“Fundamentos de la programación estructurada”

Conceptos Iniciales

- Programa
  - Un programa informático o programa de computadora es una secuencia de instrucciones, escritas para realizar una tarea específica en una computadora.
- Algoritmo
  - En matemáticas, lógica, ciencias de la computación y disciplinas relacionadas, un algoritmo es un conjunto de instrucciones o reglas definidas y no-ambiguas, ordenadas y finitas que permite, típicamente, solucionar un problema, realizar un cómputo, procesar datos y llevar a cabo otras tareas o actividades
- Sentencias de Código
  - Las sentencias son los elementos básicos en los que se divide el código en un lenguaje de programación

Sintaxis

- Todo encerrado entre las etiquetas
  - <?php ..... ?>
  - <? ..... ?>
- Case sensitive
  - $Cantidad
  - $cantidad
  - $cAntidad
- No tipado
- No afectado por espacios en blanco.
- Líneas terminadas en ;

Elementos del Lenguaje

- Comentarios
  - /* esto es un comentario */
  - // este es otro comentario
  - /*
    - Comentario
  - */

Estructura del Lenguaje

- Variables
  - Inician con $ siempre
  - 2do carácter no debe ser un número
  - Formado con letras, números, guión bajo
  - Sin espacios en blanco, guión medio, ni otros símbolos, acentos, ñ.

```
<?php
        $nombre = ‘Juan’;
        $apellido = “Sanchez”;
?>
```

Elementos del Lenguaje

- Constantes.

bool define (string $name , mixed $value [, bool $case_insensitive = false])
Devuelve TRUE en caso de éxito o FALSE en caso de error.

```
<?php
        define("CONSTANTE", "Hola mundo.");
      echo CONSTANTE; // imprime "Hola mundo."
      echo Constante; // imprime "Constante" y emite un aviso.

      define("SALUDO", "Hola tú.", true);
      echo SALUDO; // imprime "Hola tú."
      echo Saludo; // imprime "Hola tú."

      // Funciona a partir de PHP 7
     define('ANIMALES', array(
                                        'perro',
                                        'gato',
                                        'pájaro'
                                      ));
     echo ANIMALES[1]; // muestra "gato"

?>
```

Elementos del Lenguaje

- Impresión de datos
  - echo
  - print
- Comillas
  - Se puede usar comillas simples o dobles.
  - La diferencia es la siguiente

```
<?php
        $var = ‘mundo’;
         echo ‘Hola $var’;  // muestra Hola $var
         echo “Hola $var”; // muestra Hola mundo
         echo “Hola”.$var; // muestra Hola mundo
```

Operadores

- Asignación
  - $var = 123;
- Concatenación
  - $a = ‘Ale’;
  - $b = ‘jando’;
  - $c = $a.$b;
  - echo $c; // muestra Alejandro
- O. Aritméticos
  - Suma +
  - Resta -
  - División /
  - Multiplicación *
  - Modulo %

Operadores

- O. Comparación
  - Igual ==
  - Mayor >
  - Menor <
  - Mayor o igual >=
  - Menor o igual <=
  - Distinto !=

```
<?php
        ‘1’  == 1    // compara contenido
        ‘1’  === 1   // compara contenido y tipo
```

Operadores

O. Lógicos

Ejemplo	Nombre	Resultado
$a and $b	And (y)	**TRUE** si tanto *$a* como *$b* son **TRUE**.
$a or $b	Or (o inclusivo)	**TRUE** si cualquiera de *$a* o *$b* es **TRUE**.
$a xor $b	Xor (o exclusivo)	**TRUE** si *$a* o *$b* es **TRUE**, pero no ambos.
! $a	Not (no)	**TRUE** si *$a* no es **TRUE**.
$a && $b	And (y)	**TRUE** si tanto *$a* como *$b* son **TRUE**.
$a || $b	Or (o inclusivo)	**TRUE** si cualquiera de *$a* o *$b* es **TRUE**.

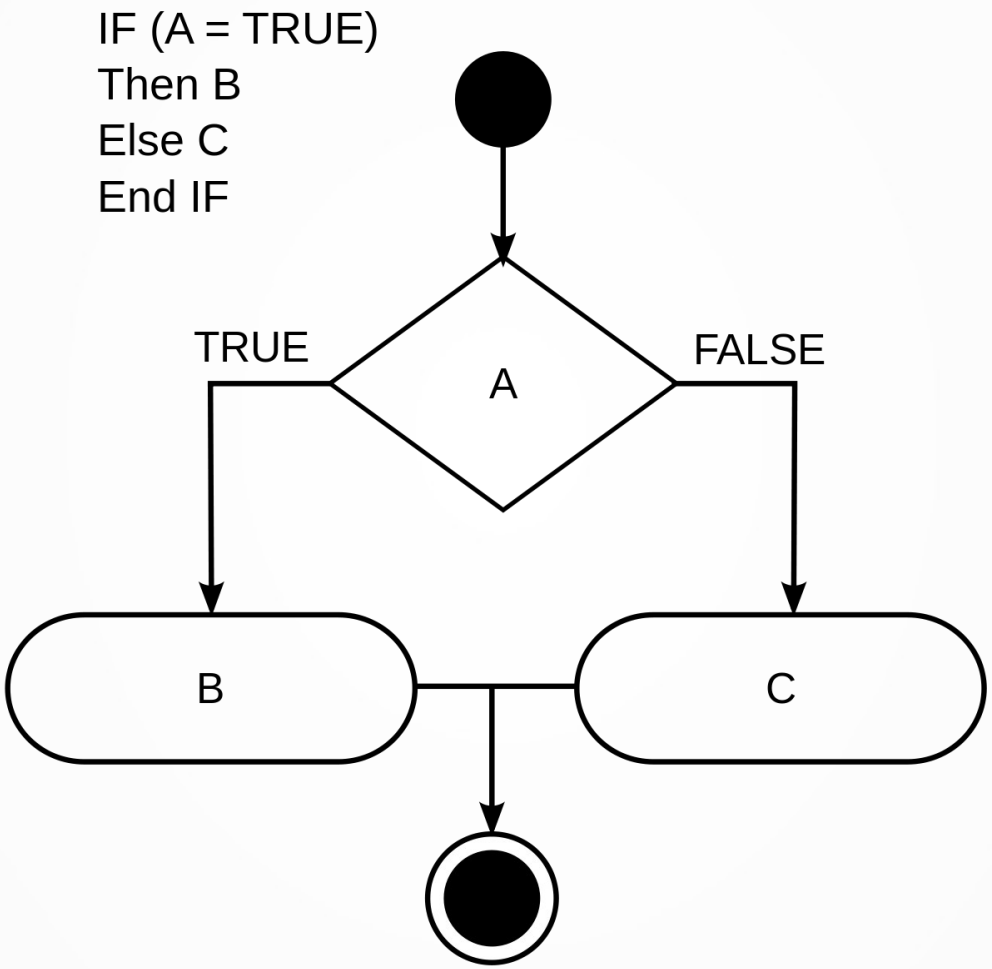
La razón para tener las dos variaciones diferentes de los operadores "and" y "or" es que ellos operan con precedencias diferentes.

Operadores

Precedencia (1 – 10)

1	()	Agrupar
2	!, ++, --	Negación, incemento y decremento.
3	*, / y %	Multiplicación, división y módulo.
4	+ y -	Adición y sustracción.
5	>, >=, <, <=	Comparaciones
6	== y !=	Igualdad y desigualdad
7	&&	AND
8	||	OR
9	?:	Operador ternario
10	=, +=, -=	Asignacion

Estructura de control.



Estructura de control.

if

```
<?php
    if ($a > $b) {
        echo "a es mayor que b";
    }
?>
```

else

```
<?php
    if ($a > $b) {
        echo "a es mayor que b";
    } else {
        echo "a NO es mayor que b";
    }
?>
```

Estructura de control.

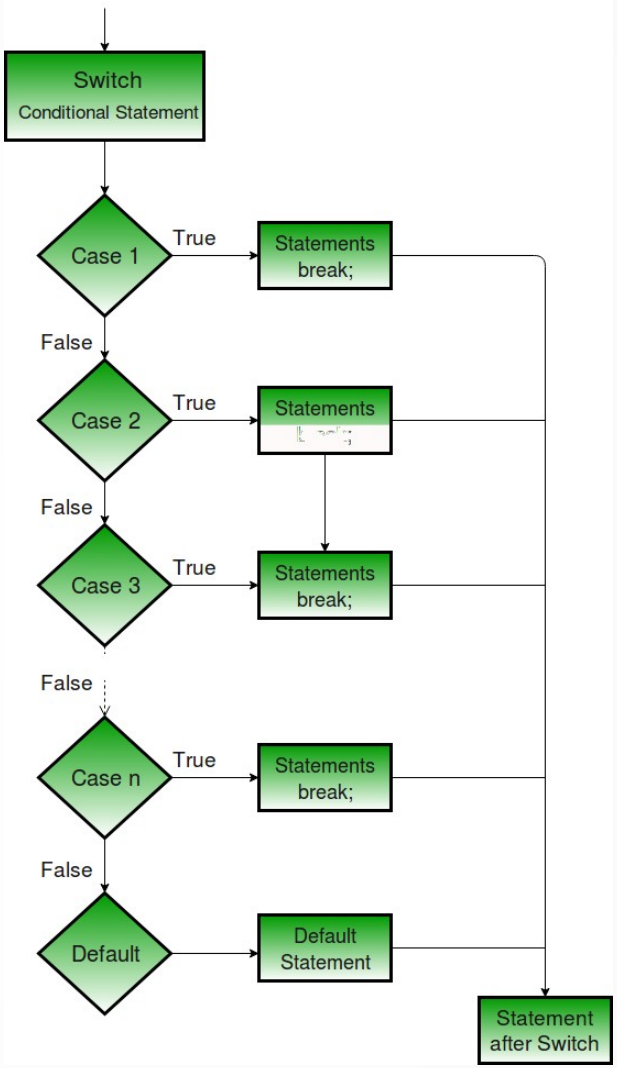
elseif/else if

```
<?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";
    }
?>
```

Estructura de control.

switch

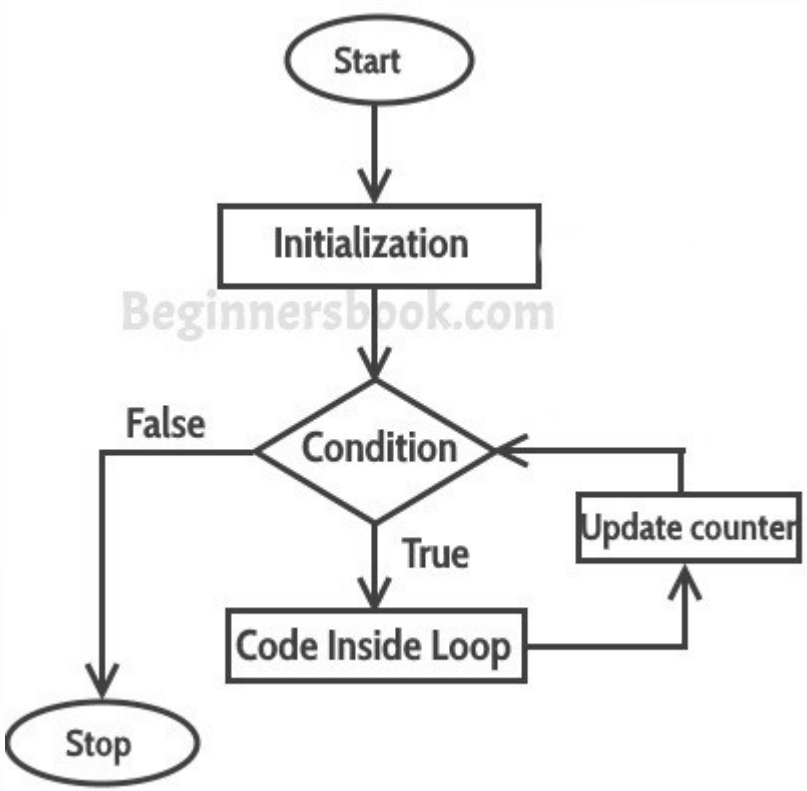
```
switch ($i) {
    case 0:
        echo "i es igual a 0";
        break;
    case 1:
        echo "i es igual a 1";
        break;
    case 2:
        echo "i es igual a 2";
        break;
    }
```



Estructura repetitivas.

for

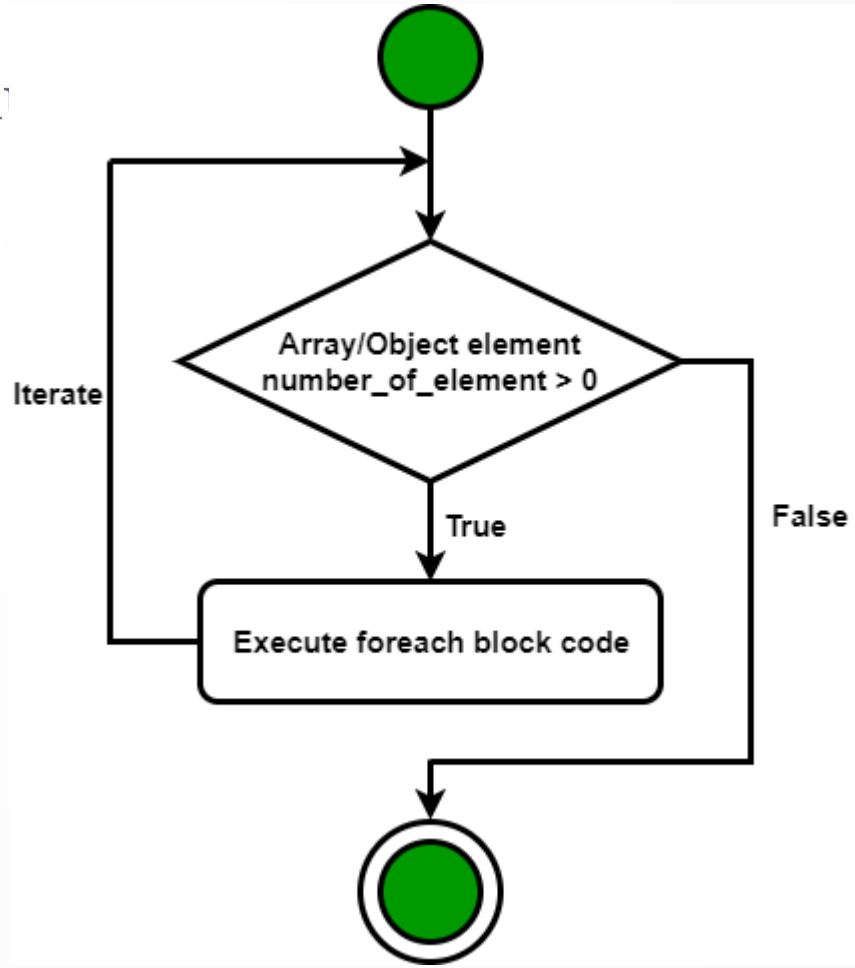
```
for($i=0; $i<20; $i++)
   {
       echo $i."<br/>";
    }
```



Estructura repetitivas.

foreach

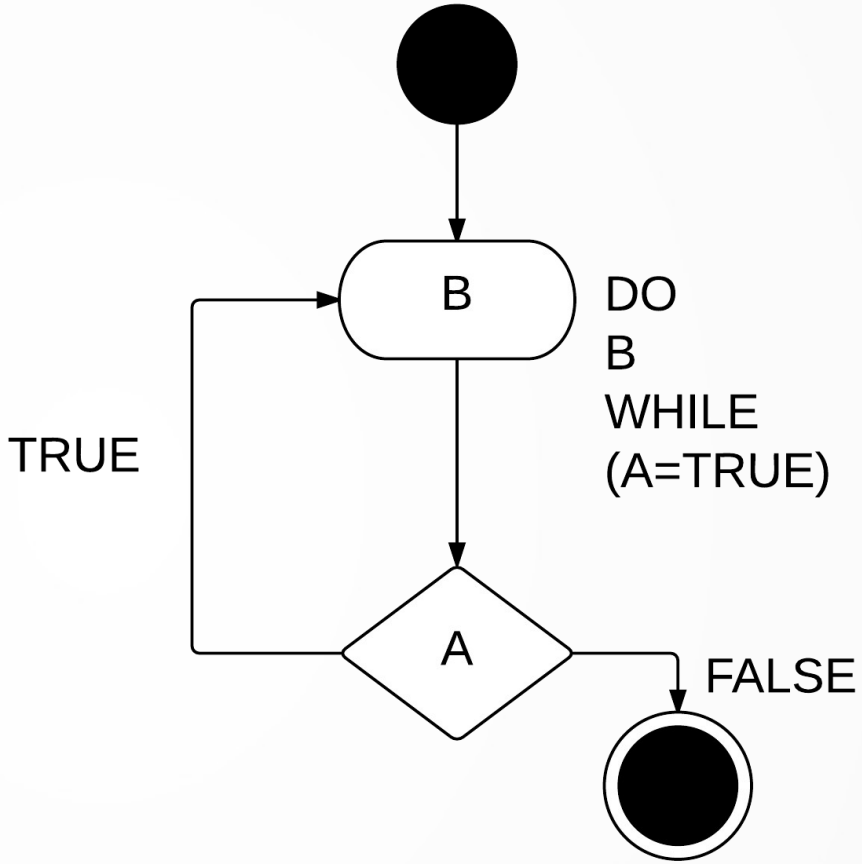
```
foreach( array as $key=>$val
   {
      ...
    }
```



Estructura repetitivas.

Do .. while

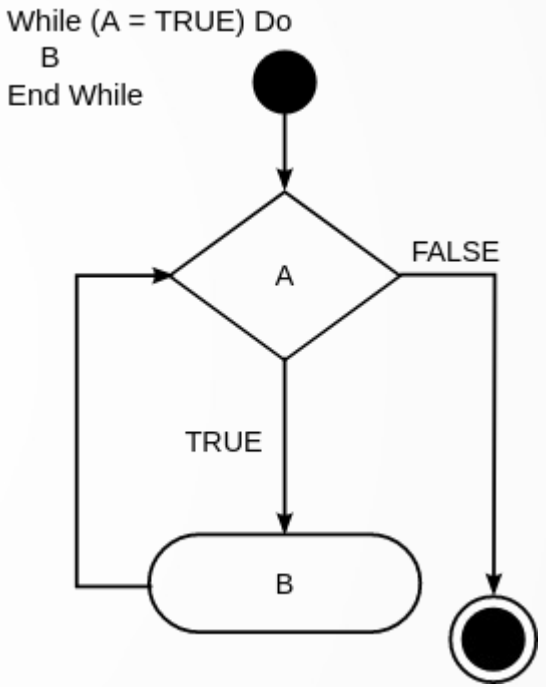
```
do
{
    $b++;
    echo $b."<br/>";
}
 while($b < 10);
```



Estructura repetitivas.

While

```
while($cortar)
 {
      $a++;
      if($a == 10)
      {
          $cortar=false;
      }
      echo $a."<br/>";
 }
```



Funciones

- Una función es una porción de código que recibe uno o más parámetros y retorna uno o mas valores.

F. con Parámetros

- Una función pude recibir uno o más parámetros los cuales son usados como variables internas de la misma.
- Estos parámetros pueden ser de cualquier tipo.

```
function sumar($num1, $num2)
{
	$sum = $num1 + $num2;
	echo "La suma es : $sum";
}
```

Sumar(10, 20); → esto imprime en la web “La suma es: 30”

F. con parámetros por referencia

- Una función pude recibir los parámetros por referencia.
- Lo cual indica que se envía la referencia de donde se encuentra la variable en memoria, en lugar de copiar el valor en otra variable.
- Por lo tanto cualquier cambio ocurrido dentro de la función también esta presente fuera de la misma.

F. con parámetros por referencia

```
function sumarCuatro($num)
{
	$num += 4;
}

function sumarSeis(&$num)
{
	$num += 6;
}

$valor = 10;

sumarCuatro( $valor );
echo "El valor original es $valor<br />";  → “El valor original es 10”

sumarSeis( $valor );
echo "El valor original es $valor<br />";  → “El valor original es 16”
```

Retorno de valores

- Para que una función retorne un valor se usa la palabra clave return.
- Puede retornar un valor, array u objeto.
- Como se puede ver la sentencia return retorna un valor, por lo cual si se quiere que una función retorne más de un valor se pasan por referencia.

```
<?php
	function sumar($num1, $num2)
	{
		$sum = $num1 + $num2;
		return $sum;
	}

	$retorno = sumar(10, 20);
	echo "Valor retornado: $retorno";
?>
```

Parámetros con valores por defecto.

```
<?php

        function imprimir($param = 'valor por defecto')
        {
                echo $param;
        }

        imprimir("esto es un test");        → “esto es un test”
        Imprimir();                         → “valor por defecto”
?>
```

- Función con un array por defecto

  funtion imprimir($param = array()){...}

Funciones dinámicas

- Es posible guardar el nombre de una función en una variable y llamar a la misma posteriormente.

```
function saludar()
{
	echo "Hola<br />";
}

$funcion = “saludar";
$funcion();

function saludar2($name)
{
	echo "Hola $name<br />";
}

$funcion = "saludar2";
$funcion('jose');
```

Vectores y Matrices

- Es una estructura de datos que almacena 1 o varios datos de tipos similares en una variable sola.
- Hay 3 tipos distintos de array.

  - Array Numéricos: Array con índices numéricos
  - Array Asociativos: Array con índices alfanuméricos
  - Array Multidimensionales: Un array que contiene 1 o mas arrays.

Array Numéricos

- Estos pueden almacenar números, cadenas y objetos.
- Sus índices son numéricos
- Por defecto se inicia desde el 0

```
/* First method to create array. */
$numbers = array( 1, 2, 3, 4, 5);
foreach( $numbers as $value ){
                        echo "Value is $value <br />";
}
/* Second method to create array. */
$numbers[0] = "one";
$numbers[1] = "two";
$numbers[2] = "three";
$numbers[3] = "four";
$numbers[4] = "five";
foreach( $numbers as $value ){
  echo "Value is $value <br />";
}
```

Array Asociativos

- Son idénticos a los numéricos con la diferencia de que estos usan cadenas como índices

```
$salaries = array(
            "jose" => 2000,
            "mario" => 1000,
            "pedro" => 500
           );
```

Array Asociativos

```
/* First method to associate create array. */
        $salaries = array(
          "jose" => 2000,
          "mario" => 1000,
          "pedro" => 500
         );
echo "Salary of jose is ". $salaries['jose'] . "<br />";
echo "Salary of mario is ".  $salaries['mario']. "<br />";
echo "Salary of pedro is ".  $salaries['pedro']. "<br />";

/* Second method to create array. */
$salaries['jose'] = "high";
$salaries['mario'] = "medium";
$salaries['pedro'] = "low";

echo "Salary of jose is ". $salaries['jose'] . "<br />";
echo "Salary of mario is ".  $salaries['mario']. "<br />";
echo "Salary of pedro is ".  $salaries['pedro']. "<br />";
?
```

Array Asociativos

```
/* First method to associate create array. */
                $salaries = array(
                  "jose" => 2000,
                  "mario" => 1000,
                  "pedro" => 500
                 );
        foreach($salaries as $key=>$value){
                echo "Salary of $key is ".  $value. "<br />";
        }
        /* Second method to create array. */
        $salaries['jose'] = "high";
        $salaries['mario'] = "medium";
        $salaries['pedro'] = "low";
        foreach($salaries as $key=>$value){
                echo "Salary of $key is ".  $value. "<br />";
        }
```

Matrices / Array Multidimensional

```
$marks = array(
        "jose" => array(
                "physics" => 35,
                "maths" => 30,
                "chemistry" => 39
                ),
        "mario" => array(
                "physics" => 30,
                "maths" => 32,
                "chemistry" => 29
        ),
       "pedro" => array(
              "physics" => 31,
              "maths" => 22,
              "chemistry" => 39
        )
      );
```

Array Multidimensional

```
   echo "Marks for jose in physics : " ;
   echo $marks['jose']['physics'] . "<br />";
   echo "Marks for mario in maths : ";
   echo $marks['mario']['maths'] . "<br />";
   echo "Marks for pedro in chemistry : " ;
   echo $marks['pedro']['chemistry'] . "<br />";


foreach($marks as $key => $notas){
       echo 'Notas de '.$key.':<br />';
      foreach($notas as $materia=>$nota){
             echo $materia.'='.$nota.'<br />';
       }
}
```

Errores

- A la hora de desarrollar una aplicación se debe tomar en cuenta los distintos puntos de la aplicación susceptible a errores y debemos actuar ante la posibilidad de que sucedan.
- A continuación tenemos distintas formas de manejar errores.
  - Declaraciones “die()”
  - Errores personalizados y disparadores
  - Errores al reportar

die();

- Esta función es una forma sencilla del manejo de errores puesto que interrumpe la ejecución de un script php.
- Es equivalente al exit().
- Por ejemplo su ejecutamos la siguiente sentencia

```
<?php
    $file=fopen("welcome.txt","r");
?>
```

El php nos retorna

Warning : fopen(welcome.txt) [function.fopen]: failed to open stream: No such file or directory in **/var/www/html/prueba.php** on line **2**

Para evitar que le usuario vea este error se puede utilizar la función die() para mostrar otro mensaje.

die();

```
<?php
if(!file_exists("welcome.txt")) {
	die("File not found");
} else {
	 $file=fopen("welcome.txt","r");
}
?>
```

Este mensaje no es mas explicativo pero termina mostrando una pagina en blanco o incompleta con ese mensaje.

Errores personalizados

- Otro método que podemos utilizar es la creación de controladores errores personalizados, los cuales son funciones definidas por el desarrollador que se ejecutan en el momento de producirse un error en php

- Esta función debe ser capaz de manejar un mínimo de dos parámetros (nivel de error y el mensaje de error), pero puede aceptar hasta cinco parámetros (file, line-number, y the error context)

- Sintaxis

  error_function(error_level,error_message,error_file,error_line,error_context)

Errores personalizados

- Sintaxis

  error_function(error_level,error_message,error_file,error_line,error_context)
- Donde

error_level	Necesario. Especifica el nivel de informe de error para el error definido por el usuario. Debe ser un número de valor. Consulte la tabla siguiente para los posibles niveles de informe de error
error_message	Necesario. Especifica el mensaje de error para el error definido por el usuario
error_file	Opcional. Especifica el nombre de archivo en el que se produjo el error
error_line	Opcional. Especifica el número de línea en el que se produjo el error
error_context	Opcional. Especifica una matriz que contiene todas las variables y sus valores, en uso cuando se produjo el error

Errores personalizados

- Niveles de errores

2	E_WARNING	Errores no fatales de ejecución. La ejecución del script no se detiene
8	E_NOTICE	Avisos en tiempo de ejecución. El script encontró algo que podría ser un error, pero también podría ocurrir cuando se ejecuta un script normalmente
256	E_USER_ERROR	Error fatal generado por el usuario. Esto es como un E_ERROR establecido por el programador utilizando la función de PHP trigger_error()
512	E_USER_WARNING	Error no fatal generado por los usuarios de advertencia. Esto es como un E_WARNING establecido por el programador utilizando la función de PHP trigger_error()
1024	E_USER_NOTICE	Aviso generado por el usuario. Esto es como un E_NOTICE establecido por el programador utilizando la función de PHP trigger_error()
4096	E_RECOVERABLE_ERROR	Error fatal capturable. Esto es como un E_ERROR pero puede ser capturado (see also set_error_handler())
8191	E_ALL	Todos los errores y advertencias (E_STRICT became a part of E_ALL in PHP 5.4)

Errores personalizados

A continuación tenemos un ejemplo de como poner una función definida por nosotros como gestor de errores del php

```
<?php
    //error handler function
    function customError($errno, $errstr) {
        echo "<b>Error:</b> [$errno] $errstr";
    }

    //set error handler
    set_error_handler("customError");

    //trigger error
    echo($test);
?>
```

Desencadenar errores

En tiempo de ejecución podemos hacer que el php dispare el gestor de errores

```
<?php
      //error handler function
      function customError($errno, $errstr) {
       echo "<b>Error:</b> [$errno] $errstr<br>";
       echo "Ending Script";
       die();
       }

       //set error handler
       set_error_handler("customError",E_USER_WARNING);

       //trigger error
      $test=2;
      if ($test>=1) {
          trigger_error("Value must be 1 or below",E_USER_WARNING);
   }
?>
```

Reporte de errores

- Además de avisar por pantalla los errores al usuario, se pueden registrar mensajes en un log, un archivo o enviarlos por mail a un administrador por ejemplo

- Para esto utilizamos a función de php “error_log”, la cual tiene la siguiente sintaxis

error_log(*message,type,destination,headers*);

Reporte de errores

- Donde

Parámetro	Descripción
message	Requerido. Especifica el mensaje de error a reportar.
type	Opcional. Indica donde almacenar el error, los posibles valores son • 0 - Default. El mensaje es enviado al logger del sistema de PHP, dependiendo de como este configurado el error_log en el php.ini • 1 – Mensaje enviado por mail a la dirección indicada en el parámetro *destination*. • 2 – No utilizado actualmente (disponible en PHP 3) • 3 – El mensaje es agregado al archivo indicado en *destination* • 4 - El mensaje es enviado directamente al controlador de logging SAPI. Donde SAPI es la api del servidor que esta ejecutando el PHP, por ejemplo el Apache.
destination	Opcional. Especifica el destino del mensaje de error. Este valor depende del type indicado.
headers	Opcional. Solo usado cuando el tipo es 1. Indica información adicional del header como From, Cc, and Bcc. Se pueden usar múltiples headers separados con CRLF (\r\n)

Reporte de errores

- Ejemplo con envió de mail
  - error_log("¡La base de datos de Oracle no está disponible!",1,
    "someone@example.com","From: webmaster@empresa.com");
  - error_log("¡La base de datos de Oracle no está disponible!", 0);
  - error_log("¡La base de datos de Oracle no está disponible!", 3, "/var/tmp/my-errors.log");

Excepciones

- Por medio del manejo de excepciones podemos controlar el correcto flujo de nuestro código y verificar que todo se ejecute de manera correcta.

- Las excepciones son utilizadas para cambiar el flujo normal de un script si se produce algún error dentro de una condición.

Lanzar Excepciones

```
$edad = -20;

function validAge($num){
      if($num > 0){
            return ‘isValid’;
      }else{
            throw new Exception (‘La edad debe ser mayor a cero’);
      }
}

validAge($edad);
//Devuelve: Fatal error: Uncaught Exception: La edad debe ser mayor a cero
```

De esta manera lanzamos una excepción.

Capturar Excepciones

De la siguiente manera podemos capturar estos errores y hacer que el flujo del script cambie.

```
$edades = array(2,-3,4-3);

foreach($edades as $edad){
	try{
		if(validAge($edad) == ‘isValid’){
			echo ‘Edad ’.$edad.’ es válida’ .’<br />’;
		}
	}
	catch(Exception $e){
		echo ‘Ha habido una excepción: ‘.$e->getMessage().’<br />’;
	}
}
```

Lanzamiento y captura de excepciones

- Las excepciones pueden ser lanzadas con thrown
- Son capturadas con catch, podemos tener tantos catch como excepciones distintas queremos capturar.
- Las excepciones son capturadas siempre que se disparen dentro de un try sin importar que sea dentro de una función que es utilizada por las sentencias dentro del try.
- Podemos utilizar finally para indicar que hacer luego del try, sea o no capturada una excepción.

Clase Exception

- Exception es una clase que tenemos definida en el PHP y posee los siguientes metidos.
    - getMessage() — Obtiene el mensaje de Excepción
    - getPrevious() — Devuelve la excepción anterior
    - getCode() — Obtiene el código de Excepción
    - getFile() — Obtiene el fichero en el que ocurrió la excepción
    - getLine() — Obtiene la línea en donde ocurrió la excepción
    - getTrace() — Obtiene la traza de la pila
    - getTraceAsString() — Obtiene la traza de la pila como una cadena de caracteres
- (Fuente: https://php.net/manual/es/class.exception.php)

Handler personalizados

- Para generar un handler de excepciones simplemente tenemos que crear un clase que extienda a exception.

```
class customException extends Exception {
    public function errorMessage() {
        // Mensaje de error
        $errorMsg = 'Error en la línea '
        .$this->getLine().' en el archivo '
        .$this->getFile() .': <b>'
        .$this->getMessage().
        '</b> no es una edad válida';
        return $errorMsg;
    }
}
```

Excepciones no capturadas

- En ocasiones no necesitamos crear excepciones en todo el script, solamente en los puntos criticas en donde debemos cambiar el flujo si se producen, como ejemplo en accesos a base de datos, lectura de archivos, intercomunicación con otros sistemas, cosas que no controlamos.

- Para capturar estas excepciones debemos definir una función que tenga el comportamiento en el caso de una excepción e indicarle al PHP que la utilice.

```
set_exception_handler('exceptionHandler');
function exceptionHandler($e){
    // Mensaje público
    echo "Ha habido un error";
    // Mensaje semi-escondido
    echo "<!--Excepción sin capturar: " . $e->getMessage() . "--><br>";
}
```