UNIVERSIDAD DON BOSCO	**UNIVERSIDAD DON BOSCO** **FACULTAD DE INGENIERIA** **ESCUELA DE COMPUTACION**
CICLO 1	GUIA DE LABORATORIO #5 **Nombre de la Practica:** **Programacion Orientada a Objetos y Expresiones regulares** **Lugar de Ejecución:** **Centro de Cómputo** **Tiempo Estimado:** **2 horas con 30 minutos** **MATERIA:** **Desarrollo de Aplicaciones Web con Software Interpretado en el Servidor (DSS404)**

I. OBJETIVOS

Con la realización de esta guía de práctica el estudiante estará en capacidad de:

- Comprender el concepto de *expresión regular* y el uso que puede darles para resolver distintos problemas.
- Construir patrones de expresiones regulares con distintos propósitos, como validación, búsqueda y reemplazo en cadenas.
- Utilizar las funciones de PHP diseñadas para trabajar con expresiones regulares.
- Crear aplicaciones informáticas basadas en la utilización de expresiones regulares.

II. INTRODUCCION TEORICA

¿Qué son las expresiones regulares?

Una *expresión regular* puede ser considerada como una secuencia o patrón de caracteres que representa o describe a un conjunto de cadenas de caracteres sin enumerar explícitamente sus elementos. Estos patrones son utilizados para ser comparados contra una cadena de texto sobre la cual se necesita hacer una búsqueda para encontrar posibles coincidencias.

Buena cantidad de lenguajes de programación utilizan las expresiones regulares porque poseen una indiscutible potencia para procesar texto cuando se intenta localizar caracteres que se ajusten a un formato o para hacer sustitución de cadenas dentro de un texto.

La utilización que se le da a las expresiones regulares suele ser:

1. Verificación de datos que son enviados desde un formulario antes de insertarlos en una base de datos.
2. Localización de subcadenas dentro de una cadena de texto más compleja.
3. Búsqueda y reemplazo de secuencias de caracteres en una cadena de varias líneas, permitiendo incluso múltiples reemplazos.

¿Por qué utilizar expresiones regulares?

Es muy probable que logre hacer con funciones de cadena y con un ciclo o lazo el mismo trabajo que con una expresión regular; sin embargo, cuando se hace uso de expresiones regulares se realiza código más compacto e intuitivo cuando se necesita hacer uso de coincidencia de patrones para realizar alguna tarea con cadenas de texto. Además de esto, la ejecución de una expresión regular será mucho más rápida que hacer uso de las funciones de cadena y los ciclos o lazos.

Conceptos generales de expresiones regulares

Comencemos por definir lo que es un **patrón**, que indirectamente viene siendo un sinónimo de expresión regular. Un **patrón** es el modelo que se define para ser comparado contra un conjunto de cadenas de caracteres. Ese modelo puede estar constituido por **caracteres normales** (o literales) y por **metacaracteres**, que no son otra cosa más que caracteres con significado especial.

Por ejemplo, en la expresión regular "^1*a$" (las comillas no son parte de la expresión regular, solo la delimitan) el "1" y la "a" son caracteres normales o literales, mientras que los símbolos: "^", "*" y "$" son **metacaracteres** o caracteres con un significado especial que permiten que esta expresión regular se pueda comparar contra muchas otras cadenas en busca de coincidencias.

Patrones de caracteres

Dentro de los patrones que representan a un único carácter, podemos distinguir dos tipos:

- Los caracteres que se representan a sí mismos.
- Las categorías o clases de caracteres.

La expresión regular más simple que se puede construir es la que está formada por un único carácter que se representa a sí mismo. Por ejemplo, la expresión: "a", describe a todas las cadenas que contengan el carácter "a", esto significa que las cadenas: "Lola", "amiga", "luna", "piano" y "sal" coincidirán con el patrón anterior, porque todas llevan al menos una "a", sin importar en qué posición esté ese carácter dentro de esas cadenas.

Las categorías o clases de carácter son patrones que definen un conjunto de caracteres con los que un carácter de una cadena puede concordar. La clase más general es el patrón . (punto), que representa cualquier carácter del juego de caracteres, a excepción de algún carácter especial dependiente de la implementación utilizada. Cuando se quiera representar una categoría más específica como pueden ser las letras minúsculas, las vocales o los dígitos, se utilizan los metacaracteres "[" y "]" (los corchetes) para encerrar entre ellos el subconjunto de caracteres específicos, de la siguiente forma:

Expresión regular (o patrón)	Descripción
`[abcdefghijklmnopqrstuvwxyz]`	El alfabeto inglés
`[aeiou]`	Las vocales
`[0123456789]`	Los dígitos
`[ÁÉÍÓÚ]`	Vocales mayúsculas acentuadas

Para ahorrar escritura, se puede utilizar el metacaracter "-" para indicar un rango de caracteres. Así se coloca el primer carácter del rango, luego el metacaracter

Agrupamiento de patrones

Los patrones que representan un carácter se pueden agrupar para formar expresiones regulares más complejas y es esta característica las que les da una gran potencia para el procesamiento de cadenas.

Entre los distintos tipos de agrupamiento de patrones se pueden mencionar:

- Las secuencias.
- Los cuantificadores.
- Los paréntesis.
- Las alternativas.
- La fijación de patrones.
- Los caracteres escapados.
- La precedencia.
- Las operaciones.

Programación Orientada a Objetos

El lenguaje PHP 5 ha sido rediseñado por completo para dar a los programadores todas las herramientas que un verdadero lenguaje orientado a objetos debe poseer.

La Programación Orientada a Objetos (POO) es un enfoque de programación en el que el diseño y desarrollo del software se fundamenta en la modelización de las características y comportamientos de elementos o sucesos reales o abstractos mediante el uso de clases y objetos. Una clase es una descripción genérica o plantilla de un determinado tipo de objetos. Los objetos, por su parte, se crean a partir de estas clases. De manera que cada vez que se crea un objeto, se dice que se está creando una instancia de la clase. Los objetos, por tanto, poseen dos características importantes, que son: el **estado** y el **comportamiento**. El estado se define mediante un conjunto de **propiedades**, en tanto que, el comportamiento se implementa mediante **métodos**. Puede considerarse a los métodos como las operaciones que son posibles llevar a cabo con las propiedades del objeto.

Definicion de una Clases

Para crear una clase con PHP5 se utiliza la palabra reservada `class`, seguida por el nombre que se le asignará a la clase. Vea la siguiente sintaxis:

```
class nombre_clase {
    //propiedades de la clase;
    //métodos de la clase;
}
```

En el interior de la clase, se definen a los miembros de la clase (**atributos** y **metodos**):

- Las **campos** o **atributos**, se declaran mediante el uso de variables.
- Los **métodos** o **acciones** de la clase se crean declarando funciones dentro de la definición de la clase.

A cada miembro de una clase se le debe especificar un *control/nivel de acceso* mediante el uso de las palabras reservadas: `public`, `private` o `protected`.

Control de acceso a los miembros de una clase

En PHP 5, se han incluido los siguientes *especificadores de acceso* para los miembros de una clase:

- `public`: este es el modificador de acceso predeterminado e indica que un campo o método será accesible desde cualquier punto del script.
- `private`: indica que el miembro de la clase se podrá acceder únicamente desde el interior de la clase.
- `protected`: utilizado al implementar Herencia. Significa que el campo o método, sólo será accesible desde el interior de una clase o desde sus clases derivadas.

A continuación se define un ejemplo del uso de estos especificadores de acceso:

```
class claseEjemplo {
   //Campos | Atributos
   public $publicprop = 'Soy atributo público';
   private $privateprop = 'Soy atributo privado';
   protected $protectedprop = 'Soy atributo protegido';
   //Métodos
   function metodoEjemplo(){ //por defecto, se aplicaca modificador public
      echo $this->publicprop;
      echo $this->privateprop;
      echo $this->protectedprop;
   }
}
//Instanciando un objeto de la clase claseEjemplo
$obj1 = new claseEjemplo();
//intenta modificador de campos del objeto
$obj1->publicprop = 'Soy campo público';
$obj1->privateprop = 'Soy privado';  //Generará un error.
$obj1->protected = 'Soy protegido';  //Generará un error.
$obj1->metodoEjemplo();
```

Constructores y destructores

Un constructor es un método especial que es invocado de forma automática, cada vez que se crea una nueva instancia de la clase; es decir, cada vez que se crea un nuevo objeto a partir de la clase. Su modificador de acceso siempre es publico (`public`).

El constructor por defecto realiza tareas de inicialización como establecer atributos con valores de inicio apropiados o crear otros objetos necesarios.

Los constructores en PHP5 tienen un nombre especial que se muestra a continuación:

```
function __construct(){
    //Establecer valores iniciales a campos;
}
```

En la versión 4, los constructores tenían el mismo nombre de la clase.

```
function nombreClase(){
    //Establecer valores iniciales a campos;
}
```

Los **destructores** tienen el propósito de liberar recursos del servidor al terminar de ejecutar un script en el que se han creado objetos a partir de una clase.

Además de esto, permiten implementar alguna funcionalidad concreta antes de que se destruya la clase.

Un destructor en PHP5 se construye de la siguiente forma:

```
function __destruct(){
    //Liberar recursos del sistema;
}
```

Creacion de Objetos (Instancias de Clases)

Una vez definida una clase, para utilizar a sus miembros (Atributos y Metodos) se deben crear *Objetos* basados en estas. A los objetos también se los denomina *Instancias de Clase*. El operador utilizado para crear objetos basados en una Clase es new. Observe un ejemplo a continuación:

```
class Automovil {
  //Miembros de clase
  //campos
  $marca;
  $modelo;
  $precio;
  //método constructor
  function _construct() {
    this->marca=”pendiente”;
    this->modelo= “pendiente”;
    this->precio=10000;
  }
}
```

Luego, utilizando el operador new, se pueden crear a 2 instancias:

```
$auto1;
$auto1 = new Automovil( );

$auto2 = new Automovil( );
```

Encapsulamiento de datos

El uso de los *especificadores de acceso* permite implementar el concepto de *Encapsulamiento de datos*, el cual es la manera de cómo se protegerán los datos almacenados en los campos cuando se creen instancias de una Clase.

Examine el siguiente código:

```
class persona {
    //Atributos
    private $nombrecompleto;
    //Métodos de la clase
    function asignarNombre($nombre, $apellido){
       $this->nombrecompleto = $nombre . " " . $apellido;
    }
    function decirNombre(){
       return $this->nombre;
    }
}
```

Y a continuación, se crea la siguiente instancia de la clase persona y se accede a su campo $nombrecompleto:

```
$unapersona = new persona();
$unapersona->nombrecompleto = "Jorge Bustamante";
```

Al ejecutar el script obtendríamos un error, ya que la propiedad `$nombrecompleto` es privada. Por lo tanto, sólo puede accederse a ella desde alguno de los métodos definidos dentro de la clase `persona`.

Ahora bien, si realizamos un cambio en la declaración de la propiedad $nombrecompleto, colocando el modificador de acceso `public`, en lugar de `prívate`. El mismo código anterior funcionaría correctamente.

```
class persona {
    //Propiedades
    public $nombrecompleto;
    //Métodos de la clase
    function asignarNombre($nombre, $apellido){
       $this->nombrecompleto = $nombre . " " . $apellido;
    }
    function decirNombre(){
       return $this->nombre;
    }
}
```

Y luego, se crea una instancia de esta clase `persona` y se accede a su campo público:

```
$unapersona = new persona();
$unapersona->nombrecompleto = "Jorge Bustamante";
```

El problema que puede ocurrir con esta solucion es que usuario podría asignar valores incorrectos al campo `nombrecompleto`. Para evitarlo, al crear objetos de esta clase `persona`, se debe ocultar (con el modificador `private`) el campo y definir un método de propiedad publico (usando el modificador `public`).

Este método de propiedad publico recibirá el valor que usuario intenta asignar al campo dentro del objeto, lo verifica y si este es correcto, lo asignara internamente al campo correspondiente. De lo contrario, no alterara el valor del campo oculto.

Miembros estáticos de clase

Un campo o método de tipo estático pertenece a la clase en la que está definido, no a los objetos creados a partir de dicha clase. De modo que, no puede ser accedido desde un objeto ni ser redefinido en las clases derivadas. No obstante, es posible llamar a un miembro estático desde fuera del contexto de un objeto.

Para declarar un miembro estático se hace uso de la palabra reservada *static* y debe colocarse después de la declaración de visibilidad o acceso para el miembro, si es que existe, tal y como se muestra:

```
prívate static $idLibro;
public function Saludar(){ }
```

Por pertenecer a la clase y no a sus instancias/objetos, los miembros estáticos deben ser accedidos dentro del contexto de la clase, empleando el nombre de clase reservado **`self`** y el operador de resolución de ámbito **`::`** Esta palabra reservada permite hacer referencia a la clase actual en la que se encuentra la

declaración. Por ejemplo, para hacer referencia dentro de la clase al campo estático $idLibro, definido anteriormente, debe utilizar una instrucción como la siguiente:

```
self::$idLibro++;
```

Ahora bien, para acceder a un campo desde fuera del contexto de la clase, su visibilidad debe ser publica. Luego, debe hacer uso del nombre de la clase, seguida del operador de resolución de ámbito (**::**) y, a continuación, el nombre del miembro.

En el caso de los métodos estáticos, se procede de igual forma, ya que pertenecen a la clase y no a los objetos. Así que los métodos pueden ser llamados desde fuera del contexto de la clase utilizando la notación:

```
nombreClase::metodo();
```

Recuerde, que al igual que con los miembros estáticos, no se puede utilizar la variable especial `$this` dentro de los métodos estáticos, puede emplearse `self::`

Constantes de clase

A partir de PHP5 es posible definir constantes dentro de una clase. Estas pueden ser utilizadas para almacenar valores escalares que permanecerán invariables a lo largo de la ejecución del script. Por ejemplo, rutas de direcciones URL, valores constantes en ciertos cálculos (constante PI), nombres de bases de datos, el valor de un impuesto (como el IVA).

Las constantes de clase –como las propiedades estáticas— pertenecen a la clase en la que están definidas, no a los objetos de dicha clase. Por esta razón, no puede accederse a través del objeto a los valores constantes definidos en la clase. El acceso debe realizarse, desde dentro de la clase, a través del nombre de clase reservado self, o bien, desde fuera de la misma a través del nombre de la clase. Por ejemplo:

```
class miClase{
   …
   const IVA = 0.13;
   …
}
```

Si accedemos a la constante desde dentro de un método de la clase, entonces podemos utilizar la siguiente instrucción:

```
$precio = $precioSinIva * self::IVA;
```

Para acceder a la misma constante desde fuera del contexto de la clase, debe utilizar el nombre de la clase y luego el operador de resolución de ámbito:

```
$precio = $precioSinIva * nombreClase::IVA;
```

Constantes mágicas

PHP proporciona dos constantes de clase especiales, denominadas también, *constantes mágicas*. Estas constantes están disponibles desde dentro de cada clase. Estas constantes son `__CLASS__`y `__METHOD__`.

La constante `__CLASS__`almacena el nombre de la clase en el que es empleada. Este nombre también puede ser obtenido a través de la función `get_class()`.

Por su parte, `__METHOD__`contiene el nombre de clase y método desde el que se accede. Así, si se accede a dicha constante desde el método denominado metodo1 de la clase clase1, la constante mágica contendrá la cadena 'clase1::metodo1'. Adicionalmente, se dispone de la función `get_class_methods($objeto)`, que recibe como argumento un nombre de clase o un objeto y devuelve una matriz indexada conteniendo todos los métodos definidos en la clase correspondiente.

Carga automática de clases

Cuando se trabaja con clases produciendo un script independiente por cada definición de clase, es necesario incluir cada una de estas definiciones de clase en el script principal de la aplicación final. En el caso que el número de clases definidas sea considerable, puede llegar a ser molesto tener que crear una instrucción `include` o `require` por cada una de las clases necesarias para el script.

Para brindar un mecanismo para que los programadores puedan cargar las clases en tiempo de ejecución o cargar las clases de forma dinámica o automática, en PHP 5 se ha incorporado un método conocido como método mágico, denominado ***spl_autoload_register()***. La llamada a este método se realiza de forma automática, no se puede invocar desde código. Esta llamada se produce cuando se intenta instanciar a una clase, sin que exista en el script una definición para esta. En ese momento, PHP buscará el método ***spl_autoload_register()*** para localizar la definición de esa clase en un archivo externo. Si no se encuentra un archivo con la definición de la clase ni con este método, entonces el script terminará con un error fatal.

El objetivo de esta técnica es facilitar al programador la inclusión de clases sin tener que recurrir a la inclusión de un sin número de instrucciones include o require.

La implementación de esta técnica requiere que se programe el método ***spl_autoload_register()*** en el script. Una implementación simple sería haciendo uso de funciones anónimas, como la siguiente:

```
spl_autoload_register(function($class) {
    include 'classes/' . $class . '.class.php';
});
```

El método `spl_autoload_register()` será invocado de forma automática cuando se instancie a una clase para crear un objeto, de esta forma:

```
$objeto = new clase1();
```

Si no se encuentra en el script o en un script externo incluido mediante una instrucción include o require, PHP intentará, como último recurso, encontrar la definición de la clase en el método `spl_autoload_register()`. Si no la encuentra, entonces lanzará un error fatal.

Hay algunas consideraciones a tomar en cuenta para usar el método `spl_autoload_register()`:

1. El método **`spl_autoload_register()`** se autoejecuta, lo que significa que no puede ser invocado por el programador mediante código PHP.
2. El método recibe como argumento una función anónima, que a su vez, recibe como parámetro el nombre de la clase que se ha intentado instanciar sin que haya sido encontrada su definición en el script. PHP es quien envía el valor adecuado de este argumento, siendo este el nombre de la clase.
3. Dentro de la función **`spl_autoload_register()`** se intenta incluir un script que debe haber sido nombrado igual que la clase para que sea encontrado.

4. Por último, para que esto funcione, debe haber creado un script PHP, por cada definición de clase.

Sobrecarga de propiedades y métodos

La **sobrecarga de propiedades** en PHP puede ser implementada a través de los métodos especiales, llamados también mágicos, `__set()` y `__get()`. Estos métodos son invocados de forma automática cuando se intenta acceder a una propiedad inexistente de un objeto.

La sintaxis de estos métodos es la siguiente:

```
void __set(string $name, mixed $value);
mixed __get(string $name);
```

El método ***__set()*** se utiliza para asignar un valor, dado por $value, a la propiedad que se ha intentado acceder y que no existe en la definición de la clase.

El método ***__get()*** permite recuperar el valor de una propiedad a la que se ha intentado acceder, sin que exista en la definición de la clase.

Un aspecto importante a considerar es el hecho que primero debe establecerse el valor, antes de intentar accederlo. Esto significa, que primero debe hacerse la asignación del valor en la propiedad y luego, intentar obtener ese valor.

Los métodos ***__set()*** y ***__get()*** son llamados únicamente si la propiedad referenciada no existe en el objeto. Si la propiedad ya fue establecida en una ejecución previa de ***__set()***, ya no se volverán a ejecutar ni ***__get()***, ni ***__set()***.

La **sobrecarga de métodos** se puede implementar con el método mágico ***__call()***. Mediante este método se podrá acceder a métodos no definidos en el objeto.

La sintaxis del método ***__call()*** es la siguiente:

```
mixed __call(string $name, array $arguments);
```

Podemos comprender mejor el funcionamiento de la sobrecarga de miembros de una clase, mediante un ejemplo simple:

```
class sinPropiedades {
   function __set($propiedad, $valor){
      echo "Asignamos $valor a $propiedad";
      $this->propiedad = $valor;
   }
   function __get($propiedad){
      echo "Acceso a la propiedad $propiedad(clase ", __CLASS__,")\n";
      return $this->propiedad;
   }
   function __call($metodo, $parametros){
      echo "Acceso al método $metodo (clase, __CLASS__, ")\nArgumentos:\n",
      var_dump($parametros), "\n";
   }
}  //Fin clase sinPropiedades
```

Para probar nuestra clase sinPropiedades, tenemos el siguiente script:

```
//Creación de un nuevo objeto sinPropiedades
$obj = new sinPropiedades();
//Asignando valores a dos propiedades no definidas
$obj->nombre = "Sergio";
$obj->edad = 25;
//Hacer un volcado del objeto
echo var_dump($obj), "\n";
//Acceder a las propiedades sobrecargadas y a otra inexistente
echo 'Nombre: ', $obj->nombre, "\n";
echo 'Edad: ', $obj->edad, "\n";
echo 'Apellido: ', @$obj->apellido, "\n";
//Intentar ejecutar un método inexistente
echo $obj->darNombre('Sergio', 'Pérez', 30);
```

Herencia de clases

La **herencia** en Programación Orientada a Objetos es la relación que se da entre dos clases por la cual una de ellas, a la que se denominará **clase hija**, **subclase** o **clase derivada**, además de poseer sus propias propiedades y métodos (o incluso constantes), tiene a su disposición; o lo que es lo mismo, hereda, los miembros definidos en la otra, denominada **clase padre** o **superclase**.

También se puede ver la **herencia** como la capacidad que tiene una clase de extender su funcionalidad. Esto se debe a que una clase hija también puede volver a definir algunos o todos los métodos, propiedades y constantes de la clase padre para proporcionar una funcionalidad adicional o diferente a la clase.

Ejemplo:

```
class computer {   //Esta es la superclase
   //Propiedades de la superclase
   private $password;  //propiedad visible únicamente por esta clase
   protected $userID;  //propiedad visible por esta clase y sus clases derivadas
   public $printer;

   //Constructor de la superclase
   function __construct(){
      echo "Llamada al constructor del padre:<br>\n";
      $this->userID = "estudiante";
      $this->password = "Pa$$w0rd";
   }
}

//Extendiendo la clase computer
class laptop extends computer{   //Subclase
   //Propiedades de la clase hija
   public $brand;
   public $weight;
   private $password = "newPa$$w0rd";

   //Constructor de la clase hija
```

```
   function __construct($brand, $weight){
      parent::construct();  //Llamada al constructor de la clase padre
      echo "Llamada al propio constructor de la clase hija
      $this->brand = $brand;
      $this->weight = $weight;
   }
}

//Aplicación que utiliza la clase
$pc = new computer();
$portable = new laptop("Waio","6.0");
$pc->printer = "Lexmark 1100";
$portable->printer = "Epson Stylus 3i";
//echo "$portable->password<br>\n";    //Arrojará un error fatal
//echo "$pc->password<br>\n";    //Arrojará también un error fatal
echo "<pre>";
//Obtenemos las propiedades públicas disponibles
print_r(get_object_vars($pc));
print_r(get_object_vars($portable));
echo "</pre>";
```

Clases y métodos abstractos

La **abstracción** es un recurso que se utiliza para brindar mayor control sobre el proceso de herencia. La característica principal de una clase abstracta es que no puede instanciarse, lo que quiere decir que no puede utilizarse de forma directa, únicamente se puede acceder a sus miembros a través de una subclase o clase derivada de esta. A esta subclase se le denomina también *clase concreta*.

Una **clase abstracta**, como cualquier clase padre, además de declarar e implementar campos, constantes y métodos, puede definir **métodos también abstractos**, los cuales no pueden ser implementados; es decir, no poseen código. La implementación de los métodos abstractos corresponde a las clases hijas.

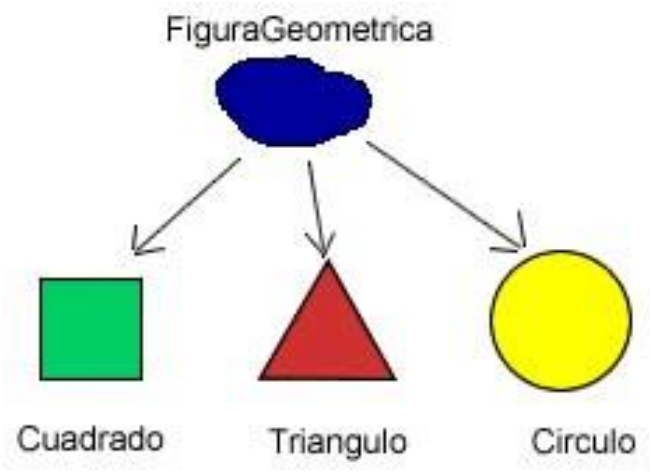
Además, las clases hijas pueden volver a declarar estos métodos abstractos, lo que significa que la responsabilidad de la implementación del método recaerá en el siguiente nivel jerárquico.

Veamos el siguiente ejemplo:

```
abstract class number {
   //Propiedades
   private $value;
   //Declaración de un método abstracto
   abstract public function value();
   public function reset(){
      $this->value = NULL;
   }
}

class integer extends number {
   //Propiedades de la clase derivada
   private $value;
   //Implementación del método abstracto
```

```
   public function value(){
      return (int)$this->value;
   }
}

//Probando la clase
$int = new integer();  //Se ejecutará correctamente
$num = new number();  //Lanzará un error
```

Un buen ejemplo de abstracción son las figuras geométricas. La figura geométrica es una clase abstracta, puesto que representa nada más un concepto. Todos entendemos lo que son las figuras geométricas, pero nadie puede decir que ha visto alguna, lo que pudiera haberse visto es un cuadrado, un triángulo, un rectángulo o un círculo, estas si son figuras geométricas concretas, con dimensiones específicas.



Interfaces

Las interfaces son similares en algunos aspectos a las clases abstractas. Permiten definir protocolos de comportamiento para los objetos en cualquier punto de la jerarquía de clases. Una interfaz permite definir un nombre a un conjunto de definiciones de métodos y a un conjunto de constantes. Una clase que cumple con una determinada interfaz debe implementar todos los métodos incluidos en la misma, adquiriendo, por tanto, un cierto comportamiento. La clase podría también, declarar alguno de los métodos definidos por la interfaz, como abstracto, forzando a que dicho método sea implementado por alguna de sus clases hijas. Además, una clase puede implementar más de una interfaz.

Una diferencia entre las clases abstractas y las interfaces es que mientras las primeras proporcionan un medio para expresar conceptos abstractos en programación, las segundas, se han diseñado para asegurar la funcionalidad dentro de una clase.

Para declarar una interfaz en PHP se utiliza la palabra clave **interface** y, a continuación, el identificador de la interfaz. De la siguiente forma:

```
interface printable {
   public function printme();
}
```

Para que una interfaz sea útil, debe ser implementada mediante una o más clases. También, es posible que una sola clase pueda implantar múltiples interfaces diferentes.

```
interface printable {
   public function printme();
}

interface Inumber {
   public function reset();
}

class integer implements printable, Inumber {
   private $value;
   function __construct($value) {
      $this->value = $value;
   }
   //Implementación de la interfaz printable
   public function printme() {
      echo (int)$this->value;
   }
   public function reset() {
      $this->value = NULL;
   }
   public function value() {
      return (int)$this->value;
   }
}

function resetNumber(Inumber $obj) {
   $obj->reset();
}
function printNumber(printable $obj) {
   $obj->printme();
}
```

Para probar la interfaz:

```
$entero = new integer(12);
printNumber($entero);
resetNumber($entero);
```

Polimorfismo

El **polimorfismo** es uno de los elementos clave o, dicho de otro modo, uno de los pilares de la Programación Orientada a Objetos. Con el **polimorfismo** es posible emplear un mismo método perteneciente a objetos de distinta clase, sin que importe realmente donde está implementado dicho método.

El siguiente ejemplo supone que están definidas las clases derivadas empleado, estudiante y bebé. Todas ellas hijas de la superclase persona. La clase hija empleado posee el método trabaja(), la clase estudiante

posee el método estudia() y la clase bebe, posee el método comeyduerme(). La idea es desarrollar una función que muestre la ocupación principal de cada persona. La función tendría la siguiente implementación:

```
function estadoPersona($persona) {
   if($persona instanceof empleado)
      echo $persona->trabaja();
   elseif($persona instanceof estudiante)
      echo $persona->estudia();
   elseif($persona instanceof bebe)
      echo $persona->comeyduerme();
   else
      echo "ERROR: Se desconoce la ocupación de esta persona.";
}
```

El operador instanceof se utiliza para determinar la clase a la que pertenece un objeto o si un objeto es una instancia de una determinada clase. También su utiliza para determinar si un objeto implementa alguna interfaz dada. Puede notar este tipo de implementación es poco escalable. Por ejemplo, si añadimos un nuevo tipo de persona a la aplicación, tendríamos que modificar el código de la función estadoPersona. Sin embargo, existe una mejor solución que consiste en hacer uso del polimorfismo. De modo, que cada una de las clases hijas implementará un método específico, denominado ocupacionPrincipal(). Así, dado un objeto de cualquiera de las tres clases hijas (empleado, estudiante y bebé), se podría realizar la llamada al método correspondiente utilizando la sintaxis:

```
$objeto->ocupacionPrincipal();
```

De este modo la función estadoPersona sería:

```
function estadoPersona($persona) {
   if($persona instanceof persona)
      echo $persona->ocupacionPrincipal();
   else
      echo "ERROR: Se desconoce la ocupación de esta persona.";
}
```

Manejo de excepciones

Una de las incorporaciones más interesantes de PHP 5 es el manejo de excepciones.

Las **excepciones** proporcionan un mecanismo para gestionar errores en el contexto de los objetos. También proporcionan ventajas significativas sobre las técnicas tradicionales de gestión de errores. Uno de los principales inconvenientes del manejo tradicional de errores es que la gestión del error está entrelazada con el flujo normal del código del script o programa. Incluir el tratamiento de errores en el punto donde este se produce provoca que el código sea poco flexible y difícil de reutilizar, ya que el mismo error puede precisar de tratamientos distintos en función de las circunstancias en las que se produzca.

La gestión de excepciones utilizando objetos intenta resolver estos problemas, permitiendo delegar el tratamiento de errores al lugar más apropiado, haciendo posible manejar múltiples condiciones de error en un único punto, separando el procesamiento de la excepción de la lógica del programa.

La clase Exception

En la práctica, las excepciones son instancias de clases que contienen información sobre el error que se ha producido durante la ejecución de la secuencia de comandos. PHP 5 proporciona internamente una clase denominada Exception. La definición de la clase Exception es la siguiente:

```
class Exception {
   protected $message;
   private $string;
   protected $code;
   protected $file;
   protected $line;
   private $trace;
   function __construct($message="", $code=0);
   function __toString();
   public function getFile();
   public function getLine();
   public function getMessage();
   public function getCode();
   public function getTrace();
   public function getTraceAsString();
}
```

Las excepciones en PHP contienen dos valores principales que son: una cadena con el mensaje que describe el error que se ha producido y un valor entero que representa el código del error. De forma predeterminada, PHP asignará automáticamente a la excepción la línea y el nombre del archivo donde se ha producido el error, así como una localización de pila que representa la ruta de acceso a la ejecución que ha resultado en error.

Arrojar y capturar excepciones

Como programadores, tenemos la posibilidad de derivar la clase Exception, para definir nuestras propias excepciones. Cuando se lance una excepción, se puede definir un mensaje descriptivo sobre el error y un código de error. Del resto de la clase no debemos preocuparnos como programadores, puesto que PHP maneja los métodos por nosotros.

Uno de los métodos más interesantes de la clase Exception es el método ***__toString()***, el cual pertenece a la categoría de los métodos conocidos como métodos mágicos. Al igual que ***__construct()***, ***__destruct()***, ***__clone()***, ***__autoload()*** y otros que hemos mencionado anteriormente. Este método es susceptible de ser sobrescrito en las clases derivadas de Exception.

Para lanzar una excepción se puede instanciar a la clase Exception o a una clase derivada de esta, incluso aunque no se haya creado un objeto como tal para su uso posterior. Para lanzar la excepción desde la secuencia de comandos se utiliza la sentencia throw junto con la instancia de la excepción a arrojar.

Veamos el siguiente ejemplo:

```
class throwExample {
   public function makeError(){
      throw new Exception("Este es un ejemplo de excepción");
   }
}
```

```
$inst = new throwExample();
$inst->makeError();
```

El manejo de excepciones en PHP 5 se lleva a cabo utilizando una estructura de control, denominada try/catch. Dentro del bloque try se incluye el código de programa que podría llegar a producir una excepción. Todo bloque try dentro de la secuencia de comando debe tener asociado, al menos un bloque catch. Con el bloque catch se realiza la gestión de las excepciones. La sintaxis es la siguiente:

```
try {
   //código que puede generar una excepción
}
catch(classException1 $e1) {
   //Procesamiento de las excepciones de classException1
}
[catch(classException2 $e2){
   //Procesamiento de las excepciones de classException2
}]
```

El funcionamiento es, en teoría, simple. Si la excepción es una instancia de la clase capturada en el primer bloque catch, será procesada en ese punto. En caso contrario, se buscará otro bloque catch que admita la clase de excepción generada u otra estructura por encima de try/catch en la que capturarla. Si no se encuentra dicho bloque, se generará un error fatal.

El siguiente ejemplo muestra cómo manejar el conocido caso de la división por cero haciendo uso de excepciones.

```
<?php
   try {
      $a=15;
      //$a=0; //quitar los comentarios luego a esta línea para probar las excepciones
      if($a==0){
         $error  = "<p style=\"font:bold 10pt Verdana;color:red;\">";
         $error .= "El divisor debe de ser diferente de CERO.</p>";
         throw new Exception($error);
         echo "<p>ESTA PARTE DE CODIGO NO SE EJECUTARÁ";
         echo "TODO ESTO SERÍA IGNORADO.</p>";
      }
      else{
         @$resultado=number_format(10/$a, 4, '.', ',');
         echo "<p style=\"font:bold 12pt Verdana;color:Green\">Respuesta de la division: ";
         echo $resultado . "<br>\n";
         echo 'No se ha producido ninguna excepción.</p>';
      }
   }
   catch (Exception $e){
      echo "<br>ERROR<br><br>";
      echo "Descripcion : ".$e->getMessage(). "<br>";
      echo "Codigo de la Excepcion : ".$e->getCode(). "<br>";
```

```
      echo "Archivo del error : ".$e->getFile(). "<br>";
      echo "Línea de llamado de Exception : ".$e->getLine(). "<br>";
   }
   // Continue execution
   echo "<br><span style=\"font:bold 10pt Arial;color:Blue;\">";
   echo "FIN DEL PROGRAMA</span>";
?>
```

III. MATERIALES Y EQUIPO

Para la realización de la guía de práctica se requerirá lo siguiente:

No.	Requerimiento	Cantidad
1	Guía de práctica #5: Expresiones regulares en PHP	1
2	Computadora con WampServer y PHP Designer 2007 instalado	1
3	Memoria USB	1

IV. PROCEDIMIENTO

Realice ordenadamente cada uno de los siguientes ejercicios. Algunos incluyen más de una script PHP junto con alguna página web en puro HTML.

Ejercicio #1: El siguiente ejemplo permite buscar una palabra dentro de un texto ingresado en un área de texto. Puede copiar y pegar en esta área de texto contenido de alguna página web y verificar ingresando en el cuadro de texto superior la palabra a buscar. Al presionar el botón de búsqueda de palabra se marcarán todas las ocurrencias de las palabras encontradas.

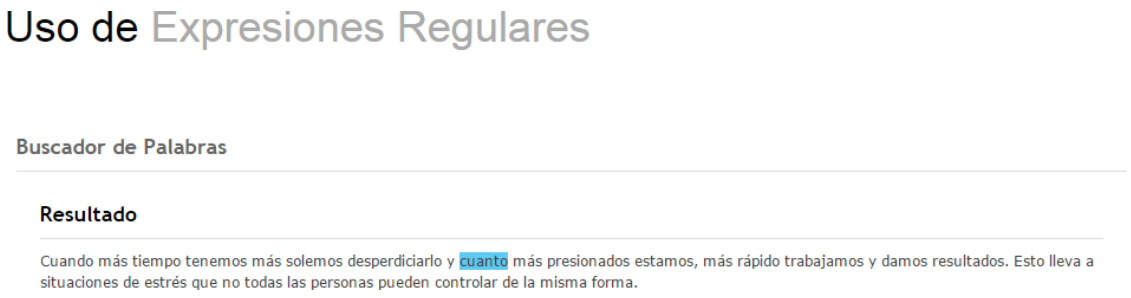
Archivo 1: buscadorpalabras.php

```
<!DOCTYPE html>
<html lang="es">

<head>
    <title>Ejercicio de Expresiones Regulares</title>
    <link rel="stylesheet" href="css/styles.css" />
    <!--[if IE]>
<script src="http://html5shiv.googlecode.com/svn/trunk/html5.js"></script>
<![endif]-->
    <!--[if IE 6]>
<script src="js/belatedPNG.js"></script>
<script>
       DD_belatedPNG.fix('*');
</script>
<![endif]-->
</head>
```

```
<body>
    <div id="bodywrap">
        <section id="pagetop"></section>
        <header id="pageheader">
            <h1>Uso de<span> Expresiones Regulares</span></h1>
        </header>
        <div id="contents">
            <section id="main">
                <div id="leftcontainer">
                    <h2>Buscador de Palabras</h2>
                    <section id="sidebar">
                        <?php
                        if (isset($_POST['Enviar'])) {
                            $text = $_POST['comment'];
                            $palabra = $_POST['palabra'];
                            $text  =  preg_replace("/\b("  .  $palabra  .  ")\b/i",  '<span
style="background:#5fc9f6">\1</span>', $text);
                        ?>
                             <div id="sidebarwrap">
                                 <h2>Resultado</h2>
                                <p><?= $text ?></p>
                            </div>
                        <?php
                        }
                        ?>
                    </section>
                    <div class="clear"></div>
                    <article class="post">
                        <form  action="<?php  echo  $_SERVER['PHP_SELF']  ?>"  method="post"
class="form">
                            <p class="textfield">
                                <label for="palabra">
                                    <small>Palabra a buscar</small>
                                </label>
                                <input   name="palabra"   id="palabra"   value=""   size="22"
tabindex="1" type="text" />
                            </p>
                            <p>
                                <small>Ingrese el texto de prueba para procesarlo con las
                                    <strong>expresiones regulares</strong>
                                </small>
                            </p>
                            <p class="text-area">
                                <textarea  name="comment"  id="comment"  cols="50"  rows="10"
tabindex="4">
                          Sample sentence from KomunitasWeb, regex has become
                          popular in web programming. Now we learn regex.
                          According to wikipedia, Regular expressions
                          (abbreviated as regex or regexp, with plural forms
```

```
                        regexes, regexps, or regexen) are written in a formal
                        language that can be interpreted by a regular
                        expression processor
                    </textarea>
                        </p>
                        <p>
                            <input name="Enviar" id="Enviar" value="1" type="hidden" />
                            <input  name="submit"  id="submit"  tabindex="5"  type="image"
src="images/submit.png" />
                        </p>
                        <div class="clear"></div>
                    </form>
                    <div class="clear"></div>
                </article>
            </div>
        </section>
        <div class="clear"></div>
    </div>
</div>
<footer id="pagefooter">
    <div id="footerwrap">
    </div>
</footer>
</body>

</html>
```

El resultado en el navegador sería el siguiente:



Resultado personal:

Ejercicio #2: El siguiente ejemplo muestra cómo utilizar una expresión regular para determinar si uno o varios archivos que se desean enviar al servidor son archivos de imagen válidos para publicar en sitios web o no. Se utiliza un control de formulario input type=file para adjuntar uno o varios archivos y en una secuencia de comando o guion PHP se procesan los nombres de los archivos enviados para determinar de acuerdo a su extensión si son o no archivos de imagen.

Archivo 1: uploadfile.php

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="utf-8" />
    <!-- Indicar al navegador que la página estará optimizada para distintos dispositivos --
>
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <!--Import Google Icon Font-->
    <link rel="stylesheet" href="http://fonts.googleapis.com/icon?family=Material+Icons" />
    <link rel="stylesheet" href="css/fonts.css" />
    <!--Import materialize.css-->
    <link rel="stylesheet" href="css/materialize.css" />
</head>

<body>
    <section>
        <article>
            <div class="row">
                <h1 class="title-form">Adjuntar un archivo de imagen</h1>
                <?php
                if (isset($_POST['send'])) :
                    //Incluir librería de funciones
                    require_once("comprobarimagen.php");
                    //Verificar si se han enviado uno o varios archivos
                    //valiéndonos de una expresión regular
                    $archivos = array();
                    if (!empty($_FILES['files']['name'][0])) :
                        $list = "<ol class=\"list-files\">\n";

                        foreach ($_FILES['files']['name'] as $i => $archivo) :
                            $archivos[$i] = $archivo;
```

```
                            //Invocar a la función que verificará mediante
                            //expresión regular si el archivo pasado como
                            //argumento es o no es imagen.
                            $list    .=    "<li>\n<a    href=\"#\">"    .    $archivos[$i]    .
comprobarimagen($archivos[$i]) . "</a>\n\t</li>\n";
                        endforeach;
                        $list .= "</ol>\n";
                        echo $list;
                    endif;

                //Obteniendo los datos del formulario

                else :
                ?>
                    <form     action="<?=     $_SERVER['PHP_SELF'];     ?>"     method="POST"
enctype="multipart/form-data" class="col s12">
                        <div class="row col s12">
                            <div class="file-field input-field col s8">
                                <div class="btn">
                                    <span>Adjuntar</span>
                                    <input type="hidden" name="MAX_FILE_SIZE" value="2097152"
/>
                                    <input type="file" name="files[]" multiple="multiple" />
                                </div>
                                <div class="file-path-wrapper">
                                    <input     type="text"     class="file-path     validate"
placeholder="Seleccione sólo archivos de imagen" />
                                </div>
                            </div>
                            <div class="row col s4">
                                <button  type="submit"  class="btn  waves-effect  waves-light"
name="send">Enviar
                                    <i class="material-icons right">send</i>
                                </button>
                            </div>
                        </div>
                    </form>
                <?php
                endif;
                 ?>
            </div>
        </article>
    </section>
</body>
<!-- Import jQuery before materialize.js -->
<script  src="//code.jquery.com/jquery-2.1.1.min.js"></script>
<script src="js/materialize.min.js"></script>

</html>
```

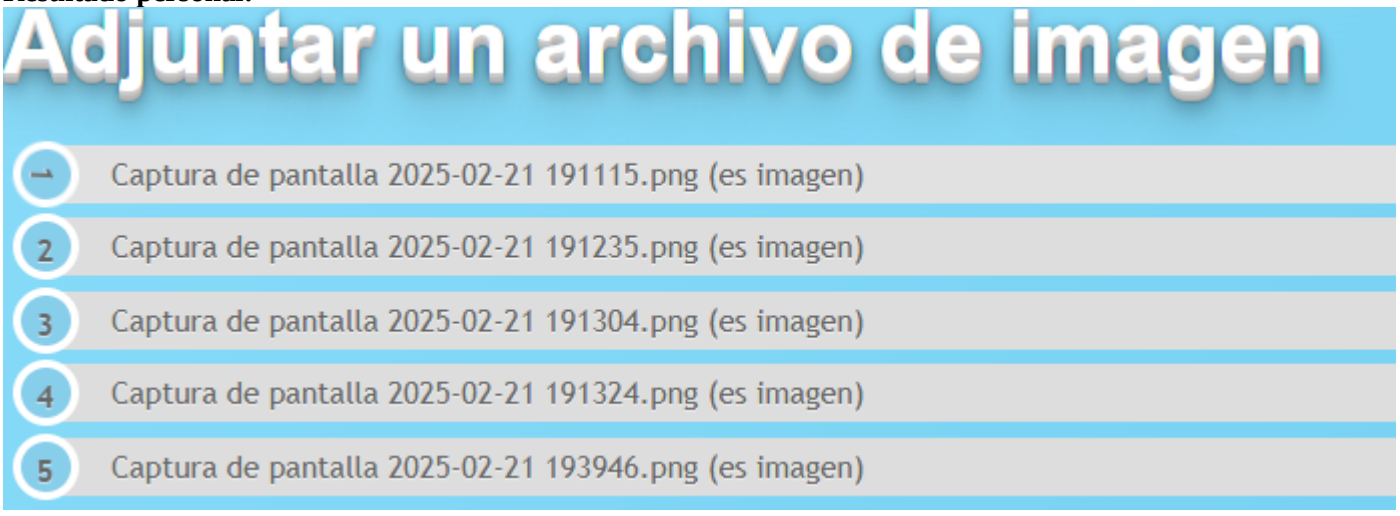
Archivo 2: comprobarimagen.php

```
<?php
  function comprobarimagen($archivo){
      //La expresión regular analiza si el archivo es de
      //una extensión válida para una imagen gif|jpeg|jpg|png
      //utilizando la función preg_match()
      $patron = "/\.(gif|jpe?g|png)$/i";
      $verificado = preg_match($patron, $archivo);
      $esimagen = $verificado == true ? " (es imagen)" : " (no es imagen)";
      return $esimagen;
  }
?>
```

El resto de archivos se le proporcionará con los recursos de la guía de práctica.

Resultado en el navegador:

Resultado personal:



Ejercicio #3: En el presente ejercicio, se realiza una aplicación donde las páginas web del sitio se crean utilizando el paradigma de la Programación Orientada a Objetos (POO) considerando una clase llamada página (page), en donde se identifican como propiedades los elementos característicos de una página web como el contenido en una propiedad $content, el título de la página en una propiedad $title, las palabras clase, típicamente utilizadas en una etiqueta meta, los enlaces del menú principal de una propiedad $buttons, que para este caso contendrá una matriz o arreglo con todos los enlaces que incluyen el título del enlace que será el texto visible para los usuarios que naveguen en la página y el enlace o URL hacia donde apunta la página.

Luego, se definen dentro de esta misma clase los métodos que permitirán generar la página web solicitada, como display(), displayTitle(), displayStyles(), displayScripts(), etc.

Archivo 1: page.class.php

```
<?php
class page {
	//Atributos de la clase
	public $content;
	public $title = "Centro de Estudios de Postgrados - Universidad Don Bosco &copy;";
	public $keywords = "Universidad Don Bosco, UDB, Educaci&oacute;n con estilo salesiano";
```

```
   public $buttons = array(
                        "Inicio" => "home.php",
                        "Carreras" => "carreras.php",
                        "Institucional" => "institucional.php",
                        "Contacto" => "contacto.php"
                    );
   //Operaciones de la clase
   public function __set($name, $value){
$this->name = $value;
   }

   public function display(){
          echo "<!DOCTYPE html>\n";
          echo "<html lang=\"es\">\n<head>\n";
echo "\t<meta charset=\"utf-8\" />\n";
          $this->displayTitle();
          $this->displayKeywords();
          $this->displayStyles("css/home.css");
          $this->displayScripts("js/modernizr.custom.lis.js");
          echo "</head>\n<body>\n";
$this->displayHeader();
$this->displayMenu($this->buttons);
echo $this->content;
$this->displayFooter();
echo "</body>\n</html>";
   }

   public function displayTitle(){
          echo "\t<title>" . $this->title . "</title>\n";
   }

   public function displayKeywords(){
          echo "\t<meta name=\"keywords\" content=\"" . $this->keywords . "\" />\n";
   }

   public function displayStyles($estilos){
          //Patrón de expresión regular para verificar
//si la extensión del archivo es .css
$patron = "%\.{1}(css)$%i";
$styles = "";
if(is_array($estilos)):
  foreach($estilos as $cssfile):
    $styles .= "\t<link rel=\"stylesheet\" href=\"" . $cssfile . "\" />\n";
  endforeach;
else:
  $styles .= "\t<link rel=\"stylesheet\" href=\"" . $estilos . "\" />\n";
endif;
echo $styles;
   }
```

```
  public function displayScripts($scripts){
    //Patrón de expresión regular para verificar
    //que la extensión del archivo es .js
    $patron = "%\.{1}(js)$%i";
    if(is_array($scripts)):
      foreach($scripts as $scriptfile):
        echo "\t<script type=\"text/javascript\" src=\"" . $scriptfile . "\"></script>\n";
      endforeach;
    else:
        if(!empty($scripts)):
          if(preg_match($patron, $scripts)):
            echo "\t<script type=\"text/javascript\" src=\"" . $scripts . "\"></script>\n";
          endif;
        endif;
    endif;
    $scripts = "\t<script type=\"text/javascript\" src=\"" . $scripts . "\"></script>\n";
  }

       public function displayHeader(){
             $header = <<<HEADER
    <!-- page header -->
    <section>
        <article>
        <table width="100%" cellpadding="12" cellspacing="0" border="0">
            <tr bgcolor="black">
                <td align="left">
                    <img src="img/logo.gif" alt="Logo URL" height="70" width="70" />
                </td>
                <td>
                    <h1>Universidad Don Bosco</h1>
                </td>
                <td align="right">
                    <img src="img/logo.gif" alt="Logo URL" height="70" width="70" />
                </td>
            </tr>
        </table>
HEADER;
        echo $header;
       }


   public function displayMenu($buttons){
     $menu  = "<ul id=\"mainmenu\">\n\t";
    //Calcular tamaño
    $width = 100/count($buttons);
    foreach($buttons as $name=>$url){
      $menu .= "<li>\n\t\t";
      $menu  .= $this->displayButton($width, $name, $url, !$this->isURLCurrentPage($url))  .
"\n\t\t";
```

```
      $menu .= "</li>\n";
    }
    $menu .= "</ul>\n";
    echo $menu;
       }

  function isURLCurrentPage($url){
    if(strpos($_SERVER['PHP_SELF'], $url) == false):
      return false;
    else:
      return true;
    endif;

  }

  public function displayButton($width, $name, $url, $active=true){
    $button = "";
    if($active):
      $button .= "<a href=\"" . $url . "\">\n\t\t";
      $button .= "<img src=\"img/url-icon.png\" alt=\"" . $name . "\" />\n\t";
      $button .= "</a>\n\t";
      $button .= "<a href=\"" . $url . "\">\n\t\t";
      $button .= "<span class=\"menu\">" . $name . "</span>\n\t";
      $button .= "</a>\n";
    else:
      $button .= "<img src=\"img/url-icon.png\" alt=\"" . $name . "\" />\n\t";
      $button .= "<span class=\"menu\">" . $name . "</span>\n";
    endif;
    return $button;
  }

  public function displayFooter(){
    $footer = <<<FOOT
        <!-- Pie de la página -->
        <table id="footer">
          <tr>
            <td>
              <p class="foot">
                <a href="http://www.udb.edu.sv" target="_blank">Universidad Don Bosco</a>
              </p>
              <p class="foot">Centro de Estudios de Postgrados.</p>
            </td>
          </tr>
        </table>
      </article>
    </section>
FOOT;
    echo $footer;
  }
```

```
}
?>
```

Archivo 2: home.php

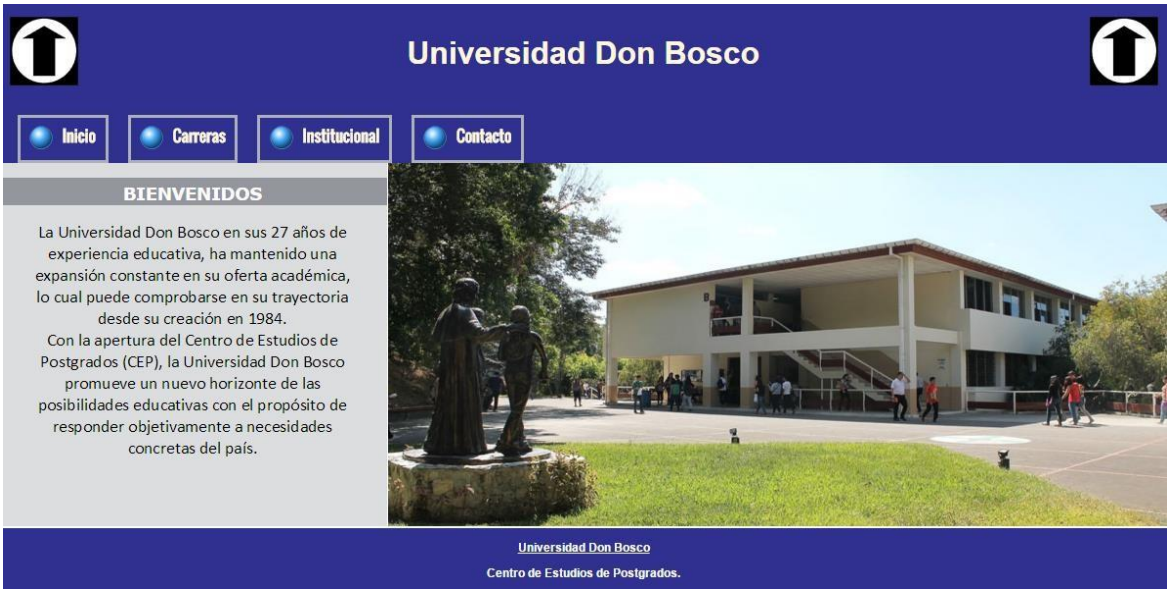
```
<?php
   spl_autoload_register(function($class_name){
      require("class/" . $class_name . ".class.php");
  });

  //Creando el objeto página
  $homepage = new page();

  $homepage->content = <<<PAGE
        <!-- page content -->
        <div id="topcontent">
           <div id="textbox">
              <div id="title">
                 <h2>BIENVENIDOS</h2>
              </div>
              <div id="paragraph">
                 <p>
                    La Universidad Don Bosco en sus 27 años de experiencia educativa,
                    ha mantenido una expansión constante en su oferta académica,
                    lo cual puede comprobarse en su trayectoria desde su creación en
                    1984.<br />
                    Con la apertura del Centro de Estudios de Postgrados (CEP), la Universidad
                    Don Bosco promueve un nuevo horizonte de las posibilidades educativas
                    con el propósito de responder objetivamente a necesidades concretas
                    del país.
                 </p>
              </div>
           </div>
           <div id="picture">
              <img  src="img/plaza-de-las-banderas.jpg"  alt="Imagen  de  portada"  width="800"
height="370" longdesc="Imagen de portada" />
           </div>
        </div>
PAGE;
  echo $homepage->display();
?>
```

Archivo 3: contacto.php

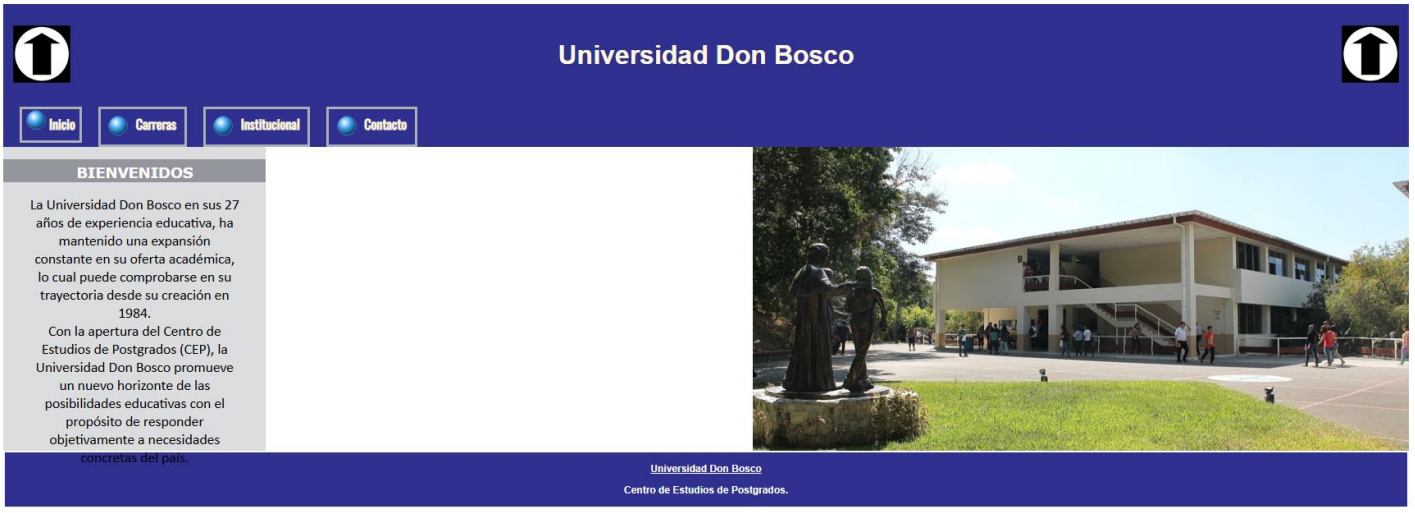
```
<?php
   spl_autoload_register(function($class_name){
      require("class/" . $class_name . ".class.php");
   });

   //Creando el objeto página
```

```
   $contactpage = new page();
   $contactpage->content = <<<PAGE
      <!-- contact content -->
         <div id="topcontent">
            <div id="textbox">
               <div id="title">
                  <h2>CONTACTO</h2>
               </div>
               <div id="paragraph">
                  <h4>Números de atención:</h4>
                  <p>
                     Universidad Don Bosco.<br />
                     Tel. (503)2251-8200.
                  </p>
                  <p>
                     Administración Académica.<br />
                     Tel. (503)2251-8200 ext. 1710.
                  </p>
                  <p>
                     Administración Financiera.<br />
                     Tel. (503)2251-8200 ext. 1700.
                  </p>
                  <p>
                     Nuevo Ingreso.<br />
                     Tel. (503)2527-2314.
                  </p>
               </div>
            </div>
            <div id="picture">
               <img   src="img/entradapostgrados.jpg"   alt="Imagen   de   portada"   width="800"
height="370" longdesc="Imagen de portada" />
            </div>
         </div>
PAGE;
   echo $contactpage->display();
?>
```

El resultado visible en cualquier navegador actual, debería ser el siguiente:



Resultado personal:



Ejercicio #4: El siguiente ejemplo nos muestra cómo se puede implementar el cálculo de la distancia entre dos puntos, dadas dos coordenadas ingresadas por el usuario.

Se ha utilizado una clase que no posee propiedades y que se definirán dinámicamente, haciendo uso de la sobrecarga implementada con los métodos mágicos __set() y __get().

Archivo 1: coordenadas.class.php

```
<?php
   class coordenadas {
      private $coords = array('x' => 0, 'y' => 0);
      //Métodos especiales __get() y __set()
         function __get($property) {
         if(array_key_exists($property, $this->coords)) {
            return $this->coords[$property];
         }
               else {
            print "Error: Sólo se aceptan coordenadas x y y.<br />\n";
         }
      }
      function __set($property, $value) {
         if(array_key_exists($property, $this->coords)) {
            $this->coords[$property] = $value;
         }
               else {
            print "Error: No se puede escribir otra coordenada más que x y y.<br />\n";
         }
      }
   }
?>
```

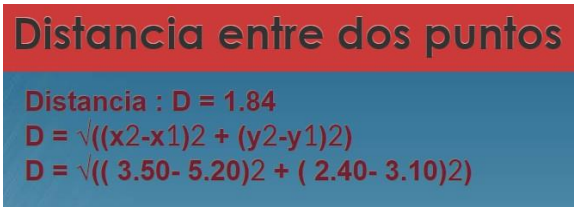
Archivo 2: distanciadospuntos.php

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="utf-8" />
    <title>La distancia entre dos puntos</title>
    <link rel="stylesheet" href="css/slick.css" />
</head>
<body>
<header id="demo">
    <h1 class="demo1">Distancia entre dos puntos</h1>
</header>
<section id="slick">
<?php
   if(isset($_POST['submit'])){
      //Capturando los datos de formulario
      $x1 = is_numeric($_POST['coordx1']) ? $_POST['coordx1'] : "Error";
      $x2 = is_numeric($_POST['coordx2']) ? $_POST['coordx2'] : "Error";
      $y1 = is_numeric($_POST['coordy1']) ? $_POST['coordy1'] : "Error";
      $y2 = is_numeric($_POST['coordy2']) ? $_POST['coordy2'] : "Error";
      if($x1 == "Error" || $x2 == "Error" || $y1 == "Error" || $y2 == "Error"){
         die("<h3 style=\"color:red;\">Los valores de x1, x2, y1 y y4 deben ser
numéricos</h3>");
      }

     //Utilizando autocarga de clases para invocar la clase
     spl_autoload_register(function($class){
         require_once "class/" . $class . ".class.php";
      });

     //Creando las coordenadas
      $coord1 = new coordenadas();
      $coord2 = new coordenadas();
      //Definiendo las coordenadas del primer punto
      $coord1->x = $x1;
      $coord1->y = $y1;
      //Definiendo las coordenadas del segundo punto
      $coord2->x = $x2;
      $coord2->y = $y2;
      //Obteniendo la distancia entre dos puntos
      $difx = pow($coord2->x - $coord1->x, 2);
      $dify = pow($coord2->y - $coord1->y, 2);
      $dist = sqrt($difx + $dify);
      printf("<p class=\"resp\">Distancia : D = " . number_format($dist, 2, '.', ',') .
"</p>\n");
      printf("<p    class=\"resp\">D    =    √((x<sub>2</sub>-x<sub>1</sub>)<sup>2</sup>    +
(y<sub>2</sub>-y<sub>1</sub>)<sup>2</sup>)</p>\n");
      printf("<p    class=\"resp\">D    =    √((%5.2lf-%5.2lf)<sup>2</sup>    +    (%5.2lf-
%5.2lf)<sup>2</sup>)</p>\n", $coord2->x, $coord1->x, $coord2->y, $coord1->y);
   }
```

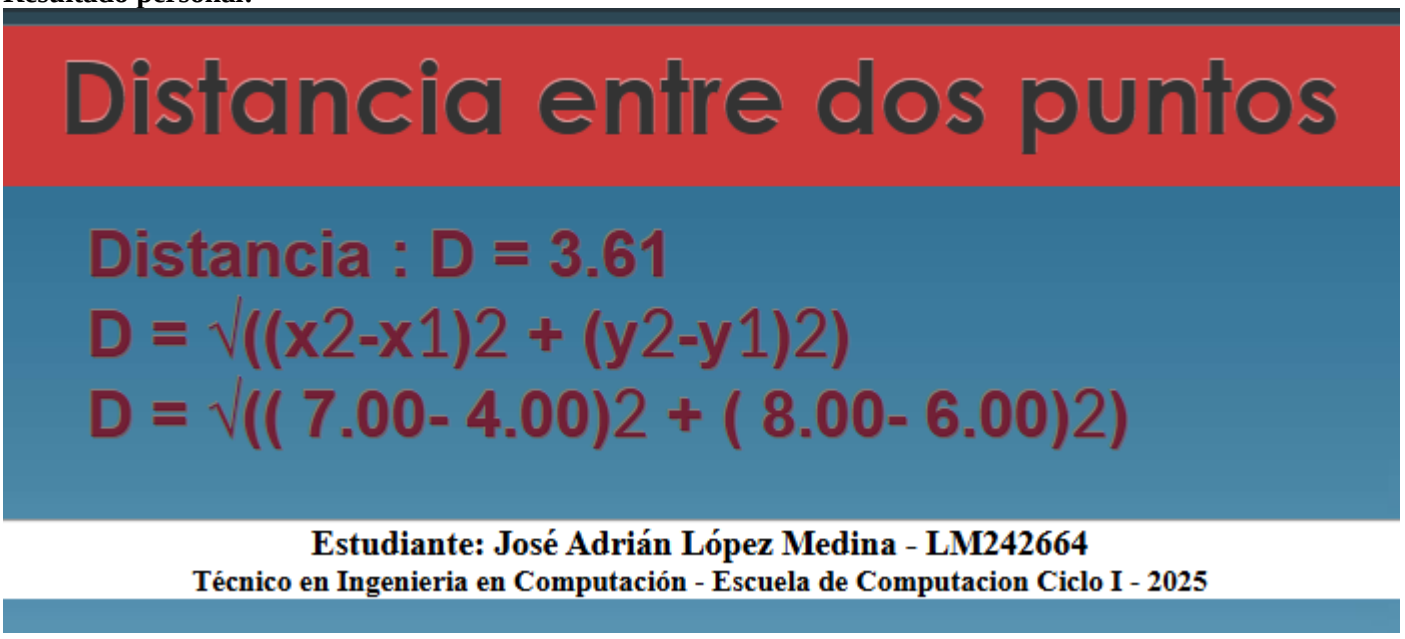
```
   else{
?>
<div class="contact-form">
   <!-- Título -->
   <div class="title">Cálculo de la distancia entre dos puntos</div>
   <!-- Texto indicativo -->
   <p class="intro">Ingrese las coordenadas</p>
   <!-- Área de formulario -->
   <div class="contact-form">
   <!-- Formulario -->
   <div class="w-100">
      <!-- Campos de formulario -->
      <form name="frmrectangulo" id="frmrectangulo" action="<?php echo $_SERVER['PHP_SELF']
?>" method="POST">
      <!-- <form name="frmrectangulo" id="frmrectangulo" action="javascript:void(0);"> -->
         <!-- Coordenada 1 (x1,y1) -->
         <label>Coordenada 1 (x1,y1): </label>
         <div class="field">
            <input type="number" name="coordx1" id="coordx1" min="0" max="1000" step=".1"
placeholder="(x1)" required />
            <span class="entypo-base icon"></span>
            <span class="slick-tip left">Ingrese la coordenada x1:</span>
         </div>
         <div class="field">
            <input type="number" name="coordy1" id="coordy2" min="0" max="1000" step=".1"
placeholder="(y1)" required />
            <span class="entypo-base icon"></span>
            <span class="slick-tip left">Ingrese la coordenada y1:</span>
         </div>
         <!-- Coordenada 2 (x2,y2) -->
         <label>Coordenada 2 (x2,y2): </label>
         <div class="field">
            <input type="number" name="coordx2" id="coordx2" min="0" max="1000" step=".1"
placeholder="(x2)" required />
            <span class="entypo-base icon"></span>
            <span class="slick-tip left">Ingrese la coordenada x2:</span>
         </div>
         <div class="field">
            <input type="number" name="coordy2" id="coordy2" min="0" max="1000" step=".1"
placeholder="(y2)" required />
            <span class="entypo-base icon"></span>
            <span class="slick-tip left">Ingrese la coordenada y2:</span>
         </div>
         <!-- Botones para hacer los cálculos -->
         <input type="submit" value="Calcular" class="send" name="submit" id="perimetro" />
         <input type="reset" value="Restablecer" class="send" name="reset" id="area" />
      </form>
   </div>
</div>
<?php
```

```
    }
?>
</section>
</body>
</html>
```

Al visualizarlo en el navegador de su preferencia puede ingresar los datos de las coordenadas y luego verificar que el cálculo de la distancia entre los dos puntos es correcta:



Resultado personal:



Ejercicio #5: Este ejemplo muestra cómo implementar una clase para crear campos de formulario. La clase campoformulario se crea como una clase abstracta con uno de sus métodos abstractos, el método que crea un campo de formulario específico. Todas las clases derivadas de esta tendrán que implementar este método abstracto para poder crear el tipo de campo de formulario adecuado.

Archivo 1: campotexto.class.php

```
<?php
  //Clase abstracta para algún tipo de campo de formulario
  abstract class campoformulario {
        //Propiedades de la clase abstracta
        protected $idcampo;
        protected $etiqueta;
        protected $capaayuda;

        //Constructor de la clase
        function __construct($id, $etiq, $ayuda){
              $this->idcampo = $id;
              $this->etiqueta = $etiq;
              $this->capaayuda = $ayuda;
       }

       //Método abstracto que será implementado
       //por alguna de las clases hijas o derivadas
       abstract function pinta_campo();

       protected function poner_eventos_js(){
              $cmd_js  = 'document.getElementById("c_" + this.name).style.visibility';
              $cmd2_js = 'document.getElementById("c_" + this.name).style.display';
```

```
            return      "      onfocus='$cmd_js=\"visible\";      $cmd2_js=\"inline-block\";'
onblur='$cmd_js=\"hidden\"; $cmd2_js=\"none\"'";
      }

      protected function poner_capa_ayuda(){
            //El identificador de la capa es
            //'c_' + nombre de la capa
            $s  = "background: Lavender; ";
            $s .= "border: 1px solid #4D274F; ";
            $s .= "color: #7B0F86; ";
            $s .= "font: Bold 0.85em \"Open Sans\",Arial,Helvetica,sans-serif; ";
            $s .= "padding: 4px 6px; ";
            $s .= "position:relative; ";
            $s .= "text-shadow: 0 -1px 0 #BFADC0,\n 0 -2px 0 #B099B2,\n 0 -3px 0 #9C889F,\n
0 0 2px #816883,\n 0 -2px 3px #715973,\n 3px -3px 15px #000; ";
            $s .= "display: none; ";
            $s .= "visibility:hidden;";
            return "<span id='c_$this->idcampo' style='$s'>$this->capaayuda</span>\n";
      }
  }

  //Clase para un campo de formulario de tipo text
  class campotexto extends campoformulario {
      //Definiendo las propiedades
      private $placeholder;
      private $maxcar;
      //Creando un nuevo constructor
      function __construct($id, $etiq, $ayuda, $placehold, $maxcar){
            parent::__construct($id, $etiq, $ayuda);
            $this->placeholder = $placehold;
            $this->maxcar = $maxcar;
      }
      //Implementando el método abstracto pinta_campo
      function pinta_campo(){
            $stag  = "";
            $stag .= "\t<div class=\"row\">\n";
            $stag .= "\t<section class=\"col col-6\">\n";
            $stag .= "\t\t<label for=\"$this->idcampo\" class=\"input\">\n";
         $stag .= "\t\t<i class=\"icon-prepend icon-user\"></i>\n";
             $stag .= "\t\t<input type=\"text\" name=\"" . $this->idcampo . "\" id=\"". $this-
>idcampo . "\" placeholder=\"" . $this->placeholder . "\" maxlength=\"" . $this->maxcar . "\""
. $this->poner_eventos_js() . " />\n";
            $stag .= $this->poner_capa_ayuda();
            $stag .= "</label>";
            $stag .= "\t</section>\n";
            $stag .= "\t</div>\n";
            echo $stag;
      }
  }
```

```
  //Clase para un campo de formulario tipo textarea
  class campotextarea extends campoformulario {
        //Propiedades
     private $placeholder;
        private $lineas;
        private $cols;
        //Definiendo un constructor para esta clase
        function __construct($id, $etiq, $lineas, $cols, $ayuda, $placehold){
              parent::__construct($id, $etiq, $ayuda);
              $this->placeholder = $placehold;
              $this->lineas = $lineas;
              $this->cols = $cols;
        }

        function pinta_campo() {
           $tag  = "\t<section>\n";
           $tag .= "\t\t<label for=\"$this->idcampo\" class=\"textarea\">$this->etiqueta\n";
               $tag .= "\t\t<textarea name=\"" . $this->idcampo . "\" id=\"" . $this->idcampo
 . "\" rows=\"" . $this->lineas . "\" cols=\"" . $this->cols . "\" " . $this->poner_eventos_js()
. ">";
              $tag .= $this->placeholder;
              $tag .= "</textarea>\n";
              $tag .= $this->poner_capa_ayuda();
              $tag .= "</label>";
              $tag .= "\t</section>\n";
              echo $tag;
        }
  }

  class campocheckbox extends campoformulario {
     //Definiendo las propiedades
     private $options = array();
     private $listed;
     //Creando el constructor de la clase campocheckbox
     function __construct($id, $name, $etiq, $ayuda, $options, $enlistados=false){
        parent::__construct($id, $name, $etiq, $ayuda);
        $this->options = $options;
        $this->etiqueta = $etiq;
        $this->enlistados = $enlistados;
     }
     //Implementando el método abstracto crearcampo
     function pinta_campo(){
        $pos = 0;  //Indica la posición en el arreglo de opciones del checkbox
        //echo "<label for=\"$this->idcampo" . $pos . "\">$this->etiqueta</label><br />\n";
        $stag  = "";
        $stag .= "\t<section>\n";
        $stag .= "\t\t<label>" . $this->etiqueta . "</label>\n";
        //Recorriendo el array con las opciones del checkbox
        foreach($this->options['opciones'] as $key => $value){
```

```
            $stag .= "\t\t<label class=\"checkbox\">\n";
            $stag .= "\t\t<input type=\"checkbox\" value=\"$value\" name=\"$this->idcampo" .
$pos . "\" id=\"$this->idcampo" . $pos . "\" ";
            $stag .= $this->options['estados'][$pos] == true ? "checked=\"checked\" " : "";
            $stag .= " /><i class=\"fa-check\"></i>$key</label>\n";
            $stag .= $this->enlistados == true ? "<br />\n" : "";
            $pos++;
         }
         $stag .= "\t</section>\n";
         echo $stag;
         echo $this->enlistados == true ? "<br />\n" : "";
      }
   }

   class camposelect extends campoformulario{
      //Propiedades de la clase
         private $size;
         private $multiple;
         private $options = array();
         //Método constructor
         function __construct($id, $etiq, $ayuda, $size, $multiple, $options){
         parent::__construct($id, $etiq, $ayuda);
         $this->size = $size;
         $this->multiple = $multiple;
         $this->options = $options;
         }
      function pinta_campo(){
        $mult = ($this->multiple != "") ? " " . $this->multiple : "";
         $seltag  = "<div class=\"row\">\n";
         $seltag .= "\t<label for=\"" . $this->idcampo . "\" class=\"label col col-4\">$this-
>etiqueta</label><br />\n";
         $seltag .= "\t<section class=\"col col-5\">";
         $seltag .= "\t\t<label class=\"select\">";
               $seltag .= "\t\t<select name=\"" . $this->idcampo . "\" id=\"" . $this->idcampo
. "\" size=\"" . $this->size . "\"" . $this->poner_eventos_js() . ">\n";
         foreach($this->options as $key => $value){
            $seltag .= "\t\t\t<option value=\"$key\">$value</option>\n";
         }
         $seltag .= "\t\t</select>\n";
         $seltag .= $this->poner_capa_ayuda();
         $seltag .= "\t\t<i></i>\n";
         $seltag .= "\t\t</label>\n";
               $seltag .= "</div>\n";
         echo $seltag;
      }
   }

   class camposubmit extends campoformulario{
      //Propiedades adicionales
         private $value;
```

```
        //Constructor de la clase camposubmit
        function __construct($id, $etiq, $value, $ayuda){
         parent::__construct($id, $etiq, $ayuda);
         $this->value = $value;
     }

     function pinta_campo(){
        $subtag  = "<input type=\"submit\" name=\"" . $this->idcampo . "\" id=\"" . $this-
>idcampo . "\" value=\"" . $this->value . "\" class=\"button\"";
              $subtag .= $this->poner_eventos_js() . " />\n";
              $subtag .= $this->poner_capa_ayuda();
              $subtag .= "<br />\n";
        echo $subtag;
     }
  }
?>
```

Archivo 2: abstractforms.php

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="utf-8" />
    <title>Formulario dinámico</title>
    <!-- <link rel="stylesheet" href="css/fonts.css" /> -->
    <link rel="stylesheet" href="css/demo.css" />
    <!-- Estilos tomados desde de los ejemplos de http://voky.com.ua/ -->
    <link rel="stylesheet" href="css/sky-forms.css" />
    <link rel="stylesheet" href="css/sky-forms-purple.css" />
    <!--[if lt IE 9]>
      <link rel="stylesheet" href="css/sky-forms-ie8.css">
    <![endif]-->

    <!--[if lt IE 10]>
      <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js"></script>
      <script src="js/jquery.placeholder.min.js"></script>
    <![endif]-->
    <!--[if lt IE 9]>
      <script src="http://html5shim.googlecode.com/svn/trunk/html5.js"></script>
      <script src="js/sky-forms-ie8.js"></script>
    <![endif]-->
</head>
<body class="bg-purple">
<div class="body">
    <h1>Generador de formulario</h1>
<?php
  spl_autoload_register(function($classname){
    include_once "class/" . $classname . ".class.php";
  });
```

```
  echo "<form name=\"form\" action=\"mostrarform.php\" method=\"POST\" class=\"sky-form\"
onsubmit=\"return true\">\n\t";
  echo "\t<header>Formulario dinámico</header>\n";
  echo "\t<fieldset>\n";
  //En el array $campos se crean todos los campos que van a conformar el formulario
  //utilizando todas las clases que se han creado para cada uno de los controles
  //de formulario considerados para este ejemplo
  $campos = array(
                new campotexto        ("nombre", "Nombre: ", "El nombre completo", "Nombre
completo", 40),
                new campotexto         ("apellido", "Apellido: ", "El apellido completo",
"Apellido completo", 30),
                new  campocheckbox       ("deportes",  "Deportes",  "Deportes:",  "Deportes
favoritos",
                                     array(
                                         "opciones" => array(
                                             "Fútbol"      => "Fútbol",
                                             "Basketball"  => "Basketball",
                                             "Volleyball"  => "Volleyball",
                                             "Beisball"    => "Beisball",
                                             "Tenis"       => "Tenis"
                                         ),
                                         "estados"  => array(
                                             true,
                                             false,
                                             false,
                                             true,
                                             false
                                         )
                                     ),
                                    false
                                    ),
                new campotextarea    ("observaciones", "Observaciones: ", 6, 50, "Háganos sus
comentarios", "Envíenos sus comentarios"),
                new camposelect         ("nacionalidad", "Nacionalidad: ", "Seleccione su
acionalidad", 1, '',
                                     array(
                                        "El Salvador" => "El Salvador",
                                        "Guatemala" => "Guatemala",
                                        "Honduras" => "Honduras",
                                        "Costa Rica" => "Costa Rica",
                                        "Nicaragua" => "Nicaragua",
                                        "Panamá" => "Panamá"
                                     )
                                    ),
                new camposubmit      ("enviar", "", "Enviar", "Enviar el formulario")
            );
  foreach($campos as $campo){
      $campo->pinta_campo();
```

```
  }
  echo "</fieldset>\n";
  echo "</form>\n";
?>
</div>
</body>
</html>
```

Archivo 3: mostrarform.php

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="utf-8" />
    <title>Datos del formulario</title>
    <link rel="stylesheet" href="http://fonts.googleapis.com/css?family=Bitter" />
    <link rel="stylesheet" href="css/liststyle.css" />
</head>
<body class="bg-cyan">
<section>
<div class="form-style">
<h1>Datos del formulario</h1>
<div class="numberlist">
<?php
   if(isset($_POST['enviar'])){
     $lista = "<ol>\n";
      foreach($_POST as $name => $value){
         if(gettype($value) == "array" && $name != "enviar"){
            foreach($value as $dato){
               $lista .= "\t<li><a href=\"javascript:void(0);\">$name: " . $dato .
"</a></li>\n";
            }
         }
         if($name != "enviar"){
            $lista .= "\t<li><a href=\"javascript:void(0);\">$name: " . $value .
"</a></li>\n";
         }
      }
      $lista .= "</ol>\n";
      echo $lista;
   }
?>
</div>
</div>
</article>
</section>
</body>
</html>
```

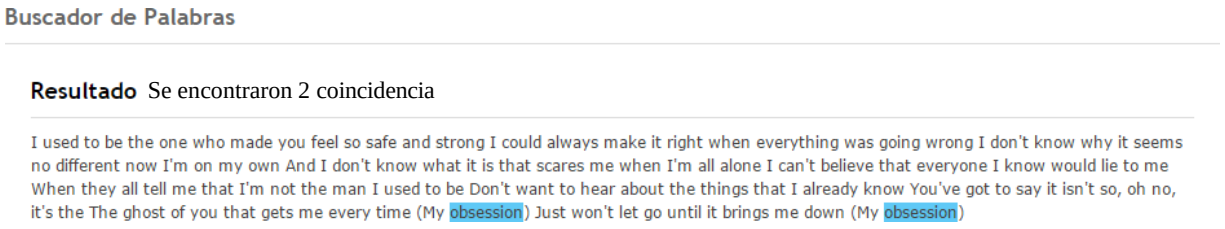
Al visualizar en el navegador el script abstractforms.php podremos observar el formulario creado dinámicamente haciendo uso de la clase campotexto.class.php, que en realidad sirve para definir todos campos de formulario que se muestran en dicho formulario.

Resultado personal:

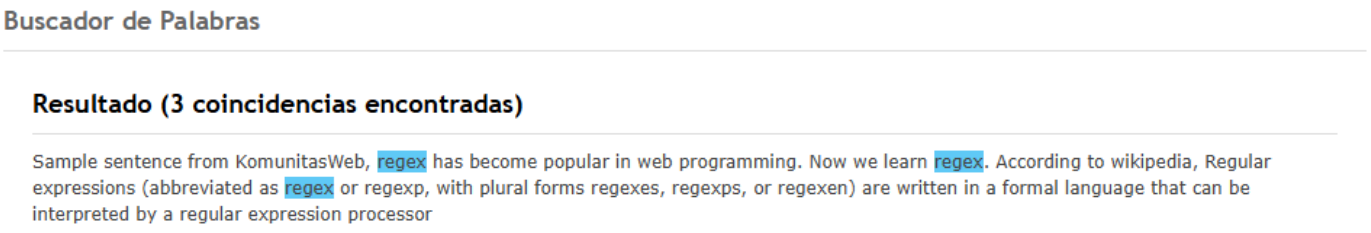
Estudiante: José Adrián López Medina - LM242664
Técnico en Ingenieria en Computación - Escuela de Computacion Ciclo I - 2025

V. ANÁLISIS DE RESULTADOS

1. En el ejemplo 1 del procedimiento de esta guía de la práctica, modifique el código para que justo a la par de Resultado una vez que se envíe el formulario con la palabra que se desea buscar en el texto ingresado en el campo textarea, se indique el número de coincidencias que fueron encontradas de la palabra buscada en el texto ingresado.

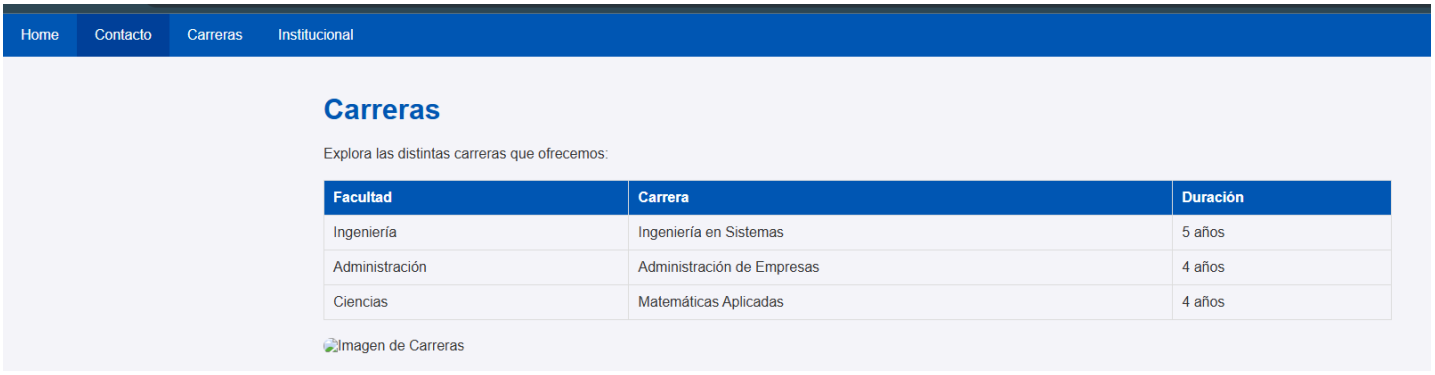


Resultado personal:



2. Implemente en el Ejercicio 3 del procedimiento las opciones de menú Carreras e Institucional que deberán llamar a los scripts carreras.php e institucional.php, respectivamente. Utilice la lógica orientada a objetos que se ha utilizado en los scripts home.php y contacto.php. Modifique el diseño de las páginas Carreras e Institucional, de modo que muestre otra distribución de elementos de página. Puede utilizar párrafos, imágenes, tablas y cualquier otro elemento HTML5. Sea creativo.

Resultado personal:



VII. BIBLIOGRAFIA

- Gil Rubio / Francisco Javier, Villaverde / Santiago Alonso. Creación de sitios web con PHP5. 1a.

edición en español. Editorial McGraw Hill. Madrid, España. 2006.

- Gutierrez, Abraham / Bravo, Ginés. PHP 5 a través de ejemplos. 1ra Edición. Editorial Alfaomega. Junio 2005. México.
- John Coggeshall. LA BIBLIA DE PHP 5. 1ra. Edición. Editorial Anaya Multimedia. Madrid, España 2005.
- Welling, Luke / Thomson, Laura. Desarrollo web con PHP y MySQL. Traducción de la 3ra Edición en inglés. Editorial Anaya Multimedia. 2005. Madrid, España.
- Ellie Quigley / Marko Gargenta. PHP y MySQL Práctico para Diseñadores y Programadores web. Primera edición. Editorial Anaya Multimedia. Madrid, España 2007.