La Sénia
Institut d'Educació Secundària
Paiporta

UD2. Introducción a PHP

Desarrollo Web en Entorno Servidor

Profesora: Silvia Vilar Pérez

curso 2024-2025

Contenidos

- Características PHP.
- Etiquetas para inserción de código.
- Uso de directivas. PHP.ini
- Ámbito de las variables.
- Constantes.
- Variables Superglobales
- Variables variables
- Tipos de datos. Conversiones entre tipos de datos.
- Sintaxis básica de PHP

Características PHP

- El código PHP está embebido en documentos HTML, para introducir dinamismo fácilmente a un sitio web.
- El intérprete PHP ignora el texto del fichero HTML hasta que encuentra una etiqueta de inicio del bloque de código PHP embebido.
- Como PHP se ejecuta del lado del servidor sólo puede tener acceso a los datos del propio servidor.
  - No puede acceder a los recursos del cliente
  - No puede saber qué hora es en el cliente
  - No puede acceder a los archivos del cliente

  (Con la excepción de las Cookies)

Etiquetas para inserción de código

Hay varias formas para delimitar los bloques de código PHP:

1. La opción que asegura portabilidad (recomendada):

<?php *codigo php* **?>**

2. Usando las short tags o etiquetas de formato corto:
   Requiere activar directiva short_open_tag=on

<? *código php* **?>**

3. Podemos asociar el lenguaje a etiquetas de scripting (Eliminada desde la versión 7.0):

<script language="PHP"> *codigo php* **</script>**

4. Usando las etiquetas de ASP (Eliminada desde la versión 7.0): ***Requiere activar directiva asp_tags=on***

<% *código php* %**>**

NOTA: el delimitador **<?=** es abreviatura de **<?php echo**

Etiquetas para inserción de código

Dentro de dichas etiquetas aplicaremos la sintaxis de PHP. Las sentencias de código finalizan con "**;**"

- Los espacios, tabulaciones, intros, etc en el código embebido no tienen otro efecto que **mejorar legibilidad**
- Los scripts se pueden incrustar en cualquier sección del HTML y puede haber un número indefinido en el fichero.
- Además, debemos guardarlo con la extensión **.php** para que el servidor use el intérprete de PHP. Éste sustituye el script por el resultado de su ejecución, incluyendo las etiquetas de inicio y fin.

Uso de directivas – php.ini

Para configurar el entorno del Servidor para PHP, se modifica el fichero **php.ini** (intérprete de PHP)

- El fichero **php.ini** indica una serie de valores que determinan el comportamiento del intérprete PHP
- Se encuentra ubicado en el directorio raíz bajo los nombres **php.ini-development** y **php.ini-production**
- En Ubuntu lo encontramos en /etc/php/8.3/cli/php.ini (8.3 es la versión instalada, puede variar)
- Las instrucciones del fichero se denominan **Directivas**

Listado de directivas de php.ini
https://www.php.net/manual/es/ini.list.php

Uso de directivas – php.ini

- Las directivas se forman por una pareja de clave-valor.
- Las directivas que comienzan por **;** están comentadas y son ignoradas por el motor del intérprete.
- Para indicar las rutas dentro del fichero se utilizan los formatos:

  C:\directorio\directorio
  \directorio\directorio
  /directorio/directorio

- El fichero php.ini se lee cada vez que se arranca el servidor web.
- El servidor busca el fichero php.ini por este orden:
  - En el propio directorio de php.
  - En la ruta definida como variable de entorno.
  - En el directorio del sistema (Ej: C:\Windows) que es la opción recomendada.

Ámbito de las variables

Es el contexto en el que se puede acceder a una variable.

- En PHP existen variables **locales** y **globales**.
- Las variables se definen como globales precediéndolas de la palabra ***global***.

```
global $var=5;
```

- También las podemos definir como globales asignándolas a la matriz superglobal $GLOBALS.

```
$GLOBALS[$var]=5;
```

Ámbito de las variables

Si queremos mantener el valor de una variable local en las sucesivas llamadas a la función hay que definirla como ***static***

```
<?php
    function test(){
        static $a = 0;
        echo $a;
        $a++;
        echo $a;
}
test();
echo "\n";
test();
echo "\n";
test();
?>
```

Salida del código:

01
12
23

Ámbito de las variables

```
<?php
function PruebaSinGlobal(){
$var++;
echo “Prueba sin global. \$var: “.$var.”\n”;
}
function PruebaConGlobal(){
global $var;
$var++;
echo “Prueba con global. \$var: “.$var.”\n”;
}
function PruebaConGlobals(){
$GLOBALS[“var”]++;
echo “Prueba con GLOBALS. \$var: “.$GLOBALS[“var”].”\n”;
}
//variable global
$var=20;
PruebaSinGlobal();
PruebaConGlobal();
PruebaConGlobals();
?>
```

Resultado:

Prueba sin global. $var: 1
Prueba con global. $var: 21
Prueba con GLOBALS. $var: 22

Constantes

- Una constante es un identificador de un dato que no cambia de valor durante toda la ejecución de un programa.
- Las constantes no se asignan con el operador =, sino con la función **define**:

  define(nombre_constante_entre_comillas, dato_constante);

  define (“PI”, 3.1416);
  print PI;
- No llevan $ delante
- La función defined(“PI”) devuelve TRUE si existe la constante.
- Son siempre globales por defecto.
- Sólo se pueden definir constantes de los tipos escalares (boolean, integer, double, string)

Constantes Predefinidas

Dependen de las extensiones que se hayan cargado en el servidor, aunque hay constantes predefinidas que siempre están presentes:

- PHP_VERSION: Indica la versión de PHP que se está utilizando.
- PHP_OS: Nombre del sistema operativo que ejecuta PHP.
- TRUE
- FALSE
- E_ERROR: Indica los errores de interpretación que no se pueden recuperar.
- E_PARSE: Indica errores de sintaxis que no se pueden recuperar.
- E_ALL: Representa a todas las constantes que empiezan por E_.

Utilización de las variables

- Restricciones sobre las variables:
  - Deben comenzar por **$**
  - Seguidamente debe haber una letra (may/min) o guión bajo (_)
  - El resto de caracteres pueden ser números, letras guiones bajos
- PHP es case sensitive
- Si una variable se compone de varias palabras, se aconseja escribirla en minúsculas excepto el inicio de la siguiente palabra (camelCase)
- Las **variables predefinidas** siguen el patrón **$_VARIABLE** por lo que se desaconseja usar este mismo patrón
- Las variables o cadenas se pueden concatenar usando el punto (**.**)

Variables

Además de las variables definidas por el programador, existen gran cantidad de *variables predefinidas* que se pueden usar libremente:

- ***Variables de entorno***: Variables que el servidor pone a disposición de PHP e indirectamente del programador
- ***Variables de PHP***: Variables predefinidas que pertenecen al intérprete PHP y que éste pone a disposición del programador.

- A partir de PHP 4 se incluyen matrices **superglobales** que centralizan todas las variables predefinidas.

$GLOBALS, $_SERVER, $_GET, $_POST, $_COOKIE, $_FILES, $_ENV, $_REQUEST, $_SESSION

Variables Superglobales

- **$_GET**: lleva los datos de forma "visible" al cliente (navegador web). El **medio de envío es la URL**. Para recoger los datos que llegan en la url se usa $_GET.

Ejemplo: www.midominio.com/action.php?nombre=silvia&apellidos1=vilar

- **$_POST**: consiste en datos "ocultos" (porque el cliente no los ve) **enviados por un formulario** cuyo método de envío es POST. Ideal para formularios. Para recoger los datos que llegan por este método se usa $_POST.
- **$_REQUEST**: Con la variable $_REQUEST **recuperaremos los datos** de los formularios enviados tanto por **GET como por POST**.

Variables Superglobales $_GET

Cadena de caracteres añadida a la URL:

http://example.com/?**nombre=Silvia&apellido=Vilar**

- Código PHP:

```
<?php
echo '¡Hola ' . $_GET["nombre"] . ' ' . $_GET["apellido"]'!';
echo '¡Hola ' . $_REQUEST["nombre"] . ' ' .
$_REQUEST["apellido"]'!';
?>
```

- Resultado:

¡Hola Silvia Vilar!

Variables Superglobales $_POST y $_REQUEST

A través de un formulario:

- Formulario HTML:

```
<form action="ejemplo.php" method="POST">
    Nombre usuario: <input type="text" name="username" /><br />
    Email:  <input type="text" name="email" /><br />
    <input type="submit" name="btnEnviar" value="Enviar" />
</form>
```

- Código PHP:

```
<?php
echo "¡Hola “ . $_POST['username'] . "!";
echo "¡Hola “ . $_REQUEST['username'] . "!";
?>
```

- Resultado:

¡Hola Silvia!
¡Hola Silvia!

Variables Superglobales $_COOKIE

$_COOKIE: Recoge las variables pasadas al script actual mediante Cookies

- Ejemplo:

```
//creación de la cookie
<?php
  setcookie("nombre", 'Silvia', time()+3600);
?>
//obtención de la cookie
<?php
echo '¡Hola ' .
htmlspecialchars($_COOKIE["nombre"]) . '!';
?>
```

- Resultado

¡Hola Silvia!

Variables Superglobales $_SERVER

$_SERVER: contiene información de cabeceras, rutas, ubicaciones, etc. del script

- Ejemplo:

```
<?php
echo $_SERVER['SERVER_NAME'];
?>
```

- Resultado

www.example.com

Variables de Variables

Se pueden crear nombres de variables dinámicamente anteponiendo **$$** a una variable.

- La variable ***variable*** toma su nombre del valor de otra variable previamente declarada.
- Ejemplo:

```
<?php
 $var = "uno";
 $$var = "dos";
 print ($var);  // produce el texto: "uno"
 print ($uno);  // produce el texto: "dos"
 print ($$var);          // produce el texto: "dos"
 print (${$var});
```

- A diferencia de las variables por referencia, se están creando dos variables distintas que ocupan direcciones de memoria distintas.

Variables de Variables

Se pueden crear nombres de variables dinámicamente anteponiendo $$ a una variable.

- Otro Ejemplo:

```
<?PHP
    $mensaje_es="Hola";
    $mensaje_en="Hello";
    $idioma = "en";
    $mensaje = "mensaje_" . $idioma;
    print $$mensaje;
?>
```

Devolvería Hello

Tipos de datos. Conversiones entre tipos

- PHP soporta los tipos de datos primitivos:
  - Números enteros
  - Números en coma flotante
  - Cadenas de caracteres
  - Booleanos
  - Objetos
  - Recursos
  - NULL
- El tipo de una variable no se suele especificar. Se decide en tiempo de ejecución en función del contexto y puede variar.

Tipos de datos. Conversiones entre tipos

- Números enteros: Enteros positivos y negativos

  $var = 20; $var = -20; // asignación decimal

  $var = 024; $var = -024; // asignación octal

  $var = 0x14; $var = -0x14; // asignación hexadecimal
- Números en coma flotante: Permiten almacenar una parte fraccionaria.

  $var = 260.78;

  $var = 26078e-2;
- Booleanos: Pueden almacenar los valores True (1) y False (0).
- Recursos: Son valores especiales que hacen referencia a una información de estado o memoria de origen externo a PHP. Ejemplo: una conexión a BD

Tipos de datos. Conversiones entre tipos

Funciones de interés:

- La función **gettype()** devuelve el tipo de una variable
- Las funciones **is_type** comprueban si una variable es de un tipo dado:

  is_array(), is_bool(), is_float(), is_integer(), is_null(), is_numeric(), is_object(), is_resource(), is_scalar(),is_string()
- La función **var_dump()** muestra el tipo y el valor de una variable. Es especialmente interesante con los arrays.

Tipos de datos. Conversiones entre tipos

Tipo string:

- Comillas simples o dobles (en este caso interpreta secuencias de escape como caracteres especiales)
- Otra forma de inicializar cadenas es utilizar la sintaxis heredoc (desde PHP 4) y nowdoc (desde PHP 5.3) que utilizan el símbolo de documento incrustado (“<<<“) y un identificador para marcar el final del documento.

Ver:

https://www.php.net/manual/es/language.types.string.php#language.types.string.syntax.heredoc
https://www.php.net/manual/es/language.types.string.php#language.types.string.syntax.nowdoc

NOTA: Es importante no escribir ningún carácter, salvo \n, antes y después del identificador de cierre de la cadena.

- Acceso a un carácter de la cadena: $inicial=$nombre**{0}**;

Tipos de datos. Conversiones entre tipos

- Ejemplos de inicialización de cadenas:

```
$a = 9;
print ‘a vale $a\n’; // muestra a vale $a\n
print “a vale $a\n”; // muestra a vale 9 y avanza una línea
print “<IMG SRC=‘logo.gif’>”; // muestra <IMG SRC=‘logo.gif’>
print “<IMG SRC=\”logo.gif\”>”; //muestra <IMG SRC=“logo.gif”>
$nombre=“Silvia”;

$var = <<<xxx // Sintaxis heredoc con delimitador xxx
Esta es una cadena que termina al encontrarse xxx. $nombre
xxx;
/* Resultado a mostrar: Esta es una cadena que termina al encontrarse xxx Silvia*/
```

Conversiones automáticas de tipos de datos.

- PHP es muy flexible en el manejo de los tipos de datos.
- PHP evalúa la operación a realizar y el tipo de los operandos, y adapta los operandos para poder realizar la operación lo más correctamente posible.
- En operaciones entre enteros y coma flotante el resultado es un número en coma flotante
- Una concatenación de cadenas con una variable numérica hace que ésta última sea convertida a cadena.

```
$varN=1;
$varC='4 flores';
$varC=$varN.$varC; // el resultado es 14 flores
```


Sin embargo, $varN=$varC+$varN el resultado sería 5

Conversiones automáticas de tipos de datos.

Reglas automáticas de conversión de tipos:

- En operaciones lógicas, los datos NULL, 0, '0' y ‘ ' se consideran FALSE. Cualquier otro dato se considera TRUE (incluida la cadena 'FALSE').
- En operaciones aritméticas no unitarias las cadenas se intentan leer como números y, si no se puede, se convierten en 0, TRUE se convierte en 1, y FALSE se convierte en 0.
- En operaciones de comparación, si un operando es un número, el otro también se convertirá en un número. Sólo si ambos operandos son cadenas se compararán como cadena.
- En operaciones de cadenas de caracteres, NULL y FALSE se convierten en ‘ ', y TRUE se convierte en '1'.

Conversión forzada de tipos de datos.

- La conversión automática que realiza PHP no siempre es lo que queremos obtener.
- PHP permite otras conversiones implícitas de tipos :

  (int) : Fuerza la conversión a entero

  (real), (double), (float): Fuerza la conversión a coma flotante.

  (string): Fuerza la conversión a cadena de caracteres.

  (array): Fuerza la conversión a matriz

  (object): Fuerza la conversión a un objeto.

Sintaxis básica PHP

- Comentarios (como en C):
  - /* … */ varias líneas
  - // una línea
  - # Comentario estilo shell para una línea
- Para imprimir: echo y print
  - **echo**: muestra una o más cadenas separadas por comas

    echo “Hola mundo”;

    echo “Hola “, “mundo”; //elementos separados

    echo “Hola “ . “mundo”; //elementos concatenados
  - **print**: muestra una cadena o varias unidas por el operador punto (.)

    print “Hola “ . “mundo”;

    print “Hola mundo”;

Sintaxis básica PHP

- Uso de \n para generar código HTML legible
  - Sin el carácter \n

Código PHP

```
print ("<P>Párrafo 1</P>");
print ("<P>Párrafo 2</P>");
```

Código HTML

```
<P>Párrafo 1</P><P>Párrafo 2</P>
```

Salida

```
Párrafo 1

Párrafo 2
```

Sintaxis básica PHP

- Uso de \n para generar código HTML legible
  - Con el carácter \n

Código PHP

```
print ("<P>Párrafo 1</P>\n");
print ("<P>Párrafo 2</P>\n");
```

Código HTML

```
<P>Párrafo 1</P>
<P>Párrafo 2</P>
```

Salida

```
Párrafo 1

Párrafo 2
```

Expresiones y Operadores

- Operadores aritméticos: +, -, *, /, %, ++, --
- Operador de asignación: =
- Operadores combinados: -=, +=, *=, /=, .=, %=
- Ejemplos:

```
$a = 3; $a += 5;   // a vale 8
$b = “hola ”;
$b .= “mundo”;// b vale “hola mundo”
```

- Operadores de comparación: ==, !=, <, >, <=, >= y otros

Expresiones y Operadores

- Operador de identidad **===** compara también el tipo de las variables.
- Operador de control de error: @

Antepuesto a una expresión, evita cualquier mensaje de error que pueda ser generado por la expresión y continua la ejecución

```
<?php
$var1=3; $var2=0;
$huboerror="no se produce resultado por error";
$nohuboerror="variable con valor";
@$resultado = $var1/$var2;
echo (empty($resultado))? huboerror : nohuboerror;
?>
```

- Operadores lógicos: && (and), || (or), ! , xor
- Operadores de cadena:
  - concatenación: **.** (punto)
  - asignación con concatenación: **.=**

Inclusión de Ficheros externos en PHP

- La inclusión de ficheros externos se consigue con:
  - **include()**
  - **require()**
- Ambos incluyen y evalúan el fichero especificado
- Diferencia: en caso de error include() produce un warning y require() un error fatal
- Se usará require() si al producirse un error debe interrumpirse la carga de la página
- Si usamos **include_once()** o **required_once()**, sólo lo realizará la primera vez

Inclusión de Ficheros externos en PHP

```
<HTML>
    <HEAD>
        <TITLE>Título</TITLE>
        <?PHP
        // Incluir bibliotecas de funciones
        require ("conecta.php");
        require ("fecha.php");
        require ("cadena.php");
        require ("globals.php");
        ?>
    </HEAD>
    <BODY>
        <?PHP
            include ("cabecera.html");
        ?>
        // Código HTML + PHP
        <?PHP
            include ("pie.html");
        ?>
    </BODY>
</HTML>
```

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)