bilib
Creer para Crear

PARQUE CIENTÍFICO
Y TECNOLÓGICO
DE CASTILLA - LA MANCHA

Castilla-La Mancha

Módulo

3

Programación Web. Operaciones

Autor del documento:

Centro de Apoyo Tecnológico a Emprendedores

Datos de contacto:

E-Mail: bilib@bilib.es

Página Web: www.bilib.es

Teléfono: 967 555 311

1. Índice

2. Objetivos.

- **Conocer** la correcta configuración de funciones para trabajar con operadores aritméticos.
- **Comprender** la estructura de las operaciones aritméticas.
- **Diferenciar** entre las distintas operaciones lógicas.
- **Analizar** las operaciones lógicas como expresiones matemáticas.
- **Diferenciar** entre operaciones lógicas, operaciones aritméticas y expresiones regulares.

3. Operaciones

3.1 Operaciones aritméticas

Números.

Los números decimales se escriben con punto (.), no con coma (,).

Operadores aritméticos.

Los operadores aritméticos básicos son los siguientes:

Ejemplo	Nombre	Resultado
-$a	Negación	El opuesto de $a.
$a + $b	Suma	Suma de $a y $b.
$a - $b	Resta	Diferencia entre $a y $b.
$a * $b	Multiplicación	Producto de $a y $b.
$a / $b	División	Cociente de $a y $b.
$a % $b	Módulo	Resto de $a dividido por $b. **Cuidado**: Los números se convierten a enteros antes de efectuar el cálculo. Por ejemplo, 5 % 2.5 da como resultado 1 y no 0 porque calcula el resto de 5 entre 2, no de 5 entre 2.5.

Hay que tener en cuenta que en PHP un entero no puede ser arbitrariamente grande. A partir de cierto valor, que depende del sistema operativo, PHP convierte automáticamente los enteros en float, perdiéndose precisión. En un sistema de 32 bits, el valor máximo es 2147483647 (231-1).

Si se necesita trabajar con enteros mayores, es necesario utilizar las "funciones bcmath".

Operadores de incremento y decremento

Ejemplo	Nombre	Efecto
++$a	Pre-incremento	Incrementa $a en uno, y luego devuelve $a.
$a++	Post-incremento	Devuelve $a, y luego incrementa $a en uno.
--$a	Pre-decremento	Decrementa $a en uno, luego devuelve $a.
$a--	Post-decremento	Devuelve $a, luego decrementa $a en uno.

La diferencia entre el pre-incremento y el post-incremento es que en el primer caso primero se incrementa la variable y después se utiliza y en el segundo primero se utiliza y después se incrementa.

```php
<?php
$valor = 9;
print "<p>" . ++$valor . "</p>\n";
?>
```

```
<p>10</p>
```

```php
<?php
$valor = 9;
print "<p>" . $valor++ . "</p>\n";
?>
```

```
<p>9</p>
```

El operador de incremento funciona también con caracteres, teniendo en cuenta que al incrementar el carácter 'z' se obtiene la cadena 'aa'. El operador de decremento no funciona con caracteres.

```php
<?php
$valor = "b";
print "<p>" . ++$valor . "</p>\n";
?>
```

```
<p>c</p>
```

```php
<?php
$valor = "z";
print "<p>" . ++$valor . "</p>\n";
?>
```

```
<p>aa</p>
```

```php
<?php
$valor = "a9z";
print "<p>" . ++$valor . "</p>\n";
?>
```

```
<p>b0a</p>
```

Resto de una división

El operador **%** calcula el resto de una división entera

```php
<?php
$resto = 17 % 3;
print "<p>El resto de 17 dividido entre 3 es " . $resto . "
</p>\n";
?>
```

```
<p>El resto de 17 dividido entre 3 es 2</p>
```

La función **fmod** calcula el resto de una división con números decimales

```php
<?php
print "<p>El resto de 17 dividido entre 3.1 es " . fmod(17, 3.1) .
"</p>\n";
?>
```

```
<p>El resto de 17 dividido entre 3.1 es 1.5</p>
```

Hay que tener en cuenta que en PHP un entero no puede ser arbitrariamente grande. A partir de cierto valor, que depende del sistema operativo, PHP convierte automáticamente los enteros en float, perdiéndose precisión. En un sistema de 32 bits, el valor máximo es 2147483647 (2^{31}-1).

Si se necesita trabajar con enteros mayores, es necesario utilizar las funciones bcmath.

Paréntesis

Los operadores artiméticos tienen el mismo orden de precedencia que en Matemáticas. Concretamente, el orden de precedencia de los operadores comentados anteriormente es, de mayor a menor, el siguiente (los operadores indicados en el mismo grupo se efectúan en el orden en que aparecen en la expresión):

- ++ (incremento) -- (decremento)
- * (multiplicación) / (división) % (resto)
- + (suma) - (resta)

Los paréntesis permiten agrupar operaciones de manera que las operaciones entre paréntesis se realicen antes que las operaciones fuera de los paréntesis. Se aconseja no utilizar paréntesis cuando las operaciones den el mismo resultado con o sin paréntesis.

Operadores de asignación

Los operadores de asignación permiten simplificar algunas expresiones de asignación:

Ejemplo	Nombre	Equivale a
$a += $b	Suma	$a = $a + $b
$a -= $b	Resta	$a = $a - $b
$a *= $b	Multiplicación	$a = $a * $b
$a /= $b	División	$a = $a / $b
$a %= $b	Módulo	$a = $a % $b

Redondear un número

La función **round(x)** redondea el número x al entero más próximo.

```php
<?php
print "<p>2.6 se redondea con round() a " . round(2.6) . "</p>\n";
print "<p>2.3 se redondea con round() a " . round(2.3) . "</p>\n";
print "<p>-2.6 se redondea con round() a " . round(-2.6) . "
</p>\n";
print "<p>-2.3 se redondea con round() a " . round(-2.3) . "
</p>\n";
?>
```

```
<p>2.6 se redondea con round() a 3</p>
<p>2.3 se redondea con round() a 2</p>
<p>-2.6 se redondea con round() a -3</p>
<p>-2.3 se redondea con round() a -2</p>
```

La función **round()** se puede utilizar para verificar si un número es un número entero (positivo o negativo), comprobando si un número coincide con su valor redondeado.

```php
<?php
$numero = 4.3;
if ($numero == round($numero)) {
    print "<p>$numero es un número entero</p>\n";
} else {
    print "<p>$numero no es un número entero</p>\n";
}

$numero = -6;
if ($numero == round($numero)) {
    print "<p>$numero es un número entero</p>\n";
} else {
    print "<p>$numero no es un número entero</p>\n";
}
?>
```

```
<p>4.3 no es un número entero</p>
<p>-6 es un número entero</p>
```

La función **round(x,n)** redondea x con n decimales (si n es negativo redondea a decenas, centenas, etc.).

```php
<?php
print "<p>2.6574 se redondea con round() con dos decimales a " .
round(2.6574, 2) . "</p>\n";
print "<p>3141592 redondeado con round() con centenas es " .
round(3141592, -2) . "</p>\n";
?>
```

```
<p>2.6574 se redondea con round() con dos decimales a 2.66</p>
<p>3141592 redondeado con round() con centenas es 3141600</p>
```

La función **floor(x)** redondea el número x al entero inferior (es decir, devuelve la parte entera).

```php
<?php
print "<p>2.6 se redondea con floor() a " . floor(2.6) . "</p>\n";
print "<p>2.3 se redondea con floor() a " . floor(2.3) . "</p>\n";
print "<p>-2.6 se redondea con floor() a " . floor(-2.6) . "
</p>\n";
print "<p>-2.3 se redondea con floor() a " . floor(-2.3) . "
</p>\n";
?>
```

```
<p>2.6 se redondea con floor() a 2</p>
<p>2.3 se redondea con floor() a 2</p>
<p>-2.6 se redondea con floor() a -3</p>
<p>-2.3 se redondea con floor() a -3</p>
```

La función **ceil(x)** redondea el número x al entero superior.

```php
<?php
print "<p>2.6 se redondea con ceil() a " . ceil(2.6) . "</p>\n";
print "<p>2.3 se redondea con ceil() a " . ceil(2.3) . "</p>\n";
print "<p>-2.6 se redondea con ceil() a " . ceil(-2.6) . "</p>\n";
print "<p>-2.3 se redondea con ceil() a " . ceil(-2.3) . "</p>\n";
?>
```

```
<p>2.6 se redondea con ceil() a 3</p>
<p>2.3 se redondea con ceil() a 3</p>
<p>-2.6 se redondea con ceil() a -2</p>
<p>-2.3 se redondea con ceil() a -2</p>
```

Potencias

La función **pow(x, y)** calcula x elevado a y.

```php
<?php
print "<p>2<sup>3</sup> = " . pow(2, 3) . "</p>\n";/?>
```

```
<p>2 = 8</p>
```

Máximo y mínimo

Las funciones **max()** y **min()** devuelven el máximo y el mínimo, respectivamente, de una lista o matriz de valores.

```php
<?php
print "<p>El máximo es " . max(20, 40, 25.1, 14.7) . "</p>\n";
?>
```

```
<p>El máximo es 40</p>
```

```php
<?php
print "<p>El mínimo es " . min(20, 40, 25.1, 14.7) . "</p>\n";
?>
```

```
<p>El mínimo es 14.7</p>
```

```php
<?php
$datos = [20, 40, 25.1, 14.7];
print "<p>El mínimo es " . min($datos) . "</p>\n";
?>
```

```
<p>El mínimo es 14.7</p>
```

Formatear un número

Para escribir un número con los símbolos de separación de decimales y de miles españoles, es decir, una coma (,) para separar la parte entera de la decimal y un punto (.) para separar las cifras de la parte entera en grupos de tres, se puede utilizar la función **number_format()**.

```php
<?php
print "<p>" . number_format(1300, 5) . "</p>\n";
?>
```

```
<p>1,300.00000</p>
```

```php
<?php
print "<p>" . number_format(123456.789, 2) . "</p>\n";
?>
```

```
<p>123,456.79</p>
```

```php
<?php
print "<p>" . number_format(123456789123, 0, ',', '.') . "</p>\n";
?>
```

```
<p>123.456.789.123</p>
```

```php
<?php
print "<p>" . number_format(123456789123456.789, 2, ',', '.') . "
</p>\n";
?>
```

```
<p>123.456.789.123.456,78</p>
```

La función requiere dos o cuatro argumentos:

- el primer argumento es el número a formatear.
- el segundo argumento es el número de decimales a mostrar (el número se redondea o trunca dependiendo de la longitud del número, compárese el segundo y el cuarto ejemplo de los ejemplos anteriores).
- el tercer argumento es el carácter a utilizar como separador de la parte entera de la decimal.
- el cuarto argumento es el carácter a utilizar como separador de miles.

La función devuelve el número formateado. Si sólo se utilizan dos argumentos, se utiliza el punto como separador de parte entera y decimal y la coma como separador de miles (notación inglesa).

Números aleatorios

Para obtener un número entero aleatorio entre dos valores determinados, se pueden utilizar la función **rand()** o la función **mt_rand()** (que se supone que es más rápida). Ambas funciones requieren dos argumentos:

- El primer argumento es el valor mínimo que se quiere obtener
- El segundo argumento es el valor máximo que se quiere obtener.

```php
<?php
print "<p>" . mt_rand(1, 6) . "</p>\n";
print "<p>" . mt_rand(1, 6) . "</p>\n";
print "<p>" . mt_rand(1, 6) . "</p>\n";
?>
```

```
<p>5</p>
<p>3</p>
<p>6</p>
```

```php
<?php
print "<p>" . rand(-10, 10) . "</p>\n";
print "<p>" . rand(-10, 10) . "</p>\n";
print "<p>" . rand(-10, 10) . "</p>\n";
?>
```

```
<p>-8</p>
<p>-10</p>
<p>3</p>
```

Si se llama a las funciones sin argumentos, estas devuelven un número entero aleatorio entre 0 y un valor máximo que es el que devuelve la función **getrandmax()** o **mt_getrandmax()** (este valor depende del sistema operativo).

```php
<?php
print "<p>" . mt_getrandmax() . "</p>\n";
print "<p>" . mt_rand() . "</p>\n";
print "<p>" . mt_rand() . "</p>\n";
?>
```

```
<p>2147483647</p>
<p>2023208892</p>
<p>800052801</p>
```

```php
<?php
print "<p>" . getrandmax() . "</p>\n";
print "<p>" . rand() . "</p>\n";
print "<p>" . rand() . "</p>\n";
?>
```

```
<p>32767</p>
<p>12279</p>
<p>24164</p>
```

La función **rand()** puede devolver valores mayores que los que devuelve cuando se la llama sin argumentos, pero no se puede superar el mayor número entero que maneja PHP (PHP_INT_MAX).

```php
<?php
print "<p>" . PHP_INT_MAX . "</p>";
print "<p>" . rand(100000, 200000) . "</p>\n";
print "<p>" . mt_rand(100000000000, 200000000000) . "</p>\n";
?>
```

```
<p>2147483647</p>
<p>166031</p>
Warning: mt_rand(): max(-1863462912) is smaller than
min(1215752192) in prueba.php on line 3
```

Las funciones **rand()** y **mt_rand()** generan números aleatorios que no se pueden considerar criptográficamente seguros. En PHP 7.0 se incorporó una nueva función, **random_int()**, que sí puede considerarse criptográficamente segura.

3.2 Operaciones lógicas

Las operaciones lógicas son expresiones matemáticas cuyo resultado es un valor booleano (verdadero o falso **true** o **false**). Estas expresiones se utilizan principalmente en las estructuras de control.

Comparaciones

Las comparaciones permiten comparar variables o expresiones entre sí o con valores concretos. El resultado de la comparación es un valor booleano (**true** o **false**).

Ejemplo	Nombre	Resultado
$a == $b	Igual	**true** si $a es igual a $b.
$a === $b	Idéntico	**true** si $a es igual a $b, y son del mismo tipo. (a partir de PHP 4)
$a != $b $a <> $b	Diferente	**true** si $a no es igual a $b.
$a !== $b	No idénticos	**true** si $a no es igual a $b, o si no son del mismo tipo. (a partir de PHP 4)
$a < $b	Menor que	**true** si $a es estrictamente menor que $b.
$a > $b	Mayor que	**true** si $a es estrictamente mayor que $b.
$a <= $b	Menor o igual que	**true** si $a es menor o igual que $b.
$a >= $b	Mayor o igual que	**true** si $a es mayor o igual que $b.

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

if ($nombre == "Juan") {
  print "<p>Tu nombre es Juan.</p>\n";
}

if ($nombre != "Juan") {
  print "<p>Tu nombre no es Juan.</p>\n";
}
?>
```

```
<p>Tu nombre no es Juan.</p>
```

```php
<?php
$nombrePadre = "Pepe";
$nombreHijo  = "Pepe";

if ($nombrePadre == $nombreHijo) {
  print "<p>El hijo se llama como el padre.</p>\n";
}

if ($nombrePadre != $nombreHijo) {
  print "<p>El hijo no se llama como el padre.</p>\n";
}
?>
```

```
<p>El hijo se llama como el padre.</p>
```

Un error típico cuando se empieza a programar es confundir el operador comparación (==) con el operador de asignación (=), lo que produce resultados inesperados, como en el ejemplo siguiente:

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

if ($nombre = "Juan") {
  print "<p>Tu nombre es Juan.</p>\n";
}

if ($nombre != "Juan") {
  print "<p>Tu nombre no es Juan.</p>\n";
}

?>
```

```
<p>Tu nombre es Juan.</p>
```

Al escribir = en vez de ==, PHP no compara el valor de la variable $nombre con la cadena "Juan", sino que cambia el valor de $nombre a "Juan". Como la asignación no da error, PHP le asigna el valor booleano **true**, y escribe que el nombre es Juan.

Al no producir mensaje de error, este error puede ser difícil de localizar en programas largos o complejos.

La diferencia entre los operadores == y === es que en el primer caso si las cadenas se pueden interpretar como números entonces se convierten a números antes de realizar la comparación, mientras que en el segundo caso no.

```php
<?php
$numero = 5;
$texto = "5";

if ($numero == $texto) {
    print "<p>Las variables son iguales.</p>\n";
}

if ($numero != $texto) {
    print "<p>Las variables no son iguales.</p>\n";
}
?>
```

```
<p>Las variables son iguales.</p>
```

```php
<?php
$numero = 5;
$texto = "5";

if ($numero === $texto) {
    print "<p>Las variables son idénticas.</p>\n";
}

if ($numero !== $texto) {
    print "<p>Las variables no son idénticas.</p>\n";
}
?>
```

```
<p>Las variables no son idénticas.</p>
```

Operadores lógicos

Los operadores lógicos permiten combinar expresiones simples en expresiones más complejas.

Ejemplo	Nombre	Resultado
$a && $b $a and $b	Y	**true** si los dos, $a y $b, son **true**.
$a || $b $a or $b	O	**true** si uno de los dos, $a o $b, es **true**.
$a xor $b	O exclusivo (Xor)	**true** si sólo uno de los dos, $a o $b, es **true**, pero no ambos.
! $a	Negación	**true** si $a no es **true**.

Al escribir expresiones en las que se combinan varias comparaciones mediante operadores lógicos es conveniente utilizar paréntesis, aunque en muchos casos no sean necesarios porque las comparaciones tienen precedencia sobre los operadores lógicos.

Diferencia entre and y && y entre or y ||

Los operadores **and** y **&&** y los operadores **or** y **||** no son completamente equivalentes, ya que no tienen la misma precedencia. Concretamente, **&&** y **||**tienen mayor prioridad que **and** y **or**. Como además el operador de asignación **=** tiene una prioridad intermedia, se pueden producir situaciones inesperadas, como muestran los siguientes ejemplos.

El ejemplo siguiente muestra el resultado esperado:

```php
<?php
$var1 = true;
$var2 = false;
$todo = $var1 && $var2;
if ($todo) {
    print "<p>verdadero</p>\n";
} else {
    print "<p>falso</p>\n";
}
?>
```

```
<p>falso</p>
```

La variable $todo sólo tomaría el valor **true** si tanto $var1 como $var2 fueran **true**, pero como $var2 es **false**, $todo toma el valor **false**.

Sin embargo si se utiliza el operador **and** en vez de **&&**, el resultado no es el esperado:

```php
<?php
$var1 = true;
$var2 = false;
$todo = $var1 and $var2;
if ($todo) {
    print "<p>verdadero</p>\n";
} else {
    print "<p>falso</p>\n";
}
?>
```

```
<p>verdadero</p>
```

¿Por qué se produce ese resultado? Porque el operador de asignación = tiene precedencia sobre el operador **and**. Eso quiere decir que PHP realiza antes la asignación que la operación lógica, es decir, como si la expresión estuviese escrita así:

```
($todo = $var1) and $var2;
```

```
<p>verdadero</p>
```

En esa expresión, la variable $todo almacena el valor de la variable $var1 (**true**), por lo que $todo toma el valor **true**. La operación lógica **and** no modifica el valor de $todo.

Si se quiere obtener el mismo resultado con **and** que con **&&**, se deben utilizar paréntesis, para forzar que la operación lógica **and** se realice antes de la asignación:

```
<?php
$var1 = true;
$var2 = false;
$todo = ($var1 and $var2);
if ($todo) {
    print "<p>verdadero</p>\n";
} else {
    print "<p>falso</p>\n";
}
?>
```

```
<p>falso</p>
```

3.3 Expresiones regulares

Las expresiones regulares permiten definir patrones de coincidencia y aplicarlas a cadenas de texto para saber si la cadena (o parte de ella) cumple el patrón e incluso realizar transformaciones de la cadena.

En los años 90, PHP utilizaba las expresiones regulares POSIX extendido, pero a partir de PHP 5.3.0 (junio de 2009) las expresiones regulares POSIX extendido se consideran obsoletas.

Desde PHP 4.2.0 (abril de 2002), PHP cuenta con las expresiones regulares compatibles con Perl (en inglés, PCRE), que siguen la sintaxis y semánticas del

lenguaje de programación Perl 5. PHP 4.2.0 y posteriores incluyen la biblioteca de código libre escrita en C PCRE (Perl Compatible Regular Expressions).

Funciones de expresiones regulares compatibles con Perl

La función de expresiones regulares compatibles con Perl **preg_match($patron, $cadena [, $matriz_coincidencias [, $modificadores [, $desplazamiento]]])** compara una cadena con un patrón y devuelve 1 si el patrón ha coincidido o 0 si no. La primera coincidencia encontrada se puede guardar en el argumento opcional $matriz_coincidencias y, si se añade el modificador **PREG_OFFSET_CAPTURE**, se guarda también en el argumento opcional $matriz_coincidencias la posición de la coincidencia encontrada. El argumento opcional $desplazamiento es un número que permite indicar en qué carácter se inicia la búsqueda.

Los patrones deben empezar y acabar con el carácter / (barra).

```php
<?php
$cadena1 = "1234567";
$cadena2 = "abcdefg";
$patron = "/^[[:digit:]]+$/";

if (preg_match($patron, $cadena1)) {
    print "<p>La cadena $cadena1 son sólo números.</p>\n";
} else {
    print "<p>La cadena $cadena1 no son sólo números.</p>\n";
}

if (preg_match($patron, $cadena2)) {
    print "<p>La cadena $cadena2 son sólo números.</p>\n";
} else {
    print "<p>La cadena $cadena2 no son sólo números.</p>\n";
}
?>
```

La cadena 1234567 son sólo números.
La cadena abcdefg no son sólo números.

La función **preg_match()** distingue entre mayúsculas y minúsculas. Para que no distinga, debe añadirse el modificador "i" (sin comillas) al final del patrón. Este modificador no afecta las clases [[::]].

```php
<?php
$cadena = "aaAA";
$patron1 = "/^[a-z]+$/";
$patron2 = "/^[a-z]+$/i";

if (preg_match($patron1, $cadena)) {
      print "<p>La cadena $cadena son sólo letras minúsculas.
</p>\n";
} else {
     print "<p>La cadena $cadena no son sólo letras minúsculas.
</p>\n";
}

if (preg_match($patron2, $cadena)) {
      print "<p>La cadena $cadena son sólo letras minúsculas o
mayúsculas.</p>\n";
} else {
     print "<p>La cadena $cadena no son sólo letras minúsculas o
mayúsculas.</p>\n";
}
?>
```

```
La cadena aaAA no son sólo letras minúsculas.
La cadena aaAA son sólo letras minúsculas o mayúsculas.
```

La función de expresiones regulares compatibles con Perl **preg_match_all($patron, $cadena [, $matriz_coincidencias [, $modificadores [, $desplazamiento]]])** compara una cadena con un patrón y devuelve el número de coincidencias encontradas. Las coincidencias encontradas se pueden guardar en el argumento opcional $matriz_coincidencias y, si se añade el modificador **PREG_OFFSET_CAPTURE**, se guardan también en el argumento opcional $matriz_coincidencias la posición de cada coincidencia encontrada. El argumento opcional $desplazamiento es un número que permite indicar en qué carácter se inicia la búsqueda.

```php
<?php
$cadena = "Esto es una cadena de prueba";
$patron = "/de/";
$encontrado = preg_match_all($patron, $cadena, $coincidencias,
PREG_OFFSET_CAPTURE);

if ($encontrado) {
    print "<pre>"; print_r($coincidencias); print "</pre>\n";
    print "<p>Se han encontrado $encontrado coincidencias.</p>\n";
    foreach ($coincidencias[0] as $coincide) {
        print "<p>Cadena: '$coincide[0]' - Posición: $coincide[1]
</p>\n";
    }
} else {
    print "<p>No se han encontrado coincidencias.</p>\n";
}
?>
```

```
Array
(
    [0] => Array
        (
            [0] => Array
                (
                    [0] => de
                    [1] => 14
                )

            [1] => Array
                (
                    [0] => de
                    [1] => 19
                )

        )

)
Se han encontrado 2 coincidencias.
Cadena: 'de' - Posición: 14
Cadena: 'de' - Posición: 19
```

Sintaxis de las expresiones regulares compatibles con Perl

Los patrones de las expresiones reguladores compatibles con Perl deben empezar y acabar con un delimitador, que normalmente es la barra (/). En caso de que la barra forme parte del patrón, se puede:

- escribir en el patrón la barra precedida de una contrabarra (\/)
- utilizar otro carácter, que no sea uno de los caracteres especiales, como delimitador (por ejemplo, !, -, etc.)

Los patrones de expresiones regulares compatibles con Perl admiten modificadores, que se incluyen en el patrón, después del limitador final.

Los siguientes patrones son comunes a POSIX extendido y a compatibles con Perl, con una diferencia muy importante que es que las clases de carácter que incluyen caracteres alfabéticos ([[:alnum:]], [[:alpha:]], etc.) en POSIX **no** incluyen vocales ace

Patrón	Significado
c	carácter c
.	cualquier carácter
^c	empezar por el carácter c
c$	terminar por el carácter c
c+	1 o más caracteres c
c*	0 o más caracteres c
c?	0 o 1 caracteres c
\n	nueva línea
\t	tabulador
\	escape, para escribir delante de caracteres especiales: ^ . [ ] % ( ) | * ? { } \
(cd)	caracteres c y d agrupados
c|d	carácter c o d

c{n}	n veces el carácter c
c{n,}	n o más caracteres c
c{n,m}	desde n hasta m caracteres c
[a-z]	cualquier letra minúscula
[A-Z]	cualquier letra mayúscula
[0-9]	cualquier dígito
[cde]	cualquiera de los caracteres c, d o e
[c-f]	cualquier letra entre c y f (es decir, c, d, e o f)
[^c]	que no esté el carácter c
[[:alnum:]]	cualquier letra o dígito
[[:alpha:]]	cualquier letra
[[:digit:]]	cualquier dígito
[[:lower:]]	cualquier letra minúscula
[[:punct:]]	cualquier marca de puntuación
[[:space:]]	cualquier espacio en blanco
[[:upper:]]	cualquier letra mayúscula

Los siguientes patrones son exclusivos de compatibles con Perl y no existen en POSIX extendido:

Patrón	**Significado**
[[:ascii:]]	caracteres con código ASCII de 0 a 127
[[:blank:]]	espacios o tabuladores
[[:cntrl:]]	caracteres de control
[[:graph:]]	caracteres de impresión, salvo el espacio

[[:print:]]	caracteres de impresión, espacio incluido
[[:word:]]	cualquier letra o dígito y el guión bajo
[[:xdigit:]]	cualquier dígito hexadecimal
\w	cualquier letra o dígito y el guión bajo
\W	cualquier cosa que no sea letra o dígito y el guión bajo
\s	cualquier espacio en blanco
\S	cualquier cosa que no sea un espacio en blanco
\d	cualquier dígito
\D	cualquier cosa que no sea un dígito
\b	inicio o final de palabra
\A	comienzo
\Z	final (incluido salto de línea)
\z	final

Ejemplos de expresiones regulares

Patrón	Cadena	¿Cumple?	Comentario
abc	awbwc	No	Los caracteres tienen que estar seguidos.
	34abc	Sí	No importa que hayan caracteres antes...
	cbabcba	Sí	... o después.
a2b	g1da2b3	Sí	Las expresiones regulares detectan letras, números, ...
áb	3áb4	Sí	... incluso acentos, ...
a\$b	1a$b2	Sí	... salvo los caracteres ^ . [$ ( ) | * + ? { \ € que deben llevar una contrabarra \ antes, además de \n (nueva línea) y \t (tabulador)
[aeiou]	bic	Sí	Los corchetes definen los caracteres admitidos en una posición ...
	bcd	No	
[^aeiou]	bic	Sí	... o no admitidos
	aei	No	
[p-t]	avr	Sí	Se pueden definir rangos de caracteres...
	av1	No	

[B-D]	PMD	Sí	... en minúsculas o mayúsculas ...
	AV1	No	
[0-9]	b9d	Sí	... o números
	bcd	No	
[[:alpha:]]			Cualquier carácter alfabético
[[:digit:]]			Cualquier número
[[:alnum:]]			Cualquier número o carácter alfabéticos
[[:punct:]]			Cualquier carácter que no sean letras y números (menos el euro)
[[:space:]]			Cualquier tipo de espacio en blanco
[[:upper:]]			Cualquier mayúscula
[[:lower:]]			Cualquier minúscula
^ab	cab	No	Los caracteres tienen que estar al principio
	abc	Sí	No importa que hayan caracteres después
ab$	abc	No	Los caracteres tienen que estar al final
	cab	Sí	No importa que hayan caracteres antes
^ab$	ab	Sí	Tiene que empezar y acabar por ab ...

	abab	No	... y no puede haber nada antes o después
ab?c	abcde	Sí	El carácter b puede estar entre a y c...
	acde	Sí	... o no estar entre a y c ...
	adcde	No	... pero no puede haber otro carácter
a.c	abc	Sí	El . representa cualquier carácter ...
	a c	Sí	... incluso el espacio el blanco, ...
	ac	No	pero no la ausencia del carácter
	abdc	No	o varios caracteres.
ab+c	abcde	Sí	El carácter b puede estar una vez...
	abbbcde	Sí	... o varias ...
	acde	No	... pero tiene que estar al menos una vez.
ab*c	abcde	Sí	El carácter b puede estar una vez...
	abbbcde	Sí	... o varias ...
	acde	Sí	... o ninguna.
ab{3}c	abbbc	Sí	Las llaves indican el número exacto de repeticiones del carácter, ...
	abbbbc	No	... no puede haber más ...

	abbc	No	... ni menos.
ab{2,4}c	abc	No	Se pueden definir rangos con límite inferior y superior
	abbc	Sí	
	abbbc	Sí	
	abbbbc	Sí	
	abbbbbc	No	
ab{2,}c	abc	No	Se pueden definir rangos sin límite superior
a(bc){2}d	abcbcd	Sí	Los paréntesis definen agrupaciones de caracteres. En este caso bc tiene que aparecer repetido
a(bc)?d	abcd	Sí	Aquí bc puede estar ...
	ad	Sí	... o no estar, ...
	abd	No	... pero no puede aparecer sólo la b, o sólo la c u otro carácter
^a(b|d)c$	abc	Sí	Entre la a al principio y la c al final puede estar el carácter b...
	adc	Sí	... o el carácter d, ...
	abdc	No	... pero no los dos, ...

	ac	No	... ni ninguno de ellos.
^(ab)|(dc)$	abc	Sí	Está la pareja ab al principio ...
	adc	Sí	... o dc ...
	abdc	Sí	... o las dos, ...
	ac	No	... pero no ninguna
^(ab)$\|^(dc)$	abc	No	Está la pareja ab, pero sobra la c ...
	adc	No	... o está la pareja dc, pero sobra la a.
	dc	Sí	Está una de las dos
Patrón	Cadena	¿Cumple?	Comentario
abc	awbwc	No	Los caracteres tienen que estar seguidos.
	34abc	Sí	No importa que hayan caracteres antes...
	cbabcba	Sí	... o después.
a2b	g1da2b3	Sí	Las expresiones regulares detectan letras, números, ...
áb	3áb4	Sí	... incluso acentos, ...
a\$b	1a$b2	Sí	... salvo los caracteres ^ . [$ ( ) | * + ? { \ € que deben llevar una contrabarra \ antes, además de \n (nueva línea) y \t (tabulador)

[aeiou]	bic	Sí	Los corchetes definen los caracteres admitidos en una posición ...
	bcd	No	
[^aeiou]	bic	Sí	... o no admitidos
	aei	No	
[p-t]	avr	Sí	Se pueden definir rangos de caracteres...
	av1	No	
[B-D]	PMD	Sí	... en minúsculas o mayúsculas ...
	AV1	No	
[0-9]	b9d	Sí	... o números
	bcd	No	
[[:alpha:]]			Cualquier carácter alfabético
[[:digit:]]			Cualquier número
[[:alnum:]]			Cualquier número o carácter alfabéticos
[[:punct:]]			Cualquier carácter que no sean letras y números (menos el euro)
[[:space:]]			Cualquier tipo de espacio en blanco

[[:upper:]]			Cualquier mayúscula
[[:lower:]]			Cualquier minúscula
^ab	cab	No	Los caracteres tienen que estar al principio
	abc	Sí	No importa que hayan caracteres después
ab$	abc	No	Los caracteres tienen que estar al final
	cab	Sí	No importa que hayan caracteres antes
^ab$	ab	Sí	Tiene que empezar y acabar por ab ...
	abab	No	... y no puede haber nada antes o después
ab?c	abcde	Sí	El carácter b puede estar entre a y c...
	acde	Sí	... o no estar entre a y c ...
	adcde	No	... pero no puede haber otro carácter
a.c	abc	Sí	El . representa cualquier carácter ...
	a c	Sí	... incluso el espacio el blanco, ...
	ac	No	pero no la ausencia del carácter
	abdc	No	o varios caracteres.
ab+c	abcde	Sí	El carácter b puede estar una vez...
	abbbcde	Sí	... o varias ...

	acde	No	... pero tiene que estar al menos una vez.
ab*c	abcde	Sí	El carácter b puede estar una vez...
	abbbcde	Sí	... o varias ...
	acde	Sí	... o ninguna.
ab{3}c	abbbc	Sí	Las llaves indican el número exacto de repeticiones del carácter, ...
	abbbbc	No	... no puede haber más ...
	abbc	No	... ni menos.
ab{2,4}c	abc	No	Se pueden definir rangos con límite inferior y superior
	abbc	Sí	
	abbbc	Sí	
	abbbbc	Sí	
	abbbbbc	No	
ab{2,}c	abc	No	Se pueden definir rangos sin límite superior
a(bc){2}d	abcbcd	Sí	Los paréntesis definen agrupaciones de caracteres. En este caso bc tiene que aparecer repetido
a(bc)?d	abcd	Sí	Aquí bc puede estar ...

	ad	Sí	... o no estar, ...
	abd	No	... pero no puede aparecer sólo la b, o sólo la c u otro carácter
^a(b|d)c$	abc	Sí	Entre la a al principio y la c al final puede estar el carácter b...
	adc	Sí	... o el carácter d, ...
	abdc	No	... pero no los dos, ...
	ac	No	... ni ninguno de ellos.
^(ab)|(dc)$	abc	Sí	Está la pareja ab al principio ...
	adc	Sí	... o dc ...
	abdc	Sí	... o las dos, ...
	ac	No	... pero no ninguna
^(ab)$\|^(dc)$	abc	No	Está la pareja ab, pero sobra la c ...
	adc	No	... o está la pareja dc, pero sobra la a.
	dc	Sí	Está una de las dos

Enlaces para prácticas de expresiones regulares:

- Página de práctica “Debuggex Beta”: https://www.debuggex.com/
- Tester on line para la práctica de expresiones regulares: https://www.regextester.com/
- Página de práctica “Regex101” https://regex101.com/

4. Glosario

ALGORITMO: Conjunto de sentencias / instrucciones en lenguaje nativo, los cuales expresan la lógica de un programa.

ALGORITMO CUALITATIVO: Son aquellos que resolver un problema no ejecuta operaciones matemática en el desarrollo de algoritmo.

ALGORITMO CUANTITATIVO: Son aquellos algoritmos que ejecutan operaciones numéricas durante su ejecución.

ARCHIVO: Son un conjunto de registros lógicos.

BASE DE DATOS: Es un almacenamiento colectivo de las bibliotecas de datos que son requeridas y organizaciones para cubrir sus requisitos de procesos y recuperación de información.

BASIC: (BIGINNERS ALL PURPUS SIMBOLIC INSTRUTION CODE), Lenguaje de instrucciones simbólicas de propósito general para principiantes, está disponible en modo compilador e interprete, siendo este último el más popular para el usuario circunstancial y para el programador principiante.

BIT:(dígito binario) un dígito simple de un numero binario (1 ó 0) en el ordenador.

BUFFERS: Memoria intermedia, una porción reservada de la memoria, que se utiliza para almacenar datos mientras son procesados.

BYTE: Grupo de bits adyacentes operados como una unidad. (grupos de 8 bits).

CAMPO: Es el espacio en la memoria que sirve para almacenar temporalmente un dato durante el proceso, Su contenido varia durante la ejecución del programa.

CAMPO NUMÉRICO: el que solo puede almacenar valores (dígitos).

CAMPO ALFANUMERICO: el que puede almacenar cualquier carácter (dígito, letra, símbolo especial).

CODIFICACIÓN: Es la operación de escribir la solución del problema (de acuerdo a la lógica del diagrama de flujo o pseudocódigo), en una serie de instrucciones detalladas, en un código reconocible por el ordenador, la serie de instrucciones detalladas se le conoce como código fuente, el cual se escribe en un lenguaje de programación o lenguaje de alto nivel.

COMPILADOR: Programa de ordenador que produce un programa en lenguaje de máquina, de un programa fuente que generalmente está escrito por el programador en un lenguaje de alto nivel.

CONSTANTE: Valor o conjunto de caracteres que permanecen invariables durante la ejecución del programa.

DATO: El término que usamos para describir las señales con las cuales trabaja el ordenador es dato; Aunque las palabras dato e información muchas veces son usada indistintamente, si existe una diferencia importante entre ellas. En un sentido estricto, los datos son las señales individuales en bruto y sin ningún significado que manipulan los ordenadores para producir información.

DIAGRAMA DE FLUJO: Es la representación gráfica de una secuencia de instrucciones de un programa que ejecuta un ordenador para obtener un resultado determinado.

DOCUMENTACIÓN: Es la guía o comunicación escrita es sus variadas formas, ya sea en enunciados, procedimientos, dibujos o diagramas. A menudo un programa escrito por una persona, es usado por otra. Por ello la documentación sirve para ayudar a

comprender o usar un programa o para facilitar futuras modificaciones (mantenimiento).

La documentación de un programa se divide en interna, externa y de usuario final.

EDITOR: Es un software empleado para crear y manipular archivos de texto, tales como programas en lenguaje fuente, lista de nombres y direcciones.

FREEWARE: (Software gratis del inglés free software, aunque esta denominación también se confunde a veces con "libre" por la ambigüedad del término en el idioma inglés) define un tipo de software que se distribuye sin costo, disponible para su uso y por tiempo ilimitado, siendo una variante gratuita del shareware, en el que la meta es lograr que un usuario pruebe el producto durante un tiempo ("trial") limitado, y si le satisface, pague por él, habilitando toda su funcionalidad.

LENGUAJE DE ALTO NIVEL: Los lenguajes de programación de alto nivel (BASIC, pascal, cobol, fortran, etc.) son aquellos en los que las instrucciones o sentencias al ordenador son escritas con palabras similares a los lenguajes humanos (en general en inglés), lo que facilita la escritura y comprensión del programa.

LENGUAJE DE BAJO NIVEL (ENSAMBLADOR): En este lenguaje las instrucciones se escriben en códigos alfabéticos conocidos como mnemotécnicos para las operaciones y direcciones simbólicas.

LENGUAJE MÁQUINA: Son aquellos cuyas instrucciones son directamente entendibles por el ordenador y no necesitan traducción posterior para que la CPU pueda comprender y ejecutar el programa. Las instrucciones en lenguaje maquina se expresan en términos de la unidad de memoria más pequeña el bit (dígito binario 0 ó 1).

LENGUAJE DE PROGRAMACIÓN: Sistema de símbolos y reglas que permite la construcción de programas con los que el ordenador puede operar así como resolver problemas de manera eficaz. Estos contienen un conjunto de instrucciones que nos permiten realizar operaciones de entrada / salida, calculo, manipulación de textos, lógica / comparación y almacenamiento / recuperación.

MySQL: Es un sistema de gestión de bases de datos relacional, multihilo y multiusuario con más de seis millones de instalaciones.

OPERADORES: Un operador es un símbolo que indica al compilador que realice manipulaciones lógicas o matemáticas específicas. Los operadores del mismo nivel de precedencia son evaluados por el compilador de izquierda a derecha. Por supuesto, se puede utilizar paréntesis para ordenar la evaluación. También, conviene utilizar paréntesis para hacer más claro el orden en que se producen las evaluaciones, tanto para la persona que lo elabora o para los que después tengan que seguir el programa.

PROGRAMA: Es una colección de instrucciones que indican al ordenador que debe hacer. Un programa se denomina software, por lo tanto, programa, software e instrucción son sinónimos.

PROGRAMA FUENTE: Instrucción escrita por el programador en un lenguaje de programación para plantear al ordenador el proceso que debe ejecutar.

PROGRAMACIÓN ESTRUCTURADA: Método disciplinado de escribir programas que sean claros, que se demuestren que son correctos y fáciles de modificar

PROGRAMADOR: Un individuo que diseña la lógica y escribe las líneas de código de un programa de ordenador.

PRUEBA Y DEPURACIÓN: Los errores humanos dentro de la programación de ordenadores son muchos y aumentan considerablemente con la complejidad del problema. El proceso de identificar y eliminar errores, para dar paso a una solución sin errores se le llama depuración. La prueba consiste en la captura de datos hasta que el programa no presente errores (los más comunes son los sintácticos y lógicos).

PSEUDOCÓDIGO: Mezcla de lenguaje de programación y español (o inglés o cualquier otro idioma) que se emplea, dentro de la programación estructurada, para realizar el diseño de un programa. En esencial, el pseudocódigo se puede definir como un lenguaje de especificaciones de algoritmos.

Es la representación narrativa de los pasos que debe seguir un algoritmo para dar solución a un problema determinado. El pseudocódigo utiliza palabras que indican el proceso a realizar.

El pseudocódigo se concibió para superar las dos principales desventajas del Diagrama de Flujo: el diagrama de flujo es lento de crear y difícil de modificar sin un nuevo redibujo. Por otra parte el pseudocódigo es más fácil de utilizar ya que es similar al lenguaje natural.

VARIABLE: En programación es una estructura que contiene datos y recibe un nombre único dado por el programador, mantiene los datos asignados a ella hasta que un nuevo valor se le asigne o hasta que el programa termine.

5. Bibliografía

http://tutorialphp.net/iniciacion-a-php-7/introduccion-a-php-7/

http://php.net/manual/es/index.php

http://www.ww.mclibre.org/consultar/php/

http://administraciondesistemas.pbworks.com/f/Manual_PHP5_Basico.pdf

https://www.mundomanuales.com/manuales/3144.pdf

https://www.taringa.net/posts/info/16954106/Glosario-de-Lenguajes-de-Programacion-y-Base-de-datos.html

http://php.net/manual/es/history.php.php

https://desarrolloweb.com/articulos/436.php

https://www.debuggex.com/

https://www.regextester.com/

https://regex101.com/

6. Anexos

Manual de Programación en PHP.pdf

Manual imprescindible PHP5.pdf

Manual_PHP5_Basico.pdf

PHP Manual Completo Español.pdf

7. Cuestionario de autoevaluación.

1. **Los números decimales se escriben con coma (,):**

a) Verdadero.

b) Falso.

2. **El operador "%" calcula:**

a) El resto de una división entera.

b) El resto de una división con números decimales.

c) La pérdida de precisión con números enteros grandes.

d) La conversión de los enteros en "float".

3. **La función round(x,n) sirve para:**

a) Redondear el número x al entero más próximo.

b) Redondear el número x al entero inferior (es decir, devuelve la parte entera).

c) Redondear x con n decimales (si n es negativo redondea a decenas, centenas, etc.).

4. **Para obtener un número entero aleatorio entre dos valores determinados, siempre que se llame a funciones con argumentos, se pueden utilizar las siguientes funciones:**

a. rand() ó mt_rand()

b) getrandmax() ó mt_getrandmax()

c) random_int() ó mt_rand()

d) rand() ó random_int()

5. ¿Cuál de las siguientes operaciones lógicas daría como resultado "true si $a es estrictamente menor que $b"?

a. $a > $b
b. $a <= $b
c. $a >= $b
d) $a < $b

6. Los operadores "and" y "&&"…

a) No son completamente equivalentes, aunque tienen la misma precedencia.
b) No son equivalentes ya que "and" y "or" tienen mayor prioridad que "&&" y "||".
c) No son equivalentes ya que "&&" y "||" tienen mayor prioridad que "and" y "or".

7. Las expresiones regulares permiten…

a) Definir patrones de coincidencia y aplicarlas a cadenas de texto para saber si la cadena (o parte de ella) cumple el patrón e incluso realizar transformaciones de la cadena.
b) Combinar expresiones simples en expresiones más complejas.
c) Comparar variables o expresiones entre sí o con valores concretos.

8. La función de expresiones regulares "preg_match($patron, $cadena [, $matriz_coincidencias [, $modificadores [, $desplazamiento]]])"…

a) Compara una cadena con un patrón y devuelve 1 si el patrón ha coincidido o 0 si no.
b) Compara una cadena con un patrón y devuelve el número de coincidencias encontradas.

9. Los patrones de expresiones regulares compatibles con Perl…

a) No admiten modificadores que se incluyan en el patrón, después del delimitador final.
b) Permiten utilizar los símbolos "{}" como delimitadores
c) Deben empezar y acabar con un delimitador.

10. La función "pow(x, y)":

a) Devuelven el máximo y el mínimo de una lista o matriz de valores.
b) Redondea el número "x" al entero superior
c) Calcula "x" elevado a "y"