Elementos básicos del lenguaje

Como hemos visto en el documento anterior, la sintaxis básica para escribir código PHP consiste en un bloque como el siguiente:

```
<?php
    ... código PHP ...
?>
```

Dentro de los símbolos `<?php` (de apertura) y `?>` (de cierre) pondremos las instrucciones que consideremos. Todo esto puede ocupar una sola línea (para instrucciones simples y cortas) o dividir el código en varias líneas, una por instrucción.

1. Sintaxis básica y comentarios

Dentro de cada bloque de código PHP que escribamos en una página, debemos tener en cuenta que:

- Cada instrucción en PHP termina siempre en un punto y coma (`;`) que la separa de la siguiente, aunque estén en líneas separadas.
- Los comentarios en PHP se pueden poner de diversas formas.

```
/* Comentario que puede ocupar varias líneas */
// Comentario de una línea
# Otro ejemplo de comentario de una línea
```

Veamos a continuación un ejemplo sencillo de código PHP, embebido dentro de una página HTML. En este caso, creamos unas variables en PHP que almacenan un nombre y un año, y luego mostramos esas variables entre el contenido HTML propiamente dicho:

```
<!DOCTYPE html>
<html lang="es">
<head>
    ...
</head>
<body>
    <h1>Página de prueba en PHP</h1>
    <?php
        // Variables para almacenar el nombre y el año actual
        $nombre = "Nacho Iborra";
        $anyo = 2014;
    ?>
    <p>El autor de esta página es <?php echo $nombre; ?> y está
    realizada en el año <?php echo $anyo; ?>.</p>
...
```

Observa cómo podemos incluir código PHP en cualquier zona de la página. En la primera, hemos definido dos variables, `$nombre` y `$anyo`, y después, con instrucciones cortas, las hemos mostrado en otras zonas de la página (instrucción `echo`).

Lo que hace el servidor cuando se le solicita esta página es procesarla, detectar el código PHP, ejecutarlo y sustituirlo en la página por el código HTML que éste genera (en el ejemplo anterior, mostrar el valor de las variables en los lugares correspondientes).

Opcionalmente, en el caso de que sólo tengamos instrucciones `echo`, podemos sustituir la estructura `<?php ... ?>` por la estructura `<?= ... ?>` y ahorrarnos la instrucción *echo*. Así, el párrafo que se muestra en el ejemplo anterior lo podríamos poner así:

```
<p>El autor de esta página es <?= $nombre; ?> y está
realizada en el año <?= $anyo; ?>.</p>
```

También podemos utilizar indistintamente la instrucción `print` en lugar de `echo` para mostrar información por pantalla.

```
<p>El autor de esta página es <?php print $nombre; ?> y está
realizada en el año <?php print $anyo; ?>.</p>
```

> **Ejercicio 1:**
>
> Crea una carpeta llamada **ejercicios1** en tu carpeta de documentos de Apache. En esta carpeta guardarás este ejercicio y los siguientes, ya que serán muchos y así evitamos llenar la carpeta de documentos de demasiadas subcarpetas con ejercicios cortos.

> Para este ejercicio, crea un documento en esta carpeta llamado **info_basica.php**, similar al del ejemplo anterior, pero mostrando tu nombre y tu año de nacimiento usando variables. Es decir, crearás dos variables para almacenar estos dos datos, y los mostrarás en una frase que diga "Me llamo XXXX y nací en el año YYYY". Prueba la página en un navegador y echa un vistazo al código fuente, intentando detectar qué contenidos HTML se han generado desde PHP.

2. Variables y tipos de datos

Como en todo lenguaje de programación, las variables en PHP nos van a servir para almacenar información, de manera que, además de tenerla disponible, podemos modificarla o cambiarla por otra durante el tiempo de ejecución de la aplicación web.

Como hemos visto en el ejemplo anterior, las variables en PHP se definen mediante el símbolo del dólar (`$`) seguido de una serie de letras, números o caracteres de subrayado, aunque no pueden empezar por número. Ejemplos de nombres de variables válidos son: `$nombre`, `$primer_apellido`, `$apellido2`... En las variables, se distinguen mayúsculas de minúsculas, y no hace falta declararlas (es decir, no se indica de qué tipo son, como en lenguajes como C o Java, ni se les reserva memoria de antemano).

Veamos algunos ejemplos de uso de variables:

```
<?php
    $edad = 36;
    $nombre = "Nacho";
    ...
    echo $nombre;
    echo $edad;
?>
```

2.1. Comprobar el estado de las variables

Es posible que, en algún momento de la ejecución del programa, una variable no tenga un valor definido, o queramos eliminar el valor que tiene. Para ello tenemos algunas instrucciones útiles:

- `unset($variable)` permite borrar la variable (como si no la hubiéramos creado)
- `isset($variable)` permite comprobar si una variable existe
- `empty($variable)` permite comprobar si una variable está vacía, es decir, no tiene un valor concreto asignado.

```php
<?php
    $dato="Hola";
    unset($dato);       // La variable dato deja de existir
    $dato = "";
    echo empty($dato);  // Diría que es cierto, porque $dato está vacía
?>
```

2.2. Tipos de datos

Las variables almacenan datos, y esos datos son de un tipo concreto. Por ejemplo, pueden ser números, o textos. En concreto, PHP soporta los siguientes tipos de datos básicos:

- **Booleanos**: datos que sólo pueden valer verdadero o falso (en PHP se representan con las palabras TRUE y FALSE, respectivamente, en mayúsculas).
- **Enteros**: números sin decimales. Los podemos representar en formato decimal (el normal, por ejemplo 79), formato octal (poniendo un 0 delante, por ejemplo 0233) o hexadecimal (poniendo 0x delante, por ejemplo 0x1A3).
- **Reales**: números con decimales. La parte decimal queda separada de la entera con un punto. Por ejemplo, 3.14. También podemos usar notación científica, como por ejemplo 1.3e3, que equivale a $1.3 \cdot 10^3$.
- **Textos**: se pueden representar entre comillas simples o dobles. Si los representamos con comillas dobles, podemos intercalar variables dentro.

Veamos algunos ejemplos de cada tipo:

```php
<?php
    $edad = 20;
    $esMayorEdad = TRUE;
    $nota = 9.5;
    $nombre = "Juan Pérez";

    echo "El alumno $nombre tiene una nota de $nota";
?>
```

Además, en PHP podemos manejar otros tipos de datos algo más complejos, que son:

- **Array**: conjuntos de datos
- **Object**: para programación orientada a objetos, almacena instancias de clases
- **Resource**: para recursos externos, como una conexión a una base de datos
- **null**: es un valor especial para darle a variables que, en un momento dado, no tengan un valor concreto. Así, quedan vacías, sin valor.

2.2.1. Conversiones entre tipos de datos

Si tenemos un dato de un tipo y lo queremos convertir a otro, se tienen una serie de funciones que nos permiten hacer estas conversiones.

- **intval** sirve para convertir un valor (en formato cadena, o real) a entero.
- **doubleval** sirve para convertir un valor a número real.
- **strval** sirve para convertir un valor a una cadena de texto.

Es habitual usarlas cuando le pedimos datos al usuario en un formulario. Si por ejemplo le pedimos que escriba su edad en un cuadro de texto, este valor se envía como tipo cadena de texto, no como un número, y luego tendríamos que convertirlo a número entero. Realmente, estas conversiones son automáticas con las últimas versiones de PHP, pero conviene saber que estas funciones existen para poderlas utilizar si es el caso. Veamos otro ejemplo más sencillo, con una variable cadena de texto que convertimos al entero correspondiente:

```
<?php
    $textoEdad='21';
    $edad = intval($textoEdad);
?>
```

2.3. Constantes

Hemos visto que las variables son datos cuyo valor puede cambiar a lo largo de la ejecución de la aplicación. A veces nos puede interesar almacenar otros datos que no queramos que cambien a lo largo del programa. Por ejemplo, si almacenamos el número *pi*, ese valor siempre va a ser el mismo, no lo vamos a cambiar.

Para definir constantes en PHP se utiliza la función `define`, y entre paréntesis pondremos el nombre que le damos a la constante (entre comillas), y el valor que va a tener, separados ambos datos por coma. Después, para utilizar la constante más adelante, usamos el nombre que le hemos dado, pero sin las comillas. Veamos este ejemplo que calcula la longitud de una circunferencia:

```
<?php
    define('PI', 3.1416);
    $radio = 5;
    $longitud = 2 * PI * $radio;

    echo "La longitud de la circunferencia es $longitud";
?>
```

Aunque no es obligatorio, sí es bastante convencional que las constantes tengan todo su nombre en mayúsculas, para distinguirlas a simple vista de las variables (aunque, además, las variables en PHP empiezan por un dólar, y las constantes no).

> **Ejercicio 2:**

> Crea una página en la carpeta de ejercicios llamada **area_circulo.php**. En ella, crea una variable `$radio` y ponle el valor 3.5. Según esa variable, calcula en otra variable el área del círculo (PI * radio^2, deberás definir la constante PI), y muestra por pantalla el texto "El área del círculo es XX.XX", donde XX.XX será el resultado de calcular el área.

2.4. Operaciones

Podemos realizar distintos tipos de operaciones con los datos que manejamos en PHP: aritméticas, comparaciones, asignaciones, etc. Veremos los operadores que podemos utilizar en cada caso.

Operaciones aritméticas

Son las operaciones matemáticas básicas (sumar, restar, multiplicar...). Los operadores para llevarlas a cabo son:

Operador	Significado
`+`	Suma
`-`	Resta
`*`	Multiplicación
`/`	División
`%`	Resto de división
`.`	Concatenación (para textos)
`++`	Autoincremento
`--`	Autodecremento

El operador . sirve para concatenar o enlazar textos, de forma que podemos unir varios en una variable o al sacarlos por pantalla:

```
$edad = 36;
$nombre = "Nacho";
$texto = "Hola, me llamo " . $nombre . " y tengo " . $edad . " años.";
// En este punto, $texto vale "Hola, me llamo Nacho y tengo 36 años."
$edad++;
echo "El usuario se llama " . $nombre;
```

Operaciones de asignación

Permiten asignar a una variable un cierto valor. Se tienen los siguientes operadores:

Operador	Significado
`=`	Asignación simple
`+=`	Autosuma
`-=`	Autoresta
`*=`	Automultiplicación
`/=`	Autodivisión
`%=`	Autoresto
`.=`	Autoconcatenación

La asignación simple ya la hemos visto en ejemplos previos, y sirve simplemente para darle un valor a una variable.

```
$dato = 3;
```

Los operadores de Auto... afectan a la propia variable, sumándole/restándole/... etc. un valor. Por ejemplo, si hacemos algo como:

```
$dato *= 5;     // Dato = 15
```

Operaciones de comparación

Estas operaciones permiten comparar valores entre sí, para ver cuál es mayor, o si son iguales, entre otras cosas. En concreto, los operadores disponibles son:

Operador	Significado
`==`	Igual que
`===`	Idéntico a (igual y del mismo tipo)
`!=, <>`	Distinto de
`!==`	No idéntico
`<`	Menor que
`<=`	Menor o igual que
`>=`	Mayor o igual que

Lo que se obtiene con estas comparaciones es un valor booleano (es decir, verdadero o falso). Por ejemplo, 5 != 3 sería verdadero, y 4 <= 1 sería falso.

Además, en versiones recientes de PHP se han añadido algunos operadores adicionales, como el operador *nave espacial* y el operador de comprobación de nulos.

- El operador de nave espacial `<=>` compara dos datos y devuelve -1 si el primero es menor, 1 si es mayor o 0 si son iguales. Tiene un funcionamiento similar a la función *compareTo* que existe en otros lenguajes como Java o C#.

```
$a = 3;
$b = 5;
echo $a <=> $b;     // -1
```

- Por su parte, el operador de comprobación de nulo `??` se emplea para determinar si una variable tiene valor nulo, y ofrecer una alternativa en ese caso:

```
// Mostrará "No existe el dato" si $dato es nulo
echo $dato ?? 'No existe el dato';
```

Operaciones lógicas

Estas operaciones permiten enlazar varias comprobaciones simples, o cambiarles el sentido, según el operador. En concreto tenemos estos operadores lógicos en PHP:

Operador	Significado
`and , &&`	Operador AND (Y)
`or, ||`	Operador OR (O)
`xor`	Operador XOR
`!`	Negación (NO)

Ejemplos:

```
echo $n1 >= 0 && $n2 >= 0;
echo $n1 >= 0 || $n2 >= 0;
echo $n1 >= 0 xor $n2 >= 0;
```

El operador de negación invierte el sentido de una comprobación (si era verdadera, la vuelve falsa, y viceversa). Por ejemplo, si queremos ver si una edad no es mayor de edad, podríamos ponerlo así:

```
echo !($edad >= 18);
```

Precedencia de operadores

¿Qué ocurre si tenemos varios operadores de distintos tipos en una misma expresión? PHP sigue un orden a la hora de evaluarlos:

1. Toda expresión entre paréntesis
2. Autoincrementos y autodecrementos
3. Negaciones y cambios de signo
4. Multiplicaciones, divisiones y restos
5. Sumas, restas y concatenaciones
6. Comparaciones
7. Operaciones lógicas (`&&` , `||`)
8. Asignaciones

Existen, además, otros operadores que no hemos visto aquí, como operadores de bits, conversiones tipo cast, operador ternario... pero no son tan habituales ni importantes.

> **Ejercicio 3:**
>
> Intenta predecir qué resultado va a sacar por pantalla cada instrucción echo de este código PHP. Luego podrás comprobar si estabas en lo cierto poniendo el código en una página y probándolo en un navegador.

```
<?php
    $num1 = 3;
    $num2 = 5;
    $num3 = 8;
    $num1 *= 4;

    echo $num1;
    echo $num1 <= $num2;
    echo $num3 > $num1 and $num3 > $num2;
    echo $num3 > $num1 or $num3 > $num2;
    echo $num1 > $num2 xor $num1 > $num3;
    $num3--;
    echo $num3;
    $num3 += $num1;
    echo $num3;
?>
```

2.5. Control de flujo

2.5.1. Estructuras selectivas

A veces nos interesa realizar una operación si se cumple una determinada condición y no hacerla (o hacer otra distinta) si no se cumple esa condición. Por ejemplo, si está vacía una variable querremos hacer una cosa, y si no lo está, hacer otra. Para decidir entre varios caminos a seguir en función de una determinada condición, al igual que en otros lenguajes como Javascript, Java o C, se utiliza la estructura **if..else**:

```
if (condicion)
{
    instruccion1a;
    instruccion2a;
    ...
}
else
{
    instruccion1b;
    instruccion2b;
    ...
}
```

Si queremos elegir entre más de dos caminos, podemos utilizar la estructura *if..elseif.. elseif..* y poner una condición en cada if para cada camino. Por ejemplo, imaginemos que tenemos una variable edad donde almacenaremos la edad del usuario. Si el usuario no llega a 10 años le diremos que no tiene edad para ver la web. Si no llega a 18 años, le diremos que aún es menor de edad, pero puede ver la web, y si tiene más de 18 años le diremos que está todo correcto:

```
$edad = ...
if ($edad < 10)
{
    echo "No tienes edad para ver esta web";
}
elseif ($edad < 18)
{
    echo "Aún eres menor de edad, pero puedes acceder a esta web";
}
else
{
    echo "Todo correcto";
}
```

> **Ejercicio 4:**
>
> Crea una página llamada **prueba_if.php** en la carpeta de ejercicios del tema. Crea en ella dos variables llamadas `$nota1` y `$nota2`, y dales el valor de dos notas de examen cualesquiera (con decimales si quieres). Después, utiliza expresiones *if..else* para determinar qué nota es la mayor de las dos.

> **Ejercicio 5:**
>
> Modifica el ejercicio anterior añadiendo una tercera nota `$nota3`, y determinando cuál de las 3 notas es ahora la mayor. Para ello, deberás ayudarte esta vez de la estructura *if..elseif..else*.

También disponemos de la estructura **switch..case**, similar a otros lenguajes, para elegir entre el conjunto de valores que puede tomar una expresión:

```
switch($variable)
{
    case 0:
        echo "La variable es 0";
        break;
    case 1:
        echo "La variable es 1";
        break;
    default:
        echo "La variable es otra cosa";
}
```

2.5.2. Estructuras repetitivas o bucles

Para poder repetir código o recorrer conjuntos de datos, al igual que en otros lenguajes de programación, disponemos de algunas estructuras repetitivas o bucles.

Una de ellas es la estructura **for**. Supongamos que tenemos una variable `$lista` con una lista de elementos. Si queremos recorrerla, tenemos dos opciones: la primera es utilizar una variable que vaya desde el principio de la lista (posición 0) hasta el final (indicado por la función `count`):

```
for ($i = 0; $i < count($lista); $i++)
{
    echo $lista[i];         // Muestra el valor de cada elemento
}
```

La segunda alternativa es utilizar una variable que sirva para almacenar cada elemento de la lista. Para esta opción usamos la estructura **foreach**:

```
foreach ($lista as $elemento)
{
    echo $elemento;         // Muestra el valor de cada elemento
}
```

Además, existe una tercera forma de repetir un conjunto de instrucciones: la estructura **while**. Pondremos entre paréntesis una condición que debe cumplirse para que las instrucciones entre las llaves se repitan. Este código cuenta del 1 al 5:

```
$numero = 1;
while ($numero <= 5)
{
    echo $numero;
    $numero++;
}
```

Una variante de la estructura while es la estructura **do..while**, similar a la anterior, pero donde la condición para terminar el bucle se comprueba al final:

```
$numero = 1;
do
{
    echo $numero;
    $numero++;
} while ($numero <= 5);
```

> **Ejercicio 6:**
>
> Crea una página llamada **contador.php** en la carpeta de ejercicios del tema. Utiliza una estructura *for* para contar los números del 1 al 100 (separados por comas), y luego una estructura *while* para contar los números del 10 al 0 (una cuenta atrás, separada por guiones). Al final debe quedarte algo como esto:

```
1,2,3,4,5,6,7,8,9,10,11,12,13,14,15...
10-9-8-7-6-5-4-3-2-1-0
```

2.6. Intercalar control de flujo y HTML

Podemos intercalar bloques de código PHP y bloques HTML, incluso cortando bloques de control de flujo para introducir el código HTML. Por ejemplo:

```
<?php
if ($edad < 10=)
{
?>
   <div class="menor">Eres demasiado pequeño para entrar a esta web</div>
<?php }  elseif ($edad < 18) {  ?>
   <div class="mediano">No eres mayor de edad, pero puedes entrar</div>
<?php } else {  ?>
   <div class="mayor">Todo correcto. Bienvenido</div>
<?php } ?>
```

Sería equivalente a haber hecho un solo bloque PHP que, con instrucciones echo, sacara el contenido HTML, pero a veces es más cómodo poner el HTML directamente.

> **Ejercicio 7:**
>
> Modifica el ejercicio anterior y añádele algún *h1* y párrafos explicativos a la página, fuera del código PHP, explicando lo que se va a hacer. Por ejemplo, que te quede algo así:

Contadores

Este contador va del 1 al 100:

1,2,3,4,5,6,7,8,9,10,11,12,13,14,15...

Este otro va del 10 al 0:

10-9-8-7-6-5-4-3-2-1-0