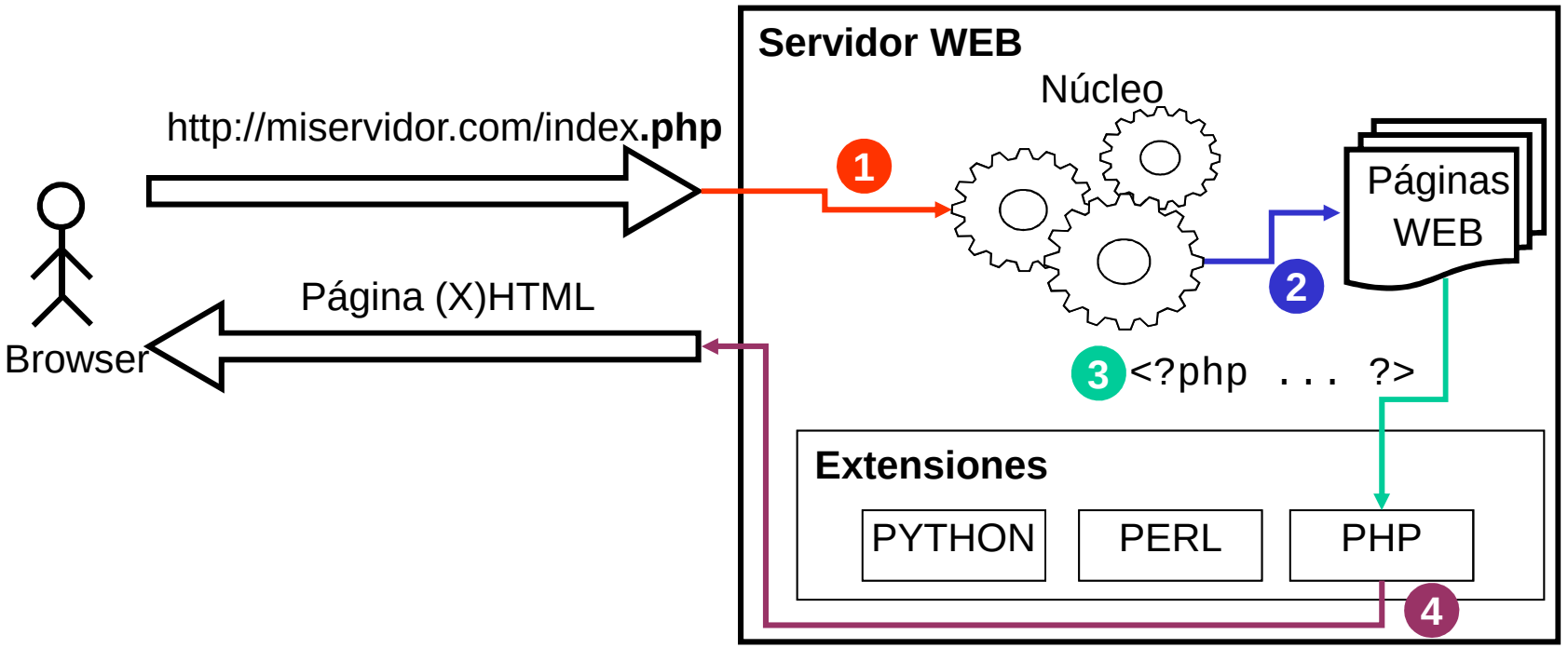
PHP Introducción y sintaxis

Tecnologías Web

¿Qué es PHP?

- PHP: Hypertext Preprocessor
- Es un lenguaje de guiones que se ejecuta en el servidor.



¿Qué se puede hacer con PHP?

- Realizar operaciones sobre ficheros del sevidor
- Obtener datos de un formulario (X)HTML
- Acceder a Bases de Datos
- Gestionar "cookies"
- Gestionar la seguridad de un sitio web (autorización)
- Crear imágenes
- Crear PDF
- Tratamiento de XML
- Comunicación con Java Servlets

Historia de PHP

- 1994. Rasmus Lerdorf crea un parser (compilador) que se encarga de ejecutar macros: tratamiento de formularios y acceso a base de datos.
  - El procesador es llamado PHP/FI
- 1998. Multitud de sitios web usan PHP/FI. Se añaden numerosas funcionalidades y se da soporte a varias plataformas y servidores.
  - Se libera PHP 3.0 como proyecto Open Source.
- Zend Technologies crea PHP 4.0
  - Reescritura completa del núcleo del intérprete
  - Mejora notable en el rendimiento.
  - Nuevas funcionalidades: IMAP, SNMP
  - Capacidades iniciales de Orientación a Objetos
- PHP 5.0
  - Remodelación completa del soporte de Objetos
  - "PHP5: Coming Soon to a Webserver Near You," http:// www. sitepoint. com/article/ 1192/

- Páginas PHP
  - Archivo `.php`
  - El intérprete de PHP intentará ejecutar todas las instrucciones que estén entre los delimitadores de instrucciones PHP.
  - Existen varios tipos de delimitadores de código PHP

`delimitadores.php`

```
<html>
  <head><title>Tipos de delimitadores para páginas PHP</title></head>
  <body>
    <!-- Llamada como instrucción de procesamiento SGML -->
    <?
      echo "<p>Este es el primer tipo de delimitador</p>";
    ?>
    <!-- Llamada como instrucción de procesamiento de XML -->
    <?php
      echo "<p>Este es el segundo tipo de delimitador</p>";
    ?>
    <!-- Llamada desde un editor HTML -->
    <script language="php">
      echo " "<p>Este es el tercer tipo de delimitador</p>";
    </script>
  </body>
</html>
```

Esta es la más común

PHP y (X)HTML (cont.)

- Existe una versión adicional para llamar al intérprete de PHP que se denomina evaluación de expresiones en línea

delimitadoresEnLinea.php

```
<html>
  <head><title>Ejemplo de delimitadores en línea</title></head>
  <body>
    <!-- Expresión en línea -->
    Dos mas dos es: <?= 2 + 2 ?>

    <!– Expresión equivalente -->
    Dos mas dos es: <?php echo(2+2); ?>
  </body>
</html>
```

Instrucciones

- Dentro de los delimitadores de PHP se puede escribir un número cualquiera de instrucciones PHP.
- Las instrucciones PHP de un bloque deben acabar con ";"
  - En la última instrucción de un bloque PHP no hace falta

```
<?php echo(2+2); ?>
<?php echo(2+2) ?>
```

  - Si no se incluye el ";" se generará un error en tiempo de ejecución
- Las instrucciones pueden contener espacios en blanco y saltos de línea

```
<?php
  echo(2+2);
  echo(3+2)
?>
```

```
<?php
echo( 2
      +2
    ); echo(3+2)
?>
```

Comentarios

- PHP permite introducir comentarios de varias maneras

Ejemplos de comentarios

```
<?php
  echo ("Hola Mundo"); // Imprime el mensaje "Hola Mundo"
  echo ("Esto es CSW"); # Imprime el mensaje "Esto es CSW"

  echo (
    6 // euros para la comida
    +
    20 # eros para gasolina
  );

  /* Las sentencias anteriores son un ejemplo de instrucciones
     simples PHP que imprimen por pantalla el valor devuelto
     al evaluar las expresiones que le pasamos como argumentos
  */
?>
```

- Los comentarios PHP sólo son comentarios dentro de los delimitadores, fuera de ellos son tratados como texto.

PHPDoc → http://www.phpdoc.de/
PHPDocumentor → http://www.phpdoc.org/

Tipos

- Tipos primitivos soportados
  - Tipos básicos (tipos escalares)
    - `string`
    - `boolean`
    - `integer`
    - `float` (`double` es el mismo no hay diferencia en tamaño)
  - Tipos compuestos
    - `array`
    - `object`
  - Tipos especiales
    - resource
    - `NULL`

Boolean

- Este tipo fue introducido en PHP 4
- Para especificar un valor booleano se usan las palabra reservadas TRUE, FALSE que son insensibles a mayúsculas y minúsculas

Ejemplos de comentarios

```
<?php
  $foo = TRUE; // Asigna el valor TRUE a la variable $foo
?>
```

Integer

- Podemos especificar enteros en decimal (base 10), en hexadecimal (base 16) y octal (base 8), opcionalmente podemos incluir el signo (+,-)

Ejemplos de literales enteros

```
<?php
  $a = 1234; // Número decimal
  $a = -123; // Número negativo
  $a = 0123; // Número Ocatl ( 83 decimal)
  $a = 0x1A; // Número hexadeciaml ( 26 decimal)
?>
```

Sintaxis

```
decimal     : [1-9][0-9]*
              | 0

hexadecimal : 0[xX][0-9a-fA-F]+

octal       : 0[0-7]+

integer     : [+-]?decimal
              | [+-]?hexadecimal
              | [+-]?octal
```

- El tamaño depende de la plataforma.
  - Normalmente los valores máximos son los valores permitidos para un entero con signo de 32 bits. [-2147483648 … 2147483648]
- Desbordamiento de enteros
  - Si al evaluar una expresión se sobrepasa el valor máximo de un entero será interpretado como un valor `float`

Float

- Los números en punto flotante (`float` ⇔ `double`)

Ejemplos de literales punto flotante

```
<?php
  $a = 1.234;
  $b = 1.2e3;
  $c = 7E-10;
?>
```

Sintaxis

```
LNUM          [0-9]+
DNUM          ([0-9]*[\.]{LNUM}) | ({LNUM}[\.][0-9]*)
EXPONENT_DNUM ( ({LNUM} | {DNUM}) [eE][+-]? {LNUM})
```

- El tamaño depende de la plataforma.
  - Normalmente los valores máximos son ~1.8e308 con una precisión de 14 dígitos.

String

- Las cadenas de caracteres PHP sólo contienen caracteres ASCII
  - No se da soporte nativo a Unicode.
  - Existen funciones de extensión que nos permiten trabajar con Unicode `utf8_encode()`, `utf8_decode()`.
- No hay restricción para el tamaño de las cadenas.
- Podemos especificar cadenas de 3 maneras: cadenas entre comillas simples, cadenas entre comillas dobles y "here documents"
- Acceso/Modificación de los caracteres de la cadena
  - El índice del primer carácter es 0.

Sintaxis de acceso a caracteres de una cadena

```
<?php
  $str = 'Esto es un test.';
  // Sintaxis PHP >=4, Obtenemos el primer caracter de la cadena
  $first = $str{0};

  // Esta sintaxis es obsoleta.
  echo $str[0]
;?>
```

String (cont.)

- Cadena con comillas simples
  - Encerramos una cadena de texto entre `'...'`
  - Para poder usar una comilla simple tenemos que escaparla (`\'`)
  - Si queremos que aparezca una `\` delante de una comilla simple o al final de la cadena tenemos que escaparla (`\\`)
  - Si intentamos escapar otro carácter que no sean ' ó \ la barra \ será también mostrada
  - Las variables y otras secuencias de escape no serán expandidas.

Ejemplos de cadenas

```
<?php
  echo 'esta es una cadena simple';
  echo 'Tambi&eacute;n puede tener saltos de l&iacute;nea embebidos
        en las cadenas de esta forma, ya que
        es v&aacute;lido';
  echo 'this is a simple string';
?>
```

String (cont.)

Escapar \' y \\

```
<?php
  echo 'Esto imprme una comilla simple: \'';
  echo 'Para imprimir una comilla simple
        hay que escaparla de escribiendo: \\\'';
?>
```

Ejemplos de cadenas

```
<?php
  // Imprime: Arnold once said: "I'll be back"
  echo 'Arnold once said: "I\'ll be back"';

  // Imprime : You deleted C:\*.*?
  echo 'You deleted C:\\*.*?';

  // Imprime : You deleted C:\*.*?
  echo 'You deleted C:\*.*?';

 // Imprime: No se expandirá la cadena: \n en una nueva línea
  echo 'No se expandirá la cadena: \n en una nueva línea';

  // Imprime: Las variables $expand $either no se expandirán
  echo 'Las variables $expand $either no se expandirán';
?>
```

String (cont.)

- Cadenas con comillas dobles
  - Podemos escapar más caracteres.

Secuencia de escape	Significado
`\n`	Salto de línea
`\r`	Retorno de carro
`\t`	Tabulador horizontal
`\\`	Barra '\'
`\$`	Signo del dolar
`\"`	Comillas dobles
`\[0-7]{1,3}`	Carácter en notación octal
`\x[0-9A-Fa-f]{1,2}`	Carácter en notación hexadecimal

  - Las variables que usemos dentro de la cadena serán expandidas

String (cont.)

- Cadenas "heredoc"

Syntaxis "heredoc"

```
<<<[identificador][nueva línea]

[identificador]
```

No se puede incluir ningún otro carácter excepto ';'. Esto incluye espacios en blanco → NO SE PUEDE INDENTAR

Ejemplo de cadena "heredoc"

```
<?php
$str = <<<EOD
  Este es un ejemplo de cadena
  de texto que se expande en varias
  líneas usando la sintaxis heredoc.
EOD;
```

- Las cadenas heredoc permiten la inclusión de caracteres escapados anteriores, además también realizan la expansión de variables.

Expansión de variables dentro de cadenas

- Expansión de variables
  - Cuando usamos una cadena entre comillas dobles o una cadena heredoc las variables dentro de ella son expandidas.
- Sintaxis Simple
  - Cuando el intérprete encuentra el signo '$' tomará todos los caracteres válidos (válidos para un identificador) que le sigan, como nombre de variable.
  - Puedes usar `'{'` y `'}'` para que no haya confusión con el nombre de la variable

Ejemplo 1: Expansión de variables, sintaxis simple

```
<?php
  $beer = 'Heineken';
  // funciona, "'" es un caracter no válido para nombres de variable
  echo "$beer's taste is great";
  // no funciona, 's' es un caracter válido para nombres de variable
  echo "He drank some $beers";
  echo "He drank some ${beer}s"; // funciona
  echo "He drank some {$beer}s"; // funciona
?>
```

Expansión de variables dentro de cadenas (cont.)

- Sintaxis avanzada (sintaxis compleja)
  - Se denomina compleja porque podemos incluir una expresión compleja.
  - La sintaxis avanzada es reconocida por el intérprete si detrás de `'{'` le sigue el carácter `'$'`.
    - Si no le sigue justo después el intérprete lo tomará como una cadena.

Ejemplo 2: Expansión de variables, sintaxis avanzada

```
<?php
  // Habilitamos el que se muestren todos los errores posibles
  error_reporting(E_ALL);
  $great = 'fantastic';

  // No funciona, salida: This is { fantastic}
  echo "This is { $great}";

  // Funciona
  echo "This square is {$square->width}00 centimeters broad.";

  // Funciona
  echo "This works: {$arr[4][3]}";
?>
```

Funciones para tratar con tipos

`var_dump(<variable>)`

- Imprime información (tipo, valor) de la variable `'<variable>'`.

`gettype(<variable>)`

- Devuelve el tipo de `'<variable>'` en forma de cadena
- Esta función debe ser usada para depurar, para realizar comparaciones entre tipos usar las funciones `is_*()`.
- Posibles valores:
  - `"boolean"` (desde PHP 4)
  - `"integer"`
  - `"double"` (por razones históricas se devuelve en vez de "float")
  - `"string"`
  - `"array"`
  - `"object"`
  - `"resource"` (since PHP 4)
  - `"NULL"` (since PHP 4)
  - `"unknown type"`

Funciones para tratar con tipos

`settype(<var>, <tipo>)`

- Fuerza a la variable `'<var>'` a tener el tipo `'<tipo>'`.
- Los valores permitidos para `'<tipo>'` son los mismos que para la función `gettype()`.

Conversiones de tipos

- PHP es un lenguaje débilmente tipado.
  - La definición de una variable no requiere que se declare el tipo de la misma.
  - El tipo de variable se infiere por el valor que tiene asignado.
- Cuando usamos una función / operador que espera un valor de un tipo determinado y le suministramos otro, PHP realiza por nosotros una conversión automática de tipos.
- También podemos hacer una conversión explícita (casting) del valor de una variable.
  - Cast permitidos: `(int)`, `(integer)`, `(string)`, `(double)`, `(bool)`, `(boolean)`, `(array)`, `(object)`.

Ejemplos de variables

```
<?php
  $foo = 10;               // $foo es un entero
  $bar = (boolean) $foo;   // $bar es boolean
?>
```

Conversión de tipos (cont.)

- Cuando se convierte una expresión a `boolean`, los siguientes valores son tomados como falsos:
  - 0 (entero)
  - 0.0 (float)
  - "" (cadena vacía), "0"
  - Array de 0 elementos
  - NULL y variables sin inicializar
  - Objetos (PHP >=4) sin ningún atributo miembro
  - Cualquier otro valor es considerado `TRUE`
- Cuando se convierte una expresión a `integer`
  - `boolean` a `integer` → `FALSE` es considerado 0, `TRUE` es considerado 1
  - `float` a `integer` → Se redondea el valor hacia abajo. Si el valor sobrepasa el valor máximo el resultado es indefinido
  - `string` a `integer` → Ver la conversión de cadenas.
  - Para otro valor, primero se convierte a `boolean` y después se realiza la conversión

Conversión de tipos (cont.)

- Cuando se convierte una expresión a `string`.
  - `boolean` a `string` → `FALSE` es considerado "" (cadena vacía), `TRUE` es considerado "1".
  - `integer`/`float` a `string` → Se convierte a una cadena que representa sus dígitos y exponente en el caso de `float`.
  - `array` a `string` → Se convierte a la cadena "Array".
  - `object` a `string` → Se convierte a la cadena "Object".
  - `NULL` → Se convierte a "" (cadena vacía).

Variables

- PHP identifica las variables usando el signo '$'
  - Los nombres de variable son sensibles entre mayúsculas y minúsculas.
  - Sintaxis: $<identificador>
  - El identificador de la variable tiene que comenzar por letra o '_'. El resto de caracteres pueden ser letras, números o '_'.
- Hasta PHP3, las variables sólo podían ser asignadas por valor
- A partir de PHP4, las variables también pueden ser asignadas por referencia.
  - Al asignar una variable por referencia, un cambio en la referencia también se aplica sobre la variable original.

Ejemplos de variables

```
$foo = 'Bob';               // Asignamos 'Bob' a $foo
$bar = &$foo;               // Referenciamos $foo vía $bar.
$bar = "My name is $bar";   // Modificamos $bar...
echo $bar;
echo $foo;                  // $foo también se modifica.
```

Variables predefinidas

- Todo guión PHP puede acceder a un conjunto de variables que tiene predefinidas (variables *superglobales*).
  - `$_GET` → Contiene cualquier variable que se le haya pasado al guión por el método GET
  - `$_POST` → Contiene cualquier variable que se le haya pasado al guión por el método POST
  - `$_COOKIE` → Contiene cualquier variable que se le haya pasado al guión a través de cookies
  - `$_FILE` → Contiene cualquier variable que se le haya pasado al guión como un archivo que ha sido subido al servidor.
  - `$_ENV` → Contiene todas las variables que se le pasan al guión con información sobre el entorno del servidor.
  - `$_SESSION` → Contiene todas las variables que han sido registradas en una sesión
  - `$_SERVER` → Información de la petición HTTP, entorno de servidor, paths
  - `$GLOBALS` → Almacena las variables globales accesibles por el guión
- Todas estas variables son arrays

Variables predeterminadas (cont.)

- Estas variables superglobales son válidas para PHP >= 4.1.0
  - Para otras versiones de PHP consulta: http://www.php.net/manual/en/reserved.variables.php
- El mal uso de variables globales y superglobales puede provocar problemas de seguridad.
  - Consulta: http://www.php.net/manual/en/security.globals.php

Agujero de seguridad por uso de register_globals

```
<?php
// Definimos $authorized = true si el usuario tiene privilegios
if (authenticated_user()) {
   $authorized = true;
}

// Como no la hemos inicializado a false, la variable
//puede ser establecida por el uso de register_globals,
//por ejemplo desde una petición GET auth.php?authorized=1
// So, anyone can be seen as authenticated!
if ($authorized) {
   include "/highly/sensitive/data.php";
}
?>
```

Ámbito de variables

- El ámbito de una variable es el contexto dentro del cual la variable se encuentra definida.
- La mayoría de variables de PHP tienen un único ámbito.
  - En este ámbito también se encuentran las variables que definamos dentro de un fichero que incluyamos con `include` o `require`.

```
<?php
  $a = 1;
  include 'b.inc';
?>
```

La variable $a puede ser usada por el código PHP de 'b.inc'

- Cuando creamos funciones propias estamos definiendo un ámbito local a dicha función
  - Cualquier variable que usemos en dicha función está limitada a ese ámbito

```
<?php
  $a = 1; /* ámbito global */
  function Test(){
    echo $a; /* referencia al ámbito local */
  }
  Test();
?>
```

La llamada a Test() no imprime nada por pantalla ya que $a no ha sido definida en el ámbito local

Ámbito de variables (cont.)

- Tenemos dos maneras de acceder al ámbito global del guión
  - Usando la palabra reservada `global`.

```
<?php
  $a = 1;  $b = 2;
  function Sum(){
    global $a, $b;
    $b = $a + $b;
  }
  Sum();
  echo $b;
?>
```

  - Usando la variable superglobal $GLOBALS

```
<?php
  $a = 1;  $b = 2;
  function Sum(){
    $b = $GLOBALS['a'] + $GLOBALS['b'];
  }
  Sum();
  echo $b;
?>
```

Ámbito de variables (cont.)

Ejemplo de variables superglobales

```
<?php
  function test_global()
  {
   // La mayoría de variables predefinicas no son superglobales y necesitamos
   // usar 'global' para que estén disponibles en el ámbito local de la
   // función.
   global $HTTP_POST_VARS;

   echo $HTTP_POST_VARS['name'];

   // Las variables superglobales están disponibles en cualquier ámbito
   // no necesitamos usar 'global'. Las variables supergloblales
   // están disponibles a partir de PHP 4.1.0, y la variable
   // HTTP_POST_VARS está 'deprecated'.
   echo $_POST['name'];
  }
?>
```

- $HTTP_POST_VARS contiene los mismos valores que $_POST
- $_POST solo está disponible para PHP >= 4.1.0, para versiones anteriores hay que usar las variables $HTTP_*_VARS

Ámbito de variables (cont.)

- Podemos declarar una variable dentro de una función con la palabra reservada `static`
- Estas variables mantendrán el valor que le hayamos asignado entre distintas llamadas a dicha función.

Uso de variables static

```
<?php
  function foo(){
    static $int = 0;            // OK
    //static $int = 1+2;        // incorrecta  (no se pueden usar expresiones)
    //static $int = sqrt(121);  // incorrecta  (también es una expresión)

    $int++;
    echo $int;
  }
  foo(); // salida: 1
  foo(); // salida: 2
?>
```

Variables variables

- A veces es conveniente tener variables cuyo nombre sea variable, i.e, su nombre puede ser modificado dinámicamente.

Declarando variable variable

```
<?php
 $a='hello';
 $$a = 'world';
?>
```

  - Cuando usamos variables variables, tomamos el valor de la variable como el nombre de variable.
    - En el ejemplo anterior $$a hace uso del valor almacenado en la variable $a para definir una nueva variable $hello.

Uso de variable variable

```
<?php
 echo "$a ${$a}";
?>
```

Ejemplo equivalente

```
<?php
 echo "$a $hello";
?>
```

Funciones para tratar con variables

`isset(<variable>)`

- Devuelve TRUE si a la variable `'<variable>'` se le ha asignado un valor, devuelve FALSE en otro caso.

`unset(<variable>)`

- Elimina la variable '<variable>'

`is_<tipo>(<variable>)`

- Devuelve TRUE si a la variable `'<variable>'` tiene asignado un valor de tipo `boolean`, devuelve FALSE en otro caso.
- `<tipo>` → Puede tomar los valores `int`, `bool`, `int`, `float`, `double`, `null`, `object`, `array`, `string`.

`is_callable(<variable>)`

- Devuelve TRUE si a la variable `'<variable>'` tiene asignado una cadena con el nombre de una función válida o
- si contiene: `array($algunObjeto, nombreMetodo')`
- o devuelve FALSE en otro caso.

Constantes

- En PHP podemos definir un identificador para un valor constante
  - Una vez que la constante de un valor mantiene dicho valor mientras se ejecuta el guión.
  - Las constantes pueden ser definidas por el programador o pueden estar definidas por PHP
  - Normalmente los identificadores de las constantes se escriben todo en mayúsculas para distinguirlas, además no llevan el signo '$' delante.
  - Sólo podemos declarar constantes de tipos básicos.

`defined("<nombre>")` → Verifica si existe la constante con nombre `'<nombre>'`.

Definición de una constante (defConstantes.php)

```
<html>
  <head><title>Ejemplo de definición de constantes</title></head>
  <body>
    <?php
      define("MI_CONSTANTE", "<p>Este es el valor de la constante</p>");
      echo MI_CONSTANTE;
    ?>
  </body>
</html>
```

Constantes (cont.)

- Constantes predefinidas
  - `__FILE__` → Almacena el nombre del guión que se está ejecutando
  - `__LINE__` → Almacena el número de la línea donde se encuentra la instrucción que se está ejecutando.
  - `PHP_VERSION` → Almacena la versión de PHP que se está ejecutando.
  - `PHP_OS` → Almacena el tipo de Sistema Operativo sobre el que se ejecuta PHP
  - `E_*` → Distintos niveles de error
- Consultar: http://www.php.net/manual/en/reserved.constants.core.php

`usoConstantesPredefinidas.php`

```
<html>
  <head><title>Ejemplo de uso de las constantes predefinidas</title></head>
  <body>
    <?php
      echo "<p>Este es guión: " . __FILE__ . "</p>";
      echo "<p>Este es la línea: " . __LINE__ . "</p>";
    ?>
  </body>
</html>
```

Operadores

- Principales tipos de operadores existentes en PHP
  - Operador de asignación
    - Ej: "=", "+=", "-=", "&=", "|=", "^=", ">>=", "<<="
  - Operadores aritméticos
    - Ej: "+", "-", "*", "/", "%" (módulo)
  - Operadores binarios
    - Ej, "&", "|", "~" (not), "^" (xor), "<<" (desp. izquierda), ">>"
  - Operadores de comparación
    - Ej: ">", "<", ">=", "<=", "==", "!=", "===" (identidad), "!=="
  - Operadores lógicos
    - Ej: "&&", "and", "||", "or", "!", "xor"
  - Operadores pre/post incremento / decremento
    - Ej: "$a++", "++$a", "$a--", "--$a"
  - Operadores de cadena
    - Ej: ".", ".="

Operadores (cont.)

- Existen un conjunto de operadores adicionales que son especiales
  - Operador @
    - Se coloca al comienzo de una expresión en PHP, cualquier mensaje de error que pudiera generarse a causa de esa expresión será ignorado.

```
<?php
  $arch = @file('archivo no existente') or die("No existe el archivo!");
?>
```

  - Operador `` `<comando>` ``
    - PHP ejecutará la cadena `<comando>` como un comando de la consola de nuestro sistema. El resultado del comando será volcado por la salida.

```
<?php
  $salida = `ls -al`;
  echo "<pre>$salida</pre>";
?>
```

  - (PHP 5) Operador `instanceof`
    - Verifica si un objeto es de una clase.

```
<?php if ( $obj instanceof B){ ... } ?>
```

Ejemplos de usos de operadores

Operadores de asignación (operAsignacion.php)

```
<html>
  <head><title>Ejemplo de uso de operadores de asignación</title></head>
  <body>
  <?php
    $var = 100;
    echo "<p>Valor inicial de \$var=$var</p>";
    $var += 25;
    echo "<p>Hemos sumado a la variable, ahora su valor es= $var</p>";
   $var -= 11;
    echo "<p>Hemos restado a la variable, ahora su valor es= $var</p>";
  ?>
  </body>
</html>
```

Operadores aritméticos (operAritmeticos.php)

```
<html>
  <head><title>Ejemplo de uso de operadores aritméticos</title></head>
  <body>
  <?php
    $var1 = 50; $var2 = 40; $result = $var1 / $var2;
    echo "<p>La división de \$var1 y \$var2 es=$result</p>";
  ?>
  </body>
</html>
```

Ejemplos de usos de operadores (cont.)

Operadores de comparación (operRelYLogicos.php)

```
<html>
  <head><title>Ejemplo de uso de operadores de comparación</title></head>
  <body>
  <?php
    $a = 20; $b = 30;
    echo "<p>\$a=$a \$b=$b</p>";
    $var += 25;

    if ( $a == $b ){
      echo "<p>\$a y \$b tienen el mismo valor.</p>"
    }else{
      echo "<p>\$a y \$b tienen valores distintos.</p>"
    }
    $mostrarMensaje="si";
    if ( ($a < $b) && ($mostrarMensaje == "si") ){
      echo "<p>\$a es estrictamente menor que \$b.</p>"
    }
  ?>
  </body>
</html>
```

Instrucciones condicionales

- If-then-else

Sintaxis

```
if( condicion ){
  // Instrucciones
}elseif (condicion){
  // Instrucciones
}else{
  // Instrucciones
}
```

Sintaxis alternativa

```
<?php
  if( condicion ) :
?>
  <!-- Codigo HTML, JavaScript, Texto -->
<?php
  endif;
?>
```

`elseif` puede ir junto o separado

- Switch

Sintaxis

```
switch( expresion ){
  case valor1:
    // Instrucciones
  break;
  case valor2:
    // Instrucciones
  break;
  default:
    // Instrucciones
}
```

Es legal cualquier expresión que evalue a un tipo básico, i.e objetos y arrays no están permitidos

Bucles

While-do

Sintaxis

```
while( condicion ){
  // Instrucciones
}
```

Sintaxis alternativa

```
<?php while( condicion ) : ?>
  <!-- Codigo HTML, JavaScript, Texto  -->
<?php endwhile; ?>
```

Do-while

Sintaxis

```
do{
  // Instrucciones
} while( condicion );
```

Bucles (cont.)

- For

Sintaxis

```
for( expresion1; expresion2; expresion3 ){
  // Instrucciones
}
```

- `expresion1` → Se evalúa antes de que comience el bucle
- `expresion2` → Se evalúa al comienzo de cada iteración si evalúa a `false` no se itera más
- `expresion3` → Se evalúa al final de cada vuelta del bucle

Sintaxis alternativa

```
<?php for( expresion1; expresion2; expresion3 ) : ?>
  <!-- Codigo HTML, JavaScript, Texto  -->
<?php enfor; ?>
```

Bucles (cont.)

- Foreach (PHP 4)
  - Nos permite iterar de manera simple sobre un array
  - Se ejecuta sobre una copia del array que le pasamos
  - El puntero del array original es modificado

Sintaxis

```
foreach( expresion_array as $valor ){
  // Instrucciones
}
```

  - Itera sobre el array asignando a `$valor`, el valor de la posición del array en la que nos encontramos.

Sintaxis alternativa

```
foreach( expresion_array as $clave => $valor ){
  // Instrucciones
}
```

  - Igual que el anterior pero además se asigna la clave que se usa como entrada del array a la variable `$key`.

Bucles (cont.)

- En todos los bucles la instrucción `break;` hace que se salga de dicho bucle
  - `break` permite que le pasemos un entero como argumento adicional, que nos indica de cuantas instrucciones saldremos.

Uso de break; con parámetro

```
<?php
  $i = 0;
  while (++$i) {
   switch ($i) {
   case 5:
       echo "Van 5<br />\n";
       break 1;  /* Sólo salimos del switch. */
   case 10:
       echo "Van 10; salimos<br />\n";
       break 2;  /* Salidmos del switch y del while. */
   default:
       break;
   }
}
?>
```

Bucles (cont.)

- En todos los bucles la instrucción `continue;` hace que se dejen de ejecutar las instrucciones posteriores y se itere de nuevo
  - `continue` permite que le pasemos un entero como argumento adicional, que nos indica de cuantas instrucciones saldremos.

Uso de continue; con parámetro

```
<?php
  $i = 0;
  while ($i++ < 5) {
    echo "Fuera<br />\n";
    while (1) {
      echo "  En medio<br />\n";
      while (1) {
        echo "  dentro<br />\n";
        continue 3;
      }
      echo "Esto nunca se imprimirá.<br />\n";
   }
   echo "Esto tampoco.<br />\n";
}
?>
```

- También podemos usar la instrucción `continue;` dentro de una instrucción `switch`

Bucles (cont.)

- Ejercicio 1
  - Que sucede al ejecutar este fragmento de código

```
<?php
  for ($i = 0; $i < 5; ++$i) {
     if ($i == 2)
         continue
     print "$i\n";
  }
?>
```

¿Por qué?

Funciones

- Cualquier código PHP puede aparecer dentro de una función.
  - Esto incluye definición de funciones y clases

Sintaxis

```
function <nombre funcion> ($arg1, $arg2, ...){
  // Instrucciones

  return valor; // Opcional
}
```

- El nombre de las funciones no es sensible a mayúsculas y minúsculas
  - Es recomendable que la llamada a las funciones se realicen con el mismo nombre con el que han sido declaradas.
- Las funciones en PHP no pueden sobrecargarse
  - i.e, no podemos eliminar la definición de una función y volverla a definir.
- Es posible hacer llamadas recursivas.
  - El límite del número de llamadas lo depende de nuestro PHP

Argumentos de funciones

- PHP soporta varios tipos de paso de parámetros a una función/método
  - Por valor (por defecto)
  - Por referencia (PHP >= 4)
  - Número variable de argumentos (PHP >= 4)
- Argumentos por referencia
  - Por defecto se realiza una copia del argumento que le pasamos a la función.
  - Los cambios realizados al argumento dentro de la función no serán visibles fuera.
  - Para que un argumento se pase por referencia, es necesario que delante de la definición del argumento pongamos un `'&'`.

Ejemplo de parámetro por referencia

```
function miFuncion ( $arg1, &$arg2){
  $arg2 = 10;
}
```

Argumentos de funciones (cont.)

- Argumentos por defecto
  - Similar a como funciona en C++

Ejemplo de uso de parámetros por defecto

```
<?php
  function makecoffee($type = "cappuccino") {
    return "Making a cup of $type.\n";
  }
  echo makecoffee();
  echo makecoffee("espresso");
?>
```

  - Pueden usarse arrays y el tipo NULL como parámetros por defecto.
  - En general tiene que ser una expresión constante, i.e, no puede ser una variable o un atributo de una clase o una función.
  - NOTA: Los argumentos que tienen valores por defecto tienen que situarse al final de la declaración de argumentos.

Uso incorrecto de argumentos por defecto

```
function funcionMalDeclarada ( $arg1 = "Valor", $arg2) {
  ...
}
```

Argumentos de variables (cont.)

- Argumentos de longitud variable
  - Sólo está disponible en PHP 4 y posteriores
  - Para tratar con este tipo de argumentos simplemente tenemos que usar un conjunto de funciones predefinidas en PHP.
  - `func_num_args()`
    - Devuelve el número de argumentos que se le han pasado a la función
  - `func_get_arg(`*`idx`*`)`
    - Devuelve el idx-ésimo argumento que se le ha pasado a la función
    - Si `idx > func_num_args()`, `func_get_arg()` devuelve `false`.
  - `func_get_args()`
    - Devuelve los parámetros que se le han pasado a la función como un array.

Variables de funciones

- Si cuando usamos una variable le añadimos '()' PHP buscará una función cuyo nombre es el valor de la variable en forma de cadena.
  - Esto nos permite crear tablas de funciones.
  - Creación de Hooks y callbacks

Ejemplo de uso de variables de función

```
<?php
  function funcion1() { echo "Dentro de funcion1()<br />\n"; }
  function funcion2($arg = '') {
   echo "Dentro de funcion2(); Mi argumento es '$arg'.<br />\n";
  }
  function doEcho($string) { echo $string; }

  $func = 'funcion1';
  $func();        // Llamada a funcion1()

  $func = 'funcion1';
  $func('test');  // Llamada a funcion2()

  $func = 'doEcho';
  $func('test');  // Llamada a doEcho()
?>
```

Funciones predefinidas de tratamiento de funciones

`function_exits(<nombre funcion>);`

- Devuelve `true` si la función cuyo nombre es `'<nombre funcion>'` existe o `false` en otro caso. Tanto como función predefinida como definida por el usuario.

`call_user_func(<nombre funcion>, [<arg1>,<arg2>, …]);`

- Llama a la funcion `'<nombre funcion>'` pasándole el resto de argumentos como parámetros.

- Para ver el resto de funciones para el tratamiento de funciones consulta http://www.php.net/manual/en/ref.funchand.php

Instrucciones `require` y `require_once`

`require(<path a un guion php>)`

- Incluye y evalúa el guión PHP cuya ruta es `'<path a un guion php>'`
- Si no se encuentra el guión se genera un **error fatal**.
- Hasta PHP 4.0.2 la instrucción `require` siempre se ejecutaba aunque estuviera dentro de una instrucción condicional que no se ejecutara.
- Los bucles no afectan el comportamiento de la instrucción `require`, la inclusión del archivo sólo se realizará una vez. El código que se incluye si que se ve afectado.
- Con PHP sobre Linux el path puede ser una URL.

`require_once(<path a un guion php>)`

- Igual que `require` excepto que si el guión ya ha sido incluido no volverá a incluirse

```
<?php
  // Esto incluye a.php
  require_once("a.php");
  // Esto incluye a.php ¡de nuevo en Windows! (PHP 4 only)
  require_once("A.php");
?>
```

Instrucciones `include` e `include_once`

`include(<path a un guion php>)`

- Incluye y evalúa el guión PHP cuya ruta es `'<path a un guion php>'`
- Si no se encuentra el guión se genera una **advertencia**.
- Hasta PHP 4.0.2 la instrucción `require` siempre se ejecutaba aunque estuviera dentro de una instrucción condicional que no se ejecutara.
- Los bucles no afectan el comportamiento de la instrucción `require`, la inclusión del archivo sólo se realizará una vez. El código que se incluye si que se ve afectado.
- Con PHP sobre Linux el path puede ser una URL.

`include_once(<path a un guion php>)`

- Igual que `require` excepto que si el guión ya ha sido incluido no volverá a incluirse

```
<?php
  // Esto incluye a.php
  require_once("a.php");
  // Esto incluye a.php ¡de nuevo en Windows! (PHP 4 only)
  require_once("A.php");
?>
```

Ejemplos de include

- Ejemplo 1

vars.php

```
<?php
  $color = 'green';
  $fruit = 'apple';
?>
```

test1.php

```
<?php
  echo "A $color $fruit"; // A
  include 'vars.php';
  echo "A $color $fruit"; // A green apple
?>
```

- Ejemplo 2
  - Si la inclusión se realiza dentro de una llamada a función las variables del archivo incluido solo serán visibles dentro de la función

test2.php

```
function foo() {
    global $color; include 'vars.php'; echo "A $color $fruit";
}
/* vars.php está en el ámbito de foo() de manera *
 * que $fruit no está disponible fuera de ella.  *
 * $color si lo está porque la hemos declarado   *
 * como global.                                  */
foo();                    // A green apple
echo "A $color $fruit";  // A green
```

Array

- En PHP los Array son tablas asociativas
  - Almacenan pares <clave, valor>
  - Están optimizadas para ser usadas como los arrays a los que estamos acostumbrados.
- Creación de un array
  - Si no especificamos una clave para uno de los valores, la clave se obtiene sumándole 1 al máximo índice entero que haya existido como clave en nuestro array, si no existía ningún índice entero se toma como clave 0.

Sintaxis

```
array( [key =>] value
     , ...
     )
// key puede ser un entero o cadena
// value puede ser de cualquier tipo
```

Ejemplo de uso

```
<?php
  $arr = array("foo" => "bar", 12 => true);
  echo $arr["foo"]; // bar
  echo $arr[12];    // 1

  // Este array es igual a ...
  array(5 => 43, 32, 56, "b" => 12);

  // ...este array
  array(5 => 43, 6 => 32, 7 => 56, "b" => 12);
?>
```

Array (cont.)

Sintaxis alternativa

```
$arr[key] = value;
$arr[] = value;
// key puede ser un entero o cadena
// value puede ser de cualquier tipo
```

Ejemplo de uso

```
<?php
$arr = array("foo" => "bar", 12 => true);
echo $arr["foo"]; // bar
echo $arr[12];    // 1
?>
```

- Si aún no existe `$arr` se crea dicha variable
- Eliminando un par <clave, valor>

Ejemplo de eliminación de componentes

```
<?php
$arr = array(5 => 1, 12 => 2);

$arr[] = 56;    // Es equivalente a $arr[13] = 56; con el estado actual del array

$arr["x"] = 42; // Añade un nuevo elemento con clave "x"

unset($arr[5]); // Eliminamos un elemento del array

unset($arr);    // Eliminamos el array completo
?>
```

Array (cont.)

Ejemplo de array multidimensional

```
<?php
  $arr = array("somearray" => array(6 => 5, 13 => 9, "a" => 42));

  echo $arr["somearray"][6];    // 5
  echo $arr["somearray"][13];  // 9
  echo $arr["somearray"]["a"];  // 42
?>
```

inserción y borrado de elementos

```
<?php
  // Creamos un array
  $array = array(1, 2, 3, 4, 5);
  print_r($array);

  // Eliminamos todas las componentes
  foreach ($array as $i => $value) {
    unset($array[$i]);
  }
  print_r($array);

// Añadimos un nuevo valor
// el valor de la nueva clave que
// ha sido generada es 5 !!
$array[] = 6;
print_r($array);
```

```
  // reindexamos:
  $array = array_values($array);
  $array[] = 7;
  print_r($array);
?>
```

El valor máximo de clave no tiene por qué estar insertado dentro del array

Array (cont.)

- Funciones para tratar con array

`reset(&<array>)`

- Establece el puntero interno de `<array>` al comienzo.

`end(&<array>)`

- Establece el puntero interno de `<array>` a su último elemento.

`next(&<array>)`

- Avanza el puntero interno de `<array>`.

`prev(&<array>)`

- Retrasa una posición el puntero interno de `<array>`.

`current(&<array>)`

- Devuelve el valor actual de `<array>`.

`in_array(<valor>, &<array>)`

- Devuelve TRUE si `'<valor>'` se encuentra como un valor de `<array>`.

Array (cont.)

`list(<var1>, <var2>, ... ) = $arr`

- Asigna los valores de `$arr` a las variables `<var*>`.

Ejemplo de uso de list()

```
<?php
  $info = array('coffee', 'brown', 'caffeine');
  list($drink, $color, $power) = $info;
  echo "$drink is $color and $power makes it special.\n";

  // List parcial
  list($drink, , $power) = $info;
  echo "$drink has $power.\n";

  // Saltamos las dos primeras excepto la tercera
  list( , , $power) = $info;
  echo "I need $power!\n";
?>
```

Para obtener mayor información sobre las funciones relacionadas con arrays, consultar: http://www.php.net/manual/en/ref.array.php

Array (cont.)

- Operadores para arrays
  - $a + $b → Unión (como conjunto) de $a y $b.
  - $a == $b → Verifica si $a y $b tienen los mismos pares <clave, valor>
  - $a != $b → Verfica si $a y $b tienen pares distintos
  - $a === $b → Verifica si $a y $b tienen los mismos pares <clave, valor> en el mismo orden y con el mismo tipo.
  - $a !== $b → Verifica si $a y $b no son idénticos.

Unión de arrays

```
<?php
$a = array( "a" => "apple",
            "b" => "banana");
$b = array( "a" => "pear",
            "b" => "strawberry",
            "c" => "cherry");

$c = $a + $b; // Union of $a and $b
echo "Union de \$a y \$b: \n"; var_dump($c);

$c = $b + $a; // Union of $b and $a
echo "Union de \$b y \$a: \n"; var_dump($c);
?>
```

Salida por pantalla

```
Unión de $a y $b:
array(3) {
  ["a"]=> string(5) "apple"
  ["b"]=> string(6) "banana"
  ["c"]=> string(6) "cherry"
}
Uniónn de $b y $a:
array(3) {
  ["a"]=> string(4) "pear"
  ["b"]=> string(10) "strawberry"
  ["c"]=> string(6) "cherry"
}
```

Trabajo con formularios

Tecnologías Web

Introducción

- La interacción con el usuario se realiza mediante formularios y cookies.
- En ambos casos tenemos que tratar con cadenas de texto.
  - Conversión de cadena a valores numéricos
  - Validación de las cadenas que nos pasan
  - Buscar patrones en dichas cadenas.
- Para el tratamiento de cadenas de texto tenemos dos aproximaciones
  - Tratamiento de cadenas mediante las funciones asociadas.
  - Tratamiento de las cadenas usando expresiones regulares.
- El conjunto de funciones PHP que trata cadenas de texto es muy extenso → http://www.php.net/manual/en/ref.strings.php.
  - Conversión de caracteres especiales a etiquetas HTML
  - Conversiones entre arrays y cadenas de texto.

Funciones para tratamiento de cadenas

`substr(<cadena>, <idx>, [<long>])`

- Devuelve una subcadena de `<cadena>`, comenzando a partir de la posición `<idx>`. Si especificamos el último parámetro se toman los `<long>` siguientes caracteres, si no lo especificamos se obtiene hasta el final de la cadena.

`strpos(<cadena>, <cadena2>, [<desp>])`

- Buscan en `<cadena>`, la primera aparición de `<cadena2>`. Si especificamos el último parámetro se comienza a buscar a partir de la posición `<desp>` de `<cadena>`.

`htmlspecialchars(<cadena>)`

- Reemplaza en `<cadena>`, caracteres que no son válidos en HTML y los convierte en sus equivalentes válidos.
- & ⇔ & ,"⇔", <⇔<, >⇔>

Funciones para cadena: http://www.php.net/manual/en/ref.strings.php

Funciones para el tratamiento de cadenas (cont.)

`strlen(<cadena>)`

- Devuelve la longitud de `<cadena>`.

`printf/sprintf(<formato>,<arg1>, <arg2>,…)`

- Imprime `<arg1>`, `<arg2>`, etc. Dando formato a dichos valores siguiendo las especificaciones de `<formato>`.
- La diferencia entre ambas es que `sprintf()` devuelve la cadena generada y `printf()` la imprime.
- `<formato>` → http://www.php.net/manual/en/function.sprintf.php

`nl2br(<cadena>)`

- Se cambian los saltos de línea `'\n'` `<cadena>`, por etiqueta HTML `<br>`.

`trim(<cadena>)`

- Elimina los espacios en blanco iniciales y finales de `<cadena>`.

Expresiones regulares

- Son un sistema complejo y potente de búsqueda de patrones en cadenas de texto (http://www.php.net/pcre).
- Una expresión regular es una cadena de texto.
  - Esta cadena de texto define un patrón de coincidencias en otras cadenas.
  - Ej:`\d{2}-\d{7}`
    - \d → Representa un dígito de 0 al 9
    - {2} → Encontrar 2 elementos de lo anteriormente especificado (dígitos)
    - - → Encontrar un carácter '-'
  - Ej: </?[bBiI]>
    - < → Encontrar el carácter `'<'`
    - / → Encontrar el carácter `'/'`
    - ? → Hace opcional el elemento anterior (el carácter `'/'`)
    - [bBiI] → Encontrar `b`, `B`, `i`, `I`
    - > → Encontrar el carácter `'>'`
- Dentro de una expresión regular, algunos caracteres coinciden consigo mismo y otros tienen un significado especial. Estos caracteres son denominados "metacaracteres".

Expresiones regulares (cont.)

- Cuantificadores
  - Indica cuantos elementos deseamos de un cierto tipo
  - Hay que situar el cuantificador justo después del elemento que deseamos cuantificar
  - `*` → Cero o más repeticiones
  - `+` → Una o más repeticiones
  - `?` → Opcional (cero o una)
  - `{x}` → Exáctamente x
  - `{x,}` → Al menos x
  - `{x,y}` → Al menos x, pero no más de y
- Ej: `ba+` ("b", luego al menos una "a")
  - Encaja: ba, baa, baaa, rumba
  - No encaja: b, abs, taaa-daaa
- Ej: `ba(na){2}` ("ba", luego "na" al menos dos veces)
  - Encaja: banana, bananas, semibanana
  - No encaja: cabana, bananarama

Expresiones regulares (cont.)

- Anclas
  - En los del apartado anterior hemos visto que `ba(na)+` encaja con banana pero también con bananas.
  - Las anclas alinean un patrón para hacer una coincidencia más específica.
  - Un ancla hace que el patrón coincida al principio o final de línea
    - El ancla `'^'` hace que el patrón coincida a principio de línea.
    - El ancla `'$'` hace que el patrón coincida al final de línea.
- Ej: `^Hol`
  - Encaja: "Hola", "Hola mundo", "Hola tío".
  - No encaja: "HHola", "GHola"
- Ej: `^(W|w|B|b)illy$`
  - Encaja: Willy, willy, billy, Billy
  - No encaja: TWillyam

Expresiones regulares (cont.)

- Clases de caracteres
  - Una clase de caracteres permite representar un conjunto de caracteres como un solo elemento en una expresión regular.
    - Ej: `p[eo]pa` (coincide con pepa y con popa)
  - Para situar un conjunto completo de caracteres se sitúa el primer y último carácter, separados por un guión.
    - Ej: `[a-zA-Z]` (coincide con todos los caracteres del alfabeto)
    - Si queremos incluir un guión `'-'` dentro de una clase, hay que escaparlo `'\-'`.
  - También se puede crear una clase de caracteres negada, es decir, coincide con cualquier carácter que no se encuentre en la clase.
    - Ej: `[^a-zA-Z]` (coincide con cualquier cosa que no sean letras)

Expresiones regulares (cont.)

- Para las clases más comunes existen metacaracteres propios

Metacaracter	Descripción	Clase equivalente
`.`	Cualquier carácter (excepto nueva línea)	`[a-zA-Z0-9\t\f]`
`\d`	Dígitos	`[0-9]`
`\D`	Cualquier carácter que no sea dígito	`[^0-9]`
`\w`	Carácter de palabra	`[a-zA-Z0-9]`
`\W`	Cualquier carácter que no sea de palabra	`[^a-zA-Z0-9]`
`\s`	Espacio en blanco	`[\t\n\r\f]`
`\S`	Cualquier carácter que no sea espacio en blanco	`[^\t\n\r\f]`

- Ejercicio
  - ¿Cuál es la expresión regular para los caracteres hexadecimales válidos?

Expresiones regulares en PHP

- Cuando le pasamos un patrón a una de las funciones de tratamiento de expresiones regulares en PHP, éste tiene que ir entre delimitadores
  - Un delimitador puede ser cualquier carácter que no sea letra.
  - Si queremos usar el delimitador dentro de la expresión regular hay que escaparlo
  - Habitualmente se usa '/'.
  - Ej: "/[a-zA-Z]/", "@.*?@"
- Funciones

  `preg_match(<expresión>, <cadena>, [<array>])`
  - <expresion> → Expresión regular
  - <cadena> → Cadena sobre la que queremos aplicarlo
  - <array> → Array en el que se nos devuelve los caracteres capturados. Un conjunto de paréntesis "( )" captura todos los caracteres que coincidan con la parte del patrón dentro de los paréntesis.

Expresiones regulares en PHP (cont.)

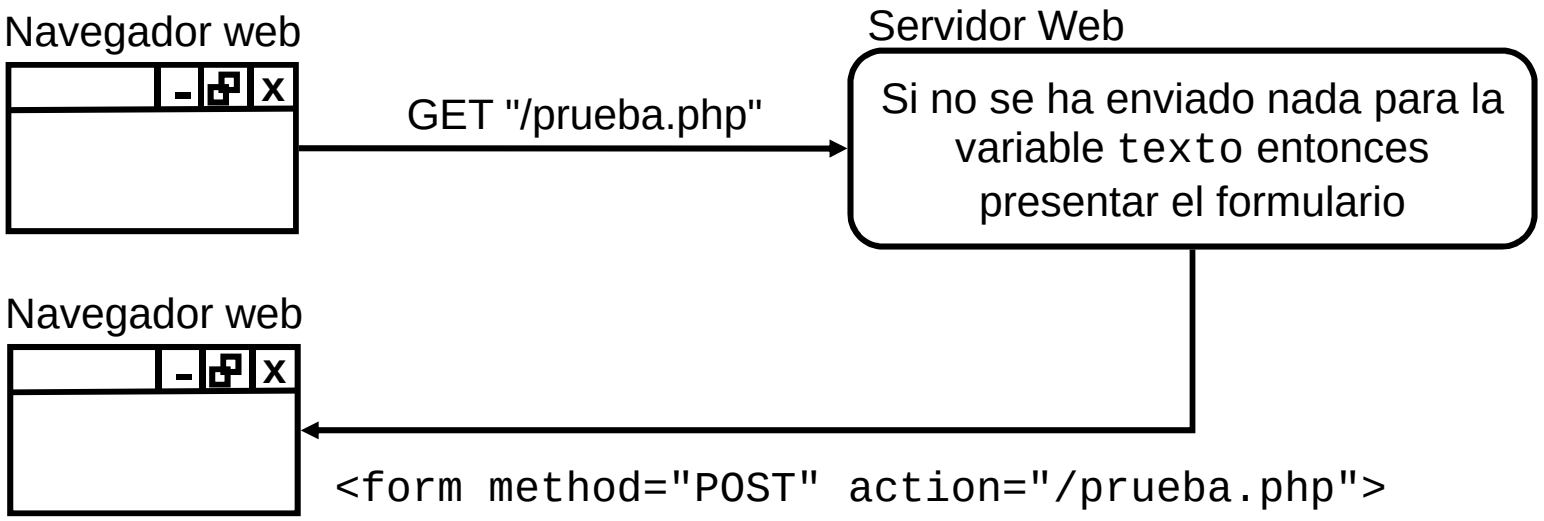
Ejemplo de captura

```
// Patron ^(\d{2,3})[\-\.]?(\d{7})
if(preg_match("/(^\d{2,3})[\-\.]?(\d{7})/", $_POST['tel'], $resultados)){
  print "El teléfono es: $resultados[0]";
  print "El prefijo es: $resultados[1]";
  print "El número de abonado es: $resultados[2]";
}
```

- El primer elemento del array (elemento 0) contiene la cadena que coincide con todo el patrón.
- Los siguientes elementos del array contienen las cadenas que coinciden con las partes del patrón en cada conjunto de paréntesis

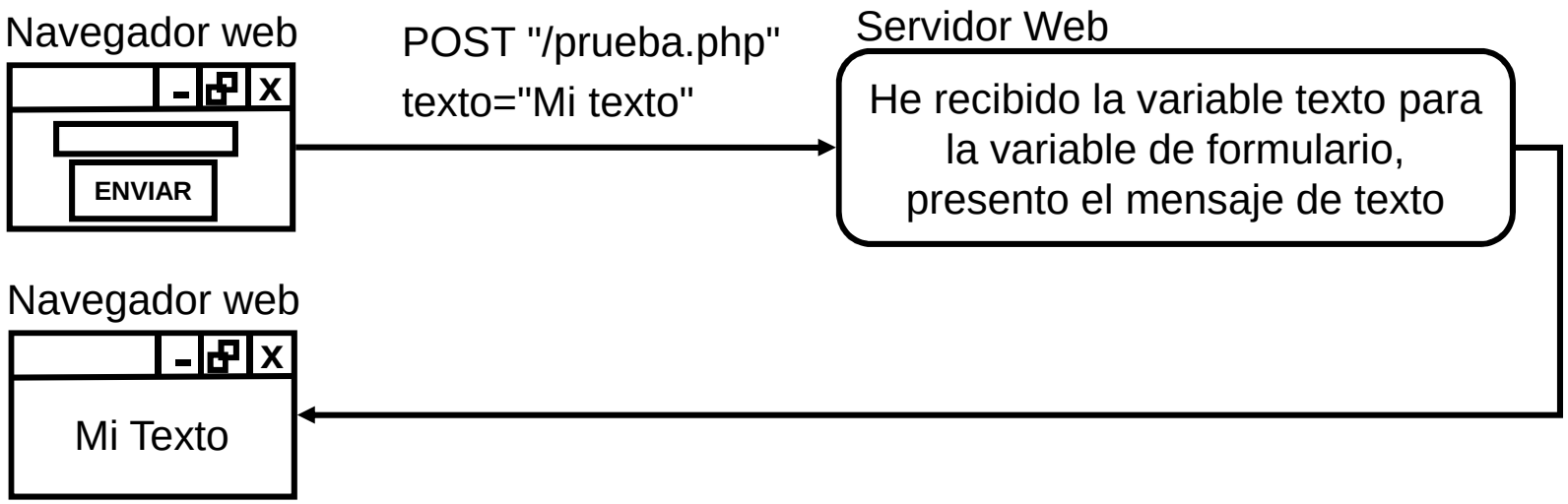
Interacción típica con una página PHP

- Habitualmente las páginas PHP cuando presentan un formulario, también son las encargadas de procesarlos.




```
<form method="POST" action="/prueba.php">
  Texto: <input type="text" name="texto" /><br />
  <input type="submit" value="Muestra texto" />
</form>
```

Interacción típica con una página PHP



Obteniendo datos del usuario

interacUsuario.php

```
<html>
  <head><title>Interacción con el usuario</title></head>
  <body>
    <?php if($_POST["submit"] == "Adelante") : ?>
      <p>Acabas de escribir: <?php echo $_POST["texto"] ?></p>
    <?php else : ?>
      <form action="<?php echo($_SERVER["PHP_SELF"])?>" method="POST">
        <p>Texto: <input type="text" size="20" id="texto" name="texto" /></p>
        <input type="submit" id="submit" name="submit" value="Adelante" />
      </form>
    <?php endif; ?>
  </body>
</html>
```

Aquí distinguimos si el usuario nos ha enviado datos o si hemos cargado la página en nuestro navegador (petición GET)

Ejercicio 1: hacer una calculadora simple con un formulario

Interacción avanzada con formularios

formularioComplejo.php

```
<html>
  <head><title>Formulario complejo</title></head>
  <body>
  <?php
    if($_POST["submit"]=="Adelante"){
      echo "<p>Id Producto seleccionado: $_POST[product_id]</p>";
     var_dump ($_POST["category"]);
      foreach($_POST["category"] as $cat){
        echo "<p>Categoria: $cat</p>";
      }
    }else{
       $form = <<< _HTML_
        <form action="$_SERVER[PHP_SELF]" method="POST">
          <p>Id Producto: <input type="text" size="20" name="product_id" /></p>
          <p><select name="category[]" multiple="multiple">
            <option value="electronica">Sección de electrónica</option>
            <option value="electrodomesticos">Sección de electrodomésticos</option>
            <option value="muebles">Sección de mobiliario</option>
          </select></p>
          <input type="submit" name="submit" value="Adelante" />
        </form>
_HTML_;
      echo $form;
    }
  ?>
  </body>
</html>
```

Si no usamos "[]" en el <select> PHP no entiende que el campo puede contener varios valores.

Procesando formularios con funciones

- Podemos hacer más flexible el trabajo con formularios usando funciones.

formularioConFunciones.php

```
// Comprobamos nos envían datos
if( array_key_exists('_form_enviado',$_POST)){
  // Comprobamos los datos que nos han pasado
  if( validate_form() ){
    // Datos correctos, hacemos algo con ellos
    process_form();
  }else {
    // Datos incorrectos, mostrarmos errores y el formulario
    show_form();
  }
}else{
  // Mostramos el formulario
  show_form();
}
```

Procesando formularios con funciones (cont.)

formularioConFunciones.php

```
// Procesamiento de los datos del formulario
function process_form(){
  echo "Texto introducido: $_POST[texto]";
}

// Mostramos el formulario
function show_form(){
  print <<<_HTML_
<form method="POST" action="$_SERVER[PHP_SELF]"
Texto: <input type="text" name="texo" /> <br />
<input type="submit" />
<input type="hidden" name="_form_enviado" value="1" />
</form>
_HTML_;
}

function validate_form(){
  if( strlen($_POST["texto"]) < 3 ){
    return false;
  }else{
    return true;
  }
}
```

Procesando con formularios (cont.)

- Cuando la validación de un formulario falla, es conveniente que el usuario sea informado de qué es lo que es incorrecto, en vez de simplemente volverle a presentar el formulario.

formularioControlErrores.php

```
// Mostramos el formulario
function show_form($errores=''){
  if($errores){
    print "Por favor, corrija los siguientes erroes: <ul><li>";
    print implode("<li></li>", $errores);
    print "</il></ul>"
  }
  // Impresión del formulario
  ...
}
function validate_form(){
  $errores = array();
  if( strlen($_POST["texto"]) < 3 ){
    $errores[] = "Tiene que introducir al menos 3 caracteres";
  }else{
    return true;
  }
  return errores;
}
```

Validando elementos del formulario

- Validar elementos obligatorios
  - Para tener la certeza de que se ha escrito algo en un elemento obligatorio, podemos usar la función `strlen()`.

```
if ( strlen($_POST['email']) == 0){
  $errores[] = "Tiene que introducir una dirección de correo válida";
}
```

- Validar elementos númericos
  - Para asegurarnos que un valor enviado es un entero o un número en coma flotante, tenemos que usar las funciones `intval()`, `floatval()` que convierten de cadena a número. Una vez que hemos hemos convertido a número, realizamos la conversión inversa con `strval()` y comparamos.

```
if ($_POST['edad']) == strval(intval($_POST['edad'])){
  $errores[] = "Introduzca una edad válida.";
}
```

```
if ($_POST['precio']) == strval(floatval($_POST['precio']))){
  $errores[] = "Introduzca un precio válido.";
}
```

Validando elementos del formulario

- Validar cadenas
  - Para comprobar que se ha introducido texto que no sean únicamente caracteres en blanco, podemos usar `trim()` y `strlen()`.
  - Si además vamos a utilizar más adelante el valor de la cadena habiendo quitado los espacios en blanco podemos modificar el valor obtenido a través de `$_POST`.

```
if ( strlen(trim($_POST['nombre'])) == 0 ){
  $errores[] = "Es necesario que introduzca su nombre";
}
```

```
// Modificamos el valor del array $_POST
$_POST['nombre']=trim($_POST['nombre']);

if ( strlen( $_POST['nombre']) ){
  $errores[] = "Es necesario que introduzca su nombre";
}
```

Validando elementos de formulario

- Rangos numéricos
  - Comprobamos primero que se trata de un número y después si está entre el rango que nos interesa.

```
if ($_POST['edad']) == strval(intval($_POST['edad'])){
  $errores[] = "Introduzca una edad válida.";
} elseif($_POST['edad'] < 18 || $_POST['edad'] > 65 ){
  $errores[] = "Su edad debe estar entre 18 y 65 años."
}
```

- Direcciones de correo electrónico
  - Como validación de una dirección de correo electrónico, entenemos por validación de la cadena de texto que la representa.

```
if ( ! Preg_match('/^[^@\s]+@([a-z0-9]+\.)+[a-z]{2,}$/i', $_POST['email'])){
  $errores[] = "Su edad debe estar entre 18 y 65 años."
}
```

Uso de variables: Variables de entorno

- Cuando el navegador realiza una petición HTTP al servidor, envía además una lista de variables.
  - Algunas de ellas son:
    - `REMOTE_ADDR` → Almacena la IP de la máquina que ha realizado la petición.
    - `HTTP_USER_AGENT` → Almacena el tipo de navegador que ha realizado la petición.
  - Para obtener mayor información sobre el conjunto de variables visita http://hoohoo.ncsa.uiuc.edu/cgi/env.html
- Utilidad
  - Contenido dinámico para un navegador específico sin usar JavaScript.
  - Controlar la autorización y autenticación sobre nuestro sitio WEB.

Uso de variables: Variables de entorno

- Uso
  - Podemos acceder a través del array de variables superglobales $_SERVER, ej: `$_SERVER['REMOTE_ADDR']`
  - Para no confundirnos con una variable que hayamos definido dentro de nuestro guión podemos usar la función `getenv("VARIABLE")`

variablesHTTP.php

```
<html>
  <head><title>Resultado de los cálculos</title></head>
  <body>
    <?php
      $ip = getenv("REMOTE_ADDR"); // $_
      $userAgent = getenv("HTTP_USER_AGENT");
    ?>
    <p>Tu IP es: <?= $ip ?></p>
    <p>Estás usando: <?= $userAgent ?></p>
  </body>
</html>
```

Uso de variables: Trabajo con formularios

calculadora.html

```
<html>
  <head><title>Ejemplo de calculadora mediante formularios</title></head>
  <body>
     <form method="POST" action="calculos.php">
       <p>Primer valor: <input type="text" id="val1" name="val1" size="10" /></p>
       <p>Segundo valor: <input type="text" id="val2" name="val2" size="10" /></p>
       <p>Calculo a realizar</p>
       <input type="radio" id="calculo"
                           name="calculo" value="sum" />Suma<br />
       <input type="radio" id="calculo"
                           name="calculo" value="res" />Resta<br />
       <input type="radio" id="calculo"
                           name="calculo" value="mul" />Multiplicación<br />
       <input type="radio" id="calculo"
                           name="calculo" value="div" />División<br />
       <input type="submit" value="Calcula" />

     </form>
  </body>
</html>
```

Uso de variables: Trabajo con formularios (cont.)

calculos.php

```
<?php
  // Verifica que nos han pasado todos los datos
  if( ($_POST["val1"]=="")|| ($_POST["val2"]=="")|| ($_POST["calculo"]=="")){
    // Indicamos al browser que carge de nuevo la página HTML de la calculadora
    header("Location: calculadora.html");
    // Indicamos al procesador de PHP no siga adelante
    exit;
  }
  // Realizamos la operación
  if($_POST["calculo"]=="sum"){
    $result = $_POST["val1"] + $_POST["val2"];
  } else  if($_POST["calculo"]=="res"){
    $result = $_POST["val1"] - $_POST["val2"];
  } else  if($_POST["calculo"]=="mul"){
    $result = $_POST["val1"] * $_POST["val2"];
  } else  if($_POST["calculo"]=="div"){
    $result = $_POST["val1"] / $_POST["val2"];
  }
?>
<html>
  <head><title>Resultado de los cálculos</title></head>
  <body>
    <p>El resultado es <?= $result ?></p>
  </body>
</html>
```

Sesiones y Cookies

Tecnologías Web

Introducción

- Las sesiones y las cookies proporcionan la habilidad de "recordar" información sobre los usuarios.
- Se diferencian en como son capaces de almacenar dicha información
  - Las sesiones almacenan los datos en archivos temporales en el disco duro del servidor.
  - Las cookies son pequeños archivos que se almacenan en la máquina del cliente.
- Soporte de Sesiones en PHP
  - PHP3 no nos proporciona soporte nativo de sesiones.
    - Necesitamos usar la librería PHPLib (http://phplib.sourceforge.net)
  - PHP4 tiene soporte nativo de sesiones, además de ser más rápido que el soporte proporcionado por PHPLib

Cookies

- Existen dos tipos de cookies
  - Persistentes
    - La cookie es almacenada por el navegador
  - No Persistentes
    - Los valores de la cookie se pierden al cerrar el navegador.
- Las cookies son útiles para los programadores ya que permiten una manera fácil y segura de poder compartir variables que se necesitan en múltiples páginas.
- Las cookies son simples pares cadenas de texto "nombre=valor", que tienen asociada una URL.
  - El navegador usa esta URL para decidir si envía o no la cookie al servidor.
- Los navegadores WEB tienen control sobre las cookies
  - Los usuarios pueden desactivarlas
  - No aceptan almacenar más de 300 cookies, de las cuales como máximo 20 pueden ser del mismo servidor.

Consideraciones sobre la seguridad de las cookies

- Las cookies persistentes están mal vistas por los usuarios
- Las cookies no pueden acceder a ninguna información sobre el sistema del usuario.
- Las cookies tienen su propio sistema de seguridad que nos navegadores pueden usar.
- Las cookies están restringidas a un cierto ámbito o rango de direcciones, en el cual puede ser usada.
  - El programador define este ámbito
  - El navegador lee la información sobre este ámbito y determina si un servidor tiene acceso a dicha cookie.

Leyendo cookies

- Las cookies que llegan al servidor son leídas automáticamente por PHP.
- Podemos acceder a los valores almacenados en dichas cookies de diferentes maneras. Ej: cookie con nomre `"login"` y valor `"Ivan"`
  - `$login` → Como variable global. Sólo es posible si tenemos activadas el registro automático de variables globales (`register_globals`).
  - `$HTTP_COOKIE_VARS["login"]` → En este array global se almacenan todas las variables que nos envía el navegador dentro de las cookies. Es más seguro utilizar este método de acceso que el anterior.
  - `$_COOKIE["login"]` → Array superglobal (es accesible dentro de funciones). Tiene las mismas ventajas que el método anterior.

Ejemplo de lectura de una cookie

```
<html>
  <head><title>Bienvenido a nuestro sistema</title></head>
  <body>
    <?php echo "<h1>Hola $_COOKIE['login']</h1>"; ?>
  </body>
</html>
```

Estableciendo cookies

- Para establecer una cookie tenemos que usar la función `setcookie()`.
  - Esta función le dice a un navegador que recuerde el nombre y valor de una cookie y lso envíe de vuelta al servidor en peticiones posteriores.
- Siempre tenemos que llamar a `setcookie()` antes de que la página genere cualquier resultado.
  - Esto signitica que `setcookie()` tienen que venir antes de cualquier sentencia `echo`, `print`.
  - También significa que no puede haber ningún texto delante del delimitador PHP.

Ejemplo de escritura de una cookie

```
<?php setcookie("login", "ivan"); ?>
<html>
  <head><title>Bienvenido a nuestro sistema</title></head>
  <body>
    <?php echo "<h1>Hola $_COOKIE['login']</h1>"; ?>
  </body>
</html>
```

Función `setcookie()`

```
int setcookie(string nombre [,string valor] [,string
   tiempoDeVida] [, string ruta] [,string dominio] [,
   integer seguridad])
```

- `nombre` → Nombre de la cookies a establecer.
- `valor` → Valor de la cookie a establecer.
- `tiempoDeVida` → Tiempo de vida a partir del cual los datos de la cookie no serán válidos.
- `ruta` → Ruta absoluta (desde la raíz del servidor) a partir de la cual la cookie es accesible. Esta ruta es recursiva y permite que todos sus subdirectorios puedan acceder a la cookie.
- `dominio` → El dominio a partir del cual la cookie es accesible
- `seguridad` → Este valor indica si la cookie es accesible fuera de una petición HTTPS. El valor por defecto es 0, indicando que las peticiones HTTP normales pueden acceder a los datos de la cookie.

Si omitimos el valor de la cookie, ésta será eliminada

Estableciendo el tiempo de vida de una cookie

- El tiempo de vida de una cookie es relativo al número de segundos que han pasado desde el 1 de Enero de 1970. Por tanto el tiempo de expiración de la cookie tiene que ser relativo a esta fecha.
- PHP nos proporciona un conjunto extenso de funciones para el tratamiento de fechas, en particular `time()` nos proporciona el número de segundos que han transcurrido desde 1 de Enero de 1970 hasta la fecha y momento en el que se realiza la llamada a dicha función.
- Si omitimos este parámetro la cookie desparecerá al cerrar el cliente el navegador.

Ejemplo de escritura de una cookie con tiempo de vida

```
<?php
  // Esta cookie expirará en media hora
  setcookie("mi_cookie", $value, time() 60*60);
  // Esta cookie caduca en un día
  setcookie("login", "ivan", 60*60*24);
  // Esta cookie caduca a las 0h:00m:00s del 20 de Mayo de 2006
  setcookie("mi_cookie2", $value2, mktime(0,0,0,05,20,2006);)
?>
```

Estableciendo el ámbito de una cookie

- Con los parámetros `"ruta"` y `"dominio"` controlamos el ámbito donde puede ser accesible una cookie.
- La ruta de acceso por defecto de una cookie es "/" lo que quiere decir que la cookie es accesible por cualquier subdirectorio del raiz.
- Para limitar el acceso a la cookie a un subdirectorio tenemos que usar como parámetro la ruta a dicho subdirectorio
  - Ej: `/user/`
  - NOTA: Si establecemos `"/user"` como parámetro, la cookie podrá ser accesible por `/user.php`, `/user1/index.html`.
- Para limitar el acceso a la cookie a un único archivo tenemos que establecer como parámetro la ruta al archivo.
  - Ej: `/user/scrip.php`
  - NOTA: Podemos sufrir los problemas mencionados en el apartado anterior.

Estableciendo el ámbito de una cookie (cont.)

- Podemos limitar que dominios pueden acceder a una cookie.
- Para comprobar si un dominio puede acceder se comprueba si el nombre del dominio que quiere acceder a la cookie encaja por el final con el valor de dominio que hemos establecido en la cookie.
  - `dominio.com` → podrán acceder a la cookie el dominio `www.dominio.com`, `midominio.com`, pero no podrán acceder `dominio.org`.
- Si lo que queremos es restringir el acceso a la cookie a varios servidores del mismo dominio, por e.j. `www1.midominio.com`, `www2.midominio.com`, tenemos que establecer como parámetro de dominio la cadena `".midominio.com"`.

Ejemplo de escritura de una cookie con restricción de ámbito

```
<?php
  setcookie("micookie", $valor, time() + 3600, ".dominio.com");
?>
```

Uso de se Sesiones en PHP

- Normalmente, las variables son destruidas por defecto cuando el guión PHP ha terminado su ejecución.
- Esto permite liberar memoria y nos permite reutilizar nombres de variables.
- Ciclo de vida de una sesión
  - Normalmente la sesión se inicia cuando un usuario visualiza una página de un sitio WEB
  - Termina ya sea por expirar la sesión llegado un tiempo límite (configurado en el `php.ini`), o por terminarla de manera explícita dentro de un guión PHP.
- Las sesiones utilizan una cookie llamada `PHPSESSID`.
  - Cuando se inicia una sesión en una página, el intérprete PHP comprueba la presencia de esta cookie y la establece si no existe.
  - El identificador de sesión en la cookie `PHPSESSID` identifica ese cliente WEB de forma única en el servidor.

Trabajando con sesiones PHP

- Comenzando una sesión
  - Para utilizar una sesión en una página, tenemos que llamar a la función `session_start()` al principio del guión.
- Almacenar y recuperar información
  - Podemos almacenar y recuperar información de las sesiones de múltiples maneras.
  - PHP < 4.0 (no utilizar)

    ```
    boolean session_register(string varname [, string varname])
    ```

    - Almacena las variables cuyos nombres le pasamos como argumento dentro de la sesión y las registra como variables globales.
    - Para acceder a las variables usamos el array `$GLOBALS` o el propio nombre de variable si tenemos activa la directiva `register_globals`.
  - PHP >= 4.0 (recomendada)
    - Utilizamos el array `$_SESSION` o el array `$HTTP_SESSION_VARS` para establecer y leer un valor.
    - Cuando añadimos / modificamos una entra de estos arrays se queda almacenado en la sesión del usuario.

Ejemplo de sesiones

Ejemplo de formulario con sesiones

```
<?php
  require "form_utils.php"
  session_start();

  $platos = array(  'Ensalda' => 'Ensalda de tomate y lechuga',
                    'Pollo' => 'Pollo a la plancha con patatas fritas',
                    'Escalope' => 'Escalope empanado');
  if($_POST['_submit_check']){
    if($firm_errors = validate_form()){
      show_form($form_errors);
    }else{
      process_form();
    }
  }else{
    show_form();
  }

  ...
```

Ejemplo de sesiones (cont.)

Ejemplo de formulario con sesiones (cont.)

```
function show_form($errors=''){
  echo '<form method="POST" action=".$_SERVER['PHP_SELF'].'">';
  if($errors){
    print '<ul></li>';
    echo implode('</li><li>', $errors);
    print '</li></ul>';
  }
  echo "Plato: ";
  input_select('plato',$_POST, $GLOBALS['platos']);
  echo "<br />";

  echo "Cantidad: ";
  input_text('cantidad', $_POST);
  echo "<br />";

  input_submit('submit', 'realizar pedido');

  echo '<input type="hidden" name="_submit_check" value="1" />';
  echo '</form>';
}
```

Ejemplo de sesiones (cont.)

Ejemplo de formulario con sesiones (cont.)

```
  function validate_form() {
    $errors = array();

    if(! array_exists($_POST['plato'], $GLOBALS['platos'])){
      $erros[] = "Por favor, seleccione un plato";
    }

    if ( ! is_numeric($_POST['cantidad']) || intval($_POST['cantidad']) <= 0){
      $errors[] = "Introduzca una cantidad valida";
    }
    return $errors;
  }

  function process_form(){
    $_SESSION['pedido'][] = array ( 'plato' => $_POST['plato'],
                                    'catidad' => $_POST['cantidad']);

    echo 'Gracias por su pedido';
  }
?>
```

Ejemplo de sesiones (cont.)

Impresión de datos de la sesión

```
<?php
  session_start();
  $platos = array(  'Ensalda' => 'Ensalda de tomate y lechuga',
                    'Pollo' => 'Pollo a la plancha con patatas fritas',
                    'Escalope' => 'Escalope empanado');

  if(count($_SESSION['pedido']) > 0 ){
    echo '<ul>';
    foreach($_SESSION['pedido'] as $pedido){
      $nombre_plato = $platos[$pedido['plato']];
      echo "<li> $pedido[cantidad] de $nombre_plato"
    }
    echo "</ul>";
  }else{
    echo "No ha realizado ningún pedido":
  }
?>
```

Configurar sesiones

- Los datos de sesión permanecen mientras se acceda a la sesión al menos una vez cada 24 minutos (valor por defecto de configuración).
- A veces se puede necesitar una duración de sesión más corta.
- Para configurar el tiempo máximo de vida de una sesión podemos establecer el valor de la directiva de configuración `'session.gc_maxlifetime'`.
  - Hay que usar la función `ini_set(string varConfig, valor)` para establecer el valor.
  - La llamada a `init_set()` hay que realizarla antes de llamar a `session_start()`.
- Cuando indicamos que deseamos borrar una sesión no se borrar inmediatemente, sino que existe una probabilidad de que el intérprete PHP elija eliminar sesiones caducadas.
  - La probabilidad por defecto es del 1%
  - Podemos configurar esta probabilidad con el parámetro `'session.gc_probability'`

Utilidades de formulario

form_utils.php

```
function input_text($element_name, $values){
  echo '<input type="text" name="'.$element_name.'" value="';
  echo htmlentities($values[$element_name]) .'" />';
}

function input_submit($element_name, $label){
  echo '<input type="submit" name="'.$element_name.'" value="';
  echo htmlentities($label) .'" />';
}

function input_textarea($element_name, $values){
  echo '<textarea name="'.$element_name.'" >';
  echo htmlentities($values[$element_name]) .'</textarea>';
}

function input_radiocheck($type, $element_name, $values, $element_value){
  echo '<input type="'.$type.'" name="'.$element_name;
  echo '" value="'.$element_value.'" ';
  if($element_value== $values[$element_name]){
    echo ' checked="checked"';
  }
  echo ' />';
}
```

Utilidades de formulario (cont.)

form_utils.php (cont.)

```
function input_select($element_name, $selected, $options, $multiple = false){
  echo '<select name="'.$element_name;
  if( $multiple ){
    echo '[]" multiple="multiple';
  }
  echo '" />';

  $selected_options = array();
  if($multiple){
    foreach( $selected[$element_name] as $val){
      $selected_options[$val] = true;
    }
  }else{
    $selected_options[$selected[$element_name] ] = true;
  }

  foreach($options as $option => $label ){
    echo '<option value="'.htmlentities($option).'"';
    if($selected_options[$option]){
      echo ' selected="selected"';
    }
    echo '>' . htmlentities($label).'</option>';
  }

  echo '</select>';
}
```

Acceso al sistema de ficheros desde PHP

Tecnologías Web

Introducción

- Las aplicaciones WEB trabajan con datos persistentes. Normalmente existen dos opciones para almacenar datos
  - El sistema de ficheros del servidor
  - Uso de una base de datos relacional
- Normalmente se usa una BBDD relacional en el caso de que la información que deseamos almacenar sea estructurada o la aplicación necesite acceder a dicha información según una cierta condición.
- El sistema de ficheros es útil para almacenar información como la configuración, etc.

Archivos

- Un archivo es una secuencia de bytes que está almacenado de manera persistente en un medio físico como un disco duro.
- Un archivo es un sistema de almacenamiento de datos secuencial.
  - El acceso (lectura) a un archivo lo hacemos mediante un cursor que nos indica la posición actual de lectura.
  - Este cursor siempre avanza en una dirección (hacia el final del archivo), aunque también podemos hacer que vuelva a una posición anterior.
- Cada archivo está unívocamente identificado por su ruta absoluta.
  - Ej: c:\windows\temp\prueba.txt, /home/user/prueba.txt
- En Windows tanto la barra '/' como '\' pueden ser usadas como separadores en una ruta de archivo, pero en otros sistemas operativos puede no darse el caso.
  - Es un buen hábito el definir una variable global `pathSeparator`, que almacene el separador para las rutas de archivos, y que usemos dicho separador para construir las rutas.

Abriendo y cerrando archivos

- La función `fopen()` nos permite abrir cualquier archivo en el sistema de ficheros del servidor.
  - También nos permite abrir archivos usando una conexión http o ftp.

```
int fopen( string ruta, string modo)
```

  - `ruta` → Ruta absoluta o relativa (al directorio de trabajo) al archivo que queremos abrir.
  - `modo` → Con este parámetro indicamos que tipo de operaciones deseamos realizar sobre el archivo, e.j, lectura, lectura y escritura, escritura.
    - `"R"` → Sólo operaciones de lectura
    - `"r+"` → Operaciones de lectura y escritura
    - `"W"` → Sólo operaciones de escritura. Si el archivo existía, el contenido previamente almacenado se pierde. Si el archivo no existía se crea.
    - `"w+"` → Operaciones de lectura y escritura. Si el archivo existía, el contenido previamente almacenado se pierde. Si el archivo no existía se crea
    - `"A"` → Operaciones de anexión de datos. Los datos nuevos se añaden al final del archivo. Si el archivo no existía se crea
    - `"a+"` → Operaciones de anexión y lectura de datos. Los datos nuevos se añaden al final del archivo. Si el archivo no existía se crea.
    - `"B"` → Este flag indica que vamos a leer/escribir sobre un fichero que contiene datos binarios.

Abriendo y cerrando archivos (cont.)

- La función `fopen()` nos devuelve el manejador del archivo si la operación a tenido éxito, o `false` en caso de que no.
  - Tenemos que almacenar este manejador que nos devuelve ya que el resto de operaciones que realicemos sobre el archivo tenemos que pasarlo como argumento.

Ejemplo de apertura un fichero

```
<?php
  if(!($fichero=fopen("imagen.jpg", "rb"))){
    printf("No se puede abrir el archivo imagen.jpg");
  }
?>
```

- Para cerrar un archivo tenemos que usar la función `fclose()`
  - `int fclose(int manejador)`
    - `fclose()` devuelve `true` en el caso de que la operación haya tenido éxito y `false` en otro caso.

Lectura y escritura de archivos

- Podemos volcar un archivo directamente a la salida estándar (salida que será visualizada por el navegador) usando la función `fpassthru()`.

  `int fpassthru(int manejador)`

  - `manejador` → Manejador del archivo a volcar
  - La función devuelve `true` en el caso de que la operación haya tenido éxito y `false` en otro caso.

  - NOTA: En el caso de querer volcar un archivo binario, no tenemos que olvidarnos de activar el flag `"b"` en el modo.

- En el caso de archivos de texto, también podemos volcar un archivo usando la función `readfile(string ruta)`.

  - En este caso no tenemos que realizar la apertura del archivo previamente.

Lectura y escritura de archivos (cont.)

- Lectura de archivos

  `string fread(int manejador, int long)`

  - Devuelve una cadena de cómo mucho `long` caracteres cuyo contenido es la información leída del archivo. Además avanza el cursor del archivo.

  `string fgetc(int manejador)`

  - Lee un carácter del archivo y lo devuelve, además avanza el cursor del archivo.

  `string fgets(int manejador, int long)`

  - Lee desde la posición actual del cursor hasta que lee `long-1` bytes, encuentra un salto de línea o el archivo termina devolviendo lo que ha leído en una cadena.

  `string fgetss(int manejador, int long [, string tags])`

  - `fgetss()` funciona igual que `fgets()`, que si aparece cualquier etiqueta HTML o delimitador PHP, estos son eliminados.
  - El argumento `tags` nos permite especificar mediante una lista de etiquetas separadas por comas, todas aquellas etiquetas que no queremos que PHP elimine de la cadena resultante.

Lectura y escritura de archivos (cont.)

`array file(string ruta)`

- Lee de un archivo todas las líneas de texto que contiene y las devuelve en un array de cadenas.
- Las cadenas de texto que nos devuelven aún contiene los caracteres de salto de línea y retorno de caro (`"\n\r"`) al final.

`string file_get_contents(string ruta)`

- Lee todo el contenido del y lo devuelve en una cadena.

- Escritura de archivo

```
int fputs(int manejador, string cadena [,int long])
int fwrite(int manejador, string cadena [,int long])
```

  - Las dos funciones son idénticas
  - `manejaor` → Es el manejador del archivo sobre el que queremos escribir.
  - `cadena` → Es la cadena de texto que queremos escribir en el archivo.
  - `long` → Si se incluye este parámetro, nos indica el número de caracteres a escribir de la cadena que le pasamos como parámetro.

Control y navegación de archivos

- Cuando usamos una función de lectura sobre un archivo, su cursor interno va avanzando.

`int rewind(int manejador)`

- Posiciona el cursor del archivo al comienzo del mismo.

`int fseek(int manejador, int desp [, int sentido] )`

- Permite mover el cursor del archivo a cualquier posición.
- `desp` → Indica el número de caracteres que avanzaremos (desp > 0) o que queremos retroceder (desp < 0).
- `sentido` → Puede tomar los siguientes valores
  - `SEEK_SET` → Establece la posición del cursor a `desp` bytes desde el comienzo del archivo.
  - `SEEK_CUR` → Establece la posición del cursor desde la posición actual más el valor de `desp`.
  - `SEEK_END` → Establece la posición del cursor al final del archivo más el valor de `desp`.

Control y navegación de archivos (cont.)

`int ftell(int manejador)`

- Nos devuelve la posición actual del cursor.
- NOTA: Podemos guardar esta posición para posteriormente establecerla con `fseek()`.

- `int feof(int manejador)`
  - Devuelve `true` en el caso de que hayamos llegado al final del archivo y `false` en otro caso.

Ejemplo típico de lectura de archivo

```
<?php
  while(! feof($manejador)){
    $buffer = fgets($manejador, 1024);
    // Procesamiento de los datos
  }
?>
```

Trabajo con el sistema de ficheros

- PHP soportar funciones para trabajar directamente con el sistema de ficheros donde se encuentra alojado el servidor. Podemos copiar, renombrar y borrar archivo.

`int copy(string origen, string destino)`

- Copia el acrhivo con ruta `origen` al archivo con ruta `destino`.
- Devuelve `true` en el caso de tenga éxito y `false` en otro caso.

`int rename(string antiguo, string nuevo)`

- Renombra el acrhivo con ruta `origen` al archivo con ruta `destino`.
- Devuelve `true` en el caso de tenga éxito y `false` en otro caso.
- Como especificamos la ruta, además de renombrar el acrhivo también podemos mover archivos.

`int ulink(string archivo)`

- Elimina el archivo de manera permanente.
- Devuelve `true` en el caso de tenga éxito y `false` en otro caso.

Trabajando con los atributos de ficheros

`int file_exists(string archivo)`

- Verifica si existe un archivo con ruta `archivo`.
- Devuelve `true` en el caso de que exista y `false` en otro caso.

`int fileatime(string archivo)`

- Devuelve la fecha y hora del último acceso del archivo.

`int filectime(string archivo)`

- Devuelve la fecha y hora del última vez que se modificó el archivo, tanto sus datos como información de permisos.

`int filemtime(string archivo)`

- Devuelve la fecha y hora del última vez que se modificó el archivo.

`int filesize(string archivo)`

- Devuelve el tamaño del archivo en bytes.

Trabajando con los atributos de ficheros (cont.)

- Para verificar el tipo de archivo podemos usar las siguientes funciones

```
int is_dir(string archivo)
int is_executable(string archivo)
int is_file(string archivo)
int is_link(string archivo)
```

- Además también podemos verificar otros atributos de un fichero.

```
int is_readable(string archivo)
int is_writable(string archivo)
```

Trabajando con directorios

- Además de trabajar con ficheros directamente, PHP proporciona funciones para tratar con directorios.

`int chdir(string rutaDir)`

- Permite establecer como directorio de trabajo del proceso actual que está ejecutando el guión PHP al directorio cuya ruta es `rutaDir`.

`string getcwd()`

- Devuelve la ruta al directorio de trabajo actual del proceso PHP.

`int opendir(string rutaDir)`

- Abre un directorio para poder realizar operaciones sobre él, `opendir()` devuelve el manejador al directorio.

`string readdir(int manejadorDir)`

- Permite iterar sobre el contenido de `manejadorDir`, la función devuelve el nombre de la siguiente entrada en el directorio. Si no hay más entradas en el directorio, o el manejador es incorrecto, la función devuelve `false`.

Trabajando con directorios (cont.)

- Además de las entradas que existan dentro de un directorio, existen dos especiales `"."` y `".."`.
  - `"."` → Representa el directorio actual.
  - `".."` → Representa al directorio padre.

`int mkdir(string rutaDir, int modo)`

- Permite crear un directorio en la ruta `rutaDir`.
- Adicionalmente podemos establecer sus permisos (solo UNIX) usando el parámetro `modo`.
- La función devuelve `true` cuando la operación tenga éxito y `false` en caso de fallo.

`int rmdir(string rutaDir)`

- Elimina el directorio en la ruta `rutaDir`.
- La función devuelve `true` cuando la operación tenga éxito y `false` en caso de fallo.

Subiendo archivos al servidor usando PHP

- Cuando subimos archivos al servidor, estos archivos se suben a un directorio temporal.
- Una vez que el archivo o archivos han sido subidos, podemos acceder a la información de dichos archivos usando la variable global `$HTTP_POST_FILES[]` o la variable superglobal `$_FILE[]`.
  - `$_FILE['campoFile']['name']` → Contiene el nombre original del archivo en la máquina del cliente.
  - `$_FILE['campoFile']['type']` → contiene le tipo MIME del archivo.
  - `$_FILE['campoFile']['size']` → Almacena el tamaño, en bytes, del archivo.
  - `$_FILE['campoFile']['tmp_name']` → Almacena la ruta al archivo en el directorio temporal
- Hay que tener cuidado cuando subimos archivos debido a su tamaño. La directiva PHP `upload_max_filesize` (por defecto 2MB) almacena el tamaño máximo de subida de un archivo. Cualquier intento de subir un archivo de mayor tamaño fallará.

Subiendo archivos al servidor usando PHP (cont.)

subir.php

```
<?php
  function show_form($errors=''){
    $salida = <<<_HTML_
  <form action="$_SERVER[PHP_SELF]" method="post" enctype="multipart/form-data">
    <p>Archivo: </p><input type="file" name="archivo" /><br />
    <input type="submit" name="submit" />
    <input type="hidden" name="_submit_check" value="1" />
  </form>
_HTML_;
    echo $salida;
  }
  function process_form(){
    echo "<b>Detalles del archivo subido</b><br><br>";
    echo "<p>Nombre: {$_FILE[archivo][name]}</p>";
    echo "<p>Ruta temporal: {$_FILE[archivo][tmp_name]}</p>";
    echo "<p>Tamaño: {$_FILE[archivo][size]}</p>";
    echo "<p>Tipo: {$_FILE[archivo][type]}</p>";
    if( copy($_FILE["archivo"]["tmp_name"], "c:/ruta/".$_FILE["archivo"]["name"]) ){
      echo "<p>El archivo se ha copiado correctamente</p>";
    }else{
      echo "<p>El archivo se ha copiado correctamente</p>";
    }
  }
?>
```

Trabajar con archivos CSV

- Los archivos *Comma Separated Value* (CSV), son archivos utilizados para el intercambio de información entre aplicaciones, e.j Excel o Access.
  - Para leer información de este tipo de archivos usamos la función `fgetscsv(int manejador, int longMaxLinea)`.

datos.txt

```
"Pollo con patatas", 4.25
"Ensalada mixta", 2.25
"Entrecot de ternera", 7.75
```

guion.php

```
$ar = fopen("datos.txt", "r");
// Suponemos que la longitud de la línea de mayor tamaño
// es 1024 caracteres
for( $info = fgetscvs($ar, 1024); !feof($ar); $info = fgetscvs($ar, 1024) ){
  //$info[0] almacena el nombre del palto
  //$info[1] almacena el precio del plato
}
```

Trabajar con archivos CSV (cont.)

- No existe una función predefinida para generar líneas de un archivo CSV, tenemos que crearla nosotros mismos.

Escritura en un archivo CSV

```
function crea_linea_csv($valores){

  foreach($valores as $i => ·valor){
    if((strpos($valor, ',')  !== false) ||
       (strpos($valor, '"')  !== false) ||
       (strpos($valor, ' ')  !== false) ||
       (strpos($valor, "\t") !== false) ||
       (strpos($valor, "\n") !== false) ||
       (strpos($valor, "\r") !== false)) {
      $valores[i]='"' . Str_replace('"', '""', $valor) . '"';
    }
  }

  // Unimos todos los valores del array separándolos por comas
  return implode(',', $valores) . "\n";
}
```

Críticas, dudas y sugerencias...

Federico Peinado
www.federicopeinado.es