Guía Básica del lenguaje PHP

Objetivo: Comprender el funcionamiento básico del lenguaje PHP.
Sitio oficial: http://www.php.net

ARREGLOS

Una de las estructuras de datos más poderosa que existe en PHP son los arreglos (matrices). Esto se debe a su flexibilidad en cuanto a su definición de claves y a la diversidad de datos que pueden almacenar (prácticamente todos los tipos de datos, incluyendo otras matrices).

En PHP la clave de un arreglo puede ser un valor entero, un auto-incremental, un valor alfanumérico o una mezcla de ambos.

Arreglo Básico

Un arreglo es una colección ordenada de items, llamados elementos. Cada elemento tiene un valor y es referenciado por una clave única que lo identifica dentro del arreglo. Como vimos en el párrafo anterior, las claves pueden ser números enteros o cadenas.

Los arreglos pueden ser creados de dos maneras. La primera de ellas es haciendo uso del constructor **array()**, al cual podremos pasar los valores y sus claves:

```
$valores1 = array (10, 20, 30);
$valores2 = array ('a' => 10,'b' => 20, 'cee' => 30);
$valores3 = array (5 => 1, 3 => 2, 1 => 3,);
$valores4 = array();
```

En la primer definición solamente especificamos los valores que va a contener el arreglo. De esta forma, PHP asignará automáticamente un valor numérico como clave comenzando a partir de cero. En el segundo ejemplo podemos observar que se especifican tanto las claves como los valores de cada elemento del arreglo. Como puede verse en la primer asignación, la expresión 'a'=>10 asignará un elemento cuya clave será 'a' y contendrá el valor 10. En el tercer ejemplo las claves son asignadas fuera de orden de manera que comenzamos por la clave 5 apuntando al valor 1, la clave 3 apuntando al valor 2 y así sucesivamente. Por último, en el cuarto ejemplo

creamos un arreglo vació.

Otra forma de definir un arreglo es hacer uso de los **[]**. Para esto bastará:

```
$valores1[]= 10;
$valores1[]= 20;
$valores2['aa']= 10;
```

La primer definición forzará a la variable $valores1 a tomar la estructura de array y comenzará a asignar los valores generando para ellos claves auto-incrementalos comenzando a partir del valor cero. En el segundo ejemplo forzará la estructura de array para la variable $valores2 y creará el elemento 'aa'=>10 (clave=>valor).

Mostrando Arreglos

En apartados anteriores vimos como la instrucción echo nos permite mostrar el valor de una variable o de una constante. De la misma forma podremos mostrar los valores de un arreglo. La diferencia radicará en que tendremos que indicar la clave del valor a mostrar.

```
<?php
    $dias = array ('Domingo', 'Lunes', 'Martes', 'Miércoles', 'Jueves', 'Viernes', 'Sábado');
    echo $dias[1];
?>
```

La salida en pantalla sería: // **Lunes**

Básicamente el echo nos permitirá mostrar los valores de un arreglo, pero nos presentará algunas limitaciones en cuanto a la depuración de los mismo. Para salvar esta situación aparece el la función **print_r()**.
La función **print_r()** nos listará de manera completa el contenido y la estructura de un arreglo.

```
<?php
    $frutas = array ('a'=>'Anana', 'b'=>'Banana', 'm'=>'Manzana');
    print_r ($frutas);
?>
```

La salida en pantalla sería:

```
Array(
        [a] => Anana
        [b] => Banana
        [m] => Manzana
)
```

Como se dijo anteriormente, un arreglo pude contener cualtier tipo de datos, inclusive otro arreglo. Si tomásemos una definición un poquito mas compleja:

```
<?php
        $frutas = array ('a'=>'Anana', 'b'=>'Banana', 'otros'=>array ('x', 'y','z'));
        print_r ($frutas);
?>
```

La salida en pantalla sería:

```
<?php
        Array(
                [a] => Anana
                [b] => Banana
                [otros] => Array(
                                [0] => x
                                [1] => y
                                [2] => <
                 )
        )
?>
```

Si bien echo() y print_r() presentan los datos de una manera diferente, dependerá de lo que necesite hacer el uso de uno u otro. A grandes rasgos, echo() dará una buena solución a la hora de presentar datos en pantalla, mientras que print_r() dará una muy buena opción de depuración de sus arreglos.

Arreglos Multi-dimensionales

Como un arreglo puede contener todo tipos de datos, inclusive otros arreglos, la creación de arreglos multi-dimensionales es muy sencilla. Simplemente bastará con asignar un arreglo como el valor del elemento de otro arreglo. En PHP podremos hacer esta asignación para todos los valores, lo que permitirá infinitos niveles de anidamiento.

```
<?php
        $array = array();
        $array[] = array( 'hola', 'chau' );
        $array[] = array( 'estas', 'como' );
        echo $array[0][1] . $array[1][0];
?>
```

La salida de este comando daría como resultado **chauestas**. Como se puede ver, para tener acceso a arreglos multi-dimensionales bastará con ir indicando las claves hasta llegar a referenciar el valor que queremos recuperar.

Operaciones básicas con arreglos

Para determinar el tamaño de un arreglo, haremos uso de la función count(). De esta forma:

```
<?php
        $a = array ('a', 'b', 'c');
        $b = array();
        $c = 10;
        echo count ($a); // Resultado 3
        echo count ($b); // Resultado 0
        echo count ($c); // Resultado 1
?>
```

La función **count()** no se puede utilizar para determinar si una variable es un arreglo, ya que la sola ejecución devolverá uno. La forma correcta de averiguar si una variable es un arreglo

es utilizar la función is_array() en su lugar.

```
<?php
        $a = array ('a', 'b', 'c');
        if (is_array($a))
        {
                echo “Es un array()!!!!!”;
        }
?>
```

Por otro lado si quisiésemos saber si una clave existe dentro del arreglo, podríamos hacer uso de la función **array_key_exist()** o **isset()**:

```
<?php
        $a = array ('a', 'b', 'c');
        if ( array_key_exists ($a['a']) )
        {
                echo “Existe una clave 'a' dentro del arreglo.”;
        }
        if ( isset ($a['b']) )
        {
                echo “Existe una clave 'b' dentro del arreglo.”;
        }
        if (isset ($a['5']))
        {
                echo “No existe una clave '5' dentro del array.”;
        }
?>
```

El problema que trae el uso de isset() es que no contempla al NULL como un elemento de un arreglo. Así si tuviésemos:

```
<?php
        $a = array ('a' => NULL, 'b' => 2);
        echo isset ($a['a']); // Resultado False
?>
```

Otra función muy importante dentro del manejo de arreglos es la búsqueda de valores en el mismo. Para esto, PHP nos ofrece la función in_array():

```
<?php
        $a = array ('a' => NULL, 'b' => 2);
        echo in_array (2,$a); // Resultado True
?>
```

Por último, podemos borrar un elemento de un arreglo haciendo uso de la función unset():

```
<?php
        $a = array ('a' => NULL, 'b' => 2);
        unset ($a['b']);
        echo in_array (2,$a); // Resultado False
?>
```

Iteraciones con Arreglos

Recorrer arreglos es una de las operaciones que más llevaremos a cabo cuando trabajemos con ellos. Inicialmente podemos hacer uso de las estructuras que vimos anteriormente para recorrer los mismo. Por ejemplo con for():

```
<?php
        $a = array ('Domingo','Lunes','Martes','Miércoles','Jueves','Sábado','Domingo');
        for ( $i=0;$i<count($a);$i++ )
```

```
        {
                echo “Día “. $i .$a[$i] ;
        }
?>
```

o haciendo uso del while():

```
<?php
        $a = array ('Domingo','Lunes','Martes','Miércoles','Jueves','Sábado','Domingo');
        while ($i<=count($a)){
                echo “Día “. $i .$a[$i] ;
                $i++;
        }
?>
```

Ahora, dado que en PHP los arreglos pueden contener distintos tipos de valores como clave, se necesitará de funciones especiales que correspondan a dicha flexibilidad. Por ejemplo, si quisiésemos utilizar for() o while() para listar el siguiente arreglo:

```
$arreglo = array ('a' => 10, 10 => 20, 'c' => 30);
```

nos encontraremos con que ninguna de ellas nos permitirá recorrer la estructura de dicho arreglo. Esto se debe a la diversidad que encontramos en la definición de las claves. Es para este tipo de situaciones que PHP nos brinda la función **foreach()**.

```
<?php
        $arreglo = array ('a' => 10, 10 => 20, 'c' => 30);
        foreach ($arreglo as $elemento){
                echo $elemento.'-';     // Resultado 10 – 20 - 30
        }
?>
```

Es la estructura iterativa más sencilla para el recorrido de los arreglos. Funciona de manera similar a la estructura for() pero su diferencia radica en que ejecutará el bucle tantas

veces como claves posea un arreglo. De otra forma:

```
<?php
        $arreglo = array ('a' => 10, 10 => 20, 'c' => 30);
        foreach ( $arreglo as $key => $valor )
        {
                echo $key.'-->'.echo $valor;
                echo "<br>";
        }
?>
```

nos dará como resultado:

```
a -->10
10 -->20
c-->30
```

Ordenar Arreglos

PHP nos ofrece varias funciones para ordenar arreglos cada una de ellas presentando diversa flexibilidad en cuanto a la complejidad de las tareas a llevar a cabo. Comenzando por lo básico encontraremos las funciones **sort()** y **asort()**.

```
<?php
        $arreglo = array ('a' => 'aab', 'b' => 'bbb', 'c' => 'aaa');
        sort($arreglo);
        print_r($arreglo);
?>
```

el resultado nos dará:

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

```
        [1] => aab
        [2] => bbb
    )
```

Como se puede ver sort() ordena el arreglo que le pasamos como parámetro, modificando al mismo a nivel de claves. Esto quiere decir que destruirá toda las claves que contenga y generará uno nuevo arreglo numerando los elementos a partir del 0.

Si quisiésemos mantener la relación de claves dentro del mismo, deberemos hacer uso de la función **asort()**.

```
<?php
    $arreglo = array ('a' => 'aab', 'b' => 'bbb', 'c' => 'aaa');
    asort($arreglo);
    print_r($arreglo);
?>
```

el resultado nos dará:

```
Array(
        ['c'] => aaa
        ['b'] => aab
        ['a'] => bbb
)
```

Ambas funciones **sort()** y **asort()** aceptan un segundo parámetro mediante el cual podremos indicar como queremos que se realice el orden:

SORT_REGULAR: ordena los elementos sin realizar ninguna transformación previa.
SORT_NUMERIC: convierte cada elemento a un valor numérico y luego los ordena.
SORT_STRING: ordena todos los elementos tratándolos como string

Nota: sort() y asort() ordenan de forma ascendente todo el arreglo que le pasemos como parámetro. Si necesitásemos ordenarlo de forma descendente, deberemos de hacer uso de las funciones rsort() y arsort().

Ahora, si lo que quisiésemos es ordenar un arreglo teniendo en cuenta sus claves, podremos hacer uso de las funciones ksort() y krsort().

```
<?php
        $arreglo = array ('d' => 'aab', 'c' => 'bbb', 'a' => 'aaa');
        ksort($arreglo);
        print_r($arreglo);
?>
```

el resultado nos dará:

```
        Array(
                ['a'] => aaa
                ['c'] => bbb
                ['d'] => aab
        )
```

Funciones entre Arreglos

Veremos en este apartado algunas funciones que nos provee PHP para interactuar entre arreglos. Dos operaciones comunes al trabajar con dos arreglos serían obtener la diferencia y la intersección entre ambos. Para esto existen las funciones array_diff () y array_intersect ().
Si necesitásemos obtener los elementos no coincidentes entre dos arreglos:

```
<?php
        $a = array (1,2,3);
        $b = array(1,3,6)
        print_r(array_diff($a,$b));
?>
```

el resultado nos dará:

```
        Array(
                [0 ] => 2
        )
```

Si quisiésemos los elementos que coinciden en ambos arreglos:

```
<?php
        $a = array (1,2,3);
        $b = array(1,3,6)
        print_r(array_intersec($a,$b));
?>
```

el resultado nos dará:

```
Array(
        [0 ] => 2
        [1 ] => 3
)
```

STRING

Una parte esencial de PHP es la manipulación y presentación en pantalla de cadenas de texto (strings). En PHP una cadena es cualquier conjunto de caracteres entrecomillados. PHP considera como cadena todo lo que encuentre entre un par de comillas, por eso todas las cadenas deben comenzar y terminar con el mismo tipo de comillas, simples o dobles.

Sintaxis Básica

Las cadenas pueden ser definidas por varios métodos. Comúnmente se encierran en comillas simples o dobles. A diferencia de otros lenguajes, estos dos métodos se comportan de manera muy diferente: las comillas simples representan "cadenas simples," donde casi todos los caracteres son utilizados literalmente. Las comillas dobles, por otra parte, encapsulan "cadenas complejas" que permiten las secuencias de escape especiales (por ejemplo, para insertar caracteres especiales) y para la sustitución de variables, que permite integrar el valor de una variable directamente en una cadena, sin la necesidad de que cualquier operador especial.

```
echo “Ejemplo con caracteres de escape \n”;
```

Otra característica que presenta la definición de strings por medio de las comillas dobles es que nos permite incluir el valor de una variable directamente dentro de la misma.

```
<?php
    $nombre =”Martín”;
    echo “Saludos, $nombre\n”   // Muestra: “Saludos, Martín” y baja de linea
    echo 'Saludos, $nombre\n'   // Muestra: “Saludos, $nombre\n”
?>
```

Escapando Caracteres Especiales

Dado el método que utilicemos para definir el string, encontraremos una u otra forma de escapar los caracteres especiales.
Cuando utilizamos comillas simples, los caracteres se pueden escapar mediante la barra invertida:

```
echo 'Escapamos caracteres \'con\' la barra invertida. ';
```

De manera similar podríamos hacer uso de la barra invertida para escapar el signo $ utilizados en las variables, de forma tal que podamos lograr cadenas de la siguiente forma:

```
<php
    $valor = 15;
    echo “El valor de la variable \$valor es \”$valor\”.”;
?>
```

Para los espacios en blanco, también podremos utilizar la barra invertida, anteponiendo esta a dicho espacio:

```
echo “En este ejemplo dejamos \  un espacio en blanco.”;
```

Longitud de los String

La función **strlen()** se utiliza para determinar la longitud en bytes de una cadena, esto significa que contará cada uno de los caracteres sin importar el tipo al que correspondan.

```
<php
        $cadena = “Contamos la cantidad de caracteres, aunque este el 10.”
        echo “La longitud de cadena es: “.strlen ($cadena); // La longitud de cadena es: 54
?>
```

Buscando dentro de los string

La función **strtr()** nos permite encontrar la primer ocurrencia de una secuencia de caracteres dentro de una cadena.

```
<?php
        $email  = 'contacto@gugler.com.ar';
        $dominio = strstr($email, '@');
        echo $domain; // Mostrará @gugler.com.ar
        $cuenta = strstr($email, '@', true);
        echo $user; // Mostrará contacto
?>
```

Tratando string como array()

Otra forma de manipular el contenido de un string que nos provee PHP es hacerlo como una lista ordenada de elemento, esto es, como un array.

```
<?php
        $cadena = 'contacto';
        echo $cadena[1]; // Mostrará a;
?>
```

También podríamos realizar cualquiera de las operaciones vista para arreglos:

```
<?php
        $cadena = “La cadena a mostrar.”;
        for ($i = 0; $i < strlen ($cadena); $i++) {
                if ($cadena [$i] > ’c’) {
                        echo $cadena[$i];
                }
        }
?>
```

Nota: al tratar un string como array, el primer carácter será indexado como 0. Del ejemplo anterior:

```
<?php
        $cadena = “La cadena a mostrar.”;
        echo $cadena[0]; // Mostrará L
?>
```

Comparar string

La comparación es una de las operaciones más comunes que se realizan en las cadenas. Sin embargo debemos tomar algunas precauciones al realizar las mismas. Veamos el siguiente ejemplo:

```
<?php
        $cadena = '123abc';
        if ($cadena ==123){
                El valor será true!!!
        }
?>
```

El resultado de esta comparación será true, ya que de forma transparente PHP transformará el valor 123 a su equivalente en integer, por lo que la comparación se realizará sobre

los valores numéricos idénticos. Para llevar a cabo comparaciones de este tipo, deberemos hacer uso del operador ===.

```
<?php
        $cadena = '123abc';
        if ($cadena ===123){
                El valor será false!!!
        }
?>
```

Existen funciones especializadas en realizar comparación de cadenas. Tanto strcmp() como strcasecmp() nos permitirán realizar este tipo de comparación. La diferencia entre ambas radica en que la primera no diferencia entre mayúsculas y minúsculas y la segunda si. En ambos caso, el resultado de las funciones será 0 si las cadenas a comparar son idénticas.

```
<?php
        $cadena = 'Hola a todos';
        if (strcmp($cadena, 'Hola a todos' )==0){
                echo “Las cadenas son idénticas”;
        }
        if (strcasecmp($cadena, 'HOLA a todos' )==0){
                echo “Las cadenas no son idénticas. Ojo con las mayúsculas!!!!!”;
        }
?>
```

Como una variante a estas funciones encontramos strcasencmp() la cual nos permitirá, dadas dos cadenas, definir el número de caracteres que deben coincidir.

```
<?php
        $cadena = 'Hola a todos';
        if (strcasencmp($cadena, 'Hola  a todos', 3 )==0)
        {
                echo “Las cadenas son idénticas. Se comparan los primeros 3 carcteres.”;
        }
```

```
        if (strcasencmp($cadena, 'HolA  a todos', 3 )==0)
        {
                echo “Las cadenas son idénticas. Se comparan los primeros 3 carcteres.”;
        }
?>
```

Búsquedas en string

A la hora de realizar búsquedas de cadenas, PHP nos ofrece una serie de mecanismo que van desde lo más sencillo a lo más complejo.

La forma más sencilla de buscar dentro de una cadena es utilizar las funciones strpos() y strstr(). La primera permite encontrar la posición de una subcadena dentro de una cadena. Como resultado nos devolverá la posición numérica del primer carácter de la ocurrencia o el valor 0 (falso) si no ser encontrada.

```
<?php
        $cadena = 'Hola a todos. Dentro de esta cadena realizaremos la búsqueda.';
        $buscar='realizaremos';
        if (strpos($cadena,$buscar) !== false)
        {
                echo “Se encontró la cadena buscada.”;
        }
?>
```

Debido a que las cadenas comienzan indexar los caracteres a partir del 0, será necesario que la comparación (!==) sea realizada de esta forma para asegurarnos que estamos hablando del valor false y no de la posición 0 del string.

Podemos indicar que se comiese a buscar a partir de una cierta posición:

```
<?php
        $cadena = 'abcdefghijkabcdefghijk';
        echo ( strpos($cadena,'abc'));                  // Resultado 0
```

```
        echo ( strpos($cadena,'abc',1));              // Resultado 11
?>
```

La función strstr() realiza una búsqueda de manera similar a strpos() con la diferencia que no devuelve la posición de la cadena a buscar sino que nos devuelve la cadena en sí.

```
<?php
        $cadena = '123456789';
        echo ( strstr($cadena,'45'));   // Resultado 456789
?>
```

La búsqueda de patrones con strstr() es más lenta y además, no permite indicar un punto de inicio para la búsqueda.

Hasta el momento, estas dos funciones son case-sensitive, por lo que nos serán de gran utilidad en un determinado tipo de casos. Sin embargo, PHP también nos ofrece una variedad de funciones para tratar búsquedas que no sean case-sensitive. Este es el caso de stripos() y stristr().

```
<?php
        $cadena = 'Hola para todos.';
        echo ( stripos($cadena,'hola'));        // Resultado 0
        echo ( stristr($cadena,'PaRa'));        // Resultado “para todos”
?>
```

Como se puede ver en el ejemplo, las dos funciones trabajan de manera idéntica a strpos() y strstr() pero sin tener en cuenta las mayúsculas y minúsculas.

Otro tipo de búsquedas que se puede dar es el caso en que se define una “lista blanca” de caracteres los cuales serán utilizados para la búsqueda. Cualquier cadena que contenga los caracteres especificados en la máscara será macheada como true. Las funciones en este caso son strspn() y strcspn().

Se puede utilizar el strspn() para machear una cadena contra de una "lista blanca" de caracteres permitidos. Esta función devolverá la longitud del segmento inicial de la cadena que

contenga cualquiera de los caracteres especificados en la máscara.

```
<?php
        $cadena = '1334455abcdef';
        $mask= '12345';
         strspn($cadena, $mask);      // Resultado 6
?>
```

La función strcspn() realizará la misma búsqueda pero interpretará los caracteres declarados en la mascara como “lista negra”, por lo que dará como resultado la longitud de cadena que no contenga ningún carácter declarado en la mask.

Ambas funciones aceptan dos parámetros. El primero nos permitirá indicar la posición de inicio de la búsqueda y el segundo el número de caracteres a analizar.

```
<?php
        $cadena = '1abc234';
        $mask= 'abc';
         echo strspn($cadena, $mask, 1, 4); // Resultado 3
?>
```

En este ejemplo comenzamos la búsqueda a partir del segundo carácter (a) y lo continuamos por cuatro caracteres más.

Reemplazar cadenas

Buscar y reemplazar cadenas es algo habitual dentro de las tareas de los desarrolladores. PHP nos ofrece funciones que, sensibles o no a mayúsculas y minúsculas, nos permitirán llevar a cabo estas operaciones. Entre ellas encontramos str_replace() (case-sensitive), str_ireplace() (no case-sensitive) y substr_replace().

```
<?php
        $cadena = 'En esta cadena reemplazaremos la cadena “en” por “EN”. ';
         echo str_replace('en','EN',$cadena,);
```

```
// En esta cadENa reemplazaremos la cadENa “EN” por “EN”.
?>
```

Si el mismo ejemplo lo realizaremos haciendo uso de str_ireplace(), la primer coincidencia del “En” se tomaría en cuenta.

```
<?php
        $cadena = 'En esta cadena reemplazaremos la cadena “en” por “EN”. ';
         echo str_replace('en','EN',$cadena,);
?>
// EN esta cadENa reemplazaremos la cadENa “EN” por “EN”.
```

Si necesitásemos pasar más de un argumento a la función, podríamos hacerlo haciendo uso de array():

```
<?php
        $cadena=”Debe comer frutas, vegetales y beber dos litros de agua cada dias.”
        $buscar = array('frutas','vegetales','agua');
        $reemplazar= array('milanesas','papas fritas','fernet');
        echo str_replace($buscar, $reemplazar, $cadena);
?>
// Usted debe comer milanesas, papas fritas  y beber dos litros de cerveza todos los días.
```

Si usted conoce la posición de inicio a partir de la cual debe comenzar a reemplazar caracteres dentro de una cadena, la función substr_replace() se será de gran ayuda:

```
echo substr_replace("Hola Martín", "Gonzalo", 6);
```

Además podemos indicar la posición final hasta donde deberemos reemplazar la cadena de caracteres:

```
echo substr_replace("Comer tomates hace muy bien!!!!", "milanesas", 6, 14);
```

Extraer cadenas

Otra tarea muy importante en el tratamiento de string es la extracción de cadenas. La función substr() nos dará una gran flexibilidad a la hora de realizar este tipo de operaciones. La función admite tres parámetros: la cadena de la cual extraeremos la subcadena, la posición de inicio y la cantidad de caracteres a extraer.

```
<?php
        $cadena = '123456abcdefgh';
        $numeros=substr($cadena, 0, 6);          // Resultado 123456
        $letras=substr($cadena, 7, 8);           // Resultado abcdefg
?>
```