Unidad 2- Características lenguaje PHP

ÍNDICE

1. Lenguaje PHP

PHP (Personal Home Page) es un lenguaje de código abierto muy popular especialmente adecuado para el desarrollo web y que puede ser incrustado en HTML.

Características principales:

- Sintaxis similar a C /Java
- El código se ejecuta en el servidor
- El cliente recibe el resultado generado tras interpretar el código en el servidor
- El código se almacena en archivo con extensión .php

Podemos encontrar más información y documentación extensa en:
https://www.php.net/manual/es/

IMPORTANTE: Antes de comenzar

Nos dirigimos a: C:\xampp\htdocs
En htdocs creamos una carpeta que se llame DWEES.
Dentro de esta carpeta es donde vamos a ir poniendo ejemplos y ejercicios que se irán haciendo a lo largo de los temas.

Siempre que creemos un ejemplo o ejercicio hay que seguir los siguientes pasos:
1. Abrir el Xampp y nos conectamos a las conexiones Apache y MySQL.
2. Usaremos el Visual Studio Code como editor de código, donde escribiremos el código.
3. Para ver el código, nos iremos al navegador y pondremos: http://localhost/DWEES/ y pincharemos sobre el que queramos.

1.1 Código embebido

Los bloques de código se escriben entre **<?php** y **?>**, mientras que las sentencias se separan mediante **;**

```
1   <!DOCTYPE html>
2   <html lang="es">
3   <head>
4   <meta charset="UTF-8">
5   <title>PHP fácil</title>
6   </head>
7   <body>
8   <!-- Muestra una frase con HTML -->
9   Hola mundo<br>
10  <!-- Muestra una frase con PHP -->
11  <?php echo "Es muy fácil programar en PHP."; ?>
12  </body>
13  </html>
```

Nota: Cuando codifiquemos clases o interfaces (más adelante), nuestro código solo va a contener código PHP y nada de html. En este caso, solo pondremos las etiquetas de apertura, para así indicar que es un archivo de php puro.

1.2 Generando contenido

Tenemos tres posibilidades a la hora de generar contenido en nuestros documentos PHP:

- **Echo** expresión;
- **Print** expresión;
- **<?=** expresión ?>

Se utilizará `echo` cuando estemos dentro de un bloque de instrucciones y

<?=> cuando se vaya a mostrar el valor de una variable dentro de un fragmento HTML.

En las líneas 8, 9 y 10 se muestra un ejemplo del uso de estos 3.

```php
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Echo y print</title>
</head>
<body>
<p><?php echo "Este texto se mostrará en la página web." ?></p>
<p><?= "Este texto se mostrará en la página web." ?></p>
<p><?php print("Este texto se mostrará en la página web.") ?></p>
</body>
</html>
```

1.3 Comentarios

Para comentarios de una línea se utiliza //
Para comentarios de bloque /* con cierre *\

```php
<?php
// Este es un comentario de una sola línea
/*
  Este es
  un comentario
  que ocupa
  varias líneas
*/
?>
```

1.4 Variables

- No hace falta declararlas previamente.
- Comienzan por $
- Son *case sensitive*, es decir, distingue entre minúscula y mayúscula. `$var != $vAR`
- Tras el $, el siguiente carácter debe ser una letra minúscula o guion bajo _. Se recomienda poner minúscula. IMPORTANTE: no poner número detrás de $.
- No se declara su tipo, es dinámico. Se asigna en tiempo de ejecución dependiendo del valor asignado.

A priori, **no hay tipos de datos**. PHP trabaja internamente:

Escalares	Compuestos
- Boolean - Integer - Float - String	- Array - Object - Callable - Iterable

En el siguiente ejemplo se muestran una serie de variables:

```php
<?php
$nombre = "Pilar";
$nombreCompleto = "Pilar Jiménez";
$numero = 452345;
$numero2 = 456;
$pi = 3.14;
$suerte = true;

echo $nombre
?>
```

1.5 Constantes

Las constantes son variables cuyo valor no varían. Existen 2 posibilidades:

- define(NOMBRE, valor);
- const NOMBRE; // PHP > 5.3

Se declaran siempre en mayúsculas
Hay un conjunto de constantes ya predefinidas que se conocen como: *magic constants*
https://www.php.net/manual/es/language.constants.predefined.php

2. Operadores

Un operador es algo que toma uno o más valores y produce otro valor.

En este apartado veremos operadores aritméticos, de comparación, lógicos y de asignación. Hay muchos más, se pueden ver en: https://www.php.net/manual/es/language.operators.php

IMPORTANTE – PRIORIDAD DE LOS OPERADORES

1º Paréntesis
2ºNegación (!)
3º Producto/divisiones, sumas/restas
4º Comparadores
5º Lógicos
6º Asignación

2.1 Operadores aritméticos

Nombre	Ejemplo	Resultado
Negación	-$a	Opuesto de $a
Suma	$a + $b	Suma de $a y $b
Resta	$a - $b	Diferencia de $a y $b
Multiplicación	$a * $b	Producto de $a y $b
División	$a / $b	Cociente de $a y $b
Módulo/Resto	$a % $b	Resto de $a dividido por $b
Potencia	$a ** $b	Resultado de $a elevado a $b.

Cuando trabajemos con cadenas, si queremos concatenarlas, se utiliza el operador .

```php
<?php
$x = 4;
$y = 11;
$z = $x + $y;
echo "La suma de 4 y 11 es ".44."<br />";
echo "La suma de ".$x." y ".$y." es ".(33 + 11)."<br />";
echo "La suma de ".$x." y ".$y." es ".$z."<br />";
?>
```

También podemos imprimirlas directamente, ya que se expanden automáticamente

```php
<?php
echo "La suma de $x y $y es $z <br />";
?>
```

Más adelante estudiaremos algunas funciones para el tratamiento de cadenas

2.2 Operadores de comparación

Nombre	Ejemplo	Resultado
Igual	$a == $b	true si $a es igual a $b tras de la conversión de tipos.
Idéntico	$a === $b	true si $a es igual a $b, y son del mismo tipo de dato.
Diferente	$a != $b, $a <> $b	true si $a no es igual a $b después de la conversión de tipos.
No idéntico	$a !== $b	true si $a no es igual a $b, o si no son del mismo tipo
Menor que	$a < $b	true si $a es estrictamente menor que $b
Mayor que	$a > $b	true si $a es estrictamente mayor que $b
Menor o igual que	$a <= $b	true si $a es menor o igual que $b.
Mayor o igual que	$a >= $b	true si $a es mayor o igual que $b
Nave espacial	$a <=> $b	Devuelve -1, 0 o 1 cuando $a es respectivamente menor, igual, o mayor que $b. PHP >= 7.
Fusión de null	$a ?? $b ?? $c	El primer operando de izquierda a derecha que exista y no sea null. null si no hay valores definidos y no son null. PHP >= 7.

2.3 Operadores lógicos

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

2.4 Operadores de asignación

Nombre	Ejemplo	Resultado
Asignación	$a = $b	Asigna a $a el valor de $b
Asignación de la suma	$a += $b	Le suma a $a el valor de $b. Equivalente a $a = $a + $b
Asignación de la resta	$a -= $b	Le resta a $a el valor de $b. Equivalente a $a = $a - $b
Asignación del producto	$a *= $b	Asigna a $a el producto de $a por $b. Equivalente a $a = $a * $b
Asignación de la división	$a /= $b	Asigna a $a el conciente de $a entre $b. Equivalente a $a = $a / $b
Asignación del resto	$a %= $b	Asigna a $a el resto de dividir $a entre $b. Equivalente a $a = $a % $b
Concatenación	$a .= $b	Concatena a $a la cadena $b. Equivalente a $a = $a . $b
Incremento	$a++	Incrementa $a en una unidad. Equivalente a $a = $a + 1
Decremento	$a--	Decrementa $a en una unidad. Equivalente a $a = $a - 1

3. Formularios

Un formulario en PHP o en cualquier otro lenguaje sirve para enviar datos (ingresados por el usuario) a un servidor, con el objetivo de procesarlos y/o guardarlos en una base de datos. Puede incluir distintos campos: numéricos, fechas, textos, contraseñas, etc.

El trabajo con formularios lo estudiaremos en la unidad 4 pero veamos una introducción a ellos:

Los datos se envían vía URL con el formato: `var1=valor1&var2=valor2`

Se divide en dos pasos:

1. Generar un formulario con `action= 'archivo.php' method='GET'`
2. En el archivo .php leer los datos con `$_GET ['nombreVar']`

Se separará siempre que se pueda el código HTML del de PHP.
Por ejemplo:
El formulario lo colocamos en `saluda.html`

```
<form action="saluda.php" method="get">
    <p><label for="nombre">Nombre: </label>
    <input type="text" name="nombre" id="nombre"></p>
    <p><label for="apellido1">Primer apellido:</label>
    <input type="text" name="apellido1" id="apellido1"></p>
    <p><input type="submit" value="enviar"></p>
</form>
```

Y los datos se recogerán en `saluda.php`

```
1   <?php
2   $nombre = $_GET["nombre"];
3   $apellido1 = $_GET["apellido1"];
4
5   echo "Hola $nombre $apellido1";
6   ?>
```

Si se quisiera todo en un único archivo (que no es recomendable), sería:

```php
<form action="" method="get">
    <p><label for="nombre">Nombre: </label>
    <input type="text" name="nombre" id="nombre"></p>
    <p><label for="apellido1">Primer apellido:</label>
    <input type="text" name="apellido1" id="apellido1"></p>
    <input type="submit" value="enviar">
</form>
<p>
    <?php
    if(isset($_GET['nombre'])) {
        $nombre = $_GET["nombre"];
        $apellido1 = $_GET["apellido1"];

        echo "Hola $nombre $apellido1";
    }
    ?>
</p>
```

4. Estructuras de control

4.1 Estructuras condicionales

Las estructuras condicionales de PHP son: `if, if-else, if-elseif` y `switch`

- Condición `if`

La condición se realiza mediante la instrucción if y entre llaves, la condición que se evalúa a true o false. Se recomienda poner llaves siempre porque si no, en vez de abrir un bloque, se ejecutará solo la siguiente instrucción.

Ejemplos:

```php
<?php
$hora = 8; // La hora en formato de 24 horas
if ($hora === 8) {
    echo "Suena el despertador.";
}
echo "<br>";
if ($hora === 8)
    echo "Suena el despertador.";
?>
```

- Condición compuesta: `if-else:`

```php
<?php
$hora = 17; // La hora en formato de 24 horas
if ($hora <= 12) {
    echo "Son las " . $hora . " de la mañana";
} else {
    echo "Son las " . ($hora - 12) . " de la tarde";
}
?>
```

- Condiciones anidadas: `if-else, if-elseif:`

```php
<?php
$hora = 14; // La hora en formato de 24 horas
if ($hora === 8) {
    echo "Es la hora de desayunar.";
} else if ($hora === 14) {
    echo "Es la hora de la comida.";
} else if ($hora === 21) {
    echo "Es la hora de la cena.";
} else {
    echo "Ahora no toca comer.";
}
?>
```

- Sentencia `switch:`

La sentencia switch también permite trabajar con condiciones múltiples, teniendo un código más claro.
Importante: si trabajamos con la sentencia switch, es importante NO OLVIDAR la instrucción break.
Si no lo ponemos, se ejecutará el siguiente caso automáticamente.

```php
<?php
$hora = 14; // La hora en formato de 24 horas
switch ($hora) {
    case 9:
        echo "Es la hora de desayunar.";
        break;
    case 14:
        echo "Es la hora de la comida.";
        break;
    case 21:
        echo "Es la hora de la cena.";
        break;
    default:
        echo "Ahora no toca comer";
}
?>
```

ACTIVIDADES PROPUESTAS – *Classroom: Unidad 2 - Tarea 1*

Importante: Antes de hacer las actividades, en la carpeta DWEES de htdocs, crea una carpeta que se llame U2Tarea1

4.2 Estructuras de repetición: bucles

Las estructuras de repetición o bucles sirven para repetir un conjunto de instrucción mientras se dé una condición.
PHP cuenta con las estructuras habituales: `for`, `while` y `do-while` que tienen la misma sintáxis que en Java o C.

`for`

```php
<?php
// Bucle ascendente
for ($i = 1; $i <= 10; $i++) {
    echo "Línea " . $i;
    echo "<br>";
}

// Bucle descendente
for ($i = 10; $i >= 0; $i--) {
    echo "Línea " . $i;
    echo "<br>";
}
?>
```

`While`

```php
<?php
$i = 1;
while ($i <= 10) {
    echo "Línea " . $i;
    echo "<br>";
    $i++;
}
?>
```

Do-while

```php
<?php
do {
    $dado = rand(1, 6);
    // rand() devuelve un valor aleatorio
    echo "Tirando el dado... ";
    echo "ha salido un " . $dado . ".";
    echo "<br>";
} while ($dado != 5);
echo "¡Bien! Saco una ficha de casa.";
?>
```

Importante: Se pueden romper bucles, al igual que en Java, mediante la instrucción break.

ACTIVIDADES PROPUESTAS – *Classroom: Unidad 2 - Tarea 2*

Importante: Antes de hacer las actividades, en la carpeta DWEES de htdocs, crea una carpeta que se llame U2Tarea2

5. Arrays

Los *arrays* en PHP son una estructura muy flexible y potente.

Características de los *arrays* en PHP:
- Unifica en un solo tipo lo que en otros lenguajes se consigue con *arrays* básicos, vectores, listas o diccionarios
- Los elementos de un array se identifican por una clave que puede ser: entero o cadena.
- Los elementos guardan un orden dentro del array. Este orden está determinado por el orden de los elementos al declarar el array o al añadir nuevos.

Para declarar un array se puede utilizar la función: `array( )`

Se definen del mismo modo que Java:

```php
<?php
$frutas = array("naranja", "pera", "manzana");

$frutas2 = ["naranja", "pera", "manzana"];

$frutas3 = [];
$frutas3[0] = "naranja";
$frutas3[1] = "pera";
$frutas3[] = "manzana"; // lo añade al final

```

Podemos obtener el tamaño del `array` mediante la función `count(array).`
Para recorrer el array haremos uso del bucle `for.`

```php
<?php
$tam = count($frutas); // tamaño del array

for ($i=0; $i<count($frutas); $i++) {
    echo "Elemento $i: $frutas[$i] <br />";
}
```

Otra forma de recorrer los arrays, incluso más elegantes, es hacer uso de `foreach`. Su sintaxis es `foreach (array as elemento)`.

```php
<?php
// Mediante foreach no necesitamos saber el tamaño del array
foreach ($frutas as $fruta) {
    echo "$fruta <br />";
}
```

5.1 Arrays asociativos

Cada elemento del array es un par clave-valor. En vez de acceder por la posición, lo hacemos mediante una clave. Así pues, para cada clave se almacena un valor.
A la hora de recorrer este tipo de arrays, mediante `foreach` separamos cada elemento en una pareja `clave => valor`.

```php
<?php
$capitales = ["Italia" => "Roma",
    "Francia" => "Paris",
    "Portugal" => "Lisboa"];
$capitalFrancia = $capitales["Francia"]; // se accede al elemento por la clave, no la posición

$capitales["Alemania"] = "Berlín"; // añadimos un elemento

echo "La capital de Francia es $capitalFrancia <br />";
echo "La capital de Francia es {$capitales["Francia"]} <br />";

$capitales[] = "Madrid"; // se añade con la clave 0 !!! ¡¡¡No asignar valores sin clave!!!

foreach ($capitales as $valor) { // si recorremos un array asociativo, mostraremos los valores
    echo "$valor <br />";
}

foreach ($capitales as $pais => $ciudad) { // separamos cada elemento en clave => valor
    echo "$pais : $ciudad <br />";
}
```

5.2 Operaciones con arrays

Las operaciones más importantes que podemos realizar con arrays son:

- `print_r($array)`: muestra el contenido de todo el `$array`. Si queremos mostrar el contenido con un formato determinado, hemos de recorrer el array con `foreach`.
- `var_dump($mixed)`: muestra el contenido del elemento recibido. Muestra más información que `print_r`.
- `$elem = array_pop($array)`: elimina el último `$elemento`
- `array_push($array, $elem)`: añade un `$elemento` al final
- `$booleano = in_array($elem, $array)`: averigua si `$elem` está en el `$array`

```php
<?php
$frutas = ["naranja", "pera", "manzana"];

array_push($frutas, "piña");
print_r($frutas);

$ultFruta = array_pop($frutas);
if (in_array("piña", $frutas)) {
    echo "<p>Queda piña</p>";
} else {
    echo "<p>No queda piña</p>";
}
print_r($frutas);
```

- `$claves = array_keys($array)`: devuelve las claves del `$array` asociativo
- `$tam = count($array)`: devuelve el tamaño de `$array`
- `sort($array)`: ordena los elementos del `$array`
- `isset($array[elemento])`: indica si existe/tiene valor elemento dentro del array
- `unset($array[elemento])`: elimina el elemento del array (deja un hueco)

```php
<?php
$capitales = array("Italia" => "Roma",
"Francia" => "Paris",
"Portugal" => "Lisboa");

$paises = array_keys($capitales);
print_r($paises);
sort($paises);
print_r($paises);

unset($capitales["Francia"]);
print_r($capitales);
```

Existen muchísimas más funciones para trabajar con arrays. Nuevamente, puedes consultar toda la información en: https://www.php.net/manual/es/ref.array.php

5.3 Arrays bidimensionales

Los arrays bidimensionales consisten en un array de arrays, ya sean secuenciales o asociativos. Puede haber N dimensiones.

Ejemplo de array bidimensional:

```php
<?php
$persona["nombre"] = "Bruce Wayne";
$persona["telefonos"] = ["966 123 456", "636 636 636"]; // array de arrays ordinarios
$persona["profesion"] = ["dia" => "filántropo", "noche" => "caballero oscuro"]; // array de arrays asociativos

echo $persona['nombre']." por la noche trabaja de ".$persona['profesion']['noche'];
```

Combinando los arrays asociativos en varias dimensiones podemos almacenar la información como si fuera una tabla:

```php
<?php
$menu1 = ["Plato1" => "Macarrones con queso", "Plato2" => "Pescado asado", "Bebida" => "Coca-Cola", "Postre" => "Helado de vainilla"];
$menu2 = ["Plato1" => "Sopa", "Plato2" => "Lomo con patatas", "Bebida" => "Agua", "Postre" => "Arroz con leche"];
$menus = [$menu1, $menu2]; // creamos un array a partir de arrays asociativos

foreach ($menus as $menudeldia) {
  echo "Menú del día<br/>";

  foreach ($menudeldia as $platos => $comida) {
    echo "$platos: $comida <br/>";
  }
}

// Para acceder a un elemento concreto se anidan los corchetes
$postre0 = $menus[0]["Postre"];
```

Aunque pueda parecer una buena idea crear este tipo de estructuras, es mejor utilizar objetos conjuntamente con arrays (posiblemente arrays de otros objetos) para crear estructuras complejas que permitan modelar mejor los problemas.

ACTIVIDADES PROPUESTAS – *Classroom: Unidad 2 - Tarea 3*

Importante: Antes de hacer las actividades, en la carpeta DWEES de htdocs, crea una carpeta que se llame U2Tarea3

6. Funciones

Una función es un conjunto de instrucciones que realiza una tarea concreta.
Las funciones pueden recibir argumentos, es decir, pueden recibir los datos que necesitan para llevar a cabo su contenido.
Por ejemplo: una función que abre un fichero puede recibir como argumento el nombre del fichero que tiene que abrir.

Las funciones pueden o no devolver un valor.

En PHP no es necesario declarar el tipo de dato que devuelve una función y de hecho, puede devolver tipos de datos diferentes según el caso.
Por ejemplo: en PHP hay muchas funciones que devuelven FALSE en caso de error y no devuelven un boolean si no hay error.

Al no declararse los tipos de datos, los parámetros de las funciones no tienen tipo ni se indica el tipo de dato que devuelven. El paso de parámetros se realiza por valor, es decir, se realiza una copia de la variable.

```php
<?php
function nombreFuncion($par1, $par2, ...) {
  // código
  return $valor;
}

$resultado = nombreFuncion($arg1, $arg2, …);
?>
```

Por ejemplo:

```php
<?php
function diaSemana() {
    $semana = [ "lunes", "martes", "miércoles",
        "jueves", "viernes", "sábado", "domingo" ];
    $dia = $semana[rand(0, 6)];
    return $dia;
}

$diaCine = diaSemana();
echo "El próximo $diaCine voy al cine.";
?>
```

6.1 Parámetros por referencia

Un parámetro por referencia es aquel que modifica el contenido de una variable.
Se puede pasar una variable por referencia a una función y así hacer que la función pueda modificar la variable.

Para pasar por referencia se utiliza el operador & delante de la variable.

```php
<?php
function duplicarPorValor($argumento) {
    $argumento = $argumento * 2;
    echo "Dentro de la función: $argumento.<br>";
}
function duplicarPorReferencia(&$argumento) {
    $argumento = $argumento * 2;
    echo "Dentro de la función: $argumento.<br>";
}

$numero1 = 5;
echo "Antes de llamar: $numero1.<br>";
duplicarPorValor($numero1);
echo "Después de llamar: $numero1.<br>";
echo "<br>";

$numero2 = 7;
echo "Antes de llamar: $numero2.<br>";
duplicarPorReferencia($numero2);
echo "Después de llamar: $numero2.<br>";
?>
```

6.2 Parámetros por defecto / opcionales

Permiten asignar valores en la declaración para después, dejar el argumento en blanco.

```php
<?php
function obtenerCapital($pais = "todos") {
    $capitales = array("Italia" => "Roma",
    "Francia" => "Paris",
    "Portugal" => "Lisboa");

    if ($pais == "todos") {
        return array_values($capitales);
    } else {
        return $capitales[$pais];
    }
}

print_r(obtenerCapital());
echo "<br/>";
echo obtenerCapital("Francia");
```

6.3 Parámetros variables

Podemos tener funciones donde en la declaración no se indique la cantidad de datos de entrada.

Por ejemplo:
Usando las siguientes funciones:

- `fun_get_args();` → obtiene un array con los parámetros
- `func_num_args();` → obtiene la cantidad de parámetros recibidos
- `func_get_arg(numArgumento);` → obtiene el parámetro que ocupa la posición `numArgumento`

Estas funciones no se pueden pasar como parámetro a otra función. Para ello, debemos guardar previamente la función en una variable:

```php
<?php
function sumaParametros() {
    if (func_num_args() == 0) {
        return false;
    } else {
        $suma = 0;

        for ($i = 0; $i < func_num_args(); $i++) {
            $suma += func_get_arg($i);
        }

        return $suma;
    }
}

echo sumaParametros(1, 5, 9); // 15
?>
```

6.4 Argumentos con nombre

Además de por posición, como hemos hecho hasta ahora, también podemos pasar los argumentos con el nombre (disponible desde PHP 8.0).

Los argumentos con nombre se pasan:

```
$resultado = funcion( arg1 : valor1, arg2 : valor2);
```

Esta característica complementa los parámetros opcionales (6.2)

```php
<?php
function funcionArgumentosNombre($a, $b = 2, $c = 4) {
  echo "$a $b $c";
}
funcionArgumentosNombre(c: 3, a: 1); // "1 2 3"
```

Tanto los parámetros opcionales como los obligatorios pueden tener nombre, pero los argumentos con nombre se tienen que poner después de los que no lo tienen.

```php
<?php
funcionArgumentosNombre(1, c: 3); // "1 2 3"
```

6.5 Funciones tipadas: directiva `strict_types`

Desde la versión 7 de PHP tenemos disponible la directiva `strict_types.`
Nos permite declarar el modo estricto para tipos de escalares en un archivo de PHP.

Declarar strict types en nuestros ficheros PHP, forzará a que nuestros métodos acepten variables únicamente del tipo exacto que se declaren. En caso contrario, lanzará un TypeError.

Esto nos permitirá que nuestro desarrollo sea más robusto y que nuestros métodos no se traguen cualquier cosa. También se puede usar para el retorno, es decir, si un método tiene que devolver una cadena, se forzará a que siempre sea así, si sucede algo entre medias y devuelve un entero o un array, obtendremos un TypeError

La desventaja de esto, es que, si queremos que nuestra aplicación sea estricta y robusta, tendremos que declarar el modo estricto en todos los archivos PHP de nuestro desarrollo.

La declaración strict_types debe hacerse en la primera línea, nada más abrir PHP.
Los tipos que podemos declarar son:

`int, float, string, bool, object y array`

```php
<?php
declare(strict_types=1);

function suma(int $a, int $b) : int {
    return $a + $b;
}

$num = 33;
echo suma(10, 30);
echo suma(10, $num);
echo suma("10", 30); // error por tipificación estricta, sino daría 40
?>
```

6.6 Funciones variables

PHP admite el concepto de funciones variables.
- Esto permite asignar una función a una variable.
- El nombre de la función va **entre comillas**.
- Si un nombre de variable tiene paréntesis anexos a él, PHP buscará una función con el mismo nombre que lo evaluado por la variable, e intentará ejecutarla.

Las funciones variables se pueden usar para implementar llamadas de retorno, tablas de funciones y así sucesivamente.

Las funciones variables **no funcionarán** con constructores de lenguaje como `echo`, `print`, `unset()`, `isset()`, `empty()`, `include`, `require` y similares.
Utilice **funciones de envoltura** para hacer uso de cualquiera de estos constructores como funciones variables.

```php
<?php
$miFuncionSuma = "suma";
echo $miFuncionSuma(3,4); // invoca a la función suma
?>
```

6.7 Instrucción include

La instrucción `include` la podemos usar para: usar bibliotecas o crear plantillas. Veámoslo:

Bibliotecas

Podemos agrupar un conjunto de funciones en un archivo para usarlas posteriormente. Se hace con:

- `include(archivo); / include_once(archivo);`
- `require(archivo); / require_once(archivo);`

Si no encuentra el archivo, `require` lanza un error fatal, `include` lo ignora.
Las funciones `_once` sólo se cargan una vez, si ya ha sido incluida previamente, no lo vuelve a hacer, evitando bucles.

Por ejemplo, colocamos las funciones: suma y resta en el archivo `biblioteca.php`:

```php
<?php
function suma(int $a, int $b) : int {
    return $a + $b;
}

function resta(int $a, int $b) : int {
    return $a - $b;
}
?>
```

Y posteriormente, en otro archivo:

```php
<?php
include_once("biblioteca.php");
echo suma(10,20);
echo resta(40,20);
?>
```

Plantillas

También podemos separar fragmentos de código PHP/HTML que queramos reutilizar en nuestros sitios web y crear un sistema muy sencillo de plantillas.

Por ejemplo:

Queremos separar una página en 3 partes:

La parte superior en `encabezado.php`

```php
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title><?= $titulo ?></title>
</head>
<body>
```

La parte de abajo, que solo contendrá html, y la colocamos en `pie.html`

```php
<footer>Pie de página</footer>
</body>
</html>
```

Y luego nos centramos solo en el contenido que cambia en `pagina.php`

```php
<?php
$titulo = "Página con includes";
include("encabezado.php");
?>
<h1><?= $titulo ?></h1>
<?php
include("pie.html");
?>
```

ACTIVIDADES PROPUESTAS – *Classroom: Unidad 2 - Tarea 4*

Importante: Antes de hacer las actividades, en la carpeta DWEES de htdocs, crea una carpeta que se llame U2Tarea4

7. Funciones predefinidas

PHP cuenta con una gran cantidad de funciones predefinidas para las tareas más habituales. Entre las más utilizadas están las relacionadas con las variables.
En este enlace se pueden ver todas https://www.php.net/manual/es/indexes.functions.php

En la siguiente tabla se muestran las funciones más útiles en PHP:

Funciones de variables	
isset($var)	TRUE si la variable está inicializada y no es NULL
is_null($var)	TRUE si la variable es NULL
empty($var)	TRUE si la variable no está inicializada o su valor es FALSE
is_int($var), is_float($var), is_bool($var), is_array($var)	Para comprobar el tipo de dato de $var
intval($var), floatval($var), boolvar($var), strval($var)	Para obtener el valor de $var como otro tipo de dato
Funciones de cadenas	
strlen($cad)	Devuelve la longitud de $cad
explode($cad, $token)	Parte una cadena utilizando $token como separador. Devuelve un *array* de cadenas
implode($token, $array)	Crea una cadena larga a partir de un *array* de cadenas, entre cadena y cadena se introduce $token
strcmp($cad1, $cad2)	Compara las dos cadenas. Devuelve 0 si son iguales, -1 si $cad1 es menor y 1si $cad1 es mayor
strtolower($cad), strtoupper($cad)	Devuelven $cad en mayúsculas o minúsculas, respectivamente
str($cad1, $cad2)	Busca la primera ocurrencia de $cad2 en $cad1. Si no aparece devuelve FALSE, si aparece devuelve $cad1 desde donde comienza la ocurrencia
Funciones de *arrays*	
ksort($arr), krsort($arr)	Ordena el *array* por clave en orden ascendente o descendente
sort($arr), rsort($arr)	Ordena el *array* por valor en orden ascendente o descendente
array_values($arr)	Devuelve los valores de $arr
array_keys($arr)	Devuelve las claves de $arr
array_key_exists($arr, $cla)	Devuelve verdadero si algún elemento de $arr tiene clave $cla
count($arr)	Devuelve el número de elementos del *array*

7.1 Cadenas

Ya hemos visto que se pueden crear con comillas simples ('', sin interpretación) o comillas dobles ("", interpretan el contenido y las secuencias de escape \n, \t, \$, {, … - *magic quotes*)

Además de `echo`, podemos mostrar las cadenas mediante la función `printf`. Esta función viene heredada del lenguaje C, y en la cadena se indica el tipo de dato a formatear y genera una salida formateada.
Si quiero guardar el resultado en una variable, podemos utilizar `sprintf`

```php
<?php
$num = 33;
$nombre = "Larry Bird";
printf("%s llevaba el número %d", $nombre, $num); // %d -> número decimal, %s -> string
$frase = sprintf("%s llevaba el número %d", $nombre, $num);
echo $frase
?>
```

Más ejemplos en https://www.w3schools.com/php/func_string_printf.asp

7.1.1 Operaciones básicas

Todas las funciones se pueden consultar en https://www.php.net/manual/es/ref.strings.php

Las más importantes son:

- `strlen:` obtiene la longitud de una cadena y devuelve un número entero
- `substr:` devuelve una subcadena de la cadena original
- `str_replace:` reemplaza caracteres en una cadena
- `strtolower y strtoupper:` Transforman una cadena de caracteres en la misma cadena en minúsculas o mayúsculas respectivamente.

Ejemplo:

```php
<?php
$cadena = "El caballero oscuro";
$tam = strlen($cadena);
echo "La longitud de '$cadena' es: $tam <br />";

$oscuro = substr($cadena, 13); // desde 13 al final
$caba = substr($cadena, 3, 4); // desde 3, 4 letras
$katman = str_replace("c", "k", $cadena);
echo "$oscuro $caba ahora es $katman";

echo "Grande ".strtoupper($cadena);
?>
```

Si queremos **trabajar con caracteres ASCII** de forma individual, son útiles las funciones:

- `chr:` obtiene el carácter a partir de un ASCII
- `ord:` obtiene el ASCII de un carácter

Ejemplo:

```php
<?php
function despues(string $letra): string {
    $asciiLetra = ord($letra);
    return chr($asciiLetra + 1);
}

echo despues("B");
?>
```

Si queremos **limpiar cadenas**, tenemos las funciones:

- `trim:` elimina los espacios al principio y al final
- `ltrim / rtrim o chop`: Elimina los espacios iniciales / finales de una cadena.
- `str_pad`: rellena la cadena hasta una longitud especificada y con el carácter o caracteres especificados.

Ejemplo:

```php
<?php
$cadena = " Programando en PHP ";
$limpia = trim($cadena); // "Programando en PHP"

$sucia = str_pad($limpia, 23, "."); // "Programando en PHP....."
?>
```

7.1.2 Comparación de cadenas

La **comparación de cadenas** puede ser con conversión de tipos mediante == o estricta con ===. También funcionan los operadores < y > si ambas son cadenas. Al comparar cadenas con valores numéricos podemos utilizar:

- `strcmp:` 0 iguales, <0 si a<b o >0 si a>b
- `strcasecmp`: las pasa a minúsculas y compara
- `strncmp / strncasecmp`: compara los N primeros caracteres
- `strnatcmp:` comparaciones naturales

```php
<?php
$frase1 = "Alfa";
$frase2 = "Alfa";
$frase3 = "Beta";
$frase4 = "Alfa5";
$frase5 = "Alfa10";

var_dump( $frase1 == $frase2 ); // true
var_dump( $frase1 === $frase2 ); // true
var_dump( strcmp($frase1, $frase2) ); // 0
var_dump( strncmp($frase1, $frase5, 3) ); // 0
var_dump( $frase2 < $frase3 ); // true
var_dump( strcmp($frase2, $frase3) ); // -1
var_dump( $frase4 < $frase5 ); // false
var_dump( strcmp($frase4, $frase5) ); // 4 → f4 > f5
var_dump( strnatcmp($frase4, $frase5) ); // -1 → f4 < f5
?>
```

7.1.3 Buscando en cadenas

Si lo que queremos es buscar dentro de una cadena, tenemos:

- strpos / strrpos: busca en una cadena y devuelve la posición de la primera/última ocurrencia.
- strstr / strchr (alias): busca una cadena y devuelve la subcadena a partir de donde la ha encontrado
- stristr: ignora las mayúsculas

```php
<?php
$frase = "Quien busca encuentra, eso dicen, a veces";
$pos1 = strpos($frase, ","); // encuentra la primera coma
$pos2 = strrpos($frase, ","); // encuentra la última coma
$trasComa = strstr($frase, ","); // ", eso dicen, a veces"
?>
```

Si queremos averiguar qué contiene las cadenas, tenemos un conjunto de funciones de comprobaciones de tipo, se conocen como las funciones *ctype* que devuelven un booleano:

- ctype_alpha → letras
- ctype_alnum → alfanuméricos
- ctype_digit → dígitos
- ctype_punct → caracteres de puntuación, sin espacios
- ctype_space → son espacios, tabulador, salto de línea

```php
<?php
$prueba1 = "hola";
$prueba2 = "hola33";
$prueba3 = "33";
$prueba4 = ",.()[]";
$prueba5 = " ,.()[]";

echo ctype_alpha($prueba1)."<br>"; // true
echo ctype_alnum($prueba2)."<br>"; // true
echo ctype_digit($prueba3)."<br>"; // true
echo ctype_punct($prueba4)."<br>"; // true
echo ctype_space($prueba5)."<br>"; // false
echo ctype_space($prueba5[0])."<br>"; // true
?>
```

7.1.4 Subcadenas

Si queremos **romper las cadenas en trozos**, tenemos:

- explode: convierte en array la cadena mediante un separador.
- implode / join: pasa un array a cadena con un separador
- str_split / chunk_split: pasa una cadena a una array/cadena cada X caracteres

```php
<?php
$frase = "Quien busca encuentra, eso dicen, a veces";
$partes = explode(",", $frase);

$ciudades = ["Elche", "Aspe", "Alicante"];
$cadenaCiudades = implode(">", $ciudades);

$partes3cadena = chunk_split($frase, 3);
// Qui
// en
// bus
// ca
// ...
$partes3array = str_split($frase, 3);
// ["Qui", "en ", "bus", "ca ", "enc", …]
?>
```

Si queremos **trabajar con tokens:**

- strtok(cadena, separador)
- y dentro del bucle: strtok(separador)

Para **separarla** con base al formato:

- sscanf: al revés que sprintf, crea un array a partir de la cadena y el patrón.

Finalmente, otras operaciones que podemos realizar para trabajar con subcadenas son:

- substr_count: número de veces que aparece la subcadena dentro de la cadena
- substr_replace: reemplaza parte de la cadena a partir de su posición, y opcionalmente, longitud

```php
<?php
$batman = "Bruce Wayne es Batman";
$empresa = substr($batman, 6, 5); // Wayne
$bes = substr_count($batman, "B"); // 2
// Bruce Wayne es camarero
$camarero1 = substr_replace($batman, "camarero", 15);
$camarero2 = substr_replace($batman, "camarero", -6); // quita 6 desde el final
// Bruno es Batman
$bruno = substr_replace($batman, "Bruno", 0, 11);
?>
```

También disponemos de una serie de funciones que facilitan las **codificaciones desde y hacia HTML:**

- htmlentities: convierte a entidades HTML, por ejemplo, á por á, ñ por ñ, < por <, etc..
- htmlspecialchars: idem pero solo con los caracteres especiales (&, ", ', <, >, ...)
- striptags: elimina etiquetas HTML.
- nl2br: cambia saltos de línea por
.
- rawurlencode / rawurldecode: codifica/decodifica una URL (espacios, ...).

Estas funciones se verán en la unidad 4.

7.2 Funciones matemáticas

Disponemos tanto de constantes como funciones ya definidas para trabajar con **operaciones matemáticas**: https://www.php.net/manual/es/ref.math.php

- Constantes ya definidas

  `M_PI, M_E, M_EULER, M_LN2, M_LOG2E`

  `PHP_INT_MAX, PHP_FLOAT_MAX`

- Funciones de cálculo

  `pow, sqrt, log, decbin, bindec, decoct, dechex, base_convert, max min`

- Funciones trigonométricas

  `sin, cos, tan, deg2rad, rad2deg`

- Funciones para trabajar con números aleatorios

  `rand, mt_rand (más rápida)`

Aunque la mayoría de ellas son muy específicas de problemas matemáticos / estadísticos, es muy común que tengamos que redondear y/o formatear los cálculos antes de mostrarlos al usuario.

Mediante la función `number_format(numero, cantidadDecimales, separadorDecimales, separadorMiles)` podemos pasar números a cadena con decimales y/o separadores de decimales y/o de miles.

```php
<?php
$nf = 1234.5678;
echo number_format($nf, 2); // 1,234.57
echo number_format($nf, 2, "M", "#"); // 1#234M57
?>
```

Para redondear, tenemos abs para el valor absoluto y round para redondear, ceil para aproximación por exceso y floor por defecto.

```php
<?php
$num = 7.7;
$siete = floor($num);
$ocho = ceil($num);

$otro = 4.49;
$cuatro = round($otro);
$cuatrocinco = round($otro, 1);
$cinco = round($cuatrocinco);
?>
```

7.3 Tipos de datos

Finalmente, para realizar conversiones de datos o si queremos trabajar con tipos de datos, tenemos las siguientes funciones:

- floatval, intval, strval: devuelve una variable del tipo de la función indicada
- settype: fuerza la conversión
- gettype: obtiene el tipo
- is_int, is_float, is_string, is_array, is_object: devuelve un booleano a partir del tipo recibido

```php
<?php
$uno = 1;
var_dump(is_int($uno)); // true
$unofloat = floatval($uno);
settype($uno, "string");
var_dump(is_int($uno)); // false
var_dump(is_string($uno)); // true
settype($uno, "float");
var_dump(is_int($uno)); // false
var_dump(is_float($uno)); // true
var_dump(is_int(intval($uno))); // true
?>
```

ACTIVIDADES PROPUESTAS – *Classroom: Unidad 2 - Tarea 5*

Importante: Antes de hacer las actividades, en la carpeta DWEES de htdocs, crea una carpeta que se llame U2Tarea5