Tipos de datos compuestos en PHP
Arrays

Tipos de datos compuestos:

Arrays

Objetos

- Un tipo de datos compuesto es aquel que permite almacenar más de un valor. En PHP existen dos tipos de datos compuestos: el array y el objeto.
- Un array es un tipo de datos que nos permite almacenar varios valores. Cada miembro del array se almacena en una posición a la que se hace referencia <u>utilizando un valor clave o índice</u>.
- Cada uno de los elementos de ese array se identifica por medio de un nombre (común para todos) y un índice (que diferenciaría a cada uno de ellos).

La sintaxis que permite definir elementos en un array es esta:

\$nombre[indice]

Arrays o matrices

Arrays escalares o numéricos: los índices de un array son números.

Arrays asociativos: los índices de un array son cadenas.

```
$a['indice']=valor;
$a["índice"]=valor;
```

Es posible –y bastante frecuente– <u>utilizar como índice el contenido de</u> <u>una variable</u>. El modo de hacerlo sería:

```
$a[$indice]=valor;
```

En este caso, sea cual fuere el tipo de la variable \$indice, el nombre de la variable nunca se pone entre comillas.

Arrays o matrices

Tablas unidimensionales					
Array escalar			Array asociativo		
Variable	Indice	Valor	Variable	Indice	Valor
\$a[0]	0	Domingo	\$a['Primero']	Primero	Domingo
\$a[1]	1	Lunes	\$a['Segundo']	Segundo	Lunes
\$a[2]	2	Martes	\$a['Tercero']	Tercero	Martes
\$a[3]	3	Miércoles	\$a['Cuarto']	Cuarto	Miércoles
\$a[4]	4	Jueves	\$a['Quinto']	Quinto	Jueves
\$a[5]	5	Viernes	\$a['Sexto']	Sexto	Viernes
\$a[6]	6	Sábado	\$a['Septimo']	Septimo	Sábado

Características de los Arrays

Declaración Dinámica:

En PHP, no es necesario especificar el tamaño del array antes de crearlo. Puedes comenzar a añadir valores directamente.

Tipado Flexible:

No se requiere declarar que una variable es de tipo array. PHP reconoce automáticamente el tipo de dato.

Claves Automáticas:

Al añadir elementos, si no se especifica una clave, PHP asignará automáticamente la siguiente clave numérica disponible, comenzando desde 0 si no hay claves previas.

Almacenamiento de Múltiples Tipos:

No es necesario definir el tipo de dato que se almacenará en el array. En un mismo array, puedes almacenar diferentes tipos de datos (enteros, cadenas, objetos, otros arrays, etc.).

Claves Numéricas y Asociativas:

Las claves pueden ser numéricas o alfanuméricas. Cuando se utilizan claves alfanuméricas, se habla de arrays asociativos. Es posible mezclar ambos tipos de claves en un mismo array.

Asignación de Índices:

Al definir el índice de un array con una variable, el nombre de la variable no debe ir entre comillas.

Por ejemplo, \$indice es correcto, mientras que "\$indice" no lo es

Inicializar un array

Hay diferentes maneras de "inicializar" un array, darle un índice y un valor a cada una de sus celdas Podemos realizar esa tarea de:

- -forma explícita
- -forma implícita
- -mezclando ambas formas
- -usando el constructor o función array()

Inicializar un array

1.Forma Explícita:

•Se asignan valores a cada índice de manera directa. Por ejemplo:

```
$matriz = [
     0 => 'valor1',
     1 => 'valor2',
     2 => 'valor3'
];

$matriz[]=13;
$matriz[8]=14;
$matriz['clave']='esto es un string';
```

2.Forma Implícita:

 Se inicializa el array sin especificar los índices. PHP asigna automáticamente índices numéricos comenzando desde 0:

```
$matriz = ['valor1', 'valor2', 'valor3'];
```

3. Mezclando Ambas Formas:

•Se pueden combinar índices explícitos e implícitos en el mismo array:

```
$matriz = ['clave1' => 'valor1', 'clave2' => 'valor2', 3 => 'valor3'];
```

4.Usando el Constructor array():

• Esta es una forma clásica de crear un array utilizando la función array():

```
$matriz = array('valor1', 'valor2', 'valor3');

$matriz = array(
    0 => 'valor1',
    1 => 'valor2',
    2 => 'valor3'
```

Índices en Arrays de PHP

1.Índices no necesariamente consecutivos:

•En PHP, los índices de un array no tienen que comenzar en cero ni ser necesariamente consecutivos. Puedes asignar cualquier número como índice.

2. Asignación Automática de índices:

•Si no especificas un índice al añadir un valor a un array, PHP asignará automáticamente un índice numérico. Este índice comenzará desde el menor número posible, que generalmente es 0, a menos que ya existan índices definidos.

3.Omisión de Índices Específicos:

•Al dejar los corchetes vacíos al asignar un valor, omites el paso de especificar un índice. PHP se encargará de completar este índice automáticamente:

```
$array[] = 'valor1'; // Se asigna el índice 0
$array[] = 'valor2'; // Se asigna el índice 1
```

4.Lectura de Datos:

• Para acceder a los valores almacenados, es necesario especificar el índice correspondiente. Por ejemplo:

```
echo $array[0]; // Esto mostrará 'valor1'
```

•Si intentas acceder a un índice sin especificarlo, como en echo \$array[]; obtendrás un error, ya que PHP no sabe qué celda deseas leer.

5. Especificación de índices iniciales:

• También puedes definir un índice específico para el primer elemento y luego permitir que PHP continúe asignando índices automáticamente a partir de ese valor:

```
$array[5] = 'valor1'; // Asigna 'valor1' al índice 5
$array[] = 'valor2'; // Se asigna el índice 6 automáticamente
```

Amoldamientos de claves

En PHP, las claves pueden ser de diferentes tipos, y PHP realiza ciertas conversiones automáticas.

1.Cadenas que contienen números:

- •Si <u>una cadena</u> contiene un número decimal entero válido, PHP la convertirá automáticamente a un tipo integer. Por ejemplo:
 - •La clave "8" se almacenará como 8.
 - •Sin embargo, la clave "08" no se convertirá a un integer, ya que no es un número decimal válido.

2.Booleanos:

- •Los valores booleanos se amoldan a integers:
 - •La clave true se almacenará como 1.
 - •La clave false se almacenará como 0.

3.Null:

- •Un valor null se convertirá en un string vacío:
 - •La clave null se almacenará como "".

4. Cambios en PHP 8.1:

• A partir de PHP 8.1, ya no es posible que un número de tipo float se convierta implícitamente a integer. Intentar hacerlo generará una advertencia:

Deprecated: Implicit conversion from float 1.5 to int loses precision.

5. Claves no válidas:

•Los arrays y objetos no pueden ser utilizados como claves en un array. Si intentas hacerlo, recibirás una advertencia:

Illegal offset type

Amoldamientos de claves y sobreescritura

Si varios elementos en la declaración del array usan la misma clave, sólo se conservará la última, siendo las demás sobrescritas.

Ejemplo 1: Amoldamiento de Tipo y sobreescritura

```
<?php
$array = array(
   1 => 'a', // Clave 1
    '1' => 'b', // La clave '1' se amolda a 1, sobrescribiendo 'a'
    null => 'c', // La clave null se amolda a ''
   true => 'd', // La clave true se amolda a 1, sobrescribiendo 'b'
    '' => 'e'  // La clave '' se amolda a '', sobrescribiendo 'c'
);
var_dump($array);
                               array (size=2)
?>
                                1 => string 'd' (length=1)
                                '' => string 'e' (length=1)
```

Ejemplo 2: Inclusión de un float

```
<?php
$array = array(
    1 => "a", // Clave 1
    '1' => "b", // La clave '1' se amolda a 1, sobrescribiendo 'a'
    null => "c", // La clave null se amolda a ''
    true => "d", // La clave true se amolda a 1, sobrescribiendo 'b'
    '' => "e", // La clave '' se amolda a '', sobrescribiendo 'c'
    1.5 => "f", // La clave 1.5 se amolda a 1, sobrescribiendo 'd'
    '1.5' => "g"
                   // La clave "1.5" se mantiene como cadena , no sobrescribe nada
);
                                 array (size=3)
                                   1 => string 'f' (length=1)
var dump($array);
                                   '' => string 'e' (length=1)
?>
                                   '1.5' => string 'g' (length=1)
```

Los arrays bidimensionales en PHP pueden entenderse como tablas de doble entrada, donde cada elemento se identifica mediante un nombre seguido de dos índices. Estos índices pueden ser de tipo escalar (numéricos) o alfanuméricos (asociativos), lo que permite organizar los datos de manera estructurada.

Estructura de un Array Bidimensional

- Un array bidimensional se puede visualizar como una tabla, donde:
 - Filas: Representan el primer índice del array.
 - Columnas: Representan el segundo índice del array.
- Tipos de Índices

1. Índices Escalares:

- Los índices son números enteros que indican la posición en la tabla, similar a las filas y columnas.
- 2. Índices Alfanuméricos (Arrays Asociativos):
 - Los índices son cadenas de texto, permitiendo identificar más fácilmente cada fila y columna.

Ejemplo1 : Array Bidimensional con Índices Escalares

```
<?php
$tabla = array(
   array(1, 2, 3),
   array(4, 5, 6),
   array(7, 8, 9)
);
// Utilizando sintaxis corta
$tabla = [
   [1, 2, 3],
   [4, 5, 6],
    [7, 8, 9]
];
// Accediendo a un elemento
echo $tabla[1][2]; // Salida: 6
var_dump($tabla);
?>
```

```
array (size=3)
  0 =>
      array (size=3)
         0 \Rightarrow int 1
        1 \Rightarrow int 2
         2 \Rightarrow int 3
  1 =>
      array (size=3)
         0 \Rightarrow int 4
        1 \Rightarrow int 5
         2 \Rightarrow int 6
   2 =>
      array (size=3)
         0 \Rightarrow int 7
         1 \Rightarrow int 8
         2 \Rightarrow int 9
```

Ejemplo 2: Array Bidimensional con Índices Alfanuméricos

```
<?php
$tablaAsociativa = array(
     "fila1" => array("columna1" => "A", "columna2" => "B"),
     "fila2" => array("columna1" => "C", "columna2" => "D"),
     "fila3" => array("columna1" => "E", "columna2" => "F")
);
//Utilizando la sintaxis corta de array
$tablaAsociativa = [
     "fila1" => ["columna1" => "A", "columna2" => "B"],
     "fila2" => ["columna1" => "C", "columna2" => "D"],
     "fila3" => ["columna1" => "E", "columna2" => "F"]
];
// Accediendo a un elemento
echo $tablaAsociativa["fila2"]["columna1"]; // Salida: C
var dump($tablaAsociativa);
                                                     Array
print r($tablaAsociativa);
                                                         [fila1] => Array
                                                               [columna1] => A
                                                               [columna2] => B
array (size=3)
 'fila1' =>
                                                         [fila2] => Array
   array (size=2)
     'columna1' => string 'A' (length=1)
                                                               [columna1] => C
     'columna2' => string 'B' (length=1)
                                                               [columna2] => D
  'fila2' =>
   array (size=2)
     'columna1' => string 'C' (length=1)
                                                         [fila3] => Array
     'columna2' => string 'D' (length=1)
  'fila3' =>
                                                               [columna1] => E
                                                               [columna2] => F
   array (size=2)
     'columna1' => string 'E' (length=1)
     'columna2' => string 'F' (length=1)
```

Ejemplo 3: Combinación de Índices Escalares y Alfanuméricos

```
<?php
$datos = array(
   0 => array("nombre" => "Juan", "edad" => 25),
    1 => array("nombre" => "Ana", "edad" => 30),
    2 => array("nombre" => "Luis", "edad" => 22)
// Sintaxis corta
$datos = [
    ["nombre" => "Juan", "edad" => 25],
    ["nombre" => "Ana", "edad" => 30],
    ["nombre" => "Luis", "edad" => 22]
];
// Accediendo a un elemento
echo $datos[1]["nombre"]; // Salida: Ana
echo $datos[1]["edad"]; // Salida: 30
                                           array (size=3)
var_dump($datos);
```

```
0 =>
    array (size=2)
        'nombre' => string 'Juan' (length=4)
        'edad' => int 25
1 =>
    array (size=2)
        'nombre' => string 'Ana' (length=3)
        'edad' => int 30
2 =>
    array (size=2)
        'nombre' => string 'Luis' (length=4)
        'edad' => int 22
```

Ejemplo de Array Bidimensional Asociativo

```
<?php
$frutas = array(
     "tropicales" => array(
           "frutaA" => "Mango",
           "frutaB" => "Piña",
           "frutaC" => "Papaya"
     "cítricas" => array(
           "frutaX" => "Naranja", array (size=2)
           "frutaY" => "Limón",
                                            'tropicales' =>
                                             array (size=3)
           "frutaZ" => "Pomelo"
                                               'frutaA' => string 'Mango' (length=5)
                                               'frutaB' => string 'Piña' (length=5)
                                               'frutaC' => string 'Papaya' (length=6)
                                            'cítricas' =>
                                             array (size=3)
                                               'frutaX' => string 'Naranja' (length=7)
                                               'frutaY' => string 'Limón' (length=6)
// Accediendo a un elemento
                                               'frutaZ' => string 'Pomelo' (length=6)
echo $frutas["tropicales"]["frutaA"]; // Salida: Mango
echo $frutas["cítricas"]["frutaX"]; // Salida: Naranja
var dump($frutas);
?>
```

```
<?php
$frutas =
     "tropicales" =>
         "frutaA" =>
                                         [tropicales] => Array
         "frutaB" => "Piña",
                                               [frutaA] => Mango
         "frutaC" => "Papaya"
                                               [frutaB] => Piña
    ],
"cítricas" =>
                                               [frutaC] => Papaya
         "frutaX" => "Naranja",
                                         [cítricas] => Array
         "frutaY" => "Limón",
         "frutaZ" => "Pomelo"
                                               [frutaX] => Naranja
                                               [frutaY] => Limón
                                               [frutaZ] => Pomelo
];
// Accediendo a un elemento
echo $frutas["tropicales"]["frutaA"]; // Salida: Mango
echo $frutas["cítricas"]["frutaX"]; // Salida: Naranja
print r($frutas);
?>
```

Ejemplo 3: Utilizando variables como índices

```
<?php
// Definiendo variables para los índices
$tipoPrimarios = "primarios";
$tipoSecundarios = "secundarios";
// Inicializando un array bidimensional con variables
$colores = array(
                                                   array (size=2)
     $tipoPrimarios => array(
                                                     'primarios' =>
          "Rojo" => "#FF0000",
                                                      array (size=3)
                                                        'Rojo' => string '#FF0000' (length=7)
          "Verde" => "#00FF00",
                                                        'Verde' => string '#00FF00' (length=7)
          "Azul" => "#0000FF"
                                                        'Azul' => string '#0000FF' (length=7)
                                                     'secundarios' =>
                                                      array (size=3)
     $tipoSecundarios => array(
                                                        'Amarillo' => string '#FFFF00' (length=7)
          "Amarillo" => "#FFFF00",
                                                        'Naranja' => string '#FFA500' (length=7)
                                                        'Morado' => string '#800080' (length=7)
          "Naranja" => "#FFA500",
          "Morado" => "#800080"
                                                   Código hexadecimal de Rojo: #FF0000
                                                   Código hexadecimal de Amarillo: #FFFF00
                                                   Código hexadecimal de Naranja: #FFA500?>
var_dump($colores);
// Accediendo a los elementos
echo "<br/>br>Código hexadecimal de Rojo: " . $colores[$tipoPrimarios]["Rojo"];
echo "<br>
Código hexadecimal de Amarillo: " . $colores[$tipoSecundarios]["Amarillo"];
echo "<br/>br>Código hexadecimal de Naranja: " . $colores[$tipoSecundarios]["Naranja"];
?>
```

La función count()
se utiliza para contar
todos los elementos
de un array

```
Descripción de count()
count ( mixed $array_or_countable, int $mode = COUNT_NORMAL) :int
Parámetros
```

array_or_countable: Este es el array o el objeto que deseas contar. Puede
ser un array simple o multidimensional, o un objeto que implemente la
interfaz Countable.

mode (opcional): Este parámetro determina cómo se realizará el conteo:

- ► COUNT_NORMAL (valor por defecto): Cuenta solo los elementos del primer nivel del array.
- COUNT_RECURSIVE (o 1): Cuenta todos los elementos de un array de forma recursiva, es decir, cuenta también los elementos de los arrays internos.

La función count() se utiliza para contar todos los elementos de un array

La función sizeof() es un alias de count

```
Ejemplo 1
<?php
$arraySimple = array("Rojo", "Verde", "Azul");
$conteo = count($arraySimple);
echo "Número de elementos en el array simple: " . $conteo; // Salida: 3
?>
Ejemplo 2
<?php
$arrayMultidimensional = array(
    "primarios" => array("Rojo", "Verde", "Azul"),
    "secundarios" => array("Amarillo", "Naranja", "Morado")
);
// Contar solo el primer nivel
$conteoNormal = count($arrayMultidimensional);
echo "Número de elementos en el primer nivel: " . $conteoNormal; // Salida: 2
// Contar de forma recursiva
$conteoRecursivo = count($arrayMultidimensional, COUNT RECURSIVE);
echo "<br>Número total de elementos (recursivo): " . $conteoRecursivo; // Salida: 8
```

La función is_array()

```
Descripción de is_array()
is_array ( mixed $var ) : bool
```

La función is_array() es útil para determinar si una variable es un array antes de realizar operaciones sobre ella.

```
Ejemplo 1
<?php
$array = array("Rojo", "Verde", "Azul");
if (is_array($array)) {
    echo "Es un array."; // Salida: Es un array.
} else {
    echo "No es un array.";
}
</pre>
```

Recorriendo arrays con for y while

```
<?php
$colores = array(
    0 => array("Rojo", "Verde", "Azul"),
    1 => array("Amarillo", "Naranja", "Morado")
// Contar el número de elementos en el array principal $colores
$cantidadElementos = count($colores);
// Recorrer cada elemento del array principal $colores
for ($indice = 0; $indice < $cantidadElementos; $indice++) {</pre>
    // Obtener el array interno correspondiente al índice actual
    $elemento = $colores[$indice];
    // Contar el número de valores en el array interno
    $cantidadValores = count($elemento);
    // Recorrer cada valor del array interno
    for ($j = 0; $j < $cantidadValores; $j++) {</pre>
        // Imprimir cada valor del array interno
        echo " - " . $elemento[$j] . "<br>";
```

Recorriendo arrays con for y while

```
<?php
$colores = array(
    0 => array("Rojo", "Verde", "Azul"),
    1 => array("Amarillo", "Naranja", "Morado")
);
$cantidadElementos = count($colores);
$indice = 0; // Inicializar el índice para el bucle while
while ($indice < $cantidadElementos) {</pre>
    $elemento = $colores[$indice];
    $cantidadValores = count($elemento);
    $j = 0; // Inicializar el índice para el bucle while interno
    while ($j < $cantidadValores) {</pre>
        echo " - " . $elemento[$j] . "<br>"; // Imprimir cada valor del array interno
        $j++; // Incrementar el índice del bucle while interno
    $indice++; // Incrementar el índice del bucle while externo
```

Definición

El bucle foreach() en PHP es una estructura de control que permite iterar sobre los elementos de un array. Es especialmente útil cuando se trabaja con arrays, ya que elimina la necesidad de manejar índices manualmente.

Cómo Funciona foreach

Cuando usas foreach, PHP internamente gestiona la iteración.

No es necesario calcular el tamaño del array con count(), ya que <u>el bucle se</u> <u>detiene automáticamente al llegar al final del array.</u>

Esto significa que:

- Iteración automática: foreach recorre todos los elementos del array hasta que no quedan más elementos. No es necesario especificar un límite.
- No requiere índices: A diferencia de un bucle for, donde debes usar índices para acceder a los elementos, en foreach simplemente accedes directamente a cada valor.

Opciones en el uso de foreach()

1. Lectura de valores

En esta forma, el bucle foreach recorre cada elemento del array y asigna su valor a una <u>variable temporal</u>. No se accede a los índices del array, solo a los valores.

```
<?php
foreach ($array as $valor) {
    // $valor contiene el valor del elemento actual del array
    ...instrucciones...
}</pre>
```

2. Lectura de índices y valores

El bucle foreach recorre cada elemento del array y asigna tanto la clave (índice) como el valor a dos variables temporales. Esto es útil cuando necesitas tanto el índice como el valor del array.

```
<?php
foreach ($array as $indice => $valor) {
    // $indice contiene la clave del elemento actual del array
    // $valor contiene el valor del elemento actual del array
    ...instrucciones...
}
```

Opciones en el uso de foreach()

2. Ejemplo

```
<?php
$colores = array(
    "primario" => "Rojo",
    "secundario" => "Verde",
    "terciario" => "Azul"
);

foreach ($colores as $tipo => $color) {
    echo "Tipo: $tipo, Color: $color<br>} // Imprime el tipo y el color
}
```

```
array (size=2)
  'frutas' =>
    array (size=2)
      'manzana' =>
       array (size=2)
         'color' => string 'rojo' (length=4)
          'sabor' => string 'dulce' (length=5)
      'limón' =>
        array (size=2)
          'color' => string 'amarillo' (length=8)
          'sabor' => string 'acido' (length=6)
  'verduras' =>
    array (size=2)
      'zanahoria' =>
        array (size=2)
          'color' => string 'naranja' (length=7)
          'sabor' => string 'dulce' (length=5)
      'espinaca' =>
        array (size=2)
          'color' => string 'verde' (length=5)
          'sabor' => string 'amargo' (length=6)
```

```
Categoría: frutas
Producto: manzana
color: rojo
sabor: dulce
Producto: limón
color: amarillo
sabor: ácido
Categoría: verduras
Producto: zanahoria
color: naranja
sabor: dulce
Producto: espinaca
color: verde
sabor: amargo
```

Opciones en el uso de foreach()

```
<?php
$productos = array(
    "frutas" => array(
        "manzana" => array("color" => "rojo", "sabor" => "dulce"),
        "limón" => array("color" => "amarillo", "sabor" => "ácido")
    ),
    "verduras" => array(
        "zanahoria" => array("color" => "naranja", "sabor" => "dulce"),
        "espinaca" => array("color" => "verde", "sabor" => "amargo")
foreach ($productos as $categoria => $listaProductos) {
    echo "Categoría: $categoría de la categoría (frutas o verduras)
    foreach ($listaProductos as $producto => $detalles) {
        echo " Producto: $producto <br>"; // Imprime el nombre del producto
        foreach ($detalles as $caracteristica => $valor) {
          echo " $caracteristica: $valor<br>"; // Imprime cada característica del producto
```

Ejemplo de foreach() mostrando funcionamiento de isset()

```
<?php
//isset(): Verifica si una variable está definida y no es null.
$valores = array(
                                                        isset(cadena vacía): true
     "cadena vacía" => "",
                                                        isset(cadena '0'): true
     "cadena '0'" => "0",
                                                        isset(entero 0): true
     "entero 0" => 0,
                                                        isset(flotante 0.0): true
    "flotante 0.0" => 0.0,
                                                        isset(cadena '0.0'): true
     "cadena '0.0'" => "0.0",
                                                        isset(null): false
     "null" => null,
                                                        isset(false): true
    "false" => false,
     "array vacío" => array(),
                                                        isset(array vacío): true
     "cadena no vacía" => "valor",
                                                        isset(cadena no vacía): true
    "entero no cero" => 1,
                                                        isset(entero no cero): true
    "flotante no cero" => 1.5,
                                                        isset(flotante no cero): true
     "array no vacío" => array(1, 2, 3)
                                                        isset(array no vacío): true
foreach ($valores as $descripcion => $valor) {
    echo "<br>isset($descripcion): " . (isset($valor) ? 'true' : 'false') . "<br>";
```

Ejemplo de foreach() mostrando funcionamiento de empty()

<?php

```
//La función empty() devuelve true para una variedad de valores considerados vacíos, incluyendo el número 0, una cadena vacía, un string '0', un valor null, un valor booleano false, un array vacío y un string '0.0'.
```

```
$valores = array(
                                                        empty(cadena vacía): true
     "cadena vacía" => "",
                                                        empty(cadena '0'): true
     "cadena '0'" => "0",
                                                        empty(entero 0): true
     "entero 0" => 0,
                                                        empty(flotante 0.0): true
     "flotante 0.0" => 0.0,
                                                        empty(cadena '0.0'): false
     "cadena '0.0'" => "0.0",
                                                        empty(null): true
     "null" => null,
                                                        empty(false): true
     "false" => false,
                                                        empty(array vacío): true
     "array vacío" => array(),
                                                        empty(cadena no vacía): false
     "cadena no vacía" => "valor",
                                                        empty(entero no cero): false
     "entero no cero" => 1,
                                                        empty(flotante no cero): false
     "flotante no cero" => 1.5,
                                                        empty(array no vacío): false
     "array no vacío" \Rightarrow array(1, 2, 3)
foreach ($valores as $descripcion => $valor) {
    echo "<br>empty($descripcion): " . (empty($valor) ? 'true' : 'false') . "<br>";
```

Función array_keys()

La función array keys devuelve todas las claves de un array. Si se proporciona un valor opcional, solo se devolverán las claves que coincidan con ese valor.

?>

```
Sintaxis:
array keys(array $array, mixed $filter value, bool $strict = false): array
    o $array: El array del cual se obtendrán las claves.
    o $search_value (opcional): Si se especifica, solo se devolverán las claves que
      coincidan con este valor.
    o $strict (opcional): Si es true, la comparación será estricta (===).
Ejemplo de array keys
    <?php
    $frutas = array(
        "manzana" => "roja",
        "naranja" => "naranja",
        "plátano" => "amarillo"
                                                         Array
    );
                                                             [0] => manzana
                                                             [1] => naranja
    // Obtener las claves del array asociativo
                                                             [2] => plátano
    $claves = array keys($frutas);
    // Imprimir las claves
    print r($claves);
```

Función array_values()

```
La función array values() devuelve todos los valores de un array
y reindexa el array numéricamente
Sintaxis:
array_values(array $array): array
Ejemplo
<?php
$frutas = array(
    "manzana" => "roja",
    "naranja" => "naranja",
                                              Array
    "plátano" => "amarillo"
);
                                                 [0] => roja
                                                 [1] => naranja
                                                 [2] => amarillo
$valores = array_values($frutas);
print_r($valores);
?>
```

Ordenando arrays

- **sort()**: Ordena los elementos de un array en orden ascendente. Reindexa los indices, perdiendo la asociación original de claves y valores en arrays asociativos.
- **rsort()**: Ordena los elementos de un array en orden descendente. Reindexa los índices, perdiendo la asociación original de claves y valores en arrays asociativos.
- **asort()**: Ordena los valores de un array asociativo en orden ascendente, <u>manteniendo la asociación de índices</u>.
- **arsort()**: Ordena los valores de un array asociativo en orden descendente, manteniendo la asociación de índices.
- **▶ksort()**: Ordena un array asociativo por sus claves en orden ascendente.
- krsort(): Ordena un array asociativo por sus claves en orden descendente.

Ordenando arrays

Reindexación de índices:

- sort(): Reindexa los índices del array, perdiendo los índices originales.
- asort(): Mantiene los índices originales del array.

Uso típico:

- sort(): Se usa principalmente para arrays numéricos o cuando los índices no son importantes.
- asort(): Se usa principalmente para arrays asociativos o cuando es importante mantener la asociación de índices.

Ordenando arrays sort() asort()

```
<?php
$arrayNumerico = array(3, 1, 4, 1, 5, 9);
$arrayAsociativo = array("a" => 3, "b" => 1, "c" => 4,
"d" \Rightarrow 1, "e" \Rightarrow 5, "f" \Rightarrow 9);
echo "";
echo "Usando sort():<br>";
sort($arrayNumerico);
print_r($arrayNumerico);
echo "Usando asort():<br>";
asort($arrayAsociativo);
print r($arrayAsociativo);
?>
```

```
Usando sort():
Array
    [0] => 1
    [1] => 1
    [2] => 3
    [3] \Rightarrow 4
    [4] => 5
    [5] => 9
Usando asort():
Array
     [b] => 1
    [d] => 1
     [a] => 3
     [c] => 4
     [e] => 5
    [f] => 9
```

Ordenando arrays sort()

```
<?php
//usando sort() con array asociativo
//Reindexa el array asociativo perdiendo las claves originales
$arrayAsociativo = array("a" => 3, "b" => 1, "c" => 4, "d"
=> 1, "e" => 5, "f" => 9);
echo "";
echo "Array original:<br>";
                                                         Array original:
                                                         Array
print r($arrayAsociativo);
                                                            [a] => 3
                                                            [b] => 1
// Aplicar sort() al array asociativo
                                                            [c] \Rightarrow 4
sort($arrayAsociativo);
                                                            [d] => 1
                                                            [e] => 5
                                                            [f] \Rightarrow 9
echo "Array después de aplicar sort():<br>";
                                                         Array después de aplicar sort():
print_r($arrayAsociativo);
                                                         Array
?>
                                                            [0] => 1
                                                            [1] => 1
                                                            [2] => 3
                                                            [3] => 4
                                                            [4] => 5
                                                            [5] => 9
```

Ordenando arrays ksort()

```
<?php
// Ordena el array por sus claves en orden ascendente,
manteniendo la asociación de índices.
$arrayAsociativo = array("c" => 4, "a" => 3, "f" =>
9, "b" \Rightarrow 1, "e" \Rightarrow 5, "d" \Rightarrow 1);
echo "";
echo "Array original:<br>";
print_r($arrayAsociativo);
                                                            Array original:
                                                            Array
// Aplicar ksort() al array asociativo
                                                               [c] \Rightarrow 4
ksort($arrayAsociativo);
                                                               [a] => 3
                                                               [f] => 9
                                                               [b] => 1
echo "Array después de aplicar ksort():<br>";
                                                               [e] => 5
print r($arrayAsociativo);
                                                               [d] => 1
?>
                                                            Array después de aplicar ksort():
                                                            Array
                                                               [a] => 3
                                                               [b] => 1
                                                               [c] \Rightarrow 4
                                                               [d] => 1
                                                               [e] => 5
                                                               [f] \Rightarrow 9
```

Añadir y eliminar elementos a un array

- array_push(): Añade elementos al final del array.
- array_unshift(): Añade elementos al inicio del array.
- Asignación directa: Añade elementos directamente usando índices.
- unset(): Elimina el elemento indicado

Añadir y eliminar elementos en un array

```
<?php
echo "";
// Ejemplo de array_push()
\frac{1}{2} \frac{1}{2}
array_push($arrayPush, 4, 5);
echo "Usando array_push():<br>";
print_r($arrayPush);
// Ejemplo de array unshift()
$arrayUnshift = array(1, 2, 3);
array_unshift($arrayUnshift, -1, 0);
echo "Usando array unshift():<br>";
print_r($arrayUnshift);
// Ejemplo de asignación directa
$arrayDirect = array(1, 2, 3);
$arrayDirect[] = 4;
$arrayDirect[10] = 5;
echo "Usando asignación directa:<br>";
print r($arrayDirect);
// Ejemplo de unset()
$arrayUnset = array(1, 2, 3, 4, 5);
// Elimina el elemento en el índice 2
unset($arrayUnset[2]);
echo "Usando unset():<br>";
print r($arrayUnset);
?>
```

```
Usando array push():
Array
     [0] => 1
    [1] \Rightarrow 2
     [2] => 3
    [3] \Rightarrow 4
    [4] => 5
Usando array unshift():
Array
    [0] => -1
    [1] => 0
     [2] => 1
    [3] => 2
    [4] => 3
Usando asignación directa:
Array
     [0] => 1
    [1] => 2
     [2] => 3
    [3] => 4
    [10] => 5
Usando unset():
Array
     [0] => 1
     [1] => 2
     [3] => 4
    [4] => 5
```

La función substr() se utiliza para extraer una parte de una cadena.

Sintaxis:

```
substr(string $string, int $offset, ?int $length = null): string
```

Parámetros:

- > \$string: La cadena de la cual se extraerá la subcadena.
- \$offset: La posición de inicio de la subcadena. Si es negativo, la posición se cuenta desde el final de la cadena.
- > \$length (opcional): La longitud de la subcadena. Si es negativo, se omiten los caracteres al final de la cadena. Si se omite, se extrae hasta el final de la cadena.
- > ?int significa que el parámetro puede ser entero o NULL

```
<?php
echo '<br>Función substr(string $string, int $offset, ?int $length = null):
string<br>';
$cadena = 'abcdef';
echo '<br>Cadena de ejemplo: ' . $cadena . '<br>';
echo '<br>substr($cadena, 3): ' . substr($cadena, 3) . '<br>'; // "def"
echo '<br>substr($cadena, 0): ' . substr($cadena, 0) . '<br>'; // "abcdef"
echo '<br>substr($cadena, 10): ' . substr($cadena, 10) . '<br>'; // ""
echo '<br>substr($cadena, 3, 2): ' . substr($cadena, 3, 2) . '<br>'; // "de"
echo '<br>substr($cadena, 3, 10): ' . substr($cadena, 3, 10) . '<br>'; // "def"
echo '<br>substr($cadena, -3): ' . substr($cadena, -3) . '<br>'; // "def"
echo '<br>substr($cadena, -10): ' . substr($cadena, -10) . '<br>'; // "abcdef"
echo '<br>substr($cadena, -4): ' . substr($cadena, -4) . '<br>'; // "cdef"
echo "<br>Si el tercer parámetro es negativo, indica los caracteres que se
omiten al final de la cadena<br>";
echo '<br>substr($cadena, 3, -1): ' . substr($cadena, 3, -1) . '<br>'; // "de"
echo '<br>substr($cadena, 3, -3): ' . substr($cadena, 3, -3) . '<br>'; // ""
echo '<br>substr($cadena, 3, -5): ' . substr($cadena, 3, -5) . '<br>'; // ""
echo '<br>substr($cadena, -4, -2): ' . substr($cadena, -4, -2) . '<br>'; // "cd"
?>
```

Explicación de los ejemplos con dos parámetros:

- 1.substr(\$cadena, 3): Extrae desde la posición 3 hasta el final, resultado: "def".
- 2.substr(\$cadena, 0): Extrae desde la posición 0 hasta el final, resultado: "abcdef".
- 3.substr(\$cadena, 10): La posición 10 está fuera del rango, resultado: "" (cadena vacía).
- 4.substr(\$cadena, 3, 2): Extrae 2 caracteres desde la posición 3, resultado: "de".
- **5.substr(\$cadena, 3, 10)**: Extrae desde la posición 3 hasta el final (10 es mayor que la longitud de la cadena), resultado: "def".
- **6.substr(\$cadena, -3)**: Extrae los últimos 3 caracteres, resultado: "def".
- **7.**substr(\$cadena, -10): La posición -10 está fuera del rango, resultado: "abcdef".
- **8.substr(\$cadena, -4)**: Extrae los últimos 4 caracteres, resultado: "cdef".

Con tercer parámetro negativo:

- 1.substr(\$cadena, 3, -1): Extrae desde la posición 3 y omite el último carácter, resultado: "de".
- **2.substr(\$cadena, 3, -3)**: Extrae desde la posición 3 y omite los últimos 3 caracteres, resultado: "" (cadena vacía).
- **3.substr(\$cadena, 3, -5)**: Extrae desde la posición 3 y omite los últimos 5 caracteres, resultado: "" (cadena vacía).
- **4.substr(\$cadena, -4, -2)**: Extrae desde la posición -4 y omite los últimos 2 caracteres, resultado: "cd".

```
Función substr(string $string, int $offset, ?int $length = null): string
Cadena de ejemplo: abcdef
substr($cadena, 3): def
substr($cadena, 0): abcdef
substr($cadena, 10):
substr($cadena, 3, 2): de
substr($cadena, 3, 10): def
substr($cadena, -3): def
substr($cadena, -10): abcdef
substr($cadena, -4): cdef
Si el tercer parámetro es negativo, indica los caracteres que se omiten al final de la cadena
substr($cadena, 3, -1): de
substr($cadena, 3, -3):
substr($cadena, 3, -5):
substr($cadena, -4, -2): cd
```