

U.T. 2: Características del lenguaje PHP

Contenidos:

- Elementos del lenguaje PHP
 - Generación de código HTML
 - Cadenas de texto
 - Funciones relacionadas con los tipos de datos
 - Variables especiales de PHP
- Estructuras de control
 - Condicionales
 - Bucles
- Funciones
 - Inclusión de ficheros externos
 - Ejecución y creación de funciones
 - Argumentos
- Tipos de datos compuestos
 - Recorrer arrays
 - Funciones relacionadas con los tipos de datos compuestos
- Formularios web
 - Formularios
 - Subida de ficheros al servidor
 - Procesamiento de la información devuelta por un formulario web
 - Generación de formularios web en PHP
- Cabeceras HTTP: Introducción
- Ficheros y directorios
 - Apertura y cierre
 - Procesamiento de ficheros
 - Manejo de directorios

1.- Elementos del lenguaje PHP

En la unidad anterior, aprendiste a preparar un entorno para programar en PHP. Además también viste algunos de los elementos que se usan en el lenguaje, como las variables y tipos de datos, comentarios, operadores y expresiones.

También sabes ya cómo se integran las etiquetas HTML con el código del lenguaje, utilizando los delimitadores **<?php** y **?>**.

En esta unidad aprenderás a utilizar otros elementos del lenguaje que te permitan crear programas completos en PHP. Los programas escritos en PHP, además de encontrarse estructurados normalmente en varias páginas (ya veremos más adelante cómo se pueden comunicar datos de unas páginas a otras), suelen incluir en una misma página varios bloques de código. Cada bloque de código debe ir entre delimitadores, y en caso de que genere alguna salida, ésta se introduce en el código HTML en el mismo punto en el que figuran las instrucciones en PHP.

Por ejemplo, en las siguientes líneas tenemos dos bloques de código en PHP:

```
<body>
<?php $a=1; ?>
<p>Página de prueba</p>
<?php $b=$a; ?>
...
```

Aunque no se utilice el valor de las variables, en el segundo bloque de código la variable `$a` mantiene el valor 1 que se le ha asignado anteriormente.

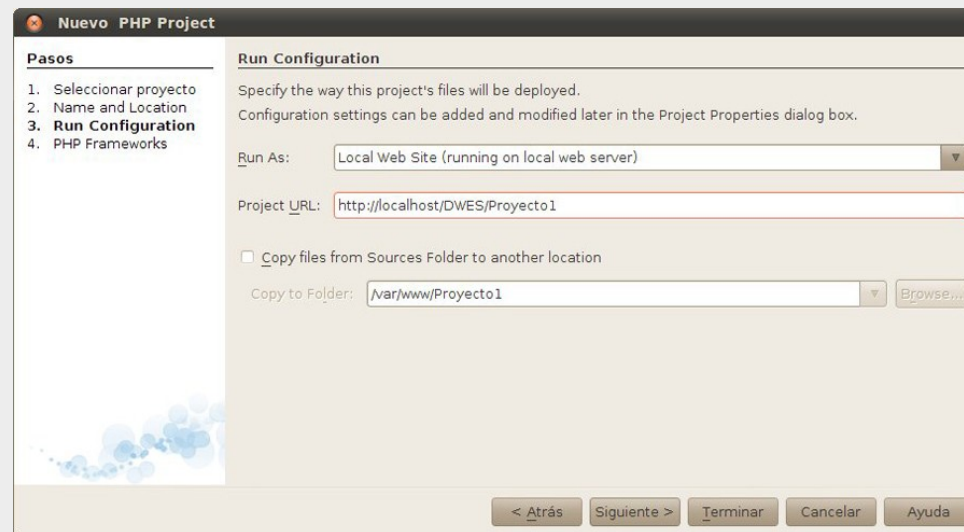
1.- Elementos del lenguaje PHP

En esta unidad empezarás a crear tus propios programas en PHP. Para ello vas a usar el IDE NetBeans, que instalaste anteriormente. Deberías organizar tus programas en proyectos, y almacenarlos en una estructura en árbol, colgando todos por ejemplo de /home/usuario/NetBeansProyectos/DWES. Para crear un proyecto nuevo vete a Archivo – Proyecto Nuevo y selecciona PHP Application.

En la siguiente pantalla de configuración del proyecto, debes indicar la ruta que se usará para acceder al mismo desde un navegador. Es decir, tienes que hacer que el servidor web Apache pueda acceder a la ruta anterior en la que vas a almacenar tus proyectos. Esto puedes hacerlo, por ejemplo, creando un enlace simbólico en la raíz del servidor web (/var/www):

```
sudo ln -s /home/usuario/NetBeansProyectos/DWES/ DWES
```

De esta forma, si creas un proyecto nuevo en la ruta /home/usuario/NetBeansProyectos/DWES/Proyecto1, la URL que tendrás que poner en la siguiente pantalla de configuración será http://localhost/DWES/Proyecto1.



1.1.- Generación de código HTML

Existen varias formas de incluir contenido en la página web a partir del resultado de la ejecución de código PHP. La forma más sencilla es usando echo, que no devuelve nada (void), y genera como salida el texto de los parámetros que recibe.

```
void echo (string $arg1, ...);
```

Otra posibilidad es print, que funciona de forma similar. La diferencia más importante entre print y echo, es que print sólo puede recibir un parámetro y devuelve siempre 1.

```
int print (string $arg);
```

Tanto print como echo no son realmente funciones, por lo que no es obligatorio que pongas paréntesis cuando las utilices.

Consulta el Anexo I: Utilización de la función print en PHP

1.1.- Generación de código HTML

printf es otra opción para generar una salida desde PHP. Puede recibir varios parámetros, el primero de los cuales es siempre una cadena de texto que indica el formato que se ha de aplicar. Esa cadena debe contener un especificador de conversión por cada uno de los demás parámetros que se le pasen a la función, y en el mismo orden en que figuran en la lista de parámetros. Por ejemplo:

```
<?php
    $ciclo="DAW";
    $modulo="DWES";
    print "<p>";
    printf("%s es un módulo de %d curso de %s", $modulo, 2, $ciclo);
    print "</p>";
?>
```

Cada especificador de conversión va precedido del carácter % y se compone de las siguientes partes:

- ✓ **signo** (opcional). Indica si se pone signo a los número negativos (por defecto) o también a los positivos (se indica con un signo +).
- ✓ **relleno** (opcional). Indica que carácter se usará para ajustar el tamaño de una cadena. Las opciones son el carácter 0 o el carácter espacio (por defecto se usa el espacio).
- ✓ **alineación** (opcional). Indica que tipo de alineación se usará para generar la salida: justificación derecha (por defecto) o izquierda (se indica con el carácter -).
- ✓ **ancho** (opcional). Indica el mínimo número de caracteres de salida para un parámetro dado.
- ✓ **precisión** (opcional). Indica el número de dígitos decimales que se mostrarán para un número real. Se escribe como un dígito precedido por un punto.
- ✓ **tipo** (obligatorio). Indica cómo se debe tratar el valor del parámetro correspondiente. En la siguiente tabla puedes ver una lista con todos los especificadores de tipo.

1.1.- Generación de código HTML

Especificadores de tipo para las funciones printf y sprintf.

Especificador	Significado
b	el argumento es tratado como un entero y presentado como un número binario.
c	el argumento es tratado como un entero, y presentado como el carácter con dicho valor ASCII.
d	el argumento es tratado como un entero y presentado como un número decimal.
u	el argumento es tratado como un entero y presentado como un número decimal sin signo.
o	el argumento es tratado como un entero y presentado como un número octal.
x	el argumento es tratado como un entero y presentado como un número hexadecimal (con minúsculas).
X	el argumento es tratado como un entero y presentado como un número hexadecimal (con mayúsculas).
f	el argumento es tratado como un doble y presentado como un número de coma flotante (teniendo en cuenta la localidad).
F	el argumento es tratado como un doble y presentado como un número de coma flotante (sin tener en cuenta la localidad).
e	el argumento es presentado en notación científica, utilizando la e minúscula (por ejemplo, 1.2e+3).
E	el argumento es presentado en notación científica, utilizando la e mayúscula (por ejemplo, 1.2E+3).
g	se usa la forma más corta entre %f y %e.
G	se usa la forma más corta entre %f y %E.
s	el argumento es tratado como una cadena y es presentado como tal.
%	se muestra el carácter %. No necesita argumento..

Por ejemplo, al ejecutar la línea siguiente, en la salida el número PI se obtiene con signo y sólo con dos decimales.

```
printf("El número PI vale %+.2f", 3.1416); // +3.14
```

Existe una función similar a printf pero en vez de generar una salida con la cadena obtenida, permite guardarla en una variable: sprintf.

```
$txt_pi = sprintf("El número PI vale %+.2f", 3.1416);
```

1.2.- Cadenas de texto

En PHP las cadenas de texto pueden usar tanto comillas simples como comillas dobles. Sin embargo hay una diferencia importante entre usar unas u otras. Cuando se pone una variable dentro de unas comillas dobles, se procesa y se sustituye por su valor. Así, el ejemplo anterior sobre el uso de print también podía haberse puesto de la siguiente forma:

```
<?php
$modulo="DWES";
print "<p>Módulo: $modulo</p>"
?>
```

La variable \$modulo se reconoce dentro de las comillas dobles, y se sustituye por el valor "DWES" antes de generar la salida. Si esto mismo lo hubieras hecho utilizando comillas simples, no se realizaría sustitución alguna.

Para que PHP distinga correctamente el texto que forma la cadena del nombre de la variable, a veces es necesario rodearla entre llaves.

```
print "<p>Módulo: ${modulo}</p>"
```

Cuando se usan comillas simples, sólo se realizan dos sustituciones dentro de la cadena: cuando se encuentra la secuencia de caracteres \', se muestra en la salida una comilla simple; y cuando se encuentra la secuencia \\, se muestra en la salida una barra invertida.

Estas secuencias se conocen como secuencias de escape. En las cadenas que usan comillas dobles, además de la secuencia \\, se pueden usar algunas más, pero no la secuencia \'. En esta tabla puedes ver las secuencias de escape que se pueden utilizar, y cuál es su resultado.

1.2.- Cadenas de texto

Secuencias de escape.

Secuencia	Resultado
\\	se muestra una barra invertida.
\'	se muestra una comilla simple.
\"	se muestra una comilla doble.
\r	se muestra un retorno de carro (CR o 0x0D (13) en ASCII).
\v	se muestra un tabulador vertical (VT o 0x0B (11) en ASCII).
\f	se muestra un avance de página (FF o 0x0C (12) en ASCII).
\\$	se muestra un signo del dólar.

En PHP tienes dos operadores exclusivos para trabajar con cadenas de texto. Con el operador de concatenación punto (.) puedes unir las dos cadenas de texto que le pases como operandos. El operador de asignación y concatenación (.=) concatena al argumento del lado izquierdo la cadena del lado derecho.

```
<?php
```

```
$a = "Módulo ";
```

```
$b = $a . "DWES"; // ahora $b contiene "Módulo DWES"
```

```
$a .= "DWES"; // ahora $a también contiene "Módulo DWES"
```

```
?>
```


1.2.- Cadenas en PHP: Búsqueda

- **strstr(cadena, cadBusq)**

- Devuelve desde la primera aparición de la cadena (**incluyéndola**) hasta el final. Case-sensitive

```
strstr("yo@midominio.es", "@");
```

```
// @midominio.es
```

- **strrchr(cadena, carBusq)**

- Devuelve desde la **última** aparición del 1o carácter (**incluído**) hasta el final. Case-sensitive

```
strrchr ("Esto es muy bonito", "so"); // s muy bonito
```

- **stristr(cadena, carBusq)**

- Igual que strstr pero no es case-sensitive.

1.2.- Cadenas en PHP: Búsqueda

- **strpos (cad1, cad2 [, desplz])**
 - Busca la primera posición de aparición de una cadena a partir de desplz (por defecto 0).
 - Si no la encuentra devuelve FALSE

```
strpos ("Este espacio es muy bonito","es",7); // 13
```
- **strrpos (cadena, cadena [, desplz])**
 - Devuelve la posición de la última aparición o FALSE

```
strrpos ("Este espacio es muy bonito","es"); // 13
```

1.2.- Cadenas en PHP: Búsqueda

- **strspn (cadena, máscara,comienzo,longitud)**

- Devuelve la longitud del segmento inicial de un string que consiste únicamente en caracteres contenidos dentro de una máscara dada .

Case-sensitive

```
strspn ("Este espacio es muy bonito","Estela"); // 4
```

```
strspn ("Este espacio es muy bonito","Estado"); // 3
```

```
strspn ("Este espacio es muy bonito","el");// 0
```

- **strcspn (cadena, máscara, comienzo, longitud)**

- Devuelve la longitud de la subcadena más larga que está formada sólo por caracteres no contenidos en la máscara. Case-sensitive

- **preg_match (patron, cadena)**

- Devuelve 1 si la cadena cumple el patrón (expresión regular) y 0 si no.
- Hay que prestar atención porque los valores devueltos NO SON BOOLEANOS sino que son numéricos.

1.2.- Cadenas en PHP: Comparación

- `strcmp(cad1, cad2)`

- Compara dos cadenas (case-sensitive) y devuelve:

- > 0 , si $cad1 > cad2$
 - < 0 , si $cad1 < cad2$
 - 0 , si ambas cadenas son iguales.

```
strcmp("Hola","hola") → no iguales  
strcasecmp("Hola","hola") → iguales
```

- `strcasecmp(cad1, cad2)`

- Igual que `strcmp()` pero insensible a mayúsculas y minúsculas.

- `strncmp(cad1, cad2, longitud)`

- Sólo compara los longitud primeros caracteres

- `strnatcmp(cad1, cad2)`

- Comparación "natural"

- con `strcmp(10,2)` 10 es menor que 2
 - con `strnatcmp(10,2)` 10 es mayor que 2

```
strncmp("Paco","Paca",4) → distintos  
strncmp("Paco","Paca",2) → iguales
```

1.2.- Cadenas en PHP: Operación

- **int strlen (string \$cadena)**

- Devuelve el número de caracteres que contiene una cadena.

```
$cadena="Hola que tal";  
echo strlen($cadena);    // 12
```

- **substr (\$cadena, \$inicio, \$lon)**

- Devuelve una subcadena de longitud \$lon a partir de la posición \$inicio de la cadena \$cadena.

```
$cad="PATITOS";  
echo substr($cad,2);      // TITOS  
echo substr($cad,2,3);    // TIT  
echo substr($cad,-2);     // OS  
echo substr($cad,2,-3);   // TI 2o no negativo = caracteres del  
                           final que se quitan
```

1.2.- Cadenas en PHP: Operación

- **substr_replace(\$cadena, \$reemplazo, \$ini, \$charsABorrar)**
 - Devuelve una cadena, resultado de reemplazar con el string dado en reemplazo, una copia de cadena empezando por el parámetro inicio hasta (opcionalmente) charsABorrar o hasta el final de cadena.
 - La cadena original no sufre ninguna modificación. Case sensitive

```
$texto="Hola a todos los chicos";  
- $texto2=substr_replace($texto,"todas", 0) // todas  
$texto2=substr_replace($texto,"todas", 7) // Hola a todas  
$texto2=substr_replace($texto,"todas", 7,5)  
// Hola a todas los chicos
```

1.2.- Cadenas en PHP: Operación

- **str_replace (\$cadBusq,\$cadReempl,\$texto)**
 - Devuelve una cadena resultado de sustituir \$cadBusq en \$texto por \$cadReempl.
 - La cadena original no sufre ninguna modificación.
 - **strtr (cadena, cadBus, cadRee)**
 - Sustituye carácter a carácter
 - La cadena original no sufre ninguna modificación.
- ```
echo strtr("Hola a todos los presentes aquí","aeiou", "AEIOU");
HOIA A tOdOs lOs prEsEntEs Aquí
```
- En lugar de dos argumentos se puede pasar un array.

## 1.2.- Cadena en PHP: Operación

- **substr\_count (\$cadena, \$patron, \$inicio, \$longitud)**
  - Devuelve el número de apariciones de \$patron dentro de la cadena \$cadena, (opcionalmente) empezando desde la posición \$inicio hasta \$longitud.

```
$cad="Hola a todos los presentes";
```

```
echo substr_count($cad,"a");
```

```
// 2
```



## 1.2.- Cadenas en PHP: Modificar

- **rtrim(\$cadena, \$listaChar)**
  - Elimina los espacios en blanco y caracteres de fin de línea {“ “, “\n”, “\t”, “\r”, “\0”, “\x0B” } del final de la cadena. Permite incluir una lista \$listaChar con los caracteres a eliminar del final. Devuelve la cadena modificada.
- **chop(\$cadena)**
  - Alias de rtrim().
- **ltrim(), trim()**
  - Eliminan caracteres a la izquierda o por ambos lados de una cadena.
- **str\_pad(\$cadena, \$long, \$cadenaRelleno)**
  - Rellena una cadena con un carácter de relleno (por defecto, es el espacio en blanco) hasta obtener \$long caracteres en total.
  - Si \$long es negativo o menor que la longitud de la cadena no rellena.
  - Parámetros: STR\_PAD\_RIGHT, STR\_PAD\_LEFT, STR\_PAD\_BOTH
    - Indican la opción para rellenar.

## 1.2.- Cadenas en PHP: Modificar

- **strtolower(\$cad) / strtoupper(\$cad)**
  - Devuelve una string con todos los caracteres alfabéticos convertidos a minúsculas/mayúsculas.
- **ucfirst(\$texto)**
  - Pone en mayúscula solo la 1a letra de \$texto
- **ucwords(\$texto)**
  - Pone en mayúscula la 1a letra de cada palabra contenida en \$texto.

## 1.2.- Cadenas en PHP: Modificar

- **addslashes(\$cadena)**
  - Antepone barras de escape \ a: ( " ), ( ' ), el NULL (el byte nulo) y ( \ ).
- **stripslashes(\$cadena)**
  - Quita barras de escape de: ( \ ' ) .
- **nl2br(\$cadena)**
  - Todos los saltos de línea serán convertidos a etiquetas <br />.
    - `echo nl2br($row['campo_BD']);`
- **addcslashes(\$cadena, \$listcar)**
  - Escapa cualquier carácter de \$listcar.
- **stripccslashes(\$cadena)**
  - Contraria a addcslashes () .

```
addcslashes("Hola","a,b"); // Hol\
```

## 1.2.- Cadenas en PHP: Modificar

- **quotemeta(\$cadena)**
  - Coloca una barra \ delante de : . , \ , + , \* , ? , [ , ^ , ] , ( , \$ , )
- **string htmlspecialchars (\$string [, \$flags [, \$codigo]])**
  - Traducen a entidades html
  - Las traducciones realizadas son:
    - &' se convierte en '&amp;'
    - ' “ ’ (comillas dobles) se convierte en '&quot;’ cuando ENT\_NOQUOTES no está establecido.
    - “ ‘ ” (comilla simple) se convierte en '&#039;’ (o &apos;) sólo cuando ENT\_QUOTES está establecido.
    - '<' (menor que) se convierte en '&lt;'
    - '>' (mayor que) se convierte en '&gt;'

ENT\_QUOTES → Convertirá tanto las comillas simples como las dobles

ENT\_NOQUOTES → Dejará tanto las comillas simples como las dobles sin convertir

## 1.2.- Cadenas en PHP: Modificar

- **string htmlentities (\$string [, \$flags [, \$codigo]])**
  - Esta función es idéntica a htmlspecialchars() en todos los aspectos, excepto que con htmlentities(), todos los caracteres que tienen equivalente HTML son convertidos a esas entidades.
- **htmlspecialchars\_decode()**
- **Html\_entity\_decode()**
- **strtok(\$cadena, \$delimitador)**
  - Divide una cadena en subcadenas separadas por \$delimitador. Devuelve la primera subcadena obtenida. Invocar desde un bucle para dividir toda una cadena.  
strtok("hola que tal", " ") → divide en palabras
- **chunk\_split(\$cadena [, \$long [, \$separador]])**
  - Inserta la cadena separador cada \$long caracteres  
chunk\_split("hola que tal", 3, '-'); → hol-a q-ue -tal

## 1.2.- Cadenas en PHP: Modificar

- `array str_split ( string $string [, int $split_length = 1 ] )`
  - Convierte un string en un array.
  - Si el parámetro opcional `split_length` se especifica, el array devuelto será separado en fragmentos los cuales cada uno tendrá una longitud de `split_length`, de otra manera cada fragmento tendrá una longitud de un carácter.
- `array explode ( string $delimiter ,string $string [, int $limit ] )`
  - Parte la cadena en función de `$delimiter` y crea un array con los elementos.  

```
$dir_correo="prueba@yahoo.com.ar";
$parte_array=explode("@",$dir_correo);
echo $parte_array[0]; // prueba
echo $parte_array[1]; // yahoo.com.ar
```

## 1.2.- Cadenas en PHP: Modificar

- **string implode ( string \$delimitador , array \$matriz )**
  - Convierte un array en una cadena de caracteres separando sus elementos con la cadena indicada en \$delimitador.  

```
$matriz=array(7,'julio',2011);
echo implode(' de ', $matriz);
$unidos=implode("@", $parte_array);
```
- **filter\_var (\$variable, \$filtro)**
  - Valida el contenido de la variable según el filtro indicado en la constante \$filtro:
  - Algunos tipos de filtro:
    - FILTER\_VALIDATE\_EMAIL: Valida emails según RFC 822
    - FILTER\_VALIDATE\_FLOAT: Valida números reales
    - FILTER\_VALIDATE\_IP: Valida IP's
    - FILTER\_VALIDATE\_URL : Valida URL's según RFC 2396

# 1.3.- Funciones relacionadas con los tipos de datos

En PHP existen funciones específicas para comprobar y establecer el tipo de datos de una variable, **gettype** obtiene el tipo de la variable que se le pasa como parámetro y devuelve una cadena de texto, que puede ser **array**, **boolean**, **double**, **integer**, **object**, **string**, **null**, **resource** o **unknowntype**.

También podemos comprobar si la variable es de un tipo concreto utilizando una de las siguientes funciones: **is\_array()**, **is\_bool()**, **is\_float()**, **is\_integer()**, **is\_null()**, **is\_numeric()**, **is\_object()**, **is\_resource()**, **is\_scalar()** e **is\_string()**. Devuelven true si la variable es del tipo indicado.

Análogamente, para establecer el tipo de una variable utilizamos la función **settype** pasándole como parámetros la variable a convertir, y una de las siguientes cadenas: **boolean**, **integer**, **float**, **string**, **array**, **object** o **null**. La función **settype** devuelve true si la conversión se realizó correctamente, o false en caso contrario.

```
<?php
$a = $b = "3.1416"; // asignamos a las dos variables la misma cadena de texto
settype($b, "float"); // y cambiamos $b a tipo float
print "\$a vale $a y es de tipo ".gettype($a);
print "
";
print "\$b vale $b y es de tipo ".gettype($b);
?>
```

El resultado del código anterior es:

```
$a vale 3.1416 y es de tipo string
$b vale 3.1416 y es de tipo double
```



# 1.3.- Funciones relacionadas con los tipos de datos

Si lo único que te interesa es saber si una variable está definida y no es **null**, puedes usar la función **isset**. La función **unset** destruye la variable o variables que se le pasa como parámetro.

```
<?php
$a = "3.1416";
if (isset($a)) // la variable $a está definida
unset($a); //ahora ya no está definida
?>
```

Es importante no confundir el que una variable esté definida o valga **null**, con que se considere como vacía debido al valor que contenga. Esto último es lo que nos indica la función **empty**.

Existe también en PHP una función, **define**, con la que puedes definir constantes, esto es, identificadores a los que se les asigna un valor que no cambia durante la ejecución del programa.

```
bool define (string $identificador , mixed $valor [, bool $case_insensitive = false]);
```

Los identificadores no van precedidos por el signo "\$" y suelen escribirse en mayúsculas, aunque existe un tercer parámetro opcional, que si vale true hace que se reconozca el identificador independientemente de si está escrito en mayúsculas o en minúsculas.

```
<?php
define ("PI", 3.1416, true);
print "El valor de PI es ".pi; //El identificador se reconoce tanto por PI como por pi
?>
```

Sólo se permiten los siguientes tipos de valores para las constantes: **integer**, **float**, **string**, **boolean** y **null**.

## 1.3.1.- Funciones relacionadas con los tipos de datos

En PHP no existe un tipo de datos específico para trabajar con fechas y horas. La información de fecha y hora se almacena internamente como un número entero. Sin embargo, dentro de las funciones de PHP tienes a tu disposición unas cuantas para trabajar con ese tipo de datos.

Una de las más útiles es quizás la función **date**, que te permite obtener una cadena de texto a partir de una fecha y hora, con el formato que tú elijas. La función recibe dos parámetros, la descripción del formato y el número entero que identifica la fecha, y devuelve una cadena de texto formateada.

```
string date (string $formato [, int $fechahora]);
```

El formato lo debes componer utilizando como base una serie de caracteres de los que figuran en la siguiente tabla.

## 1.3.1.- Funciones relacionadas con los tipos de datos

### Función date: caracteres de formato para fechas y horas.

Además, el segundo parámetro es opcional. Si no se indica, se utilizará la hora actual para crear la cadena de texto.

Para que el sistema pueda darte información sobre tu fecha y hora, debes indicarle tu zona horaria. Puedes hacerlo con la función **date\_default\_timezone\_set**. Para establecer la zona horaria en España peninsular debes indicar:

```
date_default_timezone_set('Europe/Madrid');
```

Si utilizas alguna función de fecha y hora sin haber establecido previamente tu zona horaria, lo más probable es que recibas un error o mensaje de advertencia de PHP indicándolo.

Otras funciones como **getdate** devuelven un array con información sobre la fecha y hora actual.

| Carácter | Resultado                                                                |
|----------|--------------------------------------------------------------------------|
| d        | Día del mes con dos dígitos                                              |
| j        | Día del mes con uno o dos dígitos (sin ceros iniciales)                  |
| z        | Día del año, comenzando por el cero (0 es el 1 de enero)                 |
| N        | Día de la semana (1 = lunes, ..., 7 = domingo)                           |
| w        | Día de la semana (0 = domingo, ..., 6 = sábado)                          |
| l        | Texto del día de la semana, en inglés (Monday, ..., Sunday)              |
| D        | Texto del día de la semana, solo tres letras, en inglés (Mon, ..., Sun)  |
| w        | Número de semana del año                                                 |
| m        | Número del mes con dos dígitos                                           |
| n        | Número del mes con uno o dos dígitos (sin ceros iniciales)               |
| t        | Número de días que tiene el mes                                          |
| F        | Texto del día del mes, en inglés (January, ..., December)                |
| M        | Texto del día del mes, solo tres letras, en inglés (Jan, ..., Dec)       |
| Y        | Número del año                                                           |
| y        | Dos últimos dígitos del número del año                                   |
| L        | 1 si el año es bisiesto, 0 si no lo es                                   |
| h        | Hora en formato de 12 horas, siempre con dos dígitos                     |
| H        | Hora en forma de 24 horas, siempre con dos dígitos                       |
| g        | Hora en formato de 12 horas, con uno o dos dígitos (sin ceros iniciales) |
| G        | Hora en formato 24 horas, con uno o dos dígitos (sin ceros iniciales)    |
| i        | Minutos, siempre con dos dígitos                                         |
| s        | Segundos, siempre con dos dígitos                                        |
| u        | microsegundos                                                            |
| a        | am o pm, en minúsculas                                                   |
| A        | AM o PM, en mayúsculas                                                   |
| r        | Fecha entera con formato RFC 2822                                        |

En el curso de Moodle encontrarás un enlace con información sobre las distintas zonas horarias y otro con las funciones para gestionar fechas y horas

## 1.4.- Variantes especiales de PHP

En la unidad anterior ya aprendiste qué eran y cómo se utilizaban las variables globales. PHP incluye unas cuantas variables internas predefinidas que pueden usarse desde cualquier ámbito, por lo que reciben el nombre de **variables superglobales**. Ni siquiera es necesario que uses **global** para acceder a ellas.

Cada una de estas variables es un array que contiene un conjunto de valores (en esta unidad veremos más adelante cómo se pueden utilizar los arrays). Las variables superglobales disponibles en PHP son las siguientes:

**\$\_SERVER**. Contiene información sobre el entorno del servidor web y de ejecución. Entre la información que nos ofrece esta variable, tenemos:

### Principales valores de la variable **\$\_SERVER**

| Valor                                    | Contenido                                                                 |
|------------------------------------------|---------------------------------------------------------------------------|
| <code>\$_SERVER['PHP_SELF']</code>       | guión que se está ejecutando actualmente.                                 |
| <code>\$_SERVER['SERVER_ADDR']</code>    | dirección IP del servidor web.                                            |
| <code>\$_SERVER['SERVER_NAME']</code>    | nombre del servidor web.                                                  |
| <code>\$_SERVER['DOCUMENT_ROOT']</code>  | directorio raíz bajo el que se ejecuta el guión actual.                   |
| <code>\$_SERVER['REMOTE_ADDR']</code>    | dirección IP desde la que el usuario está viendo la página.               |
| <code>\$_SERVER['REQUEST_METHOD']</code> | método utilizado para acceder a la página ('GET', 'HEAD', 'POST' o 'PUT') |

En el curso de Moodle encontrarás un enlace donde consultar toda la información que ofrece **\$\_SERVER**

## 1.4.- Variantes especiales de PHP

**\$\_GET**, **\$\_POST** y **\$\_COOKIE** contienen las variables que se han pasado al guión actual utilizando respectivamente los métodos GET (parámetros en la URL), HTTP POST y Cookies HTTP.

**\$\_REQUEST** junta en uno solo el contenido de los tres arrays anteriores, **\$\_GET**, **\$\_POST** y **\$\_COOKIE**.

**\$\_ENV** contiene las variables que se puedan haber pasado a PHP desde el entorno en que se ejecuta.

**\$\_FILES** contiene los ficheros que se puedan haber subido al servidor utilizando el método POST.

**\$\_SESSION** contiene las variables de sesión disponibles para el guión actual.

En posteriores unidades iremos trabajando con estas variables.

En el curso de Moodle encontrarás un enlace donde consultar información sobre estas variables superglobales

## 2.- Estructuras de control

**En PHP los guiones se construyen en base a sentencias.** Utilizando llaves, puedes agrupar las sentencias en conjuntos, que se comportan como si fueran una única sentencia.

Para definir el flujo de un programa en PHP, al igual que en la mayoría de lenguajes de programación, hay sentencias para dos tipos de **estructuras de control: sentencias condicionales**, que permiten definir las condiciones bajo las que debe ejecutarse una sentencia o un bloque de sentencias; y **sentencias de bucle**, con las que puedes definir si una sentencia o conjunto de sentencias se repite o no, y bajo qué condiciones.

Además, en PHP puedes usar también (aunque no es recomendable) la sentencia **goto**, que te permite saltar directamente a otro punto del programa que indiques mediante una etiqueta.

```
<?php
 $a = 1;
 goto salto;
 $a++; //esta sentencia no se ejecuta
salto:
 echo $a; // el valor obtenido es 1
?>
```

## 2.1.- Condicionales

**if / elseif / else.** La sentencia **if** permite definir una expresión para ejecutar o no la sentencia o conjunto de sentencias siguiente. Si la expresión se evalúa a true (verdadero), la sentencia se ejecuta. Si se evalúa a false (falso), no se ejecutará.

Cuando el resultado de la expresión sea false, puedes utilizar **else** para indicar una sentencia o grupo de sentencias a ejecutar en ese caso. Otra alternativa a **else** es utilizar **elseif** y escribir una nueva expresión que comenzará un nuevo condicional.

```
<?php
 if ($a < $b)
 print "a es menor que b";
 elseif ($a > $b)
 print "a es mayor que b";
 else
 print "a es igual a b";
?>
```

Cuando, como sucede en el ejemplo, la sentencia **if elseif** o **else** actúe sobre una única sentencia, no será necesario usar llaves. Tendrás que usar llaves para formar un conjunto de sentencias siempre que quieras que el condicional actúe sobre más de una sentencia.

## 2.1.- Condicionales

**switch.** La sentencia **switch** es similar a enlazar varias sentencias **if** comparando una misma variable con diferentes valores. Cada valor va en una sentencia **case**. Cuando se encuentra una coincidencia, comienzan a ejecutarse las sentencias siguientes hasta que acaba el bloque **switch**, o hasta que se encuentra una sentencia **break**. Si no existe coincidencia con el valor de ningún **case**, se ejecutan las sentencias del bloque **default**, en caso de que exista.

```
<?php
switch ($a) {
 case 0:
 print "a vale 0";
 break;
 case 1:
 print "a vale 1";
 break;
 default:
 print "a no vale 0 ni 1";
}
?>
```

Haz una página web que muestre la fecha actual en castellano, incluyendo el día de la semana, con un formato similar al siguiente: "Miércoles, 13 de Abril de 2011".  
(Solución en el Anexo II y en el curso de Moodle)



## 2.2.- Bucles

**while:** Usando **while** puedes definir un bucle que se ejecuta mientras se cumpla una expresión. La expresión se evalúa antes de comenzar cada ejecución del bucle.

```
<?php
 $a = 1;
 while ($a < 8)
 $a += 3;
 print $a; // el valor obtenido es 10
?>
```

**do / while:** Es un bucle similar al anterior, pero la expresión se evalúa al final, con lo cual se asegura que la sentencia o conjunto de sentencias del bucle se ejecutan al menos una vez.

```
<?php
 $a = 5;
 do
 $a -= 3;
 while ($a > 10);
 print $a; // el bucle se ejecuta una sola vez, con lo que el valor obtenido es 2
?>
```

## 2.2.- Bucles

**for:** Son los bucles más complejos de PHP. Al igual que los del lenguaje C, se componen de tres expresiones:

```
for (expr1; expr2; expr3)
 sentencia o conjunto de sentencias;
```

La primera expresión, **expr1**, se ejecuta solo una vez al comienzo del bucle.

La segunda expresión, **expr2**, se evalúa para saber si se debe ejecutar o no la sentencia o conjunto de sentencias. Si el resultado es false, el bucle termina.

Si el resultado es true, se ejecutan las sentencias y al finalizar se ejecuta la tercera expresión, **expr3**, y se vuelve a evaluar **expr2** para decidir si se vuelve a ejecutar o no el bucle.

```
<?php
 for ($a = 5; $a<10; $a+=3) {
 print $a; // Se muestran los valores 5 y 8
 print "
";
 }
?>
```

Puedes anidar cualquiera de los bucles anteriores en varios niveles. También puedes usar las sentencias **break**, para salir del bucle, y **continue**, para omitir la ejecución de las sentencias restantes y volver a la comprobación de la expresión respectivamente.

## 3.- Funciones

Cuando quieres repetir la ejecución de un bloque de código, puedes utilizar un bucle. Las **funciones** tienen una utilidad similar: **nos permiten asociar una etiqueta** (el nombre de la función) **con un bloque de código** a ejecutar. Además, al usar funciones estamos ayudando a estructurar mejor el código. Como ya sabes, las funciones permiten crear variables locales que no serán visibles fuera del cuerpo de las mismas.

Como programador puedes aprovecharte de la gran cantidad de funciones disponibles en PHP. De éstas, muchas están incluidas en el núcleo de PHP y se pueden usar directamente. Otras muchas se encuentran disponibles en forma de extensiones, y se pueden incorporar al lenguaje cuando se necesitan.

Con la distribución de PHP se incluyen varias extensiones. Para poder usar las funciones de una extensión, tienes que asegurarte de activarla mediante el uso de una directiva **extensión** en el fichero **php.ini**. Muchas otras extensiones no se incluyen con PHP y antes de poder utilizarlas tienes que descargarlas.

Para obtener extensiones para el lenguaje PHP puedes utilizar PECL. PECL es un repositorio de extensiones para PHP. Junto con PHP se incluye un **comando pecl** que puedes utilizar para instalar extensiones de forma sencilla:

```
pecl install nombre_extensión
```

En el curso puedes encontrar un enlace a información sobre PECL

## 3.1.- Inclusión de ficheros externos

Conforme vayan creciendo los programas que hagas, verás que resulta trabajoso encontrar la información que buscas dentro del código. En ocasiones resulta útil agrupar ciertos grupos de funciones o bloques de código, y ponerlos en un fichero aparte. Posteriormente, puedes hacer referencia a esos ficheros para que PHP incluya su contenido como parte del programa actual.

Para incorporar a tu programa contenido de un archivo externo, tienes varias posibilidades:

- ✓ **include:** Evalúa el contenido del fichero que se indica y lo incluye como parte del fichero actual, en el mismo punto en que se realiza la llamada. La ubicación del fichero puede especificarse utilizando una ruta absoluta, pero lo más usual es con una ruta relativa. En este caso, se toma como base la ruta que se especifica en la directiva **include\_path** del fichero **php.ini**. Si no se encuentra en esa ubicación, se buscará también en el directorio del guión actual, y en el directorio de ejecución.

Ejercicio resuelto:

definiciones.php

```
<?php
 $modulo = 'DWES';
 $ciclo = 'DAW';
?>
```

programa.php

```
<?php
 print "Módulo $modulo del ciclo $ciclo
"; //Solo muestra "Modulo del ciclo"
 include 'definiciones.php';
 print " Módulo $modulo del ciclo $ciclo
"; // muestra "Modulo DWES del ciclo DAW"
?>
```

Cuando se comienza a evaluar el contenido del fichero externo, se abandona de forma automática el modo PHP y su contenido se trata en principio como etiquetas HTML. Por este motivo, es necesario delimitar el código PHP que contenga nuestro archivo externo utilizando dentro del mismo los delimitadores `<?php` y `?>`.

En el curso de Moodle tienes un ejemplo de utilización de include

## 3.1.- Inclusión de ficheros externos

- ✓ **include\_once**: Si por equivocación incluyes más de una vez un mismo fichero, lo normal es que obtengas algún tipo de error (por ejemplo, al repetir una definición de una función). **include\_once** funciona exactamente igual que **include**, pero solo incluye aquellos ficheros que aún no se hayan incluido.
- ✓ **require**: Si el fichero que queremos incluir no se encuentra, **include** da un aviso y continua la ejecución del guión. La diferencia más importante al usar **require** es que en ese caso, cuando no se puede incluir el fichero, se detiene la ejecución del guión.
- ✓ **require\_once**. Es la combinación de las dos anteriores. Asegura la inclusión del fichero indicado solo una vez, y genera un error si no se puede llevar a cabo.

Muchos programadores utilizan la doble extensión `.inc.php` para aquellos ficheros en lenguaje PHP cuyo destino es ser incluidos dentro de otros, y nunca han de ejecutarse por sí mismos.

## 3.2.- Ejecución y creación de funciones

Ya sabes que para hacer una llamada a una función, basta con poner su nombre y unos paréntesis.

```
<?php
 phpinfo();
?>
```

Para crear tus propias funciones, deberás usar la palabra function.

```
<?php
 function precio_con_iva() {
 global $precio;
 $precio_iva = $precio * 1.18;
 print "El precio con IVA es ".$precio_iva;
 }
 $precio = 10;
 precio_con_iva();
?>
```

## 3.2.- Ejecución y creación de funciones

En PHP no es necesario que definas una función antes de utilizarla, excepto cuando está condicionalmente definida como se muestra en el siguiente ejemplo:

```
<?php
 $iva = true;
 $precio = 10;
 precio_con_iva(); // Da error, pues aquí aún no está definida la función
 if ($iva) {
 function precio_con_iva() {
 global $precio;
 $precio_iva = $precio * 1.18;
 print "El precio con IVA es ".$precio_iva;
 }
 }
 precio_con_iva(); // Aquí ya no da error
?>
```

Cuando una función está definida de una forma condicional sus definiciones deben ser procesadas antes de ser llamadas. Por tanto, la definición de la función debe estar antes de cualquier llamada.

## 3.3.- Argumentos

En el ejemplo anterior en la función usabas una variable global, lo cual no es una buena práctica. Siempre es mejor utilizar argumentos o parámetros al hacer la llamada. Además, en lugar de mostrar el resultado en pantalla o guardar el resultado en una variable global, las funciones pueden devolver un valor usando la sentencia **return**. Cuando en una función se encuentra una sentencia **return**, termina su procesamiento y devuelve el valor que se indica.

Puedes reescribir la función anterior de la siguiente forma:

```
<?php
 function precio_con_iva($precio) {
 return $precio * 1.18;
 }
 $precio = 10;
 $precio_iva = precio_con_iva($precio);
 print "El precio con IVA es ".$precio_iva
?>
```

Los argumentos se indican en la definición de la función como una lista de variables separada por comas. No se indica el tipo de cada argumento, al igual que no se indica si la función va a devolver o no un valor (si una función no tiene una sentencia **return**, devuelve **null** al finalizar su procesamiento).



## 3.3.- Argumentos

Al definir la función, puedes indicar valores por defecto para los argumentos, de forma que cuando hagas una llamada a la función puedes no indicar el valor de un argumento; en este caso se toma el valor por defecto indicado.

```
<?php
 function precio_con_iva($precio, $iva=0.18) {
 return $precio * (1 + $iva);
 }
 $precio = 10;
 $precio_iva = precio_con_iva($precio);
 print "El precio con IVA es ".$precio_iva
?>
```

Puede haber valores por defecto definidos para varios argumentos, pero en la lista de argumentos de la función todos ellos deben estar a la derecha de cualquier otro argumento sin valor por defecto.

## 3.3.- Argumentos

En los ejemplos anteriores los argumentos se pasaban por valor. Esto es, cualquier cambio que se haga dentro de la función a los valores de los argumento no se reflejará fuera de la función. Si quieres que esto ocurra debes definir el parámetro para que su valor se pase por referencia, añadiendo el símbolo & antes de su nombre.

```
<?php
 function precio_con_iva(&$precio, $iva=0.18) {
 $precio *= (1 + $iva);
 }
 $precio = 10;
 precio_con_iva($precio);
 print "El precio con IVA es ".$precio
?>
```

Anteriormente hiciste un ejercicio que mostraba la fecha actual en castellano. Con el mismo objetivo (puedes utilizar el código ya hecho), crea una función que devuelva una cadena de texto con la fecha en castellano, e introdúcela en un fichero externo. Después crea una página en PHP que incluya ese fichero y utilice la función para mostrar en pantalla la fecha obtenida. (Solución en Anexo III y en el curso de Moodle)

## 3.3.1.- Número variable de parámetros

- PHP permite una lista de valores de longitud variable como parámetro
- Funciones a usar:
  - **func\_num\_args()** → número de argumentos pasados a la función.
  - **func\_get\_args()** → array con los argumentos pasados a la función
  - **func\_get\_arg(num)** → el argumento que está en la posición num en la lista de argumentos. La primera posición es la 0

```
function prueba(){
 $num_args = func_num_args();
 echo "Numero de argumentos:$num_args
\n";
 if ($num_args >= 2) {
 echo "El 2º argumento es:".func_get_arg(1)."
\n";
 }
 $parametros=func_get_args();
 echo "Array con todos los argumentos:
\n";
 print_r($parametros);
}
prueba (1, 2, 3);
```

```
Numero de argumentos: 3
El 2º argumento es: 2
Array con todos los argumentos:
```

```
Array
(
 [0] => 1
 [1] => 2
 [2] => 3
)
```

### 3.3.1.- Número variable de parámetros

- PHP 5.6: las listas de argumentos de las funciones pueden incluir el token ... para indicar que aceptan un número variable de parámetros.
- Los argumentos serán pasados a la variable dada como un array

```
<?php
function sum(...$números) {
 $acu = 0;
 foreach ($números as $n) {
 $acu += $n;
 }
 return $acu;
}
echo sum(1, 2, 3, 4); // El resultado sería 10
?>
```

## 3.3.2.- Funciones variables

- Definimos una variable: **\$func="prueba";**
- Si llamamos a esa variable con paréntesis: \$func() → PHP buscará una función con el mismo nombre que su contenido y la ejecutará si existe.
- Las funciones variables no funcionarán con echo(), print(), unset(), isset(), empty(), include(), require() y derivados.

```
function prueba($arg = ' '){
 echo "Estamos en la función prueba(); y el argumento es '$arg'.
\n";
}
$func = "prueba";
$func('hola'); // Esto llama prueba('hola')
```

## 3.3.3.- Funciones recursivas

- Una función se llama recursiva cuando en algún punto de su cuerpo se llama a sí misma.
- ¡Cuidado! puede llamarse a sí misma indefinidamente.
- Es muy importante la condición de salida.

```
<?php
function recursividad($a){
 if ($a < 20) {
 echo "$a\n";
 recursividad($a + 1);
 }
}
?>
```

## 3.3.4.- Funciones anónimas

- Las funciones anónimas, también se conocen como **clausuras** (closures).
- Permiten la creación de funciones que no tienen un nombre especificado.
- Las clausuras también se pueden usar como valores de variables.

```
<?php
 $saludo = function($nombre){
 printf("Hola %s\r\n", $nombre);};
 $saludo('Mundo');
 $saludo('PHP');
?>
```

## 3.3.5.- Funciones en PHP

- **bool function\_exists ( string \$function\_name )**
  - Comprueba la lista de funciones definidas, las incluidas (internas) y las definidas por el usuario, para **function\_name**.
  - Recibe el nombre de la función buscada como cadena.
  - Devuelve TRUE si function\_name existe y es una función, si no, FALSE.

```
<?php
 if (function_exists('imap_open')) {
 echo "Las funciones de IMAP están disponibles.
\n";
 } else {
 echo "Las funciones de IMAP no están disponibles.
\n";
 }
?>
```



## 4.- Tipos compuestos

Un tipo de datos compuesto es aquel que te permite almacenar más de un valor. En PHP puedes utilizar dos tipos de datos compuestos: el **array** y el **objeto**. Los objetos los veremos más adelante; vamos a empezar con los arrays.

Un **array** es un tipo de datos que nos permite almacenar varios valores. Cada miembro del **array** se almacena en una posición a la que se hace referencia utilizando un valor clave. Las claves pueden ser numéricas o asociativas.

*// array numérico*

```
$modulos1 = array(0 => "Programación", 1 => "Bases de datos", ..., 9 => "Desarrollo web en entorno servidor");
```

*// array asociativo*

```
$modulos2 = array("PR" => "Programación", "BD" => "Bases de datos", ..., "DWES" => "Desarrollo web en entorno servidor");
```

En PHP existe la función **print\_r**, que nos muestra todo el contenido del array que le pasamos. Es muy útil para tareas de depuración. Puedes encontrar un enlace a la documentación de esta función en el curso de Moodle.

Para hacer referencia a los elementos almacenados en un array, tienes que utilizar el valor clave entre corchetes:

```
$modulos1 [9]
```

```
$modulos2 ["DWES"]
```

## 4.- Tipos compuestos

Los arrays anteriores son vectores, esto es, arrays unidimensionales. En PHP puedes crear también arrays de varias dimensiones almacenando otro array en cada uno de los elementos de un array.

```
// array bidimensional
$ciclos = array(
 "DAW" => array ("PR" => "Programación", "BD" => "Bases de datos", ..., "DWES" => "Desarrollo web en entorno servidor"),
 "DAM" => array ("PR" => "Programación", "BD" => "Bases de datos", ..., "PMDM" => "Programación multimedia y de dispositivos móviles")
);
```

Para hacer referencia a los elementos almacenados en un array multidimensional, debes indicar las claves para cada una de las dimensiones:

```
$ciclos ["DAW"] ["DWES"]
```

En PHP no es necesario que indiques el tamaño del array antes de crearlo. Ni siquiera es necesario indicar que una variable concreta es de tipo array. Simplemente puedes comenzar a asignarle valores:

```
// array numérico
$modulos1 [0] = "Programación";
$modulos1 [1] = "Bases de datos";
...
$modulos1 [9] = "Desarrollo web en entorno servidor";
// array asociativo
$modulos2 ["PR"] = "Programación";
$modulos2 ["BD"] = "Bases de datos";
...
$modulos2 ["DWES"] = "Desarrollo web en entorno servidor";
```

Ni siquiera es necesario que especifiques el valor de la clave. Si la omites, el array se irá llenando a partir de la última clave numérica existente, o de la posición 0 si no existe ninguna:

```
$modulos1 [] = "Programación";
$modulos1 [] = "Bases de datos";
...
$modulos1 [] = "Desarrollo web en entorno servidor";
```

## 4.1.- Recorrer arrays

- **Recorrer un array secuencial:**

- Usar `count($matriz)` y un bucle

- **Int `count( mixed $array)`**

- Devuelve el número de elementos que contiene el array.

```
$matriz = array('lunes', 'martes', 'miércoles', 'jueves', 'viernes', 'sábado', 'domingo');
echo count($matriz);
```

- Otra función para el tamaño de la matriz

- `sizeof($matriz)`

- Devuelve el número de elementos

## 4.1.- Recorrer arrays

Las **cadenas de texto** o **strings** se pueden tratar como arrays en los que se almacena una letra en cada posición, siendo 0 el índice correspondiente a la primera letra, 1 el de la segunda, etc.

```
// cadena de texto
$modulo = "Desarrollo web en entorno servidor";
// $modulo[3] == "a";
```

Para recorrer los elementos de un array, en PHP puedes usar un bucle específico: **foreach**. Utiliza una variable temporal para asignarle en cada iteración el valor de cada uno de los elementos del array. Puedes usarlo de dos formas. Recorriendo sólo los elementos:

```
$modulos = array("PR" => "Programación", "BD" => "Bases de datos", ..., "DWES" => "Desarrollo web en entorno servidor");
foreach ($modulos as $modulo) {
 print "Módulo: ".$modulo."
"
}
```

O recorriendo los elementos y sus valores clave de forma simultánea:

```
$modulos = array("PR" => "Programación", "BD" => "Bases de datos", ..., "DWES" => "Desarrollo web en entorno servidor");
foreach ($modulos as $codigo => $modulo) {
 print "El código del módulo ".$modulo." es ".$codigo."
"
}
```

Haz una página PHP que utilice foreach para mostrar todos los valores del array \$\_SERVER en una tabla con dos columnas. La primera columna debe contener el nombre de la variable, y la segunda su valor. (Solución en Anexo IV y en el curso de Moodle)

## 4.1.1.- Recorrer arrays

Pero en PHP también **hay otra forma de recorrer los valores de un array**. Cada array mantiene un **puntero** interno, que se puede utilizar con este fin. Utilizando funciones específicas, podemos avanzar, retroceder o inicializar el puntero, así como recuperar los valores del elemento (o de la pareja clave / elemento) al que apunta el puntero en cada momento. Algunas de estas funciones son:

### Funciones para recorrer arrays.

| Función | Resultado                                                                                                           |
|---------|---------------------------------------------------------------------------------------------------------------------|
| reset   | Sitúa el puntero interno al comienzo del array                                                                      |
| next    | Avanza el puntero interno una posición                                                                              |
| prev    | Mueve el puntero interno una posición hacia atrás                                                                   |
| end     | Sitúa el puntero interno al final del array                                                                         |
| current | Devuelve el elemento de la posición actual                                                                          |
| key     | Devuelve la clave de la posición actual                                                                             |
| each    | Devuelve un array con la clave y el elemento de la posición actual. Además, avanza el puntero interno una posición. |

## 4.1.1.- Recorrer arrays

Las funciones **reset**, **next**, **prev** y **end**, además de mover el puntero interno devuelven, al igual que **current**, el valor del nuevo elemento en que se posiciona. Si al mover el puntero te sales de los límites del array (por ejemplo, si ya estás en el último elemento y haces un **next**), cualquiera de ellas devuelve **false**. Sin embargo, al comprobar este valor devuelto no serás capaz de distinguir si te has salido de los límites del array, o si estás en una posición válida del array que contiene el valor "false".

La función **key** devuelve **null** si el puntero interno está fuera de los límites del array.

La función **each** devuelve un array con cuatro elementos. Los elementos **0** y **'key'** almacenan el valor de la clave en la posición actual del puntero interno. Los elementos **1** y **'value'** devuelven el valor almacenado.

Si el puntero interno del array se ha pasado de los límites del array, la función **each** devuelve **false**, por lo que la puedes usar para crear un bucle que recorra el array de la siguiente forma:

```
while ($modulo = each($modulos)) {
 print "El código del módulo ".$modulo[1]." es ".$modulo[0]."
"
}
```

Haz una página PHP que utilice estas funciones para crear una tabla como la del ejercicio anterior. (Solución en el Anexo V y en el curso de Moodle)

## 4.2.- Funciones relacionadas con los tipos de datos compuestos

Además de asignando valores directamente, **la función array permite crear un array con una sola línea de código**, tal y como vimos anteriormente. Esta función recibe un conjunto de parámetros, y crea un array a partir de los valores que se le pasan. Si en los parámetros no se indica el valor de la clave, crea un array numérico (con base 0). Si no se le pasa ningún parámetro, crea un array vacío.

```
$a = array(); // array vacío
```

```
$modulos = array("Programación", "Bases de datos", ..., "Desarrollo web en entorno servidor"); // array numérico
```

Una vez definido un array puedes añadir nuevos elementos (no definiendo el índice, o utilizando un índice nuevo) y modificar los ya existentes (utilizando el índice del elemento a modificar). También se pueden eliminar elementos de un array utilizando la función **unset**.

En el caso de los arrays numéricos, eliminar un elemento significa que las claves del mismo ya no estarán consecutivas.

```
unset ($modulos [0]);
```

```
// El primer elemento pasa a ser $modulos [1] == "Bases de datos";
```

## 4.2.- Funciones relacionadas con los tipos de datos compuestos

La función **array\_values** recibe un array como parámetro, y devuelve un nuevo array con los mismos elementos y con índices numéricos consecutivos con base 0.

Para comprobar si una variable es de tipo array, utiliza la función **is\_array**. Para obtener el número de elementos que contiene un array, tienes la función **count**.

Si quieres buscar un elemento concreto dentro de un array, puedes utilizar la función **in\_array**. Recibe como parámetros el elemento a buscar y la variable de tipo array en la que buscar, y devuelve true si encontró el elemento o false en caso contrario.

```
$modulos = array("Programación", "Bases de datos", "Desarrollo web en entorno servidor");
$modulo = "Bases de datos";
if (in_array($modulo, $modulos)) printf "Existe el módulo de nombre ".$modulo;
```

Otra posibilidad es la función **array\_search**, que recibe los mismos parámetros pero devuelve la clave correspondiente al elemento, o false si no lo encuentra.

Y si lo que quieres buscar es una clave en un array, tienes la función **array\_key\_exists**, que devuelve true o false.

En el curso de Moodle podrás encontrar un enlace a una lista completa de funciones para gestionar arrays.



## 4.2.1.- Modificar un array

- `mixed array_pop ( array &$matriz )`
  - Extrae y devuelve el último elemento del array. Obsérvese que esta función actúa sobre el array original como indica el hecho de que reciba el argumento implícitamente por referencia.

```
$matriz=array('lunes','martes','miércoles','jueves','viernes','sábado','domingo');
```

```
echo array_pop($matriz);
```

```
var_dump($matriz);
```

## 4.2.1.- Modificar un array

- `int array_push( array &$matriz, $var1, $var2, ...)`
  - Inserta los elementos `$var` al final del array y devuelve el número de elementos que contiene el array aumentado.

```
$matriz=array('lunes','martes','miércoles','jueves','viernes','sábado');
```

```
echo (array_push($matriz,'domingo'));
```

```
var_dump($matriz);
```

## 4.2.1.- Modificar un array

- `mixed array_shift ( array &$matriz )`
  - Extrae el primer elemento de la matriz, desplazando todos los elementos restantes hacia adelante. Devuelve el elemento extraído.

```
$matriz=array('lunes','martes','miércoles','jueves','viernes','sábado','domingo');
```

```
echo array_shift($matriz);
```

```
var_dump($matriz);
```

- `array_unshift ( $mat, $elem1, $elem2, ...)`
  - Permite añadir uno o más elementos por el inicio de la matriz indicada como parámetro. Devuelve el nuevo número de elementos del array.

## 4.2.1.- Modificar un array

- `array_walk ( matriz, func_usuario [, parametro])`
  - Nos permite aplicar una función definida por el usuario a cada uno de los elementos de un array.
  - La función `func_usuario()` recibe, al menos, dos parámetros
    - El valor del elemento
    - Su clave asociada
  - Una vez aplicada la función, el puntero interno del array se encontrará al final de él.

```
function aEuros(&$valor,$clave){
 $valor=$valor/166.386;
}
array_walk($precios,'aEuros');
```

| Producto | Precio     | Producto | Precio |
|----------|------------|----------|--------|
| prod1    | 1500 Ptas. | prod1    | 9.02 € |
| prod2    | 1000 Ptas. | prod2    | 6.01 € |
| prod3    | 800 Ptas.  | prod3    | 4.81 € |
| prod6    | 100 Ptas.  | prod6    | 0.60 € |
| prod7    | 500 Ptas.  | prod7    | 3.01 € |

## 4.2.1.- Modificar un array

- `array array_replace ( array &$matriz_destino , array &$matriz_origen)`
  - Devuelve un array que es el resultado de sobrescribir/ añadir sobre matriz destino los elementos de matriz origen (los que coinciden en índice se sobrescriben, y los que no se añaden). No afecta a las matrices que recibe como argumento.

```
$matriz_destino=array('altura'=>185,'peso'=>85);
$matriz_origen=array('pelo'=>'moreno','peso'=>95);
var_dump(array_replace($matriz_destino, $matriz_origen));
```

### 4.2.1.- Modificar un array

- `array_merge($mat1, $mat2, $mat3)`
  - Une las matrices indicadas como parámetros, empezando por la primera. Elimina los elementos con claves duplicadas (dejando la última leída).
- También podemos unir matrices con el **operador +** . Elimina claves duplicadas (dejando el primer elemento leído).

## 4.2.1.- Modificar un array

- `array_merge_recursive($mat1,$mat2,$mat3)`
  - Permite combinar matrices sin perder elementos. Devuelve la matriz resultado de la suma. Con las claves duplicadas genera una nueva matriz para ese elemento.
- `array_pad($mat, $cantidad, $relleno)`
  - Permite añadir elementos de relleno en el inicio (negativo) y fin del array (positivo). Devuelve la matriz resultado.

## 4.2.1.- Modificar un array

- `array array_slice ( array $matriz , int $inicio, int $cantidad)`
  - Devuelve un sub-array de \$matriz a partir del inicio indicado y con la cantidad de elementos indicada.
  - Si cantidad no se especifica devuelve todos los elementos desde inicio hasta el final.

```
$vec=array(10,6,7,8,23);
```

```
$res=array_slice($vec,1,3); // $res= 6,7,8
```

| Inicio   |                                                          |
|----------|----------------------------------------------------------|
| Positivo | Posición del primer elemento contando desde el principio |
| Negativo | Posición de comienzo contando desde el final             |
| Cantidad |                                                          |
| Positivo | Número de elementos a considerar                         |
| Negativo | Se detendrá a tantos elementos del final                 |
| Nulo     | Se consideran todos los elementos hasta el final         |



## 4.2.1.- Modificar un array

- array array\_splice ( array \$matriz , int \$inicio, int \$cantidad, mixed \$reemplazo)
  - Elimina de matriz cantidad elementos contados a partir del elemento inicio, los sustituye por los elementos del array reemplazo y los devuelve en un array. Si los índices son numéricos los reajusta.

```
$matriz=array('lunes','martes','miércoles','jueves','viernes','sábado',
'domingo');
```

```
var_dump (array_splice($matriz,1,2));
```

```
var_dump($matriz);
```

```
$matriz=array('altura'=>185,'peso'=>85,'pelo'=>'moreno');
```

```
var_dump(array_splice($matriz,1,2));
```

```
var_dump($matriz);
```

### 4.2.1.- Modificar un array

- `string implode ( string $delimitador, array $matriz )`
  - Convierte matriz en una cadena de caracteres separando sus elementos con la cadena indicada en delimitador.

```
$matriz=array(7,'julio',2011);
echo implode(' de ', $matriz);
```

## 4.2.1.- Modificar un array

- Intersección de matrices.
  - `array_intersect($mat1,$mat2,$mat3)`
    - Devuelve una matriz con los elementos comunes a las matrices indicadas. La comparación se hace con el operador identidad (===)
  - `array_intersect_assoc($mat1,$mat2,$mat3)`
    - Devuelve una matriz con los elementos comunes utilizando el operador identidad (===). En la comparación se tienen en cuenta también las claves.

## 4.2.1.- Modificar un array

- Creación de una matriz con los elementos únicos de otra:
  - `array_unique ($mat)`
    - Crea una nueva matriz a partir de otra original, tomando sólo los elementos no duplicados de ésta. Utiliza el operador de identidad en la comparación.
  - `array_combine ($mat1,$mat2)`
    - Crea un nuevo array a partir de otros dos. Un array le sirve para tomar las claves y el otro para tomar los valores correspondientes. Los dos arrays deben tener el mismo número de elementos.

## 4.2.1.- Modificar un array

- `array_reverse ( $array, true)`
  - Devuelve el array invertido.
  - Si el 2º parámetro es true, conserva las claves

```
$entrada = array ("php", 4, "rojo");
$resultado = array_reverse ($entrada);
$resultado_claves = array_reverse($entrada, true);
```

- `range ( low, high, paso)`
  - Crea una matriz que contiene un rango de elementos paso indica el salto

```
$numeros=range(5,9); □ // (5,6,7,8,9)
$numeros2=range(0,50,10); □ // (0,10,20,30,40,50)
$letras=range(a,f); □ // (a,b,c,d,f)
```

## 4.2.1.- Modificar un array

- `compact(var1,var2,,,,,varN)`
  - Crea un vector asociativo cuyas claves son los nombres de las variables y los valores el contenido de las mismas.

```
$ciudad="miami";
```

```
$edad="23";
```

```
$vec=compact("ciudad","edad");
```

Es equivalente a:

```
$vec=array("ciudad"=>"miami","edad"=>"23");
```

- `shuffle ($array)`
  - Desordena en forma aleatoria los elementos de un array.

## 4.2.2.- Ordenar un array

- Ordenación de arrays:
  - `bool sort ( array &$array [, int $sort_flags = SORT_REGULAR ] )`
    - Ordena un array de menor a mayor
  - `bool rsort ( array &$array [, int $sort_flags = SORT_REGULAR ] )`
    - Ordena un array en orden inverso (de mayor a menor)
  - `bool asort ( array &$array [, int $sort_flags = SORT_REGULAR ] )`
    - Ordena un array manteniendo la correlación de los índices con los elementos asociados.
  - `bool arsort ( array &$array [, int $sort_flags = SORT_REGULAR ] )`
    - Ordena un array en orden inverso, manteniendo la correlación de los índices con los elementos asociados.
  - `bool ksort ( array &$array [, int $sort_flags = SORT_REGULAR ] )`
    - Ordena un array por clave, manteniendo la correlación entre la clave y los datos.

## 4.2.2.- Ordenar un array

- Ordenación de arrays:
  - `bool krsort ( array &$array [, int $sort_flags = SORT_REGULAR ] )`
    - Ordena un array por clave en orden inverso, manteniendo la correlación entre la clave y los datos.
  - `bool usort ( array &$array , callable $value_compare_func )`
    - Ordena un array usando una función de comparación definida por el usuario. Se asignan nuevas claves a los elementos ordenados.
  - `bool uksort ( array &$array , callable $key_compare_func )`
    - Ordena las claves de un array usando una función de comparación proporcionada por el usuario.
  - `bool uasort ( array &$array , callable $value_compare_func )`
    - Ordena un array de manera que los índices mantienen sus correlaciones con los elementos del array asociados, usando una función de comparación definida por el usuario.
  - `bool array_multisort ( array &$arr [, mixed $arg = SORT_ASC [, mixed $arg = SORT_REGULAR [, mixed $... ]]] )`
    - Ordenar varios arrays al mismo tiempo, o un array multi-dimensional por una o más dimensiones. Las claves asociativas (string) se mantendrán, aunque las claves numéricas son re-indexadas.



## 5.- Formularios web

La forma natural para hacer llegar a la aplicación web los datos del usuario desde un navegador, es utilizar **formularios HTML**.

Los formularios HTML van encerrados siempre entre las etiquetas **<FORM>** **</FORM>**. Dentro de un formulario se incluyen los elementos sobre los que puede actuar el usuario, principalmente usando las etiquetas **<INPUT>**, **<SELECT>**, **<TEXTAREA>** y **<BUTTON>**.

El atributo **action** del elemento **FORM** indica la página a la que se le enviarán los datos del formulario. En nuestro caso se tratará de un guión PHP.

Por su parte, el atributo **method** especifica el método usado para enviar la información. Este atributo puede tener dos valores:

- ✓ **get**: con este método los datos del formulario se agregan al URI utilizando un signo de interrogación "?" como separador.
- ✓ **post**: con este método los datos se incluyen en el cuerpo del formulario y se envían utilizando el protocolo HTTP.

Como vamos a ver, los datos se recogerán de distinta forma dependiendo de cómo se envíen.

Crea un formulario HTML para introducir el nombre del alumno y el módulo que cursa, a escoger entre "Desarrollo Web en Entorno Servidor" y "Desarrollo Web en Entorno Cliente". Envía el resultado a la página "procesa.php", que será la encargada de procesar los datos. (Solución en Anexo VI y en el curso de Moodle)

En el curso de moodle encontraras un curso sobre formularios en HTML

## 5.1.- Formularios: Acceso desde PHP

- **Formulario:**

```
<form action="respuesta.php?valor=10" method="post">
<input type="text" name="nombre">
</form>
```

- PHP (¿todos correctos?)

```
<?
echo 'valor: '.$_GET['valor'].'
';
echo 'valor: '.$_POST['valor'].'
'; → vacío
echo 'valor: '.$_REQUEST['valor'].'
';
echo 'nombre: '.$_GET['nombre'].'
'; → vacío
echo 'nombre: '.$_POST['nombre'].'
';
echo 'nombre: '.$_REQUEST['nombre'].'
';
?>
```

## 5.1.- Formularios: Acceso desde PHP

- Acceso a los diferentes tipos de elementos de entrada de formulario
  - Elementos de tipo **INPUT**
    - TEXT
    - RADIO
    - CHECKBOX
    - BUTTON
    - FILE
    - HIDDEN
    - PASSWORD
    - SUBMIT / RESET
  - Elemento **SELECT**
    - Simple / múltiple
  - Elemento **TEXTAREA**

## 5.1.- Formularios: Acceso desde PHP

- TEXT

Introduzca la cadena a buscar:

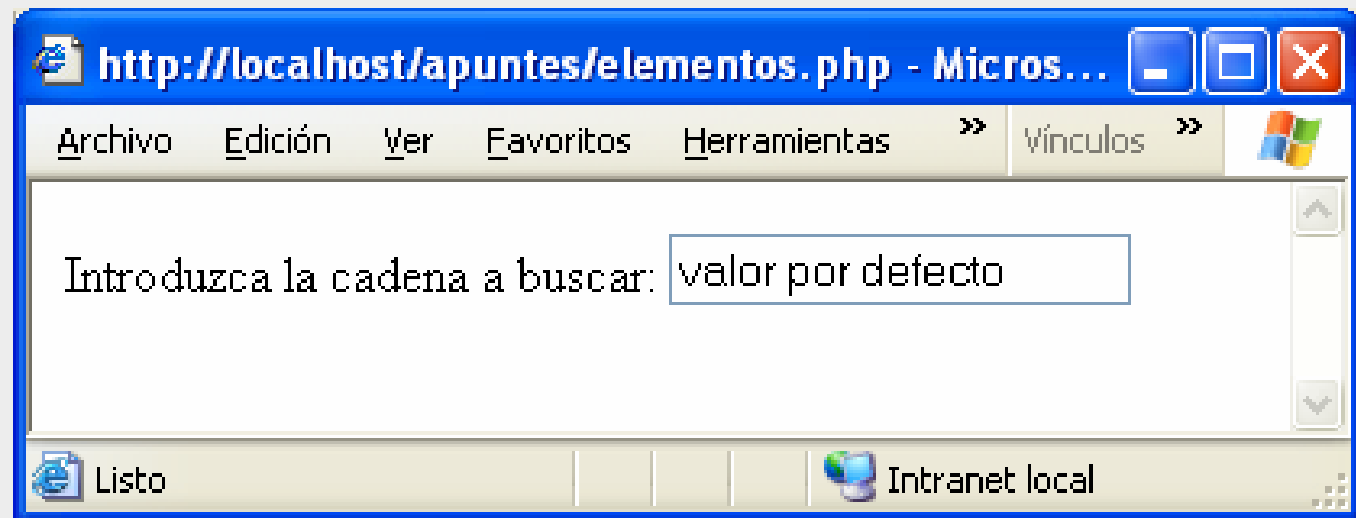
```
<INPUT TYPE="text" NAME="cadena" VALUE="valor por defecto" SIZE="20">
```

```
<?PHP
```

```
 $cadena = $_REQUEST['cadena'];
```

```
 echo $cadena;
```

```
?>
```



## 5.1.- Formularios: Acceso desde PHP

- RADIO

Sexo:

```
<INPUT TYPE="radio" NAME="sexo" VALUE="M" CHECKED>Mujer
```

```
<INPUT TYPE="radio" NAME="sexo" VALUE="H">Hombre
```

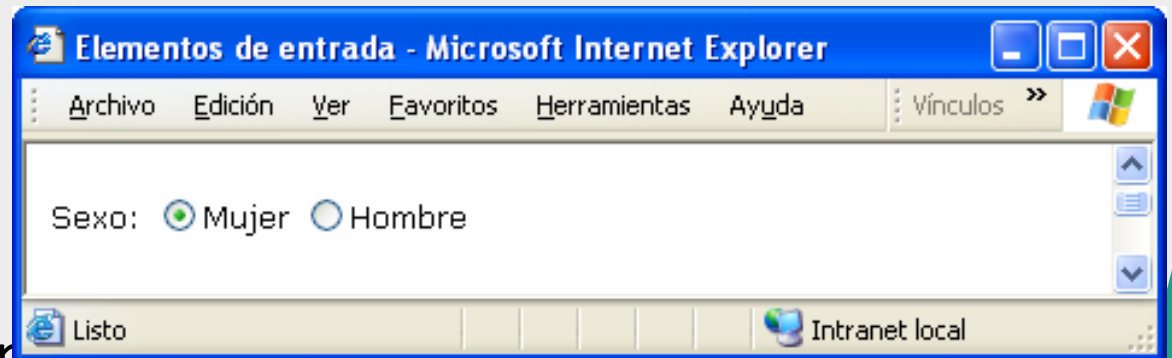
```
<?PHP
```

```
 $sexo = $_REQUEST['sexo'];
```

```
 echo ($sexo);
```

```
?>
```

- Los botones radio se llaman igual para que si se unge uno se desmarquen los otros.



## 5.1.- Formularios: Acceso desde PHP

- **BUTTON**

```
<INPUT TYPE="button" NAME="actualizar" VALUE="Actualizar datos">
```

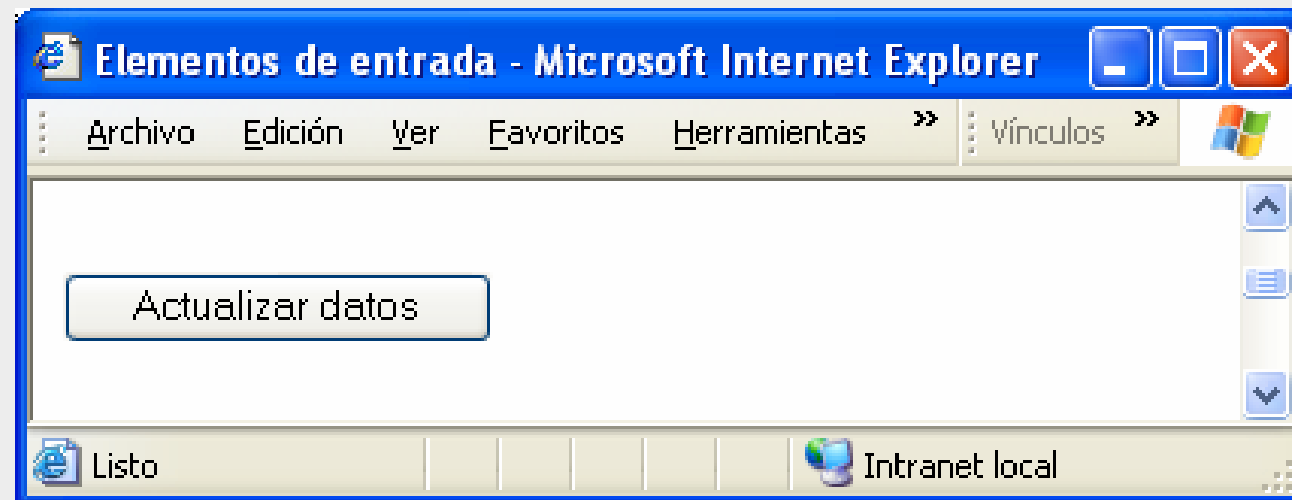
```
<?PHP
```

```
 $actualizar = $_REQUEST['actualizar'];
```

```
 if ($actualizar)
```

```
 print ("Se han actualizado los datos");
```

```
?>
```



## 5.1.- Formularios: Acceso desde PHP

- HIDDEN

```
<INPUT TYPE='hidden' NAME='username' VALUE="$usuario">
```

```
<?PHP
```

```
 $username = $_REQUEST['username'];
```

```
 echo $username;
```

```
?>
```



## 5.1.- Formularios: Acceso desde PHP

- PASSWORD

Contraseña: <INPUT TYPE="password" NAME="clave">

<?PHP

```
$clave = $_REQUEST['clave'];
```

```
echo $clave;
```

```
?>
```





## 5.1.- Formularios: Acceso desde PHP

- SUBMIT

```
<INPUT TYPE=submit NAME="enviar" VALUE="Enviar datos">
```

```
<?PHP
```

```
 $enviar = $_REQUEST['enviar'];
```

```
 if ($enviar)
```

```
 echo "Se ha pulsado el botón de enviar";
```

```
?>
```



## 5.1.- Formularios: Acceso desde PHP

- RESET

```
<INPUT TYPE=reset NAME="borrar" VALUE="Limpiar datos">
```

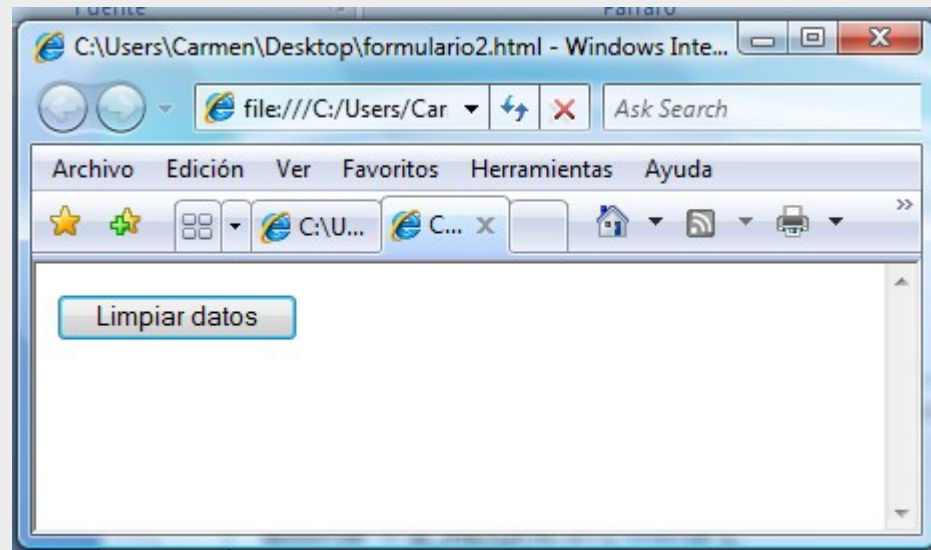
```
<?PHP
```

```
$borrar = $_REQUEST['borrar'];
```

```
if ($borrar)
```

```
 echo "Se ha pulsado el botón de limpiar datos";
```

```
?>
```



## 5.1.- Formularios: Acceso desde PHP

- SELECT Simple

Color:

```
<SELECT NAME="color">
```

```
<OPTION VALUE="rojo" SELECTED>Rojo</OPTION>
```

```
<OPTION VALUE="verde">Verde</OPTION>
```

```
<OPTION VALUE="azul">Azul</OPTION>
```

```
</SELECT>
```

```
<?PHP
```

```
 $color = $_REQUEST['color'];
```

```
 echo $color;
```

```
?>
```



## 5.1.- Formularios: Acceso desde PHP

- SELECT Múltiple

- Valores vectoriales de un formulario:

- Select múltiples o botones de comprobación con el mismo nombre
- Si no se indica nada, sólo se tiene acceso a un valor
- En el código HTML hay que añadir “[]” al nombre del control
- Devuelve un array, con count() podemos conocer su tamaño

Idiomas:

```
<SELECT MULTIPLE SIZE="3" NAME="idiomas[]">
<OPTION VALUE="ingles" SELECTED>Inglés</OPTION>
<OPTION VALUE="frances">Francés</OPTION>
<OPTION VALUE="aleman">Alemán</OPTION>
<OPTION VALUE="holandes">Holandés</OPTION>
</SELECT>
```

```
<?PHP
 $idiomas = $_REQUEST['idiomas'];
 foreach ($idiomas as $idioma)
 echo "$idioma
\n";
```

```
?>
```



## 5.1.- Formularios: Acceso desde PHP

- CHECKBOX

- botones de comprobación con el mismo nombre → usar arrays [ ]

<INPUT TYPE="checkbox" NAME="extras[]" VALUE="garaje" CHECKED>Garaje

<INPUT TYPE="checkbox" NAME="extras[]" VALUE="piscina">Piscina

<INPUT TYPE="checkbox" NAME="extras[]" VALUE="jardin">Jardín

<?PHP

```
$extras = $_REQUEST['extras'];
```

```
foreach ($extras as $extra)
```

```
 echo "$extra
\n";
```

?>



## 5.1.- Formularios: Acceso desde PHP

- TEXTAREA

Comentario:

```
<TEXTAREA COLS="50" ROWS="4" NAME="comentario">
```

Este libro me parece ...

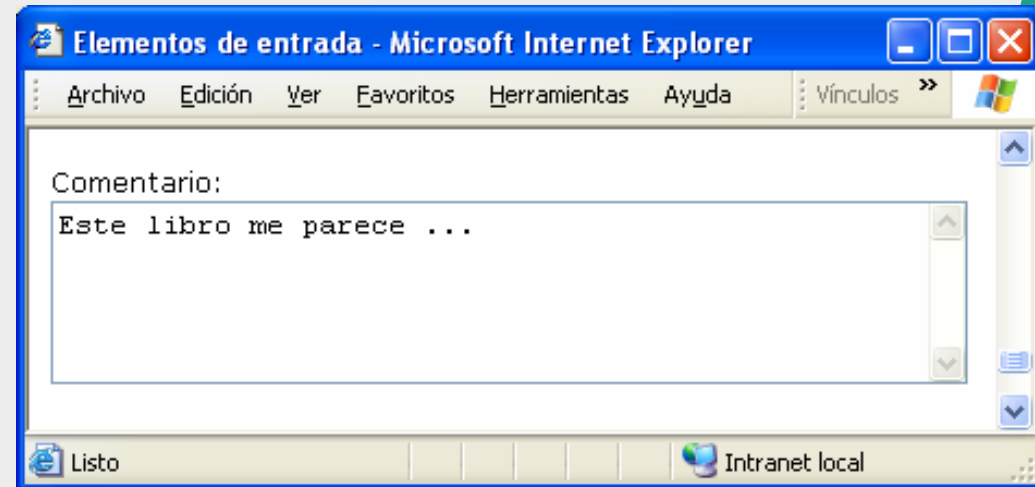
```
</TEXTAREA>
```

```
<?PHP
```

```
 $comentario = $_REQUEST['comentario'];
```

```
 echo $comentario;
```

```
?>
```



- Ejercicio: formulario simple que incluya todos los controles excepto FILE y BUTTON

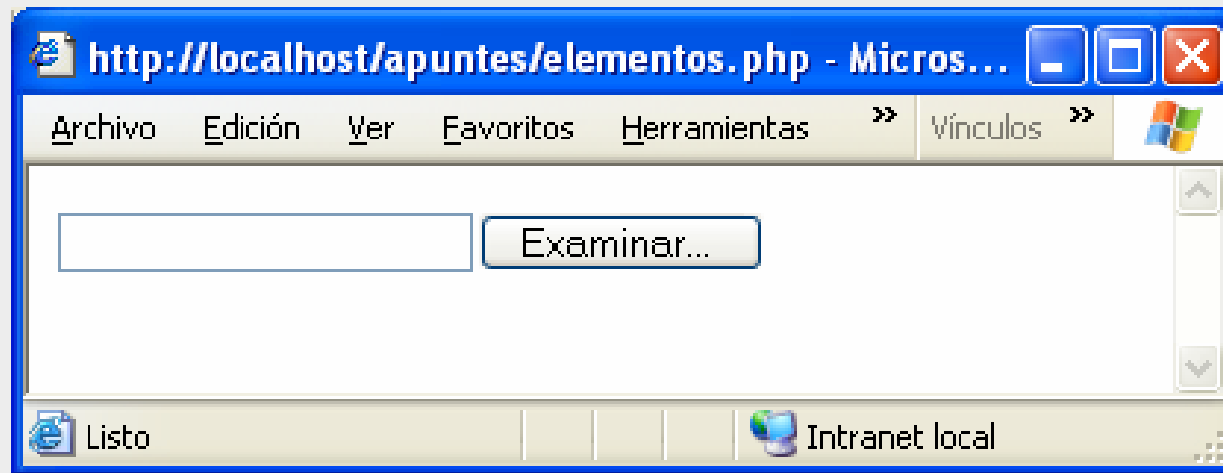
## 5.1.- Formularios: Acceso desde PHP

- FILE

<FORM ACTION="procesa.php" METHOD="post" ENCTYPE="multipart/form-data">

<INPUT TYPE="file" NAME="fichero">

</FORM>



- Ejercicio: página php que muestra los datos introducidos desde el formulario del ejercicio anterior.
  - Ilustra cómo acceder a los valores introducidos desde todos los tipos de elementos de entrada de un formulario, con excepción de los tipos BUTTON y FILE, que se tratan en ejercicios posteriores

## 5.2.- Formularios: Subida de ficheros al servidor

- Para subir un fichero al servidor se utiliza el elemento de entrada FILE
- Hay que tener en cuenta una serie de consideraciones importantes:
  - El elemento FORM debe tener el atributo `ENCTYPE="multipart/form-data"`
  - El fichero tiene un límite en cuanto a su tamaño. Este límite se fija de dos formas diferentes y complementarias:
    - En el fichero de configuración php.ini
    - En el propio formulario



## 5.2.- Formularios: Subida de ficheros al servidor

- php.ini

;;;;;;;;;

; File Uploads ;

;;;;;;;;;

; Si se permite o no subir archivos mediante HTTP

**file\_uploads = On**

;

; Tamaño máximo de **cada archivo subido**.

**upload\_max\_filesize = 2M**

; **Tamaño máximo de los datos mandados por POST**

;(incluidos los que no sean archivos)

**post\_max\_size = 8M**

- formulario

<INPUT TYPE="HIDDEN" NAME="MAX\_FILE\_SIZE" VALUE='102400'>

<INPUT TYPE="FILE" NAME="fichero">

se tiene que llamar así y es un entero con el valor en bytes

## 5.2.- Formularios: Subida de ficheros al servidor

```
<INPUT TYPE="FILE" SIZE="44" NAME="imagen">
```

- La variable `$_FILES` contiene toda la información del fichero subido:
  - `$_FILES['imagen']['name']`
    - Nombre original del fichero en el cliente
  - `$_FILES['imagen']['type']` → ¡Inseguro!, mejor usar `mime_content_type()`
    - Tipo MIME del fichero. Por ejemplo, "image/gif"
  - `$_FILES['imagen']['size']`
    - Tamaño en bytes del fichero subido
  - `$_FILES['imagen']['tmp_name']`
    - Nombre temporal del fichero que se genera para guardar el fichero subido
  - `$_FILES['imagen']['error']`
    - Código de error asociado a la subida del fichero

# ¿Por qué es inseguro usar `$_FILES['imagen']['type']`?

- Este dato es proporcionado por el navegador del cliente, por lo tanto:
  - Dependiendo del navegador la información puede ser correcta o no (algunos navegadores envía todos los archivos con el tipo genérico *application/octet-stream*)
  - Al ser proporcionado por el navegador del cliente y no ser verificado por el servidor php puede ser fácilmente manipulado con fines maliciosos.
- En su lugar debemos usar `mime_content_type()`
  - Es el servidor el que lee los primeros bytes del fichero y deduce qué contenido tiene.

`mime_content_type($_FILES['imagen']['tmp_name'])`

Listado de tipos aquí:

<http://svn.apache.org/repos/asf/httpd/httpd/trunk/docs/conf/mime.types>

## 5.2.- Formularios: Subida de ficheros al servidor

- Consideraciones:
  - Debe darse al fichero un nombre único. Por ello, y como norma general, debe descartarse el nombre original del fichero y crear uno nuevo que sea único, p.e añadiéndole la fecha y hora
  - El fichero subido se almacena en un directorio temporal y hemos de moverlo al directorio de destino usando la función

**move\_upload\_file()**

- Procedimiento:
  - si se ha subido correctamente el fichero
    - Asignar un nombre al fichero
    - Mover el fichero a su ubicación definitiva
  - si no
    - Mostrar un mensaje de error
  - finsi

## 5.2.- Formularios: Subida de ficheros al servidor

- Procedimiento:

si se ha subido correctamente el fichero

- Lo comprobamos con `is_uploaded_file("nombre temporal de $_FILES")`
- Devuelve **true** si el archivo que se le pasa se ha subido por HTTP POST. Evita que el usuario intente usar archivos del servidor `/etc/passwd`

Asignar un nombre al fichero

- Añadir marca de tiempo

Mover el fichero a su ubicación definitiva

- `move_uploaded_file ( $_FILES['archivo'] ['tmp_name'], $destino)`
- Lo mueve y si no puede da error

si no

Mostrar un mensaje de error

finsi

## 5.2.- Formularios: Subida de ficheros al servidor

- Ejemplo(I)

```
<html><body>
```

Inserción de la fotografía del usuario:

```
<form action="inserta.php" method="post" enctype="multipart/form-data">
```

```
<?php
```

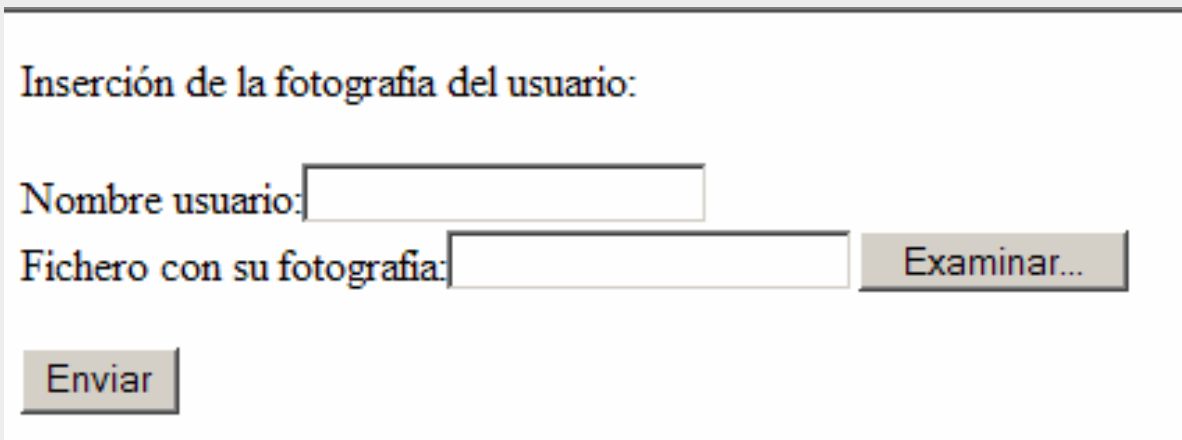
```
echo "Nombre usuario:<input type='text' name='usuario'/>
";
```

```
echo "Fichero con su fotografía:<input type='file' name='imagen'/>
";
```

```
?>
```

```
<input type="submit" value="Enviar">
```

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



Inserción de la fotografía del usuario:

Nombre usuario:

Fichero con su fotografía:

## 5.2.- Formularios: Subida de ficheros al servidor

- Ejemplo(II) - inserta.php

```
<html><body>
```

```
<?php
```

```
echo "name:".$_FILES['imagen']['name']."\n";
```

```
echo "tmp_name:".$_FILES['imagen']['tmp_name']."\n";
```

```
echo "size:".$_FILES['imagen']['size']."\n";
```

```
echo "type:".$_FILES['imagen']['type']."\n";
```

```
if (is_uploaded_file ($_FILES['imagen']['tmp_name'])){
```

```
 $nombreDirectorio = "img/";
```

```
 $nombreFichero = $_FILES['imagen']['name'];
```

```
 if (is_dir($nombreDirectorio)){ // es un directorio existente
```

```
 $idUnico = time();
```

```
 $nombreFichero = $idUnico."-".$nombreFichero;
```

```
 $nombreCompleto = $nombreDirectorio.$nombreFichero;
```

```
 move_uploaded_file ($_FILES['imagen']['tmp_name'],$nombreCompleto);
```

```
 echo "Fichero subido con el nombre: $nombreFichero
";
```

```
 }
```

```
 else
```

```
 echo 'Directorio definitivo inválido';
```

```
 } else
```

```
 print ("No se ha podido subir el fichero\n");
```

```
?>
```

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

```
name:Foto.png
```

```
tmp_name:C:\xampp\tmp\php7EE.tmp
```

```
size:15811
```

```
type:image/x-png
```

```
Fichero subido con el nombre: 1241894493-Foto.png
```

## 5.2.- Formularios: Subida de ficheros al servidor

- **is\_uploaded\_file (\$\_FILES['imagen']['tmp\_name'])**
  - Devuelve TRUE si el archivo que se pasa fue cargado a través de HTTP POST. Evita que un usuario intente que se manejen ficheros no cargados por POST. p.e /etc/passwd
  - Necesita como argumento \$\_FILES['archivo\_usuario']['tmp\_name']
  - Si se le pasa \$\_FILES['archivo\_usuario']['name'] **no funciona**.

<?php

```
if (is_uploaded_file($_FILES['archivo_usuario']['tmp_name'])) {
 echo "El archivo ". $_FILES['archivo_usuario']['name'] ." fue cargado correctamente.\n";
 echo "Mostrando su contenido\n";
 readfile($_FILES['archivo_usuario']['tmp_name']);
} else {
 echo "Posible ataque de carga de archivo: ";
 echo "nombre de archivo '". $_FILES['archivo_usuario']['tmp_name'] . "'.";}
```

?>



## 5.2.- Formularios: Subida de ficheros al servidor

- `move_uploaded_file ($_FILES['imagen']['tmp_name'],$destino)`  
nombre\_temporal\_archivo
- Esta función realiza un chequeo para asegurar que el archivo indicado por el primer parámetro sea un archivo cargado a través de HTTP POST.
- Si el archivo es válido, será movido al nombre de archivo dado por **destino**.
- Si *nombre\_temporal\_archivo* no es un archivo cargado válido, no hará nada, y devolverá FALSE.
- Si *nombre\_temporal\_archivo* es un archivo cargado válido, pero no puede ser movido por alguna razón, no hará nada, devolverá FALSE y dará una advertencia.

## 5.2.- Formularios: Subida de ficheros al servidor

- Variable predefinida `$_FILES`
  - Precauciones:
    - Permisos de escritura en el directorio temporal
    - Permisos de escritura en el directorio de destino
    - Atención con los ficheros que puedan subir los usuarios: Troyanos, scripts, ejecutables, etc.
- Ejercicio: subida de un fichero al servidor
  - Ilustra cómo subir ficheros a un servidor, cómo controlar su tamaño, cómo crear un nombre único para el fichero y cómo almacenarlo en el lugar deseado .

## 5.3.- Procesamiento de la información devuelta por un formulario web

En el ejemplo anterior creaste un formulario en una página HTML que recogía datos del usuario y los enviaba a una página PHP para que los procesara. Como usaste el método POST, los datos se pueden recoger utilizando la variable `$_POST`. Si simplemente los quisieras mostrar por pantalla, éste podría ser el código de "procesa.php":

Consulta el Anexo VII con el Código de "procesa.php"

Si por el contrario hubieras usado el método GET, el código necesario para procesar los datos sería similar; simplemente haría falta cambiar la variable `$_POST` por `$_GET`.

Consulta el Anexo VIII con el Código necesario para procesar los datos

En cualquiera de los dos casos podrías haber usado `$_REQUEST` sustituyendo respectivamente a `$_POST` y a `$_GET`.

Consulta el Anexo IX Usando `$_REQUEST` para sustituir respectivamente a `$_POST` y a `$_GET`

## 5.3.- Procesamiento de la información devuelta por un formulario web

Siempre que sea posible, es preferible **validar los datos que se introducen en el navegador antes de enviarlos**. Para ello deberás usar código en **lenguaje Javascript**.

Si por algún motivo hay datos que se tengan que validar en el servidor, por ejemplo, porque necesites comprobar que los datos de un usuario no existan ya en la base de datos antes de introducirlos, será necesario hacerlo con código PHP en la página que figura en el atributo **action** del formulario.

En este caso, una posibilidad que deberás tener en cuenta es usar la misma página que muestra el formulario como destino de los datos. Si tras comprobar los datos éstos son correctos, se reenvía a otra página. Si son incorrectos, se rellenan los datos correctos en el formulario y se indican cuáles son incorrectos y por qué.

Para hacerlo de este modo, tienes que comprobar si la página recibe datos (hay que mostrarlos y no generar el formulario), o si no recibe datos (hay que mostrar el formulario). Esto se puede hacer utilizando la función **isset** con una variable de las que se deben recibir (por ejemplo, poniéndole un nombre al botón de enviar y comprobando sobre él). En el siguiente código de ejemplo se muestra cómo hacerlo.

Consulta el Anexo X que muestra como Procesar datos en la misma página que el formulario

Fíjate en la forma de englobar el formulario dentro de una sentencia **else** para que sólo se genere si no se reciben datos en la página. Además, para enviar los datos a la misma página que contiene el formulario puedes usar **\$\_SERVER['PHP\_SELF']** para obtener su nombre; esto hace que no se produzca un error aunque la página se cambie de nombre.

## 5.4.- Generación de formularios web en PHP

Vamos a volver sobre el ejemplo anterior, revisando los datos que se obtienen antes de mostrarlos. Concretamente, tienes que comprobar que el nombre no esté vacío, y que se haya seleccionado como mínimo uno de los módulos.

Además, en el caso de que falte algún dato, deberás generar el formulario rellenando aquellos datos que el usuario haya introducido correctamente.

Lo primero que tienes que hacer es la validación de los datos. En el ejemplo propuesto será algo así como:

```
if (!empty($_POST['modulos']) && !empty($_POST['nombre'])) {
 // Aquí se incluye el código a ejecutar cuando los datos son correctos
} else {
 // Aquí generamos el formulario, indicando los datos incorrectos
 // y rellenando los valores correctamente introducidos
}
```

Para que el usuario no pierda, después de enviar el formulario, los datos correctamente introducidos, utiliza el atributo **value** en las entradas de texto (utilizamos la función **isset** para comprobar que la variable que queremos mostrar existe, para que no nos muestre un mensaje de error si es la primera vez que se carga la página y todavía no ha enviado nada el formulario):

*Nombre del alumno:*

```
<input type="text" name="nombre" value="<?php if (isset ($_POST['nombre'])) echo $_POST['nombre'];?>" />
```

## 5.4.- Generación de formularios web en PHP

Y el atributo **checked** en las casillas de verificación:

```
<input type="checkbox" name="modulos[]" value="DWES"
 <?php
 if(in_array("DWES",$_POST['modulos']))
 echo 'checked="checked"';
 ?>
/>
```

Fíjate en el uso de la función **in\_array** para buscar un elemento en un array.

Para indicar al usuario los datos que no ha rellenado (o que ha rellenado de forma incorrecta), deberás comprobar si es la primera vez que se visualiza el formulario, o si ya se ha enviado. Se puede hacer por ejemplo de la siguiente forma:

*Nombre del alumno:*

```
<input type="text" name="nombre" value="<?php echo $_POST['nombre'];?>" />
<?php
 if (isset($_POST['enviar']) && empty($_POST['nombre']))
 echo " <-- Debe introducir un nombre!!"
?>

```

Revisa el documento con el ejemplo completo, fijándote en las partes que hemos comentado anteriormente.

Consulta el Anexo XI que contiene el Documento con el ejemplo anterior completo

Una forma de enviar información de una página PHP a otra, es incluyéndola en campos ocultos dentro de un formulario.

Modifica el ejercicio que mostraba la fecha en castellano, para que obtenga lo mismo a partir de un día, mes y año introducido por el usuario. Antes de mostrar la fecha, se debe comprobar que es correcta. Utilizar la misma página PHP para el formulario de introducción de datos y para mostrar la fecha obtenida en castellano.

Consultar las funciones `checkdate` y `mktime` en el manual de PHP. (Solución en Anexo XII y en el curso de Moodle)

## 6.- Cabeceras HTTP: Introducción

- Cabeceras HTTP
  - Aportan información en peticiones y respuestas a servidores
  - Función para enviar cabeceras HTTP:  

```
header("cabecera: valor");
```
  - Ejemplos:
    - `header("location: http://www.upm.es");`
    - `header("HTTP/1.0 404 Not Found");`
    - `header("Pragma: no-cache");`
    - Otras: Cache-Control, Expires, Last-Modified, etc.
- Es importante que esta orden se encuentre antes de la etiqueta `<html>` inicial.
- Las cabeceras HTTP pueden también modificar el comportamiento del navegador que recibe la respuesta

## 6.- Cabeceras HTTP: Introducción

- Enviar cabeceras HTTP en PHP: `header()`
  - Debe aparecer antes de enviar cualquier otra cosa al cliente (antes de cualquier echo).
  - `header ('Content-Type: text/html; charset=UTF-8');`
- Extraer cabeceras HTTP del cliente en PHP:  
`apache_request_headers()`

```
<?php
 var_dump(apache_request_headers());
?>
```
- Redirigir al cliente a otra dirección.  
`header ('Location: acceso_no_autorizado.php');`
- Mostrar un mensaje y redirigir al cliente a otra dirección.  
`header ('Refresh: 5; url=http://www.google.es');`  
`echo 'Lo que busca no existe, le redirigiremos a Google en 5 segundos'`
- Ocultar la versión de nuestro intérprete PHP.  
`header( 'X-Powered-By: adivina-adivinanza' );`
- Alternativamente también podríamos ocultar nuestra versión de PHP asignando el valor 0 a la directiva `expose_PHP` del `php.ini`.



## 6.- Cabeceras HTTP: Introducción

- Ofrecer la descarga de un archivo desde PHP.

Vamos a mostrar un pdf

```
header('Content-type: application/pdf');
header('Content-Disposition: attachment;
filename="downloaded.pdf"');
readfile('original.pdf');
```

Proporciona un nombre de fichero recomendado y fuerza al navegador el mostrar el diálogo para guardar el fichero.

La fuente del PDF se encuentra en original.pdf

## 6.- Cabeceras HTTP: Introducción

- Generar contenidos diferentes a páginas HTML con PHP.
  - Imágenes, documentos PDF, películas SWF de Flash en tiempo real (sin leerlos de un archivo)
  - Es posible gracias a librerías de PHP como GD, PDFlib, Ming, etc.

```
header("Content-type:image/jpeg");
```

```
header("Content-Disposition:inline ; filename=captcha.jpg");
```



El fichero se abre en el navegador  
en lugar de descargarse  
(attachment)

# 7.- Ficheros y Directorios

- Un fichero
  - Es un almacén de datos en dispositivos externos
  - Convierte los datos en persistentes al final de los programas
- Tipos de ficheros:
  - TEXTOS:
    - La información se almacena en ASCII, es legible mediante editores, los datos requieren separadores (' ', '\n', '\t', '\r\n')
  - BINARIOS:
    - La información es binaria, NO es legible mediante editores, los datos NO requieren separadores

## 7.- Ficheros y Directorios

- Las operaciones sobre ficheros suelen constar de tres fases:
  - Apertura del fichero
    - Se abre el fichero, indicando si se realizarán operaciones para leer, escribir o añadir al final del mismo.
    - La operación devuelve un descriptor de fichero que se usará en el resto de funciones.
  - Procesamiento del fichero
    - Lectura
    - Escritura.
  - Cierre del fichero.

## 7.1.- Apertura y Cierre

- Apertura **fopen()**

`fopen( nombre_fichero, modo, include_path)`

- nombre\_fichero:

- local o remoto (“http://” o “ftp://”)

- modo:

- ‘r’ Sólo lectura. Puntero al inicio.
- ‘r+’ Lectura/escritura. Puntero al inicio.
- ‘w’ Sólo escritura. Puntero al inicio. Si existe el fichero borra lo que había, sino lo intenta crear.
- ‘w+’ Lectura/escritura. Puntero al principio. Si existe el fichero borra lo que había, sino lo intenta crear.
- ‘a’ Sólo escritura. Puntero al final. Si no existe lo intenta crear.
- ‘a+’ Lectura/escritura. Puntero al final.
- ‘x’, ‘x+’ Igual que w, w+ pero si existe da un error.
- ‘c’, ‘c+’ Igual que w , w+ pero sin truncar el fichero cuando ya existe.

## 7.1.- Apertura y Cierre

- Apertura **fopen()**
  - Devuelve un **identificador** que se emplea en el resto de funciones (o FALSE en caso de error)
  - **Include\_path** Especifica la lista de directorios donde las funciones require, include, fopen(), file(), readfile() y file\_get\_contents() buscarán ficheros.
  - **Include\_path** = true -> el fichero debe buscarse en las rutas establecidas en la directiva include\_path de php.ini.
    - Ejemplo en Windows: include\_path=".;c:\php\includes"
    - Ejemplo en Unix: include\_path="./php/includes"
  - Por portabilidad, se recomienda encarecidamente que siempre use la bandera 'b' cuando se abran ficheros binarios con fopen().
- Cierre **fclose()**

fclose(identificador)

## 7.1.- Apertura y Cierre

- ¿Para qué sirve el puntero que devuelve fopen()?
  - Define un canal a través del cual se accede al fichero.
  - Desde que el fichero está abierto se trabaja con el puntero.
  - Cuando se abre el fichero, el puntero se coloca al principio del fichero para esperar instrucciones.
- Ejemplos:
  - Apertura para lectura  
`$fichero = fopen("datos.txt", 'r');`
  - Apertura para escritura silenciando errores con @  
`$fichero = @fopen("datos.txt", 'w');`
  - Apertura para añadir, silenciando errores  
`$fichero = @fopen("datos.txt", 'a');`
- Más ejemplos:
  - `$gestor = fopen("/home/rasmus/fichero.txt", "r");`
  - `$gestor = fopen("/home/rasmus/fichero.gif", "wb");`
  - `$gestor = fopen("http://www.example.com/", "r");`
  - `$gestor = fopen("ftp://user:password@example.com/fichero.txt", "w");`

## 7.1.- Apertura y Cierre

- Para verificar que la operación `fopen()` ha tenido éxito:
  - Apertura para lectura silenciando errores con el operador `@`  
`$fichero = @fopen("datos.txt", 'r');`
  - Si no se ha podido abrir el fichero finaliza la ejecución del script devolviendo un mensaje de error:

```
if (!$fichero)
```

```
 die("ERROR: no se ha podido abrir el fichero de datos");
```

- `Die()` → Provoca la finalización de la ejecución del script mostrando el mensaje recibido como parámetro
- En caso contrario el script continúa ejecutándose.



## 7.2.- Procesamiento de ficheros

- Leer
  - `string fgets ($puntero, [bytes])`
    - Obtiene una línea desde el puntero del fichero
    - Se termina de leer cuando llega al final de línea, final del fichero o el último byte de datos
    - `byte` indica cuantos bytes (caracteres) queremos leer del fichero (opcional)
  - `string fread ($puntero, longitud)`
    - Es similar a `fgets( )`, pero se lee todo el fichero (o hasta el carácter longitud), no va línea por línea como `fgets( )`.
- Leer todo el contenido y almacenar cada línea en una posición del array que devuelve.
  - `file (nombre_fichero)`
- Leer todo el contenido y devolverlo en un string.
  - `file_get_contents (nombre_fichero)`

## 7.2.- Procesamiento de ficheros

- Ejemplo: leer el contenido de un fichero línea a línea:

```
$a = fopen('datos.txt', 'r');
while(!feof($a)){
 echo fgets($a) . '
';
}
fclose($a);
```

- Ejemplo: leer el contenido de un fichero de una vez

```
$a = file('datos.txt');
foreach($a as $linea)
 echo $linea . '
';
```

- Ejemplo: mostrar una página web de otro sitio (las imágenes fallan si las URL son relativas)

```
$a =file('http://www.upm.es/index.html');
foreach($a as $linea)
 echo $linea;
```

- Ejercicio:

- Realiza una página llamada lectura.php en la que lea el contenido de una de las páginas web hechas hasta ahora y lo muestre por pantalla.

## 7.2.- Procesamiento de ficheros

- Escribir (w ó a )
  - `fwrite( $fichero, cadena, longitud)`
    - Escribe en *\$fichero* los caracteres de cadena.
    - Si se añade el parámetro longitud, escribe hasta que termine la cadena o hasta que se alcancen los caracteres indicados en este parámetro, lo que antes ocurra.
    - Devuelve el número de caracteres escrito
  - `fputs` es una alias de `fwrite`.
- Verificar el final de fichero
  - `feof(identificador)` (TRUE -> fin, FALSE -> no fin)
- `rewind( identificador)`
  - permite colocar el puntero al principio del fichero.

```
$a = fopen('datos.txt', 'w+');
fwrite($a,"nueva linea\r\n");
fputs($a,"otra linea\r\n");
fclose($a);
```

## 7.2.- Procesamiento de ficheros

- Otras operaciones:
  - `file_exists()`
    - Determina si existe un archivo o directorio
  - `fgetss()`
    - Idéntica a `fgets` con la diferencia de que los tags html son eliminados del archivo a medida que se lee el mismo.
    - Opcionalmente puede pasarse una lista de tags que no deben ser eliminados.
    - Ejemplo:  
`$string=fgetss($des,999999,"<b> <i> <table> <tr> <td>");`
    - Lee una línea (de cualquier longitud) eliminando los tags html excepto los indicados como segundo parámetro. Los tags que cierran éstos tampoco son eliminados. (`</td>`,...)
  - `readfile(path)`
    - Lee e imprime un archivo

## 7.2.- Procesamiento de ficheros

- Otras operaciones:
  - `copy (origen, destino)`
    - Copiar un fichero
  - `rename (nombre_original, nombre_final)`
    - Renombrar (o mover) un fichero
  - `unlink(fichero)`
    - Borrar un fichero
  - `ftruncate (descriptor, longitud)`
    - Trunca el archivo a la longitud en bytes dada.
  - `filesize(path)`
    - Tamaño de un fichero en bytes
  - `filemtime(path)`
    - Devuelve marca de tiempo en que se modificó por última vez el fichero.
    - Es necesario formatearla con date

## 7.2.- Procesamiento de ficheros

- Otras operaciones .
  - **chgrp** : Cambia el grupo de un archivo.
  - **chmod** : Cambia permisos de un archivo
  - **chown** : Cambia el propietario de un archivo
  - **is\_executable** : Indica si el archivo es ejecutable
  - **is\_file** : Indica si el archivo es un archivo regular
  - **is\_link** : Indica si el archivo es un enlace simbólico
  - **is\_readable** : Indica si es posible leer el archivo
  - **is\_uploaded\_file** : Indica si un archivo fue cargado a través de HTTP POST
  - **is\_writable** : Indica si el nombre de archivo es escribible
  - **is\_writeable** : Alias de **is\_writable**
  - **filetype(path)** : Devuelve el tipo de un archivo.

## 7.2.- Procesamiento de ficheros

- Otras operaciones .
  - `int exif_imagetype ( string $filename )`: Lee los primeros bytes de una imagen y comprueba su firma.
  - devuelve el valor de la constante apropiada al tipo o FALSE si no se reconoce su tipo
  - Algunas constantes utilizadas por la función:

1	IMAGETYPE_GIF
2	IMAGETYPE_JPEG
3	IMAGETYPE_PNG
4	IMAGETYPE_SWF
5	IMAGETYPE_PSD
6	IMAGETYPE_BMP

## 7.2.- Procesamiento de ficheros

- `int exif_imagetype ( string $filename )`

```
<?php
```

```
 if (exif_imagetype('imagen.gif') != IMAGETYPE_GIF) {
```

```
 echo 'La imagen no es gif';
```

```
 }
```

```
?>
```



## 7.3.- Manejo de directorios

- Similar al procesamiento de ficheros secuenciales:
  - Se abre directorio
  - Después se procesa cada entrada del mismo
  - Se cierra el directorio

## 7.3.- Manejo de directorios

- Funciones:
  - Determinar la existencia
    - `is_dir( directorio )`: Determina si existe y es un directorio
  - Apertura
    - `opendir( directorio )`: Abre directorio y devuelve un descriptor
  - Lectura
    - `readdir($descriptor)`:
      - Devuelve el nombre del siguiente fichero en el directorio.
      - Los nombres de archivo son devueltos en el orden en que están en el sistema de archivos. ¡Ojo! los primeros “.” y “..”
  - Cierre
    - `closedir( $descriptor )`: Cierra el recurso \$descriptor

## 7.3.- Manejo de directorios

- Ejemplo: Mostrar los ficheros de un directorio con su tamaño:

```
$dir = opendir("."); //abre el directorio actual
while(false != ($fichero = readdir($dir))){
 if (is_dir($fichero))
 echo "Directorio: $fichero
";
 else
 echo "$fichero: " .filesize($fichero).'bytes
';}
closedir($dir);
```

- **chdir(directorio)** : Cambia de directorio
- **getcwd(directorio)** :Devuelve el directorio actual
- **mkdir(directorio)** : Crea un directorio
- **rmdir(directorio)** : Elimina un directorio
- **basename(path)** : Devuelve la parte del path correspondiente al nombre del archivo
- **rewinddir(directorio)**: Rebobina el puntero interno de un directorio para que apunte nuevamente al comienzo del mismo.

# Anexo I.- Utilización de la función print en PHP

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "
http://www.w3.org/TR/html4/loose.dtd">
<!-- Desarrollo Web en Entorno Servidor -->
<!-- Tema 2 : Características del Lenguaje PHP -->
<!-- Ejemplo: Utilización de print -->
<html>
 <head>
 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
 <title>Desarrollo Web</title>
 </head>
<body>
 <?php
 $modulo="DWES";
 print "<p>Módulo: ";
 print $modulo;
 print"</p>"
 ?>
</body>
</html>
```

# Anexo II.- Solución propuesta I

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "
http://www.w3.org/TR/html4/loose.dtd">
<!-- Desarrollo Web en Entorno Servidor -->
<!-- Tema 2 : Características del Lenguaje PHP -->
<!-- Ejemplo: Mostrar fecha en castellano -->
<html>
 <head>
 <meta http-equiv="content-type" content="text/html; charset=UTF-8">
 <title>Fecha en castellano</title>
 </head>
 <body>
<?php
 date_default_timezone_set('Europe/Madrid');
 $numero_mes = date("m");
 $numero_dia_semana = date("N");
 switch($numero_mes) {
 case 1: $mes = "Enero";
 break;
 case 2: $mes = "Febrero";
 break;
 case 3: $mes = "Marzo";
 break;
 case 4: $mes = "Abril";
 break;
 case 5: $mes = "Mayo";
 break;
 case 6: $mes = "Junio";
 break;
 case 7: $mes = "Julio";
 break;
 case 8: $mes = "Agosto";
 break;
 case 9: $mes = "Septiembre";
 break;
 case 10: $mes = "Octubre";
 break;
 case 11: $mes = "Noviembre";
 break;
 case 12: $mes = "Diciembre";
 break;
 }
</body>
</html>
```

# Anexo II.- Solución propuesta I

```
switch($numero_dia_semana) {
 case 1: $dia_semana = "Lunes";
 break;
 case 2: $dia_semana = "Martes";
 break;
 case 3: $dia_semana = "Miércoles";
 break;
 case 4: $dia_semana = "Jueves";
 break;
 case 5: $dia_semana = "Viernes";
 break;
 case 6: $dia_semana = "Sábado";
 break;
 case 7: $dia_semana = "Domingo";
 break;
}
print $dia_semana.", ".date("j")." de ".$mes." de ".date("Y");
?>
</body>
</html>
```

# Anexo III.- Solución propuesta II

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "
http://www.w3.org/TR/html4/loose.dtd">
<!-- Desarrollo Web en Entorno Servidor -->
<!-- Tema 2 : Características del Lenguaje PHP -->
<!-- Ejemplo: Utilización include -->
<html>
 <head>
 <meta http-equiv="content-type" content="text/html; charset=UTF-8">
 <title>Fecha en castellano</title>
 </head>
 <body>
<?php
 include 'funciones.inc.php';
 print fecha();
?>
 </body>
</html>
```

# Anexo III.- Solución propuesta II

```
<!-- Desarrollo Web en Entorno Servidor -->
<!-- Tema 2 : Características del Lenguaje PHP -->
<!-- Ejemplo: Utilización include -->
<!-- Fichero: funciones.inc.php -->
<?php
 // Función que devuelve un texto con la fecha actual en castellano
 function fecha()
 {
 date_default_timezone_set('Europe/Madrid');
 $numero_mes = date("m");
 $numero_dia_semana = date("N");
 switch($numero_mes) {
 case 1: $mes = "Enero";
 break;
 case 2: $mes = "Febrero";
 break;
 case 3: $mes = "Marzo";
 break;
 case 4: $mes = "Abril";
 break;
 case 5: $mes = "Mayo";
 break;
 case 6: $mes = "Junio";
 break;
 case 7: $mes = "Julio";
 break;
 case 8: $mes = "Agosto";
 break;
 case 9: $mes = "Septiembre";
 break;
 case 10: $mes = "Octubre";
 break;
 case 11: $mes = "Noviembre";
 break;
 case 12: $mes = "Diciembre";
 break;
 }
 }
}
```



# Anexo III.- Solución propuesta II

```
switch($numero_dia_semana) {
 case 1: $dia_semana = "Lunes";
 break;
 case 2: $dia_semana = "Martes";
 break;
 case 3: $dia_semana = "Miércoles";
 break;
 case 4: $dia_semana = "Jueves";
 break;
 case 5: $dia_semana = "Viernes";
 break;
 case 6: $dia_semana = "Sábado";
 break;
 case 7: $dia_semana = "Domingo";
 break;
}
return $dia_semana.", ".date("j")." de ".$mes." de ".date("Y");
}
?>
```

# Anexo IV.- Solución propuesta III

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "
http://www.w3.org/TR/html4/loose.dtd">
<!-- Desarrollo Web en Entorno Servidor -->
<!-- Tema 2 : Características del Lenguaje PHP -->
<!-- Ejemplo: Tabla con los valores del array $_SERVER utilizando foreach -->
<html>
 <head>
 <meta http-equiv="content-type" content="text/html; charset=UTF-8">
 <title>Tabla</title>
 <style type="text/css">
 td, th {border: 1px solid grey; padding: 4px;}
 th {text-align:center;}
 table {border: 1px solid black;}
 </style>
 </head>
 <body>
 <table>
 <tbody>
 <tr>
 <th>Variable</th>
 <th>Valor</th>
 </tr>
 <?php
 foreach ($_SERVER as $variable => $valor) {
 print "<tr>";
 print "<td>".$variable."</td>";
 print "<td>".$valor."</td>";
 print "</tr>";
 }
 ?>
 </tbody>
 </table>
 </body>
</html>
```

# Anexo V.- Solución propuesta IV

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "
http://www.w3.org/TR/html4/loose.dtd">
<!-- Desarrollo Web en Entorno Servidor -->
<!-- Tema 2 : Características del Lenguaje PHP -->
<!-- Ejemplo: Tabla con los valores del array $_SERVER utilizando función each -->
<html>
 <head>
 <meta http-equiv="content-type" content="text/html; charset=UTF-8">
 <title>Tabla</title>
 <style type="text/css">
 td, th {border: 1px solid grey; padding: 4px;}
 th {text-align:center;}
 table {border: 1px solid black;}
 </style>
 </head>
 <body>
 <table>
 <tbody>
 <tr>
 <th>Variable</th>
 <th>Valor</th>
 </tr>
 <?php
 reset($_SERVER);
 while ($valor = each($_SERVER)) {
 print "<tr>";
 print "<td>".$_SERVER[0]."</td>";
 print "<td>".$_SERVER[1]."</td>";
 print "</tr>";
 }
 ?>
 </tbody>
 </table>
 </body>
</html>
```

# Anexo VI.- Solución propuesta V

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "
http://www.w3.org/TR/html4/loose.dtd">
<!-- Desarrollo Web en Entorno Servidor -->
<!-- Tema 2 : Características del Lenguaje PHP -->
<!-- Ejemplo: Formulario web -->
<html>
 <head>
 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
 <title>Formulario web</title>
 </head>
 <body>
 <form name="input" action="procesa.php" method="post">
 Nombre del alumno: <input type="text" name="nombre" />

 <p>Ciclos que cursa:</p>
 <input type="checkbox" name="modulos[]" value="DWES" /> Desarrollo web en entorno servidor

 <input type="checkbox" name="modulos[]" value="DWECE" /> Desarrollo web en entorno cliente

 <input type="submit" value="Enviar" />
 </form>
 </body>
</html>
```

# Anexo VII.- Código de “procesa.php”

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "
http://www.w3.org/TR/html4/loose.dtd">
<!-- Desarrollo Web en Entorno Servidor -->
<!-- Tema 2 : Características del Lenguaje PHP -->
<!-- Ejemplo: Procesar datos post -->
<html>
 <head>
 <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
 <title>Desarrollo Web</title>
 </head>
 <body>
 <?php
 $nombre = $_POST['nombre'];
 $modulos = $_POST['modulos'];
 print "Nombre: ".$nombre."
";
 foreach ($modulos as $modulo) {
 print "Modulo: ".$modulo."
";
 }
 ?>
 </body>
</html>
```

# Anexo VIII.- Código necesario para procesar datos

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "
http://www.w3.org/TR/html4/loose.dtd">
<!-- Desarrollo Web en Entorno Servidor -->
<!-- Tema 2 : Características del Lenguaje PHP -->
<!-- Ejemplo: Procesar datos get -->
<html>
 <head>
 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
 <title>Desarrollo Web</title>
 </head>
 <body>
<?php
 $nombre = $_GET['nombre'];
 $modulos = $_GET['modulos'];
 print "Nombre: ".$nombre."
";
 foreach ($modulos as $modulo) {
 print "Modulo: ".$modulo."
";
 }
?>
 </body>
</html>
```

# Anexo IX.- Ejemplo de formulario web utilizando request

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "
http://www.w3.org/TR/html4/loose.dtd">
<!-- Desarrollo Web en Entorno Servidor -->
<!-- Tema 2 : Características del Lenguaje PHP -->
<!-- Ejemplo: Procesar datos request -->
<html>
 <head>
 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
 <title>Desarrollo Web</title>
 </head>
 <body>
 <?php
 $nombre = $_REQUEST['nombre'];
 $modulos = $_REQUEST['modulos'];
 print "Nombre: ".$nombre."
";
 foreach ($modulos as $modulo) {
 print "Modulo: ".$modulo."
";
 }
 ?>
 </body>
</html>
```

# Anexo X.- Procesar datos en la misma página que el formulario

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "
http://www.w3.org/TR/html4/loose.dtd">
<!-- Desarrollo Web en Entorno Servidor -->
<!-- Tema 2 : Características del Lenguaje PHP -->
<!-- Ejemplo: Procesar datos en la misma página que el formulario -->
<html>
 <head>
 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
 <title>Desarrollo Web</title>
 </head>
 <body>
 <?php
 if (isset($_POST['enviar'])) {
 $nombre = $_POST['nombre'];
 $modulos = $_POST['modulos'];
 print "Nombre: ".$nombre."
";
 foreach ($modulos as $modulo) {
 print "Modulo: ".$modulo."
";
 }
 }
 else {
?>
 <form name="input" action="<?php echo $_SERVER['PHP_SELF'];?>" method="post">
 Nombre del alumno: <input type="text" name="nombre" />

 <p>Módulos que cursa:</p>
 <input type="checkbox" name="modulos[]" value="DWES" />
 Desarrollo web en entorno servidor

 <input type="checkbox" name="modulos[]" value="DWECE" />
 Desarrollo web en entorno cliente

 <input type="submit" value="Enviar" name="enviar"/>
 </form>
 <?php
 }
?>
 </body>
</html>
```



# Anexo XI.- Ejemplo de validación

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML
4.01 Transitional//EN" "
http://www.w3.org/TR/html4/loose.dtd">
<!-- Desarrollo Web en Entorno Servidor -->
<!-- Tema 2 : Características del Lenguaje PHP --
>
<!-- Ejemplo: Validar datos en la misma página
que el formulario -->
<html>
 <head>
 <meta http-equiv="Content-Type"
content="text/html; charset=UTF-8">
 <title>Desarrollo Web</title>
 </head>
 <body>
<?php
 if (!empty($_POST['modulos']) && !
empty($_POST['nombre'])) {
 $nombre = $_POST['nombre'];
 $modulos = $_POST['modulos'];
 print "Nombre: ".$nombre."
";
 foreach ($modulos as $modulo) {
 print "Modulo: ".$modulo."
";
 }
 }
 else {
```

```
?>
 <form name="input" action="<?php echo $_SERVER['PHP_SELF'];?>"
method="post">
 Nombre del alumno:
 <input type="text" name="nombre" value="<?php if (isset
($_POST['nombre'])) echo $_POST['nombre'];?>" />
 <?php
 if (isset($_POST['enviar']) && empty($_POST['nombre']))
 echo " <!-- Debe introducir un
nombre!!"
 ?>

 <p>Módulos que cursa:
 <?php
 if (isset($_POST['enviar']) && empty($_POST['modulos']))
 echo " <!-- Debe escoger al menos
uno!!"
 ?>
 </p>
 <input type="checkbox" name="modulos[]" value="DWES"
 <?php
 if(isset($_POST['modulos']) && in_array("DWES",$_POST['modulos']))
 echo 'checked="checked"';
 ?>
 />
 Desarrollo web en entorno servidor

 <input type="checkbox" name="modulos[]" value="DWEC"
 <?php
 if(isset($_POST['modulos']) && in_array("DWEC",$_POST['modulos']))
 echo 'checked="checked"';
 ?>
 />
 Desarrollo web en entorno cliente

 <input type="submit" value="Enviar" name="enviar"/>
</form>
<?php
 }
?>
 </body>
</html>
```

# Anexo XII.- Solución propuesta VI

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "
http://www.w3.org/TR/html4/loose.dtd">
<!-- Desarrollo Web en Entorno Servidor -->
<!-- Tema 2 : Características del Lenguaje PHP -->
<!-- Ejemplo: Mostrar fecha completa a partir de día, mes y año introducidos -->
<html>
<head>
 <meta http-equiv="content-type" content="text/html; charset=UTF-8">
 <title>Fecha completa a partir de día, mes y año</title>
</head>
<body>
<?php
 date_default_timezone_set('Europe/Madrid');
 if (!empty($_POST['dia']) && !empty($_POST['mes']) && !empty($_POST['ano'])) {
 if (checkdate($_POST['mes'], $_POST['dia'], $_POST['ano']))
 {
 $fecha = mktime(0,0,0,$_POST['mes'], $_POST['dia'], $_POST['ano']);
 $numero_dia_semana = date("N", $fecha);
 switch($_POST['mes']) {
 case 1: $mes = "Enero";
 break;
 case 2: $mes = "Febrero";
 break;
 case 3: $mes = "Marzo";
 break;
 case 4: $mes = "Abril";
 break;
 case 5: $mes = "Mayo";
 break;
 case 6: $mes = "Junio";
 break;
 case 7: $mes = "Julio";
 break;
 case 8: $mes = "Agosto";
 break;
 case 9: $mes = "Septiembre";
 break;
 case 10: $mes = "Octubre";
 break;
 case 11: $mes = "Noviembre";
 break;
 case 12: $mes = "Diciembre";
 break;
```

```
 switch($numero_dia_semana) {
 case 1: $dia_semana = "Lunes";
 break;
 case 2: $dia_semana = "Martes";
 break;
 case 3: $dia_semana = "Miércoles";
 break;
 case 4: $dia_semana = "Jueves";
 break;
 case 5: $dia_semana = "Viernes";
 break;
 case 6: $dia_semana = "Sábado";
 break;
 case 7: $dia_semana = "Domingo";
 break;
 }
 print $dia_semana.", ".date("j", $fecha)." de ".$mes." de ".date("Y", $fecha);
 }
 else
 print "La fecha introducida no es correcta!!";
 }
?>
<form name="input" action="<?php echo $_SERVER['PHP_SELF'];?>" method="post">
 Dia:
 <input type="text" name="dia" value="<?php if (isset ($_POST['dia'])) echo
$_POST['dia'];?>" />
 <?php
 if (isset($_POST['enviar']) && empty($_POST['dia']))
 echo " <!-- Debe introducir un día!!"
?>

 Mes:
 <input type="text" name="mes" value="<?php if (isset ($_POST['mes'])) echo
$_POST['mes'];?>" />
 <?php
 if (isset($_POST['enviar']) && empty($_POST['mes']))
 echo " <!-- Debe introducir un mes!!"
?>

 Año:
 <input type="text" name="ano" value="<?php if (isset ($_POST['ano'])) echo
$_POST['ano'];?>" />
 <?php
 if (isset($_POST['enviar']) && empty($_POST['ano']))
 echo " <!-- Debe introducir un año!!"
?>

 <input type="submit" value="Enviar" name="enviar"/>
</form>
</body>
</html>
```