# Tipos de datos

## Cadenas

En PHP, las cadenas de texto se delimitan por comillas (dobles o simples).

```
<?php
  echo "<p>Esto es una cadena.</p>";
?>
```

```
<p>Esto es una cadena.</p>
```

```
<?php
   echo '<p>Esto es una cadena.</p>';
?>
```

```
<p>Esto es una cadena.</p>
```

Las cadenas contienen lógicamente caracteres del juego de caracteres en el que se guarda el archivo (UTF-8, CP-1252, ISO-8859-1, etc.), pero algunos caracteres necesitan un tratamiento especial, como se explicará más adelante.
Si las comillas de apertura y cierre no son del mismo tipo (una de ellas doble y otra simple), se produce un error de sintaxis.

```
<?php
   echo "<p>Esto es una cadena.</p>';
?>
```

```
Parse error: syntax error, unexpected end of file, expecting
variable (T_VARIABLE) or ${ (T_DOLLAR_OPEN_CURLY_BRACES)
or {$ (T_CURLY_OPEN) in ejemplo.php on line 2
```

### Comillas dentro de cadenas

Si una cadena está delimitada por comillas dobles, en su interior puede haber cualquier número de comillas simples, y viceversa.

```
<?php
   print "<p>Esto es comilla simple: '</p>";
?>
```

```
<p>Esto es una comilla simple: '</p>
```

```
<?php
   echo '<p>Esto es una comilla doble: "</p>';
?>
```

```
<p>Esto es una comilla doble: "</p>
```

Lo que no puede haber en una cadena es una comilla del mismo tipo que las que delimitan la cadena.

```
<?php
   echo "<p>Esto es una comilla doble: "</p>";
?>
```

```
Parse error: syntax error, unexpected '/' in ejemplo.php on line 2
```

```
<?php
   echo '<p>Esto es una comilla simple: '</p>';
?>
```

```
Parse error: syntax error, unexpected '/' in ejemplo.php on line 2
```

Para poder escribir en una cadena una comilla del mismo tipo que las que delimitan la cadena, se debe utilizar los caracteres especiales \' o \".

```
<?php
    echo "<p>Esto es una comilla simple: ' y esto una comilla
doble: \"</p>";
?>
```

```
<p>Esto es una comilla simple: ' y esto una comilla doble: "</p>
```

```
<?php
     echo '<p>Esto es una comilla simple: \' y esto una comilla
doble: "</p>';
?>
```

```
<p>Esto es una comilla simple: ' y esto una comilla doble: "</p>
```

Pero los caracteres especiales \" y \' no se pueden utilizar para delimitar cadenas

```
<?php
   print \"<p>Esto es una cadena.</p>";
?>
```

```
Parse error: syntax error, unexpected '"<p>Esto es una cadena.</p>"' (T_CONSTANT_ENCAPSED_STRING), expecting identifier (T_STRING) in ejemplo.php on line 2
```

```
<?php
   print \'<p>Esto es una cadena.</p>';
?>
```

```
Parse error: syntax error, unexpected ''<p>Esto es una cadena.</p>' (T_CONSTANT_ENCAPSED_STRING), expecting identifier (T_STRING) in ejemplo.php on line 2
```

## Diferencias entre comillas simples y dobles

Aunque en los ejemplos anteriores las comillas simples o dobles son equivalentes, en otras situaciones no lo son. Por ejemplo, PHP no sustituye las variables que se encuentran dentro de cadenas delimitadas con comillas simples, mientras que sí que lo hace (pero no siempre) si se utilizan comillas dobles, como se ve en el siguiente ejemplo:

```
<?php
  $cadena = "Hola";
  echo "<p>La variable contiene el valor: $cadena</p>";
?>
```

```
<p>La variable contiene el valor: Hola</p>
```

```
<?php
  $cadena = "Hola";
  echo '<p>La variable contiene el valor: $cadena</p>';
?>
```

```
<p>La variable contiene el valor: $cadena</p>
```

### Comillas en código html / css

En el código HTML generado con PHP también se pueden escribir comillas simples o dobles. Los dos ejemplos siguientes producen código html válido:

```
<?php
  echo "<p><strong style='color: red;'>Hola</strong></p>";
?>
```

```
<p<strong style='color: red;'>Hola</strong></p>
```

```
<?php
  echo '<p><strong style="color: red;">Hola</strong></p>';
?>
```

```
<p<strong style="color: red;">Hola</strong></p>
```

## Caracteres especiales

Por distintos motivos, algunos caracteres necesitan escribirse dentro de las cadenas de una forma especial. Por ejemplo:

- En PHP los nombres de variables empiezan por el carácter dólar ($). Cuando en una cadena aparece una palabra que empieza por el carácter $, PHP entiende que esa palabra hace referencia a una variable. Pero si a esa variable no se le ha dado valor anteriormente, se producirá un aviso. Además PHP escribe el valor de la variable (es decir, nada si la variable no está definida):

```
<?php
   echo "<p>Los nombres de variables empiezan por $. Por ejemplo
$edad.</p>";
?>
```

```
Notice: Undefined variable: edad in ejemplo.php on line 2
```

Para poder escribir el carácter $ se tiene que anteponer el carácter contrabarra (\). El ejemplo anterior quedaría por tanto.

```
<?php
    echo "<p>Los nombres de variables empiezan por $. Por ejemplo
\$edad.</p>";
?>
```

```
<p>Los nombres de variables empiezan por $. Por ejemplo $edad.</p>
```

## Concatenar cadenas

El operador . (punto) permite concatenar dos o más cadenas.

```
<?php
   print "<p>Pasa" . "tiempos</p>";
?>
```

```
<p>Pasatiempos</p>
```

El mismo resultado se puede conseguir sin utilizar el operador . (punto):

```
<?php
   print "<p>Pasatiempos</p>";
?>
```

```
<p>Pasatiempos</p>
```

El operador de asignación con concatenación (.=) permite concatenar una cadena a otra y asignarla a esta:

```
<?php
   $cadena = "Pasa";
   print "<p>$cadena</p>";
   $cadena .= "tiempos";
   print "<p>$cadena</p>";
?>
```

```
<p>Pasa</p>
<p>Pasatiempos</p>
```

## Añadiendo saltos de línea

En la red verás miles de ejemplos donde se usa **\n** para realizar saltos de línea. Si lo utilizas podrías encontrarte problemas de compatibilidad entre sistemas operativos ya que la forma de realizar un salto de línea varía en cada uno. **PHP_EOL** te hace la vida más fácil ya que selecciona automáticamente el correcto.

```
<?php
```

```
    echo '<p>En un lugar de la mancha,</p>' . PHP_EOL . '<p>de
cuyo nombre no quiero acordarme...</p>';
?>
```

```
<p>En un lugar de la mancha,</p>
<p>de cuyo nombre no quiero acordarme...</p>
```

# Variables

## ¿Qué es una variable?

En los lenguajes de programación, una variable es un elemento que permite almacenar información. Las variables se identifican por su nombre. Para facilitar la comprensión del programa, se aconseja que los nombres de las variables hagan referencia a la información que contienen. Por ejemplo, si se va a guardar en una variable la edad del usuario, un nombre adecuado de la variable sería por ejemplo $edad.
En PHP el programador puede dar el nombre que quiera a las variables, con algunas restricciones:

- Los nombres de las variables tienen que empezar por el carácter $.
- A continuación tiene que haber una letra (mayúscula o minúscula) o un guión bajo (_).
- El resto de caracteres del nombre pueden ser números, letras o guiones bajos.

```
<?php
   // Ejemplos de nombres de variables:
   $edad   = 16;
   $edad1  = 17;
   $edad_1 = 18;
   // Ejemplos de nombres de variables (estilos NO redomendados)
   $años = 19;
   $_edad  = 20;
   $Edad  = 21;
   $__edad = 22;
   $EDAD   = 23;
   $_EDAD  = 24;
?>
```

Los nombres de variables pueden contener caracteres no ingleses (vocales acentuadas, eñes (ñ) o cedillas (ç), etc.) pero se aconseja no hacerlo. Si los programas que escribimos los van a leer solamente programadores de países en los que también se utilizan estos caracteres, no habrá inconveniente, pero si colaboran con nosotros programadores de países en los que esos caracteres no se utilizan, les resultará incómodo editar los programas. De hecho, algunos aconsejan incluso utilizar términos ingleses en los nombres de variables para facilitar la colaboración entre programadores de distintos países.

En los nombres de las variables, PHP distingue entre mayúsculas y minúsculas, es decir, si se cambia algún carácter de mayúscula a minúscula o viceversa, para PHP se tratará de variables distintas.
Si el nombre de la variable contiene varias palabras, se aconseja utilizar la notación "camel case", es decir, escribir la primera palabra en minúsculas y la primera letra de las siguientes palabras en mayúsculas.

```
<?php
   // Ejemplos de nombres de variables Camel Case:
   $edadHijo          = 16;       // edad del hijo
   $edadMadre         = 47;       // edad de la madre
   $edadPadre         = 46;       // edad del padre
   $edadAbueloPaterno = 73;       // edad del abuelo paterno
?>
```

PHP define automáticamente una serie de variables (son las llamadas variables predefinidas). Los nombres de estas variables siguen siempre el mismo patrón: empiezan por un guión bajo y se escriben en mayúsculas (por ejemplo, $_REQUEST, $_SERVER, $_SESSION, etc.). Para evitar conflictos con variables predefinidas que se creen en el futuro, se recomienda no crear en los programas variables con nombres que sigan ese mismo patrón.

El identificador de variable siempre debe empezar por **$**. Esta es una peculiaridad de PHP que al principio descoloca un poco.

En PHP, no es necesario declarar las variables: al inicializarlas queda especificado el tipo.

```
<?php
   $a = 4;              // Variable entera (decimal
   $b = 017;            // Variable entera (octal)
   $c = 0x12;           // Variable entera (hexadecimal)
   $media = 52.75;           // Variable real
   $texto = "Hoy es lunes"; // Variable string
?>
```

Cualquier variable puede cambiarse de tipo con funciones como **intval(), floatval()** o **strval()**:

```
<?php
  $a = "10";           // $a es una cadena
  $b = intval($a);     // $a se convierte a entero y se asigna a
$b
?>
```

Como no hay que declarar las variables, a veces no estaremos seguros de si una variable existe y tiene un valor válido (no nulo) asignado. Para averiguarlo existe la función **isset()**, que nos devuelve true si la variable existe y false en caso contrario. Del mismo modo, hay otra función muy útil, **unset()**, que hace desaparecer a una variable ya definida y libera la memoria que ocupaba:

```
<?php
  if (isset($nombre)) {
    echo $nombre;    // Solo muestra el nombre si la variable
tiene algún valor
  } else {
    echo "El nombre no está definido";
  }
?>
```

Los tipos de datos predefinidos en PHP son:

- integer (entero)
- float (real)
- bool (booleano)
- string (cadena)
- array

Ya profundizaremos en ellos más adelante

## Usar variables

Para guardar un valor en una variable se utiliza el operador de asignación (=) escribiendo a la izquierda únicamente el nombre de la variable y a la derecha el valor que queremos guardar. Si queremos guardar un número, no hace falta poner comillas, pero si queremos guardar una cadena de texto hay que poner comillas (dobles o simples).

```
<?php
   $edad = 15;   // La variable $edad tiene ahora el valor 15
   $nombre = "Pepito Conejo";  // La variable $nombre tiene ahora
el valor Pepito Conejo
?>
```

El programa anterior es correcto, aunque no produciría ningún resultado visible para el usuario.

En el lado izquierdo de la asignación (=) no se puede escribir más que el nombre de una variable. El programa siguiente no es sintácticamente correcto por lo que no se ejecutará y PHP mostrará un mensaje de error.

```
<?php
   $edad + 1 = 16;
   echo $edad;
?>
```

```
Parse error: syntax error, unexpected '=' in ejemplo.php on line 2
```

Una vez se ha guardado un valor en una variable, se puede utilizar en el resto del programa.

```
<?php
  $edad = 15;
  echo $edad;
?>
```

```
15
```

A lo largo de un programa una variable puede guardar diferentes valores.

```
<?php
  $edad = 15;
  echo "<p>$edad</p>";
  $edad = 20;
  echo "<p>$edad</p>";
?>
```

```
<p>15</p>
<p>20</p>
```

En la parte derecha de la asignación se pueden escribir expresiones matemáticas:

```
<?php
  $radio = 15;
  $perimetro = 2 * 3.14 * 15;
  echo $perimetro;
?>
```

```
94.2
```

Esas expresiones pueden contener variables:

```
<?php
  $radio = 15;
  $perimetro = 2 * 3.14 * $radio;
  echo $perimetro;
?>
```

```
94.2
```

En PHP una igualdad no es una ecuación matemática, sino una asignación (el resultado de la derecha se almacena en la variable de la izquierda). Por ello, una asignación pueden utilizar en el lado derecho la variable en la que se va a guardar el resultado:

```
<?php
  $edad = 15;
  $edad = 2 * $edad;
  echo $edad;
?>
```

```
30
```

Lo que hace el ordenador es calcular el resultado de la expresión de la derecha utilizando el valor actual de la variable y finalmente guardar el resultado en la variable.

En los nombres de las variables, PHP distingue entre mayúsculas y minúsculas, es decir, si se cambia algún carácter de mayúscula a minúscula o viceversa, para PHP se tratará de variables distintas.

```
<?php
  $Edad = 15;
  $edad = 20;
  echo $Edad;
?>
```

```
15
```

Una variable puede tomar valores de tipos distintos a lo largo de un programa.

```
<?php
  $edad = 15;
  echo "<p>$edad</p>";
  $edad = "quince";
  echo "<p>$edad</p>";
?>
```

```
<p>15</p>
<p>quince</p>
```

## Concatenar variables y cadenas

El operador . (punto) permite concatenar dos o más cadenas o variables.

```
<?php
  $cadena1 = "Pasa";
  $cadena2 = "tiempos";
  $cadena3 = $cadena1 . $cadena2;
  echo "<p>$cadena3</p>";
?>
```

```
<p>Pasatiempos</p>
```

```
<?php
  $cadena1 = "Corre";
  $cadena2 = "ve";
  $cadena3 = "idile";
  $cadena4 = $cadena1 . $cadena2 . $cadena3;
  echo "<p>$cadena4</p>";
?>
```

```
<p>Correveidile</p>
```

El operador . (punto) se puede utilizar en la instrucción print. En el ejemplo siguiente se concatenan una cadena, una variable y una cadena.

```
<?php
  $nombre = "Don Pepito";
  print "<p>¡Hola, " . $nombre . "! ¿Cómo está usted?</p>";
?>
```

```
<p>¡Hola, Don Pepito! ¿Cómo está usted?</p>
```

**Nota**: En el ejemplo anterior, se puede obtener el mismo resultado sin utilizar el operador . (punto), escribiendo la variable en la cadena:

```
<?php
  $nombre = "Don Pepito";
  echo "<p>¡Hola, $nombre! ¿Cómo está usted?</p>";
?>
```

```
<p>¡Hola, Don Pepito! ¿Cómo está usted?</p>
```

En el ejemplo siguiente no queda más remedio que utilizar el operador . (punto) porque uno de los datos que se intercala es el resultado de una operación.

```
<?php
 $semanas = 5;
  print "<p>$semanas semanas son " . (7 * $semanas) . "
días.</p>";
?>
```

```
<p>5 semanas son 35 días.</p>
```

## Variables en cadenas

La forma de insertar variables dentro de cadenas depende del tipo de variable utilizado.

### Tipos de variables

En el caso de números, cadenas o matrices de una dimensión, las variables se puede insertar directamente:

```
<?php
  $numero = 5000;
  $texto = "cinco mil";
  $seEscribe = ["separado", "junto"];
  print "<p>El número $numero se escribe $seEscribe[0]:
$texto</p>";
?>
```

```
<p>El número 5000 se escribe separado: cinco mil</p>
```

En el caso de matrices de dos o más dimensiones, las variables no se puede insertar directamente:

```
<?php
  $nombre = "Don Pepito";
  $saludos = [["Hello", "Hola"], ["Goodbye", "Adios"]];

  print "<p>¡$saludos[0][1], $nombre! ¿Cómo está usted?</p>";
?>
```

```
PHP Notice:  Array to string conversion in ejemplo.php on line 5
<p>¡Array[1], Don Pepito! ¿Cómo está usted?</p>
```

Este resultado se obtiene porque PHP no sustituye la variable $saludos[0][1] por su valor, sino que sustituye únicamente la primera parte ($saludos[0]). Como $saludos[0] es una matriz de una dimensión, no puede escribir ningún valor y devuelve simplemente "Array". A continuación, PHP añade el [1] que quedaba y se obtiene la cadena "Array[1]".
Una solución a este problema es sacar la matriz de la cadena:

```
<?php
  $nombre = "Don Pepito";
  $saludos = [["Hello", "Hola"], ["Goodbye", "Adios"]];
  print "<p>¡" . $saludos[[0][1] . ", $nombre! ¿Cómo está
usted?</p>";
?>
```

```
<p>¡Hola, Don Pepito! ¿Cómo está usted?</p>
```

Otra solución sin necesidad de sacar la matriz de la cadena es rodear la variable con llaves ({ }):

```
<?php
  $nombre = "Don Pepito";
  $saludos = [["Hello", "Hola"], ["Goodbye", "Adios"]];
  print "<p>¡{$saludos[0][1]}, $nombre! ¿Cómo está usted?</p>";
?>
```

```
<p>¡Hola, Don Pepito! ¿Cómo está usted?</p>
```

## Operaciones aritméticas

En el interior de las cadenas no se realizan operaciones aritméticas.

```
<?php
  $x = 3;
  $y = 4;
  print "<p>Suma: $x + $y = $x + $y</p>";
  print "<p>Multiplicación: $x x $y = $x * $y</p>";
?>
```

```
<p>Suma: 3 + 4 = 3 + 4</p>
<p>Multiplicación: 3 x 4 = 3 x 4</p>
```

Si se quiere mostrar el resultado de operaciones matemáticas, es necesario efectuar las operaciones fuera de las cadenas. En algunos casos no es necesario escribir las operaciones entre paréntesis, pero se recomienda escribirlas siempre entre paréntesis para no tener que preocuparse por cuándo hacen falta y cuándo no.

```
<?php
  $x = 3;
  $y = 4;
  print "<p>Suma: $x + $y = " . ($x + $y) . "</p>";
  print "<p>Multiplicación: $x x $y = " . ($x * $y) . "</p>";
?>
```

```
<p>Suma: 3 + 4 = 7</p>
<p>Multiplicación: 3 x 4 = 12</p>
```

Si no se escriben paréntesis en las operaciones, pueden no producirse errores, pero el resultado puede ser distinto al esperado en algunos casos:

```
<?php
  $x = 3;
  $y = 4;
  print "<p>Suma: $x + $y = " . $x + $y . "</p>";
  print "<p>Multiplicación: $x x $y = " . $x * $y . "</p>";
?>
```

```
PHP Warning:  A non-numeric value encountered in ejemplo.php on
line 4
```

Por supuesto, siempre se pueden utilizar definir auxiliares que guarden los resultados y utilizar las variables auxiliares en la cadena.

```
<?php
  $x = 3;
  $y = 4;
  $suma = $x + $y;
  $producto = $x * $y;
  print "<p>Suma: $x + $y = $suma</p>";
  print "<p>Multiplicación: $x x $y = $producto</p>;
?>
```

```
<p>Suma: 3 + 4 = 7</p>
<p>Multiplicación: 3 x 4 = 12</p>
```

## Nombres de variables

Si se quiere escribir el nombre de una variable, es decir, para que PHP no sustituya la variable por su valor, hay que escribir una contrabarra (\) antes de la variable.

```
<?php
```

```
  $x = 3;
  print "<p>La variable \$x vale $x</p>";
?>
```

```
<p>La variable $x vale 3</p>
```

## Unir variables y texto

En html/css a veces es necesario juntar números y caracteres, como en el ejemplo siguiente en el que se establece el tamaño del párrafo en 30px:

```
<?php
  print "<p style=\"font-size: 30px\">Texto grande</p>";
?>
```

```
<p style="font-size: 30px">Texto grande</p>
```

Si el tamaño está almacenado en una variable, no se puede juntar la variable con los caracteres ya que se interpretaría como una variable que no está definida y toma el valor vacío:

```
<?php
  $x = 30;

  print "<p style=\"font-size: $x px\">Texto grande</p>";
?>
```

```
<p style="font-size: ">Texto grande</p>
```

... pero se pueden utilizar llaves o sacar la variable de la cadena:

```
<?php
  $x = 30;

  print "<p style=\"font-size: {$x}px\">Texto grande</p>";
?>
```

```
<p style="font-size: 30px">Texto grande</p>
```

```
<?php
  $x = 30;
  print "<p style=\"font-size:" . $x . "px\">Texto grande</p>";
```

```
?>
```

```
<p style="font-size: 30px">Texto grande</p>
```

```
<?php
  $x = 30;
  print "<p style=\"font-size: $x" . "px\">Texto grande</p>";
?>
```

```
<p style="font-size: 30px">Texto grande</p>
```

Dentro de las llaves no se pueden realizar operaciones. Si se realizan operaciones, se puede producir un error de sintaxis u obtener resultados no esperados:

```
<?php
  $x = 7;

  print "<p>{$x} por {$x} es {$x * $x}</p>";
?>
```

```
PHP Parse error:  syntax error, unexpected '*', expecting ::
(T_PAAMAYIM_NEKUDOTAYIM) in ejemplo.php on line 4
```

```
<?php
  $x = 7;

  print "<p>2 por {$x} es {2 * $x}</p>";
?>
```

```
2 por 7 es {2 * 7}
```

# Tipos de variables

## Tipos de variables básicos

Los tipos de variables básicos son los siguientes:

- lógicas o booleanas (boolean)
- enteros (integer)
- decimales (float)
- cadenas (string)
- matrices (arrays)

Existen además los tipos:

- objetos (object)
- recursos (resource)
- nulo (null)

## Variables lógicas (boolean)

Las variables de tipo lógico sólo pueden tener el valor **true** (verdadero) o **false** (falso). Se suelen utilizar en las estructuras de control.

**Nota**: El ejemplo siguiente utiliza la estructura de selección if que se explica en detalle en la lección de estructuras de control. Para entender este ejemplo, es suficiente saber que if significa si (si como condición, no sí como afirmación) y va seguida de una comparación de igualdad == (las comparaciones se explican en la lección de operaciones lógicas). En caso de que la comparación sea cierta, es decir si los dos términos a ambos lados de la comparación son iguales, se ejecuta la instrucción entre llaves{ }.

```
<?php
  $autorizado = true;

  if ($autorizado == true) {
    echo "<p>Usted está autorizado.</p>";
  }

  if ($autorizado == false) {
    echo "<p>Usted no está autorizado.</p>";
  }
?>
```

```
<p>Usted está autorizado.</p>
```

Estos valores se pueden escribir en mayúsculas o minúsculas o combinando ambas, aunque se recomienda utilizar minúsculas:

```
<?php
  $autorizado = fAlSe;

  if ($autorizado == TRUe) {
    echo "<p>Usted está autorizado.</p>";
  }

  if ($autorizado == FaLsE) {
    echo "<p>Usted no está autorizado.</p>";
  }
?>
```

```
<p>Usted no está autorizado.</p>
```

**Nota**: No es necesario comparar una variable lógica con true o false, podemos emplear variables lógicas directamente en la condición del if.

```
<?php
  $autorizado = true;

  if ($autorizado) { // equivale a $autorizado == true
    echo "<p>Usted está autorizado.</p>";
  }

  if (!$autorizado) { // equivale a $autorizado == false
    echo "<p>Usted no está autorizado.</p>";
  }
?>
```

```
<p>Usted está autorizado.</p>
```

```
<?php
  $autorizado = false;

  if ($autorizado) { // equivale a $autorizado == true
    echo "<p>Usted está autorizado.</p>";
  }

  if (!$autorizado) { // equivale a $autorizado == false
    echo "<p>Usted no está autorizado.</p>";
  }
?>
```

```
<p>Usted no está autorizado.</p>
```

## Variables enteras (integer)

Las variables de tipo entero pueden guardar números enteros (positivos o negativos).

```
<?php
  $lado = 14;
  $area = $lado * $lado;

  echo "<p>Un cuadrado de lado $lado cm tiene un área de $area
cm<sup>2</sup>.</p>";

?>
```

```
<p>Un cuadrado de lado 14 cm tiene un área de 196 cm2</p>
```

Como las variables enteras se guardan en la memoria del ordenador utilizando un número de bytes fijo (que depende del ordenador, pero se puede averiguar mediante la constante predefinida **PHP_INT_SIZE**), no se pueden guardar números arbitrariamente grandes o pequeños. El valor más grande que se puede almacenar se puede averiguar mediante la constante predefinida **PHP_INT_MAX**.

## Notación complemento a dos

En un byte (8 bits) se pueden almacenar 256 ($2^8$) valores diferentes. Eso permite almacenar por ejemplo, desde el valor 0 hasta el 255. Para almacenar tanto números negativos como positivos, se utiliza la notación complemento a dos, en la que la primera mitad de los valores representan los valores positivos (desde 0 hasta 127) y la segunda mitad de los valores representan los valores negativos (desde -128 hasta -1), como indica la tabla siguiente

| El valor ... | 0 | 1 | 2 | 3 | 4 | 5 | ... | 126 | 127 | 128 | 129 | ... | 253 | 254 | 255 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ... representa el entero ... | 0 | 1 | 2 | 3 | 4 | 5 | ... | 126 | 127 | -128 | -127 | .. | -3 | -2 | -1 |

En cuatro bytes (32 bits) se pueden almacenar 4.294.967.296 ($2^{32}$)valores distintos. En notación complemento a dos se pueden representar desde el valor -2.147.483.648 hasta 2.147.483.647.

Si se intenta guardar un número demasiado grande positivo, el resultado será seguramente incorrecto. En el ejemplo siguiente se fuerza a PHP a guardar como entero el valor siguiente al máximo valor posible. El valor se guarda, pero cuando se utiliza, PHP lo interpreta como negativo.

```
<?php
  $maximo = PHP_INT_MAX;
  echo "<p>El mayor entero que se puede guardar en una variable
entera es $maximo</p>";

  $demasiado = (int)($maximo+1);
  echo "<p>Si se intenta guardar 1 más, el resultado es
$demasiado</p>";
?>
```

```
<p>El mayor entero que se puede guardar en una variable entera
es 2147483647</p>
<p>Si se intenta guardar 1 más, el resultado es -2147483648</p>
```

## Variables decimales (float)

Las variables de tipo decimal (float) pueden guardar números decimales (positivos o negativos). Como en las calculadoras, el separador de la parte entera y la parte decimal es el punto (.), no la coma (,).

```
<?php
  $lado = 14.5;
  $area = $lado * $lado;

  print "<p>Un cuadrado de lado $lado cm tiene un área de $area
cm<sup>2</sup>.</p>";
?>
```

```
<p>Un cuadrado de lado 14.5 cm tiene un área de 210.25 cm2</p>
```

Como las variables decimales se guardan en la memoria del ordenador utilizando un número de bytes fijo (que depende del ordenador, aunque no existen constantes predefinidas para conocer su tamaño), no se pueden guardar números arbitrariamente grandes o pequeños. Normalmente las variables decimales siguen la norma IEEE 758, concretamente el formato denominado "doble precisión", que emplea 8 bytes (64 bits) para almacenar un número.
De esos 64 bits, uno se utiliza para el signo, 53 para la parte fraccionaria y 11 para el exponente. Eso hace que el valor más grande que se pueda almacenar sea aproximadamente $1{,}8.10^{308}$.
Si se intenta guardar un número demasiado grande positivo (tanto si ese valor se obtiene en un cálculo o si se escribe directamente) PHP no puede manejarlo y lo que hace es sustituirlo por la constante predefinida INF.

```
<?php
  $maximo = 10 ** 308;  // 10^308

  print "<p>10<sup>308</sup> se puede guardar en una variable
decimal: $maximo</p>";

  $demasiado = 10 * $maximo;
  print "<p>Si se intenta guardar 10<sup>309</sup>, el resultado
es $demasiado</p>";

?>
```

```
<p>10308 se puede guardar en una variable decimal: 1.0E+308</p>
<p>Si se intenta guardar 10308, el resultado es INF</p>
```

## Variables de cadenas (string)

Las variables de tipo cadena pueden guardar caracteres.

PHP no impone ningún límite al tamaño de las cadenas. Las cadenas pueden ser todo lo largas que permita la memoria del servidor.
El juego de caracteres que utiliza PHP viene determinado en principio por el juego de caracteres que utiliza el fichero fuente del programa. Pero hay que tener en cuenta que las funciones de tratamiento de cadenas no están preparadas para tratar la diversidad de juegos de caracteres: muchas suponen que cada carácter ocupa solamente un byte, otras suponen un juego de caracteres determinado (UTF-8, por ejemplo), otras utilizan el juego de caracteres definido localmente, etc.
Se puede acceder a caracteres individuales indicando la posición del carácter, como si se tratara de una matriz de una dimensión en la que el primer carácter ocupa la posición 0.

```
<?php
  $saludo = "Hola, Don Pepito";;
  print "<p>$saludo</p>";

  $saludo[0] = "M";
  print "<p>$saludo</p>";

  $saludo[14] = "n";
  print "<p>$saludo</p>";
?>
```

```
<p>Hola, Don Pepito</p>
<p>Mola, Don Pepito</p>
<p>Mola, Don Pepino</p>
```

Si se indica una posición mayor que la longitud de la cadena, la cadena se alarga con espacios hasta llegar a ese valor:

```
<?php
 $saludo = "Hola, Don Pepito";;
  print "<p>$saludo</p>";

  $saludo[16] = "n";
  print "<p>$saludo</p>";

  $saludo[25] = "!";
  print "<p>$saludo</p>";
?>
```

```
<p>Hola, Don Pepito</p>
<p>Hola, Don Pepiton</p>
<p>Hola, Don Pepiton         !</p>
```

Si en una posición se guarda una cadena vacía, la cadena se acorta eliminando el carácter de esa posición

```
<?php
  $saludo = "Hola, Don Pepito";
  print "<p>$saludo</p>";

  $saludo[4] = "";
  print "<p>$saludo</p>";
?>
```

```
<p>Hola, Don Pepito</p>
<p>Hola Don Pepito</p>
```

### Matrices (arrays)

Las matrices se tratan en un apartado más adelante.

## Conversiones de tipos

PHP permite convertir variables de un tipo en otro tipo o considerar variables de un tipo como otro.

Recordemos que PHP es un lenguaje débilmente tipado, es decir, que no controla los tipos de variables que declara. De esta manera es posible utilizar variables de cualquier tipo en un mismo escenario. El tipo de una variable en PHP cuando no es declarado, es determinado por el contexto en el cual la variable va a ser usada. A continuación veremos todas las posibles conversiones de tipo que podemos hacer en PHP.

### Conversión Explícita

Este tipo de conversión puede realizarse mediante type casting y funciona de la misma manera como funciona en el lenguaje C, anteponiendo el tipo deseado entre paréntesis justo antes de la variable.

```
<?php
  $number = 1; // esta variable será de tipo integer

   $bool = (boolean) $number; // esta variable será de tipo
boolean

   echo "<p>".gettype($bool)."</p>";   // muestra "boolean"
?>
```

```
<p>boolean</p>
```

A continuación puedes ver una tabla de todos los posibles tipos de cast en PHP.

| Expression | Final type |
|---|---|
| (int), (integer) | integer |
| (bool), (boolean) | boolean |
| (float), (double), (real) | float |
| (string) | string |
| (array) | array |
| (object) | object |

Para que la variable en sí cambie de tipo está la función **settype()**.

```
<?php
  $height = "60.5";

  settype($height, 'integer'); // ahora la convertimos en integer

   echo "<p>".gettype($height)."</p>";    // muestra "integer"
?>
```

```
<p>integer</p>
```

## Conversión Automática

La conversión automática de tipos se realiza cuando se necesita un tipo específico de datos en una expresión. Veamos el siguiente ejemplo.

```
<?php

  $a = 1.5; // esta variable es de tipo double
  $b = 2; // esta variable es de tipo entero
  $c = $a * $b; // la variable $c será de tipo double

  echo "<p>".gettype($c)."</p>";    // muestra "double"
?>
```

```
<p>double</p>
```

## Variables como variables lógicas

Si una variable entera o decimal cuyo valor sea igual a cero se considera como variable lógica, se considera que tiene el valor **false**. Si una variable entera o decimal distinta de cero se considera como variable lógica, se considera que tiene el valor **true**.

```
<?php
  $autorizado = 5;
  if ($autorizado == true) {
    echo "<p>Usted está autorizado.</p>";
  }

  if ($autorizado == false) {
    echo "<p>Usted no está autorizado.</p>";
  }
?>
```

```
<p>Usted está autorizado.</p>
```

```
<?php
  $autorizado = 0.0;
  if ($autorizado == true) {
    echo "<p>Usted está autorizado.</p>";
  }

  if ($autorizado == false) {
    echo "<p>Usted no está autorizado.</p>";
  }
?>
```

```
<p>Usted no está autorizado.</p>
```

Si una cadena vacía se considera como variable lógica, se considera que tiene el valor **false**. Si una cadena no vacía se considera como variable lógica, se considera que tiene el valor **true**.

```
<?php
  $autorizado = "a ver qué pasa";
  if ($autorizado == true) {
    echo "<p>Usted está autorizado.</p>";
  }

  if ($autorizado == false) {
    echo "<p>Usted no está autorizado.</p>";
  }
?>
```

```
<p>Usted está autorizado.</p>
```

```
<?php
  $autorizado = "";
  if ($autorizado == true) {
    echo "<p>Usted está autorizado.</p>";
  }

  if ($autorizado == false) {
    echo "<p>Usted no está autorizado.</p>";
  }
?>
```

```
<p>Usted no está autorizado.</p>
```

Las cadenas no vacías se consideran como true, aunque tengan un valor que puedan confundirnos:

```
<?php
  $autorizado = "NO";
  if ($autorizado == true) {
    echo "<p>Usted está autorizado.</p>";
  }

  if ($autorizado == false) {
    echo "<p>Usted no está autorizado.</p>";
  }
?>
```

```
<p>Usted está autorizado.</p>
```

```
<?php
  $autorizado = "false";
  if ($autorizado == true) {
    echo "<p>Usted está autorizado.</p>";
  }

  if ($autorizado == false) {
    echo "<p>Usted no está autorizado.</p>";
  }
?>
```

```
<p>Usted está autorizado.</p>
```

Las matrices se consideran siempre como **true**:

```
<?php
  $autorizado["nombre"] = "Pepe";
  if ($autorizado == true) {
    echo "<p>Usted está autorizado.</p>";
  }

  if ($autorizado == false) {
    echo "<p>Usted no está autorizado.</p>";
  }
?>
```

```
<p>Usted está autorizado.</p>
```

```
<?php
  $autorizado["nombre"] = "";
  if ($autorizado == true) {
    print "<p>Usted está autorizado.</p>";
  }

  if ($autorizado == false) {
    print "<p>Usted no está autorizado.</p>";
  }
?>
```

```
<p>Usted está autorizado.</p>
```

## Qué es una matriz

Una matriz es un tipo de variable que permite almacenar simultáneamente varios datos diferentes, a los que se accede mediante un índice, numérico o de texto. En inglés, las matrices se llaman arrays.
En PHP, una matriz es un tipo de variable muy flexible, ya que podemos añadir, modificar, eliminar o reordenar los elementos de forma individual. Además los elementos pueden ser de tipos de datos diferentes.
Si los elementos de una matriz son datos de tipos simples (booleanos, enteros, decimales o cadenas), sólo se necesita un índice para identificar los datos. Se dice entonces que las matrices son unidimensionales. A las matrices de una dimensión también se les llama vectores.
Si los elementos de una matriz son a su vez también matrices, se necesitan varios índices para identificar a los datos. Se dice entonces que las matrices son multidimensionales.

## Crear una matriz

En la notación compacta, las matrices se crean empleando corchetes ([]).

```
<?php
  // Notación compacta
  $nombres = ["Ana", "Antonio", "Carmen"];
?>
```

Los elementos de la matriz deben separarse con comas. Tras el último elemento se puede escribir o no una coma, pero la coma final no crea un nuevo elemento (la matriz obtenida es la misma, independientemente de que se escriba la coma final o no).

```
<?php
  // Notación compacta
  $nombres = ["Ana", "Antonio", "Carmen",];
?>
```

Para hacer referencia a los valores individuales de la matriz, se deben utilizar índices, que se escriben entre corchetes ([ ]). Si al crear la matriz no se han indicado otros valores de índices, el primer término tiene el índice [0], el segundo tiene el índice [1], etc.:
PHP sustituye las referencias a valores de matrices de una dimensión dentro de las cadenas, por lo que no es necesario concatenar cadenas y referencias a matrices.

```
<?php
  $nombres = ["Ana", "Antonio", "Carmen"];
  echo "<p>$nombres[1]</p>";
  echo "<p>$nombres[0]</p>";
?>
```

```
<p>Antonio</p>
<p>Ana</p>
```

Una vez creada la matriz, los elementos individuales se pueden utilizar como si fueran variables independientes. Pero para referirse a un elemento individual, hay que indicar siempre el índice correspondiente.

```
<?php
  $nombres = ["Ana", "Antonio", "Carmen"];
  print "<p>$nombres[1]</p>";
  $nombres[1] = "David";
  print "<p>$nombres[1]</p>";
?>
```

```
<p>Antonio</p>
<p>David</p>
```

Si se solicita un valor no definido de una matriz, se produce un aviso (undefined offset). Los avisos no interrumpen la ejecución del programa, pero se deben corregir porque el programa seguramente no tendrá el comportamiento esperado:

```
<?php
  $nombres = ["Ana", "Antonio", "Carmen"];
  echo "<p>$nombres[3]</p>";
?>
```

```
Notice: Undefined offset: 3 in ejemplo.php on line 4
```

Se puede crear una matriz vacía (para añadirle posteriormente elementos). Se suele hacer para asegurarse de que una matriz ya utilizada en el programa no contiene elementos o para evitar errores si las operaciones que se van a realizar posteriormente requieren la existencia de la matriz (por ejemplo, si se van a utilizar uniones de matrices).

```
<?php
  $nombres = [];
?>
```

También se pueden crear matrices asignando directamente un elemento de la matriz.

```
<?php
  $apellidos[1] = García";
  print "<p>$apellidos[1]</p>;
?>
```

```
<p>García</p>
```

## Matrices asociativas

Las matrices de PHP son matrices asociativas, es decir, que los índices no tienen por qué ser correlativos, ni siquiera tienen por qué ser números.
Al crear matrices asociativas, debemos indicar el valor de los índices, utilizando la notación **$indice => $valor**:

```
<?php
  $cuadrados = [3 => 9, 5 => 25, 10 => 100];
  echo "<p>El cuadrado de 3 es $cuadrados[3]";
?>
```

```
<p>El cuadrado de 3 es 9</p>
```

PHP sustituye las referencias a valores de matrices de una dimensión dentro de las cadenas, por lo que no es necesario concatenar cadenas y referencias a matrices, pero los índices deben escribirse sin comillas, aunque sean cadenas.

```
<?php
  $edades = ["Andrés" => 20, "Antonio" => 25, "Ana" => 18];
  echo "<p>Ana tiene $edades[Ana] años</p>";
?>
```

```
<p>Ana tiene 18 años</p>
```

```
<?php
  $edades = ["Andrés" => 20, "Antonio" => 25, "Ana" => 18];
  echo "<p>Ana tiene " . $edades["Ana"] . " años";
?>
```

```
<p>Ana tiene 18 años</p>
```

Si la referencia a un valor de una matriz está fuera de una cadena o entre llaves, los índices que son cadenas deben escribirse con comillas.

```
<?php
  $edades = ["Andrés" => 20, "Antonio" => 25, "Ana" => 18];
  echo "<p>Ana tiene" .$edades[Ana]. " años";
?>
```

```
Notice: Use of undefined constant Ana - assumed 'Ana' in
ejemplo.php on line 4

<p>Ana tiene 18 años</p>
```

```
<?php
  $edades = ["Andrés" => 20, "Antonio" => 25, "Ana" => 18];
  echo "<p>Antonio tiene " . $edades["Ana"] . "años";
?>
```

```
<p>Ana tiene 18 años</p>
```

## Matrices multidimensionales

Las matrices pueden ser multidimensionales, es decir, matrices cuyos elementos son a su vez matrices. Para referirse a los elementos concretos, se necesitan utilizar varios índices (tantos como dimensiones -niveles de anidamiento- tenga la matriz).
PHP no sustituye las referencias a valores de matrices de más de una dimensión dentro de las cadenas, por lo que se necesita o bien utilizar llaves, o bien concatenar cadenas y referencias a matrices. Como en el caso de las matrices de una dimensión, los índices que sean cadenas deben escribirse con comillas si la referencia está dentro de una cadena o entre llaves.

```
<?php
  $datos = [
   ["nombre" => "pepe", "edad" =>  25, "peso" => 80],
   ["nombre" => "juan", "edad" =>  22, "peso" => 75]
  ];

  echo "<p>$datos[0][nombre] tiene $datos[0][edad] años.</p>";
?>
```

```
Notice: Array to string conversion in ejemplo.php on line 6

Notice: Array to string conversion in ejemplo.php on line 6

Array[nombre] tiene Array[edad] años.
```

```
<?php
  $datos = [
   ["nombre" => "pepe", "edad" =>  25, "peso" => 80],
   ["nombre" => "juan", "edad" =>  22, "peso" => 75]
  ];
  echo "<p>{$datos[1]["nombre"]} pesa {$datos[1]["peso"]}
kilos.</p>";
  echo "<p>" . $datos[0]["nombre"] . " tiene " .
$datos[0]["edad"] . " años.</p>";
?>
```

```
<p>juan pesa 75 años</p>
<p>pepe tiene 25 años</p>
```

## Imprimir todos los valores de una matriz: la función print_r()

La instrucción **print** permite imprimir valores individuales de una matriz, pero no matrices completas.

```
<?php
  $datos["nombre"] = "Santiago";
  $datos["apellidos"] = "Ramón y Cajal";

  print "$datos";
?>
```

```
Notice: Array to string conversion in ejemplo.php on line 5
Array
```

```
<?php
  $datos["nombre"] = "Santiago";
  $datos["apellidos"] = "Ramón y Cajal";

  print "{$datos}";
?>
```

```
Notice: Array to string conversion in ejemplo.php on line 5
Array
```

La función **print_r($variable [, $devolver])** permite imprimir todos los valores de una matriz de forma estructurada. En general, **print_r()** imprime cualquier variable compuesta de forma legible.
Esta función no se suele utilizar en programas definitivos, pero puede ser útil mientras estamos elaborando un programa, por ejemplo para comprobar en un punto del programa si la matriz contiene los valores esperados. Una vez comprobado, la instrucción se puede comentar o borrar.
Aunque **print_r()** genera espacios y saltos de línea en el código fuente de la página para indicar el anidamiento, **print_r()** no genera etiquetas html, por lo que el navegador no muestra esos espacios y saltos de línea.

```
<?php
  $datos["nombre"] = "Santiago";
  $datos["apellidos"] = "Ramón y Cajal";

  print_r($datos);
?>
```

```
Array ( [nombre] => Santiago [apellidos] => Ramón y Cajal )
```

Para mejorar la legibilidad una solución es añadir la etiqueta <pre>, que fuerza al navegador a mostrar los espacios y saltos de línea.

```
<?php
  $datos["nombre"] = "Santiago";
  $datos["apellidos"] = "Ramón y Cajal";

  print "<pre>";
  print_r($datos);
  print "</pre>";
?>
```

```
Array
(
[nombre] => Santiago
[apellidos] => Ramón y Cajal
)
```

```
<?php
  $datos[1]["nombre"] = "Santiago";
  $datos[1]["apellidos"] = "Ramón y Cajal";
  $datos[2]["nombre"] = "Leonardo";
  $datos[2]["apellidos"] = "orres Quevedo";

  print "<pre>";
  print_r($datos);
  print "</pre>";
?>
```

```
Array
(
[1] => Array
    (
        [nombre] => Santiago
        [apellidos] => Ramón y Cajal
    )

[2] => Array
    (
        [nombre] => Leonardo
        [apellidos] => Torres Quevedo
    )
)
```

Si se añade un segundo argumento con el valor **true**, **print_r()** no imprime nada pero devuelve el texto que se imprimiría si no estuviera el argumento **true**. Se utiliza para

concatenar la respuesta de la función o para guardarla en una variable que se puede imprimir posteriormente.

```
<?php
  $datos["nombre"] = "Santiago";
  $datos["apellidos"] = "Ramón y Cajal";

  $tmp = print_r($datos, true);
  print "<p>La matriz es $tmp</p>";
?>
```

```
La matriz es Array ( [nombre] => Santiago [apellidos] => Ramón y
Cajal )
```

```
<?php
  $datos["nombre"] = "Santiago";
  $datos["apellidos"] = "Ramón y Cajal";

  $tmp = print_r($datos, true);
  echo "<p>La matriz es " . print_r($datos, true) . "</p>";

?>
```

```
La matriz es Array ( [nombre] => Santiago [apellidos] => Ramón y
Cajal )
```

## Añadir elementos a una matriz

En la notación compacta, se pueden añadir elementos a una matriz indicando o no el índice del nuevo elemento:

- Si no se indica el índice, el nuevo elemento toma como índice el siguiente al mayor de los existentes (o el índice 0 si no había ningún valor numérico):

```
<?php
  $datos["nombre"] = "Santiago";
  $datos["apellidos"] = "Ramón y Cajal";

  $tmp = print_r($datos, true);
  echo "<p>La matriz es " . print_r($datos, true) . "</p>";
?>
```

```
La matriz es Array ( [nombre] => Santiago [apellidos] => Ramón y
Cajal )
```

```php
<?php
  $nombres = ["Alba", "Bernardo"];
  $nombres[] = "Carlos";

  echo "<p>$nombres[1]</p>";
  echo "<p>$nombres[2]</p>";
?>
```

```
<p>Bernardo</p>
<p>Carlos</p>
```

```php
<?php
  $nombres = [4 => "Alba", 6 => "Bernardo"];
  $nombres[] = "Carlos";

  echo "<p>$nombres[7]</p>";
?>
```

```
<p>Carlos</p>
```

```php
<?php
  $nombres = ["a" => "Alba", "b" => "Bernardo"];
  $nombres[] = "Carlos";
  echo "<p>$nombres[0]</p>";
?>
```

```
<p>Carlos</p>
```

- Indicando el índice, se asigna el elemento de la matriz:

```php
<?php
  $nombres = ["Alba", "Bernardo"];
  $nombres[3] = "Carlos";

  echo "<p>$nombres[1]</p>";
  echo "<p>$nombres[3]</p>";
?>
```

```
<p>Bernardo</p>
<p>Carlos</p>
```

## Unión de matrices

Los operadores **+** (suma) **+=** (suma combinada) realizan la unión de dos matrices. La unión de dos matrices contiene todos los elementos de la primera matriz y únicamente los elementos de la segunda matriz cuyo índice no se encuentra en la primera matriz.

- En el ejemplo siguiente, al hacer la unión $a + $b, el resultado contiene los dos elementos de $a y el único elemento de $b cuyo índice no está en $a (el elemento $b[2]). Sin embargo, al hacer la unión $b + $a, el resultado contiene los tres elementos de $b pero ninguno de $a (ya que el índice de los dos elementos de $a, 0 y 1, están en $b).

```
<?php
 $nombres_1 = ["Alba", "Bernardo"];
 $nombres_2 = ["Antonio", "Ana", "Carlos"];

 echo "<pre>";
 print_r($nombres_1 + $nombres_2);
 echo "</pre>";

 echo "<pre>";
 print_r($nombres_2 + $nombres_1)
 echo "</pre>";
?>
```

```
Array
(
[1] => Array
    (
        [0] => Alba
        [1] => Bernardo
        [2] => Carlos
    )

[2] => Array
    (
        [0] => Antonio
        [1] => Ana
        [2] => Carlos
    )
)
```

- Si los índices no coinciden, la unión contendrá todos los elementos:

```
<?php
  $nombres_1 = [0 => "Alba", 2 => "Bernardo"];
  $nombres_2 = [1 => "Antonio", 3 => "Ana", 5 => "Carlos"];
```

```
  echo "<pre>";
  print_r($nombres_1 + $nombres_2);
  echo "</pre>";

  echo "<pre>";
  print_r($nombres_2 + $nombres_1);
 echo print "</pre>";
?>
```

```
Array
(
[1] => Array
    (
        [0] => Alba
        [2] => Bernardo
        [1] => Antonio
        [3] => Ana
        [5] => Carlos
    )

[2] => Array
    (
        [1] => Antonio
        [3] => Ana
        [5] => Carlos
        [0] => Alba
        [2] => Bernardo
    )
)
```

- El operador combinado **+=** realiza también la unión de dos matrices (y modifica, en su caso, la primera matriz).

```
<?php
  $nombres_1 = ["Alba", "Bernardo"];
  $nombres_2 = ["Antonio", "Ana", "Carlos"];
  $nombres_1 += $nombres_2;

  echo "<pre>";
  print_r($nombres_1);
  echo "</pre>";
?>
```

```
Array
(
    [0] => Alba
```

```
    [1] => Bernardo
    [2] => Carlos
)
```

```
<?php
  $nombres_1 = ["Alba", "Bernardo"];
  $nombres_2 = ["Antonio", "Ana", "Carlos"];
  $nombres_2 += $nombres_1;

  echo "<pre>"
  print_r($nombres_2);
  echo "</pre>";
?>
```

```
Array
(
    [0] => Antonio
    [1] => Ana
    [2] => Carlos
)
```

## Más información general sobre las matrices

Cada elemento de la matriz se comporta como una variable independiente y se pueden almacenar datos de tipos distintos en una misma matriz.

```
<?php
  $datos = ["Santiago", "Ramón y Cajal", 1852];
  echo "<p>$datos[0] $datos[1] nació en $datos[2].</p>";
?>
```

```
<p>Santiago Ramón y Cajal nació en 1852.</p>
```

Los valores de los índices numéricos suelen ser siempre positivos, pero PHP también permite que sean negativos., A continuación se muestran distintas formas de imprimirlos:

```
<?php
  $nombres[-3] = "Hola";
  echo "<p>$nombres[-3]</p>";
?>
```

```
<p>Hola</p>
```

```
<?php
  $nombres[-3] = "Alba";

  print "<p>{$nombres[-3]}</p>";
  print "<p>" . $nombres[-3] . "</p>";
?>
```

```
<p>Alba</p>
<p>Alba</p>
```

PHP no permite valores decimales en los índices numéricos. En caso de usarlos, PHP los trunca automáticamente a enteros (por lo que en general no se recomienda utilizarlos):

```
<?php
  $nombres[1.7] = "Alba";

  print "<p>$nombres[1]</p>";
?>
```

```
<p>Alba</p>
```

```
<?php
  $nombres[1.7] = "Alba";

  print "<p>$nombres[1.2]</p>";
?>
```

```
<p>Alba</p>
```

Como en el caso de los negativos, hay que tener cuidado al imprimirlos:

```
<?php
  $nombres[1.7] = "Alba";

  print "<p>$nombres[-1.7]</p>";
?>
```

```
Parse error: syntax error, unexpected '.', expecting ']' in
ejemplo.php on line 3
```

```
<?php
```

```
  $nombres[-1.7] = "Alba";

  print "<p>{$nombres[-1.2]}</p>";
  print "<p>" . $nombres[-1.2] . "</p>";
?>
```

```
<p>Alba</p>
<p>Alba</p>
```

En el caso de las matrices multidimensionales, la misma información se puede guardar de varias formas distintas. Por ejemplo, la matriz siguiente guarda información sobre la edad y el peso de varias personas:

```
<?php
  $datos["pepe"]["edad"] = 25;
  $datos["pepe"]["peso"] = 80;
  $datos["juan"]["edad"] = 22;
  $datos["juan"]["peso"] = 75;

  echo "<pre>";
  print_r($datos);
  echo "</pre>";
?>
```

```
Array
(
    [pepe] => Array
        (
            [edad] => 25
            [peso] => 80
        )
    [juan] => Array
        (
            [edad] => 22
            [peso] => 75
        )
)
```

En el ejemplo anterior se ha utilizado el nombre de la persona como índice de la matriz, lo que no es una buena idea, ya que no podríamos guardar los datos de dos personas con el mismo nombre. Sería mejor guardar la información de manera similar a como se guardaría en una base de datos. Una matriz de dos dimensiones sería en ese caso el equivalente a una tabla, el primer índice correspondería al número de registro y el segundo índice iría tomando los valores de los campos de la tabla (nombre, edad, peso, etc).

```php
<?php
  $datos[0]["nombre"] = "pepe";
  $datos[0]["edad"] = 25;
  $datos[0]["peso"] = 80;
  $datos[1]["nombre"] = "juan";
  $datos[1]["edad"] = 22;
  $datos[1]["peso"] = 75;

  print "<pre>"; print_r($datos); print "</pre>";
?>
```

```
Array
(
    [0] => Array
        (
            [nombre] => pepe
            [edad] => 25
            [peso] => 80
        )
    [1] => Array
        (
            [nombre] => juan
            [edad] => 22
            [peso] => 75
        )
)
```

## Borrar una matriz o elementos de una matriz

La función **unset()** permite borrar una matriz o elementos de una matriz.

```php
<?php
  $matriz = [5 => 25, -1 => "negativo", "número 1" => "cinco"];

  print "<pre>"; print_r($matriz); print "</pre>";
  unset($matriz[5]);
  print "<pre>"; print_r($matriz); print "</pre>";
?>
```

```
Array
(
    [5] => 25
    [-1] => negativo
    [número 1] => cinco
)
```

```
Array
(
    [-1] => negativo
    [número 1] => cinco
)
```

```
<?php
  $matriz = [5 => 25, -1 => "negativo", "número 1" => "cinco"];

  print "<pre>"; print_r($matriz); print "</pre>";

  unset($matriz);

  print "<pre>"; print_r($matriz); print "</pre>";
?>
```

```
Array
(
    [5] => 25
    [-1] => negativo
    [número 1] => cinco
)

Notice:  Undefined variable: matriz in ejemplo.php on line 8
```

Si se intenta borrar un elemento no definido, PHP no genera ningún aviso.

```
<?php
  $nombres = ["Alba", "Bernardo"];

  print "<pre>"; print_r($nombres); print "</pre>";

  unset($nombres[3]);

  print "<pre>"; print_r($nombres); print "</pre>";
?>
```

```
Array
(
    [0] => Alba
    [1] => Bernardo
)
Array
(
    [0] => Alba
    [1] => Bernardo
```

```
)
```

## Copiar una matriz

Se puede copiar una matriz creando una nueva variable. Las dos matrices son independientes, por lo que modificar posteriormente una de ellas no afecta a la otra.

```
<?php
  $cuadrados = [5 => 25, 9 => 81];
  $cuadradosCopia = $cuadrados;

  $cuadrados[] = 100;

  print "<p>Matriz inicial (modificada):</p>";
  print "<pre>"; print_r($cuadrados); print "</pre>";

  print "<p>Matriz inicial (sin modificar):</p>";
  print "<pre>"; print_r($cuadradosCopia); print "</pre>";
?>
```

```
<p>Matriz inicial (modificada):</p>
Array
(
    [5] => 25
    [9] => 81
    [10] => 100
)

<p>Matriz inicial (sin modificar):</p>
Array
(
    [5] => 25
    [9] => 81
)
```

# Variables predefinidas

PHP genera automáticamente una serie de variables con diversa información sobre el cliente y el servidor.

## $_REQUEST

**$_REQUEST** es una matriz asociativa que contiene los datos enviados por los formularios y las cookies guardadas en el ordenador del cliente.

## $_SERVER

$_SERVER es una matriz asociativa que contiene información sobre cabeceras, rutas y ubicaciones de scripts suministrada por el servidor (pero hay que tener en cuenta que no todos los servidores suministran todos los datos).

### $_SERVER[PHP_SELF]

$_SERVER[PHP_SELF] contiene la dirección de la página (relativo a la raíz, es decir, sin el nombre del servidor).

| URL | $_SERVER[PHP_SELF] |
|---|---|
| http://www.example.com/ejemplo.php | /ejemplo.php |
| http://www.example.com/ejercicios/ejemplo.php | /ejercicios/ejemplo.php |

Esta variable se puede utilizar en las páginas que enlazan consigo mismas. Por ejemplo, una página puede contener un formulario que envíe los datos a esa misma página. Para ello, el atributo **action** del formulario debe contener el nombre de la página. En vez de poner el nombre de la página, se puede utilizar $_SERVER[PHP_SELF]. La ventaja es que aunque se cambie el nombre del fichero, el enlace seguirá funcionando.

El ejemplo siguiente muestra cómo se escribiría un enlace que apunta a la misma página que contiene el enlace:

```
<?php

   echo "<p><a href=\"$_SERVER[PHP_SELF]\">Enlace a esta misma página</a></p>";

?>
```

# Constantes y constantes predefinidas

PHP permite definir constantes e incluye una serie de constante predefinidas que se pueden utilizar directamente:

## Constantes

Las constantes son elementos de PHP que guardan un valor fijo que no se puede modificar a lo largo del programa. Las constantes pueden ser **definidas por el programa** o estar **predefinidas** por el propio PHP o por algún módulo. Los nombres de las constantes siguen las mismas reglas que los nombres de las variables, pero sin el dólar ($) inicial. La costumbre es escribir los nombres de las constantes en mayúsculas.

En principio, se puede no utilizar constantes nunca, puesto que las constantes definidas por el programa podrían reemplazarse por variables. La ventaja de usar constantes y variables es que se puede distinguir a simple vista si a lo largo de un programa algo va a permanecer constante (si es una constante) o puede cambiar (si es una variable). El inconveniente de usar constantes es que las constantes no se sustituyen dentro de las cadenas y es necesario sacarlas fuera de las cadenas. Desde el punto de vista del rendimiento, la diferencia es inapreciable.

## Definir constantes

La función **define(nombre_constante, valor_constante)** permite definir constantes.

```
<?php
  define("EDADJUBILACION", 67);
  echo "<p>La edad de jubilación en España es " . EDADJUBILACION
. "</p>";
?>
```

```
<p>La edad de jubilación en España es 67</p>
```

```
<?php
  define("PROFESOR", "Antonio Gago");
  echo "<p>Profesor: " . PROFESOR . "</p>";
?>
```

```
<p>Profesor: Antonio Gago</p>
```

Otra sintaxis que nos ofrece PHP es usando la palabra **const**, como en JavaScript.

```
<?php
  const GRAVEDAD = 9.8;
  echo "<p>" .GRAVEDAD . "</p>";
?>
```

```
<p>9.8</p>
```

La costumbre es escribir los nombres de las constantes en mayúsculas. Los nombres de las constantes deben empezar por una letra y tener sólo letras (acentos, etc), números y guiones bajos.
Las constantes no se sustituyen dentro de una cadena (ni siquiera escribiéndolas entre llaves), por lo que es necesario concatenarlas para mostrar su valor.

```
<?php
```

```
define("EDADJUBILACION", 67);

print "<p>El valor de edad de jubilación es EDADJUBILACION</p>";
// El valor NO se sustituye
print "<p>El valor de edad de jubilación {EDADJUBILACION}</p>";
// El valor NO se sustituye
print "<p>El valor de edad de jubilación es " . EDADJUBILACION .
"</p>";
?>
```

```
<p>El valor de edad de jubilación es EDADJUBILACION</p>
<p>El valor de edad de jubilación es (EDADJUBILACION)</p>
<p>El valor de edad de jubilación es 67</p>
```

En las constantes, PHP distingue entre minúsculas y mayúsculas:

```
<?php
define("EDADJUBILACION", 67);
define("edadJubilacion", 65);

print "<p>El valor de EDADJUBILACION es " . EDADJUBILACION.
"</p>";
print "<p>El valor de edadJubilacion es " . edadJubilacion .
"</p>";
?>
```

```
<p>El valor de EDADJUBILACION es 67</p>
<p>El valor de edadJubilacion es 65</p>
```

# Constantes predefinidas

Tanto PHP como los módulos cargados definen automáticamente una serie de **constantes predefinidas**.

## Lista completa de constantes predefinidas

El número de constantes predefinidas depende de los módulos cargados en php.ini. La función **get_defined_constants()** devuelve las constantes predefinidas en el servidor que estemos utilizando:

```
<?php
  print "<pre>";
  print_r(get_defined_constants());
  print "</pre>";
?>
```

```
Array
(
    [E_ERROR] => 1
    [E_RECOVERABLE_ERROR] => 4096
    [E_WARNING] => 2
    [E_PARSE] => 4
    [E_NOTICE] => 8
    [E_STRICT] => 2048
    ...
)
```

## INF

La constante **INF** representa el infinito, es decir, cualquier número demasiado grande (positivo o negativo) para poderse guardar en una variable decimal.

## PHP_INT_MAX

PHP_INT_MAX es el valor del mayor entero que se puede guardar en una variable de tipo entero.

```
<?php
 $maximo = PHP_INT_MAX;

 print "<p>El mayor entero que se puede guardar en una variable
entera es $maximo</p>";
?>
```

```
<p>El mayor entero que se puede guardar en una variable entera
es 2147483647</p>
```

Ese valor depende del tamaño en bytes que ocupan las variables en la memoria, que depende del microprocesador, del sistema operativo y de la versión de PHP. En sistemas de 32 bits, los enteros suelen ocupar 4 bytes (32 bits). Como en una variable de tipo entero se guardan tanto valores positivos como negativos (utilizando la notación complemento a dos para los negativos), el valor máximo suele ser **2.147.483.647** ($2^{31}$ - 1). En sistemas de 64 bits, los enteros suelen ocupar 8 bytes (64 bits) y el valor máximo suele ser **9.223.372.036.854.775.807** ($2^{63}$ - 1)

## PHP_INT_SIZE

PHP_INT_SIZE es el tamaño en bytes de las variables de tipo entero, que depende del ordenador, del sistema operativo y de la versión de PHP. En sistemas de 32 bits suele ser 4 bytes y en sistema de 64 bits suele ser 8 bytes.

```
<?php
  $maximo = PHP_INT_SIZE;

  print "<p>Los enteros se guardan utilizando $tamano bytes</p>";
?>
```

```
<p>Los enteros se guardan utilizando 8 bytes</p>
```

Nota: En las versiones de PHP de 64 bits para Windows anteriores a PHP 7, los enteros tenían 4 bytes de tamaño. A partir de PHP 7, en las versiones de PHP de 64 bits para Windows los enteros tienen un tamaño de 8 bytes.