Unión Europea

Fondo Social Europeo

El FSE invierte en tu futuro

PHP

Elementos básicos del lenguaje

Pepe

GENERALITAT VALENCIANA

Conselleria d'Educació, Investigació, Cultura i Esport

GOBIERNO DE ESPAÑA

MINISTERIO DE EDUCACIÓN, CULTURA Y DEPORTE

Continguts

1 Pròleg

```
<?php
    ...código php...
?>
```

Dentro de los símbolos `<?php` (de apertura) y `?>` (de cierre) pondremos las instrucciones que consideremos. Todo esto puede ocupar una sola línea (para instrucciones simples y cortas) o dividir el código en varias líneas, una por instrucción.

2 Sintaxis básica y comentarios

Dentro de cada bloque de código PHP que escribamos en una página, debemos tener en cuenta que:

- Cada instrucción en PHP termina siempre en un punto y coma (;) que la separa de la siguiente, aunque estén en líneas separadas.
- Los comentarios en PHP se pueden poner de diversas formas.

```
/* Comentario que puede ocupar varias líneas */
// Comentario de una línea
# Otro ejemplo de comentario de una línea
```

Veamos a continuación un ejemplo sencillo de código PHP, embebido dentro de una página HTML. En este caso, creamos unas variables en PHP que almacenan un nombre y un año, y luego mostramos esas variables entre el contenido HTML propiamente dicho:

```
    <!DOCTYPE html>
    <html lang="es">
    <head>
    ...
    </head>
    <body>
        <h1>Página de prueba en PHP</h1>
        <?php
            // Variables para almacenar el nombre y el año actual
            $nombre = "Pepe Devesa";
            $anyo = 2024;
        ?>

        <p>El autor de esta página es <?php echo $nombre; ?> y está
        realizada en el año <?php echo $anyo; ?>.</p>

        ...
```

Observa cómo podemos incluir código PHP en cualquier zona de la página.

En la primera, hemos definido dos variables, `$nombre` y `$anyo` , y después, con instrucciones cortas, las hemos mostrado en otras zonas de la página (instrucción `echo`).

Lo que hace el servidor cuando se le solicita esta página es procesarla, detectar el código PHP, ejecutarlo y sustituirlo en la página por el código HTML que éste genera (en el ejemplo anterior, mostrar el valor de las variables en los lugares correspondientes).

Opcionalmente, en el caso de que sólo tengamos instrucciones `echo` , podemos sustituir la estructura `<?php ... ?>` por la estructura `<?= ... ?>` y ahorrarnos la instrucción *echo*. Así, el párrafo que se muestra en el ejemplo anterior lo podríamos poner así:

```
<p>El autor de esta página es <?= $nombre; ?> y está realizada en el año
   <?= $anyo; ?>.</p>
```

También podemos utilizar indistintamente la instrucción `print` en lugar de `echo` para mostrar información por pantalla.

```
<p>El autor de esta página es <?php print $nombre; ?> y está realizada en
   el año <?php print $anyo; ?>.</p>
```

Ejercicio 1:
Crea una carpeta llamada **ejercicios1** en tu carpeta de documentos de XAMPP y subirlo a **github**. En esta carpeta guardarás este ejercicio y los siguientes, ya que serán muchos y así evitamos llenar la carpeta de documentos de demasiadas subcarpetas con ejercicios cortos.
Para este ejercicio, crea un documento en esta carpeta llamado **info_basica.php**, similar al del ejemplo anterior, pero mostrando tu nombre y tu año de nacimiento usando variables. Es decir, crearás dos variables para almacenar estos dos datos, y los mostrarás en una frase que diga **"Me llamo XXXX y nací en el año YYYY"**.
Prueba la página en un navegador y echa un vistazo al código fuente, intentando detectar qué contenidos HTML se han generado desde PHP.

2.1 Variables y tipos de datos

Como en todo lenguaje de programación, las variables en PHP nos van a servir para almacenar información, de manera que, además de tenerla disponible, podemos modificarla o cambiarla por otra durante el tiempo de ejecución de la aplicación web.

Como hemos visto en el ejemplo anterior, las variables en PHP se definen mediante el símbolo del dólar (`$`) seguido de una serie de letras, números o caracteres de subrayado, aunque no pueden empezar por número. Ejemplos de nombres de variables válidos son: `$nombre` , `$primer_apellido` , `$apellido2` … En las variables, se distinguen mayúsculas de minúsculas, y no hace falta declararlas (es decir, no se indica de qué tipo son, como en lenguajes como **C** o **Java**, ni se les reserva memoria de antemano).

```
<?php
    $edad = 36;
    $nombre = "Pepe";
    ...
    echo $nombre;
    echo $edad;
?>
```

Para declarar una variable usando la sintaxis básica, sigue estos pasos:

- Elige un nombre que sea descriptivo y fácil de entender. Ten en cuenta las siguientes reglas:
  - Cada identificador de la variable debe ser único dentro de un programa.
  - Evita usar palabras claves reservadas por el lenguaje de programación, por ejemplo, var, function, if, for, while, etc.
  - El nombre puede incluir letras, caracteres ($, # y @) y números, pero no puede empezar con un número.
  - Los nombres de las variables no pueden empezar ni terminar con un punto.
  - Las variables no permiten espacios entre palabras. Para separar palabras en el nombre, usa el guión bajo (_) o cambia de letra minúscula a mayúscula.
- No es correcto:
  - ~~$var1, $var2,…~~
  - ~~$Elem1, $Elem2…~~

2.2 Variables variables

Una peculiaridad de PHP es la posibilidad de disponer de variables de tipo variable. Por ejemplo, en el siguiente código:

```
$varname = 'barras';
```

```php
$$varname = 5; // Equivale a $barras = 5
```

¿Qué utilidades puede tener esto? El bilingüismo

```php
<?php
    $texto_va = "Benvingut";
    $texto_en = "Welcome";
    $idioma = "va";
    $texto = "texto_" . $idioma;
    echo $$texto;
?>
```

Ejercicio 4:
Crea una página en la carpeta de ejercicios llamada **curriculum.php** donde, utilizando variables variables, muestres parte de tu currículum (por ejemplo, un párrafo con tus estudios y otro con los idiomas que hablas), tanto en español,valencià como en otro idioma que elijas.

2.3 Comprobar el estado de las variables

Es posible que, en algún momento de la ejecución del programa, una variable no tenga un valor definido, o queramos eliminar el valor que tiene. Para ello tenemos algunas instrucciones útiles:

- `unset($variable)` permite borrar la variable (como si no la hubiéramos creado)
- `isset($variable)` permite comprobar si una variable existe
- `empty($variable)` permite comprobar si una variable está vacía, es decir, no tiene un valor concreto asignado.

```php
if (isset($nombre)) {
    echo $nombre;    // Solo muestra el nombre si la variable tiene algún
       valor
} else {
    echo "El nombre no está definido";
}
```

```php
    <?php
        $dato="Hola";
        unset($dato); // La variable dato deja de existir

        $dato = "";
        echo empty($dato); // Diría que es cierto, porque $dato está vacía
    ?>
```

2.4 Tipos de datos

Las variables almacenan datos, y esos datos son de un tipo concreto. Por ejemplo, pueden ser números, o textos. En concreto, PHP soporta los siguientes tipos de datos básicos:

- **Booleanos:** datos que sólo pueden valer verdadero o falso (en PHP se representan con las palabras TRUE y FALSE, respectivamente, en mayúsculas).
- **Enteros:** números sin decimales. Los podemos representar en formato decimal (el normal, por ejemplo 79), formato octal (poniendo un 0 delante, por ejemplo 0233) o hexadecimal (poniendo 0x delante, por ejemplo 0x1A3).
- **Reales:** números con decimales. La parte decimal queda separada de la entera con un punto. Por ejemplo, 3.14. También podemos usar notación científica, como por ejemplo 1.3e3, que equivale a $1.3 \cdot 10$.
- **Textos:** se pueden representar entre comillas simples o dobles. Si los representamos con comillas dobles, podemos intercalar variables dentro.

Veamos algunos ejemplos de cada tipo:

```php
<?php
    $edad = 20;
    $esMayorEdad = TRUE;
    $nota = 9.5;
    $nombre = "Juan Pérez";

    echo "El alumno $nombre tiene una nota de $nota";
?>
```

Además, en PHP podemos manejar otros tipos de datos algo más complejos, que son:

- **Array:** conjuntos de datos
- **Object:** para programación orientada a objetos, almacena instancias de clases
- **Resource:** para recursos externos, como una conexión a una base de datos
- **null:** es un valor especial para darle a variables que, en un momento dado, no tengan un valor concreto. Así, quedan vacías, sin valor.

2.4.1 Conversiones entre tipos de datos

Si tenemos un dato de un tipo y lo queremos convertir a otro, se tienen una serie de funciones que nos permiten hacer estas conversiones.

- **intval** sirve para convertir un valor (en formato cadena, o real) a entero.
- **doubleval** sirve para convertir un valor a número real.
- **strval** sirve para convertir un valor a una cadena de texto.

Es habitual usarlas cuando le pedimos datos al usuario en un formulario. Si por ejemplo le pedimos que escriba su edad en un cuadro de texto, este valor se envía como tipo cadena de texto, no como un número, y luego tendríamos que convertirlo a número entero. Realmente, estas conversiones son automáticas con las últimas versiones de PHP, pero conviene saber que estas funciones existen para poderlas utilizar si es el caso.

Veamos otro ejemplo más sencillo, con una variable cadena de texto que convertimos al entero correspondiente:

```
<?php
    $textoEdad='21';
    $edad = intval($textoEdad);
?>
```

2.5 Constantes

Hemos visto que las variables son datos cuyo valor puede cambiar a lo largo de la ejecución de la aplicación. A veces nos puede interesar almacenar otros datos que no queramos que cambien a lo largo del programa. Por ejemplo, si almacenamos el número *pi*, ese valor siempre va a ser el mismo, no lo vamos a cambiar.

Para definir constantes en PHP se utiliza la función `define` , y entre paréntesis pondremos el nombre que le damos a la constante (entre comillas), y el valor que va a tener, separados ambos datos por coma. Después, para utilizar la constante más adelante, usamos el nombre que le hemos dado, pero sin las comillas. Veamos este ejemplo que calcula la longitud de una circunferencia:

```
<?php
    define('PI', 3.1416);
    define('NOMBRE',"Pepe Devesa");
    $radio = 5;
    $longitud = 2 * PI * $radio;
    echo "La longitud de la circunferencia es $longitud";

    echo NOMBRE; //Fíjate que no lleva delante el simbolo $


?>
```

Aunque **no es obligatorio**, sí es bastante convencional que las constantes tengan todo su **nombre en**

mayúsculas, para distinguirlas a simple vista de las variables (aunque, además, las variables en PHP empiezan por un dólar, y las constantes no).

Ejercicio 2: Crea una página en la carpeta de ejercicios llamada **area_circulo.php**. En ella, crea una variable `$radio` y ponle el valor 3.5. Según esa variable, calcula en otra variable el área del círculo (PI * $radio^2$), deberás definir la constante PI, y muestra por pantalla el texto **"El área del círculo es XX.XX"**, donde XX.XX será el resultado de calcular el área.

2.6 Operaciones

Podemos realizar distintos tipos de operaciones con los datos que manejamos en PHP: aritméticas, comparaciones, asignaciones, etc. Veremos los operadores que podemos utilizar en cada caso. ## Operadores aritméticos

Son las operaciones matemáticas básicas (sumar, restar, multiplicar…). Los operadores para llevarlas a cabo son:

Operador	Significado
+	Suma
-	Resta
*	Multiplicación
/	División
%	Resto de división
.	Concatenación (para textos)
++	Autoincremento
--	Autodecremento

El operador `.` sirve para concatenar o enlazar textos, de forma que podemos unir varios en una variable o al sacarlos por pantalla:

```
$edad = 36;
$nombre = "Pepe";
$texto = "Hola, me llamo " . $nombre . " y tengo " . $edad . " años.";
// En este punto, $texto vale "Hola, me llamo Nacho y tengo 36 años."
```

```php
$edad++;
echo "El usuario se llama " . $nombre;
```

2.7 Operadores de asignación

Permiten asignar a una variable un cierto valor. Se tienen los siguientes operadores:

Operador	Significado
=	Asignación simple
+=	Autosuma
-=	Autoresta
*=	Automultiplicación
/=	Autodivisión
%=	Autoresto
.=	Autoconcatenación

La asignación simple ya la hemos visto en ejemplos previos, y sirve simplemente para darle un valor a una variable.

```php
$dato = 3;
```

Los operadores de Auto… afectan a la propia variable, sumándole/restándole/… etc. un valor. Por ejemplo, si hacemos algo como:

```php
    $dato *= 5; // Dato = 15
```

2.8 Operaciones de comparación

Estas operaciones permiten comparar valores entre sí, para ver cuál es mayor, o si son iguales, entre otras cosas. En concreto, los operadores disponibles son:

Operador	Significado
==	Igual que
===	Idéntico a (igual y del mismo tipo)

Operador	Significado
`!=, <>`	Distinto de
`!==`	No idéntico
`<`	Menor que
`<=`	Menor o igual que
`>=`	Mayor o igual que

Lo que se obtiene con estas comparaciones es un valor booleano (es decir, verdadero o falso). Por ejemplo:

- **5 != 3 sería verdadero, y 4 <= 1 sería falso.**

Además, en versiones recientes de PHP se han añadido algunos operadores adicionales, como el operador **nave espacial** y el operador de **comprobación de nulos.**

- El operador de nave espacial `<=>` compara dos datos y devuelve:
  - **-1** si el primero es menor,
  - **1** si es mayor o
  - **0** si son iguales.

  Tiene un funcionamiento similar a la función **compareTo** que existe en otros lenguajes como **Java o C#.**

```php
$a = 3;
$b = 5;
echo $a <=> $b; // -1
```

- Por su parte, el operador de comprobación de nulo `??` se emplea para determinar si una variable tiene valor nulo, y ofrecer una alternativa en ese caso:

```php
// Mostrará "No existe el dato" si $dato es nulo
echo $dato ?? 'No existe el dato';
```

2.9 Operadores lógicos

Estas operaciones permiten enlazar varias comprobaciones simples, o cambiarles el sentido, según el operador. En concreto tenemos estos operadores lógicos en PHP:

Operador	Significado
`and` , `&&`	Operador AND (Y)
`or`, `||`	Operador OR (O)
`xor`	Operador XOR
`!`	Negación (NO)

Ejemplos:

```
echo $n1 >= 0 && $n2 >= 0;
echo $n1 >= 0 || $n2 >= 0;
echo $n1 >= 0 xor $n2 >= 0;
```

El operador de negación invierte el sentido de una comprobación (si era verdadera, la vuelve falsa, y viceversa). Por ejemplo, si queremos ver si una edad no es mayor de edad, podríamos ponerlo así:

```
echo !($edad >= 18);
```

2.9.1 Precedencia de operadores

¿Qué ocurre si tenemos varios operadores de distintos tipos en una misma expresión? PHP sigue un orden a la hora de evaluarlos:

1. Toda expresión entre paréntesis
2. Autoincrementos y autodecrementos
3. Negaciones y cambios de signo
4. Multiplicaciones, divisiones y restos
5. Sumas, restas y concatenaciones
6. Comparaciones
7. Operaciones lógicas (`&&` , `||`)
8. Asignaciones

Existen, además, otros operadores que no hemos visto aquí, como operadores de bits, conversiones tipo cast, operador ternario… pero no son tan habituales ni importantes.

Ejercicio 3: Intenta predecir qué resultado va a sacar por pantalla cada instrucción echo de este código PHP. Luego podrás comprobar si estabas en lo cierto poniendo el código en una página y probándolo en un navegador.

```php
<?php
    $num1 = 3;
    $num2 = 5;
    $num3 = 8;
    $num1 *= 4;

    echo $num1;
    echo $num1 <= $num2;
    echo $num3 > $num1 and $num3 > $num2;
    echo $num3 > $num1 or $num3 > $num2;
    echo $num1 > $num2 xor $num1 > $num3;

    $num3--;
    echo $num3;

    $num3 += $num1;
    echo $num3;
?>
```

3 Control de flujo

3.1 Estructuras selectivas

3.1.1 if

Las estructuras de control de PHP tienen una sintaxis alternativa que elimina el uso de las llaves, muy denostadas por algunos programadores. Por ejemplo, una instrucción if puede escribirse de forma tradicional:

```php
if ($i < 0) {
    echo "La variable es menor que cero";
}
```

…o bien con la "sintaxis dos puntos":

```php
if ($i < 0):
    echo "La variable es menor que cero";
endif;
```

Puedes elegir la sintaxis con la que te sientas más cómodo/a ### if..else A veces nos interesa realizar

una operación si se cumple una determinada condición y no hacerla (o hacer otra distinta) si no se cumple esa condición. Por ejemplo, si está vacía una variable querremos hacer una cosa, y si no lo está, hacer otra. Para decidir entre varios caminos a seguir en función de una determinada condición, al igual que en otros lenguajes como Javascript, Java o C, se utiliza la estructura **if..else**

```
if (condicion)
{
    instruccion1a;
    instruccion2a;
    ...
}
else
{
    instruccion1b;
    instruccion2b;
    ...
}
```

3.1.2 if..elseif.. elseif..

Si queremos elegir entre más de dos caminos, podemos utilizar la estructura **if..elseif.. elseif..** y poner una condición en cada if para cada camino.

Por ejemplo, imaginemos que tenemos una variable edad donde almacenaremos la edad del usuario. Si el usuario no llega a 10 años le diremos que no tiene edad para ver la web. Si no llega a 18 años, le diremos que aún es menor de edad, pero puede ver la web, y si tiene más de 18 años le diremos que está todo correcto:

```
$edad = ...
if ($edad < 10)
{
    echo "No tienes edad para ver esta web";
}
elseif ($edad < 18)
{
    echo "Aún eres menor de edad, pero puedes acceder a esta web";
}
else
{
    echo "Todo correcto";
}
```

Ejercicio 4

Crea una página llamada **prueba_if.php** en la carpeta de ejercicios del tema. Crea en ella dos variables llamadas `$nota1` y `$nota2`, y dales el valor de dos notas de examen cualesquiera (con decimales si quieres). Después, utiliza expresiones *if..else* para determinar qué nota es la mayor de las dos.

Ejercicio 5

Modifica el ejercicio anterior añadiendo una tercera nota `$nota3` , y determinando cuál de las 3 notas es ahora la mayor. Para ello, deberás ayudarte esta vez de la estructura if..elseif..else.

3.1.3 Switch..case

Similar a otros lenguajes, para elegir entre el conjunto de valores que puede tomar una expresión:

```
switch($variable)
{
    case 0:
        echo "La variable es 0";
        break;
    case 1:
        echo "La variable es 1";
        break;
    default:
        echo "La variable es otra cosa";
}
```

3.2 Break y Continue

Como en muchos otros lenguajes, las instrucciones **`break`** y **`continue`** pueden usarse en el interior del cuerpo de los bucles para lograr este comportamiento:

- **break**. “Rompe” el bucle, es decir, se sale del bucle y continúa ejecutando el programa por la instrucción que haya inmediatamente después del mismo.
- **continue**. Deja de ejecutar la iteración actual y vuelve al comienzo del bucle para iniciar una nueva iteración.

3.3 Estructuras repetitivas o bucles

Para poder repetir código o recorrer conjuntos de datos, al igual que en otros lenguajes de programación, disponemos de algunas estructuras repetitivas o bucles.

3.3.1 for

Supongamos que tenemos una variable `$lista` con una lista de elementos. Si queremos recorrerla, tenemos dos opciones: la primera es utilizar una variable que vaya desde el principio de la lista (posición 0) hasta el final (indicado por la función `count`):

```
for ($i = 0; $i < count($lista); $i++)
{
    echo $lista[i]; // Muestra el valor de cada elemento
}
```

3.3.2 foreach

La segunda alternativa es utilizar una variable que sirva para almacenar cada elemento de la lista.

```
foreach ($lista as $elemento)
{
    echo $elemento; // Muestra el valor de cada elemento
}
```

3.3.3 while

Pondremos entre paréntesis una condición que debe cumplirse para que las instrucciones entre las llaves se repitan. Este código cuenta del 1 al 5:

```
$numero = 1;

while ($numero <= 5)
{
   echo $numero;
   $numero++;
}
```

3.3.4 do..while

Una variante de la estructura while es la estructura do..while, similar a la anterior, pero donde la condición para terminar el bucle se comprueba al final:

```php
$numero = 1;
do
{
echo $numero;
$numero++;
} while ($numero <= 5);
```

Ejercicio 6

Crea una página llamada contador.php en la carpeta de ejercicios del tema. Utiliza una estructura for para contar los números del 1 al 100 (separados por comas), y luego una estructura while para contar los números del 10 al 0 (una cuenta atrás, separada por guiones).

Al final debe quedarte algo como esto:

1,2,3,4,5,6,7,8,9,10,11,12,13,14,15…

10-9-8-7-6-5-4-3-2-1-0

4 Intercalar control de flujo y HTML

Podemos intercalar bloques de código PHP y bloques HTML, incluso cortando bloques de control de flujo para introducir el código HTML. Por ejemplo:

```php
<?php
if ($edad < 10=)
{
?>
<div class="menor">Eres demasiado pequeño para entrar a esta web</div>
<?php } elseif ($edad < 18) { ?>
<div class="mediano">No eres mayor de edad, pero puedes entrar</div>
<?php } else { ?>
<div class="mayor">Todo correcto. Bienvenido</div>
<?php } ?>
```

Ejercicio 7
Modifica el ejercicio anterior y añádele algún h1 y párrafos explicativos a la página, fuera del código PHP, explicando lo que se va a hacer. Por ejemplo, que te quede algo así: Al final debe quedarte algo como esto:
Contadores
Este contador va del 1 al 100:
1,2,3,4,5,6,7,8,9,10,11,12,13,14,15…
Este otro va del 10 al 0:
10-9-8-7-6-5-4-3-2-1-0

5 Arrays

Las tablas o arrays nos permiten almacenar varios datos en una sola variable, de forma que podemos acceder a esos datos utilizando distintos tipos de índices. En PHP existen tres tipos de arrays:

- Numéricos: la posición que ocupa cada elemento en el array la indica un número, y podemos acceder a esa posición indicando ese número entre corchetes. Las posiciones empiezan a numerarse desde la 0.
- Asociativos: la posición que ocupa cada elemento en la lista viene dada por un nombre, y podemos localizar cada elemento a través del nombre que le hemos dado, llamado clave
- Mixtos: son arrays de varias dimensiones, donde en algunas se utilizan índices numéricos y en otras índices asociativos.

Cuidado con mezclar tipos
Como el tipado es dinámico, nuestros arrays pueden contenedor datos de diferentes tipos. No se recomienda mezclar los tipos.

5.1 Arrays numéricos

Estos arrays podemos crearlos de tres formas posibles: * Indicando entre paréntesis sus elementos, y anteponiendo la palabra `array`

```
$tabla = array('Uno', 'Dos', 'Tres');
```

- Indicando a mano el índice que queremos rellenar, y el valor que va a tener (los índices intermedios que queden sin valor se quedarán como huecos vacíos)

```
$tabla[0] = 'Uno';
$tabla[1] = 'Dos';
$tabla[2] = 'Tres';
```

- Indicando el nombre del array con corchetes vacíos cada vez que queramos añadir un elemento. Así, se añade al final de los que ya existen:

```
$tabla[] = 'Uno';
$tabla[] = 'Dos';
$tabla[] = 'Tres';
```

Después, para sacar por pantalla algún valor, o usarlo en alguna expresión, pondremos el nombre del array y, entre corchetes, la posición que queremos:

```
echo $tabla[1];  // Sacaría 'Dos' en los casos anteriores
```

Otros Ejemplos…

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

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

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

5.2 Arrays asociativos

En este caso, al crear el array debemos indicar, además de cada valor, la clave que le vamos a asociar, y por la que lo podemos encontrar, separados por el símbolo "=>". Por ejemplo, este array guarda para cada nombre de alumno su nota:

```
$notas = array('Manuel García'=>8.5, 'Ana López'=>7, 'Juan Solís'=>9);
```

También podemos rellenarlo, como en el caso anterior, indicando entre corchetes cada clave, y luego su valor:

```
$notas['Manuel García'] = 8.5;
$notas['Ana López'] = 7;
...
```

Después, si queremos sacar la nota del alumno Manuel García, por ejemplo, pondremos algo como:

```
echo $notas['Manuel García'];
```

En este tipo de arrays no podremos usar índices numéricos, porque no hay posiciones numéricas.

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 />";
}
```

5.3 Arrays multidimensionales

Los arrays creados anteriormente son unidimensionales (sólo hay una lista o fila de elementos). Pero podemos tener tantas dimensiones como queramos. Es habitual encontrarnos con arrays bidimensionales (tablas), para almacenar información. En este caso, tendremos un corchete para cada dimensión. Por ejemplo, para crear una tabla como la siguiente…

34,1	141
36,4	150
33,5	155

… necesitaremos un código como este:

```php
$tabla[0][0] = 34.1;
```

```
$tabla[0][1] = 141;
$tabla[1][0] = 36.4;
$tabla[1][1] = 150;
$tabla[2][0] = 33.5;
$tabla[2][1] = 155;
```

5.3.1 Arrays multidimensionales de tipo asociativo

Donde algunas dimensiones son numéricas y otras asociativas. Por ejemplo:

```
$productos = array(
array('BAR', 'Barras', 1),
array('ENS', 'Ensaimadas', 3),
array('NAP', 'Napolitanas',2)
);
```

Para acceder a un elemento cualquiera del array anterior usamos índices numéricos. Por ejemplo, para acceder al nombre (segundo campo, índice 1) del tercer producto (tercera posición en el array, índice 2), teclearíamos lo siguiente:

```
$productos[2][1];
```

Por supuesto también sería posible disponer de una matriz multidimensional indexada asociativamente, como la del siguiente ejemplo:

```
$tabla2 =  array(
    array('nombre' => 'Juan García',
          'dni'=> '11111111A',
          'idiomas' => array('inglés', 'valenciano', 'español')     ),
    array('nombre' => 'Elisa Rodríguez',
          'dni' => '22222222B',
          'idiomas' => array('francés', 'español')     )
          );
```

Podríamos mostrar el segundo idioma hablado por la segunda persona con:

```
echo $tabla2[1]['idiomas'][1];
```

Podemos crear igualmente estos arrays usando corchetes, definiendo lo que queremos en cada dimensión:

```
$tabla2[0]['nombre'] = 'Juan García';
$tabla2[0]['dni'] = '11111111A';
$tabla2[0]['idiomas'][0] = 'inglés';
$tabla2[0]['idiomas'][1] = 'valenciano';
```

5.4 Funciones para arrays

PHP dispone de varias funciones útiles a la hora de manipular arrays. Algunas de las más habituales son:

- `count(array)` nos indica cuántos elementos tiene el array. Es útil para utilizarlo en bucles y saber cuántas repeticiones podemos hacer sobre el array
- `sort(array)y rsort(array)` ordenan y reindexan un array numérico (la segunda en orden decreciente).
- `asort(array)y arsort(array)` ordenan y reindexan un array asociativo (la segunda en orden decreciente), por sus valores.
- `ksort(array)y krsort(array)` ordenan un array asociativo por sus claves (la segunda en orden decreciente).
- `usort(array, funcion)` ordena un array según la función que defina el usuario como segundo parámetro.
- `array_filter(array, funcion_filtrado)` devuelve un array con los elementos del array original (pasado como parámetro) que pasan la función de filtrado indicada.
- `array_sum($array)` devuelve la suma de todos los elementos en el array.
- `print_r($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 $array.
- `$claves = array_keys($array)`: devuelve las claves del $array asociativo
- `isset($array[elemento])`: indica si existe/tiene valor elemento dentro del array
- `unset($array[elemento])`: elimina el elemento del array (deja un hueco).
- `array_values($array)`: devuelve todos los valores del array array e indexa numéricamente el array.
- `array_search($valor,$array)` Busca un valor determinado en un array y devuelve la primera clave correspondiente en caso de éxito.
- `list(var1,var2,...)` asigna los elementos del array a las variables.

```php
<?php
    $my_array = array("Dog","Cat","Horse");

    list($a, $b, $c) = $my_array;
    echo "I have several animals, a $a, a $b and a $c.";
?>
```

Artículos para profundizar en el uso de arrays

- Un artículo muy completo (en inglés) de Cómo trabajar con arrays en PHP de la manera correcta.
- Otro artículo recomendable (en inglés) es Cómo ordenar arrays en PHP

6 Gestión de errores

En ocasiones podemos hacer operaciones o utilizar funciones que pueden provocar un error grave en la aplicación. Por ejemplo, una división por cero, o una lectura de un fichero que no existe. Si no controlamos esos errores, se puede "disparar" un mensaje de error en el programa que muestre su mal funcionamiento, o lo que es peor, que revele algún dato privado, como la ubicación de un fichero en el servidor, o algún nombre de usuario o contraseña. Para evitar que ciertas operaciones que puedan causar errores alteren de esa forma el funcionamiento de la aplicación web, existen varias alternativas.

6.1 Uso del operador @

Cuando se antepone el símbolo de arroba @ ante cualquier expresión, cualquier mensaje de error que pueda generar esa expresión será ignorado.

```php
$division = @($num1/$num2);
```

```php
<?php
$miArchivo = @file('archivo_que_no_existe') or die("No se ha podido abrir"
    );

// Funciona también si por ejemplo se intenta acceder al key de un array
    que no existe:
$valor = @$array[$key] or die("No se ha podido abrir");
?>
```

6.2 Uso de excepciones

Vamos a ver un ejemplo sencillo con una función que calcula el área de un cuadrado:

```
$miLado = -3;
function areaCuadrado($lado){
    if ($lado < 0){
        // Lanzamos una excepción
        throw new Exception ('Debes insertar un número positivo');
    } else {
        return $lado * $lado;
    }
}
areaCuadrado($miLado);
// Devuelve: Uncaught exception 'Exception' with message 'Debes insertar
   un número positivo'
```

Hemos lanzado una excepción y el código detiene su ejecución ya que se produce un error fatal. Podemos en cambio capturar ese error y continuar con el script:

```
// Definimos un array con los lados de los cuadrados que queremos calcular
$misLados = array(2, -6, 4);
// Creamos un loop para calcular el área de cada cuadrado
foreach ($misLados as $lado){
    try {
        echo "El área del cuadrado es: " . areaCuadrado($lado) . "<br>";
    } catch (Exception $e) {
        echo 'Ha habido una excepción: ' . $e->getMessage() . "<br>";
    }
}
/*
Devuelve:
El área del cuadrado es: 4
Ha habido una excepción: Debes insertar un número positivo
El área del cuadrado es: 16
*/
```

Ahora en lugar de parar el script, continúa y captura la excepción.

También podemos usar la instrucción **throw new** Exception($mensaje) para provocar una excepción en el caso de que alguna comprobación que hagamos nos dé un resultado incorrecto. Así provocamos un salto al **catch**, o un mensaje de error en la web si no lo hacemos dentro de un try. Se utiliza también para algunas funciones que no provocan excepciones por sí mismas (como por ejemplo, file_get_contents), para provocar el error nosotros de antemano con alguna comprobación previa.

```
try
{
    if (!file_exists("fich1.txt"))
```

```
    {
        throw new Exception("El fichero de entrada no existe");
    }
    $contenido = file_get_contents("fich1.txt");
    file_put_contents("fich2.txt", $contenido);
} catch (Exception $e) {
    echo 'Se ha producido un error: ' . $e->getMessage();
}
```

7 Ejercicios con arrays

Crear una carpeta en github con el nombre `array` y guardarlos ahí. Es conveniente que actaulizéis el archivo **README.md** con los contenidos que vayáis creando.

7.0.1 Array1.php

Rellena un array con 50 números aleatorios comprendidos entre el 0 y el 99, y luego muéstralo en una lista desordenada. Para crear un número aleatorio, utiliza la función rand(inicio, fin) => `$num = rand (0, 99)`

- Como mejora comprobar que los números no existan.
- Ordenar la salida del vector.
- Calcula:
  - El mayor
  - El menor
  - La media

7.0.2 arrayAsociativo.php

Rellena un array de 100 elementos de manera aleatoria con valores `M` o `F` (por ejemplo [“M”, “M”, “F”, “M”, . . .]). Una vez completado, vuelve a recorrerlo y calcula cuantos elementos hay de cada uno de los valores almacenando el resultado en un array asociativo **[‘M’ => 44, ‘F’ => 66]** (no utilices variables para contar las M o las F). Finalmente, muestra el resultado por pantalla

7.0.3 Personas.php

Mediante un array bidimensional, almacena el nombre, altura y email de 5 personas. Para ello, crea un array de personas, siendo cada persona un array asociativo: **[[‘nombre’=>‘Aitor’, ‘altura’=>182, ‘email’=>‘aitor@correo.com’],[...],...]** Posteriormente, recorre el array y muéstralo en una tabla HTML.

7.0.4 Garaje.php

Crea una página llamada `coches.php`. Define dentro un array bidimensional mixto donde:

La primera dimensión sea asociativa. Aquí pondremos matrículas de coches. La segunda dimensión será numérica. En cada casilla guardaremos **la marca, modelo y número de puertas del coche** en cuestión. Por ejemplo, el coche con matrícula “111BCD” puede ser un “Ford” (casilla 0), modelo “Focus” (casilla 1) de 5 puertas (casilla 2). Rellena el array con al menos 3 o 4 coches, y después utiliza las estructuras adecuadas para recorrerlo mostrando los datos de los coches ordenados por matrícula.

7.0.5 arrayBidimensional.php

Rellena un array bidimensional de 6 filas por 9 columnas con números aleatorios comprendidos entre 100 y 999 (ambos incluidos). Todos los números deben ser distintos, es decir, no se puede repetir ninguno. Muestra a continuación por pantalla el contenido del array de tal forma que:

- La columna del máximo debe aparecer en **azul**.
- La fila del mínimo debe aparecer en **verde**.
- El resto de números deben aparecer en **negro**.