RESUMEN DE PHP VI – Cadenas y Expresiones regulares

Los strings son cadenas de caracteres que se les puede asignar a una variable si se encierran entre comillas simples o dobles.

"Cadena entre comillas dobles"
'Cadena entre comillas simples'

PHP interpreta de distinta forma las cadenas que van entre comillas dobles y las que van entre comillas simples. Los strings entre comillas dobles pueden sustituir el valor de las variables. Las comillas simples, simplemente muestran todo el contenido, sin atender a las variables. Cuando utilizamos simplemente cadenas es más eficiente encerrarlas entre comillas simples ya que evitamos que el motor de PHP tenga que evaluar la cadena en busca de variables en las que sustituir por su valor.

```php
<?php
        $nombre = "Francisco";
        $frase1 = "Hola $nombre";
        $frase2 = 'Hola $nombre';

        //Imprime Hola Francisco
        echo $frase_1;

        //Imprime Hola $nombre
        echo $frase_2;
?>
```

Propiedades de los strings

Los strings son cadenas de caracteres, así podemos acceder a cualquier carácter de la cadena mediante los símbolos de llave (**{}**), y un índice numérico. Podemos observar en el ejemplo que el primer carácter de la cadena posee el índice cero.

```php
<?php
        $nombre = "Francisco";

        //Imprime F
        echo $nombre{0};

        //Imprime r
        echo $nombre{1};
?>
```

Operadores

Como en otros lenguajes de programación podemos utilizar un operador para concatenar cadenas, en PHP las cadenas se concatenan con el operador punto (**.**).

```php
<?php
        $nombre = "Francisco";

        //Imprime Hola Francisco
        $frase = 'Hola '.$nombre;
?>
```

Sintaxis para múltiples líneas

Existe en PHP una forma más de introducir cadenas, aparte de las comillas, muy recomendable para largos textos. La sintaxis es muy sencilla y comienza con el operador (<<<) seguido de una etiqueta que indica el principio del texto (**sintaxis heredoc**). Después de esto podemos escribir un conjunto de caracteres muy numerosos y, para finalizar, la etiqueta de fin.

```
<?php
        $cadena= <<<EOF
                Hola, soy una cadena que
                Contiene multiples lineas.
                Para ello utilizo la sintaxis heredoc
                que permite escribir cadenas muy largas
                mientras no incluyamos la palabra que he utilizado
                en este caso como inicio.
        EOF;
        echo $cadena;
?>
```

Funciones predefinadas en PHP para manipular cadenas

PHP cuenta con un conjunto amplio de funciones para realizar todo tipo de manipulaciones con las cadenas de caracteres. Vamos a ver unas cuantas. Para mayor información podéis acudir como siempre recomiendo a la documentación oficial en línea de PHP, donde están definidas todas las funciones que engloba PHP.

strlen()

La función **strlen()** devuelve como resultado el número de caracteres de la cadena que pasemos como parámetro.

```
<?php
     $cadena = "Hola mundo";
     $numero = strlen($cadena);

     //Imprime: La variable $cadena tiene 10 caracteres
     echo 'La variable $cadena tiene '.$numero.' caracteres';
?>
```

strpos()

La función **strpos()** encuentra en una cadena de caracteres la primera ocurrencia de un carácter determinado o una cadena de caracteres. Devuelve la posición donde se encuentra, y si no se encuentra devuelve el booleano false.

```
<?php
     $cadena = "Hola, me llamo Juanjo y esto es un resumen de PHP";

     //Imprime: La primera ocurrencia de Juanjo es 15.
     echo "La primera ocurrencia de Juanjo es ".strpos($cadena,
Juanjo).'<br />';
?>
```

Si lo que queremos es empezar a buscar la cadena o el carácter desde el último carácter podemos utilizar la función **strrpos().**

strcmp()

Con el operador == podemos evaluar si dos cadenas de caracteres son iguales. Pero también podemos utilizar la función **strcmp()** que compara dos cadenas pasadas como parámetros y devuelve un valor en función de la comparación. Si el valor que se obtiene es 0, las cadenas son exactamente iguales. Si el valor que se obtiene es <0, el primer string es menor que el segundo. Si el valor que se obtiene es >0 el primer string es mayor que el segundo.

Si queremos que no se distingan entre mayúsculas y minúsculas podemos utilizar la función **strcasecmp().**

```php
<?php
	$cadena1 = "Hola";
	$cadena2 = "Hola";

	if (strcmp($cadenal,$cadena2) == 0) {
		echo "Las dos cadenas son iguales";
	} elseif (strcmp($cadenal,$cadena2) < 0) {
		echo "La cadena1 es menor que la cadena2";
?>
```

En nuestro ejemplo, se imprimirá la cadena: Las dos cadenas son iguales.

strstr()

La función **strstr()** toma como parámetro dos cadenas de caracteres, el primer string es el string donde queremos buscar el segundo. Si lo encuentra devuelve una cadena de caracteres que comienza justo en el string encontrado hasta el final del primer string. Puede que la explicación sea un poco liosa, pero vamos a verlo más facilmente con un ejemplo.

```php
<?php
	$cadena1 = "Hola Francisco, ¿qué tal estás?";
	$cadena2 = "Francisco";

	//imprime: Francisco, ¿qué tal estás?
	echo (strstr($cadena1,$cadena2));
?>
```

substr()

La función substr() permite seleccionar un conjunto de caracteres de una cadena sin modificar el string original. Puede tomar varios parámetros:

- substr (cadena, índice): Cadena es el conjunto de caracteres que queremos cortar e índice la posición a partir de la cual se cortará la cadena hasta el final.

⇨ substr (cadena, índice, número): Cadena es el conjunto de caracteres que queremos cortar e índice la posición a partir de la cual se cortará la cadena tantas posiciones como indique el número.

```php
<?php
  $cadena1 = "Hola Francisco, ¿qué tal estás?";

  //imprime: Hola Francisco, ¿qué tal estás?
  echo (substr($cadena1,0));

  //imprime: Francisco
  echo (substr($cadena1,5,9));
?>
```

💻 ltrim(), chop(), trim()

Estas funciones borran cualquier espacio en blanco al principio, al final, o a ambos respectivamente de la cadena que se pasa como parámetro. Se utilizan por ejemplo, cuando el usuario introduce datos desde un formulario y queremos comprobar que no haya insertado ningún espacio en blanco por equivocación.

💻 str_replace()

Esta función busca una cadena de caracteres en otra cadena de caracteres y reemplaza las coincidencias encontradas por otra cadena de caracteres. Se utilizan por ejemplo para sustituir Smilies ☺ por las imágenes correspondientes.

```php
<?php
     $cadena1 = "Hola Francisco, ¿qué tal estás?";

     //imprime: Hola Paco, ¿qué tal estás?
     echo (str_replace('Francisco', 'Paco', $cadena1));
?>
```

Si en la cadena existe más de una instancia del patrón buscado se reemplaza en todas las coincidencias.

💻 strtolower(), strtoupper()

Estas dos funciones cambian la cadena pasada como parámetro a minúsculas o a mayúsculas respectivamente.

```php
<?php
     $cadena1 = "FrAnCiScO";

     //imprime: francisco
     echo (strtolower($cadena1));

     //imprime: FRANCISCO
     echo (strtoupper($cadena1));
?>
```

Expresiones regulares

Las funciones que hemos visto de manipulación de cadenas son útiles cuando tenemos que buscar o sustituir caracteres, pero son poco útiles cuando tenemos que comprobar que una cadena siga un patrón determinado. Por ejemplo, imaginaros que tenemos un formulario donde el usuario introduce su email para enviarle las ofertas relacionadas con sus preferencias. En nuestra aplicación tendremos que comprobar que el dato introducido en ese campo sea una dirección de correo válida. Para este tipo de operaciones utilizamos expresiones regulares.

Las expresiones regulares podemos decir que son patrones de búsqueda dentro de cadenas. En un principio, suele ser difícil utilizarlas, pero poco a poco vas viendo el potencial que tienen. Vamos a ver las reglas sintácticas para formar expresiones regulares POSIX que son las aceptadas por el motor de PHP.

Los caracteres no especiales se escriben como de costumbre. Si queremos buscar en un cadena de caracteres la cadena aabb, podemos utilizar la expresión regular aabb.

El **símbolo ^** indica que tenemos que empezar a buscar el patrón al inicio de la cadena. Así la siguiente expresión regular *^a* solo evalúa como ciertas aquellas cadenas que empiecen por el carácter a.

El **símbolo *$*** obliga al patrón a cumplirse hasta el final de la cadena. Así la siguiente expresión regular *aabb$* solo evaluará como ciertas aquellas cadenas que terminen por aabb.

El **símbolo .** simboliza cualquier carácter. Así la expresión regular *^a.b* evaluará como ciertas las cadenas que empiecen por el carácter a, seguido de otro carácter cualquiera, seguido del carácter b.

Los **símbolos []** indican que se pueden encontrar el carácter encerrado entre los corchetes. Vamos a ver este caso con mayor detalle. Si utilizamos la expresión regular [aeiou] evaluará como cierta cualquier cadena que contenga una vocal. Si utilizamos la expresión regular [a-z] evaluará como cierta cualquier cadena que contenga un carácter entre el rango a-z, es decir cualquier letra del abecedario en minúsculas. También puede incluir un conjunto de rangos mediante el patrón [a-zA-Z]. También podemos excluir un conjunto de rangos mediante el símbolo ^ solo si es utilizado dentro de los corchetes.

El **símbolo *** indica que puede haber cero o más instancias.

El **símbolo +** indica que pueden haber una o más instancias.

El **símbolo |** indica una alternativa en el patrón.

El **símbolo ** es el carácter de escape por si queremos buscar un carácter especial.

Hasta aquí todo lo referente a las expresiones regulares. La mejor forma de entenderlas es viendo ejemplos distintos. En este caso vamos a formar una expresión regular que permita validar correos del estilo:

correo@dominio.com

o correos del estilo:

correo@subdominio.dominio.com

Vamos a comenzar con el ejemplo. En primer lugar permitimos desde el comienzo un número de letras, al menos tiene que tener una:

^[a-zA-Z]+

Seguido de una @ y un conjunto de letras:

^[a-z]+@[a-z]+

Seguido de un .com final:

^[a-z]+@[a-z]+\.com$

Esta expresión regular ya acepta correos con el primer estilo que queríamos correo@dominio.com pero no acepta el segundo estilo ya que después de la arroba no hemos aceptado el carácter punto.

^[a-z | \.]+@[a-z | \.]+\.com$

Ahora ya acepta los dos estilos de correos electrónicos. Ya veremos esto en profundidad con ejemplos concretos.

ereg(), eregi()

La función ereg() es capaz de comprobar si una cadena cumple con el patrón de la expresión regular que se pasa como parámetro. Devuelve true si lo acepta, false en caso contrario.

```
<?php
     $cadena1 = "correo@dominio.com";

     //imprime: Correo válido
     if(ereg(^[a-z|\.]+@[a-z|\.]+\.com$,$cadena))
        echo ( "Correo válido" );
?>
```

El equivalente eregi() no es sensible a mayúsculas y minúsculas.

ereg_replace(), eregi_replace()

Esta función toma tres parámetros. El primero es una expresión regular, el segundo un string por el que se cambiarán todas las ocurrencias del patrón encontrado, y el tercer un string donde se buscará el patrón.

```php
<?php
     $cadena1 = "correo@dominio.com";

     //imprime: Hola Francisco
     echo(ereg_replace(^[a-z|\.]+@[a-z|\.]+\.com$,"Hola Francisco",
$cadena));
?>
```