Introducción PHP

Introducción I

- PHP es un **lenguaje de programación** que se utiliza para añadir programas dentro del código de una página **web**.
- Sus siglas significan *Hypertext Preprocessor.*
- Se trata de un lenguaje ejecutado **desde el servidor**(ASP, JSP,...) al contrario que, por ejemplo, Javascript, que se ejecuta en el navegador.
- La sintaxis de PHP es **muy similar** a la de Javascript , y por tanto únicamente se trataran en profundidad las peculiaridades que lo diferencian de él.

Introducción II

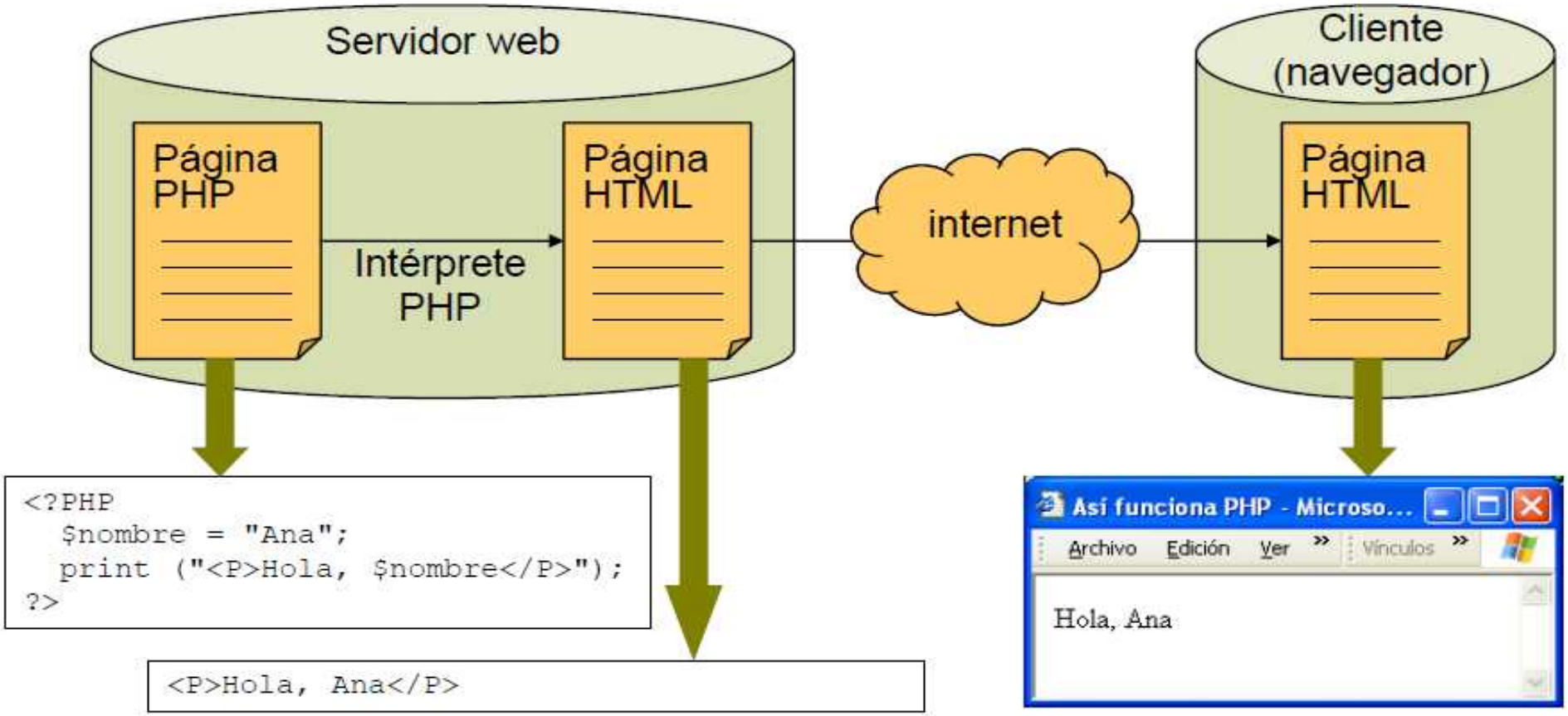
- Los scripts PHP están incrustados en los documentos HTML y el servidor los interpreta y ejecuta antes de servir las páginas al cliente.
- El cliente no ve el código PHP sino los resultados que produce
- Versión actual: PHP 7
- Potente, fácil de aprender, de libre distribución y permite el acceso a bases de datos.
- http://es.wikipedia.org/wiki/PHP
- http://php.net/manual/es/index.php

Funcionamiento I

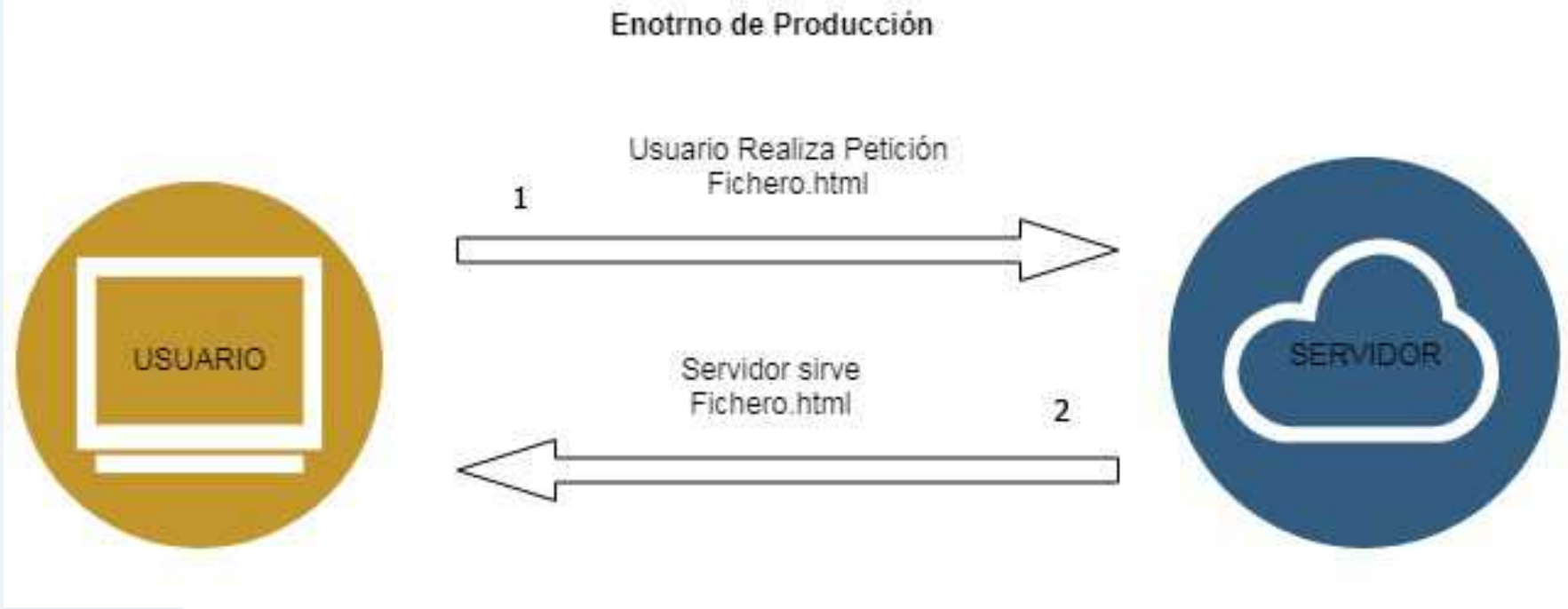
- Los scripts PHP están incrustados en los documentos HTML y el servidor los interpreta y ejecuta antes de servir las páginas al cliente.

- El cliente no ve el código PHP sino los resultados que produce

Funcionamiento II



Entorno



-Usuario realiza la petición de fichero.html a través de un explorador de internet.

-La petición se realiza a un servidor web a partir de una dirección determinada.

**Ejemplo: www.direcciónderterminada/fichero.html*

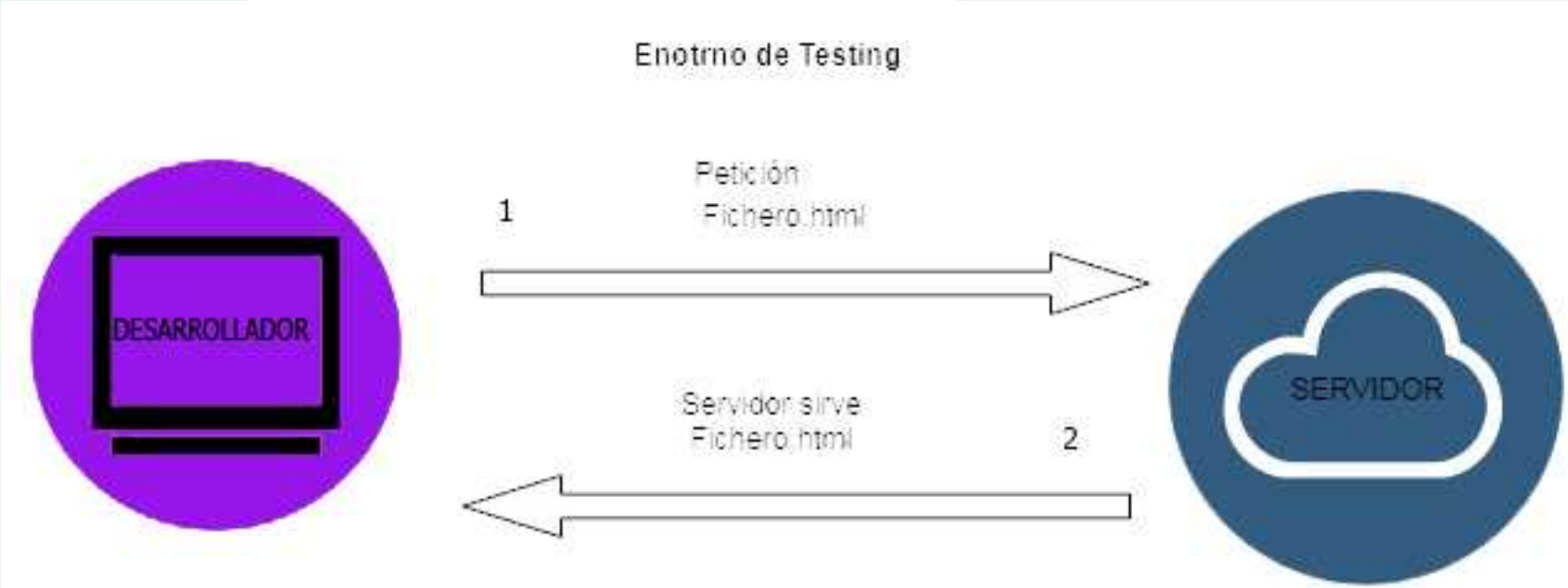
Entorno

-Usuario realiza la petición de fichero.html a través de un explorador de Internet.

-La petición se realiza a un servidor web utilizando la dirección localhost.

**Ejemplo: localhost/fichero.html*

Entorno



-Usuario realiza la petición de fichero.html a través de un explorador de internet.

-La petición se realiza a un servidor web a partir de una dirección determinada.

-El servidor debe ofrecernos un entorno lo más idéntico posible al entorno de producción .

**Ejemplo: www.direcciónderterminada/fichero.html*

Instalación Infraestructura

- **EASY PHP Devserver:**
  - **http://www.easyphp.org/**

  – **Contiene**:

  -Apache: *Servidor web HTTP.*

  -Php: *Procesador de hipertexto.*

  -MySQL: *Sistema de gestión de bases de datos relacional.*

  -PhpMyAdmin: *Gestor mysql escrito en PHP.*

Instalación Infraestructura

XAMPP:

https://www.apachefriends.org/index.html

Contiene:

-Apache: Servidor web HTTP.

-Php: Procesador de hipertexto.

-MySQL/MariaDB: Sistema de gestión de bases de datos relacional.

-PhpMyAdmin: Gestor mysql escrito en PHP.

Guía instalación

Instalación Infraestructura

- **Primera prueba de php7** crear archivo con extensión .php y contenido:

```
<?php
    phpinfo(); //Función que nos devuelve información sobre PHP

?>
```

Sintaxis básica

- **PHP** es un lenguaje sensible a las mayúsculas.

- El código ha de ir siempre entre los siguientes tags:

  <?php …codigo…. **?>** //Recomendado

  <? ….codigo… **?>** //Posibles problemas

  <?= …codigo…**?>** //Equivale a <?php echo codigo; ?>

  /* comentario de varias
  Lineas */
  //Comentario de una linea

  Las instrucciones se separan con un **;**

Sintaxis básica

- Para imprimir: **echo y print**

  echo: muestra una o más cadenas
  echo cadena1 [, cadena2…];

  echo “Hola mundo”;
  echo “Hola “, “mundo”;

  print: muestra una cadena
  print cadena;

  print “Hola mundo”;
  print “ Hola “ . “mundo”;

Sintaxis básica

- Con PHP se puede generar código HTML:
  - Código PHP

```
print("<p>Párrafo 1</p>\n");
print("<p>Párrafo 2</p>\n");
```

  - Código HTML resultante:

```
<p>Párrafo 1</p>
<p>Párrafo 2</p>
```

- Importante el uso de \n para generar código HTML legible

EJERCICIO 1

Ejemplo, probar el siguiente código:

```
<html>
    <head>
        <title>Mi primer programa en PHP</title>
    </head>

    <body>
        <?php
            print("<p>Hola mundo</p>");
        ?>
    </body>
</html>
```

EJERCICIO 1

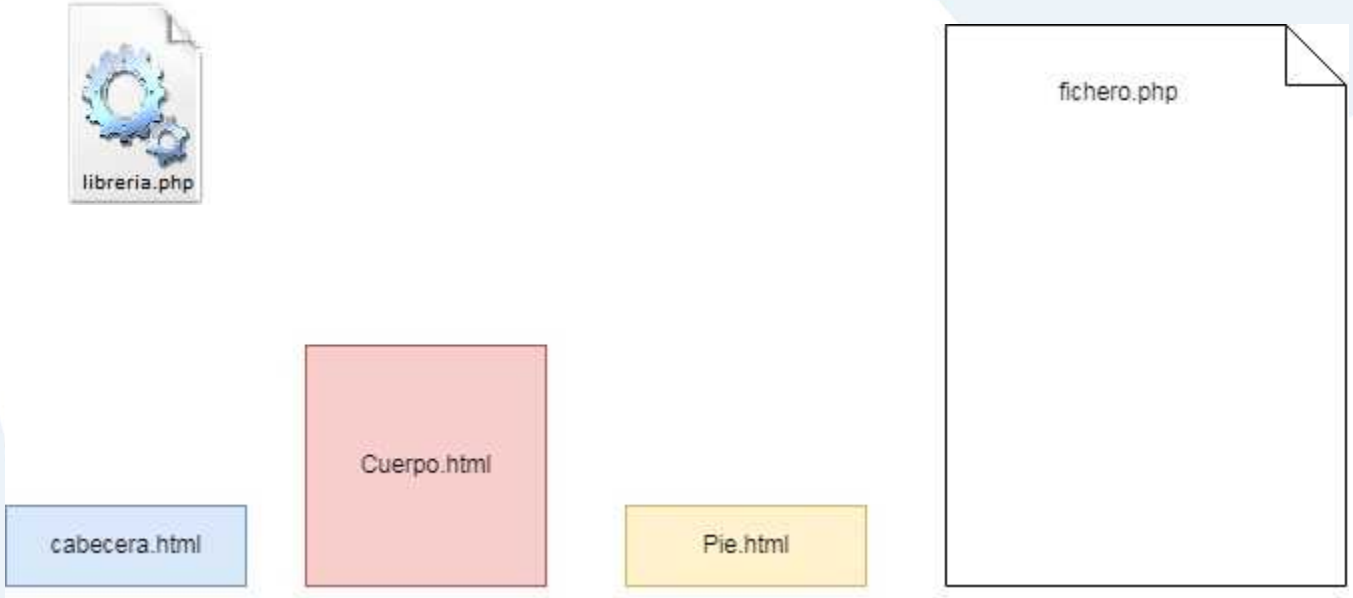
Modifica el hola mundo anterior y crea un fichero php que contenga párrafos creados en html y párrafos creados en php. Por ejemplo el resultado final podría ser este:

Éste es el párrafo 1, escrito desde HTML
Éste es el párrafo 2, escrito desde PHP con print
Éste es el párrafo 3, escrito desde HTML
Éste es el párrafo 4, escrito desde PHP con echo

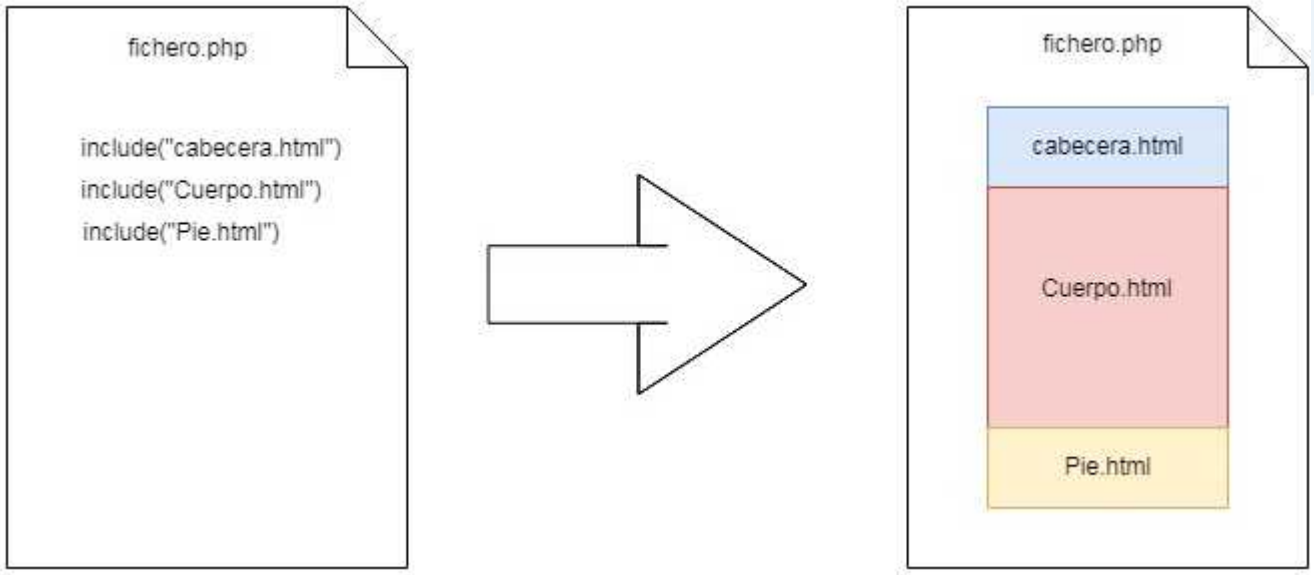
Sintaxis básica

- Inclusión de ficheros externos:
  include() **o** **require()**
- Ambos incluyen y evalúan el fichero especificado.
- Diferencia: en caso de error **include()** produce un warning y **require()** un error fatal.
- Se usará **require()** si al producirse un error debe interrumpirse la carga de la página

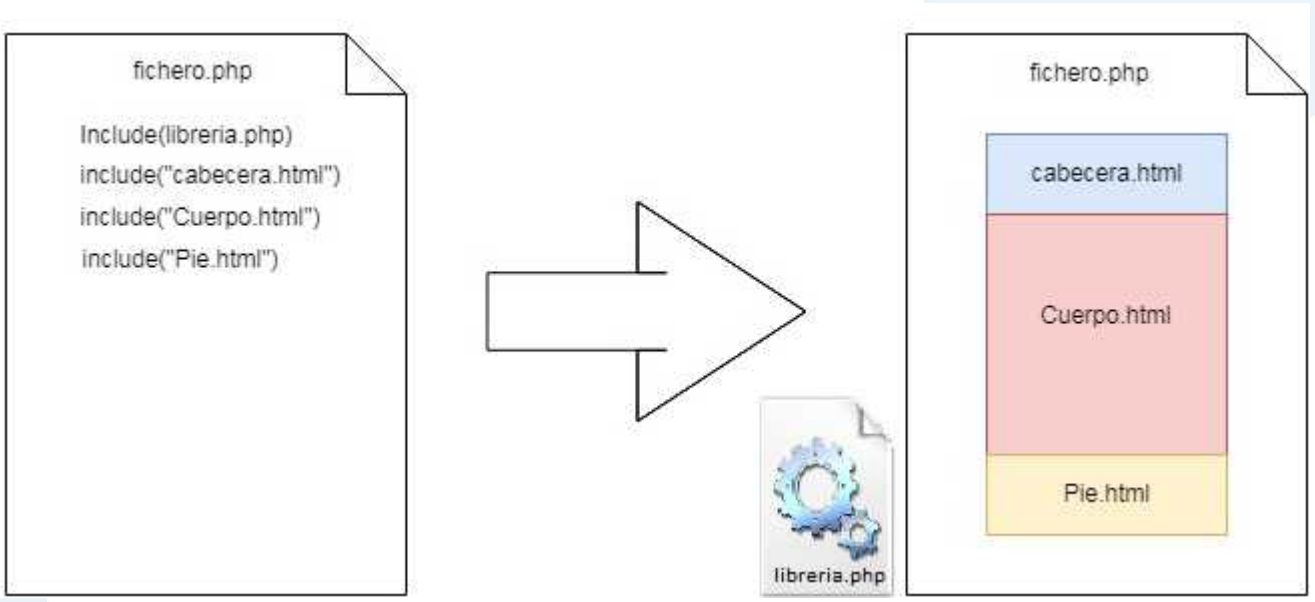
Ejemplo include()



Ejemplo include()



Ejemplo include()



Ventajas include()

- *Comprensión.*
- *Organización.*
- *Edición.*
- *Re-utilización.*
- *Trabajo en equipo/Modular.*
- *Permite usar Código de terceros.*

Sintaxis básica

```
<html>
  <head>
        <title>Título</title>
        <?php
              require(“functions.php");  // incluimos un fichero php con funciones

              ?>
  </head>
  <body>
        <?php
             include("cabecera.html"); // incluimos un fichero HTML
        ?>
        // Código HTML + PHP
        . . .
        <?php
             include("pie.html");  // incluimos un fichero HTML
        ?>
  </body>
</html>
```

Variables

- PHP soporta diferentes **tipos de datos primitivos:**
  - –Tipos escalares: boolean, integer, double, string
  - –Tipos compuestos: array
  - –Tipos especiales: NULL

- Funciones de interés:
  - –La función gettype() devuelve el tipo de una variable
  - –Las funciones is_*type* comprueban si una variable es de un tipo dado:

    is_array(), is_bool(), is_float(), is_integer(), is_null(), is_numeric(), is_string()

- La función var_dump() muestra el tipo y el valor de una variable.

Variables

- El tipo de una variable no se suele especificar. Se decide en tiempo de ejecución en función del contexto y puede variar.

- Las variables se definen **siempre** con el símbolo $ delante:

```
$a = 1;
$b= “cadena”;
```

- Empiezan por letra o _, nunca por número.

- Ámbito de las variables: son globales al fichero salvo en las funciones.

- En las funciones tienen solo un ámbito local.

Variables

- Tipo **integer: números enteros**
  Ej: 27, -5, 0

- Tipo **double: números reales**
  Ej:1.234, -5.33

- Tipo **boolean: lógico**
  –Valores: *true, false(insensibles a las mayúsculas)*
  –El 0 y la cadena vacía tienen valor *false*

Variables

Tipo **string**:

Las cadenas se encierran entre comillas simples o dobles:

- ‘simples’: admite los caracteres de escape \’ (para que no interprete la comilla simple) y \\(para que no tome como escape y interprete la barra). Las variables **NO se interpretan.**

```
// Resultado: Arnold una vez dijo: "I'll be back"
echo 'Arnold una vez dijo: "I\'ll be back" ';
```

- “dobles”: admite más caracteres de escape, como \n, \r, \t, \\, \$, \”. Los nombres de variables **SÍ se interpretan**

- Acceso a un carácter de la cadena, parecido a un array:

```
$inicial= $nombre{0};
```

Variables

•Ejemplos:

```
$a= 9;
print ‘a vale $a’; // muestra a vale $a
print “a vale $a”;// muestra a vale 9

print “<img src=‘logo.gif’>”; // muestra <img src=‘logo.gif’>
print “<img src=\”logo.gif\”>”; // muestra <img src=“logo.gif”>
```

Variables

Arrays.

Para definir un array, se utiliza…

-El carácter ‘(‘ para delimitar su comienzo
-El carácter ‘)’ para delimitar su final.
-El carácter ‘, ‘ para separar sus elementos.

*$nombre_array = array **(**valor1**,** valor2**,** …**,** valorN**)**;*

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

Cada elemento se accede indicando su posición dentro del array…

$diaSeleccionado = $dias[0]; // diaSeleccionado = "Lunes"
$otroDia = dias[5]; // otroDia = "Sábado“

* las posiciones de los elementos empiezan a contarse en el 0 y no en el 1.

Arrays

Clave=>valor

La clave puede ser un entero no negativo o una cadena, el valor contenido puede ser cualquier otro tipo válido en PHP, incluyendo otro array.

```
$color = array( ‘rojo’=>101, ‘verde’=>51, ‘azul’=>255);
```

Cada elemento se accede indicando su posición dentro del array.

```
$colorSeleccionado= $ color [‘rojo’]; //colorSeleccionado = 101
```

Variables variables

A veces es conveniente tener nombres de variables variables. Dicho de otro modo, son nombres de variables que se pueden definir y usar dinámicamente.

Ejemplo:

```
$Hello = "Geeks for Geeks"
$var = "Hello"
echo $var   //mostrará Hello
echo $$var //mostrará Geeks for Geeks
```

ya que muestra el contenido de la variable $Hello contenida dentro de la variable $var

Ejercicio 2

Ejemplo de variables variables: página web multilenguaje

```
<?php
        $mensaje_es="Hola";
        $mensaje_en="Hello";
        $idioma= "es";   //variable que decide el idioma de la pagina

        $mensaje= "mensaje_" . $idioma;
        print $$mensaje; //lo equivalente a print $mensaje_es
?>
```

- El resultado del código anterior será “Hola”

Constantes

- Definición de constantes:

```
define (“CONSTANTE”, “hola”);
print CONSTANTE;
```

- No llevan $ delante

- Sólo se pueden definir constantes de los tipos boolean, integer, double y string

Operadores

Asignación

```
$numero1 = 3;
$numero2 = 4;

5 = $numero1;  /* Error, la asignación siempre se realiza a una variable,por lo que en la izquierda no se puede indicar un número */

$numero1 = 5; // Ahora, la variable numero1 vale 5
$numero1 = $numero2; // Ahora, la variable numero1 vale 4
```

Operadores

Incremento y decremento

Estos dos operadores solamente son válidos para las variables numéricas

Pre-incremento: ++$a

Incrementa $a en uno, devuelve el valor de $a incrementado

Post-incremento: $a++

Incrementa $a en uno, devuelve el valor de $a.

Operadores

```
<?php
echo "<h3>Postincrement</h3>";
    $a = 5;
    echo “Deberia ser 5: " . $a++ . "<br />\n";
    echo " Deberia  ser 6: " . $a . "<br />\n";

    echo "<h3>Preincrement</h3>";
    $a = 5;
    echo “Deberia ser 6: " . ++$a . "<br />\n";
    echo “Deberia ser 6: " . $a . "<br />\n";


?>
```

Operadores

Pre-decremento: --$a

Decrementa $a en uno, devuelve el valor de $a decrementado

Post-decremento: $a--

Decrementa $a en uno, devuelve el valor de $a.

Operadores

```
<?php
    echo "<h3>Postdecrement</h3>";
    $a = 5;
    echo “Deberia ser 5: " . $a-- . "<br />\n";
    echo “Deberia ser 4: " . $a . "<br />\n";

    echo "<h3>Predecrement</h3>";
    $a = 5;
    echo “Deberia ser 4: " . --$a . "<br />\n";
    echo “Deberia ser 4: " . $a . "<br />\n";
?>
```

Operadores

Matemáticos

Php permite realizar manipulaciones matemáticas sobre el valor de las variables numéricas. Los operadores definidos son: suma (+), resta (-), multiplicación (*), división (/) y modulo (%).

```
$numero1 = 10;
$numero2 = 5;

$resultado = $numero1 / $numero2; // $resultado = 2
$resultado = 3 + $numero1; // $resultado = 13
$resultado = $numero2 – 4; // $resultado = 1
$resultado = $numero1 * $numero 2; // $resultado = 50
$resultado = $numero1 % $numero2; // $resultado = 0

$numero1 = 9;
$numero2 = 5;
$resultado = $numero1 % $numero2; // $resultado = 4
```

Operadores

Matemáticos

Los operadores matemáticos también se pueden combinar con el operador de asignación para abreviar su notación.

```
$numero1 = 5;

$numero1 += 3; // numero1 = numero1 + 3 = 8
$numero1 -= 1; // numero1 = numero1 - 1 = 4
$numero1 *= 2; // numero1 = numero1 * 2 = 10
$numero1 /= 5; // numero1 = numero1 / 5 = 1
$numero1 %= 4; // numero1 = numero1 % 4 = 1
```

OPERADORES

- Lógicos

  El resultado de cualquier operación que utilice operadores lógicos siempre es un valor lógico o *booleano*.

  Posibles valores: *1*==T*rue , 0==False.*

Negación

! Se utiliza para obtener el valor contrario al valor de la variable.

```
$visible = true;
echo !$visible;  // Muestra “False" y no “True"
```

Operadores

Relacionales

Los operadores relacionales son idénticos a los que definen las matemáticas: mayor que (>), menor que (<), mayor o igual (>=), menor o igual (<=), igual que (==) y distinto de (!=). El resultado de todos estos operadores siempre es un valor booleano.

```
$numero1 = 3;
$numero2 = 5;
$resultado = $numero1 > $numero2; // $resultado = false
$resultado = $numero1 < $numero2; // $resultado = true

$numero1 = 5;
$numero2 = 5;
$resultado = $numero1 >= $numero2; // $resultado = true
$resultado = $numero1 <= $numero2; // $resultado = true
$resultado = $numero1 == $numero2; // $resultado = true
$resultado = $numero1 != $numero2; // $resultado = false
```

OPERADORES

Negación

variable	!variable
true	false
false	true

- Si la variable contiene un número, se transforma en false si vale 0 y en true para cualquier otro número (positivo o negativo, decimal o entero).
- Si la variable contiene una cadena de texto, se transforma en false si la cadena es vacía ("") y en true en cualquier otro caso.

```
$cantidad = 0;
$vacio = !$cantidad; // vacio = true

$cantidad = 2;
$vacio = !$cantidad; // vacio = false

$mensaje = "";
$mensajeVacio = !$mensaje; // mensajeVacio = true

$mensaje = "Bienvenido";
$mensajeVacio = !$mensaje; // mensajeVacio = false
```

OPERADORES

AND

variable1	variable2	variable1 && variable2
true	true	true
true	false	false
false	true	false
false	false	false

La operación lógica AND (**&&**) obtiene su resultado combinando dos valores booleanos

```
$valor1 = true;
$valor2 = false;
$resultado = $valor1 && $valor2; // resultado = false

$valor1 = true;
$valor2 = true;
$resultado = $valor1 && $valor2; // resultado = true
```

OPERADORES

OR

variable1	variable2	variable1 || variable2
true	true	true
true	false	true
false	true	true
false	false	false

La operación lógica OR (||) obtiene su resultado combinando dos valores booleanos.

```
$valor1 = true;
$valor2 = false;
$resultado = $valor1 || $valor2; // resultado = true

$valor1 = false;
$valor2 = false;
$resultado = $valor1 || $valor2; // resultado = false
```

CONTROL DE FLUJO

- Estructuras de control de flujo

Son instrucciones del tipo *"si se cumple esta condición, hazlo; si no se cumple, haz esto otro"*. También existen instrucciones del tipo *"repite esto mientras se cumpla esta condición"*.

Estructura if

```
if(condicion) {
...
}
```

Ejemplo:

```
$mostrarMensaje = true;

if($mostrarMensaje == true) {
   echo "Hola Mundo";
}
```

```
if($mostrarMensaje) {
    echo “Hola Mundo";
}
```

CONTROL DE FLUJO

Estructura if...else

```
if(condicion) {
...
}

else {
...
}
```

```
<?php
    if($sexo== 'M')
        $saludo= "Bienvenido, ";
    else
        $saludo= "Bienvenida, ";

    $saludo= $saludo. $nombre;
    print($saludo);
    ?>
```

La estructura if...else se puede encadenar para realizar varias comprobaciones seguidas.

Si solo hay una sentencia ; en cada if...else, puede escribirse sin usar { }

OPERADORES/ IF-ELSE-ELSEIF

*ejemplo:

```
<?php
    $x = True; // asigna el valor TRUE a $i
    $y = False; // asigna el valor False a $j
    $i = 0; // asigna el valor 0 a $i

    if($x==$i){
        echo '$i = $x = '.$i.".<br>";
    }
    elseif ($y==$i) {
        echo '$i = $y = '.$i.".<br>";
    }
?>
```

CONTROL DE FLUJO

Estructura switch

```
switch (expression) {
    case (value1) :
    // Code for value1
    break;
    case (value2) :
    // Code for value2
    break;
    ...
    default :
    // Code for other values
}
```

OPERADORES/ SWITCH CASE

*ejemplo:

```
<?php
    $x = True; // asigna el valor TRUE a $i
    $y = False; // asigna el valor False a $j
    $i = 0; // asigna el valor 0 a $i

    switch ($i) {
        case ($x) :
            echo '$i = $x = '.$i."<br>";

        break;
        case ($y) :
            echo '$i = $y = '.$i."<br>";
        break;
    }

?>
```

CONTROL DE FLUJO

- Ejemplo:

```
switch($extension)
{
        case ("PDF"):
            $tipo= "Documento Adobe PDF";
        break;
        case ("TXT"):
            $tipo= "Documento de texto";
        break;
        case ("HTML"):
        case ("HTM"):
            $tipo= "Documento HTML";
        break;
        default:
            $tipo= "Archivo " . $extension;
}
print  ($tipo);
```

Ejercicio

Crear una página que muestre un mensaje de bienvenida que dependa de la hora actual, de la siguiente manera:

Si la hora se encuentra entre las 8 y las 13, mostrará ‘Buenos días’
Si la hora se encuentra entre las 14 y las 20, mostrará ‘Buenas tardes’
Si la hora se encuentra entre las 21 y las 7, mostrará ‘Buenas noches’

Podemos obtener la hora actual de la siguiente manera:

```
$hora = date ("H");
```

CONTROL DE FLUJO

Estructura while

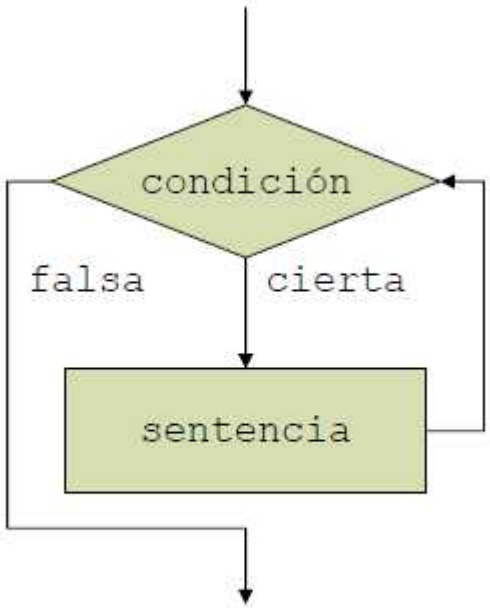
La estructura while permite crear bucles que se ejecutan ninguna o más veces, dependiendo de la condición indicada. Su definición formal es:

```
while(condicion) {
...
}
```

Ejemplo:

```
$resultado = 0;
$numero = 100;
$i = 0;

while($i <= $numero) {
    $resultado += i;
    $i++;
}
```



CONTROL DE FLUJO

Estructura do...while

El bucle de tipo do...while es muy similar al bucle while, salvo que en este caso **siempre** se ejecutan las instrucciones del bucle al menos la primera vez.

```
do{
...
} while(condicion)
```

Ejemplo:

```
$resultado = 1;
$numero = 5;

do {
    $resultado *= $numero; // resultado = resultado * numero
    $numero--;
} while($numero > 0);
```

CONTROL DE FLUJO

Estructura for

La estructura for permite realizar repeticiones (también llamadas bucles) de una forma muy sencilla.

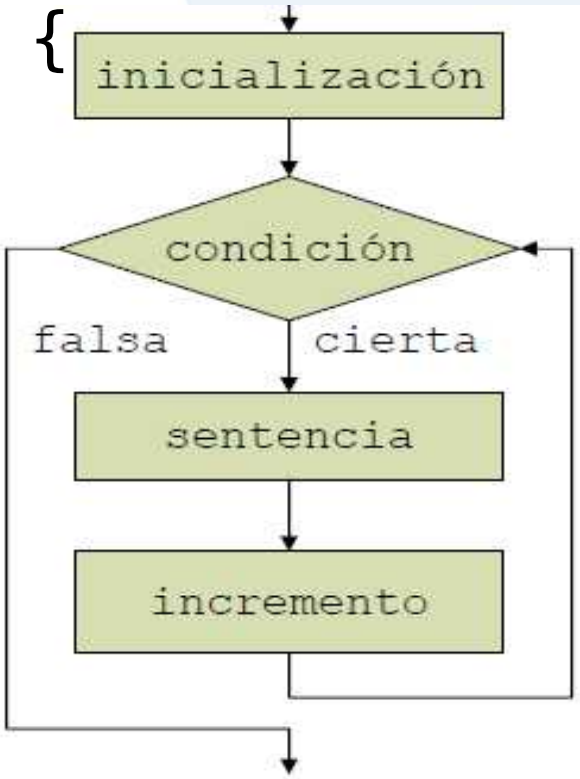
```
for(inicializacion; condicion; actualizacion) {
...
}
```

- La "inicialización" es la zona en la que se establece los valores iniciales de las variables que controlan la repetición.
- La "condición" es el único elemento que decide si continua o se detiene la repetición.
- La "actualización" es el nuevo valor que se asigna después de cada repetición a las variables que controlan la repetición.

CONTROL DE FLUJO

Estructura for

```
$mensaje = "Hola, estoy dentro de un bucle";
for($i = 0; $i < 5; $i++) {
    echo $mensaje;
}
```



CONTROL DE FLUJO

```
<?php
    print("<UL>\n");

 for($i=1; $i<=5; $i++)
     print("<LI>Elemento $i</LI>\n");

    print("</UL>\n");
?>
```

EJERCICIO 3

Ejercicio:

programa que calcula una tabla de multiplicar.

- Usar “print” o “echo” para que PHP devuelva código HTML.
- Usar Tablas i/o Listas.
- Usar Bucles for i/o while para facilitar la tarea.

CONTROL DE FLUJO

Estructura foreach

Parecido al for…in de javascript, permite iterar sobre arrays.

```
foreach($array as $value) {   ...    }
foreach($array as $clave=>$valor) {   ...    }
```

Ejemplos:

```
$color = array( ‘rojo’=>101, ‘verde’=>51, ‘azul’=>255);

foreach($color  as $valor)
        print “Valor: $valor<BR>\n”;
```

Muestra por pantalla:

??

CONTROL DE FLUJO

```
$color = array( ‘rojo’=>101, ‘verde’=>51, ‘azul’=>255);

foreach($color as $clave=> $valor)
    print“Clave: $clave; Valor: $valor<BR>\n”;
```

Muestra por pantalla:
??

EJERCICIO 4

Ejercicio:

programa que, de 10 elementos enteros, los almacena en un array y encuentra:

- El sumatorio de estos elementos.
- El máximo.
- El mínimo.

Funciones

Funciones

- Una **función** es un trozo de código que ejecuta una tarea determinada. Esta tarea está constituida por una serie de instrucciones.

- Las funciones evitan al programador tener que repetir el mismo código varias veces y hacen más inteligible el funcionamiento de un programa.

Funciones

Las funciones se definen mediante la palabra reservada function, seguida del nombre de la función.

```
function nombre_funcion() {
...
}
```

El nombre de la función se utiliza para *llamar o invocar* a esa función cuando sea necesario. los símbolos { y } se utilizan para encerrar todas las instrucciones que pertenecen a la función

Funciones

La mayoría de funciones deben acceder al valor de algunas variables para producir sus resultados. Las variables que necesitan las funciones se llaman *argumentos*.

Los argumentos se indican dentro de los paréntesis que van detrás del nombre de la función y se separan con una coma (,).

```
function nombre_funcion(arg1, arg2, …, argn) {
…
}
```

Al invocar la función, el número de argumentos que se pasa a la función debería ser el mismo que el número de argumentos que ha indicado la función. El orden de los argumentos es fundamental.

Funciones

- Se pueden añadir valores a los parámetros por defecto:

```
function muestranombre($titulo= "Sr.")
{
        print "Estimado $titulo:\n";
}
muestranombre();
muestranombre("Prof.");
```

Salida:

Estimado Sr.:
Estimado Prof.:

Funciones

- Los argumentos con valores por defecto deben ser siempre los últimos:

```
function muestranombre($nombre, $titulo="Sr.")
{
    print "Estimado $titulo$nombre:\n";
}
muestranombre(“Fernández”);
muestranombre(“Fernández”, "Prof.");
```

Salida:

Estimado Sr. Fernández:
Estimado Prof. Fernández:

Funciones

Las funciones no solamente puede recibir variables y datos, sino que también pueden devolver los valores que han calculado. Para devolver valores dentro de una función, se utiliza la palabra reservada **return**.

```
function calculaPrecioTotal($precio) {
   $impuestos = 1.16;
   $gastosEnvio = 10;
   $precioTotal = ( $precio * $impuestos ) + $gastosEnvio;
   return $precioTotal;
}
// El valor devuelto por la función, se guarda en una variable
$precioTotal = calculaPrecioTotal(23.34);

// Seguir trabajando con la variable "precioTotal“
```

Todas las instrucciones que se incluyen después de un return se ignoran y por ese motivo la instrucción return suele ser la última de la mayoría de funciones.

Funciones

Return permite devolver un array:

- útil para devolver varios valores

```
function calculaPrecioTotal($precio) {
   $impuestos = 1.16;
   $gastosEnvio = 10;
   $precioParcial =  $precio * $impuestos ;
   $precioTotal =$PrecioParcial +  $gastosEnvio;
   return array($precioTotal, $precioParcial);
}
// El valor devuelto por la función, se guarda en una variable
$precioTotal = calculaPrecioTotal(23.34);

// Seguir trabajando con la variable "precioTotal[0] “ y "precioTotal[1]
```

- RECORDAR: podéis usar clave => valor para facilitar el trabajo!!

EJERCICIO 5

- **Ejemplo**:

```
function suma ($x, $y)
{
   ?????
}
$a=1;
$b=2;
$c=suma($a, $b);
print $c;
```

Lo mismo para restar, multiplicar y dividir.

Funciones

Ámbito de las variables

Variable local solamente está definida dentro de la función

Ejemplo:

```
function creaMensaje() {
   $mensaje = “Mensaje de prueba”;
}
creaMensaje();
Echo $mensaje; //error
```

Cualquier instrucción que se encuentre dentro de la función puede hacer uso de esa variable, pero todas las instrucciones que se encuentren en otras funciones o fuera de cualquier función no tendrán definida la variable mensaje.

Funciones

Ámbito de las variables

Variable NO global, está definida en cualquier punto del programa (incluso dentro de cualquier función). Para que la función sepa cual es, necesita recibirla POR PARAMETRO

```
$mensaje = “Mensaje de prueba”;
function muestraMensaje() {
    echo $mensaje; //no tiene valor!
}
muestraMensaje();
```

Funciones

Ámbito de las variables

Variable global, está definida en cualquier punto del programa.

```
$mensaje = “Mensaje de prueba”;
function muestraMensaje() {
    global $mensaje;
    echo $mensaje; // Mensaje de prueba
}
muestraMensaje();
```

Bibliotecas de funciones

- Como en Javascript, existen muchas bibliotecas de funciones en PHP

  - **Funciones matemáticas**
  - **Funciones de manipulación de cadenas**
  - **Funciones de arrays**
  - **Funciones de fecha y hora**
  - **Funciones de cifrado**
  - Funciones de ficheros
  - Funciones de bases de datos
  - Funciones de red

Funciones

Funciones matemáticas:

round(float, precision), devuelve el número original con tantos decimales como los indicados por el parámetro precision y realiza los redondeos necesarios.

```
$numero1 = 4564.34567;
round($numero); // 4564
round($numero, 2); // 4564.35
echo round(5.045, 2);    // 5.05

echo round(5.055, 2);    // 5.06
```

Funciones

NaN, (del inglés, *"Not a Number"*) se emplea para indicar un valor numérico no definido, si el valor no es un número

```
$numero1 = 0;
$numero2 = 0;
```

$numero1/ $numero2 tiene como v*alor NaN*

is_nan(valor), permite proteger a la aplicación de posibles valores numéricos no definidos: devuelve TRUE si valor no es un número

http://php.net/manual/es/ref.math.php

Funciones

Funciones de manipulación de cadenas

explode(valor , string)

Divide una cadena en subcadenas y las almacena en un array

Array = explode(string separator, stringstring[, intlimit])

rtrim(string), ltrim(string), trim(string)

Eliminan caracteres a la derecha, a la izquierda o por ambos lados de una cadena

string = rtrim(stringstr[, stringcharlist])

strstr(string pajar, string aguja)

Busca la primera ocurrencia de una subcadena

Funciones

implode(valor , string)

Crea una cadena con los elementos de un array

strtolower(string) / strtoupper(string)

Convierte una cadena a minúscula / mayúscula

strcmp(string) / strcasecmp(string)

Compara dos cadenas con/sin distinción de mayúsculas

strlen(string)

Calcula la longitud de una cadena

Funciones

substr(string, start, [length])

Devuelve parte de una cadena.

- La cadena string empieza siempre por el carácter 0 (a tener en cuenta en start).
- La longitud empieza a contar a partir de 1, y no 0.
- Si la longitud es negativa, ese será el numero de caracteres a NO tener en cuenta en el final de la cadena.

```
echo substr('abcdef', 1);     // bcdef
echo substr('abcdef', 1, 3);  // bcd
echo substr('abcdef', 0, 4);  // abcd

echo substr("abcdef", 0, -1);  // "abcde"
echo substr("abcdef", 2, -1);  //  "cde"
echo substr("abcdef", 4, -4);  //  false
```

Funciones

eval(string)

Evalúa un string como si fuese código PHP.

Por ejemplo:

```
$cadena = “echo (2+2)”;

eval($cadena);

//mostraría por pantalla: 4
```

Funciones

str_replace(string antes, string despues, string)

Busca y reemplaza el valor ANTES por el DESPUES en un string

Por ejemplo:

str_replace(‘ ‘, ‘’, $cadena);

//nos elimina los espacios de una cadena

str_split(string);

Separa todos los elementos de una cadena (letras) y devuelve un array con estas.

http://www.php.net/manual/es/ref.strings.php

Funciones

Funciones de arrays

array_count_values(array)

Calcula la frecuencia de cada uno de los elementos de un array

array_search(aguja, array pajar)

Busca un elemento en un array

count(array)

Cuenta los elementos de un array

sort(array), rsort(array)

Ordena y reindexa un array(r=decreciente)

ksort(array), krsort(array)

Ordena por claves un array(r=decreciente)

Funciones

array_push(array, elemento)

Añade el elemento en un array

Es lo equivalente a hacer $array[] = $elemento;

array_pop(array)

Extrae el último elemento en un array (y lo devuelve)

unset($array[clave])

elimina el elemento que se encuentre en la posición de la clave en un array

http://php.net/manual/es/ref.array.php

EJERCICIO 6

- **Función que dado un string**:

  - Devuelve la letra que más aparece
  - Devuelve la letra que menos aparece

Funciones

Funciones de fecha y hora

date()

Formatea una fecha según un formato dado

- Ejemplo:

```
$fecha= date ("j/n/Y H:i");
print("$fecha");
```

Resultado:
26/9/2005 17:36

Funciones

strtotime()

Convierte una fecha en un *timestamp de UNIX*

•Ejemplo:

```
$fecha= date ("j/n/Y", strtotime(“5 april 2001"));
print("$fecha");
```

Resultado:

5/4/2001

http://www.php.net/manual/es/ref.datetime.php

EJERCICIO 7

- **Función que dado dos fechas**:

  -Nos diga cual es la mayor de las dos

EJERCICIO 8

- **Función que dada una fecha**

-Si está en formato EEUU que nos la transforme en formato europeo.

- Si está en formato europeo que nos la transforme en formato EEUU.

- Podremos añadir un parámetro de formato de fecha

Formato europeo: dd/mm/AAAA
Formato EEUU: mm/dd/AAAA

EJERCICIO 9

- **Función que dada una fecha**:

  - Se nos devuelva por escrito el día y mes como si fuese una carta formal:

    P.E. de 19/03/2014 pasamos a
    "Miercoles, 19 de Marzo del 2014"

    -Podemos dar por hecho que se recibirá esta fecha en formato europeo.

Funciones

Funciones de cifrado

-No reversibles

- Combinables entre si -> recomendable!!

sha1(valor)

md5(valor)

```
$str = 'apple';

    if (md5($str) === '1f3870be274f6c49b3e31a0c6728957f') {
        echo “¿te gustan las manzanas?";
    }
```

EJERCICIO 10

- **Función que**:

  - Reciba un string
  - Compare si el string dado es igual a la contraseña almacenada

  - Si lo es: mensaje en verde, todo correcto.
  - Si no lo es: mensaje en rojo, contraseña incorrecta.