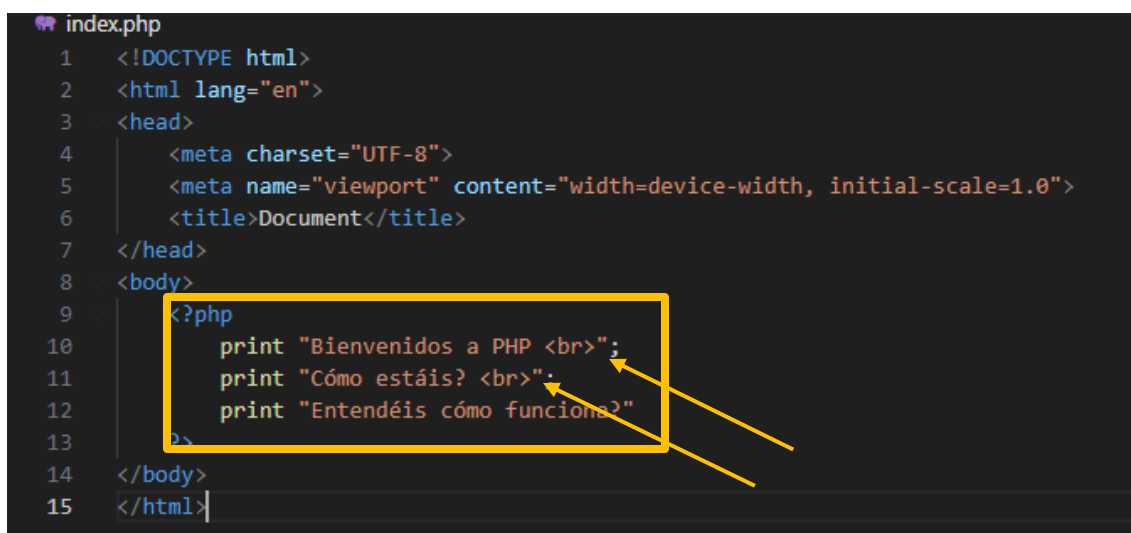


PHP

Una página PHP es una página que contiene fragmentos de código en PHP, es decir, entre dos etiquetas PHP. `<?php _____?>`

Todo el código comprendido entre las dos etiquetas será el código PHP. Debemos incluir estas instrucciones para que el servidor web sepa qué fragmento de código debe interpretar él y cuál será el código que interprete el navegador. En el siguiente código, estamos indicando al servidor que debe interpretar las líneas de código dentro del cuadrado. El servidor, por lo tanto, comprende que debe interpretar tres print, con un salto de línea en los dos primeros. El resto de código será interpretado por el navegador.



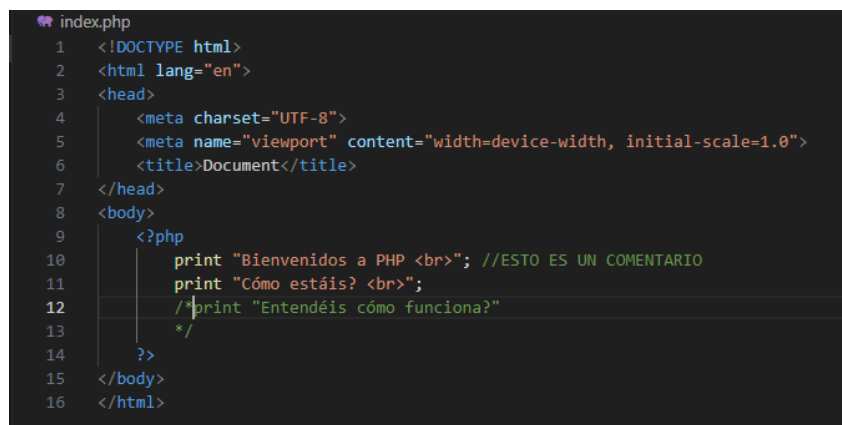
```
index.php
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Document</title>
7 </head>
8 <body>
9   <?php
10     print "Bienvenidos a PHP <br>";
11     print "Cómo estáis? <br>";
12     print "Entendéis cómo funciona?"
13   ?>
14 </body>
15 </html>
```

Los fragmentos de PHP no pueden ir en ningún caso anidados uno dentro de otro. Todo lo que no esté entre dos etiquetas de PHP en el código es HTML. Todas las instrucciones en PHP deben terminar con un ";" y las instrucciones van siempre entre llaves.

Comentarios

En los fragmentos de PHP podemos comentar líneas de código de las siguientes maneras:

- `//`: para comentar el resto de la línea
- `/* */`: para comentar todo lo que haya entre los dos signos.



```
index.php
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Document</title>
7 </head>
8 <body>
9   <?php
10     print "Bienvenidos a PHP <br>"; //ESTO ES UN COMENTARIO
11     print "Cómo estáis? <br>";
12     /*print "Entendéis cómo funciona?"
13   */
14   ?>
15 </body>
16 </html>
```

Variables

Es un espacio en la memoria del ordenador que puede cambiar a lo largo de la ejecución del programa. Para crear variables en PHP las debemos definir siempre comenzando por el signo \$, seguido del nombre de la variable. El nombre de las variables no puede llevar símbolos y, aunque pueden contener un número, no pueden comenzar por él.

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Document</title>
7 </head>
8 <body>
9   <?php
10     $nombre; //Esta es una variable válida
11     $nombre1; //Esta es una variable válida
12     $nombr?; //Esta no es una variable válida
13     $!nombre; //Esta no es una variable válida
14   ?>
15 </body>
16 </html>
```

Para inicializar una variable escribimos un = seguido del valor que le queramos dar, y si queremos imprimirla utilizamos la instrucción print.

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Document</title>
7 </head>
8 <body>
9   <?php
10     $nombre="Dani";
11     print $nombre;
12   ?>
13 </body>
14 </html>
```

Si queremos concatenar una variable dentro de una cadena de caracteres, podemos introducir la variable dentro de la propia cadena.

```
index.php
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Document</title>
7 </head>
8 <body>
9   <?php
10     $nombre="Dani";
11     $edad=26;
12     print "EL nombre del usuario es $nombre y su edad es $edad"
13   ?>
14 </body>
15 </html>
```

Print y Echo

Ambas instrucciones sirven para mostrar texto en pantalla. La gran diferencia reside en que echo nos permite imprimir varias variables seguidas sin necesidad de concatenar.

```
index.php
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Document</title>
7  </head>
8  <body>
9      <?php
10         $nombre="Dani";
11         $edad=26;
12         echo $nombre,$edad;
13     ?>
14 </body>
15 </html>
```

Además, print consume más recursos que echo, por lo que la mayoría de las veces, en programas complejos con miles de valores encontraremos más a menudo la instrucción echo.

Printf

Tercera opción para generar una salida de código HTML. Recibe varios parámetros, el primero siempre es una cadena de texto que indica el formato que se ha de aplicar. Además, la cadena debe llevar un especificador de conversión de parámetros.

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

Todos los especificadores van precedidos de "%", y se compone de:

- **Signo:** Indica si pone signo a los números negativos (lo pone por defecto) y también a los positivos con un +.
- **Relleno:** Indica qué carácter se usará para ajustar el tamaño de una cadena. Puede ser el 0 o espacios.
- **Alineación:** tipo de alineación se usará para generar la salida: justificación derecha (por defecto) o izquierda (se indica con el carácter -).
- **Ancho:** mínimo número de caracteres de salida para un parámetro dado.
- **Precisión:** número de dígitos decimales que se mostrarán para un número real. Se escribe como un dígito precedido por un punto.
- **Tipo:** cómo se debe tratar el valor del parámetro correspondiente. En la siguiente tabla puedes ver una lista con todos los especificadores de tipo.

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.
F	el argumento es tratado como un doble y presentado como un número de coma flotante.
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..

```
<?php
    printf("El número PI vale %+.2f", 3.1416);
?>
```

Existe también la función **sprintf**, que permite guardar en una variable la salida obtenida.

```
<?php
    $texto=sprintf("El número PI vale %+.2f", 3.1416);
    echo $texto;
?>
```

Tipos de variables

Variables lógicas (boolean)

Sólo pueden obtener el valor true o el valor false. Muy útiles en estructuras de control (if).

```

index.php
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Document</title>
7  </head>
8  <body>
9      <?php
10         $dentro = true;
11         if($dentro == true){
12             echo "Está dentro";
13         }
14         if($dentro != false){
15             echo "está fuera";
16         }
17     ?>
18 </body>
19 </html>

```

Variables enteras

Guardan números enteros (positivo y negativo). Tampoco están tipadas.

```

index.php
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Document</title>
7  </head>
8  <body>
9      <?php
10         $edad=23;
11         $precio=50;
12     ?>
13 </body>
14 </html>

```

Variables decimales (float)

Funcionan igual que las variables enteras, guardan números decimales positivos y negativos. Lo que diferencia la parte entera de la parte decimal es un punto, no una coma.

```

index.php
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <?php
    $pi=3.14;
    echo $pi;
  ?>
</body>
</html>

```

Ejercicio: Define dos variables enteras o decimales, calcula el área de un rectángulo y triángulo e imprime ambos por pantalla.

Variables de cadenas (string)

Guardan caracteres. En PHP no tenemos límite de caracteres que guardar, el límite lo pone la memoria que sea capaz de soportar el servidor. En un string podemos acceder a uno de los caracteres de la cadena indicando la posición del mismo, como si hablásemos de un array de una sola fila. El primer carácter ocupa la posición 0.

```

index.php
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Document</title>
7  </head>
8  <body>
9      <?php
10         $cadena="Buenas a todos <br> ";
11         print $cadena;
12         $cadena[0]="F";
13         print $cadena;
14     ?>
15 </body>
16 </html>

```

Si la posición que indicamos es mayor que la longitud de la cadena, completa con espacios hasta llegar a dicha posición, y si introducimos en una posición ya existente una cadena vacía, acorta la cadena y elimina dicha posición.

Podemos usar tanto comillas simples como comillas dobles. Sin embargo, existe diferencia entre ellas. Si ponemos una variable dentro de comillas dobles, PHP la procesa y la sustituye por su valor.

```

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

```

La variable módulo se ha “reconocido” dentro de las comillas dobles, y sustituye su valor por “DWES” antes de salir a pantalla. Con comillas simples esto no hubiera sucedido. A veces PHP necesita que pongamos la variable entre llaves para distinguir el texto de la variable.

```

<?php
    $modulo="DWES";
    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.

Secuencia	Resultado
<code>\\</code>	muestra una barra invertida.
<code>\'</code>	muestra una comilla simple.
<code>\"</code>	muestra una comilla doble.
<code>\n</code>	muestra un avance de línea (LF o 0x0A (10) en ASCII).
<code>\r</code>	muestra un retorno de carro (CR o 0x0D (13) en ASCII).
<code>\t</code>	muestra un tabulador horizontal (HT o 0x09 (9) en ASCII).
<code>\v</code>	muestra un tabulador vertical (VT o 0x0B (11) en ASCII).
<code>\f</code>	muestra un avance de página (FF o 0x0C (12) en ASCII).
<code>\\$</code>	muestra un signo de dólar.

Además, podemos crear cadenas mediante la sentencia **heredoc**. Consiste en poner el operador `<<<` seguido de un identificador de tu elección, y a continuación y empezando en la línea siguiente la cadena de texto, sin utilizar comillas. La cadena finaliza cuando escribes ese mismo identificador en una nueva línea. Esta línea de cierre no debe llevar más caracteres, ni siquiera espacios o sangría, salvo quizás un punto y coma después del identificador.

```
<?php
$a = <<<MICADENA
Desarrollo de Aplicaciones Web<br />
Desarrollo Web en Entorno Servidor
MICADENA;
print $a;
?>
```

El texto se procesa de igual forma que si fuera una cadena entre comillas dobles, sustituyendo variables y secuencias de escape. Si no quisieras que se realizara ninguna sustitución, debes poner el identificador de apertura entre comillas simples.

Conversión y comprobación de tipos

Si realizamos operaciones con variables de distintos tipos, primero se convierten a un tipo común. Si sumamos un entero a un real, el entero se convierte a real antes de realizar la suma. También podemos forzar dicho cambio de tipo.

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <title>Document</title>
7 </head>
8 <body>
9     <?php
10         $entero=4;
11         $real=2.3;
12         $resultado=$entero+$real;
13         echo $resultado;
14         print "<br>";
15
16         $resultado=$entero+(int)$real;
17         echo $resultado;
18
19     ?>
20 </body>
21 </html>
```

Podemos forzar los siguientes tipos:

- (int), (integer) - forzado a integer
- (bool), (boolean) - forzado a boolean
- (float), (double), (real) - forzado a float
- (string) - forzado a string
- (array) - forzado a array
- (object) - forzado a object

Expresiones y operadores

Al igual que en el resto de lenguajes, en PHP también se utilizan expresiones para llevar a cabo ciertas acciones en un programa. Al igual que en otros lenguajes, tenemos operadores para diversos motivos.

- **Operaciones aritméticas:** suma (+), resta (-), negación (!), multiplicación (*), división (/) y módulo (%). También los operadores de incremento (++) y decremento (--).

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Document</title>
7  </head>
8  <body>
9      <?php
10         $a=4;
11         $b=++$a;
12         echo "El valor de A es $a, y el de B es $b";
13         print "<br>";
14
15         $a=4;
16         $b=$a++;
17         echo "El valor de A es $a, y el de B es $b";
18     ?>
19 </body>
20 </html>
```

- **Realizar asignaciones:** Operador = y operadores += y -=, así como el operador .=

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Document</title>
7  </head>
8  <body>
9      <?php
10         $a=4;
11         $a+=3;
12         $b="El valor";
13         $b.=" de A es ";
14         echo $b, $a;
15     ?>
16 </body>
17 </html>
```

Implementa un programa que realice los siguientes pasos: declara tres variables \$a, \$b y \$c, asignándoles los valores 10, "Hola", y 4. Asigna a la variable \$a el valor de la variable \$c. Utiliza el operador += para sumar a \$c 5 e imprime el resultado. Utiliza el operador *= para

multiplicarlo por 8 e imprime el resultado. Utiliza el operador `.=` para agregar a `$b` el valor " a todos" e imprime el resultado

- **Comparar operandos:** En PHP tenemos los operandos comparativos típicos del resto de lenguajes (`<`, `>`, `<=`, `>=`, `!=`), pero además nos encontramos con `"=="`, `"==="`. El operador `(==)` devuelve `"true"` si los operandos son del mismo tipo y `(!==)` devuelve `"true"` si los dos operadores tienen distinto tipo.

Nombre	Ejemplo	Resultado
Igual	<code>\$a == \$b</code>	Si ambos son iguales el resultado devuelve verdadero
Idéntico	<code>\$a === \$b</code>	Si ambos son iguales y son del mismo tipo devuelve verdadero
Diferente	<code>\$a != \$b</code>	Si ambos son diferentes el resultado devuelve verdadero
No idéntico	<code>\$a !== \$b</code>	Si ambos son diferentes o son de diferentes tipos devuelve verdadero
Mayor	<code>\$a > \$b</code>	Si el primero es mayor que el segundo devuelve verdadero
Menor	<code>\$a < \$b</code>	Si el primero es menor que el segundo devuelve verdadero
Mayor o igual	<code>\$a >= \$b</code>	Si el primero es mayor o igual que el segundo devuelve verdadero
Menor o igual	<code>\$a <= \$b</code>	Si el primero es menor o igual que el segundo devuelve verdadero

```

<body>
  <?php
    $a=1;
    $b="1";
    $c=true;
    $d=3-1;
    $res;

    if($a!==$b) {
        echo "verdadero";
    }
    else {
        echo "falso";
    }
  ?>
</body>
</html>

```

Ejercicio. Implementa un programa que declare cuatro variables `a`, `b`, `c` y `d` con valores `3.7`, `false`, `1` y `"3.7"` respectivamente. Realiza las siguientes operaciones y comprueba los resultados: `$a==$c`, `$a=== $d`, `$b<=$c`.

- **Comparar expresiones booleanas:** trabajamos con operadores lógicos (`&&`, `||`, `!`)
- **Operador de control de errores:** PHP soporta un operador de control de errores: el signo de arroba (`@`). Cuando se antepone a una expresión en PHP, cualquier mensaje de error que pueden ser generado por esa expresión será ignorado. Este símbolo se puede anteponer a variables, funciones, includes y constantes. si se usa `"@"` para eliminar los errores de una cierta función y ésta no se encuentra disponible o ha sido escrita de forma incorrecta, el script se detendrá en ese punto sin indicación de por

qué. Por ello es interesante utilizarlo con precaución y teniendo en cuenta la deshabilitación de los errores.

```
<body>
  <?php
    $a=1;
    $b=0;
    $resultado = $a/$b;
    echo $resultado;
  ?>
</body>
</html>
```

Ámbito de las variables

Al igual que en Java, podemos utilizar variables en cualquier parte del código. Si la variable aún no existe, se reserva memoria para ella. Si la variable aparece por primera vez en la función, llamamos a la variable local. Si aparece otra vez la misma variable fuera de la función, la variable es distinta.

```
<?php
$a = 1;
function prueba()
{
    // Dentro de la función no se tiene acceso a la variable $a anterior
    $b = $a;
    // Por tanto, la variable $a usada en la asignación anterior es una variable nueva
    // que no tiene valor asignado (su valor es null)
}
?>
```

Si en la función anterior quisieras utilizar la variable \$a externa, podrías hacerlo utilizando la palabra global. De esta forma le dices a PHP que no cree una nueva variable local, sino que utilice la ya existente.

```
<?php
$a = 1;
function prueba()
{
    global $a;
    $b = $a;
    // En este caso se le asigna a $b el valor 1
}
?>
```

Cuando las funciones terminan, las variables pierden su valor y desaparecen. Si queremos mantener el valor de la variable en distintas llamadas a la función utilizando la palabra static. Las variables estáticas deben inicializarse en la misma sentencia en que se declaran como estáticas. De esta forma, se inician sólo la primera vez que se llama a la función.

```

<?php
function contador()
{
    static $a=0;
    $a++;
    // Cada vez que se ejecuta la función, se incrementa el valor de $a
}
?>

```

Constantes

En ocasiones necesitamos la utilización de un mismo valor en diferentes lugares del programa. Una de las opciones pasa por definir el valor como una variable y usarlo, otra pasa por utilizar la constante, tal cual, en cada uno de los lugares en los que se requiere, y la última consiste en definir una constante para poder utilizarla posteriormente. Una constante es un elemento similar a una variable, pero con la peculiaridad de que su valor no se modifica.

El formato de definición de una constante es un poco diferente al de una variable es el siguiente:

```

<?php
/*define("nombre_constante",valor);

const nombre_constante = valor;*/

define("constantePi", 3.1416);
const constanteFi = 1.6180;

echo constantePi;
print "<br>";
echo constanteFi;
?>

```

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

Funciones relacionadas con los tipos de datos

El lenguaje nos ofrece funciones especiales que nos permiten saber el tipo de variable que estamos usando y también realizar su conversión.

- **gettype()**: Devuelve el tipo de una variable.
- **settype(\$variable, 'tipo variable')**: Transforma el tipo de variable del que haya decidido PHP por el que nosotros decidamos
- **is_array()**: Comprueba si la variable es de tipo array
- **is_bool()**: Comprueba si la variable es de tipo boolean
- **is_float()**: Comprueba si la variable es de tipo double
- **is_integer()**: Comprueba si la variable es de tipo integer
- **is_null()**: Comprueba si la variable es nula
- **is_numeric()**: Comprueba si la variable es numérica o string numérico

- **is_object():** Comprueba si la variable es de tipo objeto
- **is_resource():** Comprueba si la variable es un recurso
- **is_scalar():** Comprueba si la variable es escalar (integer, double, string o boolean)
- **is_string():** Comprueba si la variable es de tipo string

```
<?php
$a = $b = "3.1416";
settype($b, "float");
print "\$a vale $a y es de tipo ".gettype($a);
print "<br />";
print "\$b vale $b y es de tipo ".gettype($b);
?>
```

También existen dos funciones que trabajan con la definición de las variables. La función **isset(\$variable)** nos permite conocer si una variable está definida, y la función **unset(\$variable)** para borrar la variable.

```
<?php
$a = "variable";
if (isset($a))
unset($a);
echo $a;
?>
```

No debemos confundir que una variable esté definida o valga null con que se considere vacía. Para lo último tenemos la función **empty()**, con sintaxis: **bool empty(\$variable)**. Esta función tan solo comprueba si la variable está vacía, no hace más. Es el equivalente a **isset(\$var) || \$var == false**; Devuelve FALSE si la variable existe y tiene un valor no vacío, distinto de cero. De otro modo devuelve TRUE

```
<?php
$a = "";
empty($a);
echo $a;
?>
```

```
$a = "";
if (empty($a)) echo "vacía";
```

Las expresiones que se consideran vacías son las siguientes:

- "" (una cadena vacía)
- 0 (0 como un integer)
- 0 (0 como un float)
- "0" (0 como un string)
- NULL
- FALSE

- `array()` (un array vacío)
- `$var;` (una variable declarada, pero sin un valor)

A menudo vamos a tener que trabajar con fechas y horas. La información relativa a la fecha y la hora se guarda como un número entero. Para trabajar de forma más cómoda tenemos la función **date()**

string date (string \$formato [, int \$fecha hora]);

Recibe dos parámetros: la descripción del formato y el entero que identifica la fecha. A cambio, devuelve una cadena formateada. Los caracteres de formato que podemos utilizar son los siguientes:

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 = 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 la 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	formato de 12 horas, siempre con dos dígitos.
H	formato de 24 horas, siempre con dos dígitos.
g	formato de 12 horas, con uno o dos dígitos (sin ceros iniciales).
G	formato de 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.

Para establecer la hora y la fecha correctas podemos usar la función **date_default_timezone_set()**.

```
<?php
    date_default_timezone_set('Europe/Madrid');
    echo date("l d H");
?>
```

Otra función útil que podemos utilizar es **getdate()** que devuelve un array con la fecha y la hora actuales.

```
<?php
    date_default_timezone_set('Europe/Madrid');
    $fecha=getdate();
    print_r($fecha);
?>
```

Variables especiales

PHP incluye variables reservadas, ya predefinidas, que pueden usarse desde cualquier lugar del código. Son llamadas variables superglobales, y no necesitamos utilizar la palabra “global” para acceder a ellas. Todas ellas son arrays con un conjunto de valores. Las principales son las siguientes:

- **\$_SERVER:** incluye información sobre el servidor web y el de ejecución. Esta variable a su vez nos ofrece la siguiente información:

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

```
<?php
    echo $_SERVER['SERVER_NAME'];
    print"<br>";
    echo $_SERVER['DOCUMENT_ROOT'];
    print"<br>";
?>
```

- **\$_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.

Estructuras de Control

En PHP, como en el resto de lenguajes, tenemos dos tipos de estructuras de control: las sentencias condicionales y las sentencias de bucle. Además, en PHP también tenemos la sentencia **goto**, que permite saltar de un punto a otro del código.

```
<?php
$a=1;
goto salto;
$a++;
salto:
echo $a;
?>
```

Condicionales

IF

Ejecuta la sentencia si esta es evaluada como true. Si, por el contrario, la sentencia es evaluada como false, no se ejecutará. Si sucede la segunda opción, podemos ejecutar una serie de sentencias en dicho caso mediante **else**, o añadir una nueva condición mediante **else if**.

SWITCH

Similar a enlazar varias órdenes if comparando una variable con diferentes valores. Cada valor que comparamos 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 encuentra un **break**. Si no encuentra ninguna coincidencia se ejecutan las sentencias del bloque default si este existe.

```
<?php
$a=3;
switch ($a) {
    case 0:
        print "a vale 0";
        break;
    case 1:
        print "a vale 1";
        break;
    default:
        print "a no vale 0 ni 1";
}
?>
```

Ejercicio 1. Haz una página web que muestre la fecha actual en castellano, incluyendo el día de la semana, con un formato similar al siguiente: "Jueves, 1 de Mayo de 1889".

Ejercicio 2.

Una agencia espacial nos ha contratado para realizar un programa. Éste debe contener 3 variables: una para almacenar la velocidad, otra para la aceleración y otra para la altura de las naves espaciales. Además, quiere 3 constantes como son la constante de gravitación

universal ($6.674E-11$), la de la aceleración de la gravedad (9.8) y la velocidad máxima (300). Los valores de las variables podemos elegirlos a nuestro gusto, ya que son para realizar pruebas antes del lanzamiento. El programa deberá comprobar que la velocidad sea menor a la velocidad máxima y que la aceleración sea mayor a la aceleración de la gravedad y, en caso afirmativo, mostrará un mensaje indicando que todos los parámetros son correctos. En caso contrario comprobará que la velocidad sea mayor a la velocidad máxima y, en ese caso, deberá mostrar un mensaje indicando que la velocidad es excesiva, en contrario deberá comprobar que si la aceleración es menor a la de la aceleración de la gravedad y si es así mostrará un mensaje indicando que la aceleración es insuficiente.

La agencia espacial quiere que amplíemos el programa. Para ello deberás añadir una comprobación, utilizando un switch, para que, en función de la aceleración muestre un mensaje u otro:

- De 0 a 2: indicará que hay que aumentar la velocidad.
- 3: indicará que le falta algo de aceleración.
- De 4 a 7: indicará que aceleración es perfecta.
- De 7 a 9: indicará que hay que decelerar por peligro de destrucción.
- En cualquier otro caso: indicará que la cifra no está contemplada.

Bucles

While

Las instrucciones se ejecutan mientras se cumple una expresión. Dicha expresión se evalúa cada vez que comienza cada ejecución del bucle.

```
<?php
$a = 1;
while ($a < 8)
$a += 3;
print $a;
?>
```

Do...While

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;
?>
```

For

El bucle más complejo de todos. Funcionan como en el resto de lenguajes. Tienen 3 sentencias. La primera expresión se ejecuta una vez al comienzo, la segunda expresión evalúa si deben ejecutarse el conjunto de sentencias. Si el resultado de esta segunda expresión es false, termina el for. En caso de ser true se ejecutan las sentencias y al terminar ejecuta la tercera expresión, antes de volver a evaluar la segunda expresión para continuar adelante o no.

```
<?php
for ($a = 5; $a<10; $a+=3) {
    print $a;
    print "<br />";
}
?>
```

Podemos anidar cualquiera de los bucles anteriores en varios niveles. En algunos casos podemos utilizar break para salir inmediatamente del bucle y continue para volver inmediatamente a la comprobación de la expresión.

Ejercicio 3. Una asociación de matemáticas nos ha encargado un programa que muestre los primeros 50 números pares. Luego, la asociación matemática quiere que, además, mostremos por pantalla los 50 primeros números impares.

Ejercicio 4. Crear un programa que escriba dos columnas de números, en la primera se colocan los números del 1 al 100, en la segunda los números del 100 al 1.

1 100

2 99

3 98

.....

Ejercicio 5. Realiza la división entera de dos números distintos de 0 utilizando únicamente restas.

Ejercicio 6. Crear el programa que escriba la tabla de multiplicar de todos los números del 1 al 15.

Ejercicio 7. Crear el programa que a partir de un número entero cree una pirámide de arrobas.

Funciones

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. En PHP nos encontramos con numerosas funciones propias que se pueden utilizar directamente.

Inclusión de ficheros externos

A medida que vayamos haciendo programas será bastante tedioso buscar ciertas partes de código a lo largo de todo el programa. Es por ello que es bastante útil agrupar ciertos bloques o funciones de código y tenerlos disponible en otro fichero aparte. En caso de necesitar esos

ficheros podemos hacer una llamada a los mismos e incluirlos como parte de la ejecución del código principal. Para llevar a cabo esto, tenemos varias opciones:

- **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.

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

```
<?php
    print "Módulo $modulo del ciclo $ciclo<br />";
    include 'archivo.php';
    print "Módulo $modulo del ciclo $ciclo<br />";
?>
```

Es necesario delimitar el archivo externo con las etiquetas básicas de php puesto que en el momento en que llamamos a un archivo externo trata el contenido nuevo como etiquetas html.

- **Include_once:** Si por equivocación incluyes más de una vez un mismo fichero, lo normal es que obtengas algún tipo de error. `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, incluye 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:** 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.

En muchas ocasiones al programar utilizamos la extensión `.inc.php` para los ficheros en PHP cuyo objetivo es ser incluidos dentro de otros y nunca ser ejecutados por sí solos.

Ejecución y creación de funciones

Para hacer una llamada a una función, basta con poner su nombre y unos paréntesis.

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

Para crear funciones utilizamos la palabra function.

```
<?php
    function saludo(){
        echo "Cómo estáis?";
    }
?>
/body>
```

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. PHP no diferencia entre mayúsculas y minúsculas a la hora de nombrar una función

Argumentos

En los ejemplos anteriores usamos una variable global. Sin embargo, no es una práctica adecuada. Lo ideal es utilizar argumentos al hacer llamada. Las funciones devuelven un resultado o un valor usando la expresión **return**. 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).

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. 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.

```
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
?>
```

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 argumentos 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.

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

Ejercicio 8. Crea una función para calcular el cuadrado de un número, ha de mostrar por pantalla el número y su cuadrado correspondiente.

Ejercicio 9. Una función que reciba como parámetros el valor del radio de la base y la altura de un cilindro y devuelva el volumen del cilindro, teniendo en cuenta que el volumen de un cilindro se calcula como $\text{Volumen} = \text{númeroPi} * \text{radio} * \text{radio} * \text{Altura}$ siendo númeroPi = 3.1416 aproximadamente.

Ejercicio 10. Realiza una función que calcula el IVA y que recibe dos parámetros. Uno el valor sobre el que se calcula y el otro el porcentaje a aplicar. Si no se indica el porcentaje de IVA se entiende que es el 18% (valor por defecto). Una vez realizada la función ejecútala con los siguientes valores: 1000, 1000 con el 8%, 10 con el 0% y muestra los respectivos resultados.

Ejercicio 11. 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.

Tipos de datos compuestos

Matrices (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.

Las definimos con corchetes, separando con una coma cada elemento. Para imprimir un solo elemento de la matriz utilizamos una referencia al índice entre corchetes. En una misma matriz podemos almacenar distintos tipos de datos.

```
index.php
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Document</title>
7  </head>
8  <body>
9      <?php
10         $matriz = ["esto", "es", "una", "matriz"];
11         print "<p>$matriz[1]</p>\n";
12         print_r($matriz);
13         print "<br>";
14         var_dump ($matriz);
15     >
16 </body>
17 </html>
```

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.

Otra manera de crear arrays:

```
<?php
// array numérico
$modulos1 = array(0 => "Programación", 1 => "Bases de datos", 2 => "Desarrollo web en entorno servidor");
// array asociativo
$modulos2 = array("PR" => "Programación", "BD" => "Bases de datos", "DWES" => "Desarrollo web en entorno servidor");
?>
```

Si queremos mostrar por pantalla un vector lo haremos recorriéndolo mediante un bucle.

```
<?php
$vector=[1,2,3,4];
for ($i=0; $i < 4 ; $i++) {
    echo $vector[$i]."<br>";
}
?>
```