unidad didáctica 2

El lenguaje PHP

Índice

1. duración y criterios de evaluación

Duración estimada: 24 sesiones

Resultado de aprendizaje y criterios de evaluación:

2. Escribe sentencias ejecutables por un servidor Web reconociendo y aplicando procedimientos de integración del código en lenguajes de marcas.

   a) Se han reconocido los mecanismos de generación de páginas Web a partir de lenguajes de marcas con código embebido.

   b) Se han identificado las principales tecnologías asociadas.

   c) Se han utilizado etiquetas para la inclusión de código en el lenguaje de marcas.

   d) Se ha reconocido la sintaxis del lenguaje de programación que se ha de utilizar.

   e) Se han escrito sentencias simples y se han comprobado sus efectos en el documento resultante.

   f) Se han utilizado directivas para modificar el comportamiento predeterminado.

   g) Se han utilizado los distintos tipos de variables y operadores disponibles en el lenguaje.

   h) Se han identificado los ámbitos de utilización de las variables.

3. Escribe bloques de sentencias embebidos en lenguajes de marcas, seleccionando y utilizando las estructuras de programación.

   a) Se han utilizado mecanismos de decisión en la creación de bloques de sentencias.

   b) Se han utilizado bucles y se ha verificado su funcionamiento.

   c) Se han utilizado "arrays" para almacenar y recuperar conjuntos de datos.

   d) Se han creado y utilizado funciones.

   e) Se han utilizado formularios web para interactuar con el usuario del navegador Web.

   f) Se han empleado métodos para recuperar la información introducida en el formulario.

   g) Se han añadido comentarios al código.

2. PHP

- Acrónimo de **P**ersonal **H**ome **P**age.
- Lenguaje de propósito general, aunque su fuerte es el desarollo web.
- Sintaxis similar a C / Java.
- El código se ejecuta en el servidor (en Apache mediante mod_php)
- El cliente recibe el resultado generado tras interpretar el código en el servidor.
- El código se almacena en archivo con extensión `.php`.

La última versión es la 8.2.8, de Julio de 2023. Además de numerosas nuevas funcionalidades que iremos viendo durante el curso, tiene más de dos veces mejor rendimiento que PHP5.

Su documentación es extensa y está traducida: https://www.php.net/manual/es/.

2.1. código embebido

Los bloques de código se escriben entre `<?php` y `?>`.

Las sentencias se separan mediante `;`.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>PHP fácil</title>
</head>
<body>
  <!-- Muestra una frase con HTML -->
  Hola mundo<br>
  <!-- Muestra una frase con PHP -->
  <?php echo "Es muy fácil programar en PHP."; ?>
</body>
</html>
```

> **sólo etiquetas de apertura:**
>
> Si nuestro código **sólo va a contener código PHP** y nada de html, como por ejemplo, cuando codifiquemos **clases o interfaces**, sólo pondremos la etiqueta de apertura, para así indicar que es una archivo de php puro.

2.2. Generando código

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

- `<?= expresión ?>`
- `print (expresión);`

Las que vamos a utilizar son:

- `echo` cuando lo hagamos dentro de un bloque de instrucciones y
- `<?=` cuando sólo vayamos a mostrar el valor de una variable dentro de un fragmento HTML.

```
<!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><br>
  <p><?php print("Este texto se mostrará en la página web."); ?></p>
</body>
</html>
```

2.3. comentarios

Podemos utilizar comentarios de una línea o de bloque:

```
<?php
// Este es un comentario de una sola línea<br>

/*
  Este es
  un comentario
  que ocupa
  varias líneas
*/
?>
```

Teclas rápidas en VS Code: **Ctrl + Shift + 7**.

2.4. errores

Si hay un error de ejecución, se produce un *Fatal Error*.

```
Fatal error: Uncaught Error: Call to undefined function plint() in
C:\xampp\htdocs\202echo.php:11
 Stack trace:
 #0 {main}
    thrown in C:\xampp\htdocs\202echo.php on line 11
```

Desde PHP 5 se lanzan como una excepción. Más adelante veremos el uso de `try` / `catch`.

2.5. variables

- Comienzan por `$`, por ejemplo `$nombre`, seguido de un caracter en letra minúscula (**recomendación**) o guión bajo _. Luego ya se pueden poner números.
- Son *case sensitive*: `$var != $vAR`.
- No se declara su tipo, el tipado es dinámico. Se asigna en tiempo de ejecución dependiendo del valor asignado.
- Conveniente: hay que inicializarlas, sino dan error.

```php
<?php
      $nombre = "Marta";
      $nombreCompleto = "Marta López";
      $numero = 123;
      $numero2 = 456;
      $pi = 3.14;
      $suerte = true;
      $sinValor;
      $_1 = "el primero";

      echo $sinValor;
?>
```

tipos:

Aunque a priori no hay tipos de datos, internamente PHP trabaja con:

- cuatro tipos escalares: *boolean*, *integer*, *float* y *string,* y
- cuatro tipos compuestos: *array*, *object*, *callable* e *iterable*.
- Existe un tipo especial para *null* (más información en http://php.net/manual/es/language.types.null.php).

2.6. constantes

Son variables cuyo valor no varían.

Existen dos posibilidades:

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

```php
<?php
      define("PI", 3.1416);
      const IVA = 0.21;

      echo PI, " ", IVA; // No se pone el símbolo dolar
?>
```

- Se declaran siempre en MAYÚSCULAS.
- Hay un conjunto de constantes ya predefinidas, también conocidas como *magic constants*: https://www.php.net/manual/es/language.constants.predefined.php.

3. operadores

3.1. trabajando con formularios

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

Por ejemplo: `ejemplo.php?nombre=Bruce&apellido1=Wayne`

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']`

Vamos a separar siempre que podamos 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 recogemos los datos en `saluda.php`:

```
<?php
	$nombre = $_GET["nombre"];
	$apellido1 = $_GET["apellido1"];

	echo "Hola $nombre $apellido1";
?>
```

Si lo quisiéramos realizar todo en un único archivo (lo cual no es recomendable), podemos hacerlo así:

```
<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";
		}
```

El trabajo con formularios lo estudiaremos en profundidad en la unidad 4, y veremos que además de `GET`, podemos enviar los datos con `POST`.

3.2. condiciones

3.2.1. if

La condición simple se realiza mediante la instrucción `if`. Entre paréntesis se pone la condición que se evalua a true o false. Si no se ponen llaves, en vez de abrir un bloque, se ejecutará sólo la siguiente instrucción.

> **siempre llaves**
>
> Es recomendable poner llaves siempre aunque en el momento de codificar sólo haya una única instrucción. De este modo, se queda preparado para añadir más contenido en el futuro sin provocar *bugs*.

```
<?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.";
?>
```

Las condiciones compuestas mediante `if-else`:

```
<?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";
	}
?>
```

Las condiciones anidadas mediante `if-else if-else`:

```
<?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.";
	}
?>
```

3.2.2. switch

La sentencia `switch` también permite trabajar con condiciones múltiples:

```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";
	}
?>
```

> **no olvides el break**
>
> Un error muy común es olvidar la instrucción `break` tras cada caso. Si no lo ponemos, ejecutará el siguiente caso automáticamente.

3.2.3. operador ternario

Finalmente, también tenemos el operador ternario `condición ? valorTrue : valorFalse`:

```php
<?php
	$hora = 14;
	$formato = ($hora > 12) ? 24 : 12;
	echo "El formato es de $formato horas"
?>
```

Si queremos comprobar si una variable tiene valor y si no darle un valor determinado, usaremos el operador `?:` (se conoce como el **operador Elvis** - https://en.wikipedia.org/wiki/Elvis_operator) con la sintáxis `expresión ?: valorSiVacio`.

```php
<?php
	$nombre = "";
	echo "Nombre: " . ($nombre ?: "desconocido") . PHP_EOL;

	// se envía a través de un formulario:
	// $nombre = "";
	// $nombre = $_GET['nombre'] ?: "desconocido";
?>
```

3.3. bucles

3.3.1. while

Mediante la instrucción `while`:

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

3.3.2. do-while

Mediante la instrucción `do-while`:

```
<?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.";
?>
```

3.3.3. for

Mediante la instrucción `for`:

```
<?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>";
	}
?>
```

Más adelante estudiaremos el bucle `foreach` para recorrer arrays.

PHP, del mismo modo que Java y C, permite romper los bucles mediante la instrucción `break`. A su vez, `continue` permite saltar a la siguiente iteración.

> **si puedes, evita break y continue**
>
> Personalmente, no me gusta su uso. Prefiero el uso de variables flag para controlar la salida

```php
<?php
$salir = false;
for ($i = 1; $i <= 10 && !$salir; $i++) {
  if ($i === 5) {
    echo "Salgo cuando i=5";
    $salir = true;
  }
}
?>
```

4. arrays

Para almacenar datos compuestos, podemos utilizar tanto arrays sencillos como arrays asociativos (similares a un **mapa**). En realidad todos los arrays son mapas ordenados compuestos de pares *clave-valor*.

> **cuidado con mezclar tipos**
>
> Como el tipado es dinámico, nuestros arrays pueden contener datos de diferentes tipos. No se recomienda mezclar los tipos.

Del mismo modo que Java, se definen mediante corchetes, son *0-index*, y se puede asignar un valor a un posición determinada:

```php
<?php
	// opción 1
	$frutas = array("naranja", "pera", "manzana");

	// opción 2
	$frutas2 = ["naranja", "pera", "manzana"];

	// opción 3
	$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 de un 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 elegante, 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 $elemento) {
		echo "$elemento <br />";
	}
```

4.1. arrays asociativos

Cada elemento 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
   $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 />";
   }
```

4.2. arrays bidimensionales

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

```
<?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
  $menu1 = ["Plato1" => "Macarrones con queso",
                        "Plato2" => "Pescado asado",
                        "Bebida" => "Coca-Cola",
                        "Postre" => "Helado de vainilla"];
```

```
                        "Bebida" => "Agua",
                        "Postre" => "Arroz con leche"];
$menus = [$menu1, $menu2]; // creamos un array a partir de arrays asociativos

echo "Menú del día<br/>";
$cont = 0;
foreach ($menus as $menudeldia) {
      echo "<br />Menú ".++$cont."<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.

5. funciones

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

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

Por ejemplo:

```
<?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.";
?>
```

5.1. parámetros por referencia

Si queremos pasar un parámetro por referencia, en la declaración de la función, indicaremos los parámetros mediante el operador `&` para indicar la dirección de memoria de la variable.

```
<?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 "Paso de parámetros por valor:<br/>";
      echo "Antes de llamar: $numero1.<br>";
      duplicarPorValor($numero1);
      echo "Después de llamar: $numero1.<br>";
      echo "<br>";
```

```
      echo "Paso de parámetros por referencia:<br/>";
      echo "Antes de llamar: $numero2.<br>";
      duplicarPorReferencia($numero2);
      echo "Después de llamar: $numero2.<br>";
?>
```

5.2. parámetros por defecto / opcionales

Permiten asignar valores en la declaración, y posteriormente, dejar el argumento en blanco.

```
<?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");
```

En el caso de convivir con otro tipo de parámetros, los parámetros que tienen el valor asignado por defecto siempre se colocan al final.

```
<?php
   function saluda($nombre, $prefijo = "Sr") {
        echo "Hola ".$prefijo." ".$nombre;
   }

   saluda("Aitor", "Mr");
   saluda("Aitor");
   saluda("Marina", "Srta");
```

5.3. parámetros variables

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

- `$arrayArgs = func_get_args();` → Obtiene un array con los parámetros.
- `$cantidad = func_num_args();` → Obtiene la cantidad de parámetros recibidos.
- `$valor = func_get_arg(numArgumento);` → Obtiene el parámetro que ocupa la posición `numArgumento`.

Estas funciones no se pueden pasar como parámetro a otra función (como funciones variable, que veremos más adelante). Para ello, debemos guardar previamente la función en una variable.

```
            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
?>
```

Desde PHP 5.6, se puede utilizar el operador `...` (*variadics*) el cual "disfraza" los parámetros como un array:

```
<?php
      function sumaParametrosMejor(...$numeros) {
            if (count($numeros) == 0) {
                  return false;
            } else {
                  $suma = 0;

                  foreach ($numeros as $num) {
                        $suma += $num;
                  }

                  return $suma;
            }
      }

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

más usos de ...

También se puede utilizar para dividir un array en variables separadas para proporcionar argumentos:

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

      echo suma(...[1, 5])."<br />";

      $a = [1, 5];
      echo suma(...$a);
?>
```

Desde PHP 8.0 podemos pasar los argumentos con el nombre (además de por posición, como hemos hecho hasta ahora). Los argumentos con nombre se pasan poniendo el nombre como prefijo del parámetro separado por dos puntos: `$resultado = funcion( arg1 : valor1, arg2 : valor2 );`

Esta característica complementa los parámetros opcionales permitiéndonos saltar su valor:

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

5.5. funciones tipadas

Desde PHP7 en las funciones, tanto los parámetros como su devolución, permiten la definición de tipos. Esto se conoce como `strict_types` (tipificación estricta) y hay que definirlo en la primera línea de cada archivo `.php` para que el propio interprete PHP compruebe los tipos y lance errores si los tipos son incorrectos, mediante la sentencia:

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

Así pues, vamos a definir los tipos de los parámetros y de los valores devueltos mediante los tipos: `int`, `float`, `string`, `bool`, `object` y `array`.

Si una función no devuelve nada se indica mediante el tipo `void`.

```
<?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
?>
```

5.6. alcance

Las variables definidas fuera de las funciones tienen alcance **global**: accesibles desde cualquier función.

Los parámetros de una función y las variables declaradas dentro de una función (se conocen como variables **locales**) sólo son accesibles desde dentro de la misma función → alcance de **función**.

En caso de conflicto, tienen prioridad las variables locales. Para evitar el conflicto, dentro de la función, podemos declarar la variable como `global`.

5.6.1. alcance local

```php
<?php
	function miCiudad() {
		$ciudad = "Elche";
		echo "Dentro de la función: $ciudad.<br>";
	}

	$ciudad = "Alicante";
	echo "Antes de la función: $ciudad.<br>";
	miCiudad();
	echo "Después de la función: $ciudad.<br>"
?>
```

5.6.2. alcance global

```php
<?php
	function miCiudad() {
		global $ciudad;
		$ciudad = "Elche";
		echo "Dentro de la función: $ciudad.<br>";
	}

	$ciudad = "Alicante";
	echo "Antes de llamar: $ciudad.<br>";
	miCiudad();
	echo "Después de llamar: $ciudad.<br>"
?>
```

no globales

Hay que evitar el uso de variables globales dentro de las funciones. En el caso de necesitarlas, es mejor pasarlas como parámetro a las funciones.

5.7. funciones variable

- Permite asignar una función a una variable.
- Nombre de la función entre comillas.
- Si una variable va seguida de paréntesis, PHP buscará una función con su valor.

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

funciones anónimas

PHP permite la definición y uso de funciones anónimas, es decir, funciones que no tienen nombre, y se utilizan principalmente para gestionar los *callbacks*. Este tipo de funciones se utiliza mucho en **Javascript** para gestionar los eventos y promesas.

```php
<?php
	$anonima = function() {
		echo "Hola";
	};
	$anonima();

	$anonimaConParametro = function($nombre) {
		echo "Hola ".$nombre;
	};
	$anonimaConParametro("Aitor");

	// Uso de variables externas a la función anónima --> `use`
	$mensaje = "Hola";
	$miClosure = function() use ($mensaje) {
		echo $mensaje;
	};
	$miClosure();

	// Uso de parámetros
	$holaPHP = function($arg) use ($mensaje) {
		echo $mensaje." ".$arg;
	};
	$holaPHP("PHP");
?>
```

Desde PHP 7.4 se han introducido las funciones flecha (*arrow functions*) para simplificar su definición y uso.

Tenéis más información sobre funciones anónimas y flecha en el siguiente artículo (en inglés): Funciones anónimas y flecha en PHP.

5.8. biblioteca de funciones

Podemos agrupar un conjunto de funciones en un archivo, para permitir su reutilización. Posteriormente, se incluye 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 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);
?>
```

5.9. plantillas mediante include

Mediante el uso de la instrucción `include` 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, vamos a separar una página en tres partes, primero 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, por ejemplo, solo va a contener HTML y la colocamos en `pie.html`:

```php
   <footer>Profesor/a X</footer>
</body>
</html>
```

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

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

<h1><?= $titulo ?></h1>

<?php
	include("pie.html");
```

6. funciones predefinidas

El lenguaje ofrece un abanico de funciones ya definidas, agrupadas por su funcionalidad: https://www.php.net/manual/es/funcref.php.

6.1. 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*):

```
<?php
      "Me llamo $nombre"
      "Son 30 {$moneda}s"
?>
```

Se acceden a los caracteres como si fuera un array.

```
<?php
      $cadena = “Yo soy Batman”;
      $ygriega = $cadena[0];
?>
```

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 queremos guardar el resultado en una variable, podemos utilizar `sprintf`.

```
<?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;
?>
```

Tenemos muchos más ejemplos en https://www.w3schools.com/php/func_string_printf.asp

6.1.1. 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.

```
<?php
```

```
    $tam = strlen($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.

```
<?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 cadenas hasta una longitud especificada y con el carácter o caracteres especificados.

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

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

6.1.2. comparando y buscando

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
        $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
?>
```

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 contienen 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 = ",.()[]";
```

```
        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
?>
```

6.1.3. Trabajando con 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
        $frase = "Quien busca encuentra, eso dicen, a veces";

        $partes = explode(",", $frase);
        // SALIDA: ["Quien busca encuentra","eso dicen","a veces"]

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

        $partes3cadena = chunk_split($frase, 3);
        // SALIDA: un string
        // "Qui en bus ca enc uen tra , e so dic en, a vec es "

        $partes3array = str_split($frase, 3);
        // SALIDA: un array
        // ["Qui", "en ", "bus", "ca ", "enc", …]
?>
```

Si queremos trabajar con tokens:

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

Finalmente, para separarla en 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 `&aacute;`, `ñ` por `&ntilde;`, `<` por `<`, etc..
- `htmlspecialchars`: idem pero solo con los caracteres especiales (`&`, `"`, `'`, `<`, `>`, ...)
- `striptags`: elimina etiquetas HTML.
- `nl2br`: cambia saltos de línea por `<br />`.
- `rawurlencode` / `rawurldecode`: codifica/decodifica una URL (espacios, ...).

Estas funciones las utilizaremos en la unidad 4.- Programación Web.

6.2. 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);   // SALIDA: 7
	$ocho = ceil($num);   // SALIDA: 8

	$otro = 4.49;
	$cuatro = round($otro);   // SALIDA: 4
	$cuatrocinco = round($otro, 1);   // SALIDA: 4.5
	$cinco = round($cuatrocinco);   // SALIDA: 5
?>
```

6.3. 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
?>
```

7. uso de regexp

1. Regexp (Expresiones regulares) son una secuencia de caracteres que define un patrón de búsqueda -> regex

- Muy útil tanto para buscar como para reemplazar texto (que es el tipo de datos más común en la web).
- Algunos caracteres tienen un comportamiento especial (por ejemplo el punto), el resto se comportan como lo que son (por ejemplo una a).
- Toda expresión regular se comporta como un autómata finito, es decir, tiene un flujo de análisis de izquierda a derecha.

2. Caracteres Especiales

- Cualquier caracter (`.`) vs un caracter concreto (por ejemplo la *a*) vs un caracter de un rango (por ejemplo *[abc]*).
- Los corchetes `[]` también sirven para que busque el caracter literal o usar rangos con el símbolo `-` . P.ej `[a.]`, que buscaría el caracter . o `[a-d]` que buscaría *a, b,c* o *d*.
- Los caracteres cuantificadores (`?`, `+` , `*`) se usan para expresar la cantidad de caracteres que pueden aparecer. Se puede hasta especificar la cantidad mínima y máxima concreta con caracter llave. P.ej. {3,10}.
- La alternancia se puede usar con corchete con el carácter `/`. Pero / es más útil porque sirve para expresiones concretas, p.ej (*hola/adios*).
- Se pueden definir principio (`^`) y fin (`$`) de línea. `^` tiene un comportamiento especial dentro de corchetes (es el operador *not*).
- Para agrupar se usan paréntesis `()`. Cada grupo a veces se puede numerar o ponerle un nombre (depende de la aplicación).

3. Algunos tutoriales muy buenos:

- Uno completo en vídeo (pero mejor no usar agrupaciones de caracteres):

- Uno para leer
- Uno del soporte de ayuda de Google

4. Diseñar y usar regexp:

- Hay muchos ejemplos muy útiles (por ejemplo aquí).
- Si ya te quieres poner a profundizar, puedes revisar REGEX101.
- En PHP se puede usar de diferentes maneras (puedes probar).

5. Ojo que hay **ligeras variaciones** entre el uso de regexp en diferentes aplicaciones sobre todo con la agrupación de caracteres genéricos (dígitos, palabras, líneas, etc).

aprender **expresiones regulares**

Hay millones de ejemplos, pero en esto 10 ejercicios. ¿Qué cadenas de texto encontrarían las siguientes regexp?

1. hola

2. h.?a

4. [2-4]?[3-9ag]
5. aju{2,8}m[-.]z
6. (ad)+juju.[0-9]?-
7. (ho|la)?[4-7zu]
8. [([0-4][0-9]|5[0-2])[0-9]{3} . Códigos Postales España
9. ^.{3,32}#[0-9]{4}$. Usuario Discord
10. (b25[0-5]|b2[0-4][0-9]|b[01]?[0-9][0-9]?)(.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)){3} . Dirección ipv4. b es un delimitador de palabra «boundary».

8. referencias

- Manual de PHP
- PHP en 2020, por Jesús Amieiro
- Apuntes de PHP de Bartolomé Sintes, profesor del IES Abastos de Valencia
- Guía de Estilo - PSR
- PHP - La manera correcta