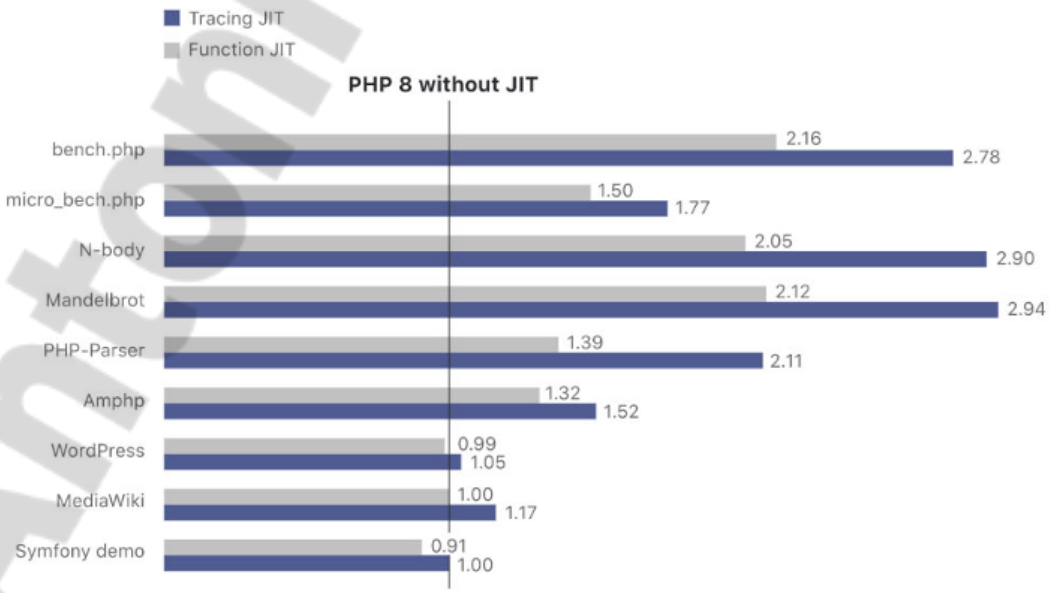
Tema 2
Conceptos básicos de PHP

PHP (Hypertext Pre-Processor) es un lenguaje de programación interpretado, concebido principalmente para el desarrollo de aplicaciones web. Incrustado en un documento HTML (o no), PHP responderá de forma dinámica a las demandas del cliente, lo que permitirá la automatización de una gran cantidad de tareas.

En sus inicios, allá por 1994, PHP no era más que un conjunto de scripts desarrollados en Perl por el ingeniero Rasmus Lerdorf, con el objetivo de controlar el acceso a sus páginas personales. Desde entonces ha evolucionado ampliamente y a día de hoy lo encontramos formando parte de millones de dominios a lo largo y ancho del globo. Actualmente se encuentra en su versión 8.0.1 incluyendo numerosas mejoras, algunas de las cuales estudiaremos a lo largo de este curso.

PHP 8 es un lenguaje interpretado y *multiparadigma*, influido por lenguajes como Perl, C o Java. Se introduce en esta nueva versión un importante cambio: la incorporación de un compilador **JIT** (compilación Just-In-Time) que interpreta y compila el código antes de su ejecución, generando un código nativo mucho más cercano a la máquina y, mejorando por tanto la velocidad de ejecución de código PHP.

Concretamente, el compilador JIT de PHP incorpora dos mecanismos: **Tracing JIT** y **Function JIT**, siendo el primero de estos el más prometedor, como podemos apreciar en los *benchmarks* oficiales publicados para la versión de PHP8.



Observamos que la incorporación de JIT mejora sustancialmente la ejecución de aquellos procesos que implican un uso intensivo de CPU. Sin embargo, dicha mejora no se pone de manifiesto en aplicaciones como Wordpress, MediaWiki o Symphony, ya que en este tipo de aplicaciones intervienen también elementos externos, como el acceso a bases de datos, penalizando de esta manera el tiempo de ejecución de los scripts PHP.

Sea como sea, la introducción de este compilador supone un paso adelante para que el lenguaje pueda ser utilizado en áreas donde anteriormente no era tan recomendado, en tanto que sus desarrolladores continúan haciendo hincapié en conseguir que las mejoras de JIT tengan un impacto más significativo en aplicaciones del día a día.

PHP incorpora además nuevas e interesantes características que se suman a las ya introducidas en su versión anterior, entre las que destacamos una mejor gestión de la memoria, reduciendo el consumo de ésta al 50% y permitiendo una mayor concurrencia de usuarios a los sitios web, además de facilitar la *retrocompatibilidad* con código escrito en versiones anteriores del lenguaje. Actualmente, PHP incluye soporte para sistemas diferentes sistemas operativos (Unix, Windows, Mac OS), servidores web (Apache, Microsoft IIS) y gestores de bases de datos (MariaDB, MySQL, PostgreSQL, DB2).

PHP es un lenguaje del lado del servidor. Esto quiere decir que el código escrito en PHP se ejecutará en el servidor, antes de dar respuesta a la petición del cliente, que recibirá el código HTML resultante tras la ejecución del script. De esta manera, a diferencia de los lenguajes del lado del cliente, los usuarios no tendrán la posibilidad de visualizar el código fuente de los scripts escritos en PHP.

Para que el servidor sea capaz de discernir cuándo debe servir el documento original, o el resultado de procesar un script escrito con un lenguaje del lado del cliente, debemos indicarlo mediante la extensión del archivo. En este caso, si utilizamos el lenguaje PHP debemos añadir la extensión apropiada (.php) para que el servidor entienda que, antes de dar una respuesta al cliente, debe ejecutar el código escrito en dicho lenguaje.

Se estima que PHP es utilizado hoy en día en el 83% de los sitios web más populares entre los usuarios, como Wikipedia, Facebook, Tumblr ó Dailymotion, además de ser el lenguaje en el que se han desarrollado gestores de contenido como Joomla, Drupal, Wordpress y Moodle.

Escribiendo PHP

Llegados a este punto se supone conocido el concepto de **script**, entendiendo por el mismo como un conjunto de instrucciones escritas en un lenguaje determinado que, incrustadas dentro de un documento web serán ejecutadas por el intérprete, en el momento en que se requiera su ejecución.

Supongo contarás en tu ordenador con un editor de código como **Sublime Text** ó **Notepad++**. Ábrelo, crea un nuevo documento, escribe en él siguiente fragmento de código y grábalo con el nombre **holamundo.php**.

```
<!DOCTYPE html>
<html>
  <head>
    <title>¡Hola mundo! en PHP</title>
  </head>
  <body>
    <?php
      echo "¡Hola mundo!" ;
    ?>
  </body>
</html>
```

Observamos que hemos incluido un fragmento de código PHP dentro de HTML, y lo hemos delimitado con las etiquetas **<?php** y **?>**. No existe límite alguno en cuanto al número de scripts que pueden añadirse a un documento web, pudiendo llegar a encontrar archivos que únicamente contengan código PHP en cuyo caso no será necesario utilizar la etiqueta **?>** para cerrar el script.

Intenta ahora abrir el archivo **holamundo.php** con el navegador. ¿Qué ha sucedido? El código PHP no tendrá efecto si el archivo no se encuentra en el servidor, ya que es en éste donde se encuentra el intérprete de PHP encargado de parsear el código y generar el resultado que se enviará al cliente. Mueve el archivo al servidor y accede a él a través del navegador. Ahora sí deberá funcionar correctamente.

Ejecutado el script y visualizado el resultado en el navegador, comprobamos que el código PHP no es visible ya que el intérprete lo ha sustituido con el resultado del script. Esto significa que desde PHP podemos generar código HTML.

```
echo "<strong>¡Hola mundo!</strong>" ;
```

La línea anterior hará que el intérprete vuelque el fragmento de código HTML indicado en el documento web que será enviado al cliente. Es importante tener en cuenta que la extensión

.php indica al servidor que el documento contiene algún script PHP que deberá ser interpretado.

Comentarios

Documentar nuestro código es siempre recomendable, con objeto de clarificar y explicar aquellos elementos del programa que consideremos importantes. Si documentamos nuestro código, lo haremos más comprensible a otros programadores que tengan que trabajar con él, además de facilitarnos las tareas de depuración y búsqueda de errores.

PHP soporta tres estilos diferentes de comentarios. Los comentarios multilínea deberán comenzar por **/*** y terminar con ***/**. También podemos añadir a nuestro código comentarios de una única línea, para lo cual nos encontramos dos posibilidades.

```
//Comentario estilo C                    # Comentario estilo shell
```

Al igual que sucede con los espacios en blanco, tabuladores y saltos de línea, el intérprete de PHP ignorará los comentarios sobre el código.

Salida básica

Las funciones de PHP que nos permiten escribir texto en el documento HTML son: **echo**, **print**, **print_r** y **var_dump**. Pero, ¿cuál debemos utilizar? Las más habituales son **echo** y **print**, generalmente utilizadas para imprimir texto en general. Ambas forman parte del lenguaje y, aunque su funcionamiento es similar, parece ser que la primera es más rápida que la segunda, permitiendo además escribir varias cadenas separadas por comas.

```
print "¡Hola mundo!" ;
echo "Hola ", "mundo" ;
```

Por otro lado, las funciones **print_r** y **var_dump** se emplean generalmente para depurar el código, imprimiendo su valor y detalles de las variables en un formato. Encontraremos quizá más útil la función **var_dump** ya que, además proporciona información sobre el tamaño, tipo de datos de las variables y, en el caso de *arrays*, de los elementos que lo componen. Más adelante veremos cómo utilizar estas funciones.

Variables y constantes

Podemos ver una **variable** como una pequeña caja, identificada mediante un nombre, donde podemos guardar de forma temporal un valor. En realidad, una variable no es más que un espacio de memoria donde guardaremos valores de cualquier tipo, que serán susceptibles de ser modificados a lo largo del script.

En el apartado anterior dijimos que, a diferencia de lo que ocurre en otros lenguajes, en PHP no es necesario definir las variables, basta con escribir su **nombre** o **identificador** en el mismo instante en que queramos darle un valor. El nombre de una variable deberá comenzar obligatoriamente por **$** y han de llevar una letra inmediatamente después de dicho símbolo. Por ejemplo, **$pepe1** es un nombre válido, pero **$1pepe** no lo es. Además, PHP diferencia entre mayúsculas y minúsculas, por lo que en ese caso **$pepe** es diferente a **$Pepe**. Recuerda que el nombre de una variable deberá ser significativo según el valor que contenga y que se escribirán empleando la regla de *Camel Case*.

Constantes

El concepto de **constante** es idéntico al de variable, pero con una salvedad, una vez definida, su valor no podrá modificarse. Las constantes pueden ser **literales** (un valor como 3.1416), aunque para mayor comodidad, a cada uno de esos valores se le asigna un nombre, de modo que cuando vaya a ser utilizado baste con invocar ese nombre. Estas constantes se conocen como **simbólicas** y, por convenio deberemos escribir su nombre en mayúsculas. En PHP las constantes se definen utilizando la función **define** o la palabra reservada **const**.

```
define("PI",3.1416) ;
const PI = 3.1416 ;
```

Como vemos, existen dos métodos diferentes para definir constantes, pero entonces, ¿cuál debe utilizarse? A la hora de elegir una u otra opción, debemos tener en cuenta que **define** es una función, en tanto que **const** es una construcción propia del lenguaje que podemos definir incluso en el interior de clases, interfaces y rasgos, como veremos en la unidad dedicada a la programación orientada a objetos.

Sin embargo, **define** presenta una serie de ventajas, que podrían hacer que nos inclinemos por su uso en detrimento de **const**. La función **define** nos permite crear constantes condicionales, además de aceptar cualquier tipo de expresión válida como nombre para nuestra constante, permitiendo la definición de constantes no sensibles a minúsculas y mayúsculas. En definitiva, a menos que necesitemos alguna de las características propias de **define** que acabamos de mencionar, se recomienda encarecidamente el utilizar la palabra reservada **const** para definir constantes en nuestro script.

Ámbito de las variables

El término **ámbito** hace referencia a los lugares de nuestro código en los que será visible una variable. Existen diferentes reglas de ámbito básicas.

a. Las **variables superglobales** son visibles desde cualquier punto del script. Este conjunto de variables es bastante amplio, contando entre ellas variables como **$_GLOBALS**, **$_SERVER**, **$_GET**, **$_POST**, **$_COOKIE**, **$_FILES**, etc. Estudiaremos algunas de ellas en temas sucesivos.
b. Las **constantes**, una vez declaradas, son visibles globalmente. Esto significa que también existirán en el interior de las funciones.
c. Las **variables globales** definidas en un script serán visibles desde cualquier punto del código, exceptuando en el interior de las funciones.

El ámbito de una **variable** definida en el interior de una función se reduce únicamente a dicha función. Las variables definidas como estáticas en el interior una función, serán invisibles fuera de ésta, pero guardarán su valor entre sucesivas llamadas a dicha función

Tipos de datos

Si recordamos del curso pasado, el **tipo de dato** nos permitirá conocer el conjunto de valores que podrá tomar una variable, así como las operaciones que se podrán realizar sobre ella. PHP incorpora cuatro tipos básicos, cuatro compuestos y dos especiales. De entre todos ellos, nosotros nos centraremos en los siguientes.

Tipos	Descripción	Ejemplos
boolean	Permite expresar un valor de verdad utilizando las constantes **true** y **false**.	`$foo = true ;` `$foo = false ;`

integer	Comprende el conjunto de los números enteros. Se puede especificar un valor de este tipo utilizando notación **decimal**, **hexadecimal** (0x), **octal** (0) ó **binaria** (0b).	`$foo = 9012 ;`
		`$foo = -231 ;`
		`$foo = 0231 ;`
		`$foo = 0x1A ;`
		`$foo = 0b11111111 ;`

float	Es también conocido como **double** y permite representar números reales. Podrán expresarse igualmente utilizando notación científica.	`foo = 1.234 ;`
		`$foo = 1.2e3 ;`
		`$foo = 7E-10 ;`

string	Utilizado para cadenas de caracteres. Entrecomillamos la secuencia de caracteres utilizando **comillas simples** o **dobles** indistintamente.	`$foo = "¡Hola mundo!" ;`

Tipos	Descripción	Ejemplos
array	Tipo compuesto que nos permite almacenar un conjunto ordenado de valores. Se estudiará más adelante.	-
object	Utilizado para almacenar instancias de clases. Nos centraremos en este tipo más adelante, cuando tratemos los conceptos de la programación orientada a objetos.	-
NULL	Se consideran de este tipo aquellas variables que no tienen un valor definido o que se les ha asignado explícitamente el valor NULL.	-
resource	Este tipo de variables contendrá una referencia a un recurso externo.	-

Al contrario de lo que ocurre con lenguajes como C o Java, PHP es un lenguaje **débilmente tipado**. Esto quiere decir que el tipo de la variable quedará determinado por el valor que se le asigne, por lo que no será necesario definir la variable previamente e indicar el tipo de la misma.

Operadores

Definimos los operadores como un conjunto de símbolos o combinaciones de éstos, que nos permitirán manipular valores y variables, realizando operaciones sobre ellos. Los argumentos de cualquier operador se conocen con el nombre de **operados**. Hasta ahora hemos utilizado uno de ellos, que es el primero que estudiamos en esa sección.

Asignación

El operador **asignación** (=) es el más básico incluido en la mayoría de los lenguajes de programación. Nos permitirá asignar valores a las variables definidas en nuestro código, debiendo aparecer el nombre de ésta a la izquierda del operador, y el valor a la derecha.

Concatenación

El operador **concatenación** (.) es específico de las cadenas de caracteres y se utiliza para unir diferentes porciones de texto. Los diferentes elementos a concatenar pueden ser cadenas literales, o fragmentos de texto almacenados en una variable. Veamos un ejemplo:

```
echo "Bienvenido/a, ".$nombre ;
```

La instrucción anterior imprimirá en el documento HTML la cadena entrecomillada, seguida del valor almacenado en la variable **$nombre**. Observa ahora la siguiente sentencia.

```
echo "Bienvenido/a, $nombre";
```

Vemos que hemos incluido la variable en la cadena y, si comprobamos el resultado en pantalla, veremos que es idéntico al anterior. Este método es conocido como **interpolación** y únicamente podemos utilizarlo con comillas dobles; esto es, la siguiente línea de código sería errónea.

```
echo 'Bienvenido/a, $nombre';
```

Aritméticos

Iguales a los de cualquier otro lenguaje de programación, los operadores **aritméticos** nos permitirán realizar operaciones básicas de suma, resta, producto y división.

Operador	Significado
-	Resta, negativo
+	Suma
*	Multiplicación
/	División (real)
%	Módulo
--	Pre/postdecremento
++	Pre/postincremento

Merecen especial mención los dos últimos operadores aritméticos: **pre/postincremento** y **pre/postdecremento**. Bastante útiles, el primero de ellos añade la unidad a su operando y el segundo la restará, siendo por tanto **operadores monarios**. En otras palabras:

```
$x++ ;       // equivale a $x = $x + 1 ;
--$x ;       // equivale a $x = $x - 1 ;
```

Pre y post incremento/decremento tienen la misma finalidad, esto es, incrementar o decrementar en la unidad el valor de una variable. Observa el siguiente ejemplo:

```
$a = 15 ;
echo $a++ ;
```

Mostrará por pantalla el valor de la variable y, posteriormente incrementará su valor. Sin embargo, si hacemos:

```
echo --$a ;
```

Incrementará el valor previamente y, a continuación mostrará el valor por pantalla. PHP también nos permite utilizar **abreviaturas** con las operaciones básicas de suma, resta, producto, división y módulo. De esta manera:

```
$a += 3 ;    // es equivalente a $a = $a + 3 ;
```

Cuando una misma instrucción contiene una secuencia con varias operaciones el orden de ejecución de las mismas sigue los mismos criterios que en matemáticas; no se realiza una ejecución secuencial sino que se respeta el orden de prioridad matemático. No obstante se recomienda el uso de paréntesis para establecer las prioridades y ahorrarnos posibles problemas.

Aunque no es operador propiamente dicho, PHP 7 introdujo la función **intdiv()** una nueva función que nos permitirá obtener el cociente de una **división entera**.

```
intdiv(3,2) ;// dará como resultado 1
```

Relacionales o de comparación

Los operadores **relacionales** nos permitirán comparar el valor de dos variables o datos. Cuando hablamos de operadores relaciones, nos referimos a la relación entre unos valores y otros, en tanto que el segundo establece las diferentes formas en que esas relaciones pueden conectarse entre sí. El resultado de evaluar una expresión relacional nos da como resultado un valor **true** o **false**.

Operador	Significado
>	Mayor
>=	Mayor o igual
<	Menor
<=	Menor o igual
!=	Distinto
==	Igual (no estricto)
===	Comprueba si son iguales y además del mismo tipo
??	Devuelve el valor del primer operando que no sea NULL.

Nos detenemos en este apartado para introducir una de las novedades de PHP8. En versiones anteriores del lenguaje, cuando se realizaba una comparación no estricta entre valores de tipo cadena y valores numéricos, se convertía la cadena a un valor numérico y posteriormente se realizaba la comparación. Esto arrojaba en ocasiones resultados no esperados. Supongamos la siguiente expresión.

```
0 == "foo" ; // se evalúa como true
```

La cadena no contiene ningún valor numérico y, sin embargo el operador de igualdad nos indica que ambos operandos son iguales. Otros ejemplos podrían ser:

```
0 == "0" ;
0 == "0.0"  ;
0 == "" ;
15 == "     15" ;
15 == "15foo" ;
```

Todas las comparaciones anteriores se evaluarían como **true**. PHP8 corrige este problema convirtiendo la cadena a un valor numérico y realizando la comparación a continuación. Sin embargo, si la cadena contiene algún dígito, será el valor numérico el que se transforme a cadena antes de evaluar la igualdad. De esta manera, los ejemplos anteriores se evaluarán como vemos.

```
0 == "0" ;        // se evalúa como true
0 == "0.0"  ;     // se evalúa como true
0 == "" ;         // se evalúa como false
0 == "foo" ;      // se evalúa como false
15 == "15" ;      // se evalúa como true
15 == "     15" ; // se evalúa como true
15 == "15foo" ;   // se evalúa como false
```

Lógicos

Los operadores **lógicos** se emplean a la hora de construir condiciones más o menos complejas, permitiéndonos evaluar un conjunto de valores lógicos, esto es, aquellos que sean únicamente **true** o **false**.

Operador	Equivalente	Descripción
&&	**and**	Devuelve **true** cuando ambas operandos son ciertos.
||	**or**	Devuelve **true** cuando una de los operandos son ciertos.
!	**not**	Devuelve **true** si el operando es falso y viceversa

Spaceship operator

Incorporado en la versión 7 de PHP, el **operador spaceship** (**<=>**) se utiliza para comparar dos expresiones, de manera que devolverá **-1**, **0** ó **1** cuando la primera expresión es respectivamente menor, igual ó mayor que la segunda. Observa los siguientes ejemplos:

```
echo 1 <=> 1 ;            // devuelve 0 ya que los valores son iguales
echo 1 <=> 2 ;            // devuelve -1 ya que 1 < 2
echo 2 <=> 1 ;            // devuelve 1 ya que 2 > 1
```

Bibliografía

PHP.net
Guía oficial del lenguaje

PHP and MySQL Web Development
Luke Welling, Laura Thompson
Developer's Library

Learning PHP7
Antonio López
Packt Publishing