

FUNDAMENTOS DE PHP - 02

Arrays, funciones.

Array

- **Array:** permite guardar una colección de variables
 - Similar al de otros lenguajes de programación
- **Ejemplo, declaración de arrays:**

```
<?php
    $mediciones = array(15, 43, 56);
    var_dump($mediciones);

    $ciudades = array("Bogota", "Ibagué", "Manizales");
    var_dump($ciudades);
?>
```

Hay tres tipos de arrays:

- Indexados: se utiliza el índice número de la posición para acceder al valor asociado a ella: 0, 1, 2.
- Asociativos: se utilizan claves únicas para acceder a cada una de las posiciones.
- Multidimensionales: Dentro de cada posición contienen uno o más arrays.

Array Indexado

- Cada posición tiene un índice (de 0 a N)
- Utilizamos el indice para: asignar, modificar o consultar el valor
- Las dimensiones del array se modifican al agregar nuevos valores.

```
<?php
    $mediciones1 = array(15, 43, 56);
    // modificar valores
    $mediciones1[0] = 16;
    $mediciones1[1] = 44;
    $mediciones1[2] = 57;
    // agregar nuevo valor en nueva posición
    $mediciones1[3] = -1;
    var_dump($mediciones1);
?>
```

Recorrer un array

- La función **count(\$array)** cuenta el número de elementos
- Usando un bucle **for** o **foreach**

```
<?php
    $mediciones2 = array(13, 27, 37);
    $stam = count($mediciones2);
    for($i = 0; $i < $stam; $i++) {
        echo $i . " : " . $mediciones2[$i] . "</br>";
    }
    foreach($mediciones2 as $medicion2) {
        echo " >>> " . $medicion2 . "</br>";
    }
?>
```

Con las funciones `sort($array)` y `rsort($array)` podemos ordenar los elementos de un array indexado de forma ascendente y descendente respectivamente.

```
<?php
    $mediciones3 = array(43, 68, 55);
    rsort($mediciones3);
    var_dump($mediciones3);

    $mediciones4 = array(83, 75, 91);
    sort($mediciones4);
    var_dump($mediciones4);
?>
```

Array asociativo

- Cada posición tiene una clave única, la clave es un valor que nosotros definimos, "clave" => "valor"
 - Similares a las tablas Hash en otros lenguajes.

```
<?php
    $poblacion1 = array("Bogota"=>6.50, "Ibagué"=>0.54,
    "Manizales"=>0.42);
    // modificar valores
    $poblacion1["Bogota"] = 7.70;
    // agregar nuevo valor en nueva posición
    $poblacion1["Barranquilla"] = 1.21;
    var_dump($poblacion1);
    // consultar valores
    echo "Población de Ibagué: " . $poblacion1["Ibagué"] . "<br>";
?>
```

Recorriendo Array asociativo

- Para recorrer estos arrays solemos utilizar un bucle `foreach`, pero de forma diferente a los arrays indexados, en este caso recorremos las claves y los valores de cada posición.

```
<?php
    $poblacion2 = array("Cali"=>2.21, "Medellín"=>2.44,
    "Pereira"=>0.39);
    foreach($poblacion2 as $clave => $valor) {
        echo $clave. " : " . $valor . "<br>";
    }
?>
```

Los elementos de estos arrays se pueden ordenar utilizando las funciones `asort($array)` y `arsort($array)` para ordenador por valor de forma ascendente y descendente respectivamente.

Para ordenarlos por su clave se pueden utilizar las funciones `ksort($array)` y `krsort($array)`

```
<?php
$frutas1 = array("d"=>"manzana", "a"=>"naranja", "b"=>"banano",
"c"=>"uva");
    ksort($frutas1);
    foreach ($frutas1 as $key => $val) {
        echo "$key = $val" . "<br>";
    }
?>
```

asort — Ordena un array manteniendo la correlación de los índices del array con los elementos con los que están asociados. Esta función se utiliza principalmente para ordenar arrays asociativos en los que el orden es importante.

```
<?php
$frutas2 = array("d"=>"manzana", "a"=>"naranja", "b"=>"banano",
"c"=>"uva");
    asort($frutas2);
    foreach ($frutas2 as $key => $val) {
        echo "$key = $val" . "<br>";
    }
?>
```

Arrays Multidimensionales

- Son los más complejos, dentro de cada posición pueden tener más arrays, de las mismas o diferentes dimensiones

```
<?php
$ciudades = array(
    "Cartagena" => array (
        "Poblacion" => 3.14,
        "Superficie" => 709,
        "Departamento" => "Bolivar"
    ),
    "Florencia" => array (
        "Poblacion" => .15,
        "Superficie" => 2292,
        "Departamento" => "Caqueta",
        "Region" => "Amazonia"
    )
);
echo "Población de Cartagena: "
    . $ciudades['Cartagena']['Poblacion'] . "</br>";
echo "Region de Florencia: "
    . $ciudades['Florencia']['Region'] . "</br>";
?>
```

Document X +

← → ⌂ i localhost/1803183/index.php

```
C:\xampp\htdocs\1803183\index.php:12:
array (size=3)
    0 => int 15
    1 => int 43
    2 => int 56

C:\xampp\htdocs\1803183\index.php:15:
array (size=3)
    0 => string 'Bogota' (Length=6)
    1 => string 'Ibagué' (Length=6)
    2 => string 'Manizales' (Length=9)

C:\xampp\htdocs\1803183\index.php:24:
array (size=4)
    0 => int 16
    1 => int 44
    2 => int 57
    3 => int -1

0 : 13
1 : 27
2 : 37
>>> 13
>>> 27
>>> 37

C:\xampp\htdocs\1803183\index.php:37:
array (size=3)
    0 => int 68
    1 => int 55
    2 => int 43

C:\xampp\htdocs\1803183\index.php:41:
array (size=3)
    0 => int 75
    1 => int 83
    2 => int 91

C:\xampp\htdocs\1803183\index.php:50:
array (size=4)
    'Bogota' => float 7.7
    'Ibagué' => float 0.54
    'Manizales' => float 0.42
    'Barranquilla' => float 1.21

Población de Ibagué: 0.54
Cali : 2.21
Medellín : 2.44
Pereira : 0.39
a = naranja
b = banano
c = uva
d = manzana
b = banano
d = manzana
a = naranja
c = uva
Población de Cartagena: 3.14
Región de Florencia: Amazonía
```

Funciones propias

- Las funciones son fragmentos de código que pueden recibir parámetros y retornar valores. Anteriormente hemos invocado a muchas funciones integradas en PHP (rsort(), var_dump(), etc.) pero además podemos crear funciones propias. El uso de funciones tiene dos partes, la declaración y las invocaciones a la función, para hacer que se ejecute su contenido. A diferencia de lo que ocurre en otros lenguajes no es necesario declarar la función antes de sus invocaciones.

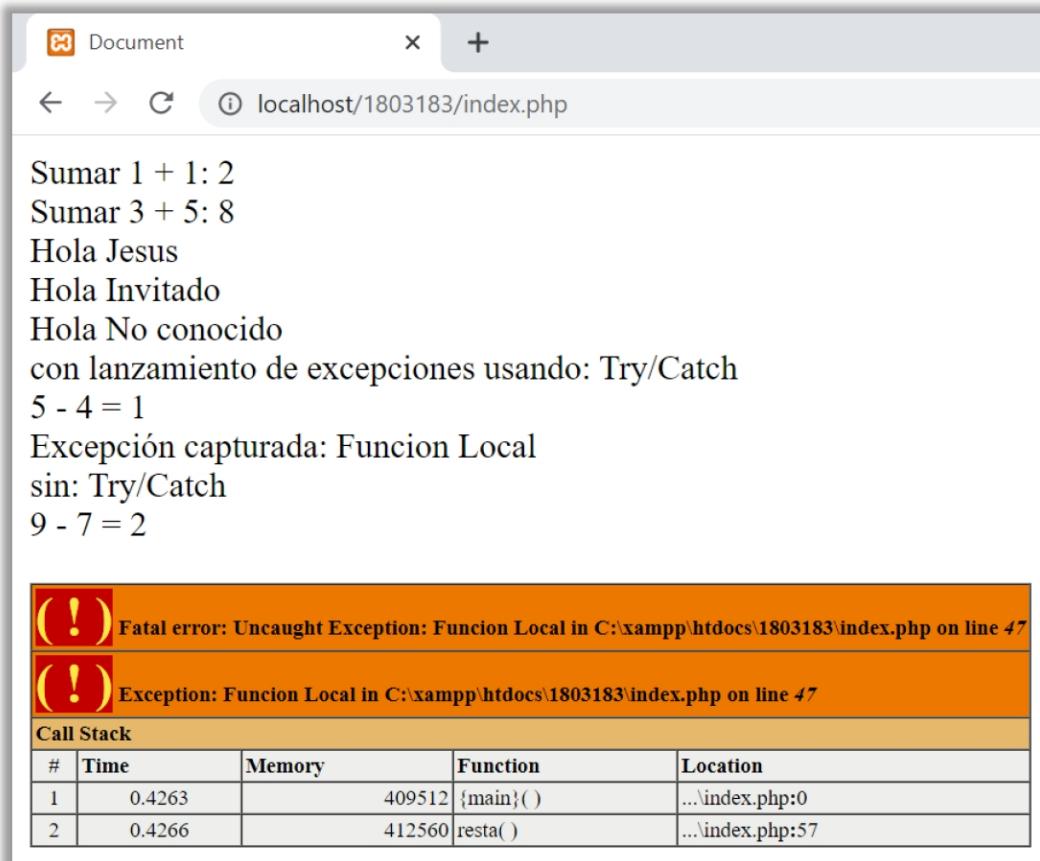
```
<?php
    // Invocación 1
    echo "Sumar 1 + 1: " . sumar(1, 1)."</br>";
    // Declaración: Parámetros $a y $b y retorno
    function sumar($a, $b) {
        $resultado = $a + $b;
        // retorno de valor
        return $resultado;
    }
    // Invocación 2
    echo "Sumar 3 + 5: " . sumar(3, 5)."</br>";
?
>
```

La recepción de parámetros por parte de la función y la posibilidad de que retorne un valor es opcional, es decir puede haber funciones que lo utilicen o no. También se pueden incluir valores por defecto a los parámetros, en ese caso las invocaciones pueden no definir un valor para el parámetro y se utilizará el valor por defecto.

```
<?php
    // sin retorno
    function saludar1($nombre) {
        echo "Hola " . $nombre . "</br>";
    }
    // sin retorno ni parámetros
    function saludar2() {
        echo "Hola Invitado</br>";
    }
    // con parámetro opcional, (valor por defecto en caso de omisión)
    function saludar3($nombre = "No conocido") {
        echo "Hola " . $nombre . "</br>";
    }
    // Invocaciones
    saludar1("Jesus");
    saludar2();
    saludar3();
?
>
```

La creación de funciones es imprescindible para realizar un desarrollo estructurado, simplificar la implementación de los programas y favorecer la reutilización de código. Las funciones deben realizar tareas específicas e independientes dentro del programa.

```
<?php
    function resta($a, $b ) {
        // "resultado" es una variable local
        // Solo se puede utilizar en la función
        $resultado = $a - $b;
        echo $a . " - " . $b . " = " . $resultado . "<br>";
        throw new Exception('Funcion Local');
    }
    try{
        echo "con lanzamiento de excepciones usando: Try/Catch" . "
"</br>;
        resta(5,4);
        echo $resultado; // <- error!!!
    }catch(Exception $e){
        echo 'Excepción capturada: ', $e->getMessage() . "<br>";
    }
    echo "sin: Try/Catch" . "</br>";
    resta(8,7);
    echo $resultado; // <- error!!! - debe mostrar un: Fatal error
?>
```



The screenshot shows a browser window with the following content:

```
Document
← → C localhost/1803183/index.php
```

Output from the PHP script:

```
Sumar 1 + 1: 2
Sumar 3 + 5: 8
Hola Jesus
Hola Invitado
Hola No conocido
con lanzamiento de excepciones usando: Try/Catch
5 - 4 = 1
Excepción capturada: Funcion Local
sin: Try/Catch
9 - 7 = 2
```

Fatal error:

(!) Fatal error: Uncaught Exception: Funcion Local in C:\xampp\htdocs\1803183\index.php on line 47

(!) Exception: Funcion Local in C:\xampp\htdocs\1803183\index.php on line 47

Call Stack

#	Time	Memory	Function	Location
1	0.4263	409512	{main}()	...index.php:0
2	0.4266	412560	resta()	...index.php:57

Funciones de php

- Existe un gran número de funciones incluidas en las librerías de PHP que son utilizadas de forma recurrente en el desarrollo de aplicaciones.

Hay muchas funciones útiles para manipular cadenas de texto. Algunas de las más utilizadas son:

`strlen(Cadena)`: para obtener el número de caracteres que hay en una cadena, los espacios en blanco también se cuentan.

`str_word_count(Cadena)`: para obtener el número de palabras que hay en una cadena.

`strpos(Cadena1, Cadena2)`: para encontrar la posición de la cadena2 dentro de la cadena1. Si efectivamente la cadena2 está contenida en la cadena1 nos retornara el índice del primer carácter, si devuelve un numero negativo significa que no hay coincidencia.

`str_replace(cadena1, cadena2, cadena3)`: Busca dentro de la cadena3 la cadena1 y si la encuentra la reemplaza por la cadena2.

`explode(cadena1,cadena2)`: Utiliza la cadena 1 como separador para dividir la cadena2, genera un array de cadenas con todos los fragmentos obtenidos en la división.

`sha1(cadena)`: Calcula el código sha1 (Algoritmo de Hash Seguro) de una cadena.

Lista completa de funciones que nos permiten operar con las cadenas
<http://php.net/manual/es/ref.strings.php>

Algunos Ejemplos de Funciones Útiles:

La función `str_repeat()`: Repite un string

```
<?php
    echo str_repeat("-*", 20) . "</br>";
?>
```

La función `date()`: permite obtener fechas con diferentes formatos, se debe indicar el formato como parámetro utilizando una serie de claves.

```
<?php
    echo date("d/m/Y") . "</br>";
    echo date("H:i:s") . "</br>";
?>
```

La función `rand` permite generar números aleatorios, por defecto genera números entre 0 y `getrandmax()` (número máximo para aleatorios), también podemos establecer un mínimo y un máximo.

```
<?php
    echo rand() . " - Aleatorio entre 0 y ".getrandmax()."";
    echo rand(10, 20) . " - Aleatorio entre 10 y 20</br>";
?>
```

La función include() y require() sirven para incluir archivos dentro de otro archivo. Include() genera una advertencia si no consigue incluir el archivo, en cambio require() genera un error.

La función get_include_path() retorna la ruta por defecto a partir de la cual se incluye los archivos, esta ruta la utilizan las funciones include, require y todas las funciones de manipulación de ficheros.

El include_path que tenemos por defecto se puede modificar utilizando la función set_include_path("nuevo path").

```
<?php
    echo "Contenido del archivo 2 <br>";
?>
```

archivo2.php (crear en la misma carpeta donde se encuentra: index.php)

```
<?php
    echo "Ruta origen del include: ".get_include_path()."<br>";
    require("archivo2.php");
?>
```

Otra función muy útil para incluir dependencias a ficheros (por ejemplo a clases o ficheros que definan funciones) es: require_once(), esta función verifica que el fichero solo haya sido incluido una vez

Funciones para la manipulación de ficheros

Las acciones más comunes consisten en abrir o crear ficheros, leerlos, escribirlos y cerrarlos (una vez acabamos de manipularlos). Para abrir un fichero podemos utilizar fopen(ruta fichero, <modo de apertura>).

Ejemplo de creación / apertura de un archivo.txt con permisos de lectura y escritura: cada vez que se abre la web se escribe una línea nueva en el fichero utilizando fwrite(fichero, Cadena), al final de la escritura se cierra el fichero fclose(fichero).

```
<?php
    $fichero = fopen("prueba.txt", "a+");
    // a+ Apertura para lectura y escritura;
    // coloca el puntero del fichero al final del mismo
    // Si el fichero no existe, se intenta crear.
    if( $fichero == false ) {
        echo "Error <br>";
    } else {
        fwrite( $fichero, "Petición Registrada a las: ".date("H:i:s")."\n" );
        // Cerrar
        fclose($fichero);
        echo "escritura finalizada <br>";
    }
?>
```

Función sleep (int \$seconds) : int

Retrasa la ejecución del programa durante el número de segundos dados por seconds.

sleep(8);

Ejemplo de lectura del fichero anterior: una vez abierto en un modo que soporte la lectura obtenemos el tamaño del fichero con filesize(ruta del fichero), posteriormente utilizamos fread(fichero, tamaño) para leer su contenido en modo texto.

```
<?php
    sleep(8);

    $fichero = fopen("prueba.txt","a+");
    $tam = filesize( "prueba.txt" );
    $texto = fread( $fichero, $tam );

    $texto = str_replace("\n", "</br>", $texto);
    // remplazar saltos de linea textuales '\n' por html
    fclose( $fichero );
    echo $texto;
    echo "escritura finalizada </br>";
?>
```

