Introducción a la Programación con PHP

Práctica Nº 4

Información extraída de:

http://ar.php.net/manual/es/langref.php

Tipos de variables

Cuatro tipos escalares:

- boolean (verdadero/falso, sí/no, 1/0)
- integer (número entero)
- float (número en coma-flotante)
- string (cadena de texto)

Dos tipos compuestos:

- array (matriz de valores)
- object (objetos)

Y finalmente dos tipos especiales:

- resource (variable especial, que contiene una referencia a un recurso externo). Ejemplos: las referencias a archivos y a imágenes.
- NULL (variable no tiene valor)

Constantes

Una constante es un identificador para expresar un valor simple. Como el nombre sugiere, este valor no puede variar durante la ejecución del script.

Ejemplo:

```
<?php
 define ("saluda", "Buenas tardes.");
 echo saluda;
?>
```

PHP dispone de varias constantes predifinidas, ejemplo **PHP_VERSION**.

Constantes

Se puede definir una constante declarándola con **const** desde PHP 5.3.0.

Una vez que la constante está definida, no puede ser cambiada o redefinida en ningún momento.

Ejemplo:

```
<?php
// Funciona desde PHP 5.3.0
const CONSTANT = 'Hola Mundo';

echo CONSTANT;
?>
```

Operadores y Expresiones

Expresión: secuencia de operaciones y operandos que especifica un cálculo.

Tres tipos de operadores:

1. **Operador unario:** el cual opera sobre un único valor. Por ejemplo ! (el operador de negación) o ++ (el operador de incremento).
2. **Operadores binarios:** este grupo contiene la mayoría de operadores que soporta PHP. Por ejemplo: +
3. **Operador ternario: ?:** Por ejemplo: $a == 1 ? $a++ :$a*3;

Funciones

Una función es un conjunto de código agrupado con la intención de ejecutarse varias veces, o con distintos parámetros o variables.

Ejemplo:

```
<?php
  function prueba($arg1, $arg2) {
 echo "Función de ejemplo.\n";
 echo "Argumento 1 vale: $arg1<br>“;
 echo "Argumento 2 vale: $arg2<br>";
  if ($arg1 < $arg2){
 return true;
  }
else {
 return false;
    }
 }
?>
```

Puede recibir como parámetros dos variables numéricas

La sentencia especial **return** sirve para devolver valores después de la ejecución de la función desde el lugar donde fue invocada.

En nuestro ejemplo devuelve **true** *verdadero* si argumento1 es menor que argumento2 y falso en el resto de los casos.

Funciones

Para ejecutar el código de una función simplemente ponemos su nombre y entre paréntesis los parámetros que necesita

```
<?php
include ("prueba.php"); //para incluir prueba.php
 $valor_devuelto = prueba(5,10);
if($valor_devuelto){
  echo "Argumento 1 es menor que Argumento 2";
 }
else{
echo "Argumento 1 es mayor o igual que Argumento 2";
 }
?>
```

Funciones

Si en una función queremos devolver un valor el modo de hacerlo es mediante la función **return(parámetro)**, donde parámetro puede ser simplemente un "1" para indicar que todo fue correctamente, un "0" para indicar un error o una variable donde devolvamos el valor de lo procesado por la función.

Ejemplo:

```
<?php
 $valor = funcion();
?>
```

donde *$valor* es el resultado devuelto por la función o bien pasando un parámetro o una variable a la función:

```
<?php
 $valor = funcion($variable);
?>
```

Funciones definidas por el usuario

Una función se puede definir con la siguiente sintaxis:

```
<?php
function foo ($arg_1, $arg_2, ..., $arg_n)
{
    echo "Funcion de ejemplo.\n";
    return $retval;
}
?>
```

Funciones definidas por el usuario

La información puede suministrarse a las funciones mediante la lista de parámetros (una lista de variables y/o constantes separadas por comas).

PHP soporta pasar:

- **Parámetros por valor** (el comportamiento por defecto),
- **Parámetros por referencia,**
- **Parámetros por defecto** (debe ser una constante).

Funciones definidas por el usuario

Los parámetros se pasan por valor

El paso de parámetros en PHP se realiza habitualmente por valor.

"Por valor" es una manera típica de pasar parámetros en funciones, quiere decir que el cambio de un dato de un parámetro no actualiza el dato de la variable que se pasó a la función.

Por ejemplo, cuando invocamos una función pasando una variable como parámetro, a pesar de que cambiemos el valor del parámetro dentro de la función, la variable original no se ve afectada por ese cambio.

```
function porvalor ($parametro1){
$parametro1="hola";
echo "<br>" . $parametro1; //imprime "hola"
}

$mivariable = "esto no cambia";
porvalor ($mivariable);
echo "<br>" . $mivariable; //imprime "esto no cambia"
```

Esta página tendrá como resultado: hola

esto no cambia

Funciones definidas por el usuario

Paso de parámetros por referencia

El cambio del valor de un parámetro dentro de una función sí afecta al valor de la variable original.

Podemos pasar los parámetros por referencia si, en la declaración de la función, colocamos un *"&"* antes del parámetro.

```
<?
function porreferencia(&$cadena)
{
$cadena = 'Si cambia';
}
$str = 'Esto es una cadena';
porreferencia ($str);
echo $str; // Imprime 'Si cambia'
?>
```

Este script mostrará por pantalla 'Si cambia'.

Funciones definidas por el usuario

Parámetros por defecto

Podemos definir valores por defecto para los parámetros. Los valores por defecto sirven para que los parámetros contengan un dato predefinido, con el que se inicializarán si no se le pasa ningún valor en la llamada de la función. Los valores por defecto se definen asignando un dato al parámetro al declararlo en la función.

```
function pordefecto ($parametro1="pepe";$parametro2=3)
```

Para la definición de función anterior, $parametro1 tiene como valor por defecto "pepe", mientras que $parametro2 tiene 3 como valor por defecto.

Si llamamos a la función sin indicar valores a los parámetros, estos tomarán los valores asignados por defecto:

```
pordefecto () // $parametro1 vale "pepe" y $parametro2 vale 3
```

Si llamamos a la función indicando un valor, este será tenido en cuenta para el primer parámetro.

```
pordefecto ("hola") // $parametro1 vale "hola" y $parametro2 vale 3
```

Funciones definidas por el usuario

Cuando se usan parámetros por defecto, estos tienen que estar a la derecha de cualquier parámetro sin valor por defecto; de otra manera no funcionarán de la forma esperada.

Uso incorrecto de parámetros por defecto en funciones

```
<?php
function makeyogurt ($type = "acidophilus", $flavour)
{
    return "Haciendo un bol de $type $flavour.\n";
}
echo makeyogurt ("mora");   // No funciona de la manera
//esperada
?>
```

La salida del ejemplo anterior es:

Warning: Missing argument 2 in call to makeyogurt() in /usr/local/etc/httpd/htdocs/php3test/functest.html on line 41

Haciendo un bol de mora.

Uso correcto de parámetros por defecto en funciones

```
<?php
function makeyogurt ($flavour, $type = "acidophilus")
{
    return "Haciendo un bol de $type $flavour.\n";
}
echo makeyogurt ("mora"); // funciona como se esperaba
?>
```

La salida de este ejemplo es:

Haciendo un bol de acidophilus mora.

Funciones

PHP dispone de funciones que sirven para averiguar en cada momento si una variable es nula o que tipo de dato contiene

gettype($mivar);

is_int($entero) ;

Ejemplo:

```
<?php
 $a = TRUE;
$b = gettype($a);
echo $b;?>
```

simulador
www.phpya.com.ar/simulador/simulador.php?cod=24

Funciones

isset

Determinar si una variable está definida.

Si una variable ha sido removida con unset(), ya no estará definida.

Devolverá **FALSE** si prueba una variable que ha sido definida como **NULL**.

Si son pasados varios parámetros, entonces **isset()** devolverá **TRUE** únicamente si todos los parámetros están definidos.

isset() sólo trabaja con variables, ya que pasar cualquier otra cosa resultará en un error de intérprete.

Estructuras de Control

Todo script PHP se compone de una serie de sentencias.

Una sentencia puede ser una asignación, una llamada a función, un bucle, una sentencia condicional e incluso una sentencia que no haga nada (una sentencia vacía).

Las sentencias normalmente acaban con punto y coma.

Las sentencias se pueden agrupar en grupos de sentencias encapsulando un grupo de sentencias con llaves.

Un grupo de sentencias es también una sentencia.

Estructuras de Control

if

La construcción *if* es una de las más importantes características de muchos lenguajes, incluido PHP.

Permite la ejecución condicional de fragmentos de código.

Estructuras de Control

else

A menudo queremos ejecutar una sentencia si se cumple una cierta condición, y una sentencia distinta si la condición no se cumple.

else extiende una sentencia *if* para ejecutar una sentencia en caso de que la expresión en la sentencia *if* se evalúe como **FALSE**.

Estructuras de Control

elseif

Combinación de *if* y *else*.

Como *else*, extiende una sentencia *if* para ejecutar una sentencia diferente en caso de que la expresión *if* original se evalúa como **FALSE**. No obstante, a diferencia de *else*, ejecutará esa expresión alternativa solamente si la expresión condicional *elseif* se evalúa como **TRUE**.

Por ejemplo, el siguiente código mostraría a es mayor que b, a es igual a b o a es menor que b:

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

Estructuras de Control

while

while (expr) sentencia

do..while

Los bucles do..while son muy similares a los bucles while, excepto que las condiciones se comprueban al final de cada iteración en vez de al principio.

for

for (expr1; expr2; expr3) sentencia

Estructuras de Control

foreach

PHP 4 (PHP3 no) incluye una construcción *foreach*.

Esto simplemente da un modo fácil de iterar sobre matrices.

foreach funciona solamente con matrices y devolverá un error si se intenta utilizar con otro tipo de datos ó variables no inicializadas.

- foreach(expresion_array as $value) sentencia

Recorre el array dado por *expresion_array*. En cada iteración, el valor del elemento actual se asigna a *$value* y el puntero interno del array se avanza en una unidad

Estructuras de Control

foreach

- foreach(expresion_array as $key => $value) sentencia

El índice del elemento actual será asignada a la variable *$key* en cada iteración.

break

Escapa de la estructuras de control iterante (bucle) actuales for, while, o switch.

Ejemplos

```
<html>
<head></head>
<body>
<?php

$amigos=array("carlos","gustavo","mariana","karla","pepe",
    "manolo");
//Este es accediendo a las claves y valores del array
foreach ($amigos as $numero=>$nombre) {
  echo "El numero ".$numero." es ".$nombre."<br>\n";}
?>
</body>
</html>
```

Ejemplos

```
<html>
<head></head>
<body>
<?php
 $amigos=array("carlos","gustavo","mariana","karla","pepe","manolo");
foreach ($amigos as $nombre) {
echo "Hola ".$nombre."<br>\n";
}
?>
</body>
</html>
```

Estructuras de Control

continue

Se usa dentro de la estructura del bucle para saltar el resto de la iteración actual del bucle y continuar la ejecución al comienzo de la siguiente iteración.

switch

La sentencia *switch* es similar a una serie de sentencias IF en la misma expresión. En muchas ocasiones, se quiere comparar la misma variable (o expresión) con muchos valores diferentes, y ejecutar una parte de código distinta dependiendo de a qué valor es igual, para ello sirve la sentencia *switch*

Ejemplo del uso del continue y del break

```php
<?php
    $stack = array('first', 'second', 'third', 'fourth', 'fifth');

    foreach($stack AS $v){
        if($v == 'second')continue;
        if($v == 'fourth')break;
        echo $v.'<br>';
    }
    /*
    first
    third
    */

    $stack2 = array('one'=>'first', 'two'=>'second', 'three'=>'third', 'four'=>'fourth',
    'five'=>'fifth');
    foreach($stack2 AS $k=>$v){
        if($v == 'second')continue;
        if($k == 'three')continue;
        if($v == 'fifth')break;
        echo $k.' ::: '.$v.'<br>';
    }
    /*
    one ::: first
    four ::: fourth
    */

    ?>
```

Estructuras de Control

require()

Incluye y evalúa el archivo especificado.

include()

Incluye y evalúa el archivo especificado.

require() e **include()** son idénticas en todos los aspectos excepto en el modo de actuar ante un error.

include() produce un **Warning** mientras que **require()** produce un **Error Fatal**.

Utilizar **require()** si quiere por ejemplo que un fichero no encontrado cuelgue el procesamiento de la página. **include()** no se comporta de esta manera, el script seguirá funcionando de todas maneras.

Sentencias del Lenguaje

Print

Muestra una cadena

int **print** (string $cadena) // Muestra el valor de *cadena* por la salida
Siempre devuelve el valor *1*.

No es realmente una función (es una **sentencia del lenguaje**) de modo que no se requiere el uso de los paréntesis.

Sentencias del Lenguaje

Ejemplos

```
<?php
print("Hola mundo");
print "print() tambien funciona sin
   parentesis.";
$saludo = "que tal";
print "hola, $saludo";
?>
```

Formularios con PHP

Los Formularios no forman parte de PHP, sino del lenguaje estándar de Internet, HTML.

Todo formulario comienza con la etiqueta

```
<FORM ACTION="lo_que_sea.php" METHOD="post/get">
```

ACTION: indica al script que va a procesar la información que recogemos en el formulario.

METHOD: indica si el usuario del formulario va ha enviar datos (post) o recogerlos (get).

La etiqueta </FORM> indica el final del formulario.

Algunas Variables características de PHP

$_SERVER['PHP_SELF']

Nombre del script que se está ejecutando, es decir devuelve la ruta absoluta al script php en ejecución

$_GET

–Se usa para recoger los valores de un formulario con method = "get".

$_POST

–Se utiliza para recoger los valores de un formulario con method = "post

Algunas diferencias…

Método GET:

Las variables se envían en la url de llamada a la siguiente página.

Al probar una página con un formulario que llama a otra usando el método **get** se observa que arriba en la barra de direcciones aparecen todos los valores al darle click a enviar.

Esto tiene dos desventajas muy grandes:

1º) Que el número de caracteres es limitado

2º) Se puede ver todo lo que se envía y eso no siempre es recomendable.

Método Post:

Los valores no se ven por ninguna parte y no tiene ningún límite en la cantidad a enviar.

Pero en ciertos sistemas puede tenerse problemas con este método.

Gestión básica de archivos con PHP

Funciones para la lectura de archivos

fopen: Abre un archivo y le asigna un identificador id.

$id = fopen($archivo, $modo)

fgets: Lee una línea de un archivo hasta un número máximo de caracteres

fgets($id,$max)

fwrite: Escribe una cadena dentro del archivo

fwrite($id, $cadena)

fseek: Avanza o retrocede el puntero del archivo un cierto número de posiciones

fseek($id,$posiciones)

feof: Comprueba si el puntero que lee el archivo ha llegado al final

feof($id)

fpassthru: Lee completamente el archivo y lo muestra

fpassthru($id)

Gestión básica de archivos con PHP

fclose: Cierra el archivo abierto previamente

```
fclose($id)
```

Modos de apertura de archivos

'r': Sólo lectura

'r+': Lectura y escritura

'w': Sólo escritura

'w+': Lectura y escritura. Suprime el contenido anterior si se escribe. El archivo es creado si no existe.

'a': Sólo escritura. El archivo es creado si no existe y el puntero se coloca al final.

'a+': Lectura y escritura. El archivo es creado si no existe y el puntero se coloca al final.

Matrices, Arrays o Vectores

Una **matriz** es un tipo de variable que permitirá almacenar **múltiples valores en una única variable**.

El término *array* inglés significa *cadena*, y es comúnmente utilizado en los lenguajes de programación al referirnos a vectores o secuencias de datos.

Nos referiremos a matrices con el término *array* ya que supone una extrapolación a varias dimensiones de un vector.

La sintaxis para construir un array es la siguiente:
Ejemplo:
$invierno = array ("Enero", "Febrero", "Marzo");

Este array asigna un número a cada elemento de forma automática, comenzando por el cero.

Matrices, Arrays o Vectores

Hay que diferenciar entre los dos tipos de matrices existentes:

- INDEXADA: aquella cuyo acceso a los elementos se realiza por la posición que ocupan dentro de la estructura (se inician siempre desde la posición 0).

```
<?php
$invierno = array ("Enero", "Febrero", "Marzo");
echo ("Uno de ".$invierno[0].", dos de ".$invierno[1].", tres de ".$invierno[2]."...");
?>
```

- ASOCIATIVA: es aquella en la que los elementos están formados por partes clave-valor y el acceso se realiza proporcionando una determinada clave.

```
$ficha = array(
  nombre=>"Gonso",
direccion=>"Alamillos",
telefono=>"10494676",
 edad=>"24“
 );
```

La forma de acceder a este array será a través de las claves que hemos definido. Por ejemplo, acceder al nombre sería: *$ficha[nombre]*.

Matrices, Arrays o Vectores

Los **arrays multidimensionales** nos permitirán construir arrays construidos por otros arrays.

Supongamos que queremos hacer una agenda, definiríamos el array $agenda de la forma habitual: $agenda = array();
y rellenamos los datos de forma referenciada por *claves*:

```
$agenda = array(
 array( nombre=>"Gonso",
        direccion=>"Alamillos",
        telefono=>"10494676",
        edad=>"24"),
 array( nombre=>"Peggy",
       direccion=>"Don Sancho",
        telefono=>"10494665",
  edad=>"22"),
  array( nombre=>"Cremy",
    direccion=>"Don Sancho",
     telefono=>"1214665",
   edad=>"25")
  );
```

Para extraer la información de los arrays multidimensionales llamaremos al array $agenda con el número que corresonda al array que queremos referenciar y la clave que queremos.
Por ejemplo, si queremos obtener el teléfono de *Peggy* lo haríamos de la siguiente manera:
echo $agenda[1][telefono];

Funciones

getdate : Obtiene información de fecha/hora

array **getdate** ([int $marca_de_tiempo])

Devuelve un valor array asociativo que contiene información de fecha sobre la *marca_de_tiempo* , o la hora local actual si no se entrega una *marca_de_tiempo*

“seconds“:	Representación numérica de segundos*0* a *59*
"minutes“:	Representación numérica de minutos*0* a *59*
"hours“:	Representación numérica de horas*0* a *23*
"mday“:	Representación numérica del día del mes*1* a *31*
"wday“:	Representación numérica del día de la semana*0* (para el Domingo) a *6* (para el Sábado)
“mon“:	Representación numérica de un mes*1* a *12*
"year“:	Una representación numérica completa de un año, 4 dígitos Ejemplos: *1999* o *2003*

Funciones

strlen

Obtiene la longitud de la cadena

int **strlen** (string $cadena)

strpos

Encuentra la posición de la primera aparición de una cadena

int **strpos** (string $cadena , string $caracter [, int $desplazamiento])

Devuelve la posición numérica de la primera aparición del *caracter* en la cadena *cadena* .

A diferencia de strrpos(), esta función puede tomar una cadena completa como *caracter* y se utilizará en su totalidad.

Si no se encuentra el *caracter* , devuelve **FALSE**.

Funciones

strpos

Ejemplo:

```
<?php
    // Se puede buscar por el caracter, ignorando cualquier
    cosa antes del offset
    $newstring = 'abcdef abcdef';
    $pos = strpos($newstring, 'a', 1); // $pos = 7, no 0
    ?>
```

Si el *caracter* no es una cadena, se convierte a entero y se aplica como el valor ordinal de un carácter.
El parámetro opcional *desplazamiento* le permite especificar a partir de qué carácter de la *cadena* se empieza a buscar. La posición devuelta sigue siendo relativa al comienzo de *cadena*.

Funciones

Otro Ejemplo:

```
<?php
    $mi_cadena = 'abc';
    $caracter   = 'a';
    $posicion = strpos($mi_cadena, $caracter);

    // Seguidamente se utiliza ===.  La forma simple de comparacion (==)
    // no funciona como debería, ya que la posición de 'a' es el caracter
    // numero 0 (cero)
    if ($posicion === false) {
        echo "No se encontro '$caracter' en la cadena '$mi_cadena'";
    } else {
        echo "Se encontro '$caracter' en la cadena '$mi_cadena'";
        echo " en la posicion $posicion";
    }

    // Se puede buscar el caracter sin tener en cuenta los caracteres anteriores
    // al desplazamiento
    $nueva_cadena = 'abcdef abcdef';
    $posicion = strpos($nueva_cadena, 'a', 1); // $posicion = 7, no 0
    ?>
```

Funciones

substr

Devuelve parte de una cadena

string **substr** (string $cadena , int $comienzo [, int $longitud])

substr() devuelve la porción de *cadena* especificada por los parámetros *comienzo* y *longitud* .

Si *comienzo* es positivo o 0, la cadena devuelta comenzará en dicho carácter de *cadena* (los caracteres empiezan a contarse en cero).

Por ejemplo, en la cadena '*abcdef*', el carácter en la posición *0* es '*a*', el carácter en la posición *2* es '*c*', y así sucesivamente.

Funciones

```
<?php
    echo substr('abcdef', 1);     // bcdef
    echo substr('abcdef', 1, 3);  // bcd
    echo substr('abcdef', 0, 4);  // abcd
    echo substr('abcdef', 0, 8);  // abcdef
    echo substr('abcdef', -1, 1); // f

    // El acceso a los caracteres dentro de una cadena se
    //puede realizar directamente mediante las llaves
    $string = 'abcdef';
    echo $string{0};                 // a
    echo $string{3};                 // d
    echo $string{strlen($string)-1}; // f

    ?>
```

Funciones

Si comienzo es negativo, la cadena devuelta comenzará en dicha posición contando desde el **final de cadena**.

Ejemplo de valores negativos de *comienzo*

```
<?php
   $rest = substr("abcdef", -1);    // devuelve "f"
   $rest = substr("abcdef", -2);    // devuelve "ef"
   $rest = substr("abcdef", -3, 1); // devuelve "d"
   ?>
```