**Alumno:**

**Donovan L Cuellar Arriaga**

**Grupo:**

**DSM-41**

Ejemplos de FOR

**Factorial**

<?php

$factorial = 1;

for($numero = 5; $numero > 1; $numero = $numero - 1 ) {

   $factorial = $factorial \* $numero;

}

echo ' El factorial de ' . $numero . ' es ' . $factorial.

?>

**Iterar usuarios por medio de arrays**

<?php  
$people = array(  
    array('name' => 'Kalle', 'salt' => 856412),  
    array('name' => 'Pierre', 'salt' => 215863)  
);  
  
for($i = 0, $size = count($people); $i < $size; ++$i) {  
    $people[$i]['salt'] = mt\_rand(000000, 999999);  
}  
?>

**Números del 1 al 15**

<?php

for($i=1; $i<=15; $i++){

echo $i;// Imprime Ciclos

echo "<br>";

}

?>

Foreach ejemplos

**Mostrar los datos en una tabla.**

**<?php**

// Array

$productos = [

['codigo' => 'A0001', 'desc' => 'Mouse'],

['codigo' => 'A0002', 'desc' => 'Teclado'],

['codigo' => 'A0003', 'desc' => 'Monitor'],

['codigo' => 'A0004', 'desc' => 'Impresor'],

];

//Abrir etiqueta de tabla

echo '<table class="table table-striped">';

//Imprimir encabezado de tabla

echo '<thead class="thead-dark">

<tr><th>Codigo</th><th>Descripcion</th></tr>

</thead>';

// Imprimir las filas de la tabla

foreach ($productos as $prod) {

echo '<tr>

<td>'. $prod['codigo'] .'</td>

<td>'. $prod['desc'] .'</td>

</tr>';

}

echo '</table>';

**?>**

**Recorrer valores**

**<?php**

$frutas = ['Mango', 'PiÃ±a', 'Uva'];

foreach ($frutas as $valor) {

echo $valor;

}

**?>**

**Asigna días de la semana**

<?php

$diassemana = array("Lunes","Martes","Miércoles","Jueves","Viernes","Sábado","Domingo");

foreach($diassemana as $dias){

echo $dias."<br>";

}

?>

**Ejemplos While**

**Mostrar ID menor o igual a 10**

<?php

$id=2; //creamos la variable id en 2

while($id <=10){

echo $id; //ejecutar id

echo "<br>";

$id+=2;

}

?>

**Factorial**

<?php

$numero = 5;

$factorial = 1;

while ( $numero > 1 ) {

   $factorial = $factorial \* $numero;

   $numero = $numero - 1;

}

echo ' El factorial de ' . $numero . ' es ' . $factorial.

?>

**Numeros del 1 al 10**

<?php

$i = 1;  
while ($i <= 10):  
    echo $i;  
    $i++;  
endwhile;  
?>

**Conceptos**

**Herencia:** En programación orientada a objetos, la herencia es, después de la agregación o composición, el mecanismo más utilizado para alcanzar algunos de los objetivos más preciados en el desarrollo de software como lo son la reutilización y la extensibilidad.

**Polimorfismo:** En programación orientada a objetos, el polimorfismo se refiere a la propiedad por la que es posible enviar mensajes sintácticamente iguales a objetos de tipos distintos. El único requisito que deben cumplir los objetos que se utilizan de manera polimórfica es saber responder al mensaje que se les envía.

**Abstracción:** La abstracción consiste en aislar un elemento de su contexto o del resto de los elementos que lo acompañan. En programación, el término se refiere al énfasis en el "¿qué hace?" más que en el "¿cómo lo hace?"

**Encapsulamiento:** La encapsulación es un mecanismo para reunir datos y métodos dentro de una estructura ocultando la implementación del objeto, es decir, impidiendo el acceso a los datos por cualquier medio que no sean los servicios propuestos.

**Clases:** En informática, una clase es una plantilla para el objetivo de la creación de objetos de datos según un modelo predefinido. Las clases se utilizan para representar entidades o conceptos, como los sustantivos en el lenguaje.

**Atributos:** Los atributos son las características individuales que diferencian un objeto de otro y determinan su apariencia, estado u otras cualidades. Los atributos se guardan en variables denominadas de instancia, y cada objeto particular puede tener valores distintos para estas variables.

**Objeto:** Se trata de un ente abstracto usado en programación que permite separar los diferentes componentes de un programa, simplificando así su elaboración, depuración y posteriores mejoras.  
Los objetos integran, a diferencia de los métodos procedurales, tanto los procedimientos como las variables y datos referentes al objeto.  
A los objetos se les otorga ciertas características en la vida real. Cada parte del programa que se desea realizar es tratado como objeto, siendo así estas partes independientes las unas de las otras. Los objetos se componen de 3 partes fundamentales: métodos, eventos y atributos.

**Operadores lógicos**

Los operadores lógicos se usan para combinar dos valores Booleanos y devolver un resultado verdadero, falso o nulo. Los operadores lógicos también se denominan operadores Booleanos.

| **Ejemplo** | **Nombre** | **Resultado** |
| --- | --- | --- |
| $a and $b | And (y) | **true** si tanto *$a* como *$b* son **true**. |
| $a or $b | Or (o inclusivo) | **true** si cualquiera de *$a* o *$b* es **true**. |
| $a xor $b | Xor (o exclusivo) | **true** si *$a* o *$b* es **true**, pero no ambos. |
| ! $a | Not (no) | **true** si *$a* no es **true**. |
| $a && $b | And (y) | **true** si tanto *$a* como *$b* son **true**. |
| $a || $b | Or (o inclusivo) | **true** si cualquiera de *$a* o *$b* es **true**. |

**Operadores aritméticos**

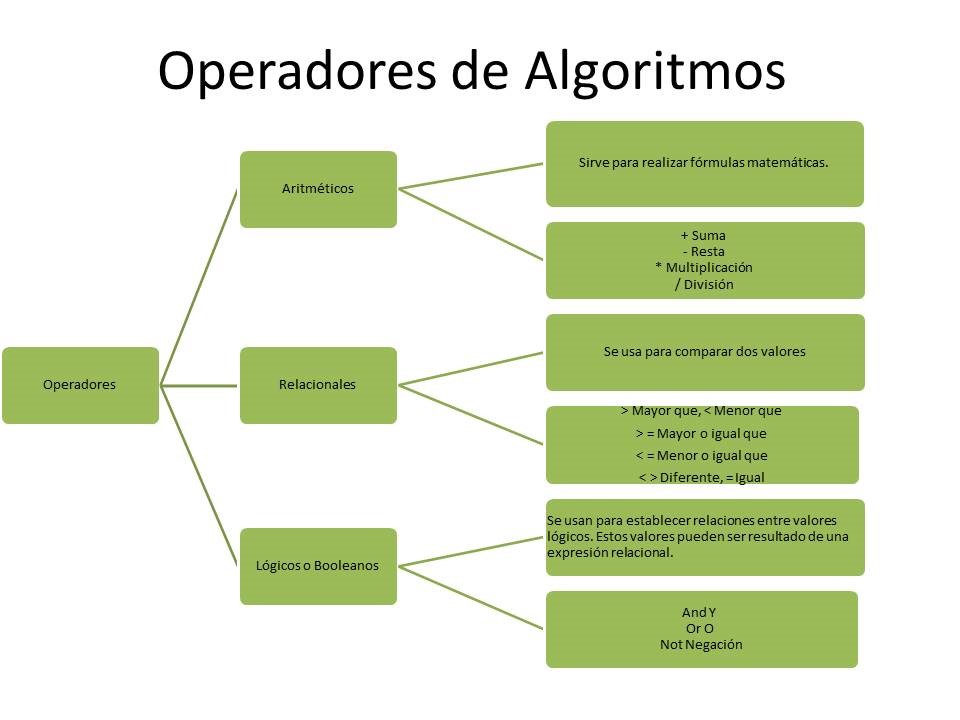
Los operadores aritméticos son aquellos que "manipulan" datos numéricos, tanto enteros como reales. Hay 2 tipos de operadores aritméticos: **unarios** y **binarios**. Los operadores unarios se anteponen a la expresión aritmética, y son los operadores de signo. Los operadores binarios se sitúan entre 2 expresiones aritméticas.

|  | | |
| --- | --- | --- |
| **Ejemplo** | **Nombre** | **Resultado** |
| +$a | Identidad | Conversión de *$a* a int o float según el caso. |
| -$a | Negación | Opuesto de *$a*. |
| $a + $b | Adición | Suma de *$a* y *$b*. |
| $a - $b | Sustracción | Diferencia de *$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*. |
| $a \*\* $b | Exponenciación | Resultado de elevar *$a* a la potencia *$b*ésima. Introducido en PHP 5.6. |

**Operadores Relacionales**

Estos operadores nos ayudan a, lo que podría llamarse, establecer una relación lógica entre dos valores por medio de una comparación, si esta relación es correcta la expresión se considera verdadera, por consiguiente al ser incorrecta automáticamente se considerara falso

|  |  |  |
| --- | --- | --- |
| Símbolo | Descripción | Ejemplo |
| == | Igual a. Usado para realizar comparaciones entre dos miembros | a == b |
| > | Mayor que. Compara si el termino de la izquierda es mayor que el de la derecha | a > b |
| < | Menor que. Compara si el termino de la izquierda es menor que el de la derecha | a < b |
| >= | Mayor o igual a. Es verdadero cuando el miembro izquierdo es mayor o igual que el derecho | a >= b |
| <= | Menor o igual a. Resulta verdadero si el termino izquierdo es menor o igual que el derecho | a <= b |
| != | No igual. Es decir el miembro de la izquierda es diferente al de la derecha | a != b |



**If anidado**

•          Los estatutos if implementan decisiones que implican una o dos alternativas, un estatuto if es anidada cuando la sentencia de la rama verdadera o la rama falsa es a su vez una sentencia if.

•          Un estatuto if anidada se puede utilizar para implementar decisiones con varias alternativas o multi-alternativas

**Sintaxis:**

**if (condición1)**

**estatuto1**

**else**

**if (condición2)**

**estatuto2**

**•**

**•**

**•**

**else**

**if (condiciónn)**

**estatuton**

**else**

**estatutoe**

Si tenemos el siguiente fragmento de código:

      if (condición 1)  
      if (condición 2)  
              estatuto;  
      else               // este else pertenece al if de la condición 2, pues se

                                // se asocia al if más cercano         
           estatuto;

Si queremos que el else pertenezca al primer if debemos poner:

      if (condición 1)  
      {  
         if (condición 2)  
               estatuto;  
      }  
      else               // con el uso de llaves cerramos el if anidado y el else  
         estatuto;     // pertenece al primer if

**Migraciones y estructura**

Las migraciones son como el control de versión para tu base de datos, permiten que tu equipo modifique y comparta el esquema de base de datos de la aplicación. Las migraciones son emparejadas típicamente con el constructor de esquema de Laravel para construir el esquema de base de datos de tu aplicación. Si inclusive has tenido que decirle a un miembro de equipo que agregue una columna manualmente a sus esquemas de bases de datos local, has encarado el problema que solucionan las migraciones de base de datos.

La clase [facade](https://styde.net/laravel-6-doc-facades/) Schema de Laravel proporciona soporte de base de datos orientado a la programación orientada a objetos para la creación y manipulación de tablas a través de todos los sistemas de bases de datos soportados por Laravel.

**Generando migraciones**

Para crear una migración, usa el [comando Artisan](https://styde.net/laravel-6-doc-consola-artisan/) make:migration:

php artisan make:migration create\_users\_table

La nueva migración estará ubicada en tu directorio database/migrations. Cada nombre de archivo de migración contiene una marca de tiempo, la cual permite que Laravel determine el orden de las migraciones.

Las opciones --table y --create también pueden ser usadas para indicar el nombre de la tabla y si la migración estará o no creando una nueva tabla. Estas opciones rellenan previamente el archivo stub de migración generado con la tabla especificada:

php artisan make:migration create\_users\_table --create=users

php artisan make:migration add\_votes\_to\_users\_table --table=users

Si prefieres especificar una ruta de directorio de salida personalizada para la migración generada, puedes usar la opción --path al momento de ejecutar el comando make:migration. La ruta de directorio dada debe ser relativa a la ruta de directorio base de tu aplicación.

**Estructura de migración**

Una clase de migración contiene dos métodos: up y down. El método up es usado para agregar nuevas tablas, columnas, o índices para tu base de datos, mientras el método down debería revertir las operaciones ejecutadas por el método up.

Dentro de ambos métodos puedes usar el constructor de esquema de Laravel para crear y modificar expresivamente las tablas. Para aprender sobre todos los métodos disponibles en el constructor Schema, [inspecciona su documentación](https://styde.net/laravel-6-doc-base-de-datos-migraciones/#creating-tables). Por ejemplo, la siguiente migración crea una tabla flights:

**<**?php

use Illuminate\Support\Facades\Schema;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Database\Migrations\Migration;

class CreateFlightsTable extends Migration

**{**

/\*\*

\* Run the migrations.

\*

\* @return void

\*/

public function up**()**

**{**

Schema::create**(**'flights', function **(**Blueprint $table**)** **{**

$table->bigIncrements**(**'id'**)**;

$table->string**(**'name'**)**;

$table->string**(**'airline'**)**;

$table->timestamps**()**;

**})**;

**}**

/\*\*

\* Reverse the migrations.

\*

\* @return void

\*/

public function down**()**

**{**

Schema::drop**(**'flights'**)**;

**}**

**}**

**Ejecutando migraciones**

Para ejecutar todas tus maravillosas migraciones, ejecuta el comando Artisan migrate:

php artisan migrate

Si estás usando [La máquina virtual de Homestead](https://styde.net/laravel-6-doc-homestead/), deberías ejecutar este comando desde dentro de tu máquina virtual.

**Forzando las migraciones para ejecutar en producción**

Algunas operaciones de migración son destructivas, lo que significa que pueden causar que pierdas tus datos. Con el propósito de protegerte de ejecutar estos comandos contra tu base de datos de producción, recibirás un mensaje de confirmación antes que los comandos sean ejecutados. Para forzar que los comandos se ejecuten sin retardo, usa el indicador --force.

php artisan migrate --force

**Revertir migraciones**

Para revertir la operación de migración más reciente, puedes usar el comando rollback. Este comando reversa el último «lote» de migraciones, los cuales pueden incluir archivos de migración múltiples.

php artisan migrate:rollback

Puedes revertir un número limitado de migraciones proporcionando la opción step al comando rollback. Por ejemplo, el siguiente comando revertirá los cinco «lotes» de migraciones más recientes:

php artisan migrate:rollback --step=5

El comando migrate:reset revertirá todas las migraciones de tu aplicación:

php artisan migrate:reset

**Roll Back & Migrate usando un único comando**

El comando migrate:refresh revertirá todas tus migraciones y después ejecutará el comando migrate. Este comando vuelve a crear efectivamente tu base de datos entera:

php artisan migrate:refresh

// Refresh the database and run all database seeds...

php artisan migrate:refresh --seed

Puedes revertir y volver a migrar un número limitado de migraciones proporcionando la opción step al comando refresh. Por ejemplo, el siguiente comando revertirá y volverá a migrar las cinco migraciones más recientes:

php artisan migrate:refresh --step=5

**Eliminando todas las tablas y migrar**

El comando migrate:fresh eliminará todas las tablas de la base de datos y después ejecutará el comando migrate:

php artisan migrate:fresh

php artisan migrate:fresh --seed