

Introducción a PHP

Este documento es una introducción a la programación con el lenguaje de servidor web PHP en su versión 8. Haremos un repaso a sus características y describiremos los componentes de la plataforma. Terminaremos realizando la instalación del servidor web Apache con el módulo PHP y MySQL

Introducción a PHP

Copyright © 2024 by Rafael Lozano Luján.

Este documento está sujeto a derechos de autor. Todos los derechos están reservados por el Autor de la obra, ya sea en su totalidad o en parte de ella, específicamente los derechos de:

- La reproducción total o parcial mediante fotocopia, escaneo o descarga.
- La distribución o publicación de copias a terceros.
- La transformación mediante la modificación, traducción o adaptación que altere la forma de la obra hasta obtener una diferente a la original.

Tabla de contenido

| | | |
|-------|---|----|
| 1 | Introducción..... | 1 |
| 1.1 | Características de PHP..... | 1 |
| 2 | Instalación de los componentes..... | 2 |
| 2.1 | Instalación de Apache Web Server..... | 3 |
| 2.2 | Instalación de MySQL..... | 4 |
| 2.3 | Instalación del módulo PHP..... | 6 |
| 2.4 | Crear un host virtual en Apache..... | 7 |
| 3 | Herramientas para programar en PHP..... | 9 |
| 3.1 | Instalación de Visual Studio Code..... | 9 |
| 3.2 | Git y GitHub..... | 10 |
| 3.3 | Documentación PHP..... | 13 |
| 3.4 | Primer script PHP..... | 13 |
| 4 | Elementos del lenguaje..... | 15 |
| 4.1 | Contenido mixto..... | 15 |
| 4.2 | Separación de instrucciones..... | 16 |
| 4.3 | Entrada y salida..... | 16 |
| 4.3.1 | echo..... | 17 |
| 4.3.2 | print..... | 18 |
| 4.4 | Comentarios..... | 19 |
| 4.5 | El archivo de configuración php.ini..... | 19 |
| 5 | Tipos de datos..... | 22 |
| 5.1 | Boolean..... | 23 |
| 5.2 | Numéricos enteros integer..... | 24 |
| 5.3 | Números en punto flotante..... | 25 |
| 5.4 | Cadena de caracteres o String..... | 25 |
| 5.4.1 | Comillas simples..... | 25 |
| 5.4.2 | Comillas dobles..... | 26 |
| 5.4.3 | Heredoc..... | 26 |
| 5.4.4 | Nowdoc..... | 27 |
| 5.4.5 | Conversión a string..... | 27 |
| 6 | Variables..... | 27 |
| 6.1 | Variables predefinidas..... | 28 |
| 6.2 | Análisis de variables..... | 29 |
| 6.2.1 | Sintaxis simple..... | 30 |
| 6.2.2 | Sintaxis compleja..... | 31 |
| 6.3 | Funciones de manejo de variables..... | 32 |
| 6.4 | Constantes..... | 33 |
| 7 | Operadores y expresiones..... | 34 |
| 7.1 | Operadores..... | 35 |
| 7.1.1 | Asignación..... | 35 |
| 7.1.2 | Aritméticos..... | 37 |
| 7.1.3 | Relacionales..... | 38 |
| 7.1.4 | Lógicos..... | 39 |
| 7.1.5 | Operador de concatenación de cadenas..... | 41 |

| | |
|--|----|
| 7.2 Precedencia y asociatividad de operadores..... | 41 |
| 8 Bibliografía..... | 43 |

Introducción a PHP

1 Introducción

PHP, acrónimo de Hypertext Preprocessor, es un lenguaje de scripting de propósito general y de código abierto que está especialmente pensado para el desarrollo web y que puede ser incrustado en páginas web HTML. Tiene una sintaxis parecida a C, Java y Perl, siendo por tanto sencillo de aprender. El objetivo principal de este lenguaje es permitir a los desarrolladores web desarrollar páginas web generadas dinámicamente, es decir, PHP es un potente y simple lenguaje diseñado para crear contenido HTML, aunque también puede crear contenido en otros formatos.

1.1 Características de PHP

PHP se puede emplear de dos formas:

- ✓ Scripting del lado de servidor → PHP fue originalmente diseñado para crear contenido web dinámico y aún es su principal tarea. Para generar contenido HTML necesitamos un módulo PHP y un servidor web a través del cual se envían los documentos codificados. PHP también es popular para generar contenido dinámico a través de conexiones con bases de datos, documentos XML, gráficos, archivos PDF y mucho más.
- ✓ Scripting en línea de comando → PHP puede ejecutar scripts desde la línea de comando, como Perl o Bash script. Podemos usar los scripts en línea de comando para tareas administrativas, como backup o traducción de logs, e incluso podemos ejecutar scripts planificados con CRON, siempre que sean tareas sin visualización.

PHP se ejecuta en la mayoría de los sistemas operativos, como distribuciones Linux (FreeBSD, Ubuntu, Debian y Solaris), Windows y macOS. También, puede emplearse en los

servidores web más empleados, como Apache, Nginx y OpenBSD. Por último, también es operativo en entornos en la nube como Azure, AWS, ...

El lenguaje es extremadamente flexible. Por ejemplo, el resultado de la ejecución de un script PHP no está limitado únicamente a HTML u otro fichero de texto, cualquier formato de documento puede ser generado por PHP ya que tiene soporte para archivos PDF, GIF, JPEG y PNG

Una de sus características más significantes es su amplio rango de soporte para bases de datos. PHP soporta la mayoría de los SDBD, como MySQL, PostgreSQL, Oracle, Sybase, MS-SQL, DB2 y ODBC. Bases de datos no SQL, como CouchDB y MongoDB también están soportadas. Con PHP, crear páginas web con contenido dinámico desde una base de datos es muy simple.

Finalmente, PHP proporciona una biblioteca de código PHP para realizar tareas comunes, como abstracción de bases de datos, gestión de errores, etc. con la *PHP Extension and Application Repository* (PEAR), un framework y sistema de distribución de componentes PHP reutilizable.

2 Instalación de los componentes

Generalmente, un servidor web que distribuye contenido dinámico generado por PHP tiene instalado un servicio web al cual se le ha añadido el módulo PHP. Por tanto, para disponer de un servicio web con PHP necesitamos:

1. Un host que actúa como servidor web en cualquiera de sus formas: servidor físico, instancia en la nube, servidor privado virtual, ... En nuestro caso usaremos Ubuntu Desktop 24.04.
2. Un servicio web instalado, como Apache o Nginx.
3. El módulo PHP instalado y configurado en el servicio web anterior.

Es posible que las aplicaciones web a desarrollar accedan a los datos almacenados y gestionados por un SGBD, el cual puede estar instalado en el mismo host o en otro diferente. En el primer caso, necesitaremos tener instalado también el SGBD, como MySQL u Oracle, en el mismo host. En el segundo debemos configurar el host con la base de datos para que acepte peticiones desde host que ejecuta el servidor web con el módulo PHP.

Como se mencionó anteriormente, PHP está disponible en muchos sistemas operativos y plataformas. Por tanto, tendremos que consultar la documentación PHP para encontrar el entorno que se ajuste mejor al que usaremos y seguir las apropiadas instrucciones de instalación. En este documento emplearemos una plataforma Linux Ubuntu.

Nuestra implementación está enfocada al desarrollo de aplicaciones por lo que emplearemos un único host para todos los componentes. Partimos de que ya tenemos instalado el sistema operativo en el host y disponemos de una cuenta de usuario con perfil administrador.

A la hora de instalar los componentes anteriores tenemos dos opciones:

- ✓ La pila LAMP → Consiste en instalar por separado Linux, Apache, MySQL y PHP.
- ✓ XAMPP → Es una distribución de Apache completamente gratuita y de fácil instalación contiene MariaDB, PHP y Perl. Hay versiones para Windows, Linux y OSX.

En este documento veremos la primera opción, mientras que para la segunda se puede consultar la documentación en <https://www.apachefriends.org/es/index.html> para ver como se hace.

2.1 Instalación de Apache Web Server

La instalación de Apache en Ubuntu requiere abrir una ventana de terminal y ejecutar el siguiente comando:

```
$ sudo apt update
$ sudo apt install apache2
```

Si es la primera vez que utilizamos `sudo` en esta sesión, se nos pedirá que proporcionemos la contraseña de usuario para confirmar que tenemos los privilegios adecuados para administrar los paquetes de instalación con `apt`. También solicitará que confirmemos la instalación de Apache.

Una vez instalado Apache podemos realizar una verificación rápida para comprobar que el servidor está en funcionamiento abriendo el navegador y escribiendo en la barra de dirección lo siguiente:

```
http://localhost
```

Se debe presentar la siguiente página:

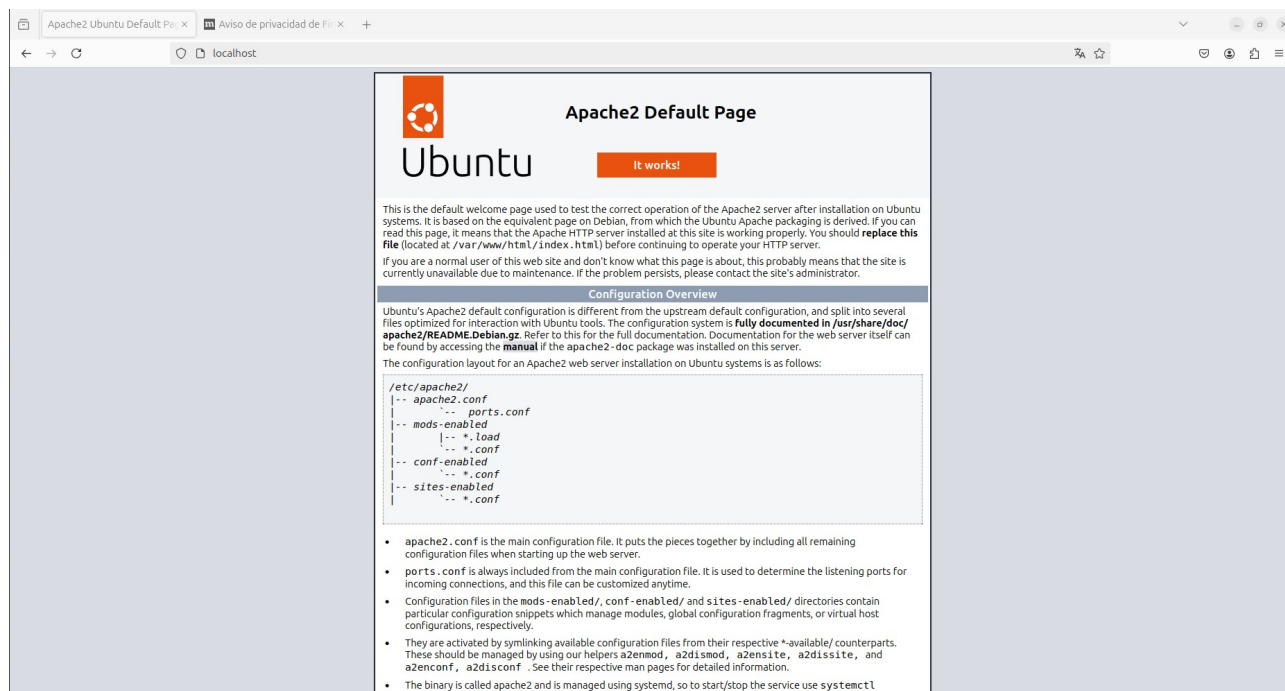


Figura 1.- Página por defecto de Apache en Ubuntu

2.2 Instalación de MySQL

Ahora que disponemos de un servidor web funcional, debemos instalar un SGBD para poder almacenar y gestionar los datos del sitio. MySQL es un SGBD popular que se utiliza en entornos PHP.

Aunque en los repositorios de Ubuntu disponemos de un paquete de instalación de MySQL podemos obtener el más actualizado en <https://dev.mysql.com/downloads/mysql/>. Cuando entremos tendremos que elegir la versión adecuada a nuestra plataforma y en el caso de Ubuntu lo mejor es utilizar el sistema `apt`. Para ello MySQL pone a disposición de los usuarios un paquete que añade un repositorio en Ubuntu para realizar la instalación de la última versión mediante `apt`. Para ello hacemos lo siguiente:

1. En `mysql.com` hacemos clic sucesivamente en los enlaces *Download* → *MySQL Community (GPL) Downloads* → *MySQL Community Server*.
2. Seleccionamos la última versión LTS disponible para Ubuntu 24.04.
3. Hacemos clic en *Download Now* en la sección MySQL APT Repository.

MySQL Community Downloads

MySQL Community Server

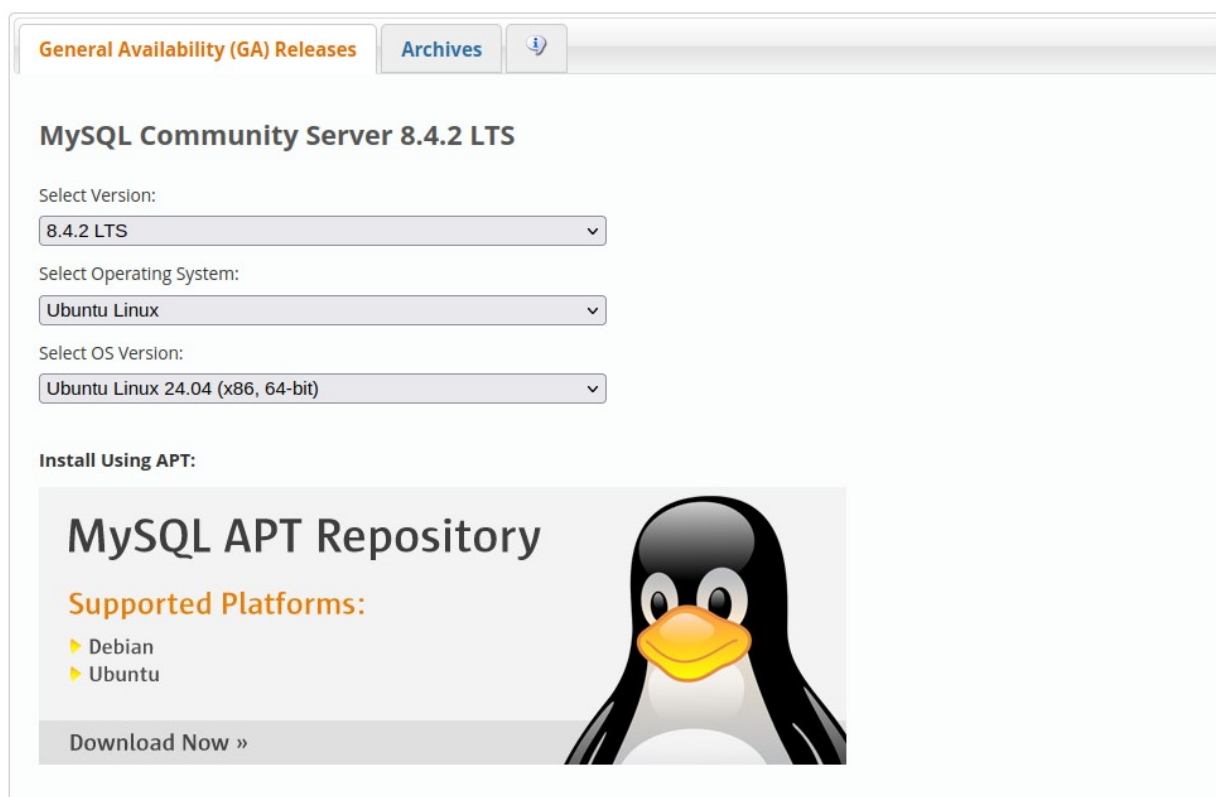


Figura 2.- Descarga de MySQL

- Se descargará un paquete de instalación en la carpeta Descargas. Abrimos una ventana de terminal y como usuario administrador ejecutamos el siguiente comando de instalación.

```
$ sudo dpkg -i mysql-apt-config_versión_all.deb
```

- Nos aparece una ventana para seleccionar el producto que queremos instalar. Dejamos activado por defecto *MySQL Server & Cluster* y seleccionamos *Aceptar*.

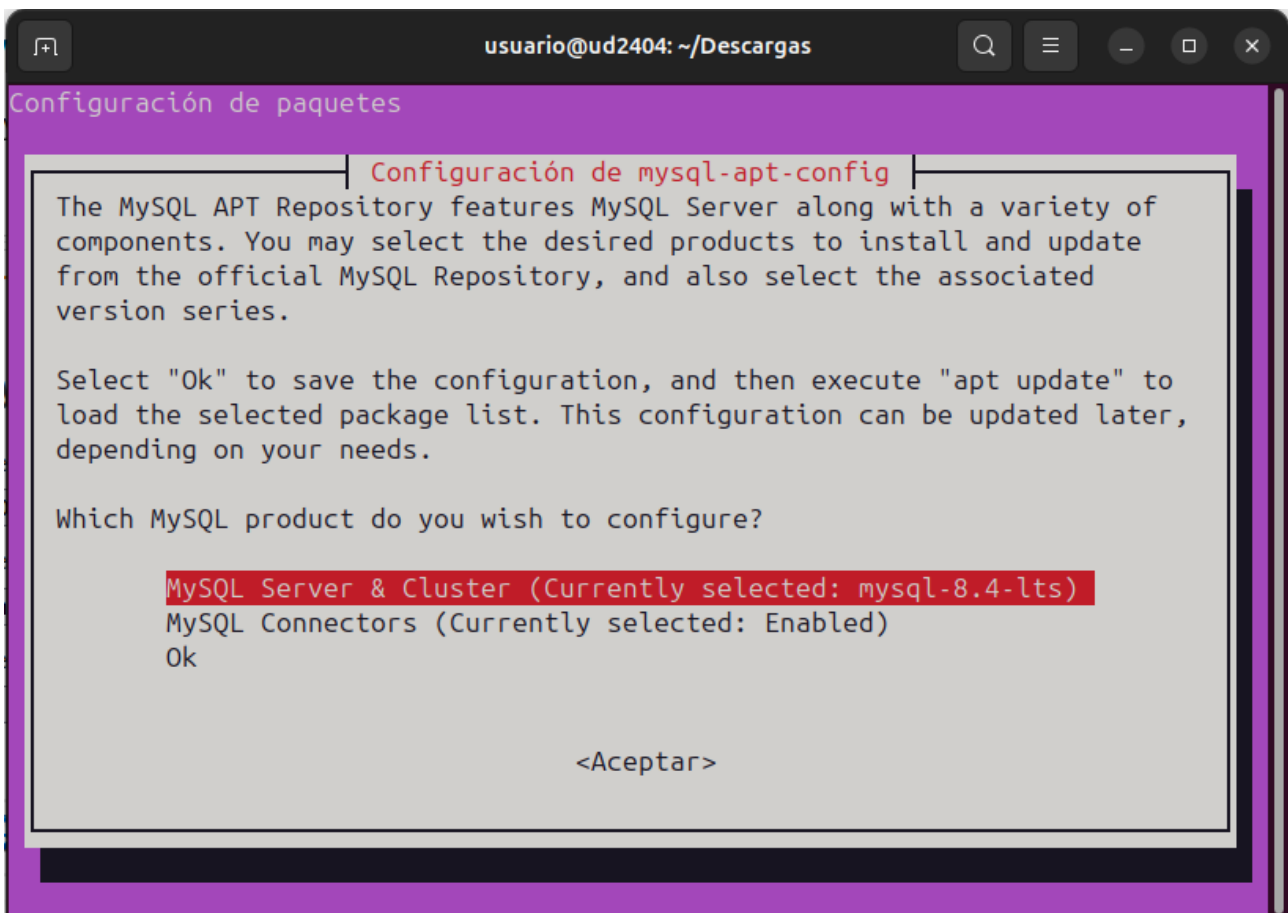


Figura 3.- Selección del producto

- Dejamos seleccionada la versión LTS de MySQL y seleccionamos *Aceptar*.
- En la pantalla de selección del producto seleccionamos *Ok* y posteriormente *Aceptar*.
- La instalación del paquete anterior ha modificado los orígenes de los repositorios para incluir uno desde donde podemos instalar MySQL. De nuevo en la ventana de terminal actualizamos la lista de paquetes en los repositorios.

```
$ sudo apt update
```

- Ahora ya podemos instalar la última versión de MySQL mediante *apt*. El comando siguiente instala el paquete para el servidor MySQL así como los paquetes para el cliente y archivos de base de datos comunes.

```
$ sudo apt install mysql-server
```


10. A continuación nos preguntará la clave para el usuario `root`. La introducimos y continuamos.

Al finalizar, el servidor MySQL arranca automáticamente. Podemos comprobar el estado con el siguiente comando.

```
$ sudo systemctl status mysql
```

Cuando la instalación se complete, se recomienda ejecutar una secuencia de comandos de seguridad que viene preinstalada en MySQL. Con esta secuencia de comandos se eliminarán algunos ajustes predeterminados poco seguros y se bloqueará el acceso a su sistema de base de datos. Iniciamos la secuencia de comandos interactiva ejecutando lo siguiente:

```
$ sudo mysql_secure_installation
```

La ejecución del script de configuración de MySQL nos pedirá la clave de usuario `root` que introdujimos anteriormente. Después hará las siguientes preguntas las cuales tendremos que contestar:

1. Activar la comprobación de complejidad de las contraseñas. Con esto los usuarios solamente podrán establecer contraseñas si son lo bastante fuertes.
2. Cambiar la contraseña del usuario `root`.
3. Eliminar el usuario anónimo.
4. Desactivar el usuario `root` remotamente.
5. Eliminar la base de datos `test`.
6. Recarga de las tablas con los privilegios de usuario.

Una vez tenemos el servidor instalado y funcionando podemos realizar una instalación de MySQL Workbench para tener un cliente de acceso a MySQL con entorno gráfico. Para instalar Workbench en Ubuntu seguimos los siguientes pasos:

1. Si estamos instalando únicamente Workbench en nuestro sistema ya que el servidor se encuentra en otro ordenador tendremos primero que añadir el repositorio APT de MySQL a la lista de repositorios del sistema, tal y como se vio en la instalación del servidor.
2. A continuación abrimos una ventana de terminal y ejecutamos el siguiente comando para instalar Workbench.

```
$ sudo apt-get install mysql-workbench-community
```

3. Cuando finalice podemos hacer una búsqueda en el tablero del Workbench y ejecutarlo.

2.3 Instalación del módulo PHP

Instalamos Apache como servidor web y MySQL como SGBD para almacenar y gestionar los datos. PHP es el componente de nuestra configuración que procesará el código PHP para mostrar contenido dinámico al usuario final. Además del paquete php,

necesitaremos el paquete `php-mysql`, un módulo PHP que permite que este se comunique con bases de datos basadas en MySQL. También necesitará `libapache2-mod-php` para habilitar Apache para gestionar archivos PHP. Los paquetes PHP básicos se instalarán automáticamente como dependencias.

En la ventana de terminal ejecutamos el siguiente comando `apt`:

```
$ sudo apt install php php-mysql
```

Una vez que la instalación se complete, podremos ejecutar el siguiente comando para confirmar su versión de PHP:

```
$ php -v
PHP 8.3.6 (cli) (built: Jun 13 2024 15:23:20) (NTS)
Copyright (c) The PHP Group
Zend Engine v4.3.6, Copyright (c) Zend Technologies
    with Zend OPcache v8.3.6, Copyright (c), by Zend Technologies
```

Debe aparecer la versión del módulo PHP para Apache.

En este punto, la pila LAMP está plenamente operativa, pero, para poder probar la configuración con una secuencia de comandos PHP, lo mejor es instalar un host virtual de Apache adecuado para almacenar los archivos y las carpetas del sitio web. Lo haremos en el siguiente paso.

2.4 Crear un host virtual en Apache

Al emplear el servidor web Apache, podemos crear hosts virtuales para alojar más de un sitio web en un único servidor y encapsular detalles de configuración de cada sitio de forma independiente. Vamos a configurar un dominio cuyo nombre será `dwes.com`, pero en un escenario real deberemos cambiar este nombre por el nuestro dominio.

Ubuntu tiene creado por defecto un sitio web cuya raíz de documentos es el directorio `/var/www/html`. En lugar de realizar modificaciones en `/var/www/html`, crearemos una estructura de directorio propia en `/home/usuario/dwes` que será la raíz de documentos para el sitio web de nuestro host virtual, y dejaremos el sitio web por defecto sin tocar.

Cree el directorio en una ventana de terminal con el siguiente comando:

```
$ sudo mkdir /var/www/dwes.com
```

Como vamos a poner en este directorio los archivos de nuestras aplicaciones web le cambiamos el propietario al usuario de nuestra instalación.

```
$ sudo chown -R usuario: /var/www/dwes.com
```

A modo de prueba podemos crear un archivo `index.html` en el directorio recién creado con el siguiente contenido:

```
<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-
```

```
scale=1.0">
  <title>Sitio web virtual DWES.com</title>
</head>
<body>
<h1>Bienvenidos a DWES.com</h1>
</body>
</html>
```

A continuación creamos el archivo con la configuración de nuestro sitio web llamado `dwes.conf` que ubicaremos en el directorio `/etc/apache2/sites-available` de Apache usando cualquier editor de texto plano. En este caso, podemos utilizar `nano`:

```
$ sudo nano /etc/apache2/sites-available/dwes.com.conf
```

El contenido del archivo anterior será el siguiente:

```
<VirtualHost *:80>
    ServerName dwes.com
    ServerAlias www.dwes.com
    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/dwes.com

    <Directory /var/www/dwes.com>
        Require all granted
    </Directory>

    ErrorLog ${APACHE_LOG_DIR}/error_dwes.com.log
    CustomLog ${APACHE_LOG_DIR}/access_dwes.com.log combined
</VirtualHost>
```

Ahora, ejecutamos la utilidad `a2ensite` para habilitar el nuevo host virtual:

```
$ sudo a2ensite dwes.com
$ sudo systemctl reload apache2
```

Para verificar que todo funciona correctamente vamos a crear el siguiente script llamado `info.php` en el directorio anterior y con el siguiente contenido.

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

Si abrimos el navegador y escribimos en la barra de dirección <http://dwes.com/info.php> veremos lo siguiente.

| PHP Version 8.3.6 | |
|---|---|
| System | Linux uid404 6.8.0-44-generic #44-Ubuntu SMP PREEMPT_DYNAMIC Tue Aug 13 13:35:26 UTC 2024 x86_64 |
| Build Date | Jun 13 2024 15:23:20 |
| Build System | Linux |
| Server API | Apache 2.0 Handler |
| Virtual Directory Support | disabled |
| Configuration File (php.ini) Path | /etc/php/8.3/apache2 |
| Loaded Configuration File | /etc/php/8.3/apache2/php.ini |
| Scan this dir for additional .ini files | /etc/php/8.3/apache2/conf.d |
| Additional .ini files parsed | /etc/php/8.3/apache2/conf.d/10-mysqld.ini, /etc/php/8.3/apache2/conf.d/10-opcache.ini, /etc/php/8.3/apache2/conf.d/10-pdo.ini, /etc/php/8.3/apache2/conf.d/20-calendar.ini, /etc/php/8.3/apache2/conf.d/20-cyber.ini, /etc/php/8.3/apache2/conf.d/20-exif.ini, /etc/php/8.3/apache2/conf.d/20-ftp.ini, /etc/php/8.3/apache2/conf.d/20-gdlib.ini, /etc/php/8.3/apache2/conf.d/20-gettext.ini, /etc/php/8.3/apache2/conf.d/20-iconv.ini, /etc/php/8.3/apache2/conf.d/20-mysql.ini, /etc/php/8.3/apache2/conf.d/20-pdo_mysql.ini, /etc/php/8.3/apache2/conf.d/20-phar.ini, /etc/php/8.3/apache2/conf.d/20-posix.ini, /etc/php/8.3/apache2/conf.d/20-readline.ini, /etc/php/8.3/apache2/conf.d/20-shmop.ini, /etc/php/8.3/apache2/conf.d/20-sockets.ini, /etc/php/8.3/apache2/conf.d/20-sysmsg.ini, /etc/php/8.3/apache2/conf.d/20-syssem.ini, /etc/php/8.3/apache2/conf.d/20-sysvshm.ini, /etc/php/8.3/apache2/conf.d/20-tokenizer.ini |
| PHP API | 20230831 |
| PHP Extension | 20230831 |
| Zend Extension | 420230831 |
| Zend Extension Build | API420230831.NTS |
| PHP Extension Build | API20230831.NTS |
| Debug Build | no |
| Thread Safety | disabled |
| Zend Signal Handling | enabled |
| Zend Memory Manager | enabled |
| Zend Multibyte Support | disabled |
| Zend Max Execution Timers | disabled |
| IPv6 Support | enabled |
| DTrace Support | disabled |
| Registered PHP Streams | https, ftps, compress.zlib, php, file, glob, data, http, ftp, phar |
| Registered Stream Socket Transports | tcp, udp, unix, udg, ssl, tls, tlsv1.0, tlsv1.1, tlsv1.2, tlsv1.3 |
| Registered Stream Filters | zlib.*, string.rot13, string.rot48, string.toupper, string.tolower, convert.*, consumed, dechunk, convert.iconv.* |

This program makes use of the Zend Scripting Language Engine:
 Zend Engine v4.3.6, Copyright (c) 2023 Zend Technologies with Zend OPcache v8.3.6, Copyright (c), by Zend Technologies

Figura 4.- Ejecución de phpinfo()

3 Herramientas para programar en PHP

Los componentes instalados en el epígrafe anterior se emplearán para poder ejecutar aplicaciones web escritas en PHP con acceso a datos en un SGBD. Sin embargo, para el desarrollo de aplicaciones web no es suficiente, necesitamos otras estrechamente vinculadas al desarrollo de aplicaciones que también tendremos que instalar. Estas son:

1. Un IDE específico para PHP.
2. Un gestor de repositorios.

En las siguientes secciones se verá la instalación y configuración de estos componentes.

3.1 Instalación de Visual Studio Code

Los scripts PHP se pueden escribir en cualquier editor de texto. Posteriormente, se publican en un directorio del servidor web y están preparados para recibir peticiones HTTP. Sin embargo, lo ideal es utilizar un editor de código PHP o un IDE específico para PHP que facilitan la tarea de desarrollar aplicaciones web en PHP. De entre los editores de PHP más destacados están Visual Studio Code, el cual instalaremos en la siguiente sección.

La instalación de VSCode en una plataforma Ubuntu Desktop se realiza de la siguiente forma:

1. Abrir el navegador e ir a <https://code.visualstudio.com/>.
2. Hacer clic en *Download*.

3. Hacer clic en la versión que mejor se ajuste a nuestro sistema. En nuestro caso es *.deb Debian Ubuntu*.
4. Una vez terminada la descarga abrir una ventana de terminal y situarse en el directorio de descarga.
5. Ejecutar el siguiente comando (el número de versión en el paquete *.deb* puede variar):

```
$ sudo dpkg -i code_1.79.2-1686734195_amd64.deb
```

6. Una vez terminada la instalación buscamos en el panel de actividades VSCode y lo ejecutamos.
7. Como VSCode puede utilizarse para el desarrollo en múltiples lenguajes de programación, tenemos que instalar las extensiones para PHP. Hacemos clic en el botón *Extensions*.
8. En el cuadro de búsqueda escribimos PHP y debajo aparece la lista de extensiones relacionadas con PHP.
9. Seleccionamos *PHP Intelephense* y hacemos clic en *Install*.
10. Seleccionamos *PHP Debug* y hacemos clic en *Install*.

Una vez instalada la extensión para programar en PHP podemos añadir el directorio `/home/usuario/dwes.com` configurado en el punto anterior para usarlo como lugar donde colocaremos nuestros scripts. Para ello, en la pantalla de bienvenida de VSCode hacemos clic en *Open Folder...* o en la opción de menú *File → Open Folder...* Nos ubicamos en `/home/usuario/dwes.com` y hacemos clic en *Aceptar*.

A partir de ahora, en el panel izquierdo veremos el contenido de `/home/usuario/dwes.com` y podremos editar los scripts PHP que formen nuestra aplicación web.

3.2 Git y GitHub

Como complemento a nuestro IDE vamos a emplear un sistema de control de versiones como Git y GitHub como proveedor de alojamiento de repositorios Git.

Un sistema de control de versiones (*VCS Versions Control System*) es un sistema que ayuda a rastrear y gestionar los cambios realizados en un archivo o conjunto de archivos, generalmente aquellos que forman un proyecto de desarrollo. Utilizado principalmente por ingenieros de software para hacer un seguimiento de las modificaciones realizadas en el código fuente, el sistema de control de versiones les permite analizar todos los cambios y revertirlos sin repercusiones en el caso de que se cometa un error.

Git es un proyecto de código abierto que se ha convertido en uno de los VCS más populares del mercado: cerca del 87% de los desarrolladores utilizan Git para sus proyectos. Se trata de un sistema de control de versiones distribuido. Esto significa que cualquier desarrollador del equipo que tenga acceso puede gestionar el código fuente y su historial

de cambios utilizando las herramientas de línea de comandos de Git.

Para instalar Git en nuestro sistema tenemos que ejecutar el siguiente comando apt de instalación en una ventana de terminal.

```
$ sudo apt install git
```

GitHub es el mayor proveedor de alojamiento de repositorios Git, y es el punto de encuentro para que millones de desarrolladores colaboren en el desarrollo de sus proyectos. Un gran porcentaje de los repositorios Git se almacenan en GitHub, y muchos proyectos de código abierto lo utilizan para hospedar su Git, realizar su seguimiento de fallos, hacer revisiones de código y otras cosas. Por tanto, aunque no sea parte directa del proyecto de código abierto de Git, es muy probable que durante un uso profesional de Git necesitemos interactuar con GitHub en algún momento.

Para poder utilizar GitHub necesitaremos una cuenta de GitHub. Si no tenemos una debemos registrarnos [aquí](#). Además, creamos un repositorio llamado `dwes`, que será con el que trabajemos.

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)


Required fields are marked with an asterisk (*).

Repository template

No template ▾

Start your repository with a template repository's contents.

Owner *

 rafagrancapitan ▾

Repository name *

php

✔ php is available.

Great repository names are short and memorable. Need inspiration? How about [turbo-engine](#) ?

Description (optional)

PHP learning



Public

Anyone on the internet can see this repository. You choose who can commit.



Private

You choose who can see and commit to this repository.

Initialize this repository with:



Add a README file

This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

.gitignore template: None ▾

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

License: None ▾

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)



You are creating a public repository in your personal account.

Create repository

Figura 5.- Repositorio en GitHub para PHP

Una vez tengamos una cuenta de GitHub tenemos que configurar VSCode para poder desarrollar un espacio de trabajo de GitHub directamente en VSCode.

Como hemos vistos en el punto anterior, nuestro directorio de trabajo es `/home/usuario/dwes.com` en el que guardaremos nuestros scripts PHP. Además, estos scripts los vamos a almacenar también en un repositorio de GitHub anterior. Para ello hacemos lo siguiente:

1. En VSCode hacer clic en el botón *Source Control* del panel izquierdo.
2. Desde aquí podemos hacer clic en *Publish to GitHub* para crear un nuevo repositorio en nuestra cuenta de GitHub directamente desde VSCode.

A partir de ahora, cuando realicemos cambios en nuestro código fuente podremos confirmarlos directamente en el repositorio creado en GitHub.

3.3 Documentación PHP

PHP tiene una extensa documentación en múltiples idiomas disponible para los desarrolladores en <https://www.php.net/manual/es/>. Además de referencias a los elementos del lenguaje y sus características específicas disponemos de ejemplos de aplicación.

3.4 Primer script PHP

Antes de empezar a programar en PHP debemos tener unas nociones básicas sobre la sintaxis de PHP. PHP está fuertemente influenciado por otros lenguajes de programación, como Perl o C. Si se tiene experiencia en estos lenguajes, el aprendizaje de PHP es fácil. Si no se tiene experiencia en programación, PHP es un buen comienzo debido a su sencillez. Vamos a comenzar con unas premisas básicas de lo que es un programa PHP como punto de partida para avanzar posteriormente en su aprendizaje.

- ✓ Todos el código fuente PHP se escriben en documentos de texto con extensión `.php` y juego de caracteres UTF8. Los scripts PHP pueden contener código HTML y PHP. El código PHP es incrustado en cualquier parte del documento delimitado por la marca de inicio `<?PHP` y la marca de fin `?>`. A esto se conoce bloque de código PHP o simplemente bloque PHP. Por ejemplo
- ✓ Dentro de un bloque PHP, el motor PHP está en modo PHP, fuera del bloque está en modo HTML. En modo PHP, todo se interpreta y ejecuta por el motor PHP, mientras que en modo HTML, el contenido del archivo se envía como respuesta al cliente tal cual.
- ✓ Para delimitar un bloque PHP podemos usar la notación corta con `<? ... ?>`. Sin embargo, esta forma es menos portable ya que el soporte para los delimitadores cortos está deshabilitado por defecto en `php.ini`.
- ✓ PHP no es sensible a la capitalización, excepto los nombres de las variables que si lo son.
- ✓ PHP emplea `;` para separar cada sentencia. Una sentencia compuesta delimitada por llaves no necesita `;` al final. Cada línea de código o sentencia debe terminar con `;` excepto la última del bloque PHP, que es opcional. De todas formas, se recomienda usar siempre `;` para delimitar las sentencias.
- ✓ Un comentario comienza por `//` o `#` si es de una línea. Desde la marca de inicio de comentario hasta el final de la línea se considera comentario. Si queremos utilizar comentarios con múltiples líneas hay que comenzarlos con `/*` y acabarlos con `*/`.

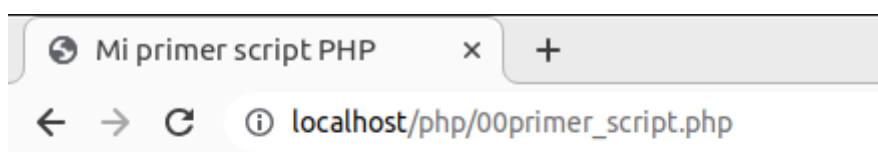
- ✓ Los espacios en blanco y saltos de línea se ignoran. Se puede distribuir una sentencia a lo largo de varias líneas o poner varias sentencias en una única línea. Esta flexibilidad nos permite hacer más legible el código.

Una vez establecidas las premisas anteriores, vamos a ver los principios básicos de la escritura y ejecución de scripts PHP con un sencillo programa que solamente visualiza un mensaje por pantalla. Abrimos el VSCode y creamos un archivo nuevo llamado `primer_script.php`. En este archivo escribimos el siguiente código:

```
<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="UTF-8">
    <title>Mi primer script PHP</title>
  </head>
  <body>
    <h1>Primer script en PHP</h1>
  <?PHP
    echo("Este es mi primer script en PHP<br>");
    echo("Este archivo es primer_script.php");
  ?>
  </body>
</html>
```

Como se puede apreciar es una página web HTML en el que se ha incluido un bloque PHP el cual está delimitado por `<?PHP` y `?>`. Sin embargo, para que el servidor web reenvíe la petición del cliente al módulo PHP el archivo debe guardarse con extensión `.php`.

Para ejecutar el script anterior tenemos que utilizar el navegador web y en la barra de dirección escribir http://dwes.com/primer_script.php. Esto provoca que el servidor web envíe el archivo `primer_script.php` al módulo PHP y este se encarga de ejecutar solamente los bloques PHP que encuentre dentro del archivo. Cuando lo haga devolverá al cliente el código HTML directamente sin procesar y el resultado de la ejecución de las sentencias dentro del bloque PHP. En el navegador veremos lo siguiente:



Primer script en PHP

Este es mi primer script en PHP
Este archivo es 00primer_script.php

Con este ejemplo hemos visto los principios básicos de la programación con PHP. A partir de ahora continuaremos profundizando en el uso de este lenguaje y sus enormes posibilidades para la programación web.

4 Elementos del lenguaje

En este epígrafe vamos a ver los elementos del lenguaje de PHP que son habituales en todos los lenguajes de programación. Concretamente:

- ✓ Tipos de datos y literales
- ✓ Comentarios
- ✓ Identificadores y variables
- ✓ Operadores y expresiones
- ✓ Instrucciones (secuencial, bifurcación condicional, repetitivas, ...)

Muchos de estos elementos necesitan para usarse una palabra reservada. Una palabra reservada es un vocablo del lenguaje con un uso específico y exclusivo, lo que significa que ningún identificador del programa puede llamarse igual que una palabra reservada. Siempre se escriben en minúscula.

Las palabras reservadas en PHP son las siguientes:

| | | | | |
|--------------------------------|---------------------------|-------------------------|-------------------------|-------------------------|
| <code>__halt_compiler()</code> | <code>abstract</code> | <code>and</code> | <code>array()</code> | <code>as</code> |
| <code>break</code> | <code>callable</code> | <code>case</code> | <code>catch</code> | <code>class</code> |
| <code>clone</code> | <code>const</code> | <code>continue</code> | <code>declare</code> | <code>default</code> |
| <code>die()</code> | <code>do</code> | <code>echo</code> | <code>else</code> | <code>elseif</code> |
| <code>empty()</code> | <code>enddeclare</code> | <code>endfor</code> | <code>endforeach</code> | <code>endif</code> |
| <code>endswitch</code> | <code>endwhile</code> | <code>eval()</code> | <code>exit()</code> | <code>extends</code> |
| <code>final</code> | <code>finally</code> | <code>fn</code> | <code>for</code> | <code>foreach</code> |
| <code>function</code> | <code>global</code> | <code>goto</code> | <code>if</code> | <code>implements</code> |
| <code>include</code> | <code>include_once</code> | <code>instanceof</code> | <code>insteadof</code> | <code>interface</code> |
| <code>isset()</code> | <code>list()</code> | <code>match</code> | <code>namespace</code> | <code>new</code> |
| <code>or</code> | <code>print</code> | <code>private</code> | <code>protected</code> | <code>public</code> |
| <code>require</code> | <code>require_once</code> | <code>return</code> | <code>static</code> | <code>switch</code> |
| <code>throw</code> | <code>trait</code> | <code>try</code> | <code>unset()</code> | <code>use</code> |
| <code>var</code> | <code>while</code> | <code>xor</code> | <code>yield</code> | <code>yield from</code> |

Conforme las vayamos avanzando en este y posteriores capítulos iremos viendo el significado y uso de las más habituales.

Aparte de las palabras clave, hay otros elementos de un programa PHP que necesita asignarse un nombre elegido por el programador: variables, parámetros, clases, interfaces, propiedades, ... En este caso el concepto que hace referencia a un nombre asignado arbitrariamente se conoce como **identificador**. Todos los identificadores, independientemente del tipo de elemento asignado, siguen unas reglas sintácticas que se verán posteriormente y que de no seguir provocaría un error.

4.1 Contenido mixto

Cualquier cosa fuera de un par de etiquetas de apertura y cierre es ignorado por el

intérprete de PHP, lo que permite que los ficheros de PHP tengan contenido mixto. Esto hace que PHP pueda ser incrustado en documentos HTML para, por ejemplo, crear plantillas.

```
<p>Esto va a ser ignorado por PHP y mostrado por el navegador.</p>
<?php echo 'Mientras que esto va a ser interpretado.'; ?>
<p>Esto también será ignorado por PHP y mostrado por el
navegador.</p>
```

Este ejemplo funciona como estaba previsto, porque cuando PHP intercepta las etiquetas de cierre `?>`, simplemente comienza a imprimir cualquier cosa que encuentre (a excepción de una nueva línea inmediatamente después del `;` véase la separación de sentencias) hasta que dé con otra etiqueta de apertura a menos que se encuentre en mitad de una sentencia condicional, en cuyo caso el intérprete determinará el resultado de la condición antes de tomar una decisión de qué es lo que tiene que saltar. Veamos el siguiente ejemplo.

```
<?php if ($expresion == true): ?>
Esto se mostrará si la expresión es verdadera.
<?php else: ?>
En caso contrario se mostrará esto.
<?php endif; ?>
```

En este ejemplo, PHP saltará los bloques donde la condición no se cumpla, incluso si están fuera de las etiquetas de apertura/cierre de PHP, los saltará según la condición debido a que el intérprete salta los bloques contenidos dentro de una condición que no se cumpla.

Para imprimir bloques de texto grandes, es más eficiente abandonar el modo intérprete de PHP que enviar todo el texto a través de `echo` o `print`.

4.2 Separación de instrucciones

Como en C o en Perl, PHP requiere que las instrucciones terminen en punto y coma al final de cada sentencia. La etiqueta de cierre de un bloque de código de PHP automáticamente implica un punto y coma; no es necesario usar un punto y coma para cerrar la última línea de un bloque de PHP. La etiqueta de cierre del bloque incluirá la nueva línea final inmediata si está presente.

```
<?php
    echo 'Esto es una prueba';
?>

<?php echo 'Esto es una prueba' ?>

<?php echo 'Hemos omitido la última etiqueta de cierre';
```

La etiqueta de cierre de un bloque de PHP es opcional al final de un fichero, y en algunos casos es útil omitirla.

4.3 Entrada y salida

En todos los lenguajes de programación hay instrucciones de entrada y salida. Las primeras permiten al usuario introducir datos por teclado, mientras que las segundas envían datos a la salida estándar, generalmente la pantalla. La entrada y salida son las instrucciones

más habituales en las aplicaciones ya que permiten la comunicación de la aplicación con el usuario. Debido a ello, son de las primeras en aprender cuando afrontamos un nuevo lenguaje de programación.

En las aplicaciones web el concepto de entrada y salida es algo diferente. Generalmente el usuario utiliza un navegador web para comunicarse con la aplicación web y su entrada de datos es a través de un formulario o de los enlaces. La salida es la respuesta del servidor web que generalmente es HTML y se visualiza en la ventana del navegador, y en el caso de ser otro tipo de dato se visualiza con una aplicación auxiliar (visor PDF, por ejemplo) o se almacena como un archivo.

En PHP el envío de datos a la salida es mediante dos formas:

- ✓ Contenido del script fuera de las etiquetas de apertura y cierre del bloque PHP. Esto lo hemos visto en el epígrafe anterior.
- ✓ Las funciones `echo` y `print`.

En el primer caso, todo contenido del archivo de script PHP que no está dentro de un bloque PHP se envía a la salida tal cual. Generalmente, aquí se pone código HTML para ser enviado directamente al navegador web.

Si dentro de un bloque PHP necesitamos enviar algún dato a la salida podemos usar la función `echo` o `print`.

4.3.1 echo

La función `echo` envía todos los parámetros a la salida. Su sintaxis es:

```
echo(string $arg1, string $... = ?): void
```

`echo` no es realmente una función, es una construcción del lenguaje, por lo que no se requiere el uso de paréntesis. A diferencia de otras construcciones del lenguaje, no se comporta como una función, es decir no siempre se puede usar en el contexto de una función. Además, si se quiere pasar más de un parámetro, éstos no deben estar entre paréntesis.

Veamos algunos ejemplos:

```
<?php
echo "Hola mundo";

echo "Esto abarca
multiple líneas. Los saltos de línea también
se mostrarán";

echo "Esto abarca\nmúltiples líneas. Los saltos de línea también\
nse mostrarán.";

echo "Para escapar caracteres se hace \"así\".";

// Se pueden usar variables dentro de una sentencia echo
$foo = "foobar";
```

```
$bar = "barbaz";  
  
echo "foo es $foo"; // foo es foobar  
?>
```

El uso de variables dentro de una cadena se verá más adelante en el epígrafe dedicado al análisis de variables.

Cuando queremos visualizar varios datos con `echo` podemos concatenarlos con el operador `.` o separarlos con coma. Un beneficio de pasar varios argumentos sobre la concatenación en `echo` es la precedencia del operador punto en PHP. Si se pasan varios argumentos, no serán necesarios paréntesis para forzar la precedencia:

```
<?php  
echo "Suma: ", 1 + 2;  
echo "Hola ", isset($name) ? $name : "John Doe", "!";  
?>
```

Con la concatenación, el operador punto tiene mayor precedencia que los operadores de adición y ternario, por lo que se deben utilizar paréntesis para un correcto funcionamiento:

```
<?php  
echo 'Suma: ' . (1 + 2);  
echo 'Hola ' . (isset($name) ? $name : 'John Doe') . '!';  
?>
```

`echo` también posee una sintaxis abreviada, donde se puede poner el símbolo igual justo después de la etiqueta de apertura de PHP. Por ejemplo:

```
<?php  
// Bloque PHP  
$edad = 10;  
?>  
Fuera del bloque PHP. Salida directa HTML:  
<p>Mi edad es <?= $edad>? y me considero bastante joven</p>
```

4.3.2 print

La función `print` muestra una cadena. Su sintaxis es:

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

Al igual que `echo`, `print` no es realmente una función, es un constructor de lenguaje, por lo tanto no es necesario usar paréntesis para indicar su lista de argumentos. Veamos algunos ejemplos.

```
<?php  
print("Hola mundo");  
  
print "print() también funciona sin paréntesis."  
  
print "Esto separa  
múltiples líneas. Los saltos de línea también  
se mostrarán";
```

```
print "Esto separa\nmúltiples líneas. Los salos de línea también\nse mostrarán.";\n\nprint "para escapar caracteres se \"hace así\".";\n\n// También se puede usar variables usando print\n$foo = "foobar";\n$bar = "barbaz";\n\nprint "foo es $foo"; // foo es foobar\n?>
```

4.4 Comentarios

Una costumbre muy sana en programación es comentar el código. Añadir líneas de comentarios a bloques de código o una nota aclaratoria en una línea concreta viene muy bien para clarificar su propósito. En PHP disponemos de los comentarios de una sola línea y de múltiples líneas.

PHP admite comentarios al estilo de 'C', 'C++' y de consola de Unix (estilo de Perl). Por ejemplo:

```
<?php\n    echo 'Esto es una prueba'; // Esto es un comentario al estilo\nde c++ de una sola línea\n    /* Esto es un comentario multilínea\n       y otra línea de comentarios */\n    echo 'Esto es otra prueba';\n    echo 'Una prueba final'; # Esto es un comentario al estilo de\nconsola de una sola línea\n?>
```

Los comentarios de una sola línea solo comentan hasta el final de la línea o del bloque actual de código de PHP, lo primero que suceda.

Los comentarios al estilo de 'C' finalizan con el primer `*/` que se encuentre y no pueden anidarse.

4.5 El archivo de configuración `php.ini`

Entre todos los archivos que forma parte de la instalación del lenguaje PHP, hay uno que destaca sobre el resto debido a su importancia a la hora de trabajar con este lenguaje. Se trata del archivo `php.ini`, un fichero que contiene la configuración de PHP que vamos a utilizar en el servidor donde alojemos nuestro portal.

El archivo `PHP.ini` es un archivo de configuración que contiene las opciones de PHP que controlan aspectos de su funcionamiento en el servidor web. Cada vez que se inicia PHP, el sistema lo buscará y aplicará la configuración que contiene y que regirán la ejecución de scripts PHP en el servidor. El archivo incluye una configuración estándar de PHP por defecto, aunque es posible necesitar cambiar la configuración predeterminada de PHP para adaptarla a nuestras necesidades.

Este archivo lo solemos encontrar dentro del directorio donde hemos realizado la

instalación de la distribución PHP. El resultado de la función `phpinfo()` muestra la ubicación del archivo en *Loaded Configuration File*. Se trata de un archivo de texto que puede ser editado con cualquier editor de texto plano que tengamos instalado en nuestro equipo.

El archivo contiene un conjunto de directivas cada una con un valor que regula el funcionamiento de algún aspecto de la ejecución de los scripts PHP. La mayoría de las directivas tienen un valor por defecto, por lo que si no se encuentran en el archivo, se aplicaría este valor por defecto.

Su sintaxis es:

- ✓ Líneas que comienzan por punto y coma → Son comentarios explicando el funcionamiento de ciertas directivas, siendo todas ellas ignoradas. Se puede dar el caso de que tengamos directivas con un punto y coma al principio. Esto quiere decir que están comentadas y que no son aplicadas por PHP. Si las quisiéramos utilizar, deberíamos eliminar el punto y coma del principio.
- ✓ Texto marcado con corchetes → Por ejemplo `[PHP]` que indican la cabecera de una determinada sección.
- ✓ Directivas → Es la parte más importante de este archivo y están formadas por una pareja compuesta por una clave y su determinado valor con el formato `directiva = valor`. Las directivas son sensibles a la capitalización.

Los valores que asignamos a las directivas pueden ser:

- ✓ Una cadena.
- ✓ Un valor `On` u `Off`.
- ✓ Un número.
- ✓ Una constante PHP.
- ✓ Una constante INI.
- ✓ Una expresión. Las expresiones en el archivo INI se limitan a operadores bit a bit o paréntesis.
- ✓ Una cadena entre comillas
- ✓ Una referencia a una variable establecida previamente.

Podemos establecer configuraciones en el archivo INI de las siguientes formas:

- ✓ En el propio archivo INI.
- ✓ Cuando se usa PHP como un módulo del servidor web Apache, se pueden cambiar los ajustes de configuración usando directivas en los ficheros de configuración de Apache.
- ✓ En tiempo de ejecución mediante la función `ini_set()`.

Si cambiamos el valor de una directiva en el archivo INI o en la configuración de Apache tendremos que reiniciar el servidor para que este cambio tenga efecto y afectará a la ejecución de todos los scripts. Esto podría tener efectos no deseados si en ese momento se están ejecutando scripts PHP. Los cambios realizados desde los propios script PHP con la función `ini_set()` tendrán efecto inmediatamente y solo afectan a la ejecución del script.

En <https://www.php.net/manual/es/ini.list.php> disponemos de un listado de las directivas disponibles en el que también se incluye información como:

- ✓ Nombre de la directiva.
- ✓ Valor por defecto.
- ✓ Modo de cambio.

Además, en el propio archivo INI, antes de cada directiva hay unas líneas de comentario explicando que función desempeña la directiva y los posibles valores que puede tener. Algunas de las directivas más habituales son:

- ✓ `max_execution_time`. Tiempo máximo que un script podrá estar ejecutándose antes de que sea cancelado. El tiempo es indicado en segundos.
- ✓ `memory_limit`. Memoria que se podrá utilizar para la ejecución del código PHP.
- ✓ `post_max_size`. Se trata de la directiva que permite controlar el tamaño de los datos enviados mediante POST, afectando también a los archivos que se subirán al servidor. Si el archivo supera el límite indicado, dará un error y no se subirá.
- ✓ `upload_max_filesize`. Tamaño máximo de ficheros que se puede subir a la web de forma simultánea. Se puede subir uno o más de uno siempre y cuando no se supere el valor indicado aquí.
- ✓ `default_charset`. En esta directiva se le puede indicar el tipo de codificación que se utilizará por defecto, siendo el de por defecto UTF-8.
- ✓ `include_path`. Especifica la lista de directorios donde las funciones `require`, `include`, `fopen()`, `file()`, `readfile()` y `file_get_contents()` buscarán ficheros. Permite una lista de directorios separados por punto y coma.
- ✓ `file_uploads`. Si permite o no la subida de archivos vía HTTP.
- ✓ `upload_tmp_dir`. Aquí indicaremos el directorio temporal utilizado durante el proceso de subida de los archivos. Si no se especifica, PHP utilizará el de por defecto del sistema.
- ✓ `max_file_uploads`. El máximo número de archivos que se podrán subir de forma simultánea.
- ✓ `display_errors`. Indica si se envían a la salida o no los mensajes de error.
- ✓ `error_reporting`. Indica los tipos de mensajes de error que se enviarán a la salida.

- ✓ `short_open_tag`. Indica si se admite `<?` en lugar de `<?php` para iniciar el modo PHP.

5 Tipos de datos

PHP admite diez tipos de datos primitivos.

Cuatro tipos escalares:

- ✓ `boolean`
- ✓ `integer`
- ✓ `float` (número de punto flotante, también conocido como `double`)
- ✓ `string`

Cuatro tipos compuestos:

- ✓ `array`
- ✓ `object`
- ✓ `callable`
- ✓ `iterable`

Y finalmente dos tipos especiales:

- ✓ `resource`
- ✓ `NULL`

El tipo de una variable usualmente no lo declara el programador, es decidido en tiempo de ejecución por PHP dependiendo del contexto en el que se emplea dicha variable.

Para comprobar el tipo y el valor de una expresión, usamos la función `var_dump()`. Para obtener una representación legible de un tipo de datos con propósitos de depuración, usamos la función `gettype()`. Para comprobar un cierto tipo, utilizamos las funciones `is_tipo`. Por ejemplo:

```
<?php
$un_bool = TRUE;    // un valor booleano
$un_str  = "foo";   // una cadena de caracteres
$un_str2 = 'foo';   // una cadena de caracteres
$un_int  = 12;      // un número entero

echo gettype($un_bool); // imprime: boolean
echo gettype($un_str);  // imprime: string

// Si este valor es un entero, incrementarlo en cuatro
if (is_int($un_int)) {
    $un_int += 4;
}

// Si $un_bool es una cadena, imprimirla
```

```
// (no imprime nada)
if (is_string($un_bool)) {
    echo "Cadena: $un_bool";
}
?>
```

Para forzar la conversión de una variable a un cierto tipo, podemos amoldar la variable o usar la función `settype()` con ella.

En los siguientes apartados nos centraremos en los tipos escalares y dejaremos para capítulos posteriores los tipos compuestos. Los tipos escalares son `boolean`, `integer`, `float` y `string`. Una variable de tipo primitivo puede almacenar sólo un valor de su tipo declarado a la vez. Por ejemplo, una variable `integer` puede almacenar un número entero completo a la vez. Cuando se asigna otro valor a esa variable, se sustituye su valor inicial.

5.1 Boolean

Este es el tipo más simple. Un `boolean` expresa un valor que indica verdad. Puede ser `true` (verdadero) o `false` (falso). Para especificar un literal de tipo `boolean` se emplean las constantes `true` o `false` no sensibles a la capitalización.

```
<?php
$foo = True; // asigna el valor TRUE a $foo
?>
```

Usualmente, el resultado de un operador que devuelve un valor de tipo `boolean` es pasado a una estructura de control.

```
<?php
// == es un operador que comprueba la
// igualdad y devuelve un booleano
if ($accion == "mostrar_version") {
    echo "La versión es 1.23";
}

// esto no es necesario...
if ($mostrar_separadores == TRUE) {
    echo "<hr>\n";
}

// ...porque se puede escribir simplemente:
if ($mostrar_separadores) {
    echo "<hr>\n";
}
?>
```

Para convertir explícitamente un valor al tipo `boolean`, usamos un *casting* (`bool`) o (`boolean`). Sin embargo, en la mayoría de casos es innecesario, ya que un valor será convertido automáticamente si un operador, función o estructura de control requiere un argumento de tipo `boolean`.

Cuando se realizan conversiones a `boolean`, los siguientes valores se consideran `false`:

- ✓ el boolean `false` mismo
- ✓ el integer `0` y `-0`
- ✓ el float `0.0` y `-0.0`
- ✓ el valor string vacío, y el string `"0"`
- ✓ un array con cero elementos
- ✓ el tipo especial `NULL` (incluidas variables no establecidas)

Cualquier otro valor se considera como `true` (incluido cualquier `resource` y `NAN`).

5.2 Numéricos enteros `integer`

Un número entero es un número del conjunto $\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$.

Los `integer` pueden especificarse mediante notación decimal (base 10), hexadecimal (base 16), octal (base 8) o binaria (base 2), opcionalmente precedidos por un signo (- o +).

Para utilizar la notación octal, se antepone al número un `0` (cero). Para utilizar la notación hexadecimal, se antepone al número `0x`. Para utilizar la notación binaria, se antepone al número `0b`.

```
<?php
$a = 1234; // número decimal
$a = -123; // un número negativo
$a = 0123; // número octal (equivale a 83 decimal)
$a = 0x1A; // número hexadecimal (equivale a 26 decimal)
$a = 0b11111111; // número binario (equivale al 255 decimal)
?>
```

El tamaño de un `integer` depende de la plataforma, aunque el valor usual es un valor máximo de aproximadamente dos mil millones (esto es, 32 bits con signo). Las plataformas de 64 bits normalmente tienen un valor máximo de aproximadamente $9E18$. PHP no tiene soporte para el tipo `integer` sin signo. El tamaño de un `integer` se puede determinar mediante la constante `PHP_INT_SIZE`, el valor máximo mediante la constante `PHP_INT_MAX` y el valor mínimo mediante la constante `PHP_INT_MIN`.

Para convertir explícitamente un valor al tipo `integer` se puede emplear tanto `(int)` como `(integer)`. Sin embargo, la mayoría de las veces la conversión no es necesaria, ya que un valor es convertido de forma automática cuando un operador, función o estructura de control requiera un argumento de tipo `integer`. Un valor también puede ser convertido al tipo `integer` mediante la función `intval()`.

La conversión a `integer` es:

- ✓ Desde datos booleanos es `false` producirá `0` (cero), y `true` producirá `1` (uno).
- ✓ Desde datos `float` el número será redondeado hacia cero.
- ✓ Desde `string`, si la cadena de caracteres no contiene ninguno de los caracteres `!`, `'e'`, o `'E'`, y el valor numérico está entre los límites del tipo `integer`.

5.3 Números en punto flotante

Los números de punto flotante, o de coma flotante, pueden ser especificados usando cualquiera de las siguientes sintaxis:

```
<?php
$a = 1.234;
$b = 1.2e3;
$c = 7E-10;
$d = 1_234.567; // a partir de PHP 7.4.0
?>
```

El tamaño de un `float` depende de la plataforma, aunque un valor común consiste en un máximo de aproximadamente $1.8e308$ con una precisión cercana a los 14 dígitos decimales (el formato de 64 bit del IEEE).

Para conversión a tipo `float` se siguen las siguientes reglas:

- ✓ Desde `string`, si la cadena de caracteres contiene alguno de los caracteres `.`, `'e'`, o `'E'`, o el valor numérico está fuera de los límites del tipo `integer`.
- ✓ Para valores de otros tipos, la conversión es la misma que si el valor hubiese sido convertido primero a `integer` y luego a `float`.

5.4 Cadena de caracteres o String

Un `string`, o cadena, es una serie de caracteres donde cada carácter es lo mismo que un `byte`. Esto significa que PHP solo admite un conjunto de 256 caracteres, y de ahí que no ofrezca soporte nativo para Unicode. Un literal de tipo `string` se puede especificar de cuatro formas diferentes:

5.4.1 Comillas simples

La manera más sencilla de especificar un `string` es delimitarlo con comillas simples (el carácter `'`). Para especificar una comilla simple literal, se ha de escapar con una barra invertida (`\`). Para especificar una barra invertida literal, se duplica (`\\`). Todas las demás instancias de barras invertidas serán tratadas como una barra invertida literal: esto significa que otras secuencias de escape que podrían utilizarse, tales como `\r` o `\n`, serán mostradas literalmente tal y como se especifican, en lugar de tener cualquier otro significado especial.

```
<?php
echo 'Esto es una cadena sencilla';

echo 'También se pueden incluir nuevas líneas en
un string de esta forma, ya que es
correcto hacerlo así';

// Resultado: Arnold una vez dijo: "I'll be back"
echo 'Arnold una vez dijo: "I\'ll be back"';

// Resultado: Ha borrado C:\*..*?
echo 'Ha borrado C:\\*..*?';
```

```
// Resultado: Ha borrado C:\*.*?
echo 'Ha borrado C:\*.*?';

// Resultado: Esto no se expandirá: \n una nueva línea
echo 'Esto no se expandirá: \n una nueva línea';

// Resultado: Las variables $stampoco se $expandirán
echo 'Las variables $stampoco se $expandirán';
?>
```

5.4.2 Comillas dobles

Si un `string` está delimitado con comillas dobles (`"`), PHP interpretará las siguientes secuencias de escape como caracteres especiales:

| Secuencia | Significado |
|---------------------------------|---|
| <code>\n</code> | avance de línea (LF o 0x0A (10) en ASCII) |
| <code>\r</code> | retorno de carro (CR o 0x0D (13) en ASCII) |
| <code>\t</code> | tabulador horizontal (HT o 0x09 (9) en ASCII) |
| <code>\v</code> | tabulador vertical (VT o 0x0B (11) en ASCII) (desde PHP 5.2.5) |
| <code>\e</code> | escape (ESC o 0x1B (27) en ASCII) (desde PHP 5.4.4) |
| <code>\f</code> | avance de página (FF o 0x0C (12) en ASCII) (desde PHP 5.2.5) |
| <code>\\</code> | barra invertida |
| <code>\\$</code> | signo de dólar |
| <code>\"</code> | comillas dobles |
| <code>\[0-7]{1,3}</code> | la secuencia de caracteres que coincida con la expresión regular es un carácter en notación octal, que silenciosamente desborda para encajar en un byte (p.ej. <code>"\400" === "\000"</code>) |
| <code>\x[0-9A-Fa-f]{1,2}</code> | la secuencia de caracteres que coincida con la expresión regular es un carácter en notación hexadecimal |
| <code>\u{[0-9A-Fa-f]+}</code> | la secuencia de caracteres que coincida con la expresión regular es un punto de código de Unicode, la cual será imprimida al <code>string</code> como dicha representación UTF-8 del punto de código (añadido en PHP 7.0.0) |

Al igual que en el entrecomillado simple de un `string`, escapar cualquier otro carácter puede dar lugar a que se muestre también la barra invertida. La característica más importante del entrecomillado doble de un `string` es el hecho de que se expanden los nombres de las variables. Se verá esto más adelante.

5.4.3 Heredoc

Una tercera forma de delimitar un `string` es mediante la sintaxis heredoc: `<<<`. Después de este operador, se deberá proporcionar un identificador y justo después una nueva línea. A continuación va el propio `string`, y para cerrar la notación se pone el mismo identificador.

El identificador de cierre debe empezar en la primera columna de la nueva línea. Asimismo, el identificador debe seguir las mismas reglas de nomenclatura de las etiquetas

en PHP: debe contener solo caracteres alfanuméricos y guiones bajos, y debe empezar con un carácter alfabético o un guión bajo.

```
<?php
$str = <<<EOD
Ejemplo de una cadena
expandida en varias líneas
empleando la sintaxis heredoc.
EOD;
?>
```

5.4.4 Nowdoc

Nowdoc es a los `string` con comillas simples lo mismo que Heredoc lo es a los `string` con comillas dobles. Un nowdoc se especifica de forma análoga a un heredoc, pero no se realiza ningún análisis dentro del nowdoc. La construcción es ideal para incrustar código de PHP o grandes fragmentos de texto sin necesidad de escaparlos.

Un nowdoc se identifica con la misma secuencia empleada para heredoc, `<<<`, pero el identificador que le sigue está delimitado con comillas simples, p.ej., `<<<'EOT'`. Todas las reglas para los identificadores de heredoc también son aplicables a los identificadores de nowdoc, especialmente aquellas que se refieren al empleo del identificador de cierre.

```
<?php
$str = <<<'EOD'
Ejemplo de un string
expandido en varias líneas
empleando la sintaxis nowdoc.
EOD;
?>
```

5.4.5 Conversión a string

Un valor puede convertirse a un `string` empleando el molde `(string)` o mediante la función `strval()`. La conversión automática a `string` tiene lugar en el ámbito de una expresión donde sea necesario un `string`. Esto ocurre cuando se utilizan las funciones `echo` o `print`, o cuando se compara una variable con un `string`.

El valor `true` del tipo `boolean` es convertido al `string` `"1"`. El valor `false` del tipo `boolean` es convertido al `string` `""` (el `string` vacío). Esto permite la conversión en ambos sentidos entre los valores de los tipos `boolean` y `string`.

Un `integer` o `float` es convertido a un `string` que representa textualmente el número (incluyendo la parte exponencial para los `float`. Los números de punto flotante pueden ser convertidos mediante la notación exponencial (`4.1E+6`).

`null` siempre es convertido a un `string` vacío.

6 Variables

Una variable es un espacio de memoria que contiene un valor y que se referencia por un nombre. Por tanto una variable tiene un tamaño, un dato y una dirección de memoria para

referenciarla. Las variables se emplean para almacenar datos que el programa procesa. El tamaño de la variable es la cantidad de memoria necesaria para almacenar su valor y es función del tipo de datos del valor que almacena. Este valor puede cambiar a lo largo de la ejecución del programa.

En PHP las variables se representan con un signo de dólar seguido por el nombre de la variable. El nombre de la variable es sensible a minúsculas y mayúsculas.

Los nombres de variables siguen las mismas reglas que otras etiquetas en PHP. Un nombre de variable válido tiene que empezar con una letra o un carácter de subrayado, seguido de cualquier número de letras, números y caracteres de subrayado.

```
<?php
$var = 'Roberto';
$Var = 'Juan';
echo "$var, $Var";           // imprime "Roberto, Juan"

$4site = 'aun no';          // inválido; comienza con un número
$_4site = 'aun no';         // válido; comienza con un carácter de
subrayado
?>
```

De forma predeterminada, las variables siempre se asignan por valor. Esto significa que cuando se asigna una expresión a una variable, el valor completo de la expresión original se copia en la variable de destino. Esto quiere decir que, por ejemplo, después de asignar el valor de una variable a otra, los cambios que se efectúen a una de esas variables no afectará a la otra.

No es necesario inicializar variables en PHP, sin embargo, es una muy buena práctica. Las variables no inicializadas tienen un valor predeterminado de acuerdo a su tipo dependiendo del contexto en el que son usadas. En principio una variable no inicializada tiene el valor `null`, salvo las booleanas que es `false`. Sin embargo, si la variable sin inicializar está como operando en una expresión, entonces las booleanas se asumen como `false`, los enteros y flotantes como `0`, las cadenas se establecen como una cadena vacía y los arrays se convierten en un array vacío.

6.1 Variables predefinidas

PHP proporciona una gran cantidad de variables predefinidas a cualquier script que se ejecute. Muchas de éstas, sin embargo, no pueden ser completamente documentadas ya que dependen del servidor que esté ejecutándose, la versión, configuración de dicho servidor, y otros factores. Algunas de estas variables no estarán disponibles cuando se ejecute PHP desde la línea de comandos.

La lista de variables predefinidas son:

- ✓ `$GLOBALS` — Hace referencia a todas las variables disponibles en el ámbito global
- ✓ `$_SERVER` — Información del entorno del servidor y de ejecución. Es un array que contiene información, tales como cabeceras, rutas y ubicaciones de script. Las entradas de este array son creadas por el servidor web. No hay garantía que cada

servidor web proporcione alguna de estas entradas, existen servidores que pueden omitir algunas o proporcionar otras. Podemos ver todas ellas en <https://www.php.net/manual/es/reserved.variables.server.php>.

- ✓ `$_GET` — Variables HTTP GET. Un array asociativo de variables pasado al script actual vía parámetros URL (también conocida como cadena de consulta). Tener en cuenta que el array no solo se rellena para las solicitudes GET, sino para todas las solicitudes con una cadena de consulta.
- ✓ `$_POST` — Variables POST de HTTP. Un array asociativo de variables pasadas al script actual a través del método POST de HTTP cuando se emplea `application/x-www-form-urlencoded` o `multipart/form-data` como `Content-Type` de HTTP en la petición.
- ✓ `$_FILES` — Variables de subida de ficheros HTTP. Un array asociativo de elementos subidos al script en curso a través del método POST.
- ✓ `$_REQUEST` — Variables HTTP Request. Un array asociativo que por defecto contiene el contenido de `$_GET`, `$_POST` y `$_COOKIE`.
- ✓ `$_SESSION` — Variables de sesión. Es un array asociativo que contiene variables de sesión disponibles para el script actual.
- ✓ `$_ENV` — Variables de entorno. Una variable tipo array asociativo de variables pasadas al script actual a través del intérprete. Estas variables son importadas en el espacio de nombres global de PHP desde el entorno bajo el que está siendo ejecutado el intérprete PHP. Muchas son entregadas por el intérprete de comandos bajo el que PHP está ejecutándose y diferentes sistemas suelen tener diferentes tipos de intérpretes de comandos.
- ✓ `$_COOKIE` — Cookies HTTP. Una variable tipo array asociativo de variables pasadas al script actual a través de Cookies HTTP.
- ✓ `$argc` — El número de argumentos pasados a un script. Contiene el número de argumentos pasados al script actual cuando se ejecuta desde la línea de comandos. El nombre del script es pasado siempre como argumento del script, por lo tanto, el valor mínimo de `$argc` es 1.
- ✓ `$argv` — Array de argumentos pasados a un script. Contiene un array de todos los argumentos pasados a un script cuando se ejecuta desde la línea de comandos. El primer argumento `$argv[0]` siempre es el nombre del fichero que fue usado para ejecutar el script.

6.2 Análisis de variables

Cuando un `string` es especificado mediante comillas dobles o mediante heredoc, las variables que haya dentro de dicho `string` se analizarán. Existen dos tipos de sintaxis: una simple y otra compleja. La sintaxis simple es la más empleada y práctica. Proporciona una forma de embeber una variable, un valor de un array o una propiedad de un object dentro de un `string` con el mínimo esfuerzo.

La sintaxis compleja puede ser reconocida por las llaves que delimitan la expresión.

6.2.1 Sintaxis simple

Si se encuentra un signo de dólar (\$), el analizador tomará el mayor número de símbolos para formar un nombre de variable válido. Delimitar el nombre de la variable con llaves permite especificar explícitamente el final del nombre. Por ejemplo.

```
<?php
$zumo = "manzana";

echo "Él tomó algo de zumo de $zumo.". PHP_EOL;
// Inválido. "s" es un carácter válido para un nombre de
// variable, pero la variable es $jugo.
echo "Él tomó algo de zumo hecho de $zumos.";
// Válido. Explícitamente especifica el final del nombre de la
// variable encerrándolo entre llaves:
echo "Él tomó algo de zumo hecho de ${zumo}s."
?>
```

El resultado del ejemplo sería:

```
Él tomó algo de zumo de manzana.
Él tomó algo de zumo hecho de .
Él tomó algo de zumo hecho de manzanas.
```

De forma parecida, se puede analizar el índice de un array o la propiedad de un object. Con los índices de los arrays, el corchete de cierre marca el final del índice. La misma regla se puede aplicar a las propiedades de los objetos y a las variables simples.

```
<?php
$zumos = array("manzana", "naranja", "koolaid1" => "púrpura");

echo "Él tomó algo de zumo de $zumos[0].".PHP_EOL;
echo "Él tomó algo de zumo de $zumos[1].".PHP_EOL;
echo "Él tomó algo de zumo $zumos[koolaid1].".PHP_EOL;

class persona {
    public $john = "John Smith";
    public $jane = "Jane Smith";
    public $robert = "Robert Paulsen";

    public $smith = "Smith";
}

$persona = new persona();

echo "$persona->john tomó algo de zumo de $zumos[0].".PHP_EOL;
echo "$persona->john entonces dijo hola a $persona-
>jane.".PHP_EOL;
echo "La esposa de $persona->john saludó a $persona-
>robert.".PHP_EOL;
echo "$persona->robert saludó a los dos $persona->smiths."; // No
funcionará
?>
```

El resultado del ejemplo sería:

```
Él tomó algo de zumo de manzana.  
Él tomó algo de zumo de naranja.  
Él tomó algo de zumo púrpura.  
John Smith tomó algo de zumo de manzana.  
John Smith entonces dijo hola a Jane Smith.  
La esposa de John Smith saludó a Robert Paulsen.  
Robert Paulsen saludó a los dos .
```

6.2.2 Sintaxis compleja

Esta sintaxis se llama compleja porque permite el empleo de expresiones complejas. Cualquier variable escalar, elemento de array o propiedad de objeto con una representación de tipo `string` puede ser incluido a través de esta sintaxis. Simplemente se escribe la expresión del mismo modo en que aparecería por fuera del `string`, y delimitándola con `{` y `}`. Dado que `{` no puede ser escapado, esta sintaxis será reconocida únicamente cuando el `$` siga inmediatamente al `{`. Utilice `{\ $}` para obtener un `{ $` literal. Algunos ejemplos para que quede más claro:

```
<?php  
// Mostrar todos los errores  
error_reporting(E_ALL);  
  
$genial = 'fantástico';  
  
// No funciona, muestra: Esto es { fantástico}  
echo "Esto es { $genial}";  
  
// Funciona, muestra: Esto es fantástico  
echo "Esto es {$genial}";  
  
// Funciona  
echo "Este cuadrado tiene {$cuadrado->width}00 centímetros de  
lado.";  
  
// Funciona, las claves entre comillas sólo funcionan usando la  
sintaxis de llaves  
echo "Esto funciona: {$arr['clave']}";  
  
// Funciona  
echo "Esto funciona: {$arr[4][3]}";  
  
// Esto no funciona por la misma razón que $foo[bar] es  
incorrecto fuera de un string.  
// En otras palabras, aún funcionaría, pero solamente porque PHP  
primero busca una  
// constante llamada foo; se emitirá un error de nivel E_NOTICE  
// (constante no definida).  
echo "Esto está mal: {$arr[foo][3]}";  
  
// Funciona. Cuando se usan arrays multidimensionales, emplee  
siempre llaves que delimiten
```

```
// a los arrays cuando se encuentre dentro de un string
echo "Esto funciona: {$arr['foo'][3]}";

// Funciona.
echo "Esto funciona: " . $arr['foo'][3];

echo "Esto también funciona: {$obj->valores[3]->nombre}";

echo "Este es el valor de la variable llamada $nombre: {$$nombre}";

echo "Este es el valor de la variable llamada por el valor devuelto por getNombre(): {$getNombre()}";

echo "Este es el valor de la variable llamada por el valor devuelto por \$objeto->getNombre(): {$$objeto->getNombre()}";

//No funciona, muestra: Esto es el valor devuelto por getNombre(): {getNombre()}
echo "Esto es el valor devuelto por getNombre(): {getNombre()}";
?>
```

6.3 Funciones de manejo de variables

PHP dispone de un conjunto de funciones para gestionar las variables. Entre las más destacadas están:

- ✓ `empty` → Determina si una variable está vacía. Los siguientes valores son considerados como vacíos: "" (una cadena vacía), 0 (0 como un integer), 0.0 (0 como un float), "0" (0 como un string), `null`, `false` y un array vacío.
- ✓ `gettype` → Obtener el tipo de una variable.
- ✓ `is_array` → Comprueba si una variable es un array.
- ✓ `is_bool` → Comprueba si una variable es de tipo booleano.
- ✓ `is_int` → Comprueba si el tipo de una variable es integer.
- ✓ `is_float` → Comprueba si el tipo de una variable es float.
- ✓ `is_string` → Comprueba si una variable es de tipo string.
- ✓ `isset` → Determina si una variable está definida y no es `null`.
- ✓ `unset` → Destruye una o más variables especificadas.
- ✓ `var_dump` → Muestra información sobre una variable.

En <https://www.php.net/manual/es/ref.var.php> tenemos una referencia completa de estas y otras funciones de gestión de variables que nos pueden resultar útiles en nuestras aplicaciones.

6.4 Constantes

Una constante es un identificador para un valor simple. Como el nombre sugiere, este valor no puede variar durante la ejecución del script, a excepción de las constantes mágicas, que en realidad no son constantes. Por defecto, una constante distingue mayúsculas y minúsculas. Por convención, los identificadores de constantes siempre se declaran en mayúsculas. El nombre de una constante sigue las mismas reglas que cualquier otro identificador de PHP.

Se puede definir una constante usando la función `define()` o con la palabra reservada `const`. Mientras que `define()` permite definir una constante con una expresión arbitraria, la palabra reservada `const` tiene restricciones. Una vez que la constante está definida, no puede ser cambiada o redefinida.

Para obtener el valor de una constante solo es necesario especificar su nombre. A diferencia de las variables, no se debe prefijar una constante con el signo `$`. Por ejemplo:

```
<?php
define("CONSTANTE", "Hola mundo.");
echo CONSTANTE; // muestra "Hola mundo."
echo Constante; // muestra "Constant" y se emite un aviso.
?>
```

En el siguiente ejemplo usamos la palabra reservada `const`.

```
<?php
const CONSTANTE = 'Hola Mundo';
echo CONSTANTE;

// Funciona a partir de PHP 5.6.0
const OTRA_CONSTANTE = CONSTANTE. '; Adiós Mundo';
echo OTRA_CONSTANTE;

const ANIMALES = array('perro', 'gato', 'pájaro');
echo ANIMALES[1]; // muestra "gato"

define('ANIMALES', array(
    'perro',
    'gato',
    'pájaro'
));
echo ANIMALES[1]; // muestra "gato"
?>
```

A diferencia de definir constantes usando `define()`, las constantes definidas con la palabra clave `const` deben declararse en el nivel superior del entorno de la aplicación porque se definen en tiempo de ejecución. Esto significa que no pueden declararse dentro de funciones, bucles, sentencias `if` o bloques `try/catch`.

PHP ofrece un gran número de constantes predefinidas a cualquier script en ejecución. Muchas de estas constantes, sin embargo, son creadas por diferentes extensiones, y sólo estarán presentes si dichas extensiones están disponibles, bien por carga dinámica o porque han sido compiladas. Se puede ver una referencia de las mismas en

<https://www.php.net/manual/es/reserved.constants.php>.

Hay nueve constantes mágicas que cambian dependiendo de dónde se emplean. Por ejemplo, el valor de `__LINE__` depende de la línea en que se use en el script. Todas estas constantes «mágicas» se resuelven durante la compilación, a diferencia de las constantes normales que lo hacen durante la ejecución. Estas constantes especiales son sensibles a mayúsculas. Estas constantes especiales distinguen mayúsculas y minúsculas, y son las siguientes:

| Nombre | Descripción |
|-------------------------------|--|
| <code>__LINE__</code> | El número de línea actual en el fichero |
| <code>__FILE__</code> | Ruta completa y nombre del fichero con enlaces simbólicos resueltos. Si se usa dentro de un <code>include</code> , devolverá el nombre del fichero incluido. |
| <code>__DIR__</code> | Directorio del fichero. Si se utiliza dentro de un <code>include</code> , devolverá el directorio del fichero incluido. Esta constante es igual que <code>dirname(__FILE__)</code> . El nombre del directorio no lleva la barra final a no ser que esté en el directorio root. |
| <code>__FUNCTION__</code> | Nombre de la función. |
| <code>__CLASS__</code> | Nombre de la clase. El nombre de la clase incluye el namespace declarado en (p.e.j. <code>Foo\Bar</code>). |
| <code>__TRAIT__</code> | El nombre del trait. El nombre del trait incluye el espacio de nombres en el que fue declarado (p.e.j. <code>Foo\Bar</code>). |
| <code>__METHOD__</code> | Nombre del método de la clase. |
| <code>__NAMESPACE__</code> | Nombre del espacio de nombres actual. |
| <code>ClassName::class</code> | El nombre de clase completamente cualificado. Véase también <code>::class</code> . |

7 Operadores y expresiones

Un **operador** es un símbolo formado generalmente por uno, dos o tres caracteres que indica una operación a realizar. Para realizar la operación se necesitan datos u **operandos**. Los operadores pueden ser:

- ✓ Unario → Solamente necesitan un operando para realizar la operación.
- ✓ Binario → Emplean dos operandos para realizar la operación.

Los operandos pueden ser:

- ✓ Un valor literal.
- ✓ Una variable.
- ✓ El valor de devolución de una función (o método).
- ✓ Un valor resultado de otra operación.

De éste último tipo podemos inferir que las operaciones establecidas por un operador y uno o dos operandos pueden combinarse formando una expresión. Una **expresión** es una combinación de operadores y operandos que, una vez realizadas todas las operaciones indicadas por cada operador, arroja un resultado.

La realización de todas las operaciones indicadas en una expresión se denomina evaluar la expresión la cual siempre da como resultado un valor con un tipo de datos. Obviamente las operaciones de una expresión no se ejecutan simultáneamente, sino que hay un orden, es decir, primero se realiza una operación (indicada por su operador) y el resultado se emplea como operando en otra operación. Así, sucesivamente sin límite. El orden de ejecución de las operaciones está establecido por las reglas de precedencia y asociatividad de los operadores, las cuales indican qué operaciones se realizan antes y cuáles después. Este orden de precedencia predeterminado puede alterarse a conveniencia del programador como veremos posteriormente.

Una expresión puede ser tan simple como un único operando (un literal, una variable o una constante) sin operador, lo que arrojaría como resultado el valor del propio operando, o una combinación de varios operadores (de diferentes tipos) y operandos (también de diferentes tipos) que implican múltiples operaciones a realizar.

7.1 Operadores

En este apartado vamos a ver los operadores que emplea PHP clasificados por tipo de operación.

7.1.1 Asignación

El operador de asignación = asigna el resultado de evaluar una expresión a una variable. Su sintaxis es:

```
$<variable> = <expresión>;
```

Donde

- ✓ <variable> → Identificador de la variable a la que se le asigna un nuevo valor.
- ✓ <expresión> → Expresión que se evalúa para asignar el resultado a la variable.

En la parte izquierda de la asignación siempre tiene que ir una variable y la expresión a la derecha del operador de asignación puede ser tan simple o compleja como se necesite. Podría ser un literal simple o una expresión con múltiples operandos (literales, variables, valores devueltos por una función) y operadores.

Para asignar un valor nulo a una variable empleamos la palabra clave `NULL`. Un valor nulo indica ausencia de valor.

Existen una serie de operadores de asignación que conllevan una operación aritmética en la que está implicada como operando la propia variable a la que se le asigna el valor. Estos operadores son los siguientes:

| Operador | Descripción | Ejemplo |
|----------|-------------|---------|
|----------|-------------|---------|

| | | |
|-----|----------------------------|-----------------------|
| += | Suma y asignación | \$numero+=30 |
| -= | Resta y asignación | \$peso-=3 |
| *= | Producto y asignación | \$sueldo*=2 |
| **= | Potencia y asignación | \$numero**=3 |
| /= | División y asignación | \$gastos/=4 |
| .= | Concatenación y asignación | \$nombre.=\$apellidos |
| %= | Módulo y asignación | \$numero%=3 |

Vemos que cada operador lleva el símbolo de la operación y el símbolo de la asignación. Veamos algunos ejemplos:

```
<?php
$a = ($b = 4) + 5; // ahora $a es igual a 9
                  // y $b se ha establecido en 4.

$a = 3;
$a += 5;          // establece $a en 8, igual que: $a = $a + 5;
$b = "Hola ";
$b .= "ahí!";     // establece $b en "Hola ahí!",
                  // al igual que $b = $b . "ahí!";

?>
```

Hay dos operadores parecidos pero son unarios y se emplean para aumentar o decrementar el valor de una variable numérica en 1. Además, se pueden usar delante o detrás del operando, provocando diferente resultado en función de si forma parte de una expresión mayor. Estos operadores se les conoce como incremento y decremento, y cuando preceden al operando es preincremento y predecremento, o postincremento y postdecremento. Son estos:

| Operador | Descripción | Ejemplo |
|----------|----------------|------------|
| ++ | Preincremento | ++\$numero |
| | Postincremento | \$numero++ |
| -- | Predecremento | --\$numero |
| | Postdecremento | \$numero-- |

Cuando en un programa escribimos una expresión como la siguiente:

```
++$numero;
```

Es equivalente a escribir lo siguiente

```
$numero = $numero + 1;
```

Si hubiéramos puesto el operador después de la variable, también estamos incrementando en uno el valor de la variable. La diferencia de poner el operador antes o después está en cuando se produce el incremento. Veamos un ejemplo

```
<?php
$numero = 3;
$resultado = ++$numero - 2;
echo("El valor de resultado es: " + $resultado);
?>
```

Fijémonos en la segunda línea. `++$numero` es un operando en una operación de resta. Al estar el operador `++` delante de la variable `$numero`, primero se incrementa la variable (valdría 4) y luego se evalúa la expresión. Por tanto la variable `$resultado` tendrá el valor 2. Sin embargo, si el operador está detrás usando el mismo ejemplo anterior.

```
<?php
$numero = 3;
$resultado = $numero++ - 2;
echo("El valor de resultado es: " + $resultado);
?>
```

En este caso, al estar detrás el operador `++`, primero se emplea el valor actual de la variable `$numero` para evaluar la expresión y, posteriormente, se incrementa. En este caso el valor de `$resultado` sería 1.

Nótese que cuando el operador `++` está en una expresión con únicamente la variable que acompaña, es indiferente si se pone delante o detrás, ya que no se está utilizando en una expresión de ámbito mayor.

El operador `--` tiene un comportamiento similar, pero restando uno en lugar de sumar.

Todos los operadores anteriores de asignación e incremento, o decremento, tienen que estar pegados a la variable que acompañan, sin espacio entre ellos.

7.1.2 Aritméticos

Un operador aritmético se emplea para representar una operación aritmética. Java dispone de los siguientes operadores aritméticos.

| Ejemplo | Nombre | Resultado |
|-------------------------|----------------|--|
| <code>+\$a</code> | Identidad | Conversión de <i>\$a</i> a int o float según el caso. |
| <code>-\$a</code> | Negación | Opuesto de <i>\$a</i> . |
| <code>\$a + \$b</code> | Suma | Suma de <i>\$a</i> y <i>\$b</i> . |
| <code>\$a - \$b</code> | Resta | Diferencia de <i>\$a</i> y <i>\$b</i> . |
| <code>\$a * \$b</code> | Multiplicación | Producto de <i>\$a</i> y <i>\$b</i> . |
| <code>\$a / \$b</code> | División | Cociente de <i>\$a</i> y <i>\$b</i> . |
| <code>\$a % \$b</code> | Módulo | Resto de <i>\$a</i> dividido por <i>\$b</i> . |
| <code>\$a ** \$b</code> | Exponenciación | Resultado de elevar <i>\$a</i> a la potencia <i>\$b</i> ésima. |

El operador de división `/` devuelve un valor flotante a menos que los dos operandos sean integers (o strings que se conviertan a integers) y los números sean divisibles, en cuyo caso será devuelto un valor integer.

Los operandos del módulo se convierten en integers (por extracción de la parte decimal) antes del procesamiento.

El resultado del operador módulo % tiene el mismo signo que el dividendo, es decir, el resultado de $\$a \% \b tendrá el mismo signo que $\$a$. Por ejemplo

```
<?php
echo (5 % 3) . "\n";           // muestra 2
echo (5 % -3) . "\n";         // muestra 2
echo (-5 % 3) . "\n";         // muestra -2
echo (-5 % -3) . "\n";        // muestra -2
?>
```

7.1.3 Relacionales

Los operadores relacionales o de comparación devuelven un resultado verdadero o falso en función de la comparación de sus operandos. Generalmente se emplean para escribir expresiones relacionales y lógicas en estructuras de control, en la que el flujo del programa ejecuta una serie de instrucciones en función del resultado de la expresión. Son los siguientes:

| Ejemplo | Nombre | Resultado |
|---------------------|-------------------|---|
| $\$a == \b | Igual | true si $\$a$ es igual a $\$b$ después de la manipulación de tipos. |
| $\$a === \b | Idéntico | true si $\$a$ es igual a $\$b$, y son del mismo tipo. |
| $\$a != \b | Diferente | true si $\$a$ no es igual a $\$b$ después de la manipulación de tipos. |
| $\$a <> \b | Diferente | true si $\$a$ no es igual a $\$b$ después de la manipulación de tipos. |
| $\$a !== \b | No idéntico | true si $\$a$ no es igual a $\$b$, o si no son del mismo tipo. |
| $\$a < \b | Menor que | true si $\$a$ es estrictamente menor que $\$b$. |
| $\$a > \b | Mayor que | true si $\$a$ es estrictamente mayor que $\$b$. |
| $\$a <= \b | Menor o igual que | true si $\$a$ es menor o igual que $\$b$. |
| $\$a >= \b | Mayor o igual que | true si $\$a$ es mayor o igual que $\$b$. |
| $\$a <=> \b | Nave espacial | Un integer menor que, igual a, o mayor que cero cuando $\$a$ es respectivamente menor que, igual a, o mayor que $\$b$. |
| $\$a ?? \$b ?? \$c$ | Fusión de null | El primer operando de izquierda a derecha que exista y no sea null. null si no hay valores definidos y no son null. |

Si se compara un número con un string o la comparación implica strings numéricos, entonces cada string es convertido en un número y la comparación realizada numéricamente. Estas reglas también se aplican a la sentencia switch. La conversión de tipo no tiene lugar cuando la comparación es `===` o `!==` ya que esto involucra comparar el tipo así como el valor.

7.1.4 Lógicos

Generalmente se emplean para unir dos o más expresiones relacionales. Cada expresión relacional arroja un resultado de tipo boolean. Cuando queremos comprobar el cumplimiento, o no, de un conjunto de comparaciones tenemos que emplear un operador lógico. Son los siguientes:

| Ejemplo | Nombre | Resultado |
|---------------------------------|-------------------|--|
| <code>\$a and \$b</code> | And (y) | <code>true</code> si tanto <code>\$a</code> como <code>\$b</code> son <code>true</code> . |
| <code>\$a or \$b</code> | Or (o inclusivo) | <code>true</code> si cualquiera de <code>\$a</code> o <code>\$b</code> es <code>true</code> . |
| <code>\$a xor \$b</code> | Xor (o exclusivo) | <code>true</code> si <code>\$a</code> o <code>\$b</code> es <code>true</code> , pero no ambos. |
| <code>! \$a</code> | Not (no) | <code>true</code> si <code>\$a</code> no es <code>true</code> . |
| <code>\$a && \$b</code> | And (y) | <code>true</code> si tanto <code>\$a</code> como <code>\$b</code> son <code>true</code> . |
| <code>\$a \$b</code> | Or (o inclusivo) | <code>true</code> si cualquiera de <code>\$a</code> o <code>\$b</code> es <code>true</code> . |

La razón para tener las dos variaciones diferentes de los operadores `and` y `or` es que ellos operan con precedencias diferentes.

El resultado de estos operadores está regulado por las tablas de verdad. En las mismas se indica el valor de los operadores y el resultado proporcionado por el operador lógico. La siguiente tabla de verdad corresponde a los operadores **and** y **&&**.

| Expresión 1 | Expresión 2 | Expresión1 and Expresión2 Expresión1 && Expresión2 |
|-------------|-------------|---|
| false | false | false |
| false | true | false |
| true | false | false |
| true | true | true |

Expresión1 y Expresión2 corresponden a dos expresiones relacionales o lógicas. Como podemos ver para que el resultado sea `true`, ambos operandos tienen que ser `true`. La siguiente tabla de verdad corresponde a los operadores **or** y **||**.

| Expresión 1 | Expresión 2 | Expresión1 or Expresión2 Expresión1 Expresión2 |
|-------------|-------------|--|
| false | false | false |
| false | true | true |
| true | false | true |
| true | true | true |

En este caso solo hace falta que uno de los operandos sean `true` para que el resultado del operador **or** o **||** sea `true`.

El operador `xor` funciona de forma parecida al anterior, pero con una diferencia. Solamente es verdadero cuando uno de los operandos es verdadero y el otro es falso. Su tabla de verdad es la siguiente:

| Expresión 1 | Expresión 2 | Expresión1 xor Expresión2 |
|-------------|-------------|---------------------------|
|-------------|-------------|---------------------------|

| | | |
|-------|-------|-------|
| false | false | false |
| false | true | true |
| true | false | true |
| true | true | false |

Por último el operador `!` niega el resultado de la expresión. Si es `true` el resultado es `false` y si es `false`, el resultado será `true`. Su tabla de verdad es la siguiente:

| Operando | ! Operando |
|----------|------------|
| false | true |
| true | false |

Cada uno de los operandos que emplean estos operadores es una expresión lógica. Veamos los siguientes ejemplos. Supongamos que declaramos las siguientes variables

```
$numero = 4;
$nombre = "Juan"
```

El resultado de las siguientes expresiones lógicas sería el siguiente:

| Expresión | Resultado |
|--|-----------|
| <code>\$numero > 3 && \$nombre == "José"</code> | false |
| <code>\$numero != 3 \$nombre == "Maria"</code> | true |
| <code>\$numero != 8 xor \$nombre == "Juan"</code> | false |
| <code>!\$nombre == "María"</code> | true |

En el tercer ejemplo las dos expresiones relacionales son verdaderas, pero el operador `xor` es un O exclusivo. Es necesario que una de ellas sea falsa y la otra verdadera para que arroje un resultado verdadero.

Los operadores `and` y `&&` son similares, lo mismo ocurre con `or` y `||`. La diferencia estriba en que `&&` y `||` tienen mayor precedencia que `and` y `or`.

En cualquier caso las expresiones lógicas se evalúan en cortocircuito. Esto significa que si al evaluar la primera expresión lógica ya se puede obtener el resultado de la expresión lógica sin necesidad de evaluar la segunda expresión, esta segunda expresión no será evaluada, con el ahorro en ciclos de CPU. Repasemos los ejemplos anteriores.

```
$numero > 3 && $nombre == "José"
```

El resultado es `false`, tal y como se vio antes. La primera expresión relacional `$numero > 3` es `false`. No importa el resultado de `$nombre == "José"`, el resultado global es `false`. Debido a ello, al ejecutar el código no se evalúa la segunda condición, ya que no afecta al resultado final. Así, se ahorra tiempo de ejecución de la instrucción.

Si `$numero > 3` fuera `true`, entonces no habría más remedio que evaluar la segunda expresión relacional, ya que será su resultado el que determine el resultado de la expresión lógica global.

Con los operadores `or` y `||` ocurre algo similar. Veamos el siguiente ejemplo

```
$numero != 3 || $nombre == "Maria"
```

Con el resultado de `$numero != 3` ya se puede obtener el resultado global y es `true`. Por ello no se evalúa la segunda expresión. Solamente si la primera fuera `false`, entonces se evaluaría la segunda.

7.1.5 Operador de concatenación de cadenas

El operador `.` se utiliza para la concatenación de cadenas de caracteres. Por ejemplo

```
<?php
$a = "Hello ";
$b = $a . "World!"; // ahora $b contiene "Hello World!"

$a = "Hello ";
$a .= "World!";      // ahora $a contiene "Hello World!"
?>
```

7.2 Precedencia y asociatividad de operadores

La precedencia es el orden que determina la secuencia en que deben realizarse las operaciones cuando en una expresión intervienen operadores de distinto tipo. Las reglas de precedencia de operadores que utiliza PHP coinciden con las reglas de las expresiones del álgebra convencional, es decir, la multiplicación, división y módulo de una operación se evalúan primero. Si dentro de la misma expresión tengo varias operaciones de este tipo, empezaré evaluándolas de izquierda a derecha.

La suma y la resta se aplican después que las anteriores. De la misma forma, si dentro de la misma expresión tengo varias sumas y restas empezaré evaluándolas de izquierda a derecha.

A la hora de evaluar una expresión es necesario tener en cuenta la asociatividad de los operadores. La asociatividad indica qué operador se evalúa antes, en condiciones de igualdad de precedencia. Los operadores de asignación, el operador condicional (`?:`), los operadores incrementales (`++`, `--`) y el casting son asociativos por la derecha. El resto de operadores son asociativos por la izquierda, es decir, que se empiezan a calcular en el mismo orden en el que están escritos: de izquierda a derecha. Por ejemplo, en la expresión siguiente:

```
10 / 2 * 5
```

Realmente la operación que se realiza es $(10 / 2) * 5$, porque ambos operadores, división y multiplicación, tienen igual precedencia y por tanto se evalúa primero el que antes nos encontramos por la izquierda, que es la división. El resultado de la expresión es 25. Si fueran asociativos por la derecha, puedes comprobar que el resultado sería diferente, primero multiplicaríamos $2 * 5$ y luego dividiríamos entre 10, por lo que el resultado sería 1. En esta otra expresión:

```
$x = $y = $z = 1
```

Realmente la operación que se realiza es $\$x = (\$y = (\$z = 1))$. Primero

asignamos el valor de 1 a la variable `$z`, luego a la variable `$y`, para terminar asignando el resultado de esta última asignación a `$x`. Si el operador asignación fuera asociativo por la izquierda esta operación no se podría realizar, ya que intentaríamos asignar a `$x` el valor de `$y`, pero `$y` aún no habría sido inicializada.

El uso de paréntesis, incluso cuando no es estrictamente necesario, a menudo puede aumentar la legibilidad del código haciendo grupos explícitamente en lugar de confiar en la precedencia y asociatividad implícitas del operador.

La siguiente tabla enumera los operadores en orden de precedencia, con los de más alta precedencia al inicio. Los operadores en la misma línea tienen igual precedencia, en cuyo caso la asociatividad decide el agrupamiento.

| Asociatividad | Operadores |
|---------------|--|
| izquierda | [|
| derecha | ** |
| derecha | ++ -- ~ (int) (float) (string) (array) (object) (bool) @ |
| no asociativo | instanceof |
| derecha | ! |
| izquierda | * / % |
| izquierda | + - . |
| izquierda | << >> |
| no asociativo | < <= > >= |
| no asociativo | == != === !== <> <=> |
| izquierda | & |
| izquierda | ^ |
| izquierda | |
| izquierda | && |
| izquierda | |
| derecha | ?? |
| izquierda | ? : |
| derecha | = += -= *= **= /= .= %= &= = ^= <<= >>= |
| izquierda | and |
| izquierda | xor |
| izquierda | or |

8 Bibliografía

OLSSON, M. *PHP 8 Quick Scripting Reference - 3rd Edition*. 2021 Apress, ISBN 978-1-4842-6619-9

TATROE, K. y MACINTYRE, P., *Programming PHP – 4th Edition*. 2020. O'Reilly, ISBN 978-1-492-05413-9

HEIDI, E. *Cómo instalar la pila Linux, Apache, MySQL y PHP (LAMP) en Ubuntu 20.04*. (19 May 2020) <https://www.digitalocean.com/community/tutorials/how-to-install-linux-apache-mysql-php-lamp-stack-on-ubuntu-20-04-es>