Capitulo 9: Manejo de Strings y expresiones regulares.

A continuación se describe un resumen de las funciones mas importantes de PHP para manejo de strings.

Mayúsculas y minúsculas:

string=strtoupper(string);

Pasa un string a mayúsculas.

string=strtolower(string);

Pasa un string a minúsculas.

string=ucfirst(string);

Pasa a mayúscula el primer carácter de un string

string=ucwords(string);

Pasa a mayúsculas el primer carácter de cada palabra de un string (separadas por blancos, tabulaciones y saltos de línea)

Trimming:

string=chop(string);

Elimina blancos y saltos de línea a la derecha de un string dado.

string=ltrim(string);

Elimina blancos y saltos de línea a la izquierda de un string.

string=trim(string);

Elimina blancos y saltos de línea a derecha e izquierda de un string.

Comparaciones:

int=strpos(string1,string2);

Devuelve la posición de la primera ocurrencia de string 2 dentro de string1.

int=strspn(string1,string2);

Devuelve la longitud en caracteres de s1 contando desde el principio hasta que aparece un caracter en s1 que no esta en s2.

int=strcmp(string1,string2);

Compara dos strings y devuelve 1, 0 o –1 según sea mayor el primero, iguales o el segundo.

int=strcasecmp(string1,string2);

Idem anterior pero case-insesitive (no distingue mayúsculas y minúsculas)

int=strcspn(string1,string2);

Devuelve la longitud de s1 desde el principio hasta que aparece un caracter que pertenece a s2.

int=strstr(string1,string2);

Devuelve todos los caracteres de s1 desde la primera ocurrencia de s2 hasta el final.

int=stristr(string1,string2);

Idem anterior pero case-insensitive (no distingue mayúsculas de minúsculas)

int=similar_text(string1,string2,referencia);

Analiza la semejanza entre dos strings, devuelve la cantidad de caracteres iguales en los dos strings, si se pasa como tercer parámetro una referencia a una variable devuelve en la misma el porcentaje de similitud entre ambos strings de acuerdo al algoritmo de Oliver (1993).

Ejemplo:

similar_text($st1,$st2,&$porcentaje);

Funciones de Parsing:

array=explode(separator,string);

Devuelve un vector donde cada elemento del vector es un substring del string pasado particionado de acuerdo a un cierto caracter separador.

Ejemplo:

$st=”hola,mundo,como,estan”
$vec=explode(“,”,$st); //$vec=(“hola”,”mundo”,”como”,”estan”);

string=implode(separator,array);

Genera un string concatenando todos los elementos del vector pasado e intercalando separator entre ellos.

string=chunk_split(string,n,end);

end es opcional y por default es “\r\n”, devuelve un string en donde cada “n” caracteres del string original se intercala el separador “end”.

Ejemplo:

$st=”hola mundo”;
$st2=chunk_split($st,2,”,”);
//$st2=”ho,la, m,un,do”;

array=count_chars(string);

Devuelve un vector de 256 posiciones donde cada posición del vector indica la cantidad de veces que el caracter de dicho orden aparece en el vector.

string=nl2br(string);

Devuelve un string en donde todos los saltos de línea se han reemplazado por el tag
 de html.

string=strip_tags(string,string_tags_validos);

Devuelve un string eliminando del string original todos los tags html, si se pasa el segundo parámetro opcional es posible especificar que tags no deben eliminarse (solo hace falta pasar los tags de apertura)

ejemplo:

$st2=strip_tags($st1,"
 <table>");

Elimina todos los tags html de $st1 excepto
 , <table> y </table>

string=metaphone(string);

Devuelve una representación metafónica (similar a soundex) del string de acuerdo a las reglas de pronunciación del idioma ingles.

string=strtok(separador,string);

Dado un separador obtiene el primer "token" de un string, sucesivas llamadas a strtok pasando solamente el separador devuelven los tokens en forma sucesiva o bien falso cuando ya no hay mas tokens.

Ejemplo:

```
$tok=strtok($st,"/");
while($tok) {
  //Hacer algo
  $tok=strtok("/");
}
```

parse_string(string);

Dado un string de la forma "nombre=valor&nombre2=valor2&nombre3=valor3", setea las variables correspondientes con los valores indicados, ejemplo:

```
parse_string("v1=hola&v2=mundo");
//Seteta $v1="hola" y $v2="mundo"
```

Codificación y decodificación ASCII.

char=chr(int);

Devuelve el caracter dado su número ascii.

int=ord(char);

Dado un caracter devuelve su código Ascii.

Substrings:

string=substr(string,offset,longitud);

Devuelve el substring correspondiente al string pasado desde la posición indicada por offset y de la longitud indicada como tercer parámetro, si no se pasa el tercer parámetro se toman todos los caracteres hasta el final del string.

string=substr_replace(string,string_reemplazo,offset, longitud);

Idem anterior pero el substring seleccionado es reemplazado por string_reemplazo, si string_reemplazo es “” entonces sirve para eliminar una porción de un string.

Búsquedas y Reemplazos.

str_replace(string1,string2,string3);

Reemplaza todas las ocurrencias de string1 en string3 por string2. Esta función no admite expresiones regulares como parámetros.

string=strtr(string1,string_from,string_to);

Reemplaza en string1 los caracteres en string_from por su equivalente en string_to (se supone que string_from y string_to son de la misma longitud, si no lo son los caracteres que sobran en el string mas largo se ignoran)

Ejemplo:

```
$st=”hola mundo”
strtr($st,”aeiou”,”12345”);
//$st=”h4la m5nd4”
```

array=split(pattern,string);

Idem a explode pero el separador puede ser ahora una expresión regular.

boolean=ereg(pattern,string,regs);

Devuelve true o false según si el string matchea o no una expresión regular dada, el tercer parámetro es opcional y debe ser el nombre de un vector en donde se devolverán los matches de cada paréntesis de la expresion regular si es que la misma tiene paréntesis.

boolean=eregi(pattern,string,regs);

Idem anterior pero case-insensitive.

ereg_replace(pattern_from,string_to,string);

Reemplaza todas las ocurrencias de una expresión regular en string por el contenido de string_to.

eregi_replace(pattern_from,string_to,string);

Idem anterior pero no considera mayúsculas y minúsculas para la búsqueda de la expresión regular en el string.

Sintaxis básica de una expresión regular:

Los símbolos especiales "^" y "$" se usan para matchear el principio y el final de un string respectivamente. Por ejemplo:

"^el"	Matchea strings que empiezan con "el"
"colorin colorado$"	Matchea strings que terminan en "colorin colorado"
"^abc$"	String que empieza y termina en abc, es decir solo "abc" matchea
"abc"	Un string que contiene "abc" por ejemplo "abc" ,"gfabc", "algoabcfgeh", etc...

Los símbolos "*" , "+" y "?" denotan la cantidad de veces que un caracter o una secuencia de caracteres puede ocurrir. Y denotan 0 o más, una o más y cero o una ocurrencias respectivamente.
Por ejemplo:

"ab*"	Matchea strings que contienen una "a" seguida de cero o mas "b" Ej: "a", "ab", "cabbbb", etc
"ab+"	Matchea strings que contienen una "a" seguida de una o mas "b"
"ab?"	Matchea strings que contienen una "a" seguida o no de una "b" pero no mas de 1.
"a?b+$"	Matchea "a" seguida de una o mas "b" terminando el string.

Para indicar rangos de ocurrencias distintas pueden especificarse la cantidad máxima y mínima de ocurrencias usando llaves de la forma {min,max}

"ab{2}"	Una "a" seguida de exactamente 2 "b"
"ab{2,}"	Una "a" seguida de 2 o mas "b"
"ab{3,5}"	Una "a" seguida de 3 a 5 "b" ("abbb", "abbbb", "abbbbb")

Es obligatorio especificar el primer número del rango pero no el segundo. De esta forma
+ equivale a {1,}. * equivale a {0,} y ? equivale a {0,1}

Para cuantificar una secuencia de caracteres basta con ponerla entre paréntesis.

"a(bc)*"	Matchea una "a" seguida de cero o mas ocurrencias de "bc" ej: "abcbcbc"

El símbolo "|" funciona como operador "or"

"hola|Hola"	Matchea strings que contienen "hola" u "Hola"
"(b|cd)ef"	Strings que contienen "bef" o "cdef"
"(a|b)*c"	Secuencias de "a" o "b" y que termina en "c"

El carácter "." matchea a cualquier otro caracter.

"a.[0-9]"	Matchea "a" seguido de cualquier caracter y un dígito.
"^.{3}$"	Cualquier string de exactamente 3 caracteres.

Los corchetes se usan para indicar que caracteres son validos en una posición única del string.

"[ab]"	Matchea strings que contienen "a" o "b"
"[a-d]"	Matchea strings que contienen "a", "b" , "c" o "d"
"^[a-zA-Z]"	Strings que comienzan con una letra.
"[0-9]%"	Un dígito seguido de un signo %

También puede usarse una lista de caracteres que no se desean agregando el símbolo "^" dentro de los corchetes, no confundir con "^" afuera de los corchetes que matchea el principio de línea.

"[^abg]"	Strings que NO contienen "a" , "b" o "g"

“[^0-9]” Strings que no contienen dígitos

Los caracteres “^.[$()|*+?{\” deben escaparse si forman parte de lo que se quiere buscar con una barra invertida adelante. Esto no es válido dentro de los corchetes donde todos los caracteres no tienen significado especial.

Ejemplos:

Validar una suma monetaria en formato: “10000.00”, “10,000.00” .,“10000” o “10,000” es decir con o sin centavos y con o sin una coma separando tres dígitos.

^[1-9][0-9]*$

Esto valida cualquier número que no empieza con cero, lo malo es que “0” no pasa el test. Entonces:

^(0|[1-9][0-9]*)$

Un cero o cualquier número que no empieza con cero. Aceptemos también un posible signo menos delante.

^(0|-?[1-9][0-9]*)$

O sea cero o cualquier número con un posible signo “-“ delante.

En realidad podemos admitir que un número empiece con cero para una cantidad monetaria y supongamos que el signo “-“ no tiene sentido, agreguemos la posibilidad de decimales:

^[0-9]+(\.[0-9]+)?$

Si viene un “.” debe esta seguido de un dígito, 10 es válido pero “10.” no
Especifiquemos uno o dos dígitos decimales:

^[0-9]+(\.[0-9]{1,2})?$

Ahora tenemos que agregar las comas para separar de a miles.

^[0-9]{1,3}(,[0-9]{3})*(\.[0-9]{1,2})?$

O sea un conjunto de 1 a 3 dígitos seguido de uno más conjuntos de “,” seguido de tres dígitos. Ahora hagamos que la coma sea opcional.

^([0-9]+|[0-9]{1,3}(,[0-9]{3})*)(\.[0-9]{1,2})?$

Y de esta forma podemos validar números con los 4 formatos validos en una sola expresión.