RESUMEN DE PHP VII – Arrays

Un array es una colección de datos que podemos almacenar en una variable con un único nombre. Para acceder a los datos podemos utilizar índices numéricos o alfanuméricos. En PHP los arrays no se definen con un tipo base y pueden almacenar distintos tipos de variables en su interior, ya sean integer, cadenas u objetos. Además son dinámicos, esto quiere decir que no necesitamos definirlos con un tamaño como en la mayoría de lenguajes de programación, ya que van creciendo dinámicamente a medida que utilizamos nuevos índices.

Para crear un array podemos utilizar la función **array()** que crea un array con los datos que se le pasen como parámetros.

```php
<?php
	$numeros = array(1,2,3,4,5,6);
?>
```

En nuestro ejemplo hemos creado un array de índice numérico ya que para acceder a los datos de su interior utilizamos índices numéricos de esta forma:

```php
<?php
	$numeros = array(1,2,3,4,5,6);

	$numero1=numeros[0]; //$numero1=1
	$numero2=numeros[1]; //$numero2=2
	$numero3=numeros[2]; //$numero3=3
?>
```

La función **array()** también nos permite crear arrays asociativos para acceder a los datos utilizando claves alfanuméricas.

```php
<?php
	$agenda = array("Lunes" => "Estudiar PHP",
			"Martes" => "Estudiar CSS",
			"Miércoles" => "Examen de JavaScript",
			"Jueves" => "Estudiar SQL",
			"Viernes" => "Salir de fiesta");

	//Imprime la cadena: Hoy es Lunes, toca Estudiar PHP
	echo 'Hoy es Lunes, toca '.$agenda['Lunes'];
?>
```

En este ejemplo hemos almacenado en un array 5 par de claves/valor y hemos accedido al valor de la clave Lunes.

Arrays multidimensionales

Hasta ahora hemos utilizado un array unidimensional. PHP soporta arrays de varias dimensiones, aunque no suele utilizarse más allá de una tercera dimensión porque aumenta bastante la complejidad de manipularlos.

```php
<?php
	$tecnologias=array('web'=>array('programacion'=>'PHP','marcas'
=>'XHTML','presentacion'=>'CSS'),'escritorio'=>array('programacion'=>'
Java','otro' => 'C++'))
?>
```

Hemos creado un array de dos dimensiones. Para acceder a los datos podemos utilizar:

```php
<?php
	echo $tecnologias['web']['programacion']; //Imprime PHP
?>
```

Funciones de manipulación de arrays

La API de PHP tiene numerosas funciones de manipulación de arrays, sólo vamos a repasar algunas, pero podéis encontrar información más exhaustiva en la documentación oficial en línea de PHP.

💻 count(), sizeof()

Esta función cuenta el número de elementos de un array. También puede utilizarse la función **sizeof()** que funciona de la misma manera.

```php
<?php
	$agenda = array("Lunes" => "Estudiar PHP",
			"Martes" => "Estudiar CSS",
			"Miércoles" => "Examen de Java",
			"Jueves" => "Estudiar SQL",
			"Viernes" => "Salir de fiesta");

	echo count( $agenda ); //Imprime 5
?>
```

💻 in_array()

Esta función busca un valor pasado como argumento dentro del array pasado como argumento. Si lo encuentra devuelve true, en caso contrario, devuelve false.

```php
<?php
	$agenda = array("Lunes" => "Estudiar PHP",
			"Martes" => "Estudiar CSS",
			"Miércoles" => "Examen de Java",
			"Jueves" => "Estudiar SQL",
			"Viernes" => "Salir de fiesta");

	if (in_array("Salir de fiesta",$agenda))
		echo "En nuestra agenda hay un día para salir de fiesta";
?>
```

💻 unset()

Podemos utilizar esta función para borrar un dato del interior del array, así como el array entero.

```php
<?php
	unset($agenda['Viernes']);
	//Borramos de nuestra agenda el par clave/valor del Viernes.

	unset($agenda)
	//Borramos el array entero
?>
```

Recorrer un array mediante foreach

Cuando repasamos las estructuras de control me salté la estructura de control llamada foreach que se utiliza para recorrer arrays. Su sintaxis es la siguiente:

foreach($array as $valor)

o

foreach($array as $indice=>$valor)

Así podemos utilizar dentro del bucle las variables $indice y $valor que contendrán el índice y el valor respectivamente de cada elemento del array.

```php
<?php
	foreach( $agenda as $clave => $valor) {
		echo 'El día '.$clave.' toca '.$valor.'<br />'
	}
?>
```

En nuestro ejemplo recorremos el array asociativo $agenda. En la variable $clave almacenamos la clave de cada elemento y en la variable $valor almacenamos el valor de cada elemento del array asociativo.

Recorrer un array utilizando funciones

Los arrays también podemos recorrerlos utilizando una serie de funciones que manejan el puntero interno del array. El puntero inicialmente apunta al inicio del array.

La función **current()** devuelve el valor del elemento al que apunta el puntero. Podemos incrementar el puntero con la función **next()** y decrementarlo con la función **prev()**. También podemos situar el puntero al inicio del array con la función **reset()** o al final del array con la función **end().**

Si el puntero llega al final del array la función **next()** devuelve false.

```php
<?php
	$valor_inicial=current($agenda);
	echo $valor_inicial.'<br />'; //Imprime Estudiar PHP

	while( next($agenda) )
	{
		echo current($agenda).'<br />';
	}
	reset($agenda);
?>
```

En este ejemplo hemos recorrido el array *$agenda* de uno de los ejemplos anteriores utilizando el puntero interno y las funciones de recorrido. Podéis utilizar esta forma también, aunque es mucho más *elegante* recorrerlos con la estructura de control *foreach*.

Funciones para insertar o extraer elementos

array_push(), array_pop()

La función **array_push()** inserta un elemento al final del array. Análogamente, la función **array_pop()** extrae y devuelve el último elemento del array.

```
<?php
  $numeros=array(1,5,6,8,10,25);
  array_push($numeros, 2, 5, 6);
  echo array_pop($numeros); //Imprime 6
?>
```

Funciones para ordenar arrays

asort(), arsort(), ksort(), krsort(), sort(), rsort()

Esta serie de funciones se utilizan para ordenar arrays. La función **asort()** ordena los valores del array ascendentemente. La función **arsort()** ordena el array por valores descendentes.

La función **ksort()** ordena el array por claves ascendentes. La función **krsort()** ordena el array por claves descendentes.

La función **sort()** ordena el array ascendentemente con el inconveniente de que se pierde la asociatividad entre la clave y el valor, es decir si le pasamos un array asociativo de vuelve un array de índice numérico.

La función **rsort()** tiene el mismo inconveniente y ordena los valores descendentemente.

```
<?php
  $array_numerico=array("Lunes","Martes","Miercoles","Jueves","Vi
ernes"  );

  var_dump(sort($array_numerico));
  var_dump(rsort($array_numerico));

  $array_asociativo=array("Lunes" => "Lentejas",
                          "Martes" => "Macarrones",
                          "Miercoles" => "Spaguettis",
                          "Jueves" => "Albondigas",
                          "Viernes" => "Morcilla");

  var_dump(asort($array_asociativo));
  var_dump(arsort($array_asociativo));
  var_dump(ksort($array_asociativo));
  var_dump(krsort($array_asociativo));
?>
```

En este ejemplo, hemos ordenado un array de índice numérico y un array asociativo, y hemos mostrado el valor de los arrays mediante la función var_dump(). Esta función más que nada sirve para depuración de los códigos ya que muestra el contenido de cualquier variable pasada como parámetro.