Práctica 3

Sesiones 5, 6, 7 y 8

Objetivos

- Conocer los fundamentos del lenguaje de programación PHP.
- Entender la programación web en el servidor.
- Comprender cómo PHP sirve para crear páginas dinámicas.
- Ser capaz de desarrollar guiones en PHP utilizando todas las prestaciones que ofrece el lenguaje.

Introducción a PHP

PHP (*PHP Hypertext Preprocessor*) es un lenguaje de programación de propósito general, muy parecido a C o Java, aunque orientado al desarrollo web, utilizable en cualquier sistema operativo y servidor web. La versión actual es la 8.2. Es un lenguaje del lado del servidor, lo que indica que se ejecuta en el servidor en lugar del cliente como, por ejemplo, JavaScript. El código PHP se puede incorporar directamente en un documento HTML o bien se puede ejecutar directamente en línea de comandos, al estilo de otros lenguajes como Perl o Python. Nosotros, en la asignatura, vamos a emplearlo en el contexto de la web mediante guiones *o scripts* incorporados en documentos HTML, con el objeto de dotar de dinamismo a los desarrollos web que hagamos a partir de ahora.

Primeros pasos en PHP

Para comprobar que tenemos instalado PHP en el servidor, podemos emplear el siguiente código en PHP (prueba.php):

```
<html>
	<head>
		<title>Prueba de PHP</title>
	</head>
	<body>
		<p>Esto es una línea de HTML.</p>
		<?php
			echo "<p>Y esto una línea de PHP.</p>";
			phpinfo();
		?>
	</body>
</html>
```

Las sentencias en PHP deben ir incluidas entre *"<?php"* y "*?>*", es decir, *"<?php" <!-- Código PHP --> ?>* y se incluyen en código HTML. El servidor interpretará todas las sentencias que vayan contenidas en ese elemento y la salida será siempre código HTML que será visualizado en el cliente. Los guiones escritos en PHP suelen tener extensión *.php*. Desde el navegador nunca se puede leer el código PHP pues éste no se muestra, sino su salida. En el ejemplo anterior, se emplea la sentencia *echo* para mostrar un texto en la salida (entrecomillado con comillas dobles o simples) y la función *phpinfo();* que muestra información sobre PHP en el servidor. Todas las sentencias de PHP deben ir finalizadas por punto y coma (;).

De forma general, la sentencia *echo* puede mostrar varias cadenas de caracteres o números, separadas por comas. Las cadenas siempre tienen que ir entre comillas, mientras que los números no.

```
<?php
        /* Esto son unos ejemplos de la sentencia echo. */
        echo "Hola, caracola", 134;
        echo "Adiós ", "Bye";
        echo '<p>Nueva \n linea</p>';
        # No funciona: la salida de PHP es en HTML, y en él '\n' no se interpreta como salto de línea
        # Comentario de una línea.
        # Al usar <br/> ahora sí se produce el salto de línea
        echo "<p>Nueva <br/>\n linea </p>";
?>
```

El carácter especial '\n' introduce una nueva línea, pero HTML no lo interpreta así. Para que haya una nueva línea en HTML tenemos que meter la marca *
*. Por tanto, '\n' se emplea simplemente para formatear código, ya que los saltos de línea con '\n' no son visibles para el usuario de la página web (solo se ven al inspeccionar el código fuente de la página). Los comentarios que pueden contener más de una línea se ponen en PHP mediante */* COMENTARIO */*, mientras que los de una línea empleando # o *//* al principio de la misma.

Variables y expresiones

https://www.php.net/manual/es/language.variables.php

https://www.php.net/manual/es/language.expressions.php

Las variables en PHP:

- ➔ Comienzan con el signo del dólar.
- ➔ Pueden tener cualquier longitud.
- ➔ Sólo incluyen letras, números y guiones bajos (_).
- ➔ Puede empezar por una letra o por un guión bajo.
- ➔ PHP es un lenguaje sensible a las mayúsculas, por lo que *$ciudad* es diferente de *$Ciudad*.

La asignación de valores a variables se hace mediante el signo igual (=) y para mostrar contenidos de variables se emplea la función *print_r* o la sentencia *echo*, como se aprecia en el ejemplo siguiente. La diferencia entre *print_r* y *echo* es que *print_r* muestra información detallada acerca de la variable, por lo que es más útil para depurar el código..

```
<?php
        $edad=21;
        $precio=343.45;
        $nombre= "Pepe";
        $nombre2 = $nombre;
```

```
        print_r($nombre);
        echo $nombre2;
        $pseudonimo= &$nombre2; // Asignación de una variable por referencia.
?>
```

Las variables en PHP no se declaran previamente a su uso. En el momento en que se asignan por primera vez, el intérprete las crea, según el tipo del valor que se le esté asignado. Si luego se le asigna otro valor de otro tipo, PHP cambia su tipo automáticamente. Si se emplea una variable que no se ha inicializado anteriormente, el intérprete mostrará un error.

En PHP, las variables pueden tomar números enteros, reales, caracteres y booleanos (constantes TRUE y FALSE, en mayúscula o minúscula). Este lenguaje también considera falsas las cadenas *"false" y "FALSE"*, el entero 0, el real 0.0, una cadena vacía, una cadena con el carácter "0", un array con cero elementos y la constante NULL. Cualquier otro valor se considera verdadero (*se consideran true y no false (https://www.php.net/manual/en/language.types.boolean.php).*

Para hacer un casting, se pone el tipo de datos entre paréntesis al que se quiere convertir (int, float, string, bool): *$nuevoEntero=(int) $var1;* Para saber el tipo de una variable se emplea la función *var_dump($variable) o gettype($variable).*

Las operaciones básicas que pone PHP a disposición del programador son +, -, *, / y % (resto).

También existen los operadores de incremento (++) y decremento (– (dos guiones)) de una unidad (*$var= $var +1;* ó *$var+=1*). Para añadir, restar, multiplicar o dividir por un valor, se emplea la sentencia *$var *=3*; (para el caso del producto).

PHP ofrece un amplio número de funciones, como son *sqrt, ceil* (redondeo al siguiente entero), *floor* (redondeo al entero anterior), etc (véase https://www.php.net/manual/es/funcref.php). Su uso es el mismo que en otros lenguajes de programación.

Para incluir el valor de una variable en un elemento HTML se puede hacer como sigue:

```
…
<p> La temperatura es de <?php echo $temperatura; ?> grados. </p>
<!-- Ó -->
<?php
    echo "<p>La temperatura es de ", $temperatura," grados. </p>";
?>
<!-- Ó -->
<?php
    echo "<p>La temperatura es de $temperatura grados. </p>";
?>
...
```

La función *number_format(numero, numero_de_decimales, "separador_decimales", "separador_miles")* permite informar a PHP de la forma en que desea mostrar los números:

```
<?php
        $numero=12321.666;
        /* Formatea el número con dos decimales y '.' y ',' como separadores decimales y miles */
        echo number_format($numero,2),"<br/>";
        /* Formatea el número con 2 decimales y con él ',' y de decimales  y la '.' de miles */
        echo number_format($numero,2,",",".");
```

```
?>
```

Las funciones *printf* y *sprintf* (https://www.php.net/manual/es/function.printf.php y https://php.net/manual/es/function.sprintf.php) permite formatear de una forma más compleja cadenas de caracteres y números (la primera muestra y la segunda asigna a una variable la cadena resultante ya formateada): *printf("formato",$var1,$var2,. . .);* y *$var = sprintf("formato",$var1,$var2,. . .);*

Existe también la posibilidad de que el valor que se guarde en una variable sea el nombre de una variable:

```
<?php
        $nombre_de_la_variable="temperatura"
        $$nombre_de_la_variable= 30.5;
        /* Esto implicaría que $temperatura=30.5; */
?>
```

En este caso se ha creado una variable (*nombre_de_la_variable*) que almacena un identificador. Para acceder a la variable que almacenará el valor, se emplea un signo de dólar adicional ($$).

Para destruir una variable se usa la función unset: *unset($variable).*

Las constantes se crean mediante: *define("nombre_de_la_constante", valor)* y se acceden sin usar *$* (ej.: *echo nombre_de_la_constante;).* También se puede emplear la palabra reservada *const*. Esta opción es una mejor alternativa a *define*.

```
<?php
        define("PI",3.1415);
        define("TIEMPO", "soleado");
        const PI = 3.1415;
?>
```

PHP pone a disposición del programador una gran cantidad de constantes predefinidas, las cuales se identifican de la forma __CONSTANTE__ (*echo __FILE__;*) (dos guiones bajos consecutivos al principio y al final de la constante).

Véase https://www.php.net/manual/es/language.constants.predefined.php.

Existe también un operador llamado de fusión de *null* que permite asignar un valor a una variable si un valor de otra es *null*:

```
<?php
        $usuario = $_GET['usuario'] ?? 'Invitado';
        echo $usuario; // Si no existe "usuario", muestra "Invitado".
        // O también
        $nombre = null;
        nombre ??= "Anónimo";
        echo $nombre; // "Anónimo"
?>
```

Ejercicio:

Escribe un documento HTML que contenga código PHP para probar cada uno de estos conceptos que se acaban de comentar.

Escribe también otro documento HTML que albergue un guión en PHP para calcular el total de un almuerzo en un restaurante, incluyendo el IVA (21%), almacenado como una variable. Invéntate los platos y los precios.

Cadenas de caracteres

https://www.php.net/manual/es/language.types.string.php

Como ya hemos visto anteriormente, las cadenas de caracteres van entre comillas mediante comillas simples o dobles. Básicamente, se pueden emplear indistintamente unas u otras, pero hay una diferencia: ¿cuál es a luz de este código? Pruébalo.

```
<?php
        $nombre = "Juan";
        $var1 = "$nombre";
        $var2 = '$nombre';
        echo $var1,"\n";
        echo $var2,"\n";
        $cad1 = "- Cadena entre\n\tcomillas dobles - ";
        $cad2 = '- Cadena entre\n\tcomillas simples - ';
        echo $cad1,"\n";
        echo $cad2,"\n";
        $numero = 10;
        $cad1 = "- Hay '$numero' personas en la cola. - ";
        $cad2 = '- Hay "$numero" personas esperando. -';
        echo $cad1,"\n";
        echo $cad2;
?>
```

Como se puede ver, las expresiones (ej.: $numero en el ejemplo) se evalúan cuando aparecen entre comillas dobles (") pero no cuando aparecen entre comillas simples (').

Los caracteres se escapan mediante '\': *$cad1="Where is Juan\'s son?";*

La concatenación de caracteres se hace mediante el operador '.':

```
<?php
        $cad1 = "Hola,";
        $cad2 = ' caracola.';
        $cad3=$cad1.$cad2;
        $cad4=$cad1." ".$cad2;
        echo $cad3." – ".$cad4;
        $cad4 .= " Este texto se añade por detrás a la cadena.";
```

```
?>
```

Algunas funciones útiles para manipular cadenas son las siguientes:

- *trim($cad)* → Elimina espacios en blanco al principio y al final.
- *ltrim($cad)* → Elimina espacios al principio.
- *rtrim($cad)* → Elimina espacios al final.
- *str_word_count($cad, formato)* → Divide una cadena en palabras y las guarda en un array

```
<?php
        $cad = "Contando palabras";
        /* Si no se incluye el formato, cuenta el número de palabras. */
        $numeroDePalabras = str_word_count($cad);
        $pal1 = str_word_count($cad,1);
        /* Se crea un vector con dos posiciones: pal1[0] conteniendo "Contando" y pal1[1]
         almacenando "palabras". */
        $pal2 = str_word_count($cad,2);
        /* Se crea un vector con dos posiciones: pal2[0] conteniendo "Contando" y pal2[9]
         almacenando "palabras". */
?>
```

En https://www.php.net/manual/es/ref.strings.php existe una lista completa con las funciones para manejo de cadenas de caracteres.

Gestión de errores

Cuando hay un error en el código, PHP5 ofrece varios tipos de mensajes según la gravedad del mismo: mensajes de error, avisos y notificaciones. Éstos pueden mostrarse dependiendo del nivel de error que haya sido asignado a PHP y su visualización dependerá de si estamos en desarrollo o en producción. Para modificar la conducta de PHP frente a errores, se puede cambiar el valor asignado a la variable error_reporting del fichero php.ini, si se es administrador de PHP (por ejemplo, asignándoles las constantes E_ALL, E_WARNING, E_NOTICE). En caso de no serlo, se pueden cambiar a nivel de guión, añadiendo al principio la función error_reporting (http://php.net/manual/es/book.errorfunc.php):

```
<?php
        error_reporting(E_ALL); # Se muestran los tres tipos → Mejor opción en desarrollo
        error_reporting(E_ALL & ~E_NOTICE); /* Se muestran errores y avisos, pero no
                                                        notificaciones. */
        error_reporting(0); # No se muestra ningún tipo de error.
?>
```

Ahora, desde PHP7, se incorpora un modelo de manejo de excepciones y errores similar a otros lenguajes y basado en el try/catch, lanzando o capturando excepciones Error (errores de PHP internos): (https://php.net/manual/es/language.errors.php7.php):

```
<?php
                try {
                    $resultado= numero/0;
                } catch (DivisionByZeroError $error)
                   {
```

```
                error_log("Se ha producido un error en la condición",3,"/temp/err_log");
                /* O incluso enviar al administrador un mensaje de correo electrónico:
                error_log("Se ha producido un error en la condición",1,"admin@midominio.es");
                exit();
              }
            try {
              throw new Exception("This is an exception");
            } catch (Throwable $e) {
                        echo $e->getMessage();
            }
            try {
                require "index3.php";
            } catch (ParseError $e) {
            echo $e->getMessage();
            }
?>
```

Vectores (arrays)

https://www.php.net/manual/es/language.types.array.php

Un vector puede verse como una lista de pares clave (índice) / valor, almacenados como sigue:

```
$nombre_del_vector['clave1'] = valor1;
$nombre_del_vector['clave2'] = valor2;
$nombre_del_vector['clave3'] = valor3;
```

La clave puede ser una cadena de caracteres o un número entero: *$capitales['Spain']='Madrid';*

También se pueden crear sin indicar el índice, pero en este caso la clave es entera y comienza por cero:

```
$capitales[] = "Madrid";
$capitales[] = "Londres";
```

Así, "Madrid" estará situada en la posición 0, mientras que "Londres" lo estará en la 1.

Otra forma de crear vectores es enumerando en una misma sentencia sus elementos (comenzando por el índice 0):

```
$capitales = array("Londres","Madrid","Berlín");
```

Para indicar el índice por el que se desea comenzar se emplea la siguiente sintaxis:

```
$capitales = array(12  =>"Londres","Madrid","Berlín");
```

Para crear vectores por enumeración pero con cadenas de caracteres como claves:

```
$capitales = array ( "SP" => "Spain","UK "=> "Reino Unido" , "USA"=> "Salem" );
```

PHP también nos permite crear vectores mediante rangos:

```
$anios = range(2001, 2010);
$letras=range("z", "a");
```

Para mostrar vectores se pueden emplear las funciones *print_r* y *var_dump*, con el vector a visualizar como argumento.

Si queremos mostrar el contenido de algún vector mediante alguna de estas dos funciones en un navegador, es conveniente incluir su salida en un elemento *pre* de HTML y así no aparecerá todo en una única línea en el navegador:

```
<?php
        echo "<pre>";
        var_dump($customers);
        echo "</pre>";
?>
```

Si queremos modificar un valor para una clave, simplemente asignarle el nuevo valor a la clave correspondiente. Si no existe dicha clave, se añade al final.

```
$capitales['Andalucia'] = "Granada";
```

Igual ocurre si se asigna un valor sin especificar a la clave, aunque en este caso para claves numéricas:

```
$capitales[1] = "Madrid";
$capitales[2] = "Murcia";
$capitales[] = "Granada"; /* En este caso se asigna al índice 3. */
```

Para obtener un valor, simplemente se indexa el vector por la clave correspondiente:

```
$ciudad= $capitales[1];
```

Para copiar un vector íntegramente en otro, simplemente asignarlos: *$unVector= $otroVector;*

Para borrar un elemento, se debe emplear la función unset: *unset($capitales[2]);*

Nótese que las claves no cambian al borrar un elemento intermedio.

El tamaño de un vector se obtiene con las funciones *count($vector)* y *sizeof($vector)* (ambas funcionan de forma idéntica).

Para ordenar los valores de vectores con números como claves en orden ascendente, se debe emplear la función *sort* (en sentido descendente, *rsort*). En el caso de desear ordenar un vector con cadenas de caracteres, se utiliza la función *asort* (análogamente *arsort*). Y cuando se quiera ordenar las claves, se utiliza *ksort* (o *krsort* para ordenarlo en sentido contrario).

Se pueden asignar los n primeros valores de un vector a n variables mediante la sentencia *list*:

```
list($primero, $segundo, $tercero)= array_values$capitales; /* Se copian los 3primeros valores */
// También se puede poner [$primero, $segundo, $tercero] = $capitales; (recomendado)
echo $primero, " ", $segundo, " ", $tercero;
```

Para recorrer un vector de forma manual se emplean punteros a los elementos del vector, por medio de las siguientes funciones:

- *$valor= current($vector);* → devuelve el valor del elemento actual (al que apunta el vector).
- *$valor= next($vector);* → devuelve el valor del elemento siguiente al actual y modifica a dicho elemento el puntero.
- *$valor= previous($vector);* → devuelve el valor del elemento anterior al actual y modifica a dicho

elemento el puntero.

- *$valor= end($vector);* → mueve el puntero al último valor del vector y lo devuelve.
- *$valor=reset($vector);* → mueve el puntero al primer valor del vector y lo devuelve.

Las funciones *previous*, *next*, *end* y *reset* pueden emplearse sin almacenar el valor devuelto. Para obtener la clave del elemento del vector en el que se encuentra el puntero actualmente, se emplea la función key($vector);

También podemos iterar por un vector mediante la función *foreach*:

```
<?php
        $vectorPoblaciones = array ( "ES" => 34501130, "FR" => 5494423, "CH" => 333472867);
        // Ó $vectorPoblaciones = ["ES" => 34501130, "FR" => 5494423, "CH" => 333472867];


        ksort($vectorPoblaciones);
        foreach($vectorPoblaciones as $nacion => $poblacion )
                {
                        $poblacion = number_format($poblacion);
                        echo "$nacion: $poblacion.<br/>";
                }
        /* Si se utilizara foreach tal que así:
        foreach($vectorPoblaciones as $poblacion)
                {…}
        Se almacenan sólo los valores en la variable $población sucesivamente. */
?>
```

Para convertir una cadena de caracteres a un vector, se emplea la función *explode*:

```
<?php
        $cadena = "Esto es una cadena de caracteres";
        $vector = explode(" ",$cadena); /* El primer argumento establece la cadena usada para
                                           separar y el segundo la cadena en sí. */
        print_r($vector);
?>
```

Y al contrario, la función *implode*:

```
<?php
        $vector = array("Esto", "es", "una", "cadena", "de", "caracteres");
         // ó $vector = ["Esto", "es", "una", "cadena", "de", "caracteres"];
        $cadena = implode(";",$vector); /* El primer argumento establece la cadena usada para
                                           separar los elementos del vector y el segundo la cadena  en sí. */
        echo $cadena;
        ?>
```

Otras operaciones que podemos realizar con vectores son las siguientes:

- Para convertir variables en vectores y viceversa, se emplean las funciones *extract* y *compact*, respectivamente.
- Y para mezclar vectores, *array_merge.*
- Si queremos comparar vectores, debemos emplear la función *array_diff($vector1, $vector2)*, la cual crea un nuevo vector con los elementos que están en el primer vector pasado como argumento y no en el segundo.
- La intersección de vectores se puede hacer con la función *array_intersect($vector1, $vector2,...);*
- Para sumar los elementos de un vector se emplea *array_sum($vector);*
- El borrado de valores duplicados se hace mediante *$sin_duplicados=array_unique($vector);*
- Para intercambiar claves y valores (los valores serán las claves y éstas los nuevos valores) se usa la función *$intercambiado= array_flip($vector);*

También, y desde PHP7, se pueden crear arrays constantes:

```
<?php
        define('VEHICULOS', [  'coche', 'moto', 'avion'  ]);
        echo VEHICULOS[1]; // imprime "moto"
?>
```

Las matrices (vectores de dos dimensiones o vectores de vectores) se pueden crear en PHP de las siguientes formas:

```
<?php
        $precios['verdura']['patata'] = 1.00;
        $precios['fruta']['manzana'] = 2.50;
        $vehiculos['coche'][] = "Ford";
        $vehiculos['coche'][] = "Seat";
        $precios = array(
                        "verdura"=>array("patata"=>1.00,"cebolla"=>.50),
                        "fruta"=>array("manzana"=>2.50,"naranja"=>2.00));
        /* Podríamos decir que las claves son "verdura" y "fruta" y sus valores un vector cada una.*/
?>
```

El acceso a un valor pasa por la indicación de las claves correspondientes, como hemos visto en el ejemplo.

Por extensión, se pueden crear vectores de más de dos dimensiones.

En PHP existen varios arrays predefinidos que almacenan datos relacionados con el servidor, sistema operativo, o incluso variables globales. En https://php.net/manual/en/reserved.variables.php puedes encontrar la lista de dichos vectores, así como su descripción y ejemplos de uso.

Ejercicio:

Escribe un guión PHP integrado en un documento HTML para trabajar con vectores. Pon en práctica todas las sentencias que los manejan y muestran.

Control del flujo

https://www.php.net/manual/es/language.control-structures.php

Antes de entrar en las sentencias de control de flujo, mencionar que los operadores relacionales de que dispone PHP son los siguientes: = = (igualdad de valores), = = = (igualdad de valores y tipos de datos), >, >=, <, <=, != y <> (distinto), != = (desigualdad de valores y de tipos de datos). El operador de negación es !.

Se pueden comparar entre sí números y cadenas de caracteres. Las comparaciones devuelven los valores booleanos TRUE y FALSE.

Existen algunas funciones interesantes para saber si:

- una variable existe: *isset($var*,
- si está vacía: *empty($var*), es decir, si el valor es 0 o en caso de ser una cadena de caracteres si no tiene ninguno.

Para saber el tipo de dato de una variable: i*s_array, is_int, is_string, is_bool, is_object is_float, is_null, is_numeric* (en general el tipo de una variable se puede obtener con *gettype($variable)*).

Los operadores lógicos son *and*, *or* y *xor* (también se pueden emplear los caracteres && y || en lugar de *and* y *or*, respectivamente). Se pueden agrupar las expresiones lógicas mediante paréntesis.

Las sentencias condicionales en PHP son dos: *if* y *switch*.

```
<?php
        $nota = 3;
        if ($nota > 9 )
        {
                $mensaje = "Matrícula de honor";
        }
        # También se puede escribir else if y funciona igual
        elseif ($nota < 9 and $nota >= 8 )
        {
                $mensaje = "Sobresaliente";
        }
        elseif ($nota < 8 and $nota >= 7 )
        {
                $mensaje = "Notable";
        }
        elseif ($nota <7 and $nota >= 5 )
        {
                $mensaje = "Aprobado";
        }
        else
        {
                $mensaje = "Suspenso!";
        }
        echo $mensaje,". \n";
        $moneda = "EURO";
        switch ( $moneda )
        {
                case "DOLLAR" :
                        $cambio = 1.39;
```

```
                    break;
                case "LIBRA" :
                    $cambio = 0.75;
                    break;
                default:
                    $cambio = .5;
                    break;
        }
        echo $moneda,". \n";
?>
```

Una alternativa más concisa y estricta que switch es match:

```
<?php
        $estado = "ok";
        $mensaje = match ($estado) {
          "ok" => "Todo bien",
          "error" => "Algo salió mal",
          default => "Estado desconocido"
         };
?>
```

Además se puede emplear el operador ternario (?:) para escribir sentencias if-else más compactas:

```
<?php
        if (rand(0, 1)) {
            return "Texto";
        } else {
            return 42;
        }
        // Es equivalente a:
        $valor = rand(0, 1) ? "Texto" : 42;
?>
```

Existen tres tipos de bucles: *for*, *while* y *do... while*:

```
<?php
        /* Bucle for */
        for($i=1;$i<=9;$i++)
        {
                echo "\nTabla del multiplicar del $i \n";
                for($j=1;$j<=9;$j++)
                {
```

```
                $result = $i * $j;
                echo "<p>$i x $j = $result<p>\n";
        }
    }
    /* Bucle while */
    $fruta = array ( "naranja", "manzana", "uvas" );
    $encontrada = "no";
    $k = 0;
    while ( $encontrada != "si" )
    {
        if ($fruta[$k] == "manzana" )
        {
            $encontrada = "sí";
            echo "manzana\n";
        }
        else
          echo "$fruta[$k] no es una manzana\n";

        $k++;
    }
    /* Bucle do... while */
    $encontrada = "no";
    $k = 0;
    do
    {
        if ($fruta[$k] == "manzana" )
            {
                $encontrada = "si";
                echo "manzana\n";
            }
        else
            echo "$fruta[$k] no es una manzana\n";
        $k++;
    } while ( $encontrada != "si" )
?>
```

Para salir de un bucle, se emplea la sentencia break (finaliza la ejecución del bucle) y para saltar a la siguiente iteración sin completar la actual, se usa *continue*.

Ejercicio:

Escribe un guión que cree y recorra una matriz con cualquiera de los tres tipos de bucles y muestre su contenido, empleando las funciones que nos permiten acceder de forma manual a los valores almacenados en la matriz.

Reutilización de código

Si tenemos código que se repetirá en varias partes del guión, lo mejor es reutilizar dicho código para evitar tener que copiarlo cuando nos haga falta. Esta labor se suele hacer incluyendo el código en ficheros externos, que luego se introducirán en el guión en el que se vayan a emplear, y también, mediante el uso de funciones.

Para incluir código PHP en otros guiones PHP se emplea la sentencia *include*. Veamos un ejemplo. Definimos dos constantes en el fichero tamanio.inc (se suele emplear en estos casos la extensión .inc – el intérprete de PHP no procesará nunca estos ficheros aisladamente):

```
<?php
        /* Fichero tamanio.inc */
        define("ALTURA",60);
        define("ANCHURA",60);
?>
```

Seguidamente, en el fichero imagen.php, podemos incluir dicho fichero y hacer uso de las constantes:

```
<?php
            include('tamanio.inc');
?>
<img src="imagen.jpg" height="<?php echo ALTURA?>" width="<?php echo ANCHURA?>" />
```

Una variación del uso de include es una mejora recomendada sobre el uso tradicional de include es la que se muestra a continuación. Garantiza que el archivo se cargue correctamente sin importar desde qué directorio se ejecute el script (evita problemas con las rutas relativas). Para eso se emplea la constante __DIR__, que contiene el directorio actual donde está ubicado el script.

```
<?php
          include __DIR__.'/tamanio.inc' // ó include(__DIR_.'/tamanio.inc')
?>
<img src="imagen.jpg" height="<?php echo ALTURA?>" width="<?php echo ANCHURA?>" />
```

Para evitar inclusiones múltiples, se emplea la función *include_once("nombre_de_fichero");*

Análogamente a *include* e *include_once*, existen las funciones *require* y *require_once*, las cuales se diferencian de las primeras en que si el fichero a incluir no existe, se produce un error en tiempo de ejecución que hace que el guión finalice su ejecución (esto no ocurre con *include*, donde sólo se da un aviso).

Las funciones en PHP

(https://www.php.net/manual/es/language.functions.php)

Al igual que en otros lenguajes de programación, las funciones en PHP pueden o no tener argumentos de entrada y pueden o no devolver un valor como resultado de su ejecución.

Se suelen definir en un fichero .inc separado del lugar donde se usan, aunque también se pueden definir dentro del mismo fichero .php en el que se emplean.

Veamos un ejemplo de función que no tiene argumentos de entrada y que no devuelve ningún valor, pero sí escribe un texto en la salida:

```
<?php
	function aniadir_pie()
	{
		echo ' <footer> <img src="escudo_ugr.jpg" width="100%" height="7" />
		<address>Universidad de Granada
		<br />Cuesta del Hospicio, s/n
		<br />Granada, 18071
		</address>
		<p>Contacto: <a href="mailto:info@ugr.es">UGR </a> </p> </footer>';
	}
?>
```

La misma función pero devolviendo una cadena de caracteres y con un par de parámetros de entrada quedaría:

```
<?php
	function aniadir_pie2($servicio,$dirCElectronico)
	{
		$cadena= ' <footer> <img src="escudo_ugr.jpg" width="100%" height="7" />
		<address> <p>'. $servicio.' Universidad de Granada</p>
		<br />Cuesta del Hospicio, s/n
		<br />Granada, 18071
		</address>
		<p>Contacto: <a href="mailto:'.$dirCElectronico.'">UGR </a> </p> </footer>';
		return $cadena;
	}
?>
```

Para devolver un valor se emplea la palabra reservada *return* seguida de la variable, expresión o literal que se desea retornar.

Para usar las funciones, simplemente se invocan de la manera habitual:

```
<?php
	aniadir_pie();
	$salida=aniadir_pie2("Servicio de habilitación", "habilitacion@ugr.es");
```

```
?>
```

Las variables que se utilizan en el cuerpo de una función son locales salvo que se indique explícitamente que son globales anteponiendo a la variable la palabra reservada global: *global $fecha;*

Se pueden asignar valores por defecto a los parámetros de entrada de la siguiente forma:

```
<?php
	function suma($num1=1,$num2=1)
	{
		$total = $num1 + $num2;
		return $total;
	}
?>
```

La función se puede invocar con cero, uno o dos parámetros de entrada. Si no aparece ninguno, se le asigna 1 por defecto a los parámetros num1 y num2. Si es uno, num1 tomará el valor del parámetro de entrada, mientras que a num2 se le asigna el valor por defecto, 1. Finalmente, si se pasan dos argumentos de entrada, num1 y num2 toma los valores correspondientes:

```
<?php
	$resultado=suma(2,2);
	$resultado=suma(2);
	$resultado=suma();
?>
```

También se pueden pasar valores por referencia. Para ello, se pone delante del parámetro correspondiente un &:

```
<?php
	function incrementar(&$num) {
		$num=$num+1;
	}
?>
```

Las declaraciones de tipo permiten establecer (requerir) el tipo de los parámetros (self, array, bool, float, int, string, void, …) en una llamada. Si el tipo es incorrecto se generará una excepción TypeError. Para poner esta tipificación estricta se debe poner *declare(strict_types=1);*

```
<?php
	declare(strict_types=1);
	function producto (int $n1, int $n2 ) {
		return $n1 * $n2;
	}
	var_dump(producto(5,3));
	var_dump(producto(5.4,3.2)); /* Se lanzará TypeError. Si no está la primera sentencia,
					no ocurrirá nada */
?>
```

Además, se puede establecer el tipo de dato que devuelve la función:

```php
<?php
	declare(strict_types=1);
	function sumar(int $a, int $b): int {
		return $a + $b;
	}
	echo sumar(3, 5); // 8
?>
```

Una misma función puede devolver cualquier tipo de dato. Para indicarlo se emplea la palabra reservada *mixed*:

```php
<?php
	function obtenerDato(): mixed {
		 return rand(0, 1) ? "Texto" : 42;
	}
?>
```

Las funciones también pueden aceptar múltiples tipos de datos como argumentos:

```php
<?php
	function mostrarValor(int|string $valor): void {
		echo $valor;
	}
	mostrarValor(10); // 10
	mostrarValor("Hola"); // "Hola"
?>
```

A veces la ejecución de una función puede fallar (imagínese una función que accede a una base de datos para recuperar ciertos datos y ésta no está operativa). Se puede dejar que PHP muestre un aviso o generar nuestro propio mensaje mediante la función *die("mensaje")* y parar la ejecución del guión. La sentencia or die(…) solo se ejecutará si la función previa devuelve false (o no devuelve ningún valor), pero no en caso de que se produzca una excepción o error. Para esto último, sería necesario usar un bloque catch. El uso de *die* se hace conjuntamente con la invocación de la función:

```php
<?php
	consultarBD($consulta) or die("Error: no se puede acceder a la base de datos");
?>
```

Las funciones anónimas en PHP (también llamadas closures o funciones lambda) son funciones que no tienen un nombre y se asignan a una variable o se pasan como argumento a otra función.

```php
<?php
	$suma = function ($a, $b) {
	   return $a + $b;
	};
	echo $suma(5, 3); // 8
?>
```

Se emplea la función *use* para capturar variables exteriores:

```
<?php
        $factor = 3;
        $multiplicar = function ($n) use ($factor) {
           return $n * $factor;
        };
        echo $multiplicar(4); // 12
?>
```

Otro tipo de funciones son las conocidas como funciones flecha, que permite escribir funciones anónimas de una forma más corta. Estas se definen con fn en lugar de function, no necesitan return ni *use* y sólo contienen una línea de código y no pueden incluir estructuras repetitivas ni condicionales.

```
<?php
        $doblar = fn($n) => $n * 2;
        echo $doblar(5); // 10
?>
```

Ejercicio:

Escribe varias funciones que devuelvan código HTML para confeccionar un documento HTML que contenga elementos header, footer y tres section.

Bibliografía

https://www.php.net/manual/es/index.php