Tarea de Recuperación 2

Funciones en PHP

Sandra Manzano Martínez

Índice:

Funciones en Php

Una función es un conjunto de instrucciones que son ejecutadas múltiples veces a lo largo de un programa. Una función puede estar formada por otras funciones.

Sintaxis:

```
<?php
      function nombreFuncion ($argumento1, … , $argumentoN){
            //Instrucciones a realizar;
            return instruccionesARealizar;
      }
?>
```

Existen unas reglas predefinidas para asignar un nombre a una función. Los nombres de las funciones deben de empezar por una letra o por un guión bajo, después pueden ir seguidos de cuantas letras, números y guiones bajos se quiera.
No se diferencia entre mayúsculas y minúsculas, pero es una buena práctica, si el nombre de la función contiene dos palabras, que la segunda palabra empiece por mayúscula (Ej → sumarNumeros).

También se pueden crear funciones anónimas (sin nombre).

```
<?php
      $saludo = function($nombre){
            $printf(“Hola %s\r\n”, $nombre);
      };
      $saludo(“Mundo”);
      $saludo(“php”);
?>
```

Todas las funciones y clases de PHP tienen ámbito global, es decir, se pueden llamar desde fuera de una función y viceversa. No se puede redefinir.

También se puede llamar a funciones recursivas:

```
<?php
	function recursividad($a){
		if($a<20){
			echo $a;
			recursividad($a+1);
		}
	}
?>
```

Una función puede definir valores predeterminados al estilo de C++ para argumentos escalares como sigue:

```
<?php
	function hacer_café($tipo = "capuchino"){
	    return "Hacer una taza de $tipo.\n";
	}

	echo hacer_café();
	echo hacer_café(null);
	echo hacer_café("espresso");
?>
```

PHP tiene soporte para listas de argumentos de longitud variable en funciones definidas por el usuario. Esto se implementa utilizando el token ...

```
	<?php
		function sum(...$números) {
			$acc = 0;
			foreach ($números as $n) {
				$acc += $n;
			}
		   return $acc;
		}
		echo sum(1, 2, 3, 4);
	?>
```

En estos casos podemos utilizar las siguientes funciones:

func_num_args() → devuelve el número de argumentos

func_get_arg(índice) → devuelve el argumento con el índice que hayamos especificado

func_get_args() → devuelve un array con todos los argumentos

La función **function_exists(string $function_name)** nos permite comprobar la lista de funciones definidas, las incluidas (internas) y las definidas. Devuelve TRUE si function_name existe y si es una función, en caso contrario devuelve FALSE

```
<?php
      if (function_exists('imap_open') {
      echo "Las funciones de IMAP están disponibles.<br />\n";

      } else {
      echo "Las funciones de IMAP no están disponibles.<br
/>\n";
      }
?>
```

Include y Exclude

Include

La sentencia include incluye y evalúa el archivo especificado.

Cuando se incluye un archivo, el código que contiene hereda el ámbito de las variables de la línea en la cual ocurre la inclusión. Cualquier variable disponible en esa línea del archivo que hace el llamado, estará disponible en el archivo llamado, desde ese punto en adelante. Sin embargo, todas las funciones y clases definidas en el archivo incluido tienen el ámbito global.

```
vars.php:
<?php
      $color = 'verde';
      $fruta = 'manzana';
?>

test.php:
<?php
      echo "Una $fruta $color"; // Una
      include 'vars.php';
      echo "Una $fruta $color"; // Una manzana verde
?>
```

Si la inclusión ocurre al interior de una función dentro del archivo que hace el llamado, entonces todo el código contenido en el archivo llamado se comportará como si hubiera sido definida dentro de esa función. Por lo tanto, seguirá el ámbito de las variables de esa función. Una excepción a esta regla son las constantes mágicas las cuales son evaluadas por el intérprete antes que ocurra la inclusión.

```php
<?php
function foo(){
    global $color;
    include 'vars.php';
    echo "Una $fruta $color";
}

/* vars.php está en el ámbito de foo() así que *
* $fruta NO está disponible por fuera de éste  *
* ámbito. $color sí está porque fue declarado *
* como global.                                 */
foo();  // Una manzana verde
echo "Una $fruta $color";   // Una verde
?>
```

Cuando un archivo es incluido, el intérprete abandona el modo PHP e ingresa al modo HTML al comienzo del archivo objetivo y se reanuda de nuevo al final. Por esta razón, cualquier código al interior del archivo objetivo que deba ser ejecutado como código PHP, tendrá que ser encerrado dentro de etiquetas válidas de comienzo y terminación de PHP.

Include_once

La sentencia include_once incluye y evalúa el fichero especificado durante la ejecución del script.

Tiene un comportamiento similar al de la sentencia include, siendo la única diferencia de que si el código del fichero ya ha sido incluido, no se volverá a incluir, e include_once devolverá TRUE. Como su nombre indica, el fichero será incluido solamente una vez.

include_once se puede utilizar en casos donde el mismo fichero podría ser incluido y evaluado más de una vez durante una ejecución particular de un script, así que en este caso, puede ser de ayuda para evitar problemas como la redefinición de funciones, reasignación de valores de variables, etc.

```
<?php
     include_once "a.php"; // esto incluirá a.php
     include_once "A.php"; // esto incluirá a.php otra vez
     (sólo PHP 4)
?>
```

Require

Require es idéntico a 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 (E_WARNING) lo cual permite continuar el script.

```
<?php require('archivo.php'); ?>
```

Require_once

La sentencia require_once es idéntica a require excepto que PHP verificará si el archivo ya ha sido incluido y si es así, no se incluye (require) de nuevo.

```
<?php require_once('/var/www/public_html/config.php'); ?>
```

Funciones de cadenas

- **Strlen(cad)**: Devuelve la longitud de una cadena

```
<?php
      $str = 'abcdef';
      echo strlen($str); // 6
      $str = ' ab cd ';
      echo strlen($str); // 7
?>
```

- **Strstr(cad, cad_bus), strchr(cad, cad_bus):** Busca la aparición de una cadena dentro de otra y devuelve la subcadena comprendida entre la primera aparición y el final de cadena. Es case sensitive. Si no la encuentra devuelve una cadena vacía.

```
<?php
      $email  = 'name@example.com';
      $domain = strstr($email, '@');
      echo $domain; // mostrará @example.com
      $user = strstr($email, '@', true); // Desde PHP 5.3.0
      echo $user; // mostrará name
?>
```

- **stristr(cad, cad_bus):** Igual que la anterior. No es case sensitive.

```
<?php
  $email = 'USER@EXAMPLE.com';
  echo stristr($email, 'e'); // salida ER@EXAMPLE.com
```

```
  echo stristr($email, 'e', true); // A partir de PHP 5.3.0,
salida US

?>
```

- **strchr(cad, car_bus):** Busca la aparición de un carácter y devuelve la subcadena comprendida entre la primera aparición y el final de cadena. Si no la encuentra devuelve una cadena vacía.

```
<?php

     echo strchr("Hello world!","world"); //world!

?>
```

- **strpos(cad1, cad2, [, desplz]):** Encuentra la primera posición de aparición de una cadena a partir de cierta posición (si no se indica el valor es 0). Es case sensitive.

```
<?php

     $mystring = 'abc';

     $findme   = 'a';

     $pos = strpos($mystring, $findme);

     // Nótese el uso de ===. Puesto que == simple no
     funcionará como se espera

     // porque la posición de 'a' está en el 1° (primer)
     carácter.

     if ($pos === false) {

         echo "La cadena '$findme' no fue encontrada en la
     cadena '$mystring'";

     } else {

         echo "La cadena '$findme' fue encontrada en la cadena
     '$mystring'";
```

```
        echo " y existe en la posición $pos";
    }
?>
```

- **strrpos(cad, carácter):** Devuelve la posición de la última aparición de un carácter determinado en una cadena. Si no lo encuentra devuelve false. Es case sensitive.

```
<?php
    $pos = strrpos($mystring, "b");
    if ($pos === false) { // nota: tres signos de igual
        // no encontrado...
    }
?>
<?php
    $foo = "0123456789a123456789b123456789c";
    var_dump(strrpos($foo, '7', -5));
    // Comienza mirando hacia atrás cinco posiciones
     // desde el final. Resultado: int(17)
    var_dump(strrpos($foo, '7', 20));
    // Inicia la búsqueda 20 posiciones en el
    // string. Resultado: int(27)
    var_dump(strrpos($foo, '7', 28));  // Resultado:
    bool(false)
?>
```

- **strspn(cadena, máscara):** Obtenemos la longitud de la subcadena más larga que está formada sólo por caracteres contenidos en la máscara. Una vez encontrado un carácter que NO esté en la máscara se abandona la búsqueda. Es case sensitive

```
<?php

	// subject no inicia con ningún carácter de mask

	var_dump(strspn("foo", "o"));

	// examina dos caracteres de subject iniciando en el
	offset 1

	var_dump(strspn("foo", "o", 1, 2));

	// examina un caracter de subject iniciando en el offset
	1

	var_dump(strspn("foo", "o", 1, 1));

?>
```

- **strcspn(cad, máscara):**Obtenemos la longitud de la subcadena más larga que está formada sólo por caracteres NO contenidos en la máscara. Es case sensitive.

```
<?php

	$a = strcspn('abcd',  'apple');

	$b = strcspn('abcd',  'banana');

	$c = strcspn('hello', 'l');

	$d = strcspn('hello', 'world');

	$e = strcspn('abcdhelloabcd', 'abcd', -9);

	$f = strcspn('abcdhelloabcd', 'abcd', -9, -5);
```

```
    var_dump($a);

    var_dump($b);

    var_dump($c);

    var_dump($d);

    var_dump($e);

    var_dump($f);

?>
```

Comparación de cadenas

- **strcmp(cad1, cad2):** Compara dos cadenas y devuelve un valor <0, si la segunda cadena es menor que la primera, mayor que 0 en caso contrario y 0 si son iguales. Es case sensitive

```
<?php

    $var1 = "Hola";

    $var2 = "hola";

    if (strcmp($var1, $var2) !== 0) {

        echo '$var1 no es igual a $var2 en una comparación
    que considera mayúsculas y minúsculas';

    }

?>
```

- **strcasecmp(cad1, cad2):** Igual que la anterior NO case sensitive

```
<?php

    $var1 = "Hello";

    $var2 = "hello";
```

```
    if (strcasecmp($var1, $var2) == 0) {

        echo '$var1 is equal to $var2 in a case-insensitive
    string comparison';

    }

?>
```

- **strncmp(cad1, cad2, num):** Funciona como strcmp, solo que permite comparar los num primeros caracteres de dos cadenas. Si alguna cadena es menor que num se usará su longitud como num para la comparación.

```
<?php

    $str1 = "phpaaa";

    $str2 = "php";

    echo strcmp($str1, $str2); // 3

?>
```

- **strnatcmp(cad1, cad2):** Igual que strcmp pero utiliza la comparación natural de cadenas alfanuméricas. Es case sensitive

```
<?php

    $arr1 = $arr2 = array("img12.png", "img10.png",
    "img2.png", "img1.png");

    echo "Standard string comparison\n";

    usort($arr1, "strcmp");

    print_r($arr1);

    echo "\nNatural order string comparison\n";

    usort($arr2, "strnatcmp");

    print_r($arr2);

?>
```

- **strnatcasecmp(cad1, cad2):** Igual que la anterior NO case-sensitive

```php
<?php

	echo strnatcasecmp("2Hello world!","10Hello WORLD!");
	//-1

	echo "<br>";

	echo strnatcasecmp("10Hello world!","2Hello WORLD!"); //1

?>
```

Operar con subcadenas

- **substr(cad, inicio [, tamaño]):** Devuelve la subcadena que se encuentra a partir de la posición dada y hasta el final si no se indica el tamaño o hasta el tamaño.

  Los argumentos pueden ser negativos de modo que:

  ➔ Una posición de inicio negativa significa que debe empezarse desde el final.
  ➔ Un tamaño negativo indica cuántos caracteres del final de cadena ignoraremos.

```php
<?php

	$rest = substr("abcdef", -1);    // devuelve "f"

	$rest = substr("abcdef", -2);    // devuelve "ef"

	$rest = substr("abcdef", -3, 1); // devuelve "d"

?>
```

- **substr_replace(cad1, cad2, inicio [, tamaño]):** Devuelve una cadena que es el resultado de sustituir parte de la cadena original (indicado por la posición de inicio y opcionalmente por el tamaño) por el contenido de otra cadena. Es case sensitive y los argumentos de inicio y tamaño pueden ser negativos

```
<?php

	$var = 'ABCDEFGH:/MNRPQR/';

	echo "Original: $var<hr />\n";

	/* Estos dos ejemplos reemplazan todo $var por 'bob'. */

	echo substr_replace($var, 'bob', 0) . "<br />\n";

	echo substr_replace($var, 'bob', 0, strlen($var)) . "<br
	/>\n";

	/* Inserta 'bob' justo al comienzo de $var. */

	echo substr_replace($var, 'bob', 0, 0) . "<br />\n";

	/* Estos dos siguientes reemplazan 'MNRPQR' en $var por
	'bob'. */

	echo substr_replace($var, 'bob', 10, -1) . "<br />\n";

	echo substr_replace($var, 'bob', -7, -1) . "<br />\n";

	/* Elimina 'MNRPQR' de $var. */

	echo substr_replace($var, '', 10, -1) . "<br />\n";

?>
```

- **str_replace(cad1, cad2, cad_or):** Devuelve una cadena que es el resultado de sustituir todas las apariciones de cad1 por cad2 en cad_or. Case sensitive

```
<?php

	// Produce: <body text='black'>

	$bodytag = str_replace("%body%", "black", "<body
	text='%body%'>");

	// Produce: Hll Wrld f PHP

	$vowels = array("a", "e", "i", "o", "u", "A", "E", "I",
	"O", "U");

	$onlyconsonants = str_replace($vowels, "", "Hello World
	of PHP");



	// Produce: You should eat pizza, beer, and ice cream
	every day

	$phrase  = "You should eat fruits, vegetables, and fiber
	every day.";

	$healthy = array("fruits", "vegetables", "fiber");

	$yummy   = array("pizza", "beer", "ice cream");


	$newphrase = str_replace($healthy, $yummy, $phrase);



	// Produce: 2

	$str = str_replace("ll", "", "good golly miss molly!",
	$count);

	echo $count;

?>
```

- **strtr(cadena, cadBus, cadRem):** Devuelve una cadena que es el resultado de la sustitución de cada una de las apariciones, en la cadena original, de cada uno de los caracteres de una subcadena, por sus correspondientes caracteres dentro de una cadena de sustitución dada. Case sensitive

```
<?php
	//De esta forma, strtr() hace una conversión byte a byte
	//Por lo tanto, aquí se asume una codificación de un solo byte:
	$addr = strtr($addr, "äåö", "aao");
?>
```

- **substr_count(cadena, patron):** Devuelve el número de apariciones de una subcadena dentro de una cadena. Case sensitive

```
<?php
	$text = 'This is a test';
	echo strlen($text); // 14
	echo substr_count($text, 'is'); // 2

	// el string es reducido a 's is a test', así que muestra 1
	echo substr_count($text, 'is', 3);

	// el texto es reducido a 's i', así que muestra 0
	echo substr_count($text, 'is', 3, 3);
```

```
        // genera una advertencia debido a que 5+10 > 14

        echo substr_count($text, 'is', 5, 10);


        // muestra sólo 1, debido a que no cuenta subcadenas
        traslapadas.

        $text2 = 'gcdgcdgcd';

        echo substr_count($text2, 'gcdgcd');

    ?>
```

Modificación del contenido

Limpieza

- **chop(cad) rtrimcad():** quita espacios en blanco y finalización de a la derecha (final)

```
<?php

        $str = "Hello World!";

        echo $str . "<br>"; //Hello World!

        echo chop($str,"World!"); //Hello

?>
```

- **trim(cad):** limpia al principio y al final

```
<?php
	$str = "Hello World!";
	echo $str . "<br>"; //Hello World!
	echo trim($str,"Hed!"); //llo Worl
?>
```

Relleno

- **str_pad(cad, long):** Rellena una cadena con un carácter de relleno (por defecto es el espacio en blanco) hasta que la cadena resultante tenga la longitud deseada. Opcionalmente se puede indicar el modo de relleno con los siguientes valores:
  - STR_PAD_RIGHT: Rellena por la dcha. Opción por defecto
  - STR_PAD_LEFT: Rellena por la izda
  - STR_PAD_BOTH: Intenta rellenar por ámbos lados

```
<?php
	$input = "Alien";
	echo str_pad($input, 10);// produce "Alien     "
	echo str_pad($input, 10, "-=", STR_PAD_LEFT); // produce
	"-=-=-Alien"
	echo str_pad($input, 10, "_", STR_PAD_BOTH); // produce
	"__Alien___"
	echo str_pad($input,  6, "___"); // produces "Alien_"
	echo str_pad($input,  3, "*");// produces "Alien"
?>
```

Conversación entre may y min

- **strtolower(cad):** Convierte una cadena a minúsculas

```php
<?php

	$str = "Mary Had A Little Lamb and She LOVED It So";

	$str = strtolower($str);

	echo $str; // Muestra: mary had a little lamb and she
	loved it so

?>
```

- **strtoupper(cad):** Convierte cadena a mayúsculas

```php
<?php

	$str = "Mary Had A Little Lamb and She LOVED It So";

	$str = strtoupper($str);

	echo $str; // muestra: MARY HAD A LITTLE LAMB AND SHE
	LOVED IT SO

?>
```

- **ucfirst(cad):** Convierte a mayúsculas el primer carácter de una cadena, siempre que sea alfabético

```php
<?php

	$foo = 'hello world!';

	$foo = ucfirst($foo); // Hello world!

	$bar = 'HELLO WORLD!';

	$bar = ucfirst($bar);  // HELLO WORLD!

	$bar = ucfirst(strtolower($bar)); // Hello world!

?>
```

- **ucwords(cad)**: Convierte a mayúsculas el primer carácter de cada palabra que forma una cadena

```
<?php
	$foo = 'hello world!';
	$foo = ucwords($foo);  // Hello World!
	$bar = 'HELLO WORLD!';
	$bar = ucwords($bar); // HELLO WORLD!
	$bar = ucwords(strtolower($bar)); // Hello World!
?>
```

Enmascaramiento de caracteres

- **addslashes(cad):** Devuelve una cadena igual que la original en la que se han escapado los caracteres especiales “, ‘ y \

```
<?php
	$str = "Is your name O'Reilly?";
	// Outputs: Is your name O\'Reilly?
	echo addslashes($str);
?>
```

- **stripslashes(cad):** Contrariamente a la función anterior elimina ‘\’ de escape

```
<?php
	$str = "Is your name O\'reilly?";
	// Salida: Is your name O'reilly?
	echo stripslashes($str);
?>
```

- **addcslashes(cad):** Nos permite enmascarar cualquier carácter que desee el usuario

```
<?php

     echo addcslashes('foo[ ]', 'A..z');

     // salida:  \f\o\o\[ \]

     // Serán escapadas todas las letras mayúsculas y
     minúsculas

     // ... pero también [\]^_`

?>
```

- **stripcslashes(cadena):** Realiza lo contrario a lo anterior

```
<?php

     echo stripcslashes("Hello \World!"); //Hello World!

?>
```

- **quotemeta(cadena):** Devuelve una cadena de caracteres igual que la original en la que se han escapado los caracteres especiales: . \ + * ? [^] ($)

```
<?php

     $str = "Hello world. (can you hear me?)";

     echo quotemeta($str); //Hello world\. \(can you hear
     me\?\)

?>
```

División de cadenas

- **strtok(cadena, divisor):** Divide una cadena en diferentes subcadenas

```php
<?php

	$string = "This is\tan example\nstring";

	/* Utiliza tabulador y nueva línea como caracteres de
	tokenización, así  */

	$tok = strtok($string, " \n\t");

	while ($tok !== false) {

	    echo "Word=$tok<br />";

	    $tok = strtok(" \n\t");

	}

?>
```

- **chunk_split(): (cadena [, long [, separador]]):** Permite marcar una cadena en porciones diferenciadas de tamaño menor. Podemos indicarle el tamaño de estas porciones e, incluso, el carácter o cadena con el que las separará. No modificará la cadena original.

```php
<?php

	// formatea $datos usando la semántica del RFC 2045

	$nueva_cadena = chunk_split(base64_encode($datos));

?>
```

- **split(patron, cadena):** Devuelve un array resultado de dividir una cadena en diferentes subcadenas.

```
<?php

	// Los delimitadores pueden ser barra, punto o guión

	$fecha = "04/30/1973";

	list($mes, $día, $año) = split('[/.-]', $fecha);

	echo "Mes: $mes; Día: $día; Año: $año<br />\n";

?>
```

- **implode():** Une los elementos de un array en un string

```
<?php

	$array = array('apellido', 'email', 'teléfono');

	$separado_por_comas = implode(",", $array);

	echo $separado_por_comas; // apellido,email,teléfono


	// Devuelve un string vacío si se usa un array vacío:

	var_dump(implode('hola', array())); // string(0) ""

?>
```

- **explode():** Divide un String en un array con varios strings

```
<?php

	// Ejemplo 1

	$pizza  = "porción1 porción2 porción3 porción4 porción5
	porción6";
```

```
        $porciones = explode(" ", $pizza);

        echo $porciones[0]; // porción1

        echo $porciones[1]; // porción2

        // Ejemplo 2

        $datos = "foo:*:1023:1000::/home/foo:/bin/sh";

        list($user, $pass, $uid, $gid, $gecos, $home, $shell) =
        explode(":", $datos);

        echo $user; // foo

        echo $pass; // *

    ?>
```

Relacionadas con HTML

- **htmlspecialchars(cad):** Se encarga de convertir los caracteres con significado especial en HTML en entidades HTML según lo siguiente:
    - & → &< → <
    - “ → "> → >

```
    <?php

        $nuevo = htmlspecialchars("<a href='test'>Test</a>",
        ENT_QUOTES);

        echo $nuevo; // &lt;a
        href=&#039;test&#039;&gt;Test&lt;/a&gt;

    ?>
```

- **htmlentities(cad):** Similar a la anterior pero traduce todos los caracteres a su entidad HTML

```
<?php

      $str = "A 'quote' is <b>bold</b>";

      // Produce: A 'quote' is &lt;b&gt;bold&lt;/b&gt;

      echo htmlentities($str);

      // Produce: A &#039;quote&#039; is
      &lt;b&gt;bold&lt;/b&gt;

      echo htmlentities($str, ENT_QUOTES);

?>
```

Otras funciones

- **chr(entero):** Recibe el carácter ASCII asociado al entero

```
<?php

      $str = "La cadena termina en un escape: ";

      $str .= chr(27); /* añade un carácter de escape al final
      de  $str */

      /* A menudo esto es más útil */

      $str = sprintf("TLa cadena termina en un escape: %c",
      27);

?>
```

- **count_chars(cad [, modo]):** Cuenta el número de apariciones de cada carácter dentro de una cadena. La manera que devuelve la información depende de un parámetro opcional cuyos valores son:

- 0: Devuelve una ,matriz asociativa con el valor del carácter como clave y su frecuencia como valor
- 1: Como el 0 pero sólo los caracteres que aparecen alguna vez
- 2: Como el 0 pero solo los caracteres que no aparecen
- 3: devuelve una cadena que contiene los caracteres utilizados
- 4: Devuelve una cadena que contiene los caracteres NO utilizados

```
<?php

	$data = "Two Ts and one F.";

	foreach (count_chars($data, 1) as $i => $val) {

		echo "Se ha encontrado $val instancia (s) de \"" , chr($i) , "\" en la cadena.\n";

	}

?>
```

- **strrev(cad):** Da la vuelta a una cadena

```
<?php

	echo strrev("Hello world!"); // outputs "!dlrow olleH"

?>
```

- **str_repeat(cad, veces):** devuelve cad repetidas tantas veces como indique veces

```php
<?php

	echo str_repeat("-=", 10);

?>
```