Usuario:ManuelRomero/php/NewPHP/B2T1/Sintaxis

De WikiEducator
< Usuario:ManuelRomero

LENGUAJE PHP: **EL LENGUAJE EN GENERAL**

¡El servidor te responde

PHP Un lenguaje de script al lado del servidor

Sintaxis del lenguaje | Ejercicios | Práctica | Volver

TEMA 3: LENGUAJE PHP

Contenido

Show presentation

Introducción a PHP

Sección de introducción a PHP

- En esta sección veremos qué es el lenguaje php y para qué sirve

Qué es php

- ***PHP*** (acrónimo de PHP: Hypertext Preprocessor)

De php podríamos decir

- Es un lenguaje de código abierto
- Muy popular (Podríamos pensar en un estándar?), una gran comunidad de soporte en internet que aporta, colabora y soluciona dudas

- Especialmente adecuado para desarrollo web (Se puede usar como lenguaje de escritorio, pero no es su propósito).
- Actualmente está la versión 7.2, aún en beta (28-Septiembre-2017), siendo la 7.1.10 la última estable
- Ver las nuevas características de php

Caracterísitcas de php 7

http://php.net/manual/es/migration70.php

- Se usa mucho la versión 5.9

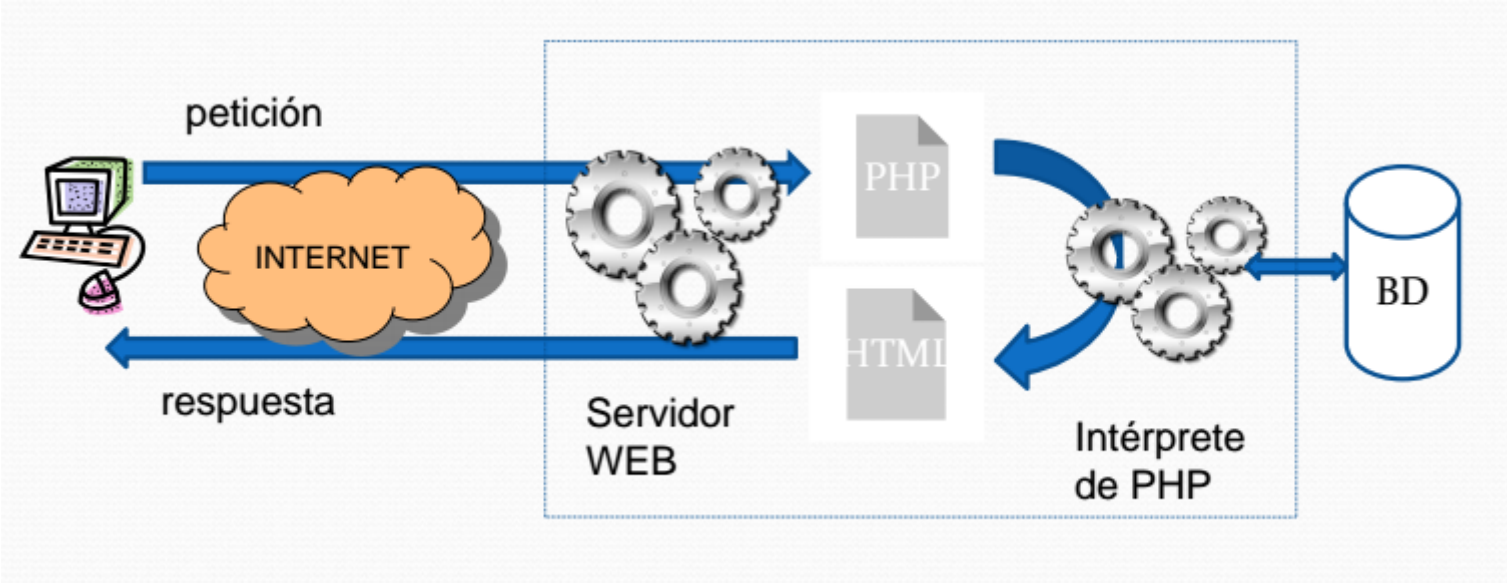
características de php

Tip:

- ***En las aplicaciones de desarrollo web***

1. Se ejecuta en el ***servidor web***
2. Es ***incrustado*** en HTML.
3. El cliente solo ve ***el resultado de la ejecución nunca el código***

Obtención del lenguaje de marcas para mostrar en el cliente



Puntos clave

El documento PHP, una vez interpretado correctamente en el servidor, produce una página HTML que será enviada al cliente.

El servidor en acción

Puntos clave

El código PHP está embebido en documentos HTML,

Esto permite introducir dinamismo fácilmente a un sitio web.

El servidor en acción

Puntos clave

El intérprete PHP ignora el texto del fichero HTML
Hasta que encuentra una etiqueta de inicio del bloque de código PHP embebido.

- Entonces interpreta las instrucciones hasta el final de etiqueta generando la salida correspondiente que se añade al documento html que se entrega al cliente (en caso de que las instrucciones lo generen)

Restricciones del servidor con php

- Como PHP se ejecuta del lado del servidor sólo puede tener acceso a los datos del propio servidor.
  - No puede acceder a los recursos del cliente
  - No puede saber qué hora es en el cliente
  - No puede acceder a los archivos del cliente
    - Salvo la excepción de las Cookies

Configuraciones

- PHP se puede instalar como un servicio independiente (PHP-FPM (FastCGI Process Manager)) o como un módulo de apache php5-mod. Realmente es más eficaz por temas de memoria que corra como un servicio independiente, siendo éste, un tema más de administración que de desarrollo.

Configuración

Por comodidad (todo centrado en el servicio de apache2) en este módulo lo hemos instalado como un módulo de apache, pero en producción se suele instalar como servicio independiente (en este caso se ha de rebotar el servicio de apache o ngix (según servidor) independientemente del servicio de php según los ficheros de configuración que se modifiquen en un momento dado. En cualquier caso, al instalar php, bien como módulo de apache o como servicio independiente, se crea un fichero de cofiguración dónde tenemos las diferentes directivas que podremos modificar (recordad xdebug que modificamos en php.ini).

Directivas de PHP.ini

```
http://www.php.net/manual/es/ini.list.php
```

Funciones que quedaron obsoletas en PHP 5.3.x

```
http://php.net/manual/es/migration53.deprecated.php
```

Características obsoletas en PHP 7.0.x

```
http://php.net/manual/es/migration70.deprecated.php
http://php.net/manual/es/migration70.incompatible.php
```

Cómo escribir PHP

- Dentro de páginas html

```
<?php
   instrucciones
?>
```

- Nosotros siempre usaremos este estilo para escribir código
- Lo podemos embeber en código html .... o no.

Otros modos menos usados

Estilo asp

```
<%
   instrucciones
%>
```

- Para ello hemos de tener habilitado la etiqueta de php.ini

```
asp_tags 1
```

Estilo corto

```
<?
   instrucciones
?>
```

- Para ello hemos de tener habilitado la etiqueta de php.ini

```
short_open_tag 1
```

Sintaxis para editores HTML

```
<SCRIPT LANGAGE="PHP">
   instrucciones
</SCRIPT>
```

Tip:

- En la versión 7.0.0 de php se elimina las etiquetas siguientes:

1. ***<%, %> (formato ASP)***
2. ***#<%= (formato corto)***
3. ***y la etiqueta de script <script language=""***

- Guardamos el fichero con extensión .php
  - Así sabemos que el interprete php tiene que ejecutar código

Probando primer programa

Tip: existen una función llamada ***phpinfo()***.

- Vamos a probarla y ver la información que genera

- Haz un programa que en php que ejecute la función ***phpinfo()***

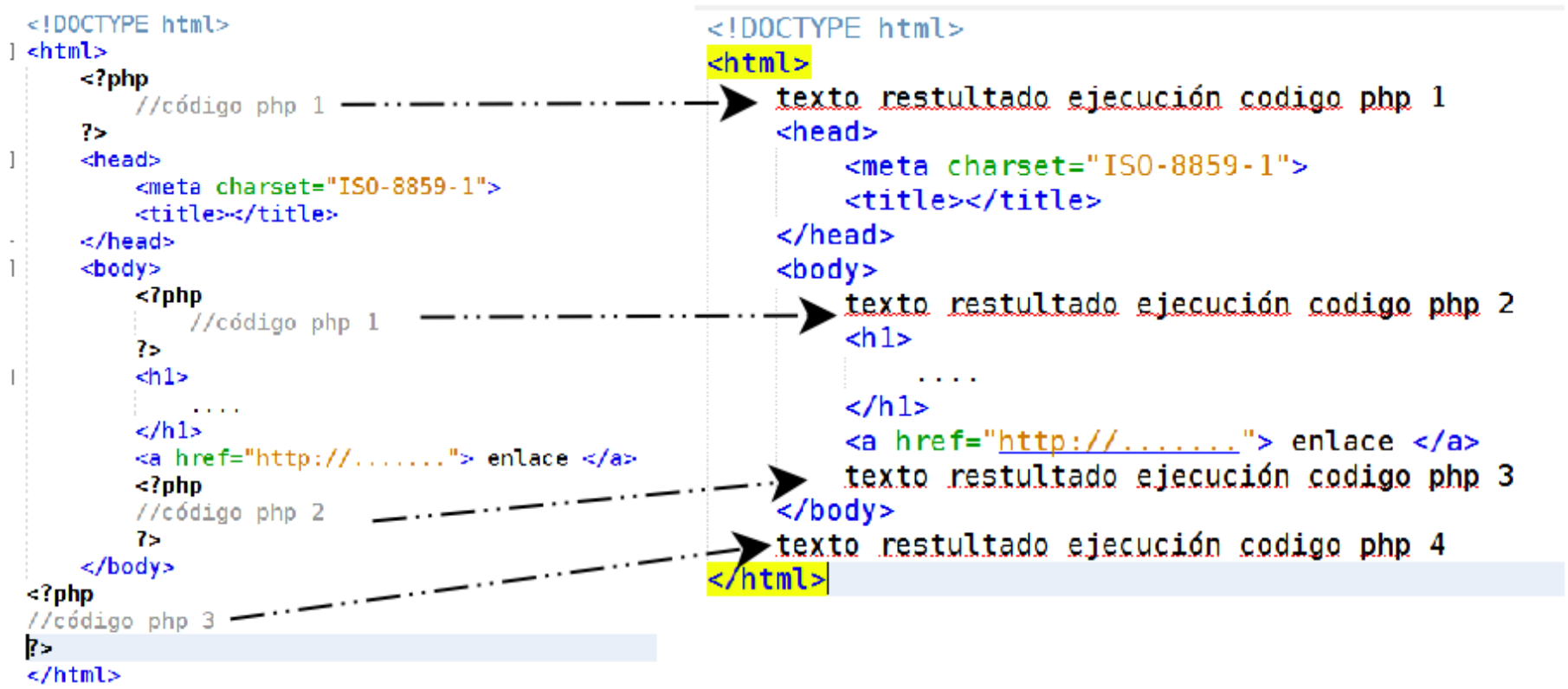
Primer programa [▼]

Dónde poner el código embebido

Pregunta

Dónde escribir código php

- Donde queramos que se ejecute algo



- Se ejecuta como si fuera un solo programa
- A la hora de incluir php podemos se usa mucho la filosofía Modelo Vista /Controlador que veremos en otro tema

Resumen

- Trata de separar el código según su cometido, de forma muy resumida
- Hay acciones que realizan cálculos y obtienen resultados (Controlador
- Hay acciones que lo que hacen es visualziar valores (Vista)
- Hay acciones que se encargan de interactuar con la base de datos (Modelo)

Empecemos aplicar el modelo vista controlador

- Podemos llevar este criterio desde el principio siguiente el esquema siguiente

esqueleto MVC [▼]

Programa para ver si un número es o no par

No te preocupes si no entiendes el código, lo iremos viendo

```
<?php

//-----------CONTROLADOR-----------------------------
//Instrucciones que realizan cálculos
//Guardo el resultado que quiero mostrar en variables
$a = rand(1, 100); //Genero un número aleatorio
$fecha = date("d m Y", time()); //Obtenemos la fecha
$title = "Hoy, $fecha, veremos si '$a' es par o impar "; //Creo un título
if ($a % 2 == 0) //Si el número es par
    $msj = "El número $a es par";
else
    $msj = "El número $a es impar";
?>

<!-- Ahora metemos el código php cuando lo necesitemos-->
<!------------------- VISTA (solo la parte php)----------------------->
<!DOCTYPE html>
<html>
```

```
    <head>
        <meta charset="UTF-8">
        <title>probando php</title>
    </head>
    <body>
        <h1> <?php echo $title ?></h1>
        <hr  />
        <h3> <?php echo $msj ?></h3>

    </body>
</html>
```

Escribir PHP con directivas de inclusión

- Podemos escribir el código php escribiéndolo en un fichero aparte y luego lo incluimos.
  - Incluimos el fichero explícitamente
  - Para ello usamos directivas o instrucciones del tipo ***include***

```
include('ruta/nobrefichero');
require('ruta/nobrefichero');
include_once('ruta/nombrefichero');
require_once('ruta/nombrefichero');
```

Ambas son palabras reservadas del lenguaje y sirven para incluir el contenido de un fichero con sentencias php en esa posición del código.

Include Vs Require

- Ambas incluyen el contenido de un fichero php en esa posición
  - Con include si no se encuentra se continúa ejecutando en esa posición
  - Con require si no está el fichero se detiene en ese punto la ejecución del script

include/require Vs include_once/require_once

- Ambas incluyen el contenido de un fichero php en esa posición
  - include/require siempre buscan e incluyen el fichero en esa posición
  - include_once/require_once antes de incluirlo mira a ver si ya lo incluyó previamente en cuyo caso ya no lo hace

Probando include require

Crea 4 ficheros con el siguiente código

fichero_include.php

```
<?php
  echo "<b><i>Hola desde un fichero include </b></i><hr />";
?>
```

fichero_include_once.php

```
<?php
 echo "<b><i>Hola desde un fichero include once </b></i><hr />";
?>
```

fichero_require.php

```php
<?php
  echo "<b><i>Hola desde un fichero require </b></i><hr />";
?>
```

fichero_require_once.php

```php
<?php
  echo "<b><i>Hola desde un fichero require once </b></i><hr />";
?>
```

Ahora crea un programa principal dónde uses las instrucciones de inclusión vistas en este apartado.

```php
<?php

echo "<h2>Ahora vamos a incluir un fichero con include</h2>";
include 'ficheros/fichero_include.php';

echo "<h2>ahora vamos a incluir un fichero con require</h2>";
require 'ficheros/fichero_require.php';

echo "<h2>Ahora vamos a incluir un fichero con include_once</h2>";
include_once 'ficheros/fichero_include_once.php';

echo "<h2>Ahora vamos a incluir un fichero con require_once</h2>";
require_once 'ficheros/fichero_require_once.php';

echo "<h2>Ahora vamos a incluir un fichero que no existe  con include</h2>";
include 'ficheros/fichero_no_existe_include.php';
echo "Vemos que no pasa nada, por que el fichero no existe pero sigue <hr />";

echo "<h2>Ahora volvemos  a incluir un fichero con include_once</h2>";
include_once 'ficheros/fichero_include_once.php';
echo "Vemos que no pasa nada, por que el fichero ya se había incluido y no se vuelve a incluir<hr />";

echo "<h2>Ahora volvemos  a incluir un fichero con require_once</h2>";
require_once 'ficheros/fichero_require_once.php';
echo "Vemos que no pasa nada, por que el fichero ya se había incluido y no se vuelve a incluir<hr />";

echo "<h2>Ahora vamos a incluir un fichero con include para ver que sí que se vuelve a incluir</h2>";
include 'ficheros/fichero_include.php';

echo "<h2>Ahora vamos a incluir un fichero con require y vemos que sí se vuelve a incluir</h2>";
require 'ficheros/fichero_require.php';

echo "<h2>Ahora no incluimos con require un fichero que no existe</h2>";
require 'ficheros/fichero_no_existe_require.php';
echo "Esta línea ya no se imprimirá ni nada que vaya después de aquí";
?>
```

Tip:

- Siéntete cómoda en modificar el fichero y ver el resultado
- Observa que estas instrucciones no son exactamente funciones, por lo que no necesitan paréntesis (aunque se puede poner por mantener una homogeneidad con sintaxis del uso o invocación de funciones)

```
include 'ficheros/fichero_include.php'
//Es léxicamente y sintácticamente igual que
include ('ficheros/fichero_include.php');
```

Recursos de la Web

- php (http://php.net/manual/es/) página oficial en español LO MEJOR!!!!
- http://php.net/manual/es/
-

Comentarios

- Son ignorados por el intérprete, no generan instrucciones, pero se consideran parte del software
- En php tengo 4 formas de hacer comentarios

```php
<?php
 /*
 Este código no hace nada pero muestra
 la sintaxis de los comentarios
 como este que ocupa varias líneas tipo lenguaje C o Java
 */
 $miVariable= 8;// Esta parte de la línea se ignorará
 $miVariable+= 8;# y esta parte de línea tambien
 echo "Valor de la variable $miVariable";
//Este es otro comentario de una sola línea
#Este es otro modo de hacer comentarios tipo script de linux
/**
Este comentario permite insertar información
Para que luego phpDocumentor genere una página web
Con la información de mi código
*/
```

- Así como comentar las funciones como veremos
- Este código nos visualizará lo siguiente

```
Valor de la variable 16
```

El resto del código será ignorado

- Es bueno escribir al principio del script

```php
<?php
/**
 * User: Nombre y apellidos
 * Date: 19/10/17
 * Version: 17:40
 */
```

<>/div

- Aquí tienes la referencia para ver las diferentes directivas para generar código
- Más adelante en el curso las veremos.

Recursos de la Web

- Directivas para comentarios

https://phpdoc.org/docs/latest/index.html

probando comentarios

- Escribe la siguiente función anterior en un fichero php

```
function miFuncion($num1, $num2){
    if ($num1>$num2)
        return $num1;
    else
        return $num2;
```

Ahora justo en la línea de encima de función escribe

```
/**
```

- y luego presiona intro
- Te debería de quedar

```
/**
 *
 * @param type $num1
 * @param type $num2
 * @return type
 */
function miFuncion($num1, $num2){
    if ($num1>$num2)
        return $num1;
    else
        return $num2;
```

Programa: conjunto de instrucciones

Objetivo

Un programa es un conjunto de instrucciones

- Analizaremos las instrucciones que hay.
- Posteriormente Veremos cómo se escriben en php.

Planteando un lenguaje de programación

Léxicos

- Son las palabras reservadas del lenguaje.

```
http://php.net/manual/es/reserved.keywords.php
```

Sintaxis

```
Reglas de construcción.
Son las ya conocidas, pero veremos cómo se construyen las expresiones.
```

Semántica

```
Habla del significado.
```

Puntos clave

Estudiaremos alguna peculiaridad, como el hecho de que php es un lenguaje *altamente orientado a expresiones*

Instrucciones en un lenguaje de programación

1. Inicio Fin de bloque
2. Instrucción/función de leer del teclado, escribir por pantalla
3. Declaraciones (variables, constantes, funciones, clases, objetos, ...)
4. Asignación
5. Invocación (llamada a función o método)
6. Estructura de control (selectiva, iterativa)

Separando instrucciones

- Para separar una instrucción de otra usaremos ***;*** (punto y coma)
- Su uso es obligatorio a excepción de la última instrucción que se puede obviar
- Esto es por que el fin de código php ***?>*** implica esta instrucción
- Nosotros mejor lo usaremos siempre.
- La instrucción inicio de bloque y fin de bloque no lleva nunca ***;***

Tip: Si solo queremos insertar una instrucción puede suele obviarse el ***;***

```
<html>
  ......
  <?php echo "hola" ?>
  <!-- instrucciones html -->
  <?php echo "otro hola" ?>
  <!-- mas instrucciones html -->
  <?php echo "otra instrucción " ?>
  .....
</html>
```

Tip: También puede haber ;

```
<html>
  ......
  <?php echo "hola" ;?>
  <!-- instrucciones html -->
  <?php echo "otro hola" ; ?>
  <!-- mas instrucciones html -->
  <?php echo "otra instrucción " ; ?>
  .....
</html>
```

Instrucción de inicio fin de bloque

```
{ //Instrucción de inicio de bloque
} //Instrucción de fin de bloque
```

Funciones de salida

- Construcciones básicas para salida de caracteres
- En PHP,en realidad no son funciones por lo que pueden ir sin paréntesis (con o sin paréntesis):

1. ***echo***
2. ***print***

- Existen otras funciones que iremos viendo según avance el curso

echo

- Es el uso más sencillo
- Imprime una cadena como argumentos
- En la versión ***sin paréntesis*** , también puedes pasar una lista de argumentos.

```php
<?php
*echo “primer argumento”, “segundo argumento”, “tercer argumento”
?>
```

print

- Esta sentencia es igual en uso y funcionalidad que echo
- Tiene dos diferencias con echo

1. Sólo puede aceptar un argumento
2. Devuelve un valor booleano que representa si la sentencia ha tenido exito o no

Observa el siguiente código e indica si es o no correcto

1.
```php
echo 'hola caracola', 'hola','como estás'
```
   - ( ) Correcto
   - ( ) Incorrecto

2.
```php
print 'hola caracola', 'hola','como estás'
```
   - ( ) Correcto
   - ( ) Incorrecto

Declaraciones

Objetivo

Una declaración es una instrucción muy importante

- Aparentemente no hace nada (no tienen acción directa)
- Permite hacer luego cosas con los elementos declarados

Declaraciones

1. De variables
2. De constantes
3. De funciones
4. De clases
5. De objetos y recursos (clases ya creadas o incluidas)

Tip: Las funciones,clases y objetos los veremos en otro tema

Pregunta

- Qué es un ***tipo de dato***
- Qué es una ***variable***

- Hay cuestions que son importantes tenerlas claras
- Son temas de programación básica

Definición

Un tipo de dato es un conjunto de valores para los cuales hay definidos una serie de operaciones

Definición

Una variable es una posición de memoria que va almacenar algún valor de un determinado tipo, y cuyo contenido puede variar durante la ejecución de un programa

PHP

Tipado dinámico

- Una características semántica muy, muy importante de php

Puntos clave

PHP es un lenguaje tipado

- Esto es por que en todo momento todo valor y variable tienen un tipo
- Podemos ver su tipo, con funciones del lenguaje

- Los lenguajes de programación pueden ser mas o menos exigentes en cuanto a la declaración de los tipos de las variables para poder ser usados durante la ejecución de un programa.

tipado dinámico

- Hay lenguajes fuertemente tipos (Java) o débilmente tipados (Basic)
- Esto tiene que ver con el hecho de que cada variable en un momento dado tiene un tipo, y lo podemos saber
- Esto ocurre en php

Tipado dinámico

- ***Php*** no es estricto en el tipo de dato de una variable, en cuanto que éste puede cambiar durante su vida.
- En este sentido php es un lenguaje de ***tipado dinámico***, el tipo de la variable depende del valor que tiene en un momento dado o de los operadores que lo afecten.
- La declaración de tipo no existe de forma explícita
- La declaración de tipo ocurre en el momento que a una variable se le asigne un valor de un tipo.
- Si le asigno un valor de otro tipo, vuelve a ocurrir una delcaración de tipo

Puntos clave

PHP es un lenguaje de tipado dinámico

Definir variables

- En php una variable es definida la primera vez que se usa.
- El tipo de la variable depende del valor que tenga asignado en un momento dado
- El identificador de la variable en php tiene que empezar por el signo ***$***
- En 'php', las variables se representan con el signo ***$*** seguido de un carácter de subrayado o una letra y luego letras, números y caracteres de subrayado en cualquier orden y número.

PHP y variables

Resumen

1. Php es un lenguaje tipado
2. Php tipado dinámico
3. Php no es estricto en la declaración de variables (La declaración ocurre la primera vez que se usa)

identificador de variables

```
identificador = $[_|a.zA..Z][_|a..zA..Z0..9]*
```

}}

```
<?php
   $miVariable= 8;/*Variable de tipo entero*/
   edad = 5/*Error en el identificador*/
   $5edad = 5 /*Error en el identificador */
?>
```

Sensitive case?

- El lenguaje es sensible a mayúsculas y minúsculas en los siguientes casos:
  - En los identificadores de variables

```
$edad =10;
$Edad =20;
$edad y $Edad son 2 variables diferentes
```

  - En los nombres de funciones

```
function  $calculaEdad($anyo){
...
```

```
}
function $CalculaEdad($anyo){
...
}
//Son dos funciones diferentes
```

- El lenguaje ***NO*** es sensible a mayúsculas y minúsculas en:
  - las palabras reservadas (if o If o IF o iF,...)

```
If () {
 //..
}
IF () {
 //..
}
if () {
 //..
}
iF () {
 //..
}
//Todas las construcciones if son correctas
```

Valores y tipos de datos

```
http://php.net/manual/es/language.types.intro.php
```

- En Php tenemos 8 tipos de datos

1. 5 tipos básicos o primitivos (un valor)
2. 3 tipos compuestos (conjunto de valores).

Tipos primitivos

1. ***boolean***: Valores TRUE y FALSE.
2. ***entero***: números enteros ... -2,-1,0,1,2
   ...

3. ***real***: números reales
4. ***string***: Cadena de caracteres
5. ***NULL***: Valor null

Tipos compuestos

1. ***array***: conjunto enumerado por la clave de valores de diferente tipo
2. ***objeto***: Instancia de una clase en memoria
3. ***recurso***: objeto que permite utilizar elementos del sistema o **recursos**, (por ejemplo conector a una base de datos)

tipos básicos

1. **entero** *integer*

- Posible notación decimal/octal/hexadecimal

```
decimal  [0..9]+
hexadecimal 0x[0..f]+
octal 0[0..7]+
binario 0b[01]+
```

- Todos ellos pueden ser positivos o negativos

```
**$Numero=10;
*Octal
**$NumeroOctal=067;
*Hexadecimal
**$NumeroHex=0cA56B;
```

integer

- Al imprimirlos con print los veré con valor decimal
- Para verlos en otras base hay que usar printf o format o utilizar las conversiones dechex o dechex o octdec, ... que veremos en otro apartado.

cadena *string*

```
$frase="Esto es un literal de cadena de caracteres"
```

- real o coma flotante ***float***

```
$valor=$0.2345;
$valor=.54;
$valor=7E-12;
```

Booleano *boolean*

```
$estado=TRUE;
$estado=TrUe;
$estado=falsE;
```

NULL

- un tipo especial que solo tiene ese valor
- Una variable tiene el valor null

1. Si aún no se le ha asignado valor, o éste se ha destruido (unset()
2. Si se le ha asignado explícitamente el valor NULL.

```php
$a=NULL;
$a=null;
```

Tipos complejos

- Objetos básico en su aspecto de OOP
- Matrices o arrays muy muy utilizados
- Recursos este más que un tipo complejo es un tipo especial que hace referencia a un recurso externo referencia , como una conexión a una base de datos o como una referencia a un fichero pdf.
- Este tipo de variables las veremos más adelante

Ejercicio 1.- Declaración de variables

- Haz un programa donde declares variables de diferentes tipos
- Asigna los valores con diferente formato
- Visualiza sus valores

Constantes

Objetivo

Las constantes se declaran una vez
No se pueden modificar, solo usar

Constantes

- Se definen con la función ***define()***

```
define("IVA",0.21);
$total=$base*(1+IVA);
```

identificador

- Se usa el mismo criterio de construcción pero no empieza por $
- Se pueden definir y utilizar en cualquier momento que se necesiten.
- Para saber si una constante está definida ***defined()***

Constantes predefinidas

- Como en otros lenguajes, existen una serie de constantes predefinidas
- Nos las ofrece el entorno y dependerán de él para su valor
- PHP Ofrece un gran número de constantes predefinidas http://php.net/manual/es/reserved.constants.php
- En php hay 8 constantes que su valor puede cambiar dependiendo del entorno donde se ejecutan

```
constantes (http://php.net/manual/es/language.constants.predefined.php) predefinidas en php
```

Funciones

Objetivo

Las funciones es un elemento fundamental

- Permite crear código modular
- Una forma de estructurar nuestro programa

Declaración de funciones

```php
function nombreFuncion ($paramFormal1, $paramFormal2 ,...){
  //Instrucciones de la función
  return $valorRetorno //Opcionalmente en caso de que devuelva algún valor la función
}
```

- Es importante diferenciar entre declarar una función e invocar a una función
- Algo obvio, pero importante
- En la declaración tenemos tres partes

1. nombre o identificación de funciones
2. parámetros formales entre paréntesis (Estos han de existir, aunque no haya parámetros)
3. Cuerpo de la función, dentro de él puede estar la instrucción return, en cuyo momento termina la ejecución de la función y se vuelve a la siguiente instrucción del programa, siguiente a la invocación de la función.

Identificador de función

- El nombre de función es un identificador que empieza por una letra o guión bajo, seguido 0 o muchas letras, números o guiones bajos

  Tip: Expresión regular para el identificador de funciones

```
[a-zA-Z_f_][a-zA-Z0-9_]*
```

Parámetros formales

- Son nombres de variables que usaré al escribir el código o cuerpo de la función
- El nombre ha de ser significativo y se convertirán en variables locales a la función
- Una vez que se termina la función estas variables desaparecerán de memoria

parámetros formales

Los parámetros formales son variables locales a la función

Ejercicio usando funciones

Haz un programa donde en el programa principal se creen dos variables $a y $b

- Crea una función que reciba como parámetros locales ***$a*** y ***$b***
- La función visualizará el valor de las variables, las modificará y las volverá a visualizar
- El programa principal

1. asignará valor a las variables
2. las visualizará
3. invocará a la función
4. volverá a visualizar las variables

- Una posible solución

```php
<?php
   function a($a, $b){
       echo "Dentro de la función visualizando valores <hr />";
       echo "Valor de los parámetros \$a = $a \$b = $b <br />";
       $a+=5;
       $b+=5;
       echo "Valor de los parámetros \$a = $a \$b = $b <br />";
       echo "Salgo de la función";
   }
   //Ahora considero programa principal
   $a=100;
   $b=200;
   echo "En el main antes de invocar a la función visualizando variables<hr />";
   echo "Valor de variables \$a = $a \$b = $b <br />";
   a($a,$b);
   echo "En el mail  después de invocar a la función visualizando variables<hr />";
   echo "Valor de variables \$a = $a \$b = $b <br />";
?>
```

Parámetros formales: Valores y referencias

- Cómo hemos visto, los parámetros formales son valores pasados en la invocación a la función
- Si queremos que la función pueda modificar el valor de los valores de los parámetros, en este caso hemos de pasarlos por referencia
- En este caso lo que ocurre en realidad es que pasamos la dirección de memoria dónde se guarda el valor.
- La dirección de memoria, no la podremos visualizar ni operar con ella, pues en php no existe la aritmética de punteros o direcciones de memoria

Parámetros formales

Valores y referencias

Para pasar el parámetro por referencia, simplemente hay que poner el símbolo de dirección de memoria **&** antes del nombre de la variable en la declaración de parámetros

```php
function nombre_funcion(&$paramRef1, &$paramRef2, $paramVal1){
   ...
}
```

Ejercicio usando funciones parámetros

Haz un programa donde en el programa principal se creen dos variables $a y $b y $c

- Crea una función que reciba como parámetros locales ***&$num1***,***&$num2*** y ***$num3***
- La función visualizará el valor de las variables, las modificará y las volverá a visualizar
- El programa principal

1. asignará valor a las variables
2. las visualizará
3. invocará a la función
4. volverá a visualizar las variables

```php
<?php
     function a(&$num1, &$num2, $num3){
         echo "Dentro de la función visibilizando valores <hr />";
         echo "Valor de los parámetros \$num1 = $num1 \$num2 = $num2 \$num3 = $num3<br />";
         $num1+=5;
         $num2+=5;
         $num3+=5;

         echo "Valor de los parámetros \$num1 = $num1 \$num2 = $num2 \$num3 = $num3<br />";
         echo "Salgo de la función";
     }
     //Ahora considero programa principal
     $a=100;
     $b=200;
     $c=300;
```

```
        echo "En el main antes de invocar a la función visualizando variables<hr />";
        echo "Valor de variables \$a = $a \$b = $b \$c = $c <br />";
        a($a,$b,$c);
        echo "En el mail  después de invocar a la función visualizando variables<hr />";
        echo "Valor de variables \$a = $a \$b = $b \$c = $c <br />";
?>
```

Invocando funciones

- Una vez creada una función la podemos invocar como si fuera una instrucción del lenguaje
- No sin razón en determinados ambientes se conoce a las funciones y procedimientos como instrucciones virtuales ...
- En php puedo invocar a una función antes de declararla, siempre que la declare en el mismo fichero

ejemplo invocación a funciones

Tip: Este código funcionará correctamente

```
<?php
  a(5,6);
  /*Mas instrucciones*/
  function a ($a, $b){
      echo "valor de $a";
      echo "valor de $b";
  }
```

ejemplo invocación a funciones

Tip: Este código no funcionará

```php
<?php
      a(5,6);
      /*Mas instrucciones*/
      include ("funciones.php");
?>
```

- Contenido del ficheor funciones.php

```php
<?php
function a ($a, $b){
          echo "valor de $a";
          echo "valor de $b";
}
?>
```

Variables dentro de una función

- Dentro de una función las variables que declaremos son locales a esa función.
- No podré acceder a su valor fuera de la función
- Esto también implica que dentro de una función no puedo acceder al valor de una variable definida fuera de la función
- Observa el siguiente ejemplo

```php
<?php


function modifica_valor(){
    echo "Valor de <b>var1</b> dentro de función -$var1- <br /> ";
    $var1++;
    echo "Valor de <b>var1</b> dentro de función moficada -$var1- <br /> ";
}

$var1 = 20;
```

```php
echo "Valor de <b>var1</b> en programa principal antes de invocar función: -$var1- <br />";
modifica_valor();
echo "Valor de <b>var1</b> en progrma principal después de invocar la función: -$var1- <br />";
?>
```

- Vemos que genera la siguiente salida

```
 Valor de var1 en programa principal antes de invocar función: -20-
Valor de var1 dentro de función --
Valor de var1 dentro de función moficada -1-
Valor de var1 en progrma principal después de invocar la función: -20-
```

- Sin embargo si quieremos acceder al valor de ***$var*** dentro de la función, sí que podemos
- Ĥemos de usar la palabra reservada ***$global***

```php
<?php

function modifica_valor(){
    global $var1; //Indicamos que esta variables se puede globalizar
    echo "Valor de <b>var1</b> dentro de función -$var1- <br /> ";
    $var1++;
    echo "Valor de <b>var1</b> dentro de función moficada -$var1- <br /> ";
}

$var1 = 20;

echo "Valor de <b>var1</b> en programa principal antes de invocar función: -$var1- <br />";
modifica_valor();
echo "Valor de <b>var1</b> en progrma principal después de invocar la función: -$var1- <br />";
?>
```

- Ahora podemos observar cómo sí que se accede al valor dentro de la función

```
Valor de var1 en programa principal antes de invocar función: -20-
Valor de var1 dentro de función -20-
Valor de var1 dentro de función moficada -21-
Valor de var1 en progrma principal después de invocar la función: -21-
```

Funciones propias de php sobre tipos y valores

```
http://php.net/manual/es/ref.var.php
```

- Existen una serie (muchas) de funciones que son interesantes de conocer
- Estas funciones ya están creadas y se pueden usar directamente
- Están relacionadas con los tipos de datos y valores
- Alguna de ellas son extremadamente útiles y utilizadas, por ejemplo antes de procesar un dato, hay que ver que dicho dato tenga valor.
- A continuación trataremos alguna de ellas

var_dump (http://es1.php.net/manual/es/function.var-dump.php)

```
void var_dump($expresion)
```

- Nos da información sobre la estructura de un valor resultado de una expresion

isset (http://es1.php.net/manual/es/function.isset.php)

```
bool isset ( $variable )
```

- verifica que una variable tiene valor (está definida y no tiene un valor null)

```
<?php
 $VariableValor= 5;
 print ("El valor de la variable es $VariableValor");
 print ("El valor de otra variable es $OtraVariableValor");
 if (isset($VariableValor))
     print ("VariableValor tiene valor asignado");
 else
     print ("VariableValor no no tiene valor asignado");
 if (isset($OtraVariableValor))
     print ("OtraVariableValor tiene valor asignado");
 else
     print ("OtraVariableValor no no tiene valor asignado");
  ?>
```

Funciones para determinar existencia de variables

Tenemos tres funciones muy parecidas pero no del todo iguales

Función	Significado
is_null($variable)	Determina si una variable ($variable) tiene valor null
empty($variable)	Determina si una variable ($variables)está vacía
isset($variable)	Determina is una variable ha sido definida y no tiene un valor vacío.

- Es importante saber qué es para php un valor nulo, o si está vacía que no son conceptos sinónimos

Valor null

```
$a=null //$a tiene valor null.
is_null($a) //true
unset($a) //Se destruye la variable y toma el valor null
is_null($a) //true
//$b una variable que no existe tiene el valor null
is_null($b) //true
```

Variable vacía

```
$a=null //$a está vacía
empty($a) //true
$a="";
empty($a) //true
$a="hola";
empty($a) //false
unset($a);
empty($a) //true
$a=false;
empty($a) //true !OJO!
$a=0;
empty($a) //true !OJO!
```

- Puedes ver la siguiente app en la que puedes aportar valores

```
http://manuel.infenlaces.com/apuntes/existencia_valor_variables
```

- Tener en cuenta que si evaluamos si una variable está vacía no es

empty (http://es1.php.net/manual/es/function.empty.php)

```
bool empty ($varriable)
```

- Determina si una variable no existe. Devuelve true si no existe o su valor está vacío

Actividad

Probamos las fuciones var_dump() que nos da información sobre el valor y el tipo

Actividad

Usando la función xxxyyy donde xxx e yyy será dec oct bin o hex para convertir el valor de un sistema numérico a otro

Actividad

- Define las siguientes variables que se especifican en el código siguiente y verifica el resultado con empty()

```
$num=0;
$nombre="";
$nombre=null;
$nombre="0";
$pregunta = FALSE;
```

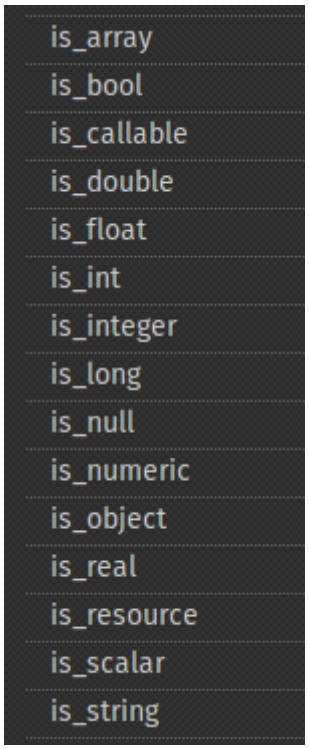
gettype (http://es1.php.net/manual/es/function.gettype.php)]

- Devuelve el tipo de una variable

```
string gettype($variable)
```

[1] (http://es1.php.net/manual/es/function.is-bool.phpis_bool)is-double (http://es1.php.net/manual/es/function.is-double.php) is-int (http://es1.php.net/manual/es/function.is-int.php), is-xxx

- son funciones donde xxx especificado en el último nombre, puede ser cualquiera de los tipos



- Todas ellas devuelve un booleano que indica si la variable, valor o expresion es o no de ese tipo,

```
string is_int($variable);
string is_double($variable);
string is_bool($variable);
string is_integer($variable);
string is_null($variable);
string is_string($variable);
...
```

Actividad

Visualizar de qué tipo es la expresión mostrada en el código siguiente y visualiza el valor de la expresión

```
$a=5;
```

unset (http://php.net/manual/es/function.unset.php)

- Destruye la variable especificada perdiéndose su valor

void unset ($var)

Cadenas

Objetivo

Cómo se trabaja con estos valores
como puedo concatenar y

- Cómo incluir en una cadena
  - valores de variables
  - retorno de funciones

- valores de expresiones

Php y los valores de tipo cadena

- En php las cadenas de caracteres, son expresiones literales.
- Tenemos 4 maneras diferentes de poder expresar una cadena de caracteres como un literal.
- Comillas dobles ""
- Comillas sencillas ''
- Sintaxis ***heredoc***
- Sintaxis ***nowdoc***

Comillas dobles

- En ellas se interpretan los caracteres especiales.

1. ***$*** seguido de un nombre, interpreta que es una variable y toma su valor (null si no tiene valor o no está definida).
2. **** es un carácter de secuencia de escape, e interpreta que el carácter siguiente tiene un significado especial \\ \a \n \r \t , ...

- Si queremos que se ignore un carácter especial, éste ha de ir precedido por el caracter ****

```
$nombre = 'pedro';
echo "El valor de la variable \$nombre es $nombre";
```

- El resultado sería

```
El valor de la variable $nombre es pedro
```

Comillas simples

- En ellas solo se interpreta el carácter, seguido de \ o bien seguido de la barra invertida **** comilla simple **\'**
- El resto de caracteres no se interpretan.

```php
$nombre = 'pedro';
echo 'El valor de la variable $nombre es $nombre y \\ \'texto\' sí que se ve entre comillas simples';
```

- La salida sería

```
El valor de la variable $nombre es $nombre y \ 'texto' sí que se ve entre comillas simples';
```

Heredoc

- Este tipo de expresión de string es útil para especificar cadenas largas en multilíneas
- Se comporta como un string entre comillas dobles para el tema de interpretar y escapar ciertos caracterres
- Se establece con el operador **<<<**
- A continuación viene un identificador
- Después empieza a especificarse la cadena de caracteres
- Para finalizarla se escribe en una nueva línea el identificador

```php
<?php
$frase = <<<FINAL
Esta es una cadena
de caracteres que se asignará
a la variable frase
y termina con la palabra
con la que hemos empezado
FINAL;
<?
```

- MUY IMPORTANTE: ***La palabra final no debe tener ningún espacio después, ni tabulador antes***.

NewDoc

- Es igual que heredoc , pero sin interpretar los caracteres especiales salvo **\\ \.**
- O sea que es como un entrecomillado sencillo
- La sintaxis es igual que la de heredoc, pero a diferencia el delimitador que se especifica al principio debe de ir entrecomillado con comillas simples

```php
<?php
$nombre=pedro;
$frase = <<<'FINAL'
```

```
El valor de $nombre
es $nombre, pero aquí
no lo veo por que es newdoc
FINAL;
```

Estructuras de control

Objetivo

Determinan el flujo de ejecución de un programa

- Tenemos tres estructuras de control
- Veremos cómo se implementan en PHP

Estructuras de control 1

A continuación veremos las estructuras de control Son de tres tipos

1. Selección
2. Iteracción
3. Secuenciales

- Para construirlas necesitamos operadores

Selección if

- Sentencia que evlúa una expresión booleana y ejecuta o no en función de que dicha expresión sea true o false

```
if (condicion)
    Sentencia 1;
```

```
if  (condicion){
    Sentencia_1;
    Sentencia_2;
}
```

Sentencias de control

if (expresion)

```
   Sentencia_1;
```

else

```
    Sentencia_2;
```

</source>

- También existe la opción elseif donde aportaremos una condición que se ha de cumplir para que se ejecuten las sentencias que a continuación acompañan.

Estructura de control

```
if (expresion){
   sentencias;
}
elseif (expresion){
   sentencias;
}
else{
   sentencias;
}
```

- Alternativamente puede usarse esta sintaxis que es usada cuando se quiere intercalar condigo html fuera del php.
- También se puede usar la sintaxis vista anteriormente, pero parece que esta quede más compacta.

```
if (condicion):
   Sentencia 1;
 endif;
```

Ahora lo vemos con código html

```
<?php if (true): ?>

    <h1>Esta frase seguro que aparece ahor</h1>
    <!--escribirmos código html -->
<?php else: ?>
    <h1>Aquí escribiré poco ya que no va a aparecer nada</h1>
    <!--escribimos código html -->
<?php endif ?>
```

Operadores ternario

- Es una forma más compacta de un if else con una única instrucción.

```
Expresión? SentenciaOKExpresion : SentenciaNoOkExpresion
```

Actividad

Programa que me de si un número aleatorio es par o impar

Seleccion switch

- Este es un selector múltiple
- La sentencia case puede albergar cualquier valor de un tipo simple, no está limitado a enteros como en otros lenguajes
- Estructura indicada cuanto tengamos más de dos casos ante una variable o situación que evaluemos excluyentes entre sí

Switch

```
<?php
switch ($nombre){
    case "Maria":
```

```
        echo "eres una chica";
        break;
    case "Pedro";
        echo "eres una chico";
        break;
    default:
        echo "no se qué nombre tienes";
}
?>
```

Iteración while

- Como en todos los bucles debemos siempre tener encuenta

1. inicializar la variable de control
2. actualizarla correctamente dentro del bucle
3. realizar de forma correcta la evaluación de condición (< o <=), (> o >=), ...

```
<?php
$i = 1;
while ($i <= 10) {
    echo "iteracción número ."$i++;
}
?>
```

- Alternativamente podemos usar la siguiente sintaxis

```
$i = 1;
while ($i <= 10):
    $i++;
    echo "iteracción número ."$i;
endwhile;
?>
```

Iteracción do-while

- Este tipo de bucle donde seguro que al menos se ejecuta un iteracción
- Respecto al anterior nos ahorra una comparación.

```
<?php
$num=10;
```

```php
$resultado=1;
*Esta es la única sintaxis posible con este tipo de sentencia
do {
   $resultado:=$resultado*$num;
   $num--;
} while ($num>0);
?>
```

Iteración for

- Es un bucle de tipo contador

```php
for (expresion_inicial; condicion;expresion_actualizar){
      sentencias;
}
```

Estructura for

- tiene tres partes

expresion_inicial

Se ejecuta una sola vez al comienzo del bucle. se usa para inicializar variables

condición

Es una expresión booleana que se evalúa en cada interacción

Si da un valor false, ya no se ejecuta ninguna vez

Si no hay expresión se toma como true

En este caso para que el bucle no sea infinito deberá llevar algún break (instrucción de terminación de bloque) en algún momento

Estructura for

condición

```php
<?php
for ($a=0; ;$a++){
    echo "$a*$a=".$a*$a."<br>";
    if ($a==10)
        break;
}
?>
```

Estructura for

expresion_actualizar
Esta expresión actualiza el valor de alguna/as variables
Se ejecuta en cada interactivo

- El ejemplo anterior

```php
<?php
for ($a=0;$a<10 ;$a++){
    echo "$a*$a=".$a*$a."<br>";
}
?>
```

Operadores y expresiones

Objetivo

Son partes de las frases de un lenguaje de programación

Operadores

- Son símbolos que realizan acciones sobre operandos y dan como resultado un valor
- Tenemos diferentes tipos de operadores en función del tipo de operandos y del resultado

operadores aritméticos (+,-,*,/,%,, ++, --)**

- Retorna un valor numérico
- el ++, -- son valores de autoincremeto y autodecremento, pueder ser pre o post

Operadores

```php
$a=5;
if ($a++==5)
    echo '$a que vale '.$a.' dice que vale 5 ???? <br>'
        . 'Esto es por que primero compara y luego incrementa<br>';

echo 'ahora $a vale '.$a.'<br>';

if (++$a==6)
    echo 'esto nunca saldrá ya que $a se incrementa antes de comparar';
else
    echo 'efectivamente ahora $a ya no vale 6 sino '.$a.'<br>';
?>
```

Operadores

- El código anterior genera la siguiente salida

```
$a que vale 6 dice que vale 5 ????
Esto es por que primero compara y luego incrementa
ahora $a vale 6
efectivamente ahora $a ya no vale 6 sino 7
```

operadores comparación (==,<,>,>=,<=,<>,!=,===,!==)

Este tipo de operadores genera un booleano como resultado de evaluar la expresión

Puntos clave

- **==** operador de comparación ***igual que*** (mismo valor)
- **===** operador de comparación ***exactamente igual que*** (mismo valor y tipo)

Operador == Vs ===

```php
$num=1;
if ($num==true)
    echo '$num es igual a true<br>';
if ($num===true){
    echo "esto nunca se ejecutará";
}else
    echo '$num no es exactamente igual a true';
```

Operador == vs ===

- El código anterior generaría la siguiente salida

```
$num es igual a true
$num no es exactamente igual a true
```

- Ver la sección ***comparación de tipos*** de la página oficial

```
http://php.net/manual/es/language.operators.comparison.php
```

operadores de concatenacón(.) concatena cadena de caracteres.

El operador + no está sobre cargado, observa el siguiente código

```php
$nombre="Maria";
$apellido = " de la Oh";

$nombreCompleto = $nombre.$apellido;
echo "el valor de nombre completo es $nombreCompleto ---<br>";

$nombreCompleto = $nombre+$apellido;
echo "el valor de nombre completo es $nombreCompleto --<br>";
```

La salida del código anterior sería

```
el valor de nombre completo es Maria de la Oh ---
el valor de nombre completo es 0 --
```

Operadores de asignación (= , =>)

Se pueden combinar con los aritméticos (+=, *=, ...) y con los de concatenación (.=)

En este caso el valor de la variable de la izquierda se toma como primero operando

Operador de asignación

```php
<?php
$b=1;
for ($a=0;$a<10;$a++){
    $b*=10;
    echo 'valor de $b ='.$b.'<br>';
}
?>
```

- El código anterior genera la siguiente salida

```
valor de $b =10
valor de $b =100
valor de $b =1000
valor de $b =10000
valor de $b =100000
valor de $b =1000000
valor de $b =10000000
valor de $b =100000000
valor de $b =1000000000
valor de $b =10000000000
```

operadores de ejecución (``)

PHP proporciona un operador especial que permite ejecutar sentencias

- Observa el siguiente código

```php
<?php
$Discos = `df`;
echo "<pre>$Discos</pre>";
?>
```

- El código anterior generará la siguiente salida

```
Filesystem     1K-blocks     Used Available Use% Mounted on
/dev/sda5       86378608  6072360  75895384   8% /
none                   4        0         4   0% /sys/fs/cgroup
udev             4023720        4   4023716   1% /dev
tmpfs             806904     1384    805520   1% /run
none                5120        0      5120   0% /run/lock
none             4034504     6588   4027916   1% /run/shm
none              102400       28    102372   1% /run/user
/dev/sda7      101797224 40480360  56122728  42% /home
```

Invocando funciones del sistema

- El operador anterior (comillas invertidas) es igual que la función ***shell_exec()***

```
http://php.net/manual/es/function.shell-exec.php
```

operadores lógicos (and,&&, or, ||, xor !)

```
http://php.net/manual/es/language.operators.logical.php
```

Funcionan por cortocircuito
El operador ***xor*** da verdad si los operando son de diferente valor uno true y el otro false
La notación ***and*** y ***&&*** representan el mismo operador, igual ocurre con ***or*** y **||**

- La diferencia entre los operadores es la prioridad

```
http://php.net/manual/es/language.operators.precedence.php
```
