UT3
PROGRAMACIÓN BASADA EN LENGUAJES DE MARCAS CON CÓDIGO EMBEBIDO

Desarrollo Web en Entorno Servidor

2° DAW

Índice

1. Estructuras de control
   a. Sentencias condicionales
   b. Bucles
2. Funciones
   a. Inclusión de ficheros externos
   b. Ejecución y creación de funciones
   c. Argumentos
3. Tipos de datos compuestos
   a. Recorrer un array
   b. Funciones relacionadas con los tipos de datos compuestos
4. Formularios
   a. Procesamientos de la información devuelta por formularios
   b. Generación de formularios en PHP

Estructuras de control

- Dos tipos:
  - Condicionales
  - Repetitivas o bucles
- También existe la sentencia goto (NO RECOMENDABLE) que permite saltar a otro punto del código indicado con una etiqueta

```php
<?php
        $a = 1;
        goto salto;
        $a++; //esta sentencia no se ejecuta
        salto:
        echo $a;
        // el valor obtenido es 1
 ?>
```

Estructuras de control: sentencias condicionales

- **if / elseif / else:** permite definir una expresión para ejecutar o no la sentencia o conjunto de sentencias siguientes

```
<?php
    if ($a < $b)
            print "a es menor que b";
    elseif ($a > $b)
            print "a es mayor que b";
    else
            print "a es igual a b";
?>
```

- **switch:** similar a enlazar varias sentencias if comparando una misma variable con diferentes valores

```
<?php
    switch ($a) {
            case 0:
                    print "a vale 0";
                    break;
            case 1:
                    print "a vale 1";
                    break;
            default:
                    print "a no vale 0 ni 1";
    }
?>
```

Estructuras de control: bucles

- **while**: define un bucle que se ejecuta mientras se cumpla una expresión. La expresión se evalúa antes de comenzar cada ejecución del bucle.

```
<?php
$a = 1;
while ($a < 8)
        $a += 3;
print $a; // el valor obtenido es 10
?>
```

- **do..while**: similar al bucle while, pero la expresión se evalúa al final, con lo cula se asegura que la sentencia/as del bucle se ejecutan al menos una vez.

```
<?php
$a = 1;
do
   $a -=3;
while ($a > 10)
print $a; // el valor obtenido es -2
?>
```

Estructuras de control: bucles

- **for**: compuesto por tres expresiones:

  for (expr1; expr2; expr3)

  sentencia o conjunto de sentencias;

  - expr1: se ejecuta una vez al comienzo del bucle.
  - expr2: se evalúa para saber si se debe ejecutar o no la/as sentencia/as
  - expr3: se ejecuta tras ejecutar todas las sentencias del bucle

```
<?php
for ($a = 5; $a<10; $a+=3) {
        print $a; // Se muestran los valores 5 y 8
        print "<br />";
}
?>
```

Funciones

- Permiten **asociar una etiqueta** (el nombre de la función) con un bloque de código a ejecutar.
- Al usar funciones estamos ayudando a **estructurar mejor** el código.
- Las funciones permiten crear variables locales que no serán visibles fuera del cuerpo de las mismas.

Funciones: inclusión de ficheros externos

- En ocasiones resulta más cómodo agrupar las funciones en ficheros externos al que se hará referencia.
- Formas de incorporar ficheros externos:
  - **include**: evalúa el contenido del fichero que se indica y lo incluye como parte del fichero actual en el punto de realización de la llamada. Se puede indicar la ruta de forma absoluta o de forma relativa. Se toma como base la ruta que se especifica en la directiva **include_path** del fichero **php.ini**, si no se encuentra en esa ubicación, se buscará en el directorio actual

```php
<?php
      include 'funciones.inc.php';
      print fecha();
?>
```

  - **include_once**: si por equivocación incluyes más de una vez un mismo fichero, lo normal es que obtengas algún tipo de error (por ejemplo, al repetir una definición de una función)
  - **require**: si el fichero que queremos incluir no se encuentra, **include** da un aviso y continua la ejecución del guión. La diferencia más importante al usar **require** es que en ese caso, cuando no se puede incluir el fichero, se detiene la ejecución del guion.
  - **require_once**: es la combinación de las dos anteriores.

Funciones: ejecución y creación

- Para crear tus propias funciones se usa la palabra **function**

```php
<?php
        function precio_con_iva(){
                global $precio;
                $precio_iva = $precio * 1.21;
                print "El precio con IVA es ".$precio_iva;
        }
        $precio = 10;
        precio_con_iva();
?>
```

Funciones: ejecución y creación

- No es necesario definir las funciones antes de usarlas excepto cuando están definida condicionalmente:

```php
<?php
$iva = true;
$precio = 10;
precio_con_iva(); // Da error, pues aquí aún no //está definida
if ($iva) {
        function precio_con_iva() {
                global $precio;
                $precio_iva = $precio * 1.21;
                print "El precio con IVA es ".$precio_iva;
        }
}
precio_con_iva(); // Aquí ya no da error
?>
```

Funciones: argumentos

- Siempre es mejor utilizar argumentos o parámetros al hacer la llamada. Además, en lugar de mostrar el resultado en pantalla o guardar el resultado en una variable global, las funciones pueden devolver un valor usando la sentencia **return**.
- Cuando en una función se encuentra una sentencia return, termina su procesamiento y devuelve el valor que se indica

```php
<?php
        function precio_con_iva($precio) {
                return $precio * 1.21;
        }
        $precio=10;
        $precio_iva =precio_con_iva($precio);
        print "El precio con IVA es ".$precio_iva;
?>
```

Funciones: argumentos

- Al definir la función, puedes indicar **valores por defecto** para los argumentos, de forma que cuando hagas una llamada a la función puedes no indicar el valor de un argumento; en este caso se toma el valor por defecto indicado.

```
<?php
        function precio_con_iva($precio, $iva=0.21) {
                return $precio * (1 + $iva);
        }
        $precio=10;
        $precio_iva =precio_con_iva($precio);
        print "El precio con IVA es ".$precio_iva;
?>
```

Funciones: argumentos

- Los argumentos anteriores se pasaban **por valor**. Esto es, cualquier cambio que se haga dentro de la función a los valores de los argumento no se reflejará fuera de la función.
- Si quieres que esto ocurra debes definir el parámetro para que su valor se pase por **referencia**, añadiendo el símbolo **&** antes de su nombre.

```php
<?php
        function precio_con_iva(&$precio , $iva=0.21) {
                $precio = $precio * (1 + $iva);
        }
        $precio=10;
        precio_con_iva($precio);
        print "El precio con IVA es ".$precio ."<br/>";
?>
```

Tipos de datos compuestos

- Un tipo de datos compuesto es aquel que te permite almacenar más de un valor. En PHP puedes utilizar dos tipos de datos compuestos: el **array** y el **objeto.**
- Un **array** es un tipo de datos que nos permite almacenar varios valores. Cada miembro del array se almacena en una posición a la que se hace referencia utilizando un valor clave. Las claves pueden ser numéricas o asociativas.

```php
<?php
    //array numérico
    $modulos1 = array(0 => "Programación", 1 => "Bases de datos", ..., 9 =>
    "Desarrollo web en entorno servidor");


    //array asociativo
    $modulos2 = array("PR" => "Programación", "BD" => "Bases de datos",
    ..., "DWES" => "Desarrollo web en entorno servidor");
?>
```

- La función **print_r,** que nos muestra todo el contenido del array que le pasamos.
- Para hacer referencia a los elementos almacenados en un array, hay que utilizar el valor clave entre corchetes:
  - $modulos1 [9];
  - $modulos2 ["DWES"];

Tipos de datos compuestos

- En PHP se pueden crear arrays de varias dimensiones almacenando otro array en cada uno de los elementos de un array.

```
$ciclos = array(
            "DAW" => array ("PR" => "Programación", "BD" => "Bases de datos", ..., "DWES" => "Desarrollo web en entorno servidor"),
            "DAM" => array ("PR" => "Programación", "BD" => "Bases de datos", ..., "AD" => "Acceso a datos")
            );
```

- Para hacer referencia a los elementos almacenados en un **array multidimensional,** hay que indicar las claves para cada una de las dimensiones:
  - $ciclos ["DAW"]["DWES"]
- No hay que indicar el tamaño del array para crearlo. Ni siquiera es necesario indicar que una variable concreta es de tipo array:

```
$modulos1[0] = "Programación";
$modulos1[1] = "Bases de datos";
$modulos1[9] = "Desarrollo web en entorno servidor";
```

- Ni siquiera es necesario especificar el valor de la clave. Si se omite, el array se irá llenando a partir de la última clave numérica existente, o de la posición 0 si no existe ninguna:

```
$modulos1[] = "Programación";
$modulos1[] = "Bases de datos";
$modulos1[] = "Desarrollo web en entorno servidor";
```

Recorrer un array

- Las **cadenas de texto** o **strings** se pueden tratar como arrays en los que se almacena una letra en cada posición, siendo 0 el índice correspondiente a la primera letra, 1 el de la segunda, etc.

```
$modulo = "Desarrollo web en entorno servidor";
// $modulo[3] == "a";
```

- Para recorrer un array se puede utilizar el bucle **foreach**. Utiliza una variable temporal para asignarle en cada iteración el valor de cada uno de los elementos del arrays. Puedes usarlo de dos formas. Recorriendo sólo los elementos:

```
foreach ($modulos1 as $modulo)
        echo $modulo . "<br/>";
```

- O recorriendo los elementos y sus valores clave de forma simultánea:

```
foreach ($modulos2 as $codigo => $modulo)
        echo "El código del módulo ".$modulo." es
   ".$codigo."<br/>";
```

Funciones relacionadas con los tipos de datos compuestos

- En PHP existen funciones específicas para comprobar y establecer el tipo de datos de una variable, **gettype** obtiene el tipo de la variable que se le pasa como parámetro y devuelve una cadena de texto, que puede ser array, boolean, double, integer, object, string, null, resource o unknown type.
- También podemos comprobar si la variable es de un tipo concreto utilizando una de las siguientes funciones: **is_array()**, **is_bool()**, **is_float()**, **is_integer()**, **is_null()**, **is_numeric()**, **is_object()**, **is_resource()**, **is_scalar()** e **is_string()**. Devuelven true si la variable es del tipo indicado.

```php
<?php
//asignamos a las dos variables la misma cadena de texto
$a = $b = "3.1416";
settype($b, "float"); //y cambiamos $b a tipo float
print "\$a vale $a y es de tipo ".gettype($a);
print "<br />";
print "\$b vale $b y es de tipo ".gettype($b);
?>
```

- Si sólo nos interesa saber si una variable está definida y no es **null**, puedes usar la función **isset**.
- La función **unset** destruye la variable o variables que se le pasa como parámetro.

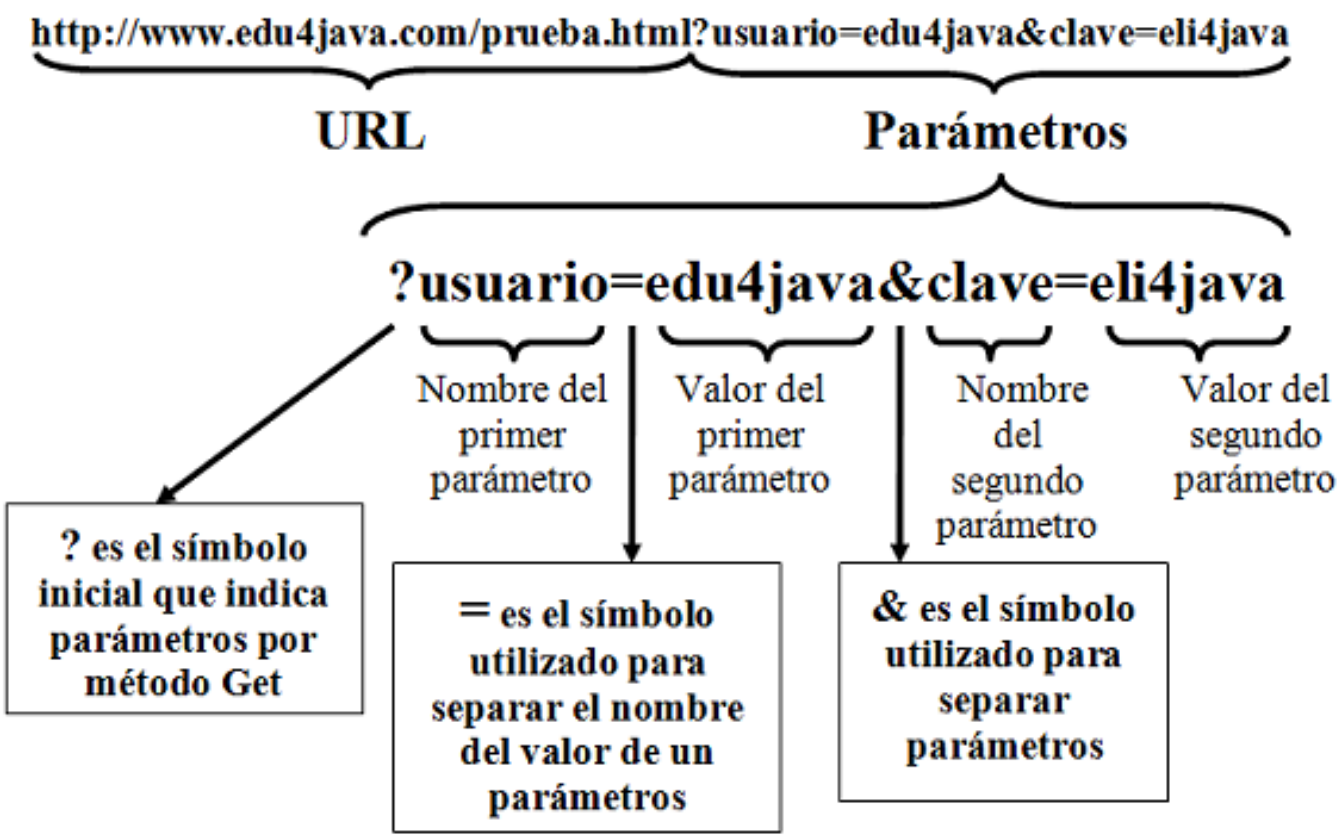
Formularios

- Son la forma de hacer llegar los datos a una aplicación web
- Van encerrados en las etiquetas
  - <form> </form>
- Dentro de las etiquetas de formulario se pueden incluir elementos sobre los que puede actuar el usuario. Ejemplo:
  - <input>
  - <select>
  - <textarea>
  - <button>
- En el atributo **action** del FORM se indica la página a la que se enviarán los datos del formulario
- En el atributo **method** se especifica el método usado para enviar la información:
  - **get:** los datos se envían en la URI utilizando el signo **?** como separador.
  - **post:** los datos se incluyen en el cuerpo del formulario y se envían usando el protocolo HTTP

Formularios: métodos GET y POST

- Son métodos del protocolo HTTP para intercambio de información entre cliente y servidor.
  - **get:** método más usado. Pide al servidor que le devuelva al cliente la información identificada en la URI.
  - **post:** se usa para enviar información a un servidor. Puede ser usado para completar un formulario de autenticación, por ejemplo.
- La principal diferencia radica en la codificación de la información.
- **Método GET:** utiliza la dirección URL que está formada por:
  - **Protocolo**: especifica el protocolo de comunicación
  - **Nombre de dominio:** nombre del servidor donde se aloja la información.
  - **Directorios:** secuencia de directorios separados por /que indican la ruta en la que se encuentra el recurso
  - **Fichero:** nombre del recurso al que acceder.
  - Detrás de la URL se coloca el símbolo **?** Para indicar el comienzo de las variables con valor que se enviarán, separadas cada una de ellas por **&.**
  - **Ejemplo:** http://www.exmaple.org/file/example1.php?v1=0&v2=3

Formularios: método GET



Formularios: método GET


```
<!DOCTYPE html>
<html>
<head>
    <title>Ejemplo Formularios</title>
</head>
<body>
    <h1>Ejemplo de procesado de formularios</h1>
    <form action="ejemplo1.php" method="get">
          Introduzca su nombre:
          <input type="text" name="nombre">
          <br/>
          Introduzca sus apellidos:<input type="text" name="apellidos"><br/>
          <input type="submit" value="Enviar">
    </form>
</body>
</html>
```

Formularios: método POST

- **Método POST:** la información va codificada en el cuerpo de la petición HTTP y por tanto viaja oculta.
- No hay un método más seguro que otro, ambas tienen sus pros y sus contras.
- Es conveniente usarlos GET para la recuperación y POST para el envío de información.

Formularios: método POST


```
<!DOCTYPE html>
<html>
<head>
    <title>Ejemplo Formularios</title>
</head>
<body>
    <h1>Ejemplo de procesado de formularios</h1>
    <form action="ejemplo1.php" method="post">
          Introduzca su nombre:
          <input type="text" name="nombre">
          <br/>
          Introduzca sus apellidos:<input type="text" name="apellidos"><br/>
          <input type="submit" value="Enviar">
    </form>
</body>
</html>
```

Hoja03_PHP_04 (Ej 1 y 2)

Formularios: recuperación de información con GET

- En el caso del envío de información utilizando el método GET existe una variable especial $_GET, donde se almacenan todas las variables pasadas con este método.
- La forma de almacenar la información es un array en el que el índice es el nombre asignado al elemento del formulario
  - $_GET['nombre'];
  - $_GET['apellidos'];
- También se puede recuperar con la función **print_r** que muestra el array completo.

```
<?php
   print_r($_GET);
?>
```

Formularios: recuperación de información con POST

- Al igual que en el método GET, se utiliza una variable interna $_POST, donde se almacenan todas las variables pasadas con este metodo.
- La forma de almacenar la información, también es un array en el que el índice es el nombre asignado al elemento del formulario
  - $_POST['nombre'];
  - $_POST['apellidos'];
- También se puede recuperar con la función print_r que muestra el array completo.
- Existe otra variable $_REQUEST que contiene tanto el contenido de $_GET como $_POST

Hoja03_PHP_04 (Ej 3 y 4)

Formularios: validación de datos

- Siempre que sea posible, es preferible **validar los datos que se introducen en el navegador antes de enviarlos. Para ello deberás usar código en lenguaje Javascript.**
- Si por algún motivo hay datos que se tengan que validar en el servidor, por ejemplo, porque necesites comprobar que los datos de un usuario no existan ya en la base de datos antes de introducirlos, será necesario hacerlo con código PHP en la página que figura en el atributo action del formulario.

Formularios: validación de datos

- Si hay que validar datos en php, será necesario hacerlo en la página que se indica en el atributo **action** del formulario.
- Una posibilidad es usar la misma página que muestra el formulario como destino de los datos:
  - Si tras comprobar que los datos son correctos, se reenvía a otra página.
  - Si son incorrectos se muestra el mismo formulario indicando los datos correctos y los incorrectos y por qué.
- Para comprobar si la página recibe o no los datos, se puede usar la función **isset** con una variable de las que se deben recibir.

Formularios: validación de datos


```
<html>
<head> <title>Desarrollo Web</title> </head>
<body>
<?php
if (isset($_POST['enviar'])) {
    $nombre = $_POST['nombre'];
    $modulos = $_POST['modulos'];
    print "Nombre: ".$nombre."<br />";
            foreach ($modulos as $modulo) { print "Modulo: ".$modulo."<br />"; }
}
else {
?>
    <form name="input" action="<?php echo $_SERVER['PHP_SELF'];?>" method="post">
    Nombre del alumno: <input type="text" name="nombre" /><br />
    <p>Módulos que cursa:</p>
    <input type="checkbox" name="modulos[]" value="DWES" />Desarrollo web en entorno servidor<br />
    <input type="checkbox" name="modulos[]" value="DWEC" />Desarrollo web en entorno cliente<br /> <br
    />
    <input type="submit" name="enviar" value="Enviar" />
    </form>
<?php       }   ?>
</body>
</html>
```

Hoja03_PHP_04