UD2 – Introducción a PHP

2º CFGS
Desarrollo de Aplicaciones Web

2022-23

Actividad para casa

Ver el vídeo de Jesús Amieiro: PHP en 2020 a partir del minuto 3:32:00

Son 40 minutos(a velocidad x1.5 son 27 min.).

https://www.youtube.com/watch?v=o3IwAqslGUM&t=12724s

Responder a las siguientes preguntas:

- ¿Qué relación tiene Facebook con PHP?
- ¿Qué versión mínima se recomienda por rendimiento?
- ¿Qué razones hacen que PHP tenga mala fama?

1.- PHP

Lenguaje de programación de **propósito general**.

Su uso se ha extendido principalmente en el **desarrollo web**.

Su **sintaxis** es similar a **C** y **Java**.

Es un lenguaje **interpretado**.

1.- PHP

El mejor sitio para consultar la sintaxis y las funciones de PHP es en la documentación oficial https://www.php.net/manual/es.

La documentación oficial siempre se encuentra actualizada.

En la documentación oficial se indica las **versiones compatibles** de las funciones del lenguaje.

2.- Integración de PHP en HTML

Los archivos con código PHP tienen la extensión **.php** y habitualmente se les llama **scripts**.

Un archivo .php (script PHP) es **un archivo HTML** que además de etiquetas HTML puede contener código PHP que el servidor web ejecutará.

Los bloques de código PHP se escriben entre los delimitadores **<?php** y **?>**.

```
<?php
   echo "Desarrollo Web Entorno Servidor";
?>
```

Las expresiones PHP han de acabar con **;**.

2.- Integración de PHP en HTML

Un archivo .php dentro puede contener:

- Solo código HTML
- Solo código PHP
- Combinación de HTML y PHP

Práctica 1

Crea el archivo **prueba.php** en el host virtual **localhost.actividades**

Añade el siguiente código al archivo.

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Primera prueba php</title>
</head>
<body>
    Este es un archivo php que se encuentra en el servidor.
</body>
</html>
```

Entra en el navegador y accede a la URL: **http://localhost.actividades/prueba.php**

Práctica 2

Crea el archivo **info.php** en el host virtual **localhost.actividades**

Añade el siguiente código al archivo.

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

Entra en el navegador y accede a la URL: **http://localhost.actividades/info.php**
Localiza: la versión de PHP y el directorio donde se encuentra el script ejecutado.

*Si un archivo .php solo contiene código PHP se aconseja no poner el delimitador de cierre para evitar que se añadan espacios en blanco o líneas accidentales.

Práctica 3

Crea el archivo **minombre.php** en el host virtual **localhost.actividades**

```php
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Presentación</title>
</head>
<body>
    <?php
        $nombre = "TuNombre TuApellido";
        echo 'Hola yo soy '. $nombre;
    ?>
</body>
</html>
```

Entra en el navegador y accede a la URL: **http://localhost.actividades/minombre.php**

2.- Integración de PHP en HTML

Tanto el servidor como los navegadores web disponen de memoria cache.

En ocasiones los cambios en las páginas web tardan en visualizarse en el navegador.

Para ayudar a que el navegador web cargue correctamente la última versión de la página web se recomienda añadir la siguiente línea en el **head**.

<meta http-equiv="expires" content="Sat, 07 feb 2016 00:00:00 GMT">

2.- Integración de PHP en HTML

El código PHP incluido en el script se ejecuta en el servidor y habitualmente como se ha visto en la **Practica 3** puede generar contenido HTML.

Hay 3 maneras de generar contenido HTML desde PHP:

- **echo** *expresión*;
- **print** *expresión*;
- **<?=** *variable* **?>**

Ni **echo** ni **print** son funciones, por eso no es necesario poner paréntesis.

La última opción no se debe poner dentro de un bloque PHP.

En la **Practica 3** se ha visto que mediante el operador **.** se pueden concatenar cadenas.

Programming tip: Cadenas de texto

En las etiquetas del lenguaje HTML se suelen utilizar atributos, el valor de estos se indica entre **comillas dobles**:

```
<a href="fotos.php"><img src="camara.png" alt="Galería fotográfica">fotos</a>
```

Como en PHP es habitual generar código HTML, para declarar y usar cadenas se recomienda el uso de comillas simples.

```
<?php
echo '<a href="fotos.php"><img src="camara.png" alt="Galería fotográfica">fotos</a>';
?>
```

Si se quiere imprimir una comilla simple se tendrá que usar la secuencia de escape: \'

3.- Comentarios

Comentarios de una línea:

```
// Función para loguearse
```

Comentarios de varias líneas:

```
/* Script desarrollado por:
      Álex Torres
      Versión: 1.2 */
```

El código comentado en PHP no aparecerá en documento HTML final que se genera.

*Recuerda que los comentarios en HTML sí que aparecerán: <!-- -->

3.- Comentarios

Comentarios con phpDoc:

Es **recomendable** utilizar **comentarios phpDoc** de manera que el IDE los reconozca y que posteriormente se pueda **generar la documentación** de manera automática.

```
/**
 * Bloque de comentario phpDoc
 *
 *
 */
```

Son similares a los comentarios JavaDoc de Java.

3.- Comentarios

Comentarios con phpDoc:

Con phpDoc es recomendable comentar todos los elementos estructurales:

- Archivo
- Código importado: require e include
- Clases
- Interfaces
- Rasgos (traits)
- Funciones y métodos
- Propiedades de las clases
- Constantes

3.- Comentarios

Comentarios con phpDoc:

Antes de continuar echemos un vistazo a la cómo documentar con phpDoc:

```
/**
 * This is the summary for a DocBlock.
 *
 * This is the description for a DocBlock. This text may contain
 * multiple lines and even some _markdown_.
 *
 * * Markdown style lists function too
 * * Just try this out once
 *
 * The section after the description contains the tags; which provide
 * structured meta-data concerning the given element.
 *
 * @author  Mike van Riel <me@mikevanriel.com>
 *
 * @since 1.0
 *
 * @param int    $example  This is an example function/method parameter description.
 * @param string $example2 This is a second example.
 */
```

3.- Comentarios

Comentarios con phpDoc:

En la documentación se pueden ver las etiquetas disponibles:

3.- Comentarios

Comentarios con phpDoc:

Para generar la documentación phpDoc se necesita incluir en el directorio del proyecto el archivo **phpDocumentor.phar**.

A continuación, se debe ejecutar por terminal desde el directorio del proyecto el siguiente comando:

```
php phpDocumentor.phar -d . -t docs/api
```

De esta manera en el directorio del proyecto se creará la ruta **docs/api** y dentro se generará la documentación en formato HTML.

4.- Inclusión de ficheros externos

Al desarrollar un programa es habitual que se **reutilice el código**.

Esto puede ocurrir con **clases**, **librerías de funciones, …** y en el desarrollo web también con secciones de la aplicación web.

En PHP existen las siguientes instrucciones para reutilizar archivos php:

```
include("ruta_al_archivo_php");
include_once("ruta_al_archivo_php");
require("ruta_al_archivo_php");
require_once("ruta_al_archivo_php");
```

4.- Inclusión de ficheros externos

Estas funciones en caso de encontrar el archivo añaden todo el contenido en el punto donde se encuentran.

Las funciones **include** si no encuentran el archivo muestran una advertencia y continúan la ejecución.

Las funciones **require** si no encuentran el archivo muestran un error y detienen la ejecución.

Las versiones **_once** aseguran que el código solo se añada **una vez**, esto permite evitar errores, por ejemplo, si el archivo contiene una función.

4.- Inclusión de ficheros externos

Es habitual que si un archivo php está pensado para que siempre que se use se haga mediante su inclusión con **include** o **require** se le ponga la extensión **.inc.php**.

Por ejemplo:

cabecera.inc.php

funciones.inc.php

4.- Inclusión de ficheros externos

Como **HTML no es un lenguaje de programación** no tiene ninguna herramienta que permita la inclusión de código HTML externo.

Por esta razón en las aplicaciones web estáticas es difícil reutilizar el código de manera sencilla.

En la realización de la actividad **El portafolio** para que todas las secciones tuvieran la misma cabecera, menú, pie y estilo se tuvo que copiar el código en los diferentes archivos html.

Gracias a las funciones **include** y **require** se soluciona el problema anterior.

Práctica

Actividad 1:

El portafolio v2.0

5.- Tipos de datos

Los tipos de datos disponibles en PHP son:

Tipo	Descripción	Ejemplo
Boolean	Verdadero o falso	false
Integer	Número sin decimales	58
Float	Número con decimales	4.7
String	Conjunto de datos delimitados por comillas simples o dobles	"Rick Sanchez"
Array	Conjunto de datos del mismo o diferente tipo	"Rick", "Sanchez", 70, 'humano'
Null	No hay valor asignado	null

5.- Tipos de datos

Boolean: valores posibles.

Verdadero	Falso
true TRUE Cualquier número entero o decimal, positivo o negativo distinto de cero. Cualquier cadena que no sea cero o cadena vacía	false FALSE 0 -0 0.0 -0.0 "0" "" null

6.- Variables

Las variables permiten **almacenar datos** de algún tipo.

Los nombres de variables deben comenzar por **$** seguido de **una letra** o el símbolo _.

No se recomienda usar caracteres no ingleses y símbolos.

Algunos ejemplos válidos:

```
$edad                    $cantidad_pelotas
$_piso                   $_3tipos
$primerApellido          $apellido2
$nombreCalle             $i
```

Los nombres son **case sensitive**, así, $nombre es diferente a $Nombre.

6.- Variables

PHP es un lenguaje de programación **débilmente tipado**, eso significa:

- Para empezar a usar una variable simplemente se le ha de asignar un valor. No es necesario declararlas previamente.

- En cualquier momento se puede cambiar el tipo de dato que almacena una variable.

Se usa el operador **=** para asignar valor a una variable:

```
$variable = 5;
$variable = "Morty";
$variable = true;
```

6.- Variables

Se pueden eliminar variables con la función **unset**.

```
$serie = "Rick & Morty";
$temporadas = 5;
$nombre = "Rick";
$apellido = "Sanchez";

unset($serie);

unset($temporadas, $nombre, $apellido);
```

7.- Variables y tipos de datos

Si en una **operación** intervienen **variables de distinto tipo** PHP intentará convertirlas a un tipo común, a esto se le llama **casting**.

```
$cantidad = 3;
$precio = 1.6;
$total = $cantidad * $precio;

$total = $cantidad * (int)$precio; // casting forzado

echo "El precio final es ". $precio ." €";
```

8.- Constantes

Las constantes almacenan valores que no pueden variar durante la ejecución del programa.

Los nombres de las constantes son en **mayúsculas**.

Hay dos maneras de crear una constante:

```
define("PI", 3.141592);

const NOMBRE = "Morty Smith";    //a partir de PHP 5.3
```

Existen constantes predefinidas que se pueden consultar en la documentación.

9.- Fechas y horas

PHP no tiene un tipo de datos concreto para fechas y horas.

Se utiliza la marca de tiempo UNIX:

Segundos que han pasado desde las 00:00:00 del 1/1/1970

La función **time** devuelve el instante actual en formato UNIX.

```
$start = time();
/* instrucciones PHP */
echo "Página generada en ". time()-$start ." segundos.";
```

9.- Fechas y horas

Es importante indicar la zona horaria.

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

Esto es necesario cuando el servidor web se encuentra en una zona horaria distinta a donde se visita la aplicación web.

Si se desea generar una fecha específica se usa la función **mktime**:

```
// mktime(hora, minutos, segundos, mes, dia, año)

$cumpleanyos = mktime(0,0,0,5,18,1994);
```

9.- Fechas y horas

Para mostrar detalles de una fecha en formato UNIX se usa la función **date**.

```
echo date("l, d M Y", time());

// Igual que la instrucción de arriba
// echo date("l, d M Y");
```

Wednesday, 18 May 1994

En la documentación se pueden consultar todas las opciones de formato.

10.- Expresiones y operadores

Mediante los operadores se realizan operaciones con variables creando así expresiones.

Los operadores disponibles en PHP son similares a los operadores de Java.

Tipo de operación	Operadores disponibles
Asignación	= += -= *= /= %=
Aritmética	+ - * / % ** ++ --
Comparación	> < >= <= == != <> !== === <=> ??
Comparación booleana	&& || ! and or xor
Sobre bits (integer)	& | ^(xor) ~(not) << >>
Concatenación (strings)	. .=

La prioridad en los operadores es **de izquierda a derecha** y siempre en este orden:
paréntesis, negación (!), producto/división, sumas/restas, comparación, lógicos y asignación

10.- Expresiones y operadores

A partir de PHP7

Nave espacial: **`$a <=> $b`**

Devuelve:

-1 si $a es menor que $b

0 si $a es igual que $b

1 si $a es mayor que $b

Fusión de null: **`$a ?? $b ?? $c`**

Tantas variables como se quiera.

Devuelve el primer operando desde la izquierda que no sea null.

11.- Ámbito de las variables

Existen tres ámbitos posibles para una variable:

- **Local al script**: a partir de que se crea una variable, esta ya se puede usar en el resto del script PHP, aunque la variable se cree dentro de una estructura de control (if, for…). Pero no se podrá usar dentro de una función.

- **Local a una función**: si la variable se crea dentro de una función, esta solo se podrá usar dentro de la propia función.

- **Global**: una variable **local al script** se podrá usar dentro de una función si se indica con la palabra reservada **global**. **Se debe evitar este uso.**

11.- Ámbito de las variables

- **Local al script**:

Si la variable se crea en el script o en una estructura de control, esa variable se podrá usar en todo el script excepto en las funciones.

```php
<?php
$i = 100;
$temp = 0;
while ($i!=0) {
    $temp += $i--;
    $suma = $temp;
}
echo 'variable temp: '. $temp;
echo '<br>';
echo 'La suma de los 100 primeros números: '. $suma;
```

11.- Ámbito de las variables

- **Local al script**:

Si la variable se crea en el script fuera de una función, esa variable no se podrá usar en las funciones del script.

```php
<?php
$nombre = 'Álex';
saludo();

function saludo() {
    echo '¡Hola soy '. $nombre .'!'; //Error
}
```

11.- Ámbito de las variables

- **Local a una función**:

Si la variable se crea dentro de una función, esa variable solo se podrá usar en esa función.

```php
<?php
saludo();
echo $nombre; // Error

function saludo() {
    $nombre = 'Alex';
    echo '¡Hola soy '. $nombre .'!';
}
```

11.- Ámbito de las variables

- **Global**:

Si la variable se crea en el script fuera de una función, esa variable se podrá usar en las funciones del script si se usa la palabra reservada **global**.

```php
<?php
$nombre = 'Álex';
saludo();

function saludo() {
    global $nombre;
    echo '¡Hola soy '. $nombre .'!';
}
```

Se debe evitar siempre el uso de variables globales.

12.- Estructuras de control

PHP dispone de sentencias que permiten alterar el flujo de ejecución.

Estas sentencias son similares a las de Java:

if **switch** **while** **do-while** **for**

12.- Estructuras de control

if y if – else.

```php
if($num == 0) {
    echo "El número es cero.";
}
```

```php
if($num == 0)
    echo "El número es cero.";
```

```php
if ($a < $b) {
    echo "a es menor que b";
} else if ($a > $b) {
    echo "a es mayor que b";
} else {
    echo "a es igual a b";
}
```

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

12.- Estructuras de control

Operador ternario

Es una manera abreviada de la estructura if – else.

```
$precio = 199.5;
$descuento = 15;
$precioFinal = ($descuento > 0) ? $precio-$precio*$descuento/100 : $precio;
```

12.- Estructuras de control

switch

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

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

¿Qué hace este código de arriba?

12.- Estructuras de control

while

```php
$a = 1;
while ($a < 8) {
    $a += 3;
}
echo $a;
```

do – while

```php
$a = 5;
do {
    $a -= 3;
} while ($a > 10);
echo $a;
```

¿Qué hace este código de arriba?

12.- Estructuras de control

for

```php
for ($a = 0; $a<10; $a++) {
    echo $a;
    echo "<br>";
}
```

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

¿Qué hace este código de arriba?

Práctica

Actividad 2:

Generando HTML

Actividad 3:

Calculando

Actividad 4:

Las tablas de multiplicar

13.- Arrays

En PHP los **array** son **mapas ordenados** estos se puede emplear como **array**, **lista**, **tabla hash**, **diccionario**, **pila**, **cola**…

Los mapas no tienen un tamaño establecido pudiendo añadir y eliminar elementos en cualquier momento.

Los mapas permiten almacenar pares **clave** – **valor**.

Las claves son los índices para acceder a los elementos del array

Los array pueden ser de 3 tipos:

- **Numérico**: todos los índices son integer (números enteros).
- **Asociativo**: todos los índices son strings.
- **Mixto**: tiene índices integer y strings.

13.- Arrays

Los arrays se pueden definir de dos maneras diferentes:

```
$palos = array("Corazones", "Picas", "Rombos", "Tréboles");

$personajes = ["Rick", "Morty", "Summer", "Beth", "Jerry"];
```

Si no se indica la clave automáticamente se asignará un **valor numérico** comenzando desde **cero**.

```
$palos[0] almacena la cadena: Corazones
$personajes[3] almacena la cadena: Beth
```

13.- Arrays

Se puede añadir elementos al final de un array de la siguiente forma:

```
$personajes = ["Luke", "Leia"];
$personajes[] = "Han Solo";   // se asigna la clave 2
```

Mediante la clave (índice) también se puede añadir elementos a un array:

```
$personajes[3] = "Darth Vader";
```

Se debe tener cuidado con esta práctica si se indica una clave mayor que la última posición del array:

```
$personajes[6] = "Amidala";   // las claves 4 y 5 no existen
$personajes[] = "Yoda";       // se asigna la clave 7
```

13.- Arrays

De esta manera también se puede inicializar un array:

```
$colores[] = "azul"; // Crea el array y añade un elemento
```

Si la variable contenía anteriormente un tipo de dato para poder crear un array en ella primero se debe eliminar la variable:

```
$serie = "Stranger Things";
unset($serie);
$serie[] = 3;
$serie[] = 49;
```

13.- Arrays

Un mismo array en PHP puede almacenar varios tipos de datos a la vez.

```
$personaje[] = "Rick Sanchez";   // nombre
$personaje[] = 70;               // edad
$personaje[] = 1.98;             // altura en cm
$personaje[] = 86;               // peso en kg
$personaje[] = "humano";         // especie
$personaje[] = "C-137";          // dimensión
$personaje[] = true;             // vivo
```

13.- Arrays

En los **arrays asociativos todas las claves son strings**. Se pueden definir con todas las técnicas vistas anteriormente añadiendo la clave:

```
$personaje = array( "nombre" => "Rick Sanchez",
                    "edad" => 70,
                    "altura" => 1.98);
```

```
$personaje = ["nombre" => "Rick Sanchez",
              "edad" => 70,
              "altura" => 1.98];
```

```
$personaje["nombre"] = "Rick Sanchez";
$personaje["edad"] = 70;
$personaje["altura"] = 1.98;
```

13.- Arrays

Al indicar una clave, PHP intenta hacer un **casting a integer**, por eso hay que tener cuidado con las claves

```
$array = [ 1 => "a",
           "1"  => "b",
           1.5  => "c",
           true => "d"];
```

Al hacer casting a integer todas las claves anteriores tienen el valor 1 por lo que el array declarado solo contiene un elemento con la clave 1:

```
$array[1]  contiene la cadena "d";
```

13.- Arrays

En los **arrays mixtos** se pueden mezclar claves integer y string.

```
$personaje["nombre"] = "Rick Sanchez";
$personaje[] = 70;                    // Se asigna la clave 0
$personaje["altura"] = 1.98;
$personaje[] = 86;                    // Se asigna la clave 1
$personaje[5] = "humano";
$personaje[] = "C-137";               // Se asigna la clave 6
```

Como se puede observar, no tiene mucho sentido crear un array de esta manera debido a la confusión de trabajar con él.

13.- Arrays

Un uso típico de los arrays mixtos es poder **acceder a todos** sus elementos tanto con la clave integer como con la clave string.

```php
$personaje = [0 => "Rick Sanchez",
              "nombre" => "Rick Sanchez",
              1 => 70,
              "edad" => 70,
              2 => 1.98,
              "altura" => 1.98,
              3 => "humano",
              "especie" => "humano"];
```

```php
$personaje[0] = "Rick Sanchez";
$personaje["nombre"] = "Rick Sanchez";
$personaje[1] = 70;
$personaje["edad"] = 70;
$personaje[2] = 1.98;
$personaje["altura"] = 1.98;
$personaje[3] = "humano";
$personaje["especie"] = "humano";
```

Las claves integer no es necesario ponerlas porque como ya se ha visto, si no se indica clave, esta será integer empezando desde cero.

Algunas funciones de acceso a bases de datos devuelven arrays mixtos.

13.- Arrays

Se pueden **eliminar** elementos del array con la función unset.

```
unset($array[2]);

unset($personaje["nombre"]);
```

Si es un array numérico habrá que volver a crear los índices mediante la función **array_values**, para evitar problemas si se recorre todo el array.

```
$array = array_values($array);
```

13.- Arrays

Como en PHP un elemento de un array puede ser de cualquier tipo, para crear matrices o arrays multidimensionales simplemente habrá que indicar que el elemento del array es un array.

```
$numeros = [[1, 2, 3], [4, 5, 6], [7, 8, 9]];

$peliculas = [["titulo" => "Matrix", "anyo" => 1999],
              ["titulo" => "Vengadores: Infinity War", "anyo" => 2018],
              ["titulo" => "Vengadores: Endgame", "anyo" => 2019],
              ["titulo" => "Rogue One", "anyo" => 2016],
];

$array = [["Ana", "Luis", "Marta"],
          "colores" => ["amarillo", "verde"],
          ["saludo" => "hola", "despedida" => "adiós", "¿Qué tal estás?"]];
```

13.- Arrays

Mediante la función **count** se obtiene la cantidad de elementos de un array.

Esta función permite recorrer un **array numérico** sin problema.

```
<ul>
<?php
$palos = array("Corazones", "Picas", "Rombos", "Tréboles");
for($i=0; $i<count($palos); $i++)
   echo '<li>'. $palos[$i] .'</li>';
?>
</ul>
```

- Corazones
- Picas
- Rombos
- Tréboles

13.- Arrays

Para recorrer un array numérico o asociativo es mejor usar **foreach**.

```
<ul>
<?php
$palos = array("Corazones", "Picas", "Rombos", "Tréboles");

foreach($palos as $valor)
    echo '<li>'. $palos[$i] .'</li>';
?>
</ul>
```

- Corazones
- Picas
- Rombos
- Tréboles

```
<?php
foreach($palos as $clave => $valor)
    echo $clave .'.- '. $valor .'<br>';
?>
```

0.- Corazones
1.- Picas
2.- Rombos
3.- Tréboles

Práctica

Actividad 5:

Ordenando datos

Actividad 6:

Ficha personal

Actividad 7:

Personas

Proyecto del trimestre

Presentación del proyecto

14.- Funciones predefinidas

PHP dispone de infinidad de funciones predefinidas (referencia funciones).

Es importante consultar la documentación para ver cómo **funciona** una función y cómo se realiza la **llamada** a la función.

empty

(PHP 4, PHP 5, PHP 7, PHP 8)

empty — Determina si una variable está vacía

Descripción

```
empty(mixed $var): bool
```

Determina si una variable es considerada vacía. Una variable se considera vacía si no existe o si su valor es igual a **false**. **empty()** no genera una advertencia si la variable no existe.

14.- Funciones predefinidas

Funciones para variables

Función	Descripción
empty	Comprueba si la variable está vacía (valor no vacío distinto de cero).
isset	Comprueba si una (o más variables) existe y no es null.
unset	Destruye una (o más variables).
is_int	Comprueba si una variable es integer.
is_float	Comprueba si una variable es float.
is_string	Comprueba si una variable es string.
is_array	Comprueba si una variable es un array
is_object	Comprueba si una variable es un objeto.
is_null	Comprueba si una variable es null.

14.- Funciones predefinidas

Funciones matemáticas

Función	Descripción
pow	Calcula la potencia (un número elevado a otro).
sqrt	Calcula la raíz cuadrada de un número.
round	Redondea un decimal dependiendo del dígito: 0 a 4 si hacia abajo, 5 a 9 hacia arriba.
ceil	Redondea un decimal hacia arriba.
floor	Redondea un decimal hacia abajo.
decbin	Convierte un número decimal a binario en forma de string.
bindec	Convierte un número binario en forma de string a decimal.
max	Encuentra el valor más alto entre las variables indicadas (también con array de integer).
min	Encuentra el valor más bajo entre las variables indicadas (también con array de integer).
sin, cos, tan…	Funciones trigonométricas.
rand	Genera un número aleatorio.
mt_rand	Genera un número aleatorio de una mejor manera que la anterior.

14.- Funciones predefinidas

Funciones para strings

Función	Descripción
strlen	Obtiene la longitud de una cadena.
substr	Devuelve una parte de una cadena.
str_replace	Sustituye parte de una cadena por otra cadena indicada.
strtolower	Transforma una cadena a minúsculas.
strtoupper	Transforma una cadena a mayúsculas.
trim	Elimina espacios al principio y al final.
explode	Divide un string en varios strings almacenándolos en un array.
implode	Une elementos de un array en un string.
strip_tags	Elimina etiquetas HTML.
strcmp	Comparación segura de dos cadenas.
strcasecmp	Comparación segura de dos cadenas convirtiéndolas primero a minúsculas.

14.- Funciones predefinidas

Funciones para arrays

Función	Descripción
count	Cuenta los elementos de un array
array_values	Devuelve un array con los valores del mismo.
array_keys	Devuelve un array con las claves del mismo.
array_key_exists	Comprueba si una clave existe en un array.
array_flip	Intercambia las claves y los valores de un array. Los valores pasarán a ser las claves y viceversa.
array_pop	Extrae el último elemento de un array.
array_push	Añade uno o más elementos al final de un array.
array_shift	Extrae el primer elemento del array. Si es numérico recalcula las claves para que empiecen en cero.
array_unshift	Añade uno o más elementos al inicio de un array.
array_reverse	Devuelve un array con los elementos en orden inverso al array indicado.
array_merge	Combina dos o más arrays.
in_array	Comprueba si un valor existe en un array.
sort	Ordena un array.

15.- Funciones propias

Para definir funciones propias se utiliza la palabra reservada **function**:

```php
<?php
function saludo () {
   echo 'Hola ¿Qué tal va?';
}
?>
```

Para usar una función, esta debe estar definida en el mismo script desde donde se realiza la llamada, ya sea **directamente** o a través de **include** o **require**.

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

15.- Funciones propias

Como se ha visto hasta el momento PHP es muy flexible.

Por esto a la hora de definir y usar funciones propias existen una serie de características que difieren de otros lenguajes de programación.

- Paso de argumentos por valor.
- Paso de argumentos por referencia.
- **Argumentos con valor por defecto/opcionales**.
- **Argumentos con nombre**.
- **Cantidad de argumentos**.
- Devolver un valor.
- Indicando el tipo de dato.

15.- Funciones propias

Paso de argumentos por valor.

Al definir una función se pueden indicar los argumentos que necesitará.

Lo más habitual es realizar el paso por **valor**, lo cual significa que se **realizará una copia** de la variable para usarla dentro de la función.

```php
function suma($op1, $op2) {
    echo $op1 .' + '. $op2 .' = '. $op1+$op2;
}

suma(14, 27);
```

15.- Funciones propias

Paso de argumentos por referencia.

Al pasar parámetros por **referencia no se crea una copia** de la variable y si se modifica dentro de la función los cambios se mantendrán fuera de esta.

```php
$numero = 6;
function incremento(&$numero, $cantidad) {
    $numero += $cantidad;
}

echo 'Número antes de la llamada: '. $numero .'<br>';
incremento($numero, 25);
echo 'Número después de la llamada: '. $numero;
```

15.- Funciones propias

Argumentos con valor por defecto/opcionales.

Se puede indicar un **valor por defecto** a los argumentos, estos argumentos además serán **opcionales**. Los argumentos con valor por defecto deben ser los últimos.

```php
function precio_final($precio, $iva=21, $descuento=0) {
    echo 'Precio sin iva: '. $precio .'€<br>';
    if($descuento!=0) {
        echo 'Descuento ('. $descuento .'%): '. $precio*$descuento/100 .'€<br>';
        $precio -= $precio*$descuento/100;
    }
    echo 'iva ('. $iva .'%): '. $precio*$iva/100 .'€<br>';
    echo 'Precio final: '. $precio + $precio*$iva/100 .'€<br>';
}

precio_final(150);
precio_final(150, 10);
precio_final(150, 10, 50);
```

15.- Funciones propias

Argumentos con valor por defecto/opcionales.

```php
function protagonistas($pelicula="todas") {
    $datos = ["El señor de los anillos" => "Frodo",
              "Matrix" => "Neo",
              "Piratas del caribe" => "Jack Sparrow",
              "Iron man" => "Tony Stark"];

    if($pelicula=="todas") {
        foreach($datos as $clave => $valor) {
            echo $clave .': '. $valor .'<br>';
        }
    }
    else if(!array_key_exists($pelicula, $datos)) {
        echo 'No hay datos de esa película.<br>';
    }
    else {
        echo $pelicula .': '. $datos[$pelicula] .'<br>';
    }
}

protagonistas();
protagonistas("Matrix");
protagonistas("Harry Potter");
```

15.- Funciones propias

Argumentos con nombre.

Desde PHP 8 se pueden realizar llamadas a funciones con el **nombre del argumento**:

```php
function precio_final($precio, $iva=21, $descuento=0) {···
}

precio_final(precio:150, descuento:15);
precio_final(150, descuento:15, iva:10);
precio_final(descuento:25, precio:199, iva:21);
```

Si se usan argumentos con y sin nombre en la misma llamada, los argumentos con nombre deben ser los últimos.

15.- Funciones propias

Cantidad de argumentos.

Aunque en la definición de la función se indiquen unos argumentos PHP al realizar la llamada a la función **se pueden indicar más argumentos** de los definidos, **nunca menos**.

```php
function funcion() {…
}
funcion();
funcion(1,2,3,4);

function suma($op1, $op2) {…
}
suma(); // Error
suma(14, 27);
suma(14, 27, 5, 36);
```

15.- Funciones propias

Cantidad de argumentos.

Aunque se indiquen más argumentos si en la definición de la función no se realiza ninguna acción con ellos no ocurrirá nada.

```php
function suma($op1, $op2) {
    echo $op1 .' + '. $op2 .' = '. $op1+$op2;
}

suma(14, 27);
suma(14, 27, 5, 36);
```

Existen dos maneras de acceder a los argumentos no indicados en la definición:

- Mediante las funciones: func_num_args y func_get_arg.
- Creando un array (PHP > 5.6).

15.- Funciones propias

Cantidad de argumentos.

Mediante las funciones: func_num_args y func_get_arg.

```php
function suma() {
    $suma = 0;
    for ($i = 0; $i < func_num_args(); $i++)
        $suma += func_get_arg($i);

    echo 'La suma de los números es: '. $suma;
}

suma(14, 27);
suma(14, 27, 5, 36);
```

15.- Funciones propias

Cantidad de argumentos.

Creando un array (PHP > 5.6).

```php
function suma(...$numeros) {
    $suma = 0;
    foreach($numeros as $numero)
        $suma += $numero;

    echo 'La suma de los números es: '. $suma;
}

suma(14, 27);
suma(14, 27, 5, 36);
```

15.- Funciones propias

Cantidad de argumentos.

Mediante la notación de tres puntos **…** se pueden definir parámetros concretos primero y al final dejar la opción de añadir más parámetros en la llamada.

```php
function operacion($operacion, ...$numeros) {
    $resultado = 0;
    if($operacion === "suma") {
        foreach($numeros as $numero) {
            $resultado += $numero;
        }
    } else if($operacion === "resta") {
        foreach($numeros as $numero) {
            $resultado -= $numero;
        }
    } else {
        $resultado = "Operación no soportada.";
    }
    return $resultado;
}

echo operacion("suma", 1, 6, 15, 4);
```

15.- Funciones propias

Cantidad de argumentos.

Los tres puntos **…** también permiten separar un array en diferentes variables cuando se pasa el array como argumento a una función.

```php
function precio_final($precio, $iva=21, $descuento=0) {…
}

$producto = ["PS5", 21.0, 0.0];
precio_final(...$producto);
$producto = ["PS4", 21.0];
precio_final(...$producto);
```

```php
function suma(...$numeros) {…
}

$numeros = [1,2,3,4];
suma(...$numeros);
```

15.- Funciones propias

Devolver valores.

Mediante **return** una función puede devolver un dato al finalizar su llamada.

```php
function suma(...$numeros) {
    $suma = 0;
    foreach($numeros as $numero)
        $suma += $numero;

    return $suma;
}
```

15.- Funciones propias

Indicando el tipo de dato.

Desde PHP 7 se permite indicar el tipo de dato de los argumentos y del valor devuelto.

Si se definen las funciones así PHP se vuelve más estricto.

```php
function precio_final(int $precio, float $iva=21, int $descuento=0) :float {
    $precio -= $precio*$descuento/100;
    return $precio -  + $precio*$iva/100;
}
```

15.- Funciones propias

Cabe recordar que cuando PHP usa una variable siempre va a intentar hacer un casting con ella para evitar posibles problemas.

Los siguiente ejemplos no producen errores ya que al hacer casting no hay error.

```php
$numero = "6";
function incremento($numero, $cantidad) {
    return $numero += $cantidad;
}
echo incremento($numero, 23);
```

```php
function suma(...$numeros) {
    $suma = 0;
    foreach($numeros as $numero)
        $suma += $numero;
    return $suma;
}
echo suma(14, "27", true);
```

16.- PHP Standards Recommendations

Como todo lenguaje de programación, en PHP existen una serie de convenciones a la hora de escribir código.

Estas convenciones se recogen en los PSR.

PSR (PHP Standards Recommendations).

Las PSR son recomendaciones que se pueden seguir o no.

Si se siguen el código será más estándar.

ACCEPTED	
NUM	TITLE
1	Basic Coding Standard
3	Logger Interface
4	Autoloading Standard
6	Caching Interface
7	HTTP Message Interface
11	Container Interface
12	Extended Coding Style Guide
13	Hypermedia Links
14	Event Dispatcher
15	HTTP Handlers
16	Simple Cache
17	HTTP Factories
18	HTTP Client

16.- PHP Standards Recommendations

Algunos ejemplos:

PSR-1: el nombre de los métodos debe ser camelCase.

PSR-12: Si el archivo .php solo contiene código PHP se omite el ?> final.
Se recomiendan 80 caracteres máximos por línea.
Se deben emplear 4 espacios para tabular (VSCode lo hace automáticamente).
Solo una instrucción por línea.
Las palabras reservadas deben escribirse en minúscula.

Debe haber un espacio tras las palabras if, else, switch, while, do, for.
No debe haber espacio tras el paréntesis de apertura ni antes del de cierre.
Debe haber un espacio entre el paréntesis de cierre y la llave de apertura.
El cuerpo de la estructura de control tiene que tener un tabulado.
Las palabras else y else if deben ir en la misma línea que la llave de cierre anterior.

Práctica

Actividad 8:

Analizando una frase

Actividad 9:

Calculando v2.0

Actividad 10:

Juegos de cartas