La Sénia
Institut d'Educació Secundària
Paiporta

UD3. Estructuras en PHP

Desarrollo Web en Entorno Servidor

Profesora: Silvia Vilar Pérez

curso 2024-2025

Contenidos

- Tomas de decisión. Estructuras de control
- Estructuras iterativas.
- Estructuras de control de flujo.
- Arrays
- Características de los Arrays
- Creación y eliminación de Arrays.
- Operaciones sobre Arrays
- Cadenas
- Funciones
- Manejo de Fecha y Hora
- Pruebas y Depuración

Tomas de decisión. Estructuras de control

Podemos usar las estructuras condicionales:

SENTENCIA SWITCH

```
<?php
switch ($i) {
    case 0:
        echo "i es igual a 0";
        break;
    case 1:
        echo "i es igual a 1";
        break;
    case 2:
        echo "i es igual a 2";
        break;
     default:
     echo "No se imprime si hay break"
}
?>
```

SENTENCIA IF

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

Estructuras repetitivas

Podemos usar las estructuras repetitivas:

SENTENCIA WHILE

```
<?php
$i = 1;
while ($i <= 10) {
    echo $i;
    $i++;
}
?>

<?php
$i = 1;
while ($i <= 10):
    echo $i;
    $i++;
endwhile;
?>
```

SENTENCIA DO WHILE

```
<?php
do {
    if ($i < 5) {
        echo "i no es lo suficientemente grande";
        break;
    }
    $i =$i * $i;
    if ($i < $minimum_limit) {
        break;
    }
   echo "i está bien";
    /* procesar i */
} while (0);
?>
```

Estructuras iterativas

Podemos usar las estructuras itrativas:

SENTENCIA FOR

```
<?php
for ($i=0;$i<10; $i++){
    $suma=$suma+$i;

    echo “suma: $i \n";
    }
?>
```

SENTENCIA FOREACH

```
<?php
foreach ($elementos as $e){
        echo “elemento: $e \n";
    }

foreach ($elementos as
$key=>$value){
        echo “$key => $value \n";
    }
?>
```

Estructuras de control de flujo

BREAK es la sentencia para salir directamente:

SENTENCIA BREAK

```
<?php
$i = 0;
while (++$i) {
    switch ($i) {
    case 5:
        echo "En 5\n";
        break 1;  /* Sólo sale del switch. */
    case 10:
        echo "En 10; saliendo\n";
        break 2;  /* Sale del switch y del while. */
    default:
        break;
    }
}
?>
```

NOTA: El número entero opcional indica el número de niveles de la estructura anidada que queremos salir

Estructuras de control de flujo

CONTINUE permite abandonar la iteración y seguir:

SENTENCIA CONTINUE

```
<?php
$i = 0;
while ($i++ < 5) {
    echo "Exterior\n";
    while (1) {
        echo "Medio\n";
        while (1) {
            echo "Interior\n";
            continue 3;
        }
        echo "Esto nunca se imprimirá.\n";
    }
    echo "Ni esto tampoco.\n";
}
?>
```

NOTA: El número entero opcional indica el número de niveles de la estructura anidada que queremos saltar y volver a evaluar la condición tras ese salto

Arrays (Matrices)

- Un array almacena pares clave-valor
- Puede tener un número variable de elementos
- Cada elemento puede tener un valor de tipo simple (integer, string, etc.) o compuesto (otro array)
- El array que contiene otro/s arrays (matrices) se denomina multidimensional
- PHP admite:
  - **Arrays escalares** donde los ***índices*** son ***números***
  - **Arrays asociativos** donde los ***índices*** son ***cadenas*** de caracteres

Arrays

Sintaxis:

array ([clave =>] valor$_1$, [clave =>] valor$_2$, ...)

- La clave es una cadena o un entero no negativo.
- El valor puede ser de cualquier tipo válido en PHP, incluyendo otro array
- Ejemplos:

```
$color = array ('rojo'=>101, 'verde'=>51, 'azul'=>25);
$medidas = array (10, 25, 15);
```

- Acceso a los arrays del ejemplo:

```
$color['rojo'] // No olvidar las comillas
$medidas[0]
```

- El primer elemento es el 0

Características de los Arrays

Características

- Los arrays no se declaran, ni siquiera para indicar su tamaño (como el resto de las variables).
- Pueden ser dispersos (se implementan como tablas hash).
- Los índices de sus elementos no tienen porque ser consecutivos.

```
$vec[1] = '1º elemento';
$vec[8] = '2º elemento';
```

- En realidad contienen un mapeo entre claves y valores (arrays asociativos)

```
array([index]=>[valor], [index2]=>[valor], ...);
```

Características de los Arrays

Características

- Los índices no tienen porque ser números (arrays asociativos cuyo índice es una cadena de caracteres)

  $vec['tercero'] = '3º elemento‘;

- Los arrays no son homogéneos. Sus elementos pueden ser de cualquier tipo (incluso tipo Array) y ser de tipos diferentes en el mismo vector.

  $vec[5] = '4º elemento';

  $vec[1000] = 5.0;

Creación de Arrays

Formas de crear arrays

- Asignación directa: Se añaden los elementos uno a uno indicando el índice entre [ ]. Si no existía, se crea

  $vec[5] = “1º elemento”; $vec[1] = “2º elemento”;

  $vec[6] = “3º elemento”; $vec[ ] = “3º elemento”; //sin indicar clave toma el valor siguiente al máximo de los índices enteros

- Usando el constructor ***array()***.
  - Se añaden entre paréntesis los elementos. El índice comienza en 0 Ej: $vec = array (2, 9.7, “Silvia”);

  // $vec[0] = 2, $vec[1] = 9.7 y $vec[2] = “Silvia”
  - Se puede fijar el índice con el operador =>

  $vec = array (4 => 2, 9.7, "nombre"=> "Silvia");

  // $vec[4] = 2, $vec[5] = 9.7 y $vec[“nombre”] = “Silvia”

Eliminación de Arrays

- Para eliminar los elementos del array se usa ***unset()***

  unset($vec[5]) unset($vec[‘nombre’]) unset($vec)
  // La última elimina el array completo

- Imprimimos el array y sus valores con ***var_dump($vec)*** o bien con ***print_r($vec)*** //no echo.
- Podemos reindexar el array para que su índice comience en 0 con ***array_values($vec)***

```
$vec[3] = 6;
$vec[] = 7; //El índice valdría 4
$array = array_values($array);
print_r($vec);
```

Resultado:

```
Vec
(
    [0] => 6
    [1] => 7
)
```

Arrays Asociativos

La clave o índice en un String. Pueden definirse:

- Mediante la función array()

```
$precios = array("Azúcar" => 1, "Aceite" => 4, "Arroz" => 0.5);
$capitales = array("Francia"=>"París", "Italia"=>"Roma");
```

- Por referencia

```
$precios["Azúcar"] = 1;
$precios["Aceite"] = 4;
$precios["Arroz"] = 0.5
$capitales["Francia"]="París";
$capitales ["Italia"]="Roma";
```

Arrays Multidimensionales

- Son arrays en los que al menos uno de sus valores es otro array
- Pueden ser escalares o asociativos

```
$pais=array(
    "espana"=>array(
            "nombre"=>"España",
            "lengua"=>"Castellano",
            "moneda"=>"Euro"),
    "uk" =>array(
            "nombre"=>"UK",
            "lengua"=>"Inglés",
            "moneda"=>"Libra"));
```

```
Resultado de var_dump($pais):
array(2) {
  ["espana"]=>
  array(3) {
    ["nombre"]=>
    string(7) "España"
    ["lengua"]=>
    string(10) "Castellano"
    ["moneda"]=>
    string(4) "Euro"
  }
  ["uk"]=>
  array(3) {
    ["nombre"]=>
    string(2) "UK"
    ["lengua"]=>
    string(7) "Inglés"
    ["moneda"]=>
    string(5) "Libra"
  }
}
```

Recorrido en Arrays

Podemos recorrer los elementos del array con bucles

```
$ciudades = array("París", "Madrid", "Londres");
```

- Mostrar el contenido del array (for)

```
for ($i=0;$i<count($ciudades); $i++){
    echo $ciudades[$i]; echo "\n";
}
```

- Mostrar el contenido del array (foreach)

```
foreach ($ciudades as $ciudad){
    echo $ciudad; echo "\n";
}
foreach ($ciudades as $key=>$ciudad){ //si el vector es asociativo
    echo “$key => $ciudad”; echo "\n";
}
```

También disponemos de multitud de funciones de arrays

https://www.php.net/manual/es/ref.array.php

Cadenas

- Echo y print no permiten formatear la salida
  print $variable == echo $variable.
  echo "hola1","hola2"; → admite parámetros
  print ("hola1","hola2"); → error! →hacer print "hola1"."hola2"
- ***sprintf*** (igual que printf):Devuelve cadena formateada con el formato indicado.

% - un carácter de porcentaje literal. No se requiere argumento.
b - argumento tipo integer y número binario.
c - argumento tipo integer carácter con valor ASCII.
d - argumento tipo integer y número decimal (con signo).
f/F - argumento tipo float y número de punto flotante (local/no local)
s - argumento tipo string.

Ejemplo: $dia= 5; $mes=3; $anno=12;
printf("%02d/%02d/%04d", $dia, $mes, $anno); (cantidad caracter/cifra)
Escribe: 05/03/0012

Cadenas

- Ejemplo

```
<?php
$s = 'mono';
$t = 'muchos monos';
printf("[%s]\n",      $s); // salida estándar de string
printf("[%10s]\n",    $s); // justificación a la derecha con espacios
printf("[%-10s]\n",   $s); // justificación a la izquierda con espacios
printf("[%010s]\n",   $s); // rellenado con ceros también funciona con strings
printf("[%'#10s]\n",  $s); // utiliza el carácter de relleno personalizado '#'
printf("[%10.10s]\n", $t); // justificación a la izquierda pero con un corte a los 10 caracteres
?>
```

- Resultado

```
[mono]
[      mono]
[mono      ]
[000000mono]
[#######mono]
[muchos mon]
```


Funciones con cadenas: https://www.php.net/manual/es/ref.strings.php

Funciones

- Sintaxis:

```
function nombreFunción (param1,param2){
    Instrucción1;
    Instrucción2;
    return valor_de_retorno;
}
```

- Ejemplo:

```
function suma ($x, $y){
$s = $x + $y;
return $s;
}
//La invocamos
$a=1;
$b=2;
$c=suma ($a, $b);
print $c;
```

Funciones

- Todas las funciones y clases de PHP tienen ***ámbito global***. Se pueden llamar desde fuera de una función incluso si fueron definidas dentro, y viceversa.

```
<?php
function externa()
{
  function interna()
  {
    echo "No existo hasta que se llame a externa().\n";
  }
}
/* No podemos llamar aún a interna() ya que no existe. */
externa();
/* Ahora podemos llamar a interna(), la ejecución de externa() la ha
hecho accesible. */
interna();
?>
```

Funciones

- Los argumentos se pueden pasar por valor ($i) o referencia (&$i). En caso de tener argumentos con valor predeterminado, deben ir a la derecha de los no predeterminados

```
<?php
function hacer_yogur($sabor, $tipo = "acidófilo")
{
   return "Hacer un tazón de yogur $tipo de $sabor.\n";
}
echo hacer_yogur("frambuesa");
echo hacer_yogur(“frambuesa”,”dulce”);
?>
```

Resultado:
Hacer un tazón de yogur acidófilo de frambuesa.
Hacer un tazón de yogur dulce de frambuesa.

Funciones Variables

- Funciones variables: si llamamos a una variable con paréntesis buscará una función con ese nombre y la ejecutará

```
<?php
function prueba($arg = ‘ ‘){
echo "Estamos en la función prueba() y el argumento es '$arg'.<br>\n";
}
$func = "prueba"; //inicializamos la variable
$func('hola'); // Esto llama a prueba('hola')
?>
```

Resultado:

Estamos en la función prueba() y el argumento es ‘hola’

Ver https://www.php.net/manual/es/functions.variable-functions.php

Funciones Anónimas

- Las funciones anónimas, también conocidas como cierres (closures), permiten la creación de funciones que no tienen un nombre especificado. Se implementan usando la clase Closure. PHP 7.4 introduce funciones de ***flecha*** con sintaxis más concisa: **fn(list_args) => expr;**

```
<?php
$saludo = function($nombre)
{
    printf("Hola %s\n", $nombre);
};     //la variable $saludo contiene la declaración de la función

$saludo1 = fn($nombre) =>printf("Hola %s\n", $nombre); //Función flecha (se obtiene el mismo resultado)

$saludo('Mundo');
$saludo1('PHP');
?>
```

Resultado:
Hola Mundo
Hola PHP

Ver https://www.php.net/manual/es/functions.anonymous.php
Funciones flecha: https://www.php.net/manual/es/functions.arrow.php

Manejo de Fecha y Hora

Clase DateTime

- Clase para trabajar con fechas y horas en PHP
- Debemos definir, en primer lugar, nuestra Zona Horaria:
  - En la sección Date del archivo php.ini

    ;[Date]
    ;Defines the default timezone used by the date functions
    date.timezone = Europa/Madrid
  - Durante la ejecución con la función ***date_default_timezone_set()***. Esta función genera un error si contradice la configuración del php.ini.
- Listado de zonas horarias soportadas: https://www.php.net/manual/es/timezones.php

Ver https://www.php.net/manual/es/book.datetime.php

Pruebas

Podemos aplicar distintas pruebas al SW:

- **Pruebas Unitarias**: En los módulos, se aplican a las funciones, estructuras de decisión, control de flujo, etc. usadas dentro de él.
- **Pruebas de Integración**: Se valida la interacción entre módulos a través de las interfaces, comunicación entre los mismos, etc.
- **Pruebas de Validación**: se aplican cuando se comprueba el cumplimiento de requisitos por la aplicación (pruebas alfa[programador-usuario] y beta[usuario])
- **Pruebas de Sistema**: Se aplican con el sistema en funcionamiento (producción). Ejemplo: pruebas de seguridad, rendimiento, recuperación, etc

Herramientas de Pruebas y depuración

Durante el desarrollo de la aplicación web, es necesario realizar las tareas de pruebas y depuración de código del programa. Para ello, básicamente nos centraremos en las siguientes herramientas disponibles para PHP:

- **PHP Unit**: Herramienta para poder diseñar y ejecutar las pruebas unitarias
- **XDebug**: Herramienta para poder depurar nuestro código, integrándola en el IDE que hayamos elegido para desarrollar código (Visual Studio Code en nuestro caso)

Herramienta PHPUnit

- Instalación: https://phpunit.de/getting-started/phpunit-10.html
  1) Abrimos terminal en nuestro proyecto y ejecutamos;
     wget https://phar.phpunit.de/phpunit-10.phar
     chmod +x phpunit-10.phar
  2) Comprobamos con ./phpunit-10.phar --version
- En Visual Studio Code podemos instalar la extensión PHP Unit
- Normalmente crearemos en nuestro proyecto una carpeta test o tests donde guardaremos los test unitarios.
- Los test deben guardarse con nombre terminado en ***Test.php**

Ver manual PHPUnit: https://docs.phpunit.de/en/10.3/index.html

PHP Unit – Ejemplo Calculadora

```
<?php       /* Calculadora.php */
class Calculadora
{
    public function sumar($a=0, $b=0)
    {
        return $a + $b;
    }
    public function restar($a=0,$b=0)
    {
        //
    }
    public function multiplicar($a=1,$b=1)
    {
       //
    }
    public function dividir($a=1,$b=1)
    {
        //
    }
}
?>
```

PHP Unit – Ejemplo CalculadoraTest

```
<?php     /* CalculadoraTest.php */

use Calculadora;
use PHPUnit\Framework\TestCase;

class CalculadoraTest extends TestCase
{    //El nombre de las funciones de pruebas debe comenzar por test*
    public function testSumar()
    {
        $cal = new Calculadora();
        $this->assertEquals( 6, $cal->sumar(2,4), "2+4 debe dar 6" );
        // más assertEquals tests...
    }
 }
?>
```

Herramienta PHP Unit - Ejemplo

- Podemos ejecutar las pruebas en el terminal de Visual Studio Code con la instrucción:

./phpunit-10.phar --bootstrap ./src/Calculadora.php test

- Añadiendo el directorio test, ejecuta todos los test de la carpeta, si queremos que ejecute sólo un test en concreto lo indicamos:

./phpunit-10.phar --bootstrap ./src/Calculadora.php **./test/CalculadoraTest.php**

- La ejecución de la instrucción nos dará el resultado de las pruebas:

```
PHPUnit 10.3.5 by Sebastian Bergmann and contributors.
Runtime:       PHP 8.1.2-1ubuntu2.14
.                                                   1 / 1 (100%)
Time: 00:00.001, Memory: 24.46 MB
OK (1 test, 1 assertion)
```

Herramienta Xdebug - Install

- Si no lo tenemos ya instalado, visitaremos https://xdebug.org/wizard.php, pegaremos el resultado de phpinfo() y nos indicará la versión adecuada para la versión de PHP instalada así como las instrucciones para instalarlo.
- Debemos actualizar php.ini con las siguientes directivas al final del fichero (a partir de xDebug 3.0):

```
[XDebug]
xdebug.remote_enable = 1
xdebug.start_with_request = yes
xdebug.idekey=”vscode”
zend_extension = /*la ubicación que se indique en la instalación*/ (.so en Ubuntu ó .dll en Windows)
```

Herramienta XDebug

- Para activar la depuración ejecutaremos **Run → StartDebugging** y seleccionaremos **PHP**.
- Lo que ejecutemos a partir de entonces se detendrá en los puntos de ruptura que hayamos establecido.
- Nos saldrá una pequeña barra con opciones para depurar paso a paso, pausar, reiniciar, parar, etc.
- Si queremos realizar una traza de las pruebas, nos debemos asegurar de activar el debugging, indicar los puntos de ruptura y en el terminal de Visual Studio Code ejecutamos las pruebas con la instrucción indicada en PHPUnit (./phpunit-10.phar)