UT 4: FUNCIONES

1. **Funciones**
2. Funciones útiles
3. Ficheros externos
    1. require
    2. include
    3. Diferencia entre require e include

1. FUNCIONES

Según aumenta la extensión y complejidad de un programa, es habitual tener que implementar en distintas partes la misma funcionalidad. Esto implica tener el mismo bloque de código en varias partes del programa.

Esto genera problemas:

- Aumenta el tamaño del código y la legibilidad del programa.
- Dificultad de mantenimiento: cualquier cambio en ese bloque de código, tendrá que hacerse en todos los sitios.
- Más probabilidad de errores: En uno de los bloques de código puede haber un error.

1. FUNCIONES

La solución para cuando se necesita la misma funcionalidad en distintos lugares de nuestro código es etiquetar con un nombre a ese bloque de código y sustituir el bloque de código por el nombre en los lugares donde aparezca.

A esto se le conoce como función.

Las funciones deben estar definidas antes de realizar la llamada a la función.

1. FUNCIONES

DECLARACIÓN Y USO DE LA FUNCIÓN

Una función puede recibir parámetros externos de los cuales dependa el resultado de dicha función. Aunque una función puede necesitar de ninguno, uno o varios parámetros para ejecutarse.

Cuando no devuelven valores, se les llama procedimientos.

```
function nombreFuncion($arg_1, $arg_2, …, $arg_n) {
          instrucciones;
 }
```

1. FUNCIONES

DECLARACIÓN Y USO DE LA FUNCIÓN

```
<?php
//Declaración de funciones
function mostrarTexto($texto) {
    echo "<strong>El texto a mostrar es el siguiente: </strong>";
    echo $texto;
}
//Fin de declaración de funciones
mostrarTexto("Hola");
?>
```

1. FUNCIONES

DECLARACIÓN Y USO DE LA FUNCIÓN

También se pueden crear funciones para que devuelvan datos, para ello se utiliza la palabra reservada `return`.

En este caso se les llama funciones.

```
function nombreFuncion($arg_1, $arg_2, …, $arg_n){
            instrucciones;
            return $valor_devuelto;
}
```

1. FUNCIONES

DECLARACIÓN Y USO DE LA FUNCIÓN

```
<?php
  function sumar($n1, $n2) {
      $resultado = 0;
      $resultado = $n1 + $n2;
      return $resultado;
  }


   // Llamar a la función operaciones
   $r = sumar(5, 7);
   echo $r . "<br>";
?>
```

1. FUNCIONES

ALCANCE DE LAS VARIABLES

Las variables definidas en una función, al finalizar se eliminan. Su ámbito es local a la función.

```
function f1 () {
    $h=9;
}
echo $h; // error : Variable  no definida
```

1. FUNCIONES

ALCANCE DE LAS VARIABLES

Ej: otro ejemplo

```
$h=5;
function f1 ( ) {
    $h=9;
}
echo $h; //escribe 5
```

1. FUNCIONES

PASO DE PARAMETROS

Las funciones reciben parámetros por valor. Esto significa que los parámetros de la función usan copias de los valores.

Ejemplo 1:

```
function f1 ($x ){
     $x=9 ;
}
$x=7 ;
f1 ($x);
echo $x; // escribe 7
```

1. FUNCIONES

PASO DE PARAMETROS

Ejemplo 2:

```
function swap( $x , $y ) {
    $aux=$x ;
    $x=$y;
    $y=$aux;
}
$valor1 =12;
$valor2=17 ;
swap ($valor1 ,$valor2 );
echo $valor1." ". $valor2; //Muestra 12 17.
```

1. FUNCIONES

PASO DE PARAMETROS

Se podría trabajar por referencia, pero no es lo más habitual.

Solución ejemplo 1:

```
function f1 ($x ){
     $x=9 ;
     return $x;
}
$x=7 ;
$x=f1 ($x);
echo $x; // escribe 9
```

1. FUNCIONES

PASO DE PARAMETROS

Solución ejemplo 2:

```
function swap($x, $y) {
    $aux = $x;
    $x = $y;
    $y = $aux;
    return array($x, $y);
}
$valor1 = 12;
$valor2 = 17;
list($valor1, $valor2) = swap($valor1, $valor2);
echo $valor1 . " " . $valor2; // Muestra 17 12
```

EJERCICIO: Escribe una función que reciba un número como parámetro de entrada y que imprima su tabla de multiplicar. (tipo procedimiento)

EJERCICIO: Realizar una función que reciba cinco números enteros como parámetros y devuelva el resultado de sumar los cinco números (tipo función, hay un valor devuelto).

UT 4: FUNCIONES

1. Funciones
2. **Funciones útiles**
3. Ficheros externos
    1. require
    2. include
    3. Diferencia entre require e include

2. FUNCIONES ÚTILES

PRINTF

Ya hemos visto cómo utilizar la función echo y print para mostrar algo por pantalla.

Printf: Es una función que permite mostrar información formateada según los valores que aparezcan en una cadena de formato integrada por caracteres fijos y por una directiva %.

```
printf ("cadena de formato", argumento1, argumento2,...)
```

2. FUNCIONES ÚTILES

PRINTF

Especificadores de formato usados en el método printf:

- **%s**: representar una cadena.
- **%d**: representar un número entero. Ej: %d ó %05d
- **%f**: representar un número de punto flotante (decimal). Ej: %.2f

Ej:

```
printf("Nombre: %s, Edad: %d, Valor de Pi: %.2f", $nombre, $edad, $pi);

printf("Valor de la variable \$i: %d, $i);
```

2. FUNCIONES ÚTILES

PRINTF

Resaltar que en el texto literal hemos tenido que utilizar el signo \ delante de $i para que la variable no se expandiera.

Toda directiva debe iniciarse por un signo %. Si se quiere anular el efecto de % de forma que la directiva se vea como un literal, hay que poner delante otro signo %. Por ejemplo, la orden:

```
printf("Valor de la variable \$i= %%d \n", $i);
```

muestra el texto Valor de la variable $i= %d sin el valor de $i ya que no hemos precisado su formato.

La salida sería:

```
Valor de la variable $i= %d
```

2. FUNCIONES ÚTILES

ADAPTAR CADENAS AL CONTEXTO

- **addSlashes(string)** devuelve una cadena con barras invertidas delante de los caracteres que necesitan marcarse en consultas de bases de datos y en otras operaciones en las que intervienen esos caracteres. Nos estamos refiriendo a la comilla simple ('), a la comilla doble ("), a la barra invertida (\) y a NULL (byte nulo).

  Es una medida de protección que se utiliza contra la inyección de SQL.

```
// Así la variable $consulta contendrá WHERE nom = ‘Ana’
$consulta="WHERE nom = 'Ana'";
// Así la variable $consulta contendrá WHERE nom = \‘Ana\’
$consulta=addSlashes($consulta);
```

2. FUNCIONES ÚTILES

ADAPTAR CADENAS AL CONTEXTO

- **stripSlashes(string)** devuelve una cadena sin barras invertidas delante de la comilla simple ('), la comilla doble ("), la barra invertida (\) y el valor NULL. Esta función, pues, realiza la función inversa de la función anterior.

```
// Así la variable $consulta contendrá WHERE nom = 'Ana'
$consulta=stripSlashes($consulta);
```

2. FUNCIONES ÚTILES

LETRAS MAYÚSCULAS/MINÚSCULAS

- **strtoupper(string)**: devuelve la misma cadena convirtiendo todos sus caracteres en mayúsculas.

Ejemplo: si $a contiene "Pepe", `echo strtoupper($a);` devuelve "PEPE".

- **strtolower(string)**: devuelve la misma cadena convirtiendo todos sus caracteres en minúsculas.

Ejemplo: si $M contiene "PEPe", `echo strtolower($M);` devuelve "pepe".

2. FUNCIONES ÚTILES

LETRAS MAYÚSCULAS/MINÚSCULAS

- **ucfirst(string)**: devuelve la misma cadena convirtiendo su primer carácter en mayúscula si es una letra.

  Ejemplo: si $M1 contiene "diosa", `echo ucfirst($M1);` devuelve "Diosa".

- **ucword**s(string): devuelve la misma cadena convirtiendo el primer carácter de todas las palabras en mayúscula.

  Ejemplo: si $M_pal contiene "diosa de la antigüedad egipcia", la instrucción `echo ucwords($M_pal);` devuelve "Diosa De La Antigüedad Egipcia".

2. FUNCIONES ÚTILES

STRLEN

strlen(string): devuelve la longitud de una cadena expresada en valor numérico entero.

Ejemplo: si $largo contiene "diosa de la antigüedad egipcia", la instrucción `echo strlen($largo);` devuelve el valor numérico 30.

2. FUNCIONES ÚTILES

LIMPIAR CADENAS DE CARACTERES

- **chop(string)** elimina los espacios en blanco que haya al final de una cadena de caracteres, incluyendo los códigos de fin de línea si los hubiere.

Ejemplo:

```
$f="La ciencia es un conjunto de verdades          \n ";
$g=chop($f);
echo "La variable \$f contiene ".strlen($f)." caracteres de los que ".(strlen($f)-
  strlen($g))."son espacios en blanco o saltos de línea.";
echo "En cambio, si usamos la función chop() la variable \$f contiene sólo
  ".strlen($g). "caracteres.";
```

generan la página siguiente:

```
La variable $f contiene 44 caracteres de los que 7 son espacios en blanco o saltos de
  línea.
En cambio, si usamos la función chop() la variable $f contiene sólo 37 caracteres.
```

2. FUNCIONES ÚTILES

LIMPIAR CADENAS DE CARACTERES

- **ltrim(string)** elimina los espacios en blanco que haya al principio de una cadena de caracteres.

Ejemplo:

```
$h="       La ciencia es un conjunto de verdades";
$i=ltrim($i);
echo "La variable \$h contiene ".strlen($h)." caracteres de los que
  ".(strlen($h)-strlen($i))."son espacios en blanco.";
echo "En cambio, si usamos la función ltrim() la variable \$i contiene sólo
  ".strlen($i)." caracteres.";
```

generan la página siguiente:

```
La variable $h contiene 42 caracteres de los que 5 son espacios en blanco.
En cambio, si usamos la función ltrim() la variable $h contiene sólo 37
caracteres.
```

2. FUNCIONES ÚTILES

LIMPIAR CADENAS DE CARACTERES

- **trim(string)** es una combinación de las dos anteriores: elimina los espacios en blanco que haya al principio y al final de una cadena.
- **strip_tags(string)** elimina los controles HTML de una cadena.

  Ejemplo:

```
$quita_html="<H1>Texto grande</H1> <B>Negrita</B><P>";
echo $quita_html;
echo strip_tags($quita_html);
```

Generan:

Texto grande

Negrita

Texto grande Negrita

2. FUNCIONES ÚTILES

FUNCIONES CON CADENAS

- **str_repeat(string, number)**: devuelve la misma cadena que toma como primer argumento repetida tantas veces como se indique en el segundo argumento.

  Ejemplo: si $repe contiene "diosa
", echo str_repeat($repe,3); devuelve la cadena "diosa" repetida tres veces en distintas líneas.

- **strtr**(): devuelve la misma cadena sustituyendo los caracteres de la misma que se indiquen en el segundo argumento por los que se indiquen en el tercer argumento haciéndolos equivaler uno a uno.

  Ejemplo: si $cambia contiene "Los días perdidos en verano" y necesitamos sustituir las as por us y las os por as, podemos escribir la sentencia siguiente: echo strtr($cambia,"ao","ua"); La función devolverá la cadena "Las díus perdidas en veruna".

2. FUNCIONES ÚTILES

FUNCIONES CON CADENAS

- **str_replace(find, replace, string)**: que reemplaza todas las apariciones del string buscado con el string de reemplazo.

  Ejemplo:

  ```
  str_replace("world","Peter","Hello world!");
  ```

  Devolveria: `"Hello Peter"`

2. FUNCIONES ÚTILES

FUNCIONES CON CADENAS

- Ej con arrays

```
$frase = "Deberías comer pizza, refrescos, y helado todos los días.";
$saludable = array("fruta", "vegetales", "fibra");
$no_saludable= array("pizza", "refrescos", "helado");
$nueva_frase = str_replace($no_saludable, $saludable, $frase);
```

La función devolverá la cadena:

```
"Deberías comer fruta, vegetales, y fibra todos los días.".
```

2. FUNCIONES ÚTILES

FUNCIONES CON CADENAS

- **strrev(string)** recibe una cadena de caracteres y la devuelve al revés.

  Por ejemplo, la sentencia echo strrev("paseo"); devuelve "oesap".

- **str_split(string)** separa los caracteres de una cadena y los introduce en un array.

2. FUNCIONES ÚTILES

FUNCIONES CON CARACTERES

- **chr(number)** recibe un número entero y devuelve el carácter correspondiente del código ASCII.

  Por ejemplo, echo chr(65); devuelve la letra A mayúscula.

- **ord(string)** recibe un carácter del código ASCII y devuelve el número entero correspondiente.

  Por ejemplo, echo ord("A"); devuelve el número 65.

2. FUNCIONES ÚTILES

FUNCIONES CON CARACTERES

- **chr(number)** recibe un número entero y devuelve el carácter correspondiente del código ASCII.

  Por ejemplo, echo chr(65); devuelve la letra A mayúscula.

- **ord(string)** recibe un carácter del código ASCII y devuelve el número entero correspondiente.

  Por ejemplo, echo ord("A"); devuelve el número 65.

2. FUNCIONES ÚTILES

FUNCIONES CON FECHAS

- **checkdate(month, day, year)** nos permite comprobar si una fecha, que se pone como argumento, es válida. Si lo es, la función devuelve el valor True (verdadero). En el caso contrario, devuelve False (falso).

  Ej: checkdate(0,25,2000); //False.

  checkdate(7,25,2000); //True.

2. FUNCIONES ÚTILES

FUNCIONES CON FECHAS

- **date(format, [timestamp])** devuelve una cadena de texto que refleja una fecha y una hora formateadas como se indique en el primer parámetro. En el segundo argumento puede indicarse la fecha que se quiere mostrar utilizando un valor de tipo timestamp.

Un "timestamp" es una representación numérica de una fecha y hora específica, generalmente expresada en segundos o milisegundos desde un punto de referencia, generalmente el 1 de enero de 1970 .

format puede ser:

- j: El dia del mes (1 to 31)
- n: El número del mes (1 al 12)
- Y: año con cuatro dígitos
- H: horas de 0 a 23
- i: minutos de 00 a 59.

2. FUNCIONES ÚTILES

FUNCIONES CON FECHAS

Ejemplo:

```
$fecha  = date ("j/n/Y H:i");
print ("$fecha");
```

Resultado: 26/9/2005 17:36

2. FUNCIONES ÚTILES

FUNCIONES CON FECHAS

- **strtotime(fecha)** devuelve el timestamp de la fecha dada.

```
$timestamp = strtotime("2023-10-26 15:30:00");
$fecha_formateada = date("d/m/Y H:i:s", $timestamp);
```

- **mktime()** se usa para extraer el valor timestamp de una fecha.

```
mktime(horas,minutos,segundos,mes,día,año);
```

Por ejemplo, si queremos saber el momento timestamp del día 16 de diciembre de 1999, a las 10 horas, 15 minutos y 20 segundos, debemos escribir la instrucción `$momento=mktime(10,15,20,12,16,1999);`. El resultado es 945339320.

2. FUNCIONES ÚTILES

FUNCIONES CON FECHAS

- **getdate()** se usa para extraer información de una fecha dada. Si no se pone nada, el timestamo es la fecha actual.

```
$fecha_actual = getdate();
echo "Día de la semana: " . $fecha_actual['wday'] . "<br>";
echo "Día del mes: " . $fecha_actual['mday'] . "<br>";
echo "Mes: " . $fecha_actual['mon'] . "<br>";
echo "Año: " . $fecha_actual['year'] . "<br>";
echo "Hora: " . $fecha_actual['hours'] . "<br>";
echo "Minutos: " . $fecha_actual['minutes'] . "<br>";
echo "Segundos: " . $fecha_actual['seconds'] . "<br>";
```

2. FUNCIONES ÚTILES

FUNCIONES CON ARRAYS

- **current(array)** devuelve el contenido de elemento sobre el que está el puntero sin variar su posición.
- **next(array)** devuelve el contenido del siguiente elemento según la posición donde esté el puntero y desplaza éste a esa posición nueva.

Ej:

```
$personas = array("Juan", "Maria", "Sonia", "Pedro");
echo current($personas); //Devuelve Juan
echo next($personas); //Devuelve Maria
echo current($personas); //Devuelve Maria
```

2. FUNCIONES ÚTILES

FUNCIONES CON ARRAYS

- **reset(array)** coloca el puntero de una matriz sobre su primer elemento y devuelve su contenido..
- **end(array)** coloca el puntero de una matriz sobre su último elemento y devuelve su contenido.

Ej:

```
$personas = array("Juan", "Maria", "Sonia", "Pedro");
echo next($personas); //Devuelve Maria
echo reset($personas); //Devuelve Juan
echo end($personas); //Devuelve Pedro
```

2. FUNCIONES ÚTILES

FUNCIONES CON ARRAYS

- **count(array)** cuenta los elementos que integran una matriz y devuelve un número entero. **sizeof(array)** funciona igual.
- **explode(carácter separador, cadena)** permite convertir una cadena de caracteres en una matriz mediante un separador dado.

```
$cadena_nombres = "Juan,María,Pedro,Laura";
$nombres = explode(",", $cadena_nombres);
foreach ($nombres as $nombre) {
    echo $nombre . "<br>";
}
```

2. FUNCIONES ÚTILES

FUNCIONES CON ARRAYS

- **implode(carácter separador, matriz)** lleva a cabo la operación inversa: lleva los elementos de una matriz a una cadena separándolos como se indique.

```
$colores = array("rojo", "verde", "azul", "amarillo");
$cadena_colores = implode(", ", $colores);
echo $cadena_colores;
```

- **is_array(variable)**: Devuelve true si es un array.

2. FUNCIONES ÚTILES

FUNCIONES CON ARRAYS

- **in_array(elem, array)** devuelve True si un valor está contenido en alguno de los elementos de una matriz y False si no lo está.
- **array_unshift(arr, v1,v2,..)**: Inserta al principio v1, v2.. Al pincipio del array.

```
$frutas = array("manzana", "naranja", "plátano");
array_unshift($frutas, "fresa", "uva");
```

- **array_push(array, v1,v2,..)**: Inserta al final del array v1, v2...

2. FUNCIONES ÚTILES

FUNCIONES CON ARRAYS

- **array_pad(array, size, var)**: Inserta elementos con el valor var las veces necesarios para que el tamaño del array sea size.

  Ejemplo:

```
$frutas = array("manzana", "naranja", "plátano");
// Rellenar el array para que tenga al menos 5 elementos
$frutas_rellenadas = array_pad($frutas, 5, "uva");
// Mostrar el contenido del array rellenado
print_r($frutas_rellenadas);  //Array ( [0] => manzana [1] =>
  naranja [2] => plátano [3] => uva [4] => uva )
```

2. FUNCIONES ÚTILES

FUNCIONES CON ARRAYS

- **list(var1, var2...)**: Se utiliza para asignar valores a una lista de variables a partir de un array.

  Ejemplo:

```
$frutas = array("manzana", "plátano", "naranja");

list($fruta1, $fruta2, $fruta3) = $frutas;

echo $fruta1; // Muestra "manzana"
echo $fruta2; // Muestra "plátano"
echo $fruta3; // Muestra "naranja"
```

2. FUNCIONES ÚTILES

FUNCIONES CON ARRAYS

- **sort(array)** ordena un array de índices en orden ascendente.
- **rsort(array)** ordena un array de índices en orden descendiente.

2. FUNCIONES ÚTILES

FUNCIONES CON ARRAYS ASOCIATIVOS

- **asort(array)** ordena un array asociativo en orden ascendente por el valor.
- **arsort(array)** ordena un array asociativo en orden descendiente por el valor.
- **ksort(array)** ordena un array asociativo en orden ascendente por la clave.
- **krsort(array)** ordena un array asociativo en orden descendiente por la clave.

UT 4: FUNCIONES

1. Funciones
2. Funciones útiles
3. **Ficheros externos**
    1. require
    2. include
    3. Diferencia entre require e include

3. FICHEROS EXTERNOS

Existen dos maneras para añadir otros ficheros a nuestros scripts en PHP:

- require
- include

UT 4: FUNCIONES

1. Funciones
2. Funciones útiles
3. Ficheros externos
    1. **require**
    2. include
    3. Diferencia entre require e include

3.1 REQUIRE

Se puede usar de dos maneras

```
require 'nombrefichero';

require ('nombrefichero');
```

3.1 REQUIRE

Hay dos tipos de errores que pueden ocurrir:

- **E_COMPILE_ERROR**: Cuando el archivo incluido da error, el script se detiene.
- «**Failed to open stream: No such file or directory**»: Se produce cuando el fichero incluido no se encuentra. El script se sigue ejecutando.

3.1 REQUIRE

ÁMBITO DE LAS VARIABLES

Cuando se añade un fichero, en ese momento se puede hacer uso de las variables que se haya declarado en ese fichero.

EJEMPLO DE REQUIRE

```
$nombre = 'Juan';
$apellido = 'Nadie';
```

```
//la línea inferior no muestra el contenido de las variables porque no existen aun en el script
echo "El nombre es $nombre y el apellido es $apellido";

//realizamos el include
require 'nombreapellido.php';

//esta línea si muestra el contenido de las variables porque ya existen en el include
echo "El nombre es $nombre y el apellido es $apellido";
```

3.1 REQUIRE

ÁMBITO DE LAS FUNCIONES

El ámbito de las funciones es igual que el de las variables, están disponibles una vez que se añade el fichero.

3.1 REQUIRE

REQUIRE_ONCE

require_once tiene el mismo comportamiento que **require**, la única diferencia entre ambas es que si el código del fichero ya se ha incluido, este no se incluye de nuevo.

UT 4: FUNCIONES

1. Funciones
2. Funciones útiles
3. Ficheros externos
    1. require
    2. **include**
    3. Diferencia entre require e include

3.2 INCLUDE

Se puede usar de dos maneras

```
include 'nombrefichero';

include ('nombrefichero');
```

3.2 INCLUDE

Hay dos tipos de errores que pueden ocurrir:

- **Aviso Warning**: El mensaje que nos aparece es el siguiente: «failed to open stream: No such file or directory«, se da cuando el fichero incluido no se encuentra. El script se sigue ejecutando.
- **Error**: Cuando se realiza el include se verifica el código del archivo incluido, en caso de ser un fichero PHP y que que contenga un error nos saldrá un mensaje por pantalla.

3.2 INCLUDE

ÁMBITO DE LAS VARIABLES Y DE FUNCIONES

Funciona igual que con el require.

3.2 INCLUDE

INCLUDE_ONCE

include_once tiene el mismo comportamiento que **include**, la única diferencia entre ambas es que si el código del fichero ya se ha incluido, este no se incluye de nuevo.

UT 4: FUNCIONES

1. Funciones
2. Funciones útiles
3. Ficheros externos
    1. require
    2. include
    3. **Diferencia entre require e include**

3.3 DIFERENCIA ENTRE REQUIRE E INCLUDE

La diferencia es:

- **include** inserta en nuestro script un código procedente de otro archivo, si no existe dicho archivo o si contiene algún tipo de error nos mostrará un ‘warning‘ por pantalla y **el script seguirá ejecutándose**.
- require hace la misma operación que include, pero en caso de no existir el archivo o error en el mismo mostrará un ‘fatal error‘ y **el script no se sigue ejecutando**.

Se recomienda usar **require** siempre que el código sea importante (Funciones reutilizables de PHP, configuraciones…), mientras que **include** se usa en casos en los que el código no es vital para la ejecución del script (cabeceras y pies HTML o similares).