DESARROLLO WEB EN ENTORNO SERVIDOR

Apuntes 1ªEvaluación

Descripción breve

Apuntes de l asignatura Desarrollo Web En Entorno Servidor del 2º curso del grado superior DAW

Raquel Rodríguez
raquelsweeta@gmail.com

Contenido

Comentarios en PHP

```php
<?php
    //comentario de una sólo línea
    $var1 = 5;
    /*comentario
    de varias líneas */
    $var2 = 10; #comentario al final de una línea
?>
```

Variables y constantes

En PHP las variables se representan con un signo de dólar seguido por el nombre de la variable. El nombre de la variable **es sensible a mayúsculas y minúsculas**.

```
<?php
$nombre = "Hola";
$Nombre = " mundo";
echo $nombre . $Nombre; #imprime Hola mundo
```

Ámbito de las variables

El ámbito de una variable es el contexto dentro del que la variable está definida. La mayor parte de las variables PHP sólo tienen un ámbito simple. Este ámbito simple también abarca los ficheros incluidos y los requeridos. Por ejemplo:

```
<?php
$a = 5;
include "archivo2.php";
```

Aquí, la variable *$a* estará disponible en el interior del *script* incluido *archivo2.php*

Tenemos varias formas de poder acceder a la variable $a del anterior ejemplo. Podemos usar la palabra reservada **global** para indicarle a la función que la variable $a es una variable global.

```
<?php
$a = 5;
function test() {   global $a;
   echo "El valor de a es $a";
}
test();
```

Isset y unset

Para comprobar si una variable está definida y no es null podemos utilizar la función isset, que devolverá TRUE si la variable existe y no es nula.

```
if (isset($a)) {
   # codigo...
} else {
   # codigo...
```

}

Si queremos eliminar una variable utilizaremos unset.

Constantes

Podemos definir constantes usando la función define o con la palabra reservada **const**. Si usamos la función *define()* debemos pasar como primer parámetro el nombre de la constante y el valor como segundo.

```
define('FRASE', 'Hola mundo');

const PI = 3.1416;
```

Operador ternario

Otro operador condicional es el operador ternario (*?*).

```
$nombre = isset($_POST['nombre']) ? $_POST['nombre'] : 'Pepe';
echo $nombre;
```

Es equivalente a la siguiente sentencia:

```
if (isset($_POST['nombre'])) {
    $nombre = $_POST['nombre'];
} else {
    $nombre = 'Pepe';
}
```

Operadores nave espacial y fusión de null

El operador *nave espacial* (⇔) se emplea para comparar dos expresiones. Devuelve -1, 0 o 1 cuando $a es respectivamente menor, igual, o mayor que $b.

```
// Numeros enteros

echo 1 <=> 1; // 0

echo 1 <=> 2; // -1

echo 2 <=> 1; // 1

// Numeros decimales

echo 1.5 <=> 1.5; // 0

echo 1.5 <=> 2.5; // -1

echo 2.5 <=> 1.5; // 1

// Cadenas de caracteres

echo "a" <=> "a"; // 0
```


```
echo "a" <=> "b"; // -1

echo "b" <=> "a"; // 1
```

Bucles

While

Los bucles while son el tipo de bucle más sencillo en PHP. Se comportan igual que su contrapartida en C. La forma básica de una sentencia *while* es:

```
while ($a <= 10) {
   # codigo...
}
```

do-while

Los bucles do-while son muy similares a los bucles *while*, excepto que la condición es verificada al final de cada iteración, en lugar de al principio. La principal diferencia es que en un bucle *do-while* se ejecutará la primera iteración siempre, mientras que un bucle *while* no tiene por qué.

```
do {
   # codigo...
} while ($a <= 10);
```

for

La sintaxis de un bucle *for* es:

```
for ($i=0; $i < 5; $i++) {
   # codigo...
}
```

PHP ofrece la posibilidad de usar expresiones vacías en los bucles *for*, lo que puede resultar útil en alguna ocasión.

```
for ($i=0; ; $i++) {
   if ($i < 5) {
      break;
   }
   echo $i;
}
```

Para salir de cualquier bucle, se puede utilizar *break*, mientras que para continuar con la siguiente iteración podemos usar *continue*.

foreach

El constructor foreach proporciona un modo sencillo de iterar sobre *arrays*. *foreach* funciona sólo sobre *arrays* y objetos, y emitirá un error al intentar usarlo con una variable de un tipo diferente de datos o una variable no inicializada. Existen dos sintaxis:

```
foreach ($array1 as $value) {
   # codigo...
}
```

```
foreach ($array1 as $key => $value) {
   # codigo...
}
```

La primera forma recorre el *array $array1*. En cada iteración el valor del elemento actual se asigna a *$value* y el puntero interno del array avanza una posición.

La segunda forma además asigna la clave del elemento actual a la variable *$key* en cada iteración.

Incluir archivos externos

Para incluir archivos externos podemos usar dos sentencias: **include** o **require**. La sentencia include incluye y evalúa el archivo especificado.

Por ejemplo, si tenemos el fichero *archivo1.php* con el siguiente contenido:

```
echo "Hola, soy el archivo 1";
```

Lo podemos incluir en un segundo fichero *archivo2.php*:

```
echo "Hola, soy el archivo 2";
include "archivo1.php";
```

La sentencia require es exactamente igual que *include*, excepto que en caso de fallo producirá un error fatal de nivel *E_COMPILE_ERROR*. En otras palabras, éste detiene el *script*, mientras que *include* sólo emitirá una advertencia, lo cual permite continuar el *script*.

Para ver la diferencia, modifica *archivo2.php* y cambia la ruta de *include* por algún archivo que no exista. A continuación, saca por pantalla la frase *Hola mundo*:

```
echo "Hola, soy el archivo 2";
include "archivo3.php";
echo "Hola mundo";
```

EXAMEN

- Para generar el proyecto comando **“composer init”**
- Para actualizar el proyecto “composer update”
- Para instalar las librerías “composer require <nombre-librería>”
- Cuando borre carpeta vendor comando “composer install”

Estructura PRÁCTICA 1

Htdocs > DWS > proyecto1 > api (privado) > html (público)

En archivo helpers.php incluir el siguiente código:

```
define("BASE_DIR", dirname(__DIR__));

if (! function_exists('base_path')) {
/**
* Devuelve la ruta indicada desde la ruta raiz del proyecto.
*
* @param  string  $path
* @return string
*/
function base_path(string $path = ''): string {
return BASE_DIR . "/" . ltrim($path, "/");
}
}
```

En archivo index.php incluir el siguiente código:

```
use App\App;

require_once __DIR__ . "/vendor/autoload.php";

$app = new App();
$app->run();
```

```
1     define("BASE_DIR", dirname(__DIR__));
2
3     if (! function_exists('base_path')) {
4         /**
5          * Devuelve la ruta indicada desde la ruta raiz del proyecto.
6          *
7          * @param  string  $path
8          * @return string
9          */
10        function base_path(string $path = ''): string {
11            return BASE_DIR . "/" . ltrim($path, "/");
12        }
13    }
```

Notas API

GET -> SELECT

POST -> INSERT

PUT -> UPDATE

DELETE -> DELETE

PATCH -> COMO POST PERO SOLO PARA PEQUEÑAS MODIFICACIONES

Nuestra capa de presentación contendrá los controladores, que serán los encargados de tratar las peticiones y las respuestas. En la capa de negocio estarán los servicios que se encargarán de conectar con la capa de persistencia para tratar los datos. Por último, la capa de persistencia será la encargada de conectar con nuestros datos.

Servicio -> método para traer actores (El método llama al controlador actores)

La clase DB ejecuta las sentencias

El QueryBuilder utiliza la clase DB y crea las sentencias SQL

La clase DAO realiza las sentencias SQL