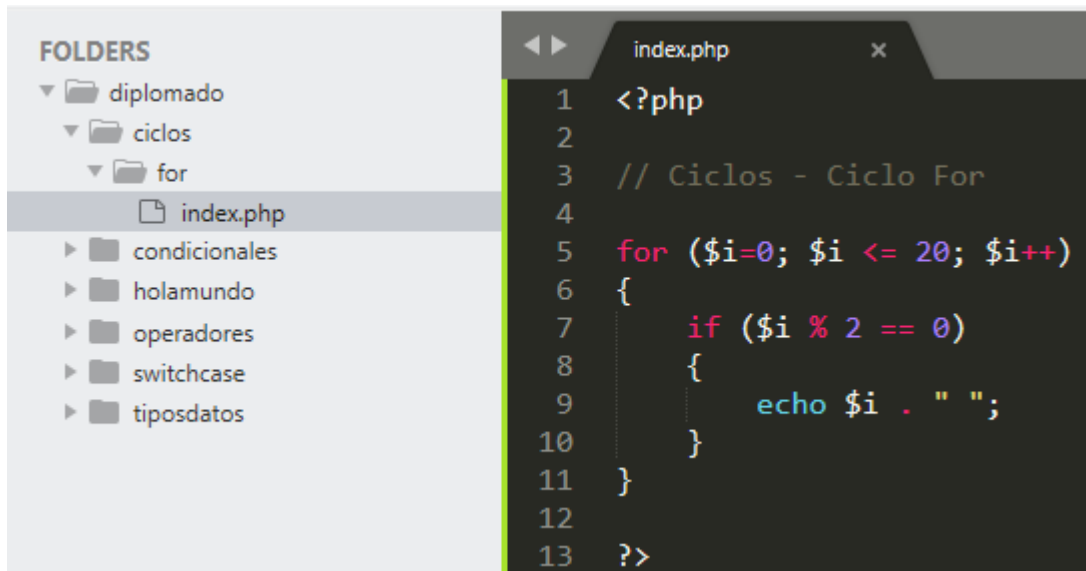


```
FOLDERS
└─ diplomado
   └─ ciclos
      └─ for
         └─ index.php
            └─ condicionales
            └─ holamundo
            └─ operadores
            └─ switchcase
            └─ tiposdatos

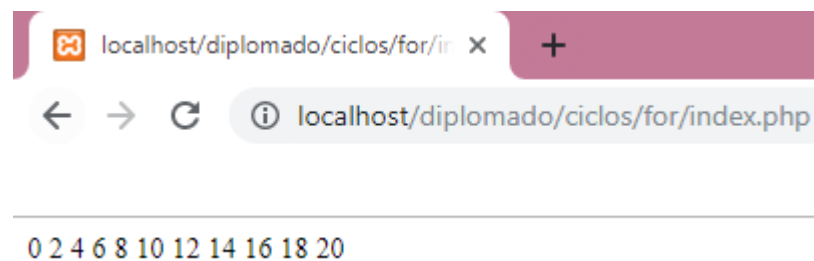
index.php
1 <?php
2
3 // Ciclos - Ciclo For
4
5 for ($i=0; $i <= 20; $i+=2)
6 {
7     echo $i . " ";
8 }
9
10 ?>
```



Para este caso, se conocen de entrada los valores de inicio (0) y cierre/final del ciclo (20), el ejercicio debe mostrar únicamente valores pares; el incremento es diferente al visto en los ejemplos de la estructura (i++) dado que, al declarar el incremento en uno, el ejercicio imprimiría todos los valores que hay entre 0 y 20 (0,1,2...18,19 y 20) por esto mismo el incremento de `$i+=2`; el ejercicio funcionaría con un incremento unario en el siguiente caso:



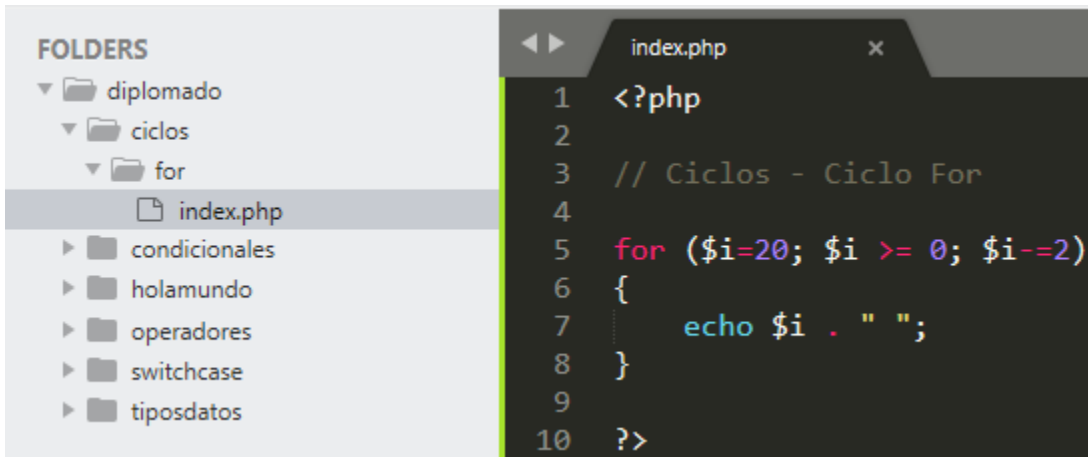
```
1 <?php
2
3 // Ciclos - Ciclo For
4
5 for ($i=0; $i <= 20; $i++)
6 {
7     if ($i % 2 == 0)
8     {
9         echo $i . " ";
10    }
11 }
12
13 ?>
```



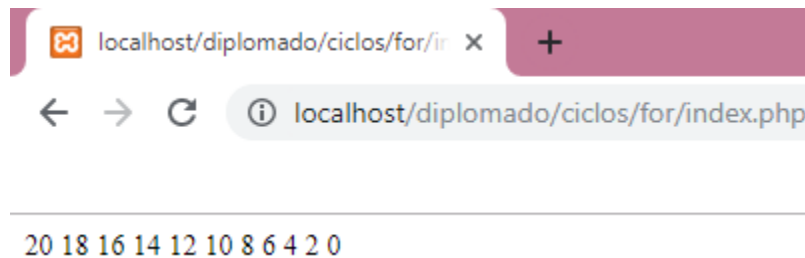
Dado que se recorre uno a uno todos los valores entre 0 y 20, pero posteriormente válida por medio de la condicional que únicamente muestre valores divisibles entre 2 (recuerda el uso del módulo "%") y cumple la misma función del primer método, aunque con más líneas de código, lo cual no es eficiente.

Retomando el primer método, el incremento debe ser en ($\$i+=2$) para que las iteraciones se realicen adecuadamente en intervalos de dos (2) en dos (2) hasta el valor de parada que será (20).

- ¿Cómo sería el ejemplo anterior descendientemente? Es decir, no de 0 a 20, sino de 20 a 0.

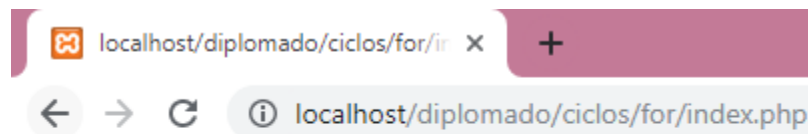
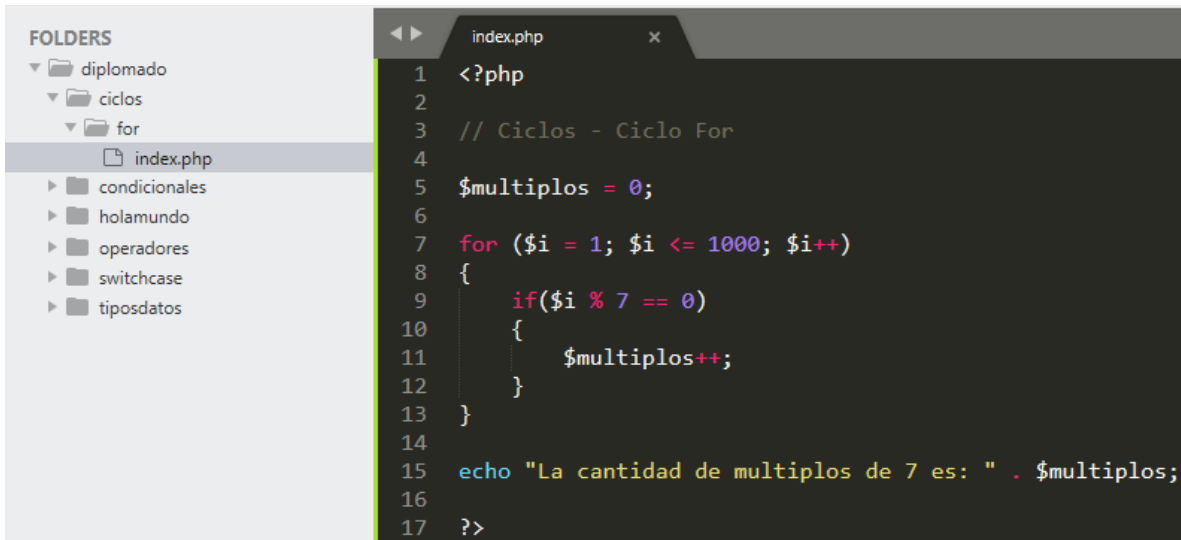


```
1 <?php
2
3 // Ciclos - Ciclo For
4
5 for ($i=20; $i >= 0; $i-=2)
6 {
7     echo $i . " ";
8 }
9
10 ?>
```



El ejercicio funciona de igual forma en base a los valores que se determinan para el funcionamiento correcto del ejercicio, el valor de inicio cambiaría a (20) para realizar el decremento, la condición de parada se realizará cuando el valor de inicio llegue a (0) y ya no se realiza un incremento gradual de dos (2) en dos (2), sino un decremento en los mismos valores.

- Desarrollar un programa que permita contar cuántos números entre 1 y 1000 son múltiplos de 7, mostrar el resultado final.



La cantidad de multiplos de 7 es: 142

En el desarrollo de este ejercicio entra en uso un concepto muy común en el uso de ciclos, el cuál son los contadores, que no son más que variables que se incrementan consecutivamente según el valor que se determine, la implementación se realiza declarando la variable contadora fuera del ciclo ¿Por qué?, si se declara dentro del ciclo pierde completamente su uso y funcionalidad, el ciclo al ser una estructura repetitiva, declarar la variable las veces que fueron determinadas en el ciclo y el valor se reiniciaría por cada iteración, al ser declarado fuera, el uso se limita sólo al ciclo.

Los valores del ciclo para este caso son: en el inicio empezará en (1) y el valor de parada será (1000), el incremento será gradual en el para recorrer todos los valores y la condicional interna del ciclo

determinará los múltiplos de 7 y realizará el incremento del contador, al final del ciclo y la condicional por defecto, se encuentra respectivamente el valor de impresión.

- Desarrollar un problema que imprimir las tablas de multiplicar del 1 al 10 con su respectivo resultado.

¿Se puede tener un ciclo dentro de un ciclo?



```
1 <?php
2
3 // Ciclos - Ciclo For
4
5 for ($i = 1; $i <= 10; $i++)
6 {
7     echo "Tabla de multiplicar del " . $i . "<br>";
8     for ($x = 1; $x <= 10; $x++)
9     {
10         echo $i . " * " . $x . " = " . $i * $x . "<br>";
11     }
12 }
13
14
15 ?>
```

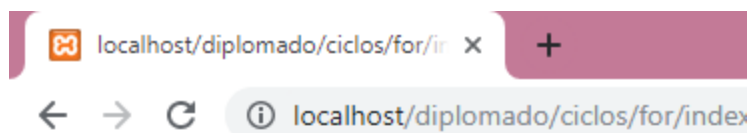


Tabla de Multiplicar del 1

1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
1 * 4 = 4
1 * 5 = 5
1 * 6 = 6
1 * 7 = 7
1 * 8 = 8
1 * 9 = 9
1 * 10 = 10

Tabla de Multiplicar del 2

2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10
2 * 6 = 12
2 * 7 = 14
2 * 8 = 16
2 * 9 = 18
2 * 10 = 20

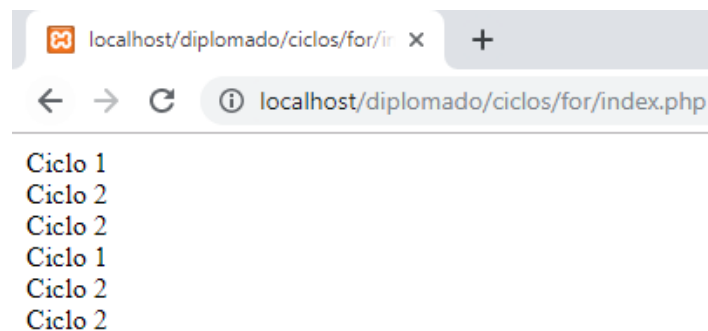
Tabla de Multiplicar del 3

3 * 1 = 3
3 * 2 = 6
3 * 3 = 9
3 * 4 = 12
3 * 5 = 15
3 * 6 = 18
3 * 7 = 21
3 * 8 = 24
3 * 9 = 27
3 * 10 = 30

El uso de un ciclo dentro de un ciclo es una operación que se puede realizar con normalidad dentro del código en PHP y opera de la siguiente forma:



```
1 <?php
2
3 // Ciclos - Ciclo For
4
5 for ($i = 1; $i <= 2; $i++)
6 {
7     echo "Ciclo 1 <br>";
8     for ($x = 1; $x <= 2; $x++)
9     {
10         echo "Ciclo 2 <br>";
11     }
12 }
13
14 ?>
```



Detenidamente el uso de un ciclo (1) dentro de otro ciclo (2) condiciona a que éste (2) se repita las veces que (1) tenga expresadas. Como se observa en el ejemplo anterior, el ciclo (1) cuenta con 2 iteraciones, al igual que el ciclo (2) y se ejecuta de la siguiente forma:

- Primera ejecución del ciclo (1):

Ciclo 1

- Primera ejecución del ciclo (2):

Ciclo 1
Ciclo 2

- Segunda ejecución del ciclo (2):

Ciclo 1	
Ciclo 2	Ciclo 2

- Segunda ejecución del ciclo (1) – reinicio del ciclo (2):

Ciclo 1	Ciclo 1
Ciclo 2	Ciclo 2

- Primera ejecución del ciclo (2):

Ciclo 1		Ciclo 1	
Ciclo 2	Ciclo 2	Ciclo 2	

- Segunda ejecución del ciclo (2) – fin del ciclo (1):

Ciclo 1		Ciclo 1	
Ciclo 2	Ciclo 2	Ciclo 2	Ciclo 2

El proceso y resultado de las tablas de multiplicar es sencillo en ambos ciclos se tienen los valores y las características para operar, véase: Ejemplo tabla de multiplicar del 1.

Ciclo 1 - I	Ciclo 2 - X	Resultado
1	1	1
1	2	2
1	3	3
1	4	4
1	5	5
1	6	6
1	7	7
1	8	8
1	9	9
1	10	10

El ciclo (1) con la variable I proporciona la tabla que se está evaluado y el ciclo (2) con la variable X proporciona los valores que se deben multiplicar de forma que por cada iteración del ciclo (1) se obtiene el resultado gracias al ciclo (2) con una simple multiplicación.

Recursos disponibles para el aprendizaje



Los ciclos for son un concepto muy importante y en especial en la programación con PHP, en la documentación del lenguaje hay información muy importante y que te puede ser útil, visítala, Disponible en:
<https://www.php.net/manual/en/control-structures.for.php>

Ejercicio

¿Deseas profundizar en la temática de los ciclos for? Entonces te sugiero realizar los siguientes ejercicios que pondrán a prueba los conocimientos adquiridos.
(MÓDULO 2 – EJERCICIOS DE CICLOS FOR)
¡Inténtalo! 👍



Ciclo While

Los ciclos While (o ciclos mientras) son una estructura de control cíclica, que permite ejecutar una o varias líneas de código de forma iterativa (o repetitiva) al igual que el ciclo for, pero sin tener un control y conocimiento sobre las iteraciones. En el ciclo For, era necesario tener un valor inicial, una condición y valor final, en el ciclo While únicamente se tiene control del ciclo por medio de una condicional en la declaración que determina si el ciclo continúa o se detiene, por lo que la sintaxis es más simple.

Es decir, un ciclo While es una estructura iterativa para ejecutar un mismo segmento de código, con la particularidad de que en la mayoría de los casos se desconoce la cantidad de veces deseada para iterar por el hecho de no tener un valor de inicio y un valor de parada, puesto que si se conoce la cantidad de veces que se desea iterar es más simple el uso del ciclo for.

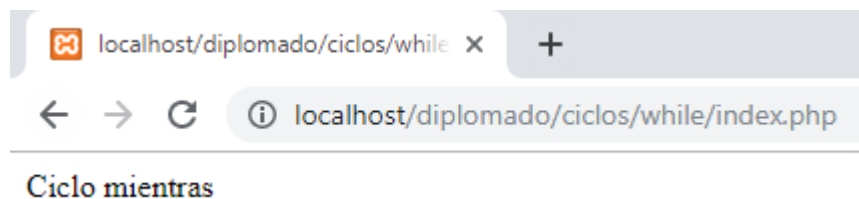
```
while (condition is true) {  
    code to be executed;  
}
```

FOLDERS

- ▼ diplomado
 - ▼ ciclos
 - ▶ for
 - ▶ while
 - ▶ condicionales
 - ▶ holamundo
 - ▶ operadores
 - ▶ switchcase
 - ▶ tiposdatos
 - ▶ variables

index.php

```
1 <?php
2
3 // Ciclos - While
4
5 $x = true;
6
7 while ($x)
8 {
9     echo "Ciclo mientras";
10    $x = false;
11 }
12
13 ?>
```

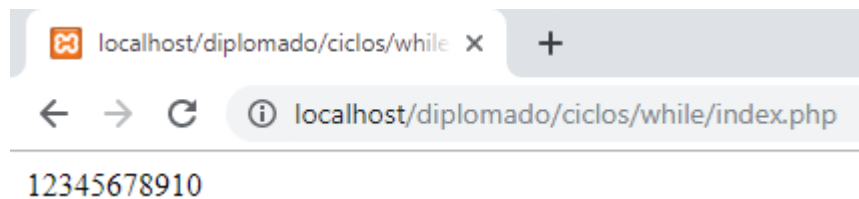


FOLDERS

- ▼ diplomado
 - ▼ ciclos
 - ▶ for
 - ▶ while
 - ▶ condicionales
 - ▶ holamundo
 - ▶ operadores
 - ▶ switchcase
 - ▶ tiposdatos
 - ▶ variables

index.php

```
1 <?php
2
3 // Ciclos - While
4
5 $x = 1;
6
7 while ($x <= 10)
8 {
9     echo $x;
10    $x++;
11 }
12
13 ?>
```



Estructura del ciclo While

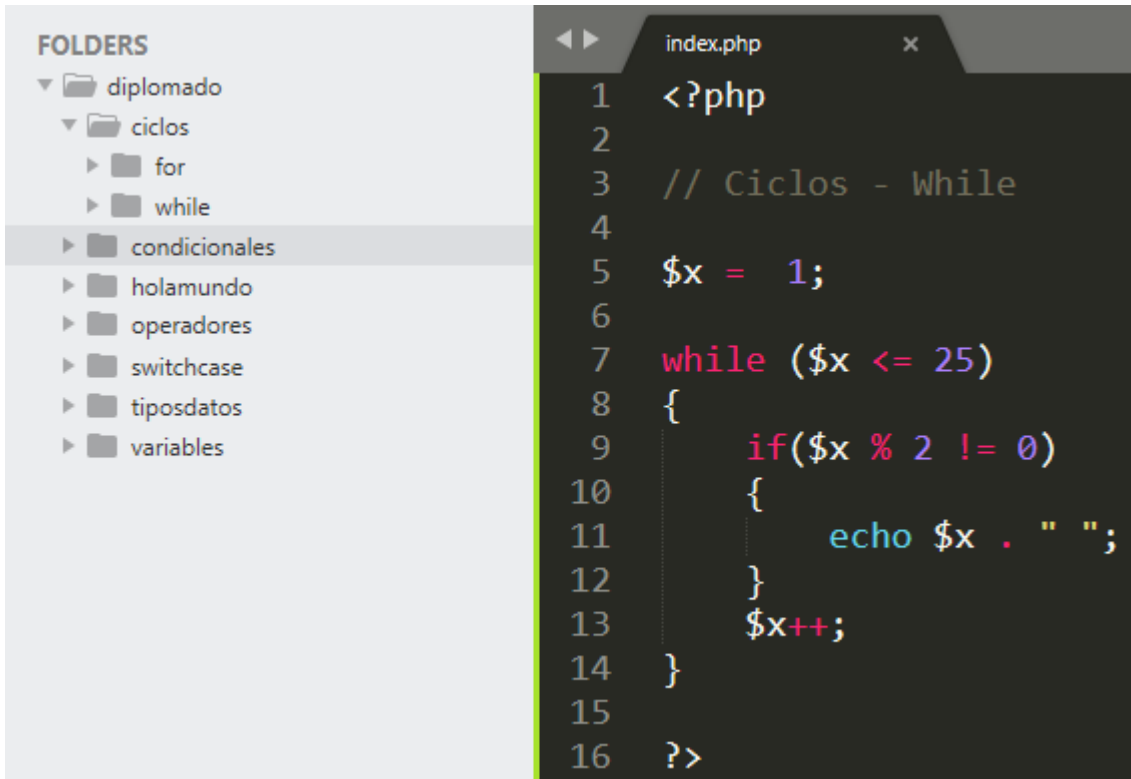
La estructura del ciclo While se componen principalmente de una condición que tiene que tomar un valor booleano (verdadero o falso). Si este valor es verdadero, se ejecutará la sentencia. Concluida esta acción se vuelve a evaluar la condición. Proseguirán el ciclo hasta que la condición cambie a falso.

Esta es una estructura de iteración preprueba, es decir primero se evalúa la condición antes de realizar cualquier acción. Si de entrada la condición es falsa nunca ejecutará el conjunto de sentencias.

Al igual que ocurre con las condicionales, dentro de la declaración de las llaves se ubica respectivamente el código que se desea ejecutar de forma repetitiva en caso de obtener un resultado verdadero, los componentes que definen la estructura son de acuerdo a cómo se detallan en los ejemplos, una variable para el control de la condicional, la respectiva y característica condicional (el punto más importante) y finalmente la iteración (incremento, decremento o cambio), esta es la estructura recomendada.

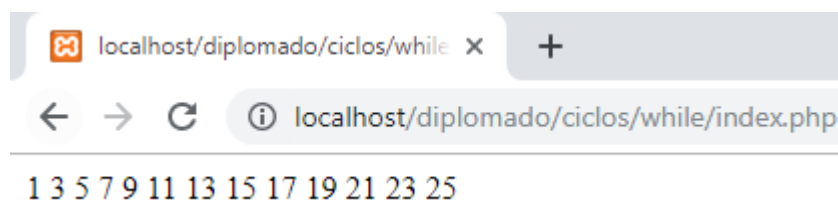
Ejercicios

- Desarrollar un programa que imprima los números impares entre 1 y 25.



```
FOLDERS
└─ diplomado
  └─ ciclos
    └─ for
    └─ while
    └─ condicionales
    └─ holamundo
    └─ operadores
    └─ switchcase
    └─ tiposdatos
    └─ variables

index.php
1 <?php
2
3 // Ciclos - While
4
5 $x = 1;
6
7 while ($x <= 25)
8 {
9     if($x % 2 != 0)
10    {
11        echo $x . " ";
12    }
13    $x++;
14 }
15
16 ?>
```



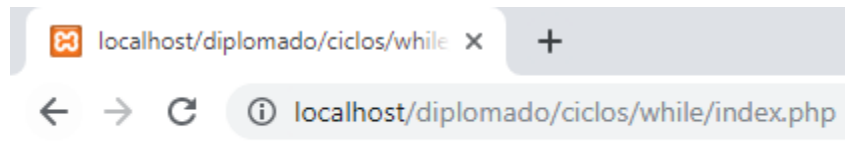
Este caso tiene un funcionamiento simple y una aplicación similar a los ejercicios desarrollados con la estructura del ciclo for; el valor de inicio y final del ciclo lo determina el ejercicio (1 – 25), la condición principal que se debe determinar dentro del While debe ser recorrer el ciclo siempre y cuando se cumpla la condición (ser menor o igual a 25), el incremento como se especifica en la estructura debe ser dentro del mismo ciclo y gradualmente en uno (1) es decir (x++). Dentro de las sentencias a ejecutar dentro del ciclo se encuentra la condicional que determina qué número es impar por medio del módulo (%) y el operador de diferencia (! =) dado que un número entero cuyo módulo sea diferente a 0 significa que es un número impar.

- Desarrollar un programa que, dada una palabra, descomponer todos sus caracteres.



The screenshot shows a code editor with a file named 'index.php'. On the left, a 'FOLDERS' panel lists a directory structure: 'diplomado' (expanded) contains 'ciclos' (expanded), which includes 'for' and 'while', and other folders like 'condicionales', 'holamundo', 'operadores', 'switchcase', 'tiposdatos', and 'variables'. The main editor area shows the following PHP code:

```
1 <?php
2
3 // Ciclos - While
4
5 $palabra = "DIEGO VALENCIA";
6 $i = 0;
7
8 while ($i < strlen($palabra))
9 {
10     echo $palabra[$i] . " ";
11     $i++;
12 }
13
14 ?>
```

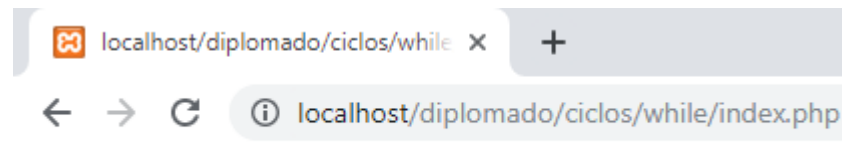
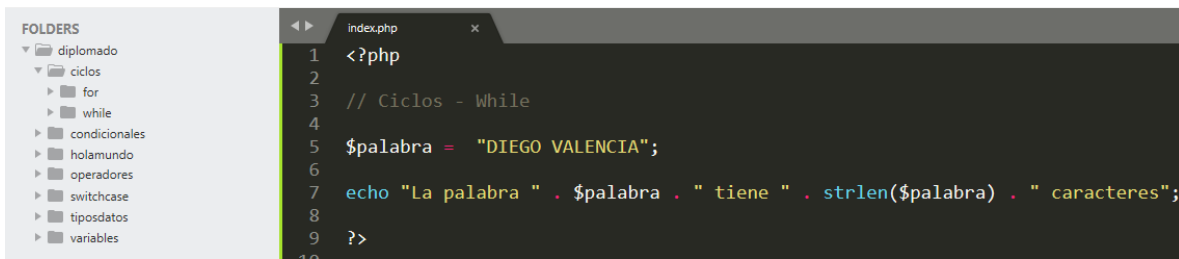


DIEGO VALENCIA

Hay un método nuevo que se aplica para solucionar este ejercicio y una función nueva de las cadenas y que son muy útiles, éstos son:

- **strlen ()**: Devuelve el número de caracteres de la cadena.

Se debe tener en cuenta que el método reconoce el espacio como un carácter.



La palabra DIEGO VALENCIA tiene 14 caracteres

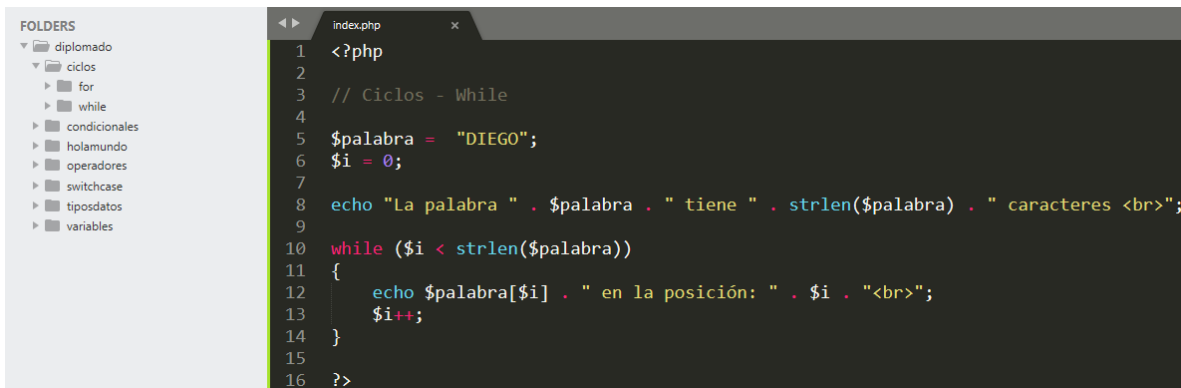
Palabra													
D	I	E	G	O	V	A	L	E	N	C	I	A	
14 Caracteres													

- **\$palabra[]**: Devuelve el carácter de una posición de la cadena.

Para describirse mejor, este se aplica sobre variables y se encarga de descomponer la variable en posiciones de 0 a N-1 (N siendo el número de caracteres) a las cuales se puede acceder por medio de un índice (un número asignado a la posición) por ejemplo:

Palabra				
D	I	E	G	O
Posiciones				
0	1	2	3	4

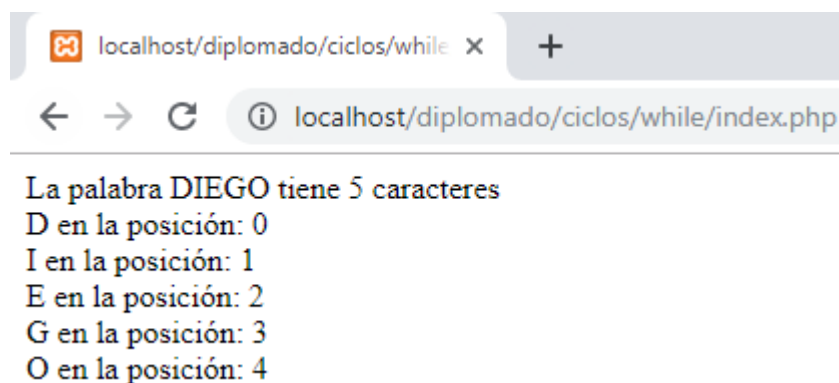
La palabra "Diego" está compuesta por 5 caracteres por lo que debe tener 5 posiciones asignadas a cada carácter (Recordar que siempre se empieza desde 0) como se observa en la tabla), ¿cómo sería el código?



```

1 <?php
2
3 // Ciclos - While
4
5 $palabra = "DIEGO";
6 $i = 0;
7
8 echo "La palabra " . $palabra . " tiene " . strlen($palabra) . " caracteres <br>";
9
10 while ($i < strlen($palabra))
11 {
12     echo $palabra[$i] . " en la posición: " . $i . "<br>";
13     $i++;
14 }
15
16 ?>

```



Lo primero a tener en cuenta es el uso correcto de las posiciones y evitar el desborde en el recorrido de la palabra, el desborde se presenta al buscar una posición que no existe dentro de la palabra. Por ejemplo, como se observa en la tabla, la palabra "Diego" cuenta con 5 caracteres,

pero la posición 5 no existe (Recuerda el N-1) y arroja lo siguiente al ejecutar:

Notice: Uninitialized string offset: 5 in C:\xampp\htdocs\diplomado\ciclos\while\index.php on line 12
en la posición: 5

PHP se sale del rango que se tiene en la palabra y realiza una excepción de tipo: Uninitializaed string offset.

Determinar la condicional dentro del ciclo define el correcto funcionamiento del ejercicio, permite aclarar el caso final de parada del ciclo y evita el desbordamiento en el recorrido de la palabra.

- Desarrollar un programa que genere números aleatorios entre 1 y 50 y se detenga cuando genere un múltiplo de 10.

Este es uno de los casos donde la implementación del ciclo While es más efectiva, dado que no se conoce cuántas veces debe iterar el ciclo, pero sí se conoce el caso de parada, un ciclo for no tendría la isma funcionabilidad que el While por su estructura:

FOLDERS

- ▼ diplomado
 - ▼ ciclos
 - ▶ for
 - ▶ while
 - ▶ condicionales
 - ▶ holamundo
 - ▶ operadores
 - ▶ switchcase
 - ▶ tiposdatos
 - ▶ variables

index.php x

```

1 <?php
2
3 // Ciclos - While
4
5 $parada = true;
6 $numero = 0;
7
8 while ($parada)
9 {
10     $numero = rand(1,50);
11     echo $numero . " ";
12
13     if($numero % 10 == 0)
14     {
15         $parada = false;
16     }
17 }
18
19 ?>

```

localhost/diplomado/ciclos/while x +

localhost/diplomado/ciclos/while/index.php

 35 7 35 14 13 5 3 25 14 41 29 28 36 12 46 5 41 22 8 27 46 7 30

localhost/diplomado/ciclos/while x +

localhost/diplomado/ciclos/while/index.php

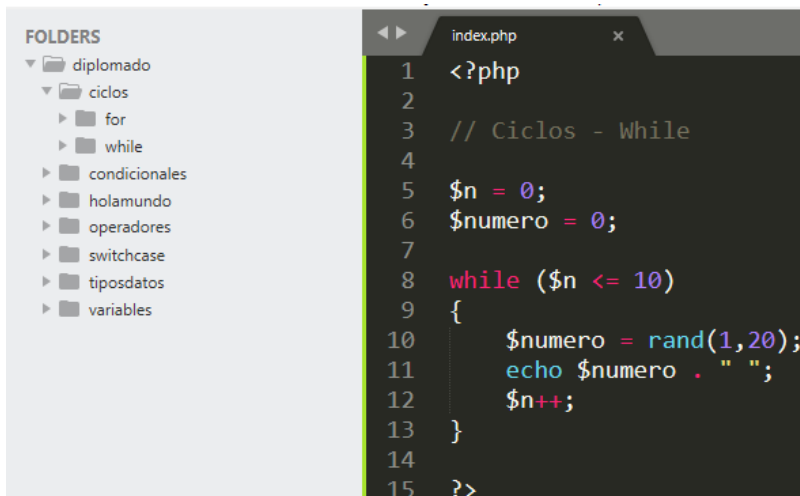
 46 13 47 3 3 40

Para la solución de este ejercicio se utilizará lo que se conoce como banderas, las banderas son variables booleanas que cambian cuando se cumplen ciertas características que se determine en el código, en este caso la bandera será la variable \$parada inicializada en verdadero (para realizar la primera iteración del ciclo) y el cambio de la bandera se dará cuando el número generado sea un múltiplo de 10 (Recordar el uso del módulo).

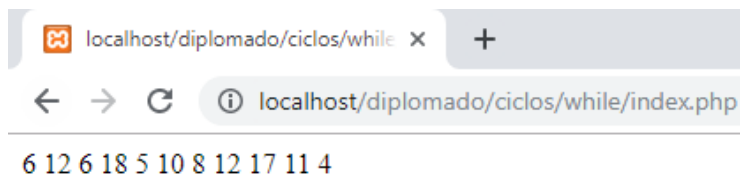
Para generar un número aleatorio se utiliza una nueva función, **rand** para generar los números entre 1 y 50.

La ejecución del ejercicio termina cuando inmediatamente encuentre el primer múltiplo de 10, dado que se desconoce en qué momento se generará, se utiliza el ciclo While y la bandera.

Rand() genera números aleatorios a partir de un rango determinado, para el funcionamiento de este, se deben determinar un límite inferior y un límite superior, en el caso del ejemplo anterior (1,5).



```
1 <?php
2
3 // Ciclos - While
4
5 $n = 0;
6 $numero = 0;
7
8 while ($n <= 10)
9 {
10     $numero = rand(1,20);
11     echo $numero . " ";
12     $n++;
13 }
14
15 ?>
```



Recursos disponibles para el aprendizaje



Los ciclos while son un concepto muy importante y en especial en la programación con PHP, en la documentación del lenguaje hay información muy importante y que te puede ser útil, visítala, Disponible en: <https://www.php.net/manual/en/control-structures.while.php>

Ejercicio

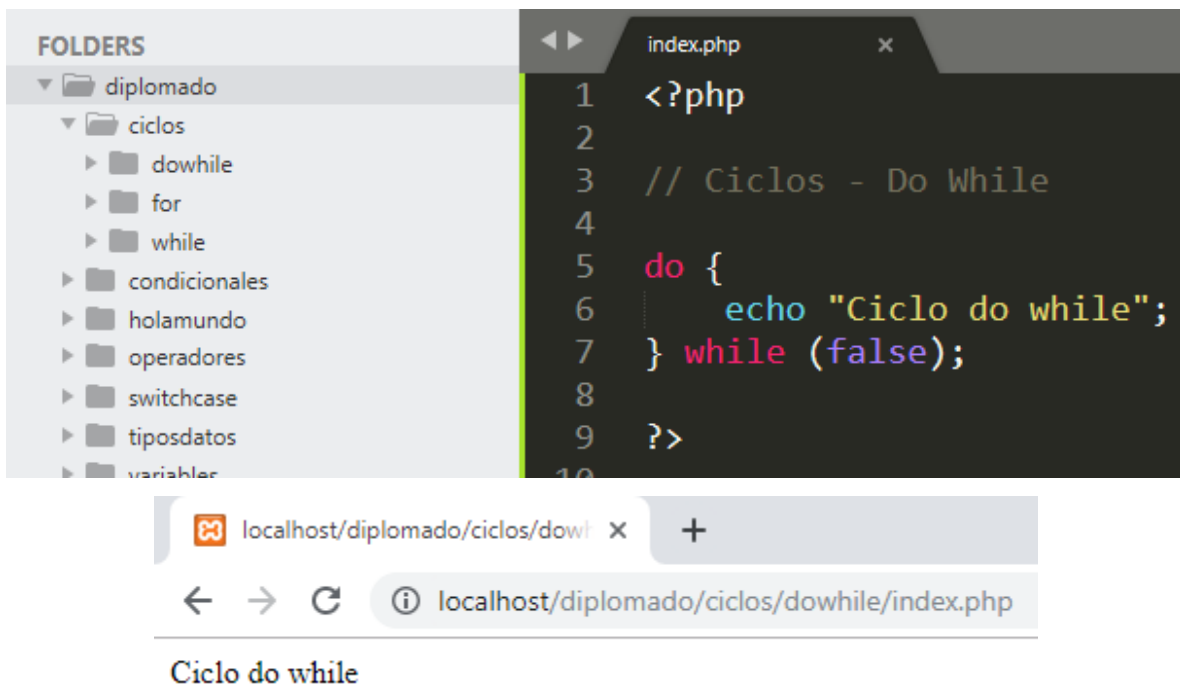
¿Deseas profundizar en la temática de los ciclos while? Entonces te sugiero realizar los siguientes ejercicios que pondrán a prueba los conocimientos adquiridos. **(MÓDULO 2 – EJERCICIOS DE CICLOS WHILE)**
¡Inténtalo! 👍

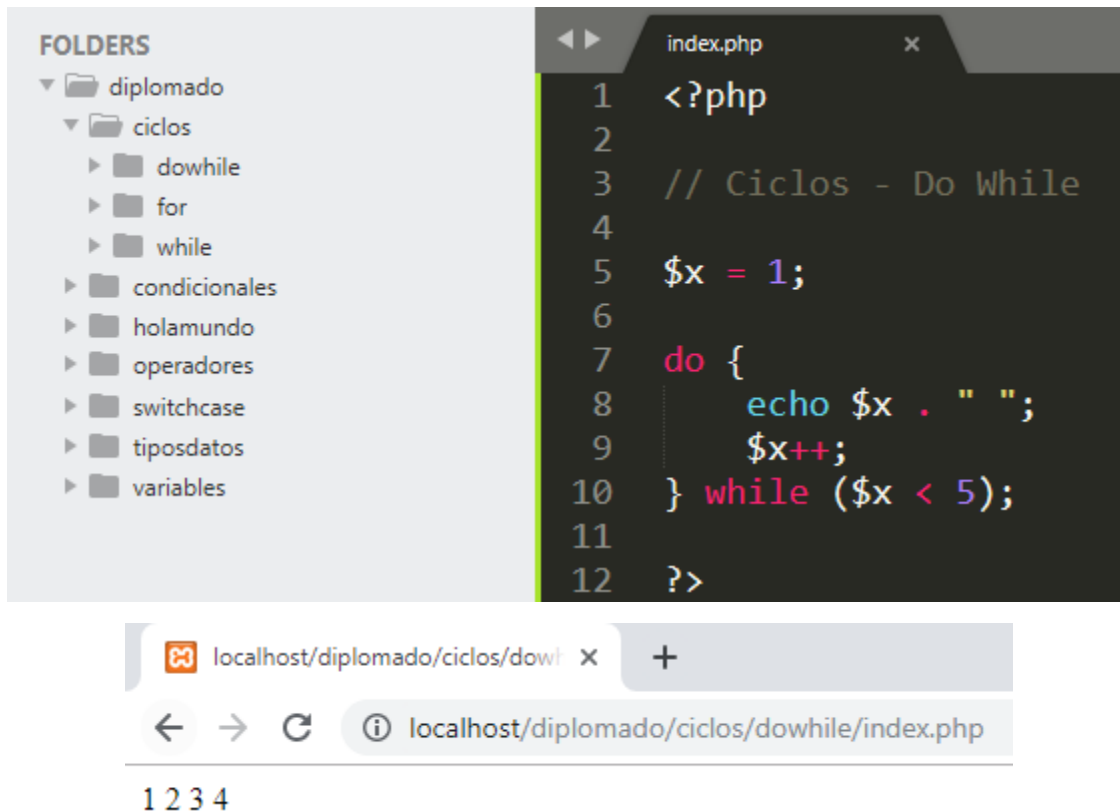


Ciclo do While

Esta estructura de iteración cumple el mismo objetivo de la estructura While con la variante que el ciclo do While ejecuta cuando menos una vez antes de evaluarse la condición del ciclo por lo que siempre se tendrá una iteración así el ciclo nunca haya entrado en ejecución.

```
do {  
    code to be executed;  
} while (condition is true);
```





The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a directory structure: diplomado > ciclos > dowhile. The code editor shows the following PHP code:

```
1 <?php
2
3 // Ciclos - Do While
4
5 $x = 1;
6
7 do {
8     echo $x . " ";
9     $x++;
10 } while ($x < 5);
11
12 ?>
```

Below the code editor, a browser window is shown with the address bar displaying: localhost/diplomado/ciclos/dowhile/index.php. The browser shows the output of the PHP code: 1 2 3 4.

Estructura del ciclo do While

La estructura del ciclo do While cuenta con los mismos componentes del ciclo While en cuando a su forma de funcionar, con la implementación del do, que se encarga de almacenar las sentencias que se ejecutarán en todas las iteraciones para su posterior ejecución.

Esta es una estructura de iteración - prueba, es decir primero se ejecutan las sentencias del ciclo una vez y después se evalúa la condición antes de realizar las iteraciones esperadas. Si de entrada la condición es falsa, se tendrá una ejecución previa.

Al igual que ocurre con las condicionales, dentro de la declaración de las llaves respectivas del do, se ubica el código de las sentencias que se desean ejecutar de forma repetitiva en caso de obtener un resultado

verdadero, los componentes que definen la estructura son de acuerdo a cómo se detallan en los ejemplos, una variable para el control de la condicional, un bloque encargado de ejecutar las sentencias y finalmente la respectiva y característica condicional del ciclo.

TEMA 4 Arrays

Los arrays en PHP son un tipo de estructura de datos que permite almacenar múltiples elementos de un tipo de datos similar en una sola variable, lo que ahorra el esfuerzo de crear una variable diferente para cada dato.

Los arrays son útiles para crear una lista de elementos de tipos similares, a los que se puede acceder utilizando su índice o clave.

Suponiendo que se desea almacenar cinco nombres e imprimirlos consecutivamente. Esto se puede hacer fácilmente mediante el uso de cinco variables de cadena diferentes. Pero si en lugar de cinco, el número aumenta a cien, entonces sería muy difícil. Aquí la matriz entra en juego y ayuda a almacenar cada elemento dentro de una sola variable y también permite un fácil acceso utilizando un índice o una clave.

Un array se crea usando una función array () en PHP.

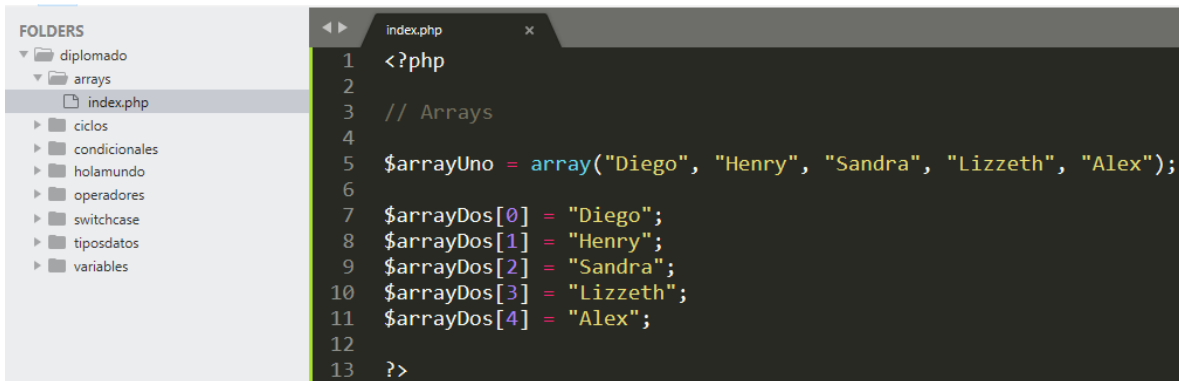
Básicamente, hay tres tipos de matrices en PHP:

- **Arrays indexadas:** un array con un índice numérico donde los valores se almacenan linealmente.
- **Arrays asociativas:** un array con un índice de cadena donde en lugar de almacenamiento lineal, a cada valor se le puede asignar una clave específica.

- **Arrays multidimensionales:** un array que contiene un array único o múltiple dentro de él y se puede acceder a través de múltiples índices.

Arrays indexados

Este tipo de array que se puede usar para almacenar cualquier tipo de elementos, pero un índice siempre es un número. Por defecto, el índice comienza en cero. Estas matrices se pueden crear de dos maneras diferentes, como se muestra en el siguiente ejemplo:



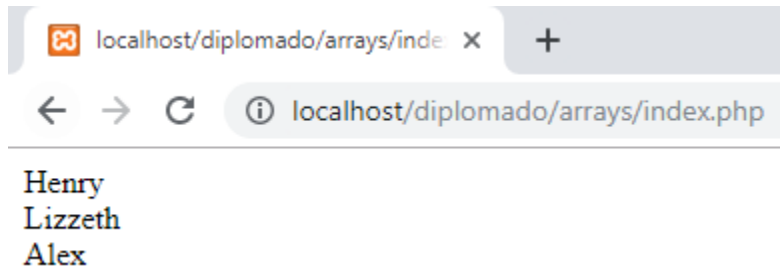
```
1 <?php
2
3 // Arrays
4
5 $arrayUno = array("Diego", "Henry", "Sandra", "Lizzeth", "Alex");
6
7 $arrayDos[0] = "Diego";
8 $arrayDos[1] = "Henry";
9 $arrayDos[2] = "Sandra";
10 $arrayDos[3] = "Lizzeth";
11 $arrayDos[4] = "Alex";
12
13 ?>
```

En la primera declaración se debe tener en cuenta que cada elemento que se desea almacenar debe ir separado por coma entre cada uno.

La forma de mostrar los datos directamente es muy simple, se acceden a estos a partir de los índices asignados automáticamente (empezando en 0) y de esta forma se obtiene el dato completo.

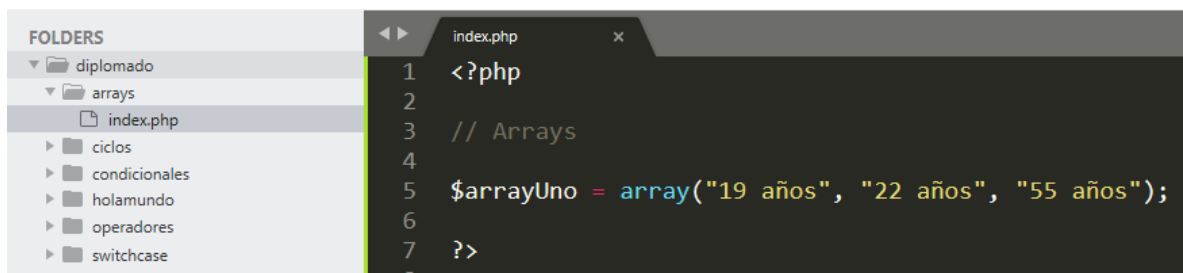


```
1 <?php
2
3 // Arrays
4
5 $arrayUno = array("Diego", "Henry", "Sandra", "Lizzeth", "Alex");
6
7 echo $arrayUno[1] . "<br>";
8 echo $arrayUno[3] . "<br>";
9 echo $arrayUno[4] . "<br>";
10
11 ?>
```



Estructura de un array indexado

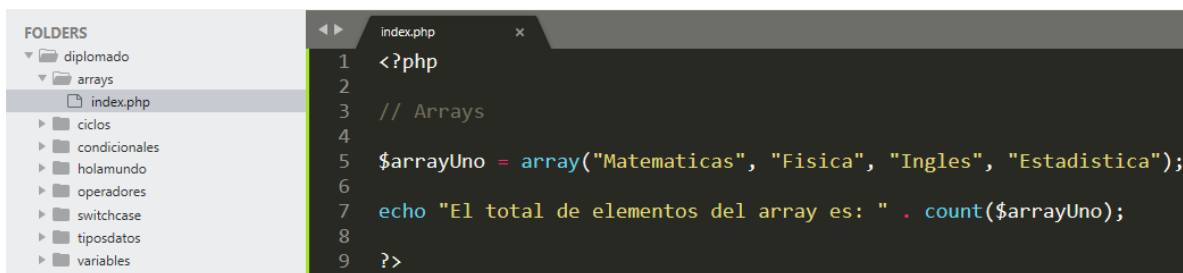
- Los elementos de un array se empiezan a numerar en 0, y permiten gestionar desde un solo elemento hasta múltiples de diferentes tipos.

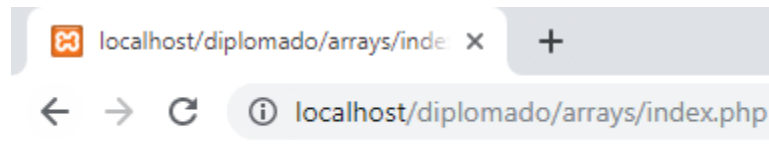


```
1 <?php
2
3 // Arrays
4
5 $arrayUno = array("19 años", "22 años", "55 años");
6
7 ?>
```

- Al igual que en la aplicación de expresión `$palabra[]` del módulo anterior, en los arrays cada dato almacenado le corresponde un índice, dado que, a partir de los índices, se operan los mismos.

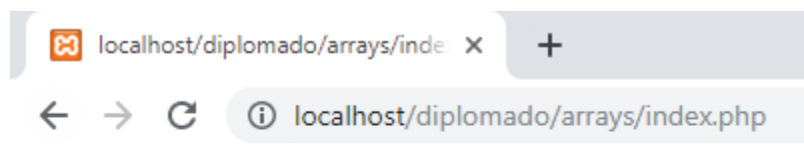
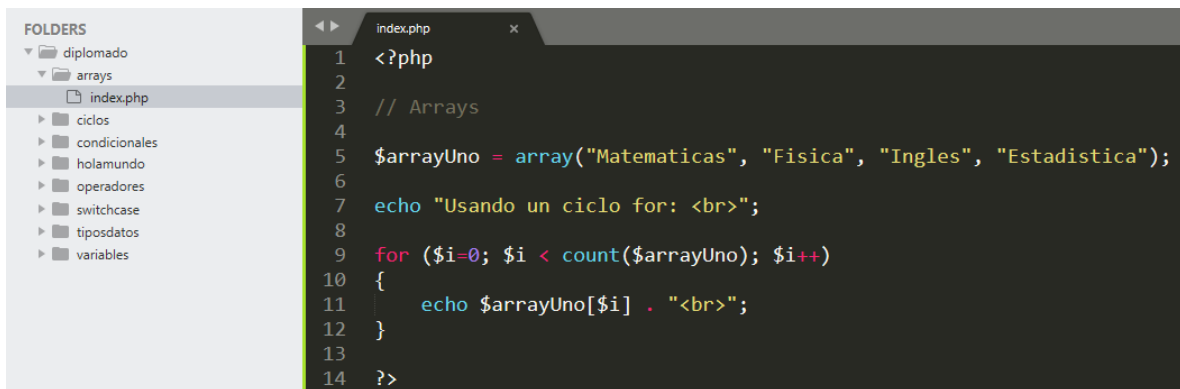
- [illegible]






El total de elementos del array es: 4

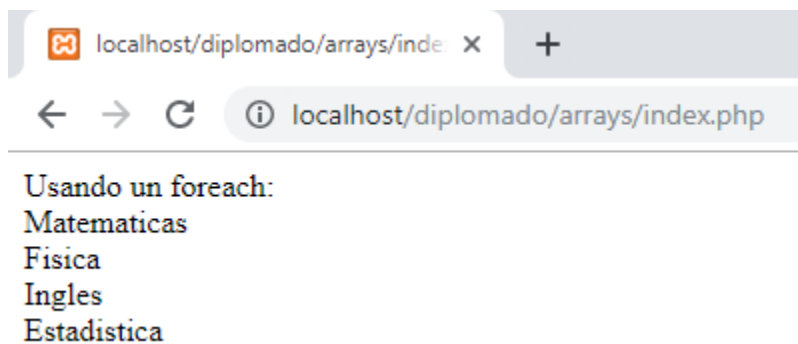
Recorrido: Se puede recorrer un array indexado usando bucles en PHP. Se puede recorrer de dos maneras. Primero usando un ciclo **for** y segundo usando un **foreach**. Ejemplo:



Usando un ciclo for:
Matematicas
Fisica
Ingles
Estadistica



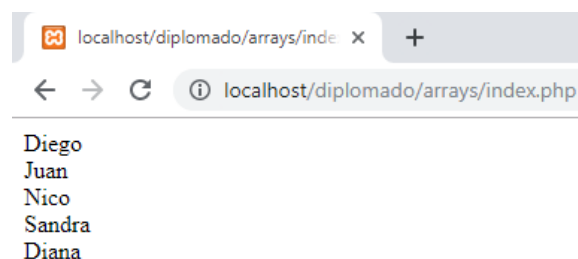
```
1 <?php
2
3 // Arrays
4
5 $arrayUno = array("Matematicas", "Fisica", "Ingles", "Estadistica");
6
7 echo "Usando un foreach: <br>";
8
9 foreach ($arrayUno as $dato)
10 {
11     echo $dato . "<br>";
12 }
13
14 ?>
```



Foreach

El bucle foreach es un tipo especial de bucle que permite recorrer estructuras que contienen varios elementos (como arrays, recursos u objetos) sin necesidad de preocuparse por el número de elementos.

```
5 $personas = array("Diego", "Juan", "Nico", "Sandra", "Diana");
6
7 foreach ($personas as $persona)
8 {
9     echo $persona . "<br>";
10 }
```



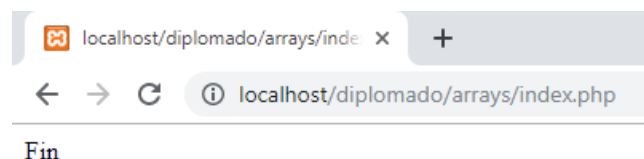
La ejecución de esta estructura de control es la siguiente:

- Si el array no contiene elementos, este no se ejecuta.
- Si el array contiene elementos:
 - Se asigna el primer valor del array a la variable auxiliar (\$persona)
 - (*) Se ejecuta el bloque de sentencias. (Impresión del valor).
 - Si el array no contiene más elementos, el array deja de ejecutarse.
 - Si el array todavía contiene más elementos:
 - Se asigna el siguiente valor del array a la variable auxiliar.
 - Se vuelve al punto (*) (es decir, se ejecuta de nuevo el bloque de sentencias, etc.)

El array se ejecuta tantas veces como elementos tiene el array. En cada iteración, la variable auxiliar \$palabra van tomando los valores del array para ese índice.

Un ejemplo del array vacío puede ser el siguiente:

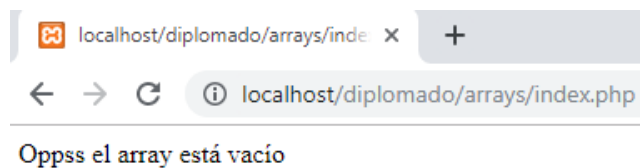
```
5  $personas = array();
6
7  foreach ($personas as $persona)
8  {
9      echo $persona . "<br>";
10 }
11
12 echo "Fin";
```



Al array estar vacío, se salta directamente el bloque del foreach y pase al echo final.

Una forma de controlar si el array está vacío es por medio de una condicional:

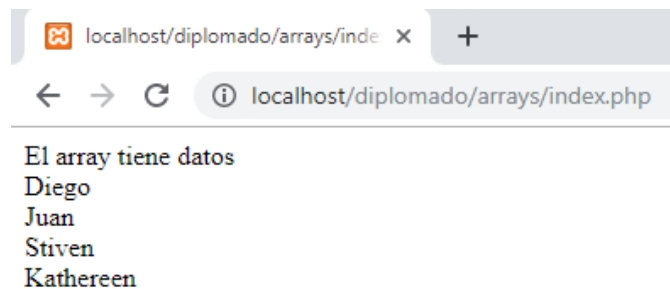
```
5  $personas = array();
6
7  if($personas != null)
8  {
9      echo "El array tiene datos <br>";
10     foreach ($personas as $persona)
11     {
12         echo $persona . "<br>";
13     }
14 }
15 else
16 {
17     echo "Oppss el array está vacío";
18 }
```



Cuando un array se encuentra vacío, directamente toma el valor de null, de esta forma es fácil realizar el control entre sí tiene datos o si está vacío.

Ahora el ejemplo teniendo datos en el array:

```
5  $personas = array("Diego","Juan","Stiven","Kathereen");
6
7  if($personas != null)
8  {
9      echo "El array tiene datos <br>";
10     foreach ($personas as $persona)
11     {
12         echo $persona . "<br>";
13     }
14 }
15 else
16 {
17     echo "Oppss el array está vacío";
18 }
```



Recursos disponibles para el aprendizaje

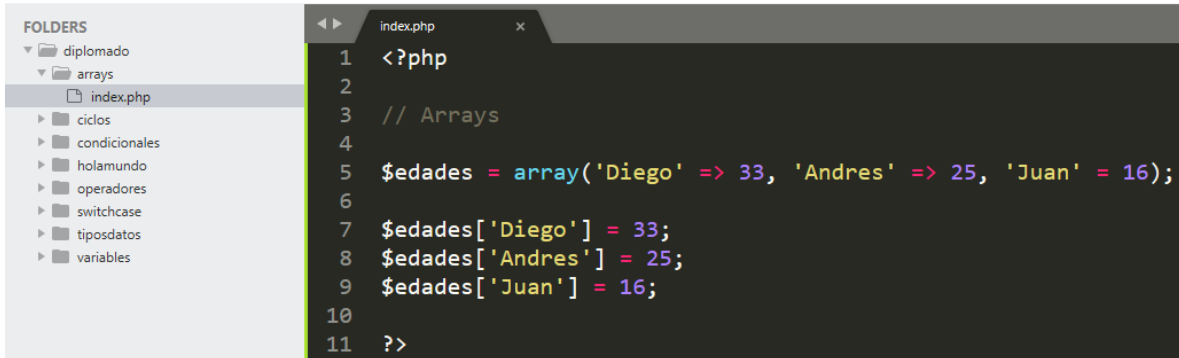


Para desarrollar las habilidades y destrezas necesarias en cada competencia, es muy importante que tengas acceso a los recursos didácticos adecuados.

Los foreach juegan un papel fundamental a la hora de trabajar con arrays, más adelante en el diplomado verás su utilidad en la programación web, por esto, te invito a leer un poco sobre la definición de este concepto, disponible en: <https://www.php.net/manual/es/control-structures.foreach.php>

Arrays asociativos

Este tipo de arrays son similares a los arrays indexados, pero en lugar de un almacenamiento lineal con indexes, cada valor puede asignarse con una clave de cadena definida por el usuario. Se pueden declarar arrays asociativos de las siguientes formas:



```
FOLDERS
└─ diplomado
  └─ arrays
    └─ index.php
      └─ ciclos
        └─ condicionales
          └─ holamundo
            └─ operadores
              └─ switchcase
                └─ tiposdatos
                  └─ variables

index.php
1  <?php
2
3  // Arrays
4
5  $edades = array('Diego' => 33, 'Andres' => 25, 'Juan' => 16);
6
7  $edades['Diego'] = 33;
8  $edades['Andres'] = 25;
9  $edades['Juan'] = 16;
10
11  ?>
```

Se debe tener en cuenta, al igual que los arrays indexados, que cada par de clave – elemento debe ir separado por una coma entre cada par; además la instrucción “=>” permite realizar la asignación directamente del elemento sobre la clave.

La forma de mostrar los datos directamente es muy simple, se acceden a estos a partir de las claves asignadas automáticamente y de esta forma se obtiene el dato sobre el array.

```

FOLDERS
└─ diplomado
  └─ arrays
    └─ index.php
      └─ ciclos
      └─ condicionales
      └─ holamundo
      └─ operadores
      └─ switchcase
      └─ tiposdatos
      └─ variables

index.php
1  <?php
2
3  // Arrays
4
5  $fechasNacimiento = array('Diego' => '1999-03-12',
6                             'Andres' => '2000-01-05',
7                             'Juan' => '1995-07-05');
8
9  echo $fechasNacimiento['Diego'] . "<br>";
10 echo $fechasNacimiento['Andres'] . "<br>";
11 echo $fechasNacimiento['Juan'] . "<br>";
12
13 ?>

```

localhost/diplomado/arrays/index.php

1999-03-12
2000-01-05
1995-07-05

```

FOLDERS
└─ diplomado
  └─ arrays
    └─ index.php
      └─ ciclos
      └─ condicionales
      └─ holamundo
      └─ operadores
      └─ switchcase
      └─ tiposdatos
      └─ variables

index.php
1  <?php
2
3  // Arrays
4
5  $notas = array('Diego' => 4.5, 'Juan' => 5, 'Alex' => 4);
6
7  echo $notas['Diego'] . "<br>";
8  echo $notas['Juan'] . "<br>";
9  echo $notas['Alex'] . "<br>";
10
11 ?>

```

localhost/diplomado/arrays/index.php

4.5
5
4

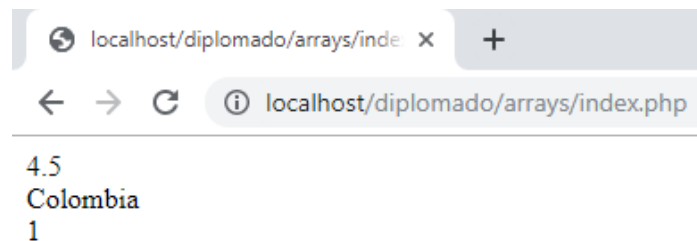
Estructura de un array asociativo

- Los elementos de un array asociativo siempre tienen asociados una clave, y permiten gestionar desde un solo elemento hasta múltiples de diferentes tipos.



```
FOLDERS
└─ diplomado
   └─ arrays
      └─ index.php
         └─ ciclos
            └─ condicionales
               └─ holamundo
                  └─ operadores
                     └─ switchcase
                        └─ tiposdatos
                           └─ variables
```

```
1 <?php
2
3 // Arrays
4
5 $notas = array('Diego' => 4.5, 'Juan' => "Colombia", 'Alex' => true);
6
7 echo $notas['Diego'] . "<br>";
8 echo $notas['Juan'] . "<br>";
9 echo $notas['Alex'] . "<br>";
10
11 ?>
```

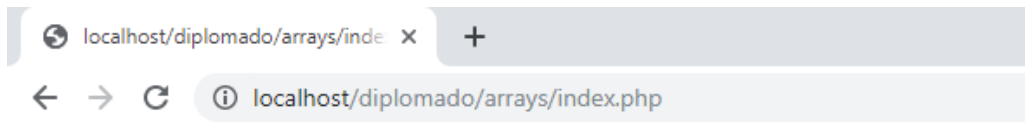


- Los arrays asociativos no pueden ser accedidos por medio de indexes, únicamente por medio de las claves.



```
FOLDERS
└─ diplomado
   └─ arrays
      └─ index.php
         └─ ciclos
            └─ condicionales
               └─ holamundo
                  └─ operadores
                     └─ switchcase
                        └─ tiposdatos
                           └─ variables
```

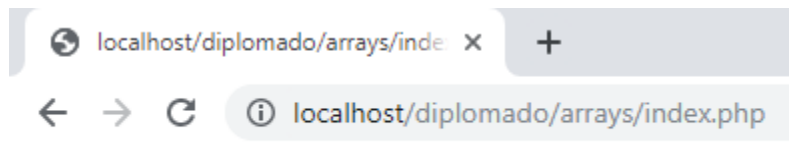
```
1 <?php
2
3 // Arrays
4
5 $array = array('Diego' => 4.5, 'Juan' => "Colombia", 'Alex' => true);
6
7 echo $array['Diego'] . "<br>";
8 echo $array['Juan'] . "<br>";
9 echo $array['Alex'] . "<br>";
10
11 ?>
```

Notice: Undefined offset: 1 in C:\xampp\htdocs\diplomado\arrays\index.php on line 7

Colombia
1

- Los array asociativos pueden ser de N posiciones, siendo N cualquier número entero.



Tamaño del array: 15

- Los arrays asociativos no permiten repetir claves en su declaración, al encontrar claves repetidas, el valor que se almacena dentro de la clave es el último que tomó la misma.

```

FOLDERS
└─ diplomado
  └─ arrays
    └─ index.php
  └─ ciclos
  └─ condicionales
  └─ holamundo
  └─ operadores
  └─ switchcase
  └─ tiposdatos
  └─ variables

index.php
1 <?php
2
3 // Arrays
4
5 $array = array('Diego' => 4.5, 'Juan' => "Colombia", 'Alex' => true,
6               'Diego' => 4.5, 'Juan' => "Colombia", 'Alex' => true,
7               'Diego' => 4.5, 'Juan' => "Colombia", 'Alex' => true,
8               'Diego' => 4.5, 'Juan' => "Colombia", 'Alex' => true,
9               'Diego' => 4.5, 'Juan' => "Colombia", 'Alex' => true,
10              'Diego' => 4.5, 'Juan' => "Colombia", 'Alex' => true,
11              'Diego' => 4.5, 'Juan' => "Colombia", 'Alex' => true,
12              'Diego' => 4.5, 'Juan' => "Colombia", 'Alex' => true,
13              'Diego' => 4.2, 'Juan' => "Colombia", 'Alex' => true);
14
15 var_dump($array);
16
17 ?>

```

```

localhost/diplomado/arrays/index.php
localhost/diplomado/arrays/index.php

array(3) { ["Diego"]=> float(4.2) ["Juan"]=> string(8) "Colombia" ["Alex"]=> bool(true) }

```

En este caso, la clave "Diego" que se repite N cantidad de veces en el array cuenta con una concurrencia de asignar el valor 4.5 a su clave, pero al encontrar en repetidas ocasiones, únicamente tomó el valor de la última asignación 4.2.

- Los arrays de PHP pueden contener claves int y string al mismo tiempo, estos arrays se conocen como arrays asociativos de tipos mixtos.

```

FOLDERS
└─ diplomado
  └─ arrays
    └─ index.php
  └─ ciclos
  └─ condicionales
  └─ holamundo
  └─ operadores
  └─ switchcase
  └─ tiposdatos
  └─ variables

index.php
1 <?php
2
3 // Arrays
4
5 $array = array("Pais" => "Argentina", 5325 => "Colombia", 2522 => "Chile");
6
7 var_dump($array);
8
9 ?>

```

```
localhost/diplomado/arrays/index.php
localhost/diplomado/arrays/index.php
array(3) { ["Pais"]=> string(9) "Argentina" [5325]=> string(8) "Colombia" [2522]=> string(5) "Chile" }
```

- La clave es opcional. Si no se especifica, PHP usará el incremento de la clave de tipo int mayor utilizada anteriormente. Es posible especificar la clave sólo para algunos elementos y excluir a los demás:



```
index.php
1 <?php
2
3 // Arrays
4
5 $array = array("a", "b", "letras " => "c", "d");
6
7 var_dump($array);
8
9 ?>
```

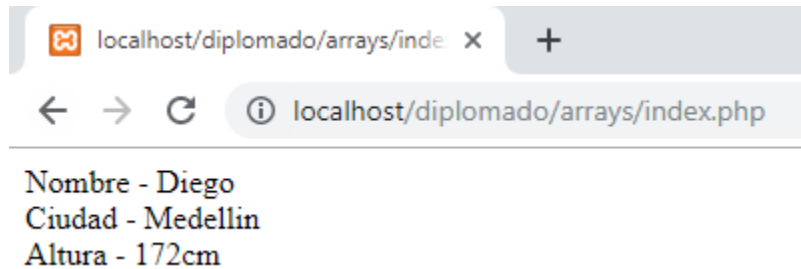
```
localhost/diplomado/arrays/index.php
localhost/diplomado/arrays/index.php
array(4) { [0] => string(1) "a" [1] => string(1) "b" ["letras "] => string(1) "c" [2] => string(1) "d" }
```

Como se puede ver, al último valor "d" se le asignó la clave 2. Esto es debido a que la mayor clave int anterior era 1.

Recorrido: Se puede recorrer un array asociativo usando el ciclo **foreach**.

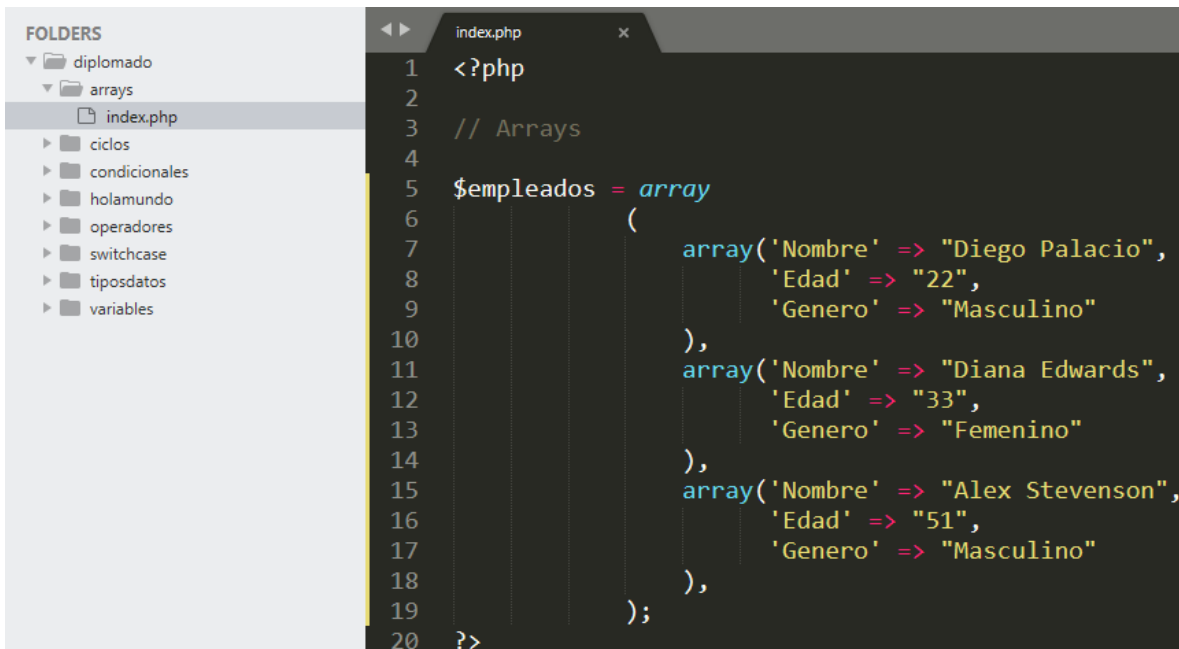


```
1 <?php
2
3 // Arrays
4
5 $persona = array("Nombre" => "Diego", "Ciudad" => "Medellin", "Altura" => "172cm");
6
7 foreach ($persona as $key => $value)
8 {
9     echo $key . " - " . $value . '<br>';
10 }
11
12 ?>
```



Arrays multidimensionales

Los arrays multidimensionales son arrays que almacenan otro array en cada índice en lugar de un solo elemento. En otras palabras, se puede definir arrays multidimensionales como un array de arrays. Como su nombre indica, cada elemento de este array puede ser un array y también pueden contener otros sub-arrays dentro. Se puede acceder a los arrays o sub-arrays en arrays multidimensionales utilizando múltiples dimensiones.



```
1 <?php
2
3 // Arrays
4
5 $empleados = array
6 (
7     array('Nombre' => "Diego Palacio",
8         'Edad' => "22",
9         'Genero' => "Masculino"
10    ),
11    array('Nombre' => "Diana Edwards",
12        'Edad' => "33",
13        'Genero' => "Femenino"
14    ),
15    array('Nombre' => "Alex Stevenson",
16        'Edad' => "51",
17        'Genero' => "Masculino"
18    ),
19 );
20 ?>
```

Se debe tener en cuenta, que se debe declarar el array multidimensional con la palabra array internamente en el array principal, y preferiblemente asignado claves a los elementos que se desean almacenar. Los arrays internos pueden o no contener la misma estructura y cantidad de elementos.

```

FOLDERS
└─ diplomado
  └─ arrays
    └─ index.php
      └─ ciclos
      └─ condicionales
      └─ holamundo
      └─ operadores
      └─ switchcase
      └─ tiposdatos
      └─ variables

index.php
1  <?php
2
3  // Arrays
4
5  $empleados = array
6      (
7      array('Nombre' => "Diego Palacio",
8          'Edad' => "22",
9          'Corre' => "diego@gmail"
10         ),
11      array('Nombre' => "Diana Edwards",
12          'Altura' => "185cm",
13          'Genero' => "Femenino"
14         ),
15      array('Nombre' => "Alex Stevenson",
16          'Genero' => "Masculino"
17         ),
18      );
19
20  ?>

```

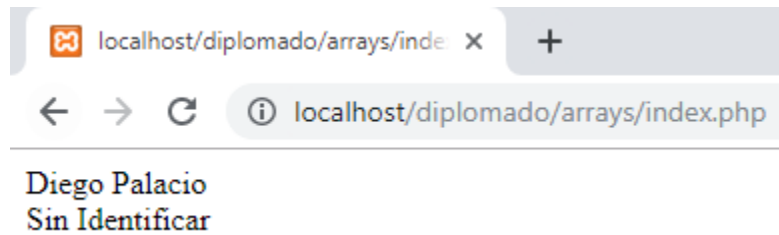
La forma de mostrar los datos directamente es muy simple, se acceden a estos a partir de dos factores, primero, el índice en el que se encuentra el array interno y segundo, la clave asignada al elemento, de esta forma se obtiene el dato sobre el array interno y los elementos.

```

FOLDERS
└─ diplomado
  └─ arrays
    └─ index.php
      └─ ciclos
      └─ condicionales
      └─ holamundo
      └─ operadores
      └─ switchcase
      └─ tiposdatos
      └─ variables

index.php
1  <?php
2
3  // Arrays
4
5  $empleados = array
6      (
7      array('Nombre' => "Diego Palacio",
8          'Edad' => "22",
9          'Corre' => "diego@gmail"
10         ),
11      array('Nombre' => "Diana Edwards",
12          'Altura' => "185cm",
13          'Genero' => "Femenino"
14         ),
15      array('Nombre' => "Alex Stevenson",
16          'Genero' => "Sin Identificar"
17         ),
18      );
19
20  echo $empleados[0]['Nombre'] . '<br>';
21  echo $empleados[2]['Genero'] . '<br>';
22
23  ?>

```



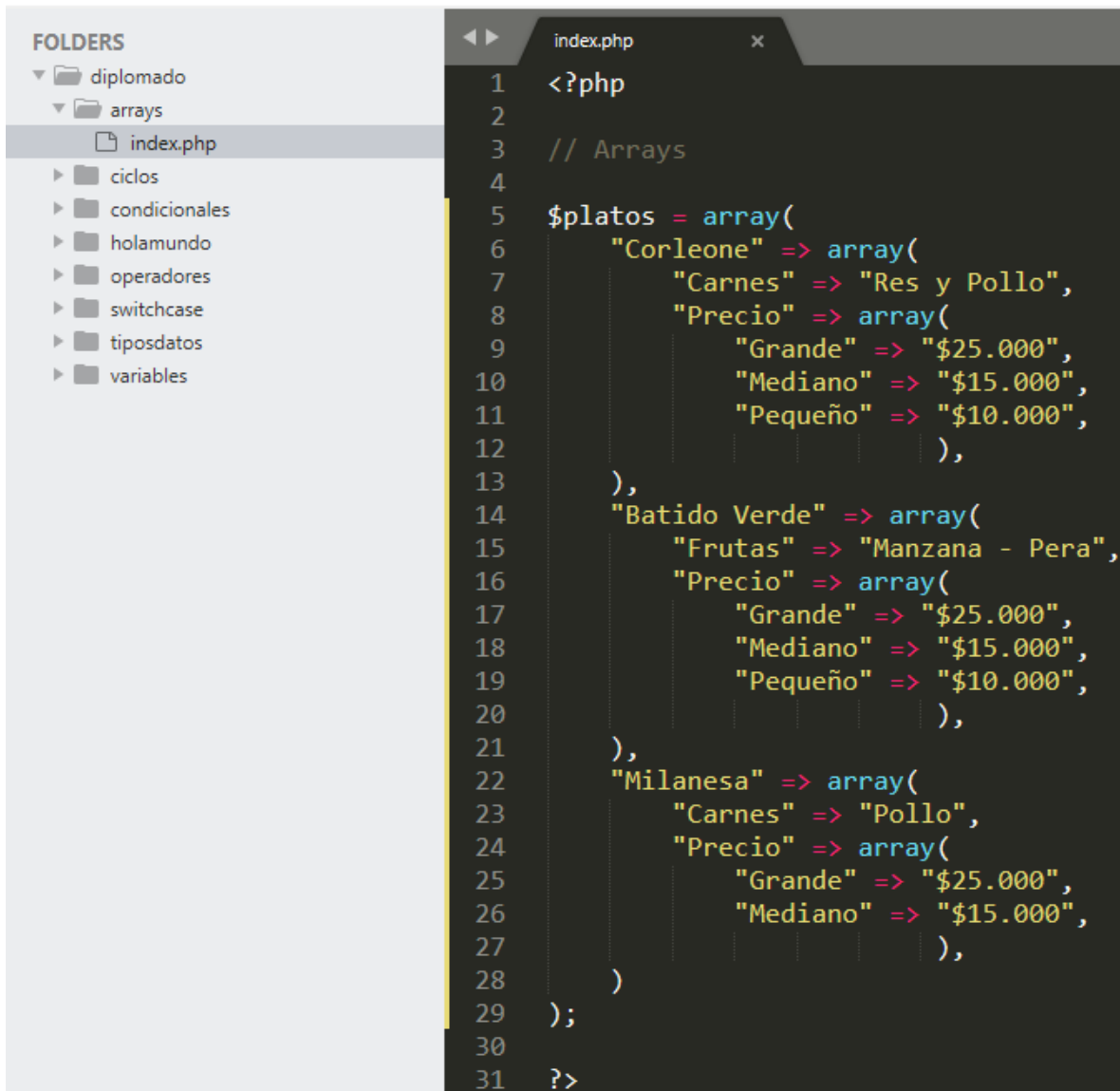
En este caso el array multidimensional cuenta con 3 arrays internos indexados, con las respectivas posiciones asignadas automáticamente (0,1,2) y estos arrays internos son asociativos, con los elementos almacenados por clave. Para acceder a esto es necesario pasar dos parámetros:

```
echo $empleados[0]['Nombre'] . '<br>';  
echo $empleados[2]['Genero'] . '<br>';
```

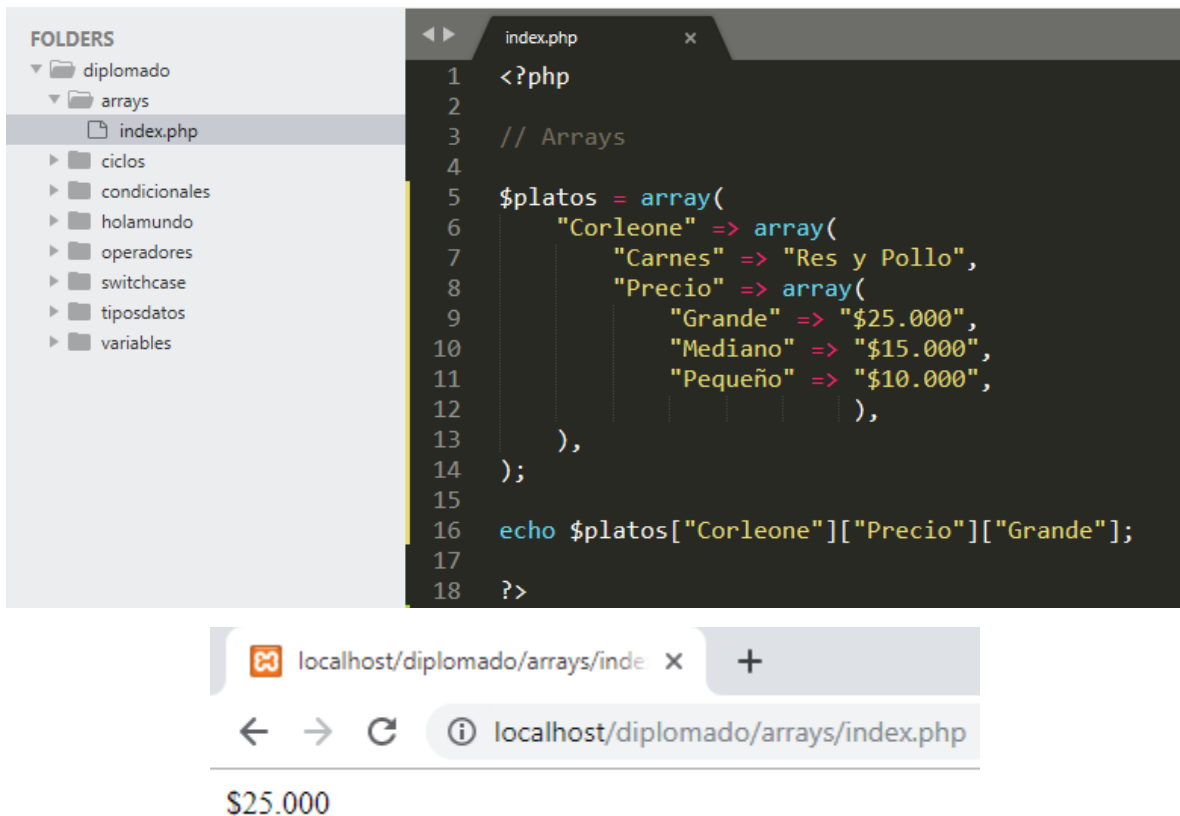
Uno que termina que array interno se va a manipular y otro que indica la clave que se desea consultar.

Estructura de los arrays multidimensionales

- PHP comprende arrays multidimensionales que tienen dos, tres, cuatro, cinco o más niveles de profundidad. Sin embargo, los arrays de más de tres niveles de profundidad son difíciles de administrar.



```
1 <?php
2
3 // Arrays
4
5 $platos = array(
6     "Corleone" => array(
7         "Carnes" => "Res y Pollo",
8         "Precio" => array(
9             "Grande" => "$25.000",
10            "Mediano" => "$15.000",
11            "Pequeño" => "$10.000",
12        ),
13    ),
14    "Batido Verde" => array(
15        "Frutas" => "Manzana - Pera",
16        "Precio" => array(
17            "Grande" => "$25.000",
18            "Mediano" => "$15.000",
19            "Pequeño" => "$10.000",
20        ),
21    ),
22    "Milanesa" => array(
23        "Carnes" => "Pollo",
24        "Precio" => array(
25            "Grande" => "$25.000",
26            "Mediano" => "$15.000",
27        ),
28    )
29 );
30
31 ?>
```

The screenshot shows a code editor with a file named `index.php` in a directory structure: `diplomado` > `arrays`. The PHP code defines a multidimensional array `$platos` and prints a specific value from it.

```
1 <?php
2
3 // Arrays
4
5 $platos = array(
6     "Corleone" => array(
7         "Carne" => "Res y Pollo",
8         "Precio" => array(
9             "Grande" => "$25.000",
10            "Mediano" => "$15.000",
11            "Pequeño" => "$10.000",
12        ),
13    ),
14 );
15
16 echo $platos["Corleone"]["Precio"]["Grande"];
17
18 ?>
```

Below the code editor, a browser window shows the output of the script: `localhost/diplomado/arrays/index.php` displays `$25.000`.

La dimensión de un array indica la cantidad de índices que necesita para seleccionar un elemento.

- Para un array bidimensional, necesita dos índices para seleccionar un elemento
 - Para un array tridimensional, necesita tres índices para seleccionar un elemento
- Un array bidimensional es un array de arrays (un array tridimensional es un array de arrays de arrays).

Nombre	Stock	Precio
BMW S1000RR	7	80000000
Kawasaki Z650	20	22000000
Suzuki DR650	17	25000000
Yamaha Tracer 900	2	49000000

```

FOLDERS
└─ diplomado
  └─ arrays
    └─ index.php
      └─ ciclos
        └─ condicionales
          └─ holamundo
            └─ operadores
              └─ switchcase
                └─ tiposdatos
                  └─ variables

index.php
1 <?php
2
3 // Arrays
4
5 $motos = array(
6     array("Moto" => "BMW S1000RR", "Stock" => "7", "Precio" => "80000000"),
7     array("Moto" => "Kawasaki Z650", "Stock" => "20", "Precio" => "22000000"),
8     array("Moto" => "Suzuki DR650", "Stock" => "17", "Precio" => "25000000"),
9     array("Moto" => "Yamaha Tracer 900", "Stock" => "2", "Precio" => "49000000"),
10 );
11
12 ?>

```

Ahora el array bidimensional \$ motos contiene cuatro arrays, y tiene dos índices: fila y columna.

```

FOLDERS
└─ diplomado
  └─ arrays
    └─ index.php
      └─ ciclos
        └─ condicionales
          └─ holamundo
            └─ operadores
              └─ switchcase
                └─ tiposdatos
                  └─ variables

index.php
1 <?php
2
3 // Arrays
4
5 $motos = array(
6     array("Moto" => "BMW S1000RR", "Stock" => "7", "Precio" => "80000000"),
7     array("Moto" => "Kawasaki Z650", "Stock" => "20", "Precio" => "22000000"),
8     array("Moto" => "Suzuki DR650", "Stock" => "17", "Precio" => "25000000"),
9     array("Moto" => "Yamaha Tracer 900", "Stock" => "2", "Precio" => "49000000"),
10 );
11
12 echo $motos[0]["Moto"] . " - Stock: " . $motos[0]["Stock"] . " - Precio: " . $motos[0]["Precio"] . "<br>";
13 echo $motos[1]["Moto"] . " - Stock: " . $motos[1]["Stock"] . " - Precio: " . $motos[1]["Precio"] . "<br>";
14 echo $motos[2]["Moto"] . " - Stock: " . $motos[2]["Stock"] . " - Precio: " . $motos[2]["Precio"] . "<br>";
15 echo $motos[3]["Moto"] . " - Stock: " . $motos[3]["Stock"] . " - Precio: " . $motos[3]["Precio"] . "<br>";
16
17 ?>

```

- Los array multidimensional pueden ser de N posiciones (N dimensiones), siendo N cualquier número entero.

```

FOLDERS
└─ diplomado
  └─ arrays
    └─ index.php
      └─ ciclos
        └─ condicionales
          └─ holamundo
            └─ operadores
              └─ switchcase
                └─ tiposdatos
                  └─ variables

index.php
1 <?php
2
3 // Arrays
4
5 $motos = array(
6     array("Moto" => "BMW S1000RR",
7         array("Moto" => "Kawasaki Z650", "Stock" => "20", "Precio" => "220000000"), "Precio" => "800000000"),
8     array("Moto" => "Kawasaki Z650", "Stock" => "20", "Precio" => "220000000"),
9     array("Moto" => "Suzuki DR650", "Stock" => "17", array("Moto" => "Yamaha Tracer 900", array("Moto" => "
10         Yamaha Tracer 900", "Stock" => "2", "Precio" => "490000000", "Precio" => "490000000")),
11     array("Moto" => "Yamaha Tracer 900", "Stock" => "2", "Precio" => "490000000"),
12 );
13 ?>

```

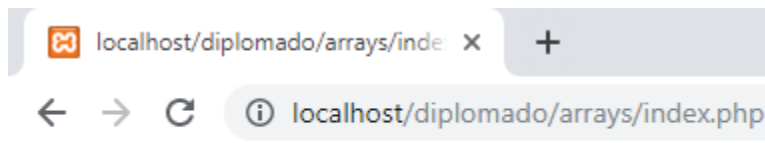
- Los arrays multidimensionales, al igual que los asociativos, no permiten repetir claves en su declaración, al encontrar claves repetidas, el valor que se almacena dentro de la clave es el último que tomó la misma.

```

FOLDERS
└─ diplomado
  └─ arrays
    └─ index.php
      └─ ciclos
        └─ condicionales
          └─ holamundo
            └─ operadores
              └─ switchcase
                └─ tiposdatos
                  └─ variables

index.php
1 <?php
2
3 // Arrays
4
5 $motos = array(
6     array("Moto" => "BMW S1000RR",
7         array("Moto" => "Kawasaki Z650", "Stock" => "20", "Precio" => "220000000"), "Precio" => "800000000"),
8     array("Moto" => "Kawasaki Z650", "Stock" => "20", "Precio" => "220000000"),
9     array("Moto" => "Suzuki DR650", "Stock" => "17", array("Moto" => "Yamaha Tracer 900", array("Moto" => "
10         Yamaha Tracer 900", "Stock" => "2", "Precio" => "490000000", "Precio" => "490000000")),
11     array("Moto" => "Yamaha Tracer 900", "Stock" => "2", "Precio" => "490000000"),
12 );
13 echo count($motos);
14
15 ?>

```



Total: 3

Funciones en arrays

PHP ofrece una gran gama de funciones sobre los arrays, desde para buscar elementos, hasta para cambiar a mayúsculas y minúsculas. Esta es una tabla con algunas de las funciones incluidas en la documentación oficial de PHP:

Función	Definición
array_change_key_case	Cambia a mayúsculas o minúsculas todas las claves en un array
array_column	Devuelve los valores de una sola columna del array de entrada
array_combine	Crea un nuevo array, usando una matriz para las claves y otra para sus valores
array_count_values	Cuenta todos los valores de un array
array_diff	Calcula la diferencia entre arrays
array_fill_keys	Llena un array con valores, especificando las keys
array_fill	Llena un array con valores
array_key_exists	Verifica si el índice o clave dada existe en el array
array_keys	Devuelve todas las claves de un array o un subconjunto de claves de un array
array_merge	Combina dos o más arrays
array_pop	Extrae el último elemento del final del array
array_push	Inserta uno o más elementos al final de un array
array_replace	Reemplaza los elementos del array original con elementos de array adicionales
array_search	Busca un valor determinado en un array y devuelve la primera clave correspondiente en caso de éxito
array_shift	Quita un elemento del principio del array
array_slice	Extraer una parte de un array
array_splice	Elimina una porción del array y la reemplaza con otra cosa
array_sum	Calcular la suma de los valores de un array
array_unique	Elimina valores duplicados de un array
array_unshift	Añadir al inicio de un array uno a más elementos
array_values	Devuelve todos los valores de un array
array_walk	Aplicar una función proporcionada por el usuario a cada miembro de un array
array	Crea un array
arsort	Ordena un array en orden inverso y mantiene la asociación de índices
asort	Ordena un array y mantiene la asociación de índices
count	Cuenta todos los elementos de un array o algo de un objeto
in_array	Comprueba si un valor existe en un array
key	Obtiene una clave de un array
krsort	Ordena un array por clave en orden inverso
ksort	Ordena un array por clave
list	Asignar variables como si fueran un array
natcasesort	Ordenar un array usando un algoritmo de "orden natural" insensible a mayúsculas-minúsculas
natsort	Ordena un array usando un algoritmo de "orden natural"
rsort	Ordena un array en orden inverso
shuffle	Mezcla un array
sizeof	Alias de count
sort	Ordena un array
uasort	Ordena un array con una función de comparación definida por el usuario y mantiene la asociación de índices
uksort	Ordena un array según sus claves usando una función de comparación definida por el usuario
usort	Ordena un array según sus valores usando una función de comparación definida por el usuario

Recursos disponibles para el aprendizaje

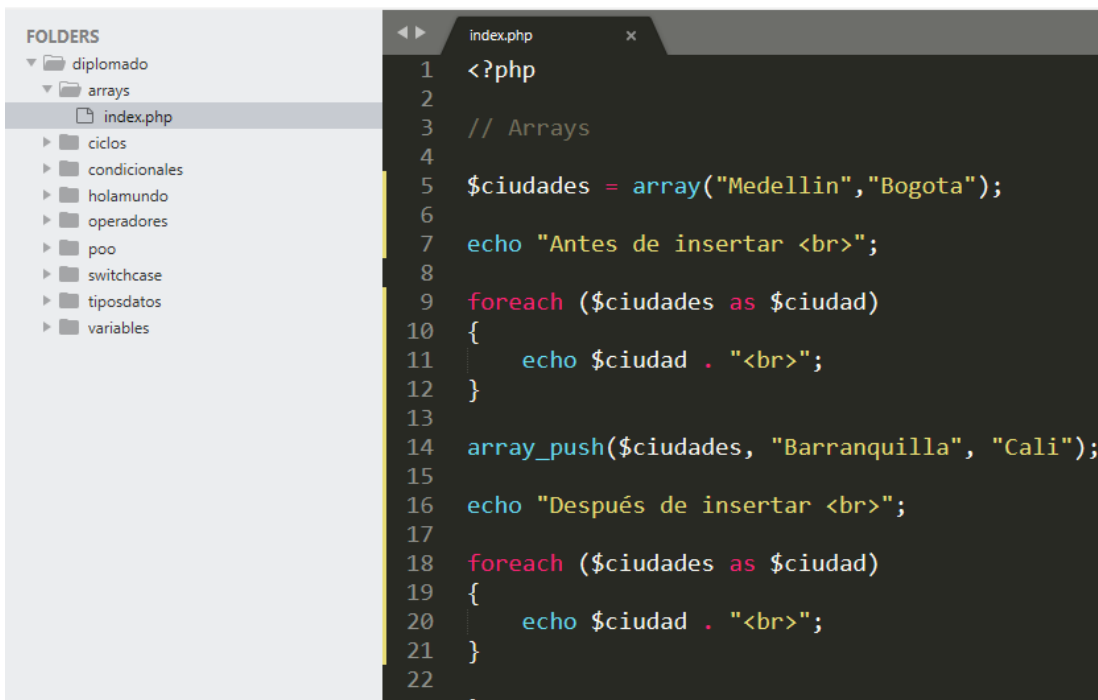


Para desarrollar las habilidades y destrezas necesarias en cada competencia, es muy importante que tengas acceso a los recursos didácticos adecuados.

Las funciones en los arrays permiten ahorrar demasiadas líneas de código, por esto te invito a consultar la tabla completa y cada uno de sus funciones y el cómo funcionan. Disponible en: <https://www.php.net/manual/es/ref.array.php>

Conociendo ahora algunas de las funciones, pongamos en práctica alguna de ellas:

- **Array_push():** esta función permite insertar uno o más elementos al final de un array. Dentro del paréntesis, recibe array al que se desea realizar la inserción y el o los elementos a insertar.



```
1 <?php
2
3 // Arrays
4
5 $ciudades = array("Medellin","Bogota");
6
7 echo "Antes de insertar <br>";
8
9 foreach ($ciudades as $ciudad)
10 {
11     echo $ciudad . "<br>";
12 }
13
14 array_push($ciudades, "Barranquilla", "Cali");
15
16 echo "Después de insertar <br>";
17
18 foreach ($ciudades as $ciudad)
19 {
20     echo $ciudad . "<br>";
21 }
22
23 }
```

```
localhost/diplomado/arrays/index.php
```

← → ↻ ⓘ localhost/diplomado/arrays/index.php

```

Antes de insertar
Medellin
Bogota
Después de insertar
Medellin
Bogota
Barranquilla
Cali
  
```

- **Array_pop():** esta función permite extraer de un array el último elemento, entiéndase extraer como eliminar. Dentro de los paréntesis recibe el array al que se desea extraer el dato.

FOLDERS

- diplomado
 - arrays
 - index.php
 - ciclos
 - condicionales
 - holamundo
 - operadores
 - poo
 - switchcase
 - tiposdatos
 - variables

```

1 <?php
2
3 // Arrays
4
5 $ciudades = array("Medellin","Bogota","Barranquilla");
6
7 echo "El último elemento es: " . array_pop($ciudades) . "<br>";
8
9 foreach ($ciudades as $ciudad)
10 {
11     echo $ciudad . "<br>";
12 }
13
14 ?>
  
```

```
localhost/diplomado/arrays/index.php
```

← → ↻ ⓘ localhost/diplomado/arrays/index.php

```

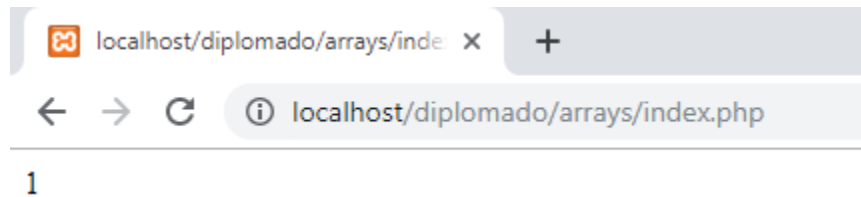
El último elemento es: Barranquilla
Medellin
Bogota
  
```

- **Array_search():** Esta función buscar un elemento determina en un array y devolver la clave donde se encuentra. Dentro de los paréntesis recibe el valor a buscar y el array donde se realizará la búsqueda.

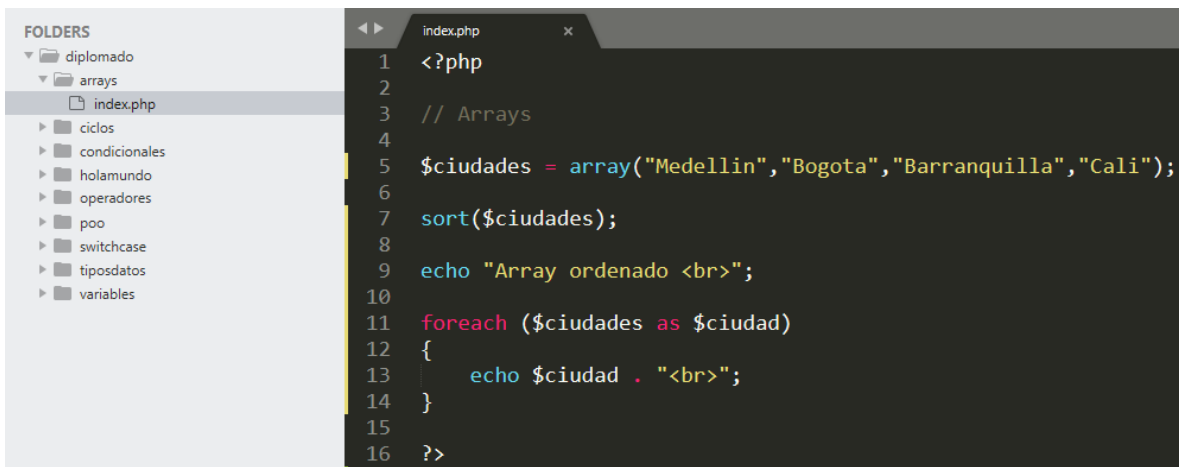


```
FOLDERS
└─ diplomado
  └─ arrays
    └─ index.php
      └─ ciclos
      └─ condicionales
      └─ holamundo
      └─ operadores
      └─ poo
      └─ switchcase
      └─ tiposdatos

index.php
1 <?php
2
3 // Arrays
4
5 $ciudades = array("Medellin","Bogota","Barranquilla","Cali");
6
7 echo array_search("Bogota", $ciudades);
8
9 ?>
```

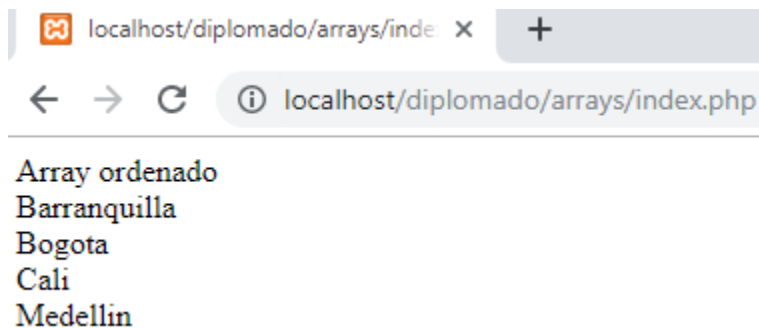


- **Sort():** esta función permite ordenar un array. Dentro de los paréntesis recibe el array que se desea ordenar:



```
FOLDERS
└─ diplomado
  └─ arrays
    └─ index.php
      └─ ciclos
      └─ condicionales
      └─ holamundo
      └─ operadores
      └─ poo
      └─ switchcase
      └─ tiposdatos
      └─ variables

index.php
1 <?php
2
3 // Arrays
4
5 $ciudades = array("Medellin","Bogota","Barranquilla","Cali");
6
7 sort($ciudades);
8
9 echo "Array ordenado <br>";
10
11 foreach ($ciudades as $ciudad)
12 {
13     echo $ciudad . "<br>";
14 }
15
16 ?>
```



Recursos disponibles para el aprendizaje



Los arrays son un concepto muy importante y en especial en la programación con PHP, en la documentación del lenguaje hay información muy importante y que te puede ser útil, visítala, Disponible en: <https://www.php.net/manual/en/control-structures.while.php>

Ejercicio

¿Deseas profundizar en la temática de los arrays? Entonces te sugiero realizar los siguientes ejercicios que pondrán a prueba los conocimientos adquiridos.
(MÓDULO 2 – EJERCICIOS DE ARRAYS)
¡Inténtalo! 👍



Recursos disponibles para el aprendizaje



Para desarrollar las habilidades y destrezas necesarias en cada competencia, es muy importante que tengas acceso a los recursos didácticos adecuados.

Entonces, si necesitas reforzar esta información, te sugerimos revisar nuevamente los **Vídeos de Apoyos**, disponibles en el campus virtual, indicados en las lecturas anteriores. Además, recuerda que puede consultar las **Fuentes Documentales** que aparecen en esta guía, particularmente, en el apartado de Referencias Bibliográficas.

MATERIAL COMPLEMENTARIO

Es importante continuar adquiriendo conocimiento y no frenar el proceso de aprendizaje, por esto es importante complementar lo aprendido en esta guía con nuevos conceptos, definiciones y característica, sugerimos revisar el siguiente material:

En internet hay demasiada información y documentación sobre PHP, si deseas obtener más información y leer a parte de este diplomado te sugiero los siguientes sitios:

<http://www.mclibre.org/consultar/php/index.html> -

<https://www.tutorialesprogramacionya.com/phpya/> -

<https://www.w3schools.com/php/default.asp>

Siempre práctica y realiza ejercicios. Busca más ejercicios referentes a los conceptos que has vistos en este módulo, estos estarán presente en toda la programación.

ASPECTOS CLAVES

Recuerda tener muy presente los conceptos visto en esta guía número 2 dado que en el trascurso del diplomado se tendrán en cuenta continuamente para su implementación.

Recuerda algunos aspectos abordados en el módulo:

- Las condicionales son una estructura de control y para estas se tiene en cuenta los conceptos de los operadores.
- Los ciclos permiten repetir diversos bloques de código
- Recuerda que existen 4 tipos de ciclos. Foreach, for, while, do while.
- Los arrays son una estructura muy interesante que cuenta con una gran cantidad de funciones para trabajar con estos, visita la documentación y lee todas las que dispone el lenguaje.
- No olvides realizar todos los ejercicios.

¡Felicidades! 🍀 Has concluido con la lectura de la Guía Didáctica N°2. Así que ya puedes realizar la Evaluación 2.

Esta guía fue elaborada para ser utilizada con fines didácticos como material de consulta de los participantes en el Diplomado Virtual en Programación en Java del Politécnico de Colombia, especialmente, a los técnicos, tecnólogos y profesionales de carreras afines, estudiantes de todas las carreras, empíricos, y público en general con conocimientos básicos en informática que intentan entrar en el mundo de la programación, que se desempeñen o no en las áreas de TIC de cualquier tipo de organización y que deseen obtener las competencias y habilidades necesarias para conocer los fundamentos prácticos del lenguaje de programación Java para la aplicación y desarrollo de algoritmos y aplicaciones, y solo podrá ser reproducida con esos fines. Por lo tanto, se agradece a los usuarios referirla en los escritos donde se utilice la información que aquí se presenta.

Derechos reservados - POLITÉCNICO DE COLOMBIA, 2019
Medellín, Colombia