UT6 - PROGRAMACIÓN ORIENTADA A OBJETOS PARA EL DESARROLLO WEB

Desarrollo Web en Entorno Servidor

2° DAW

Índice

- Creación de clases en PHP
- Utilización de objetos
- Mecanismos de mantenimiento del estado
- Herencia
- Interfaces

Características de orientación a objetos en PHP

- Ya hemos visto anteriormente que con PHP se puede utilizar dos estilos de programación: estructurado y orientado a objetos

```php
// utilizando programación estructurada
$dwes = mysqli_connect('localhost', 'root', '', 'dwes_04_tienda');

// utilizando POO
$dwes = new mysqli();
$dwes->connect('localhost', 'root', '', 'dwes_04_tienda');
```

- Sin embargo, el lenguaje PHP original no se diseñó con características de orientación a objetos.
- Sólo a partir de la versión 3, se empezaron a introducir algunos rasgos de POO en el lenguaje.
- Esto se potenció en la versión 4, aunque todavía de forma muy rudimentaria

Características de orientación a objetos en PHP

- En la versión 5 se ha reescrito y potenciado el soporte de orientación a objetos del lenguaje, ampliando sus características y mejorando su rendimiento y su funcionamiento general.
- Las características de POO que soportan PHP5 y PHP7 incluyen:
  - Métodos estáticos.
  - Métodos constructores y destructores.
  - Herencia.
  - Interfaces.
  - Clases abstractas.

- No se incluye:
  - Herencia múltiple (Java NO tiene)
  - Sobrecarga de métodos (Java SÍ tiene)
  - Sobrecarga de operadores (Java NO tiene)

Creación de clases en PHP

- La declaración de una clase en PHP se hace utilizando la palabra **class.** A continuación y entre llaves, deben figurar los miembros de la clase. Conviene hacerlo de forma ordenada, primero las propiedades o atributos, y después los métodos, cada uno con su código respectivo.

```
class Producto {
   private $codigo;
   public $nombre;
   public $PVP;
   public function muestra() {
            return "<p>" . $this->codigo . "</p>";
   }
}
```

- Recomendaciones:
  - Cada clase en un fichero distinto
  - Los nombres de las clases deben comenzar por mayúsculas para distinguirlos de los objetos y otras variables

Creación de clases en PHP

- **Ejercicio**
  - Crear vuestra clase Producto con los atributos que consideréis necesarios
- Una vez definida la clase, podemos usar la palabra **new** para instanciar objetos de la siguiente forma:

```
$p = new Producto();
```

- Para que la línea anterior se ejecute sin error, previamente debemos haber declarado la clase. Para ello, en ese mismo fichero tendrás que incluir la clase poniendo algo como:

```
require_once('producto.php');
```

- Para acceder desde un objeto a sus atributos o a los métodos de la clase, debes utilizar el **operador flecha** (sólo se pone el símbolo $ delante del nombre del objeto):

```
$p->nombre = 'Samsung Galaxy S 20';
echo $p->muestra();
```

Creación de clases en PHP

- Cuando se declara un atributo, se debe indicar su nivel de acceso. Los principales niveles son:
  - public
  - protected
  - private
- En PHP4 no se podía definir nivel de acceso para los atributos de una clase, por lo que todos se precedían de la palabra **var**. (Por ejemplo: var $nombre;)
- Hoy en día, aunque aún es aceptado por PHP7, no se recomienda su uso. Si vemos algún código que lo utilice, tendrá el mismo efecto que **public**.
- ¿Por qué debemos poner los atributos como privados?
  - Mantener un control entre los valores que puede tener

Creación de clases en PHP


```
private $codigo;
public function setCodigo($nuevo_codigo)
{
      if (noExisteCodigo($nuevo_codigo))
      {
                $this->codigo = $nuevo_codigo;
                return true;
      }
      return false;
}
public function getCodigo() { return $this->codigo; }
```

- Aunque no es obligatorio, el nombre del método que nos permite obtener el valor de un atributo suele empezar por **get**, y el que nos permite modificarlo por **set**.
- **Ejercicio:**
  - Crear los métodos get y set para vuestra clase Producto

Creación de clases en PHP

- En PHP5 se introdujeron los llamados **métodos mágicos,** entre ellos **__set y __get.** Si se declaran estos dos métodos en una clase, PHP los invoca automáticamente cuando desde un objeto se intenta usar un atributo no existente o no accesible.
- Por ejemplo, el código siguiente simula que la clase Producto tiene cualquier atributo que queramos usar.

```
class Producto
{
      private $atributos = array();
      public function __get($atributo) {
                return $this->atributos[$atributo];
      }
      public function __set($atributo, $valor) {
                $this->atributos[$atributo] = $valor;
      }
}
```

Creación de clases en PHP

- Otra manera de usar los métodos mágicos podría ser así:

```
class Persona {
      private $id;
      private $nombre;
      private $email;

      public function __set($var, $valor) {
                  if (property_exists(__CLASS__, $var)) {
                              $this->$var = $valor;
                  } else {
                              echo "No existe el atributo $var.";
                  }
            }
      public function __get($var) {
                  if (property_exists(__CLASS__, $var)) {
                              return $this->$var;
                  }
                  return NULL;
       }
}
$p = new Persona();
$p->nombre = "Pepe";
```

Creación de clases en PHP

- **Ejercicio**
  - Comenta los métodos get y set que has hecho y añade los mágicos
- Cuando desde un objeto se invoca un **método de la clase**, a éste se le pasa siempre una referencia al objeto que hizo la llamada. Esta referencia se almacena en la variable **$this**.
- Se utiliza, por ejemplo, en el código anterior para tener acceso a los **atributos privados del objeto** (que sólo son accesibles desde los métodos de la clase).

```
print "<p>" . $this->codigo . "</p>";
```

Creación de clases en PHP

- Además de métodos y propiedades, en una clase también se pueden definir **constantes**, utilizando la palabra **const**.
- No hay que confundir los atributos con las constantes. Son conceptos distintos: las constantes no pueden cambiar su valor (obviamente, de ahí su nombre), no usan el carácter **$** y está asociado a la clase, es decir, no existe una copia del mismo en cada objeto.
- Por tanto, para acceder a las constantes de una clase, se debe utilizar el nombre de la clase y el operador **::**, llamado **operador de resolución de ámbito** (que se utiliza para acceder a los elementos de una clase).

```
class BaseDatos {
      const USUARIO = 'dwes';
      …
}
echo BaseDatos::USUARIO;
```

- No es necesario que exista ningún objeto de una clase para poder acceder al valor de las constantes que defina.
- Además, sus nombres suelen escribirse en mayúsculas
- **Ejercicio**:
  - Crear una clase BaseDatos que tendrá como constantes el dominio donde está alojada, el usuario y la contraseña y la base de datos a utilizar

Creación de clases en PHP

- En PHP5, una clase puede tener atributos o métodos **estáticos**, también llamados a veces atributos o métodos de clase. Se definen utilizando la palabra clave **static**.

```
class Producto {
      private static $num_productos = 0;
      public static function nuevoProducto() {
                  self::$num_productos++;
      }
...
}
```

- Los atributos y métodos estáticos **no** pueden ser llamados desde un objeto de la clase utilizando el operador ->.
- Si el método o atributo es **público**, deberá accederse utilizando el nombre de la clase y el operador de resolución de ámbito.
  - Producto::nuevoProducto();
- Si es **privado**, como el atributo $num_productos en el ejemplo anterior, sólo se podrá acceder a él desde los métodos de la propia clase, utilizando la palabra **self**. De la misma forma que $this hace referencia al objeto actual, self hace referencia a la clase actual.
  - self::$num_productos ++;

Creación de clases en PHP

- En PHP7 puedes definir en las clases métodos constructores, que se ejecutan cuando se crea el objeto. El constructor de una clase debe llamarse **__construct**. Se pueden utilizar, por ejemplo, para asignar valores a atributos.

```
class Producto {
    private static $num_productos = 0;
    private $codigo;
    public function __construct() {
            self::$num_productos++;
    }
...
}
```

Creación de clases en PHP

- El constructor de una clase puede llamar a otros métodos o tener parámetros, en cuyo caso deberán pasarse cuando se crea el objeto.
- Sin embargo, **sólo puede haber un método constructor en cada clase.**

```
class Producto {
    private static $num_productos = 0;
    private $codigo;
    public function __construct($codigo) {
            $this->$codigo = $codigo;
            self::$num_productos++;
    }
...
}
```

- **Ejercicio:**
  - Crear un constructor para vuestra clase Producto

Creación de clases en PHP

- También es posible definir un método destructor, que debe llamarse **__destruct** y permite definir acciones que se ejecutarán cuando se elimine el objeto.

```
class Producto {
      private static $num_productos = 0;
      private $codigo;
      public function __construct($codigo) {
                $this->$codigo = $codigo;
                self::$num_productos++;
      }
      public function __destruct() {
                self::$num_productos--;
      }
      …
}
$p = new Producto('GALAXYS');
```

Utilización de objetos

- Una vez creado un objeto, puedes utilizar el operador **instanceof** para comprobar si es o no una instancia de una clase determinada.

```
if ($p instanceof Producto) {
…
}
```

- Además, en PHP7 se incluyen una serie de funciones útiles para el desarrollo de aplicaciones utilizando POO.

Función	Ejemplo	Significado
get_class	echo "La clase es: " . get_class($p);	Devuelve el nombre de la clase del objeto

class_exists	if (class_exists('Producto')) {
	$p = new Producto();
	…
	}
		Devuelve true si la clase está definida o false en caso contrario

get_declared_classes	print_r(get_declared_classes());	Devuelve un array con los nombres de las clases definidas.

Utilización de objetos

Función	Ejemplo	Significado
class_alias	class_alias('Producto', 'Articulo'); $p = new Articulo();	Crea un alias para una clase
get_class_methods	print_r(get_class_methods('Producto'));	Devuelve un array con los nombres de los métodos de una clase que son accesibles desde dónde se hace la llamada

method_exists	if (method_exists('Producto', 'vende')){
	…
	}
		Devuelve true si existe el método en el objeto o la clase que se indica, o false en caso contrario, independientemente de si es accesible o no

get_class_vars	print_r(get_class_vars('Producto'));	Devuelve un array con los nombres de los atributos de una clase que son accesibles desde dónde se hace la llamada

Utilización de objetos

Función	Ejemplo	Significado
get_object_vars	print_r(get_object_vars($p));	Devuelve un array con los nombres de los atributos de un objeto que son accesibles desde dónde se hace la llamada

property_exists	if (property_exists('Producto', 'codigo') {
	…
	}
		Devuelve true si existe el atributo en el objeto o la clase que se indica, o false en caso contrario, independientemente de si es accesible o no

- **Ejercicio**
  - Probar los métodos anteriores en el ejercicio 2 de la Hoja_06-01

Utilización de objetos

- Desde PHP5, puedes indicar en las funciones y métodos de qué clase deben ser los objetos que se pasen como parámetros. Para ello, debes especificar el tipo antes del parámetro.

```
public function vendeProducto(Producto $p) {
...
}
```

- Si cuando se realiza la llamada, el parámetro no es del tipo adecuado, se produce un error que se podría capturar.
- **Pregunta**
  - ¿Qué sucede al ejecutar el siguiente código?

```
$p = new Persona();
$p->nombre = 'Pepe';
$a = $p;
```

Utilización de objetos

- En PHP4, la última línea del código anterior crea un nuevo objeto con los mismos valores del original, de la misma forma que se copia cualquier otro tipo de variable. Si después de hacer la copia se modifica, por ejemplo, el atributo 'nombre' de uno de los objetos, el otro objeto no se vería modificado.
- Sin embargo, en PHP5 y PHP7 este comportamiento varía. El código anterior simplemente crearía un **nuevo identificador del mismo objeto**. Esto es, en cuanto se utilice uno cualquiera de los identificadores para cambiar el valor de algún atributo, este cambio se vería también reflejado al acceder utilizando el otro identificador. Aunque haya dos o más identificadores del mismo objeto, en realidad todos se refieren a la única copia que se almacena del mismo. Tiene un comportamiento similar al que ocurre en Java.
- Por tanto, a partir de PHP5 no puedes copiar un objeto utilizando el operador =. Si necesitas copiar un objeto, debes utilizar clone. Al utilizar clone sobre un objeto existente, se crea una copia de todos los atributos del mismo en un nuevo objeto.

```
$p = new Producto();
$p->nombre = 'Xiaomi Mi 10';
$a = clone $p;
```

- **Ejercicio**
  - Crear una clase Persona y probarlo

Utilización de objetos

- Además, existe una forma sencilla de personalizar la copia para cada clase particular. Por ejemplo, puede suceder que quieras copiar todos los atributos menos alguno. En nuestro ejemplo, al menos el código de cada producto debe ser distinto y, por tanto, quizás no tenga sentido copiarlo al crear un nuevo objeto. Si éste fuera el caso, puedes crear un método de nombre **__clone** en la clase. Este método se llamará automáticamente después de copiar todos los atributos en el nuevo objeto.

```
class Producto {
        ...
        public function __clone() {
                $this->codigo = nuevo_codigo();
        }
        ...
}
```

Utilización de objetos

- A veces tienes dos objetos y quieres saber su relación exacta. Para eso, en PHP5 puedes utilizar los operadores == y ===.
- Si utilizas el operador de comparación ==, comparas los valores de los atributos de los objetos. Por tanto dos objetos serán iguales si son instancias de la misma clase y, además, sus atributos tienen los mismos valores.

```
$p = new Producto();
$p->nombre = 'Xiaomi Mi 10';
$a = clone $p;
// El resultado de comparar $a == $p da verdadero pues $a y $p son dos copias idénticas
```

- Sin embargo, si utilizas el operador ===, el resultado de la comparación será **true** sólo cuando las dos variables sean referencias al mismo objeto.

```
$p = new Producto();
$p->nombre = 'Xiaomi Mi 10';
$a = clone $p ;
// El resultado de comparar $a === $p da falso pues $a y $p no hacen referencia al mismo objeto

$a = & $p;
// Ahora el resultado de comparar $a === $p da verdadero  pues $a y $p son referencias al mismo objeto.
```

Mecanismos de mantenimiento del estado

- ¿Cómo almacenábamos hasta ahora el estado de las variables para recuperarlas cuando fuese necesario?
  - Con el array superglobal $_SESSION
- El procedimiento para almacenar objetos es similar, pero hay una diferencia importante. Todas las variables almacenan su información en memoria de una forma u otra según su tipo. Los objetos, sin embargo, no tienen un único tipo. Cada objeto tendrá unos atributos u otros en función de su clase.
- Por tanto, para almacenar los objetos en la sesión del usuario, hace falta convertirlos a un formato estándar. Este proceso se llama **serialización**.

Mecanismos de mantenimiento del estado

- En PHP, para serializar un objeto se utiliza la función **serialize**. El resultado obtenido es un string que contiene un flujo de bytes, en el que se encuentran definidos todos los valores del objeto.

```
$p = new Producto();
$a = serialize($p);
```

- Esta cadena se puede almacenar en cualquier parte, como puede ser la sesión del usuario, o una base de datos. A partir de ella, es posible reconstruir el objeto original utilizando la función **unserialize.**

```
$p = unserialize($a);
```

- Las funciones serialize y unserialize se utilizan mucho con objetos, pero sirven para convertir en una cadena cualquier tipo de dato. Cuando se aplican a un objeto, convierten y recuperan toda la información del mismo, incluyendo sus atributos privados. La única información que no se puede mantener utilizando estas funciones es la que contienen los atributos estáticos de las clases.
- Si simplemente queremos almacenar un objeto en la sesión del usuario, deberíamos hacer por tanto:

```
session_start();
$_SESSION['producto'] = serialize($p);
```

Mecanismos de mantenimiento del estado

- Pero en PHP esto aún es más fácil. Los objetos que se añadan a la sesión del usuario son serializados automáticamente. Por tanto, no es necesario usar **serialize** ni **unserialize.**

```
session_start();
$_SESSION['producto'] = $p;
```

- Para poder deserializar un objeto, debe estar definida su clase. Al igual que antes, si lo recuperamos de la información almacenada en la sesión del usuario, no será necesario utilizar la función **unserialize.**

```
session_start();
$p = $_SESSION['producto'];
```

- Como ya viste en el tema anterior, el mantenimiento de los datos en la sesión del usuario no es perfecta; tiene sus limitaciones. Si fuera necesario, es posible almacenar esta información en una base de datos. Para ello tendrás que usar las funciones serialize y unserialize, pues en este caso PHP ya no realiza la serialización automática.

Herencia

- Recordatorio del curso pasado
  - La herencia es un mecanismo de la POO que nos permite definir nuevas clases en base a otra ya existente. Las nuevas clases que heredan también se conocen con el nombre de **subclases**. La clase de la que heredan se llama **clase base** o **superclase**
- Por ejemplo si tenemos la clase Persona, podemos crear las subclases Alumno y Profesor.

```
class Persona {
        protected$nombre;
        protected$apellidos;
        public function muestra() {
                print "<p>" . $this->nombre. $this->apellidos."</p>";
        }
}
```

- Esto puede ser útil si todas las personas sólo tuviesen nombre y apellidos, pero los alumnos tendrán un conjunto de notas y los profesores tendrán, por ejemplo, un número de horas de docencia.

```
class Alumno extends Persona {
        private $notas;
}
```

- **Ejercicio**
  - Codificar estas dos clases y crear un objeto de tipo Alumno. Comprobar mediante el operador instanceof si el objeto es de tipo Persona o de tipo Alumno

Herencia

Función	Ejemplo	Significado
get_parent_class	echo "La clase padre es: " . get_parent_class($t);	Devuelve el nombre de la clase padre del objeto o la clase que se indica
is_subclass_of	if (is_subclass_of($t, 'Producto')) { …	Devuelve true si el objeto o la clase del primer parámetro, tiene como clase base a la que se indica en el segundo parámetro, o false en caso contrario

- La nueva clase hereda todos los atributos y métodos públicos de la clase base, pero no los privados.
- Si quieres crear en la clase base un método no visible al exterior (como los privados) que se herede a las subclases, debes utilizar la palabra **protected** en lugar de private. Además, puedes redefinir el comportamiento de los métodos existentes en la clase base, simplemente creando en la subclase un nuevo método con el mismo nombre.

Herencia


```
class Profesor extends Persona {
        private $horas;
        public function muestra() {
                print "<p>" . $this->nombre . ": ". $this->horas .</p>";
        }
}
```

- Existe una forma de evitar que las clases heredadas puedan redefinir el comportamiento de los métodos existentes en la superclase: utilizar la palabra **final**. Si en nuestro ejemplo hubiéramos hecho:

```
class Persona {
        protected$nombre;
        protected$apellidos;
        public final function muestra() {
                print "<p>" . $this->nombre. $this->apellidos."</p>";
        }
}
```

- En este caso el método **muestra** no podría redefinirse en la clase Profesor.
- Incluso se puede declarar una clase utilizando **final**. En este caso no se podrían crear clases heredadas utilizándola como base.

```
final class Persona {
…
}
```

Herencia

- Opuestamente al modificador **final**, existe también **abstract**. Se utiliza de la misma forma, tanto con métodos como con clases completas, pero en lugar de prohibir la herencia, obliga a que se herede. Es decir, una clase con el modificador **abstract** no puede tener objetos que la instancien, pero sí podrá utilizarse de clase base y sus subclases sí podrán utilizarse para instanciar objetos.

```
abstract class Persona {
        …
}
```

- Y un método en el que se indique **abstract**, debe ser redefinido obligatoriamente por las subclases, y no podrá contener código.

```
class Persona {
        …
        abstract public function muestra();
}
```

- **Ejercicio**
  - Crear un constructor para la clase Persona
  - ¿Qué pasará ahora con la clase Alumno, qué hereda de Persona? Cuando crees un nuevo objeto de esa clase, ¿se llamará al constructor de Persona? ¿Puedes crear un nuevo constructor específico para Alumno que redefina el comportamiento de la clase base?

Herencia

- En PHP7, si la clase heredada no tiene constructor propio, se llamará automáticamente al constructor de la clase base (si existe). Sin embargo, si la clase heredada define su propio constructor, deberás ser tú el que realice la llamada al constructor de la clase base si lo consideras necesario, utilizando para ello la palabra **parent** y el operador de resolución de ámbito.

```
class Alumno extends Persona {
        private $notas;
        public function __construct($nombre,$apellidos,$notas) {
                parent::__construct($nombre,$apellidos);
                $this->notas = $notas;
        }
}
```

- La utilización de la palabra **parent** es similar a **self**. Al utilizar parent haces referencia a la clase base de la actual.

Interfaces

- Un interface es similar a una clase vacía que solamente contiene declaraciones de métodos. Se definen utilizando la palabra **interface.**
- Por ejemplo, antes viste que podías crear nuevas clases heredadas de **Persona,** como **Profesor** o **Alumno**. También viste que en las subclases podías redefinir el comportamiento del método **muestra** para que generara una salida en HTML diferente para cada tipo de persona.
- Si quieres asegurarte de que todos los tipos de persona tengan un método **muestra,** puedes crear un interface como el siguiente.

```
interface iMuestra {
        public function muestra();
}
```

- Y cuando crees las subclases deberás indicar con la palabra **implements** que tienen que implementar los métodos declarados en este interface.

```
class Profesor extends Persona implements iMuestra {
        …
        public function muestra() {
                print "<p>" . $this->nombre . ": ". $this->horas .</p>";
        }
        …
}
```

Interfaces

- Todos los métodos que se declaren en un interface deben ser públicos. Además de métodos, los interfaces podrán contener constantes pero no atributos.
- Un interface es como un contrato que la clase debe cumplir. Al implementar todos los métodos declarados en el interface se asegura la interoperabilidad entre clases. Si sabes que una clase implementa un interface determinado, sabes qué nombre tienen sus métodos, qué parámetros les debes pasar y, probablemente, podrás averiguar fácilmente con qué objetivo han sido escritos.
- Por ejemplo, en la librería de PHP está definido el interface **Countable.**

```
Countable {
        abstract public int count ( void );
}
```

- En PHP7 es posible crear clases que implementen varios interfaces, simplemente separando la lista de interfaces por comas después de la palabra **implements.**

```
class Profesor extends Persona implements iMuestra, Countable {
        …
}
```

Interfaces

- La única restricción es que los nombres de los métodos que se deban implementar en los distintos interfaces no coincidan. Es decir, en nuestro ejemplo, el interface **iMuestra** no podría contener un método **count**, pues éste ya está declarado en **Countable**.
- En PHP7 también se pueden crear nuevos interfaces heredando de otros ya existentes. Se hace de la misma forma que con las clases, utilizando la palabra **extends**.
- Una de las dudas más comunes en POO, es qué solución adoptar en algunas situaciones: interfaces o clases abstractas. Ambas permiten definir reglas para las clases que los implementen o hereden respectivamente. Y ninguna permite instanciar objetos. Las diferencias principales entre ambas opciones son:
  - En las clases abstractas, los métodos pueden contener código. Si van a existir varias subclases con un comportamiento común, se podría programar en los métodos de la clase abstracta. Si se opta por un interface, habría que repetir el código en todas las clases que lo implemente.
  - Las clases abstractas pueden contener atributos, y los interfaces no.
  - No se puede crear una clase que herede de dos clases abstractas, pero sí se puede crear una clase que implemente varios interfaces.

Interfaces

- Para finalizar con los interfaces, a la lista de funciones de PHP relacionadas con la POO puedes añadir las siguientes

Función	Ejemplo	Significado
get_declared_interfaces	print_r (get_declared_interfaces());	Devuelve un array con los nombres de los interfaces declarados.
interface_exists	if (interface_exists('iMuestra')) { … }	Devuelve true si existe el interface que se indica, o false en caso contrario.

Ejercicio de entrega (VI)

- Crearemos una copia de nuestra tienda virtual.
- Dentro de la nueva versión, crearemos una carpeta *include* en la que se crearán las siguientes clases:
  - **Producto:** las instancias de esta clase representan los productos que se venden en la tienda.
    - Tendremos como atributos los campos que habéis creado en vuestra tabla de la base de datos
    - Constructor
    - Métodos get
  - **BaseDatos**: va a ser la clase encargada de interactuar con la base de datos. No necesita almacenar ninguna información; simplemente deberá contener métodos para realizar acciones sobre la base de datos. Por tanto, vamos a definir en la misma únicamente métodos estáticos. No hará falta instanciar ningún objeto de esta clase.

    Vuestros métodos podrán ser similares a los siguientes:
    - getProductos(): devuelve un array con todos los productos de la base de datos.
    - obtieneProducto($codigo): devuelve el producto que coincide con el código
    - verificarUsuario($usuario, $password): V o F si son correctas las credenciales

Ejercicio de entrega (VI)

- **CestaCompra**: con esta clase vas a gestionar los productos que escoge el cliente de la tienda para comprar.
  - Tendremos un array como atributo
  - nuevoArticulo($codigo): introduce en la cesta el artículo indicado por su código.
  - getProductos: devuelve un array con todos los productos de la cesta.
  - getCoste: devuelve el coste de los productos que figuran en la cesta.
  - estaVacia: devuelve true o false, según la cesta esté o no vacía.

  También necesitamos dos funciones para guardar la cesta en la sesión del usuario, y para recuperarla. Y programaremos otra mas para mostrar el contenido de la cesta en formato HTML.
  - guardaCesta: guarda la cesta en la sesión del usuario.
  - carga_cesta: recupera el contenido de la cesta de la sesión del usuario.
  - muestra: genera una salida en formato HTML con el contenido de la cesta.