Depurar errores es uno de los pasos más importante de la programación. El sistema de control de errores de PHP ha evolucionado a lo largo de sus versiones. El sistema básico de control de errores incluía la generación de diferentes tipos de errores los cuales son asociados a un número y a una constante predefinida (https://www.php.net/manual/es/errorfunc.constants.php). PHP 5 introdujo un nuevo modelo de errores que te permite lanzar y capturar excepciones en tu aplicación. Y en PHP 7 aparecen las excepciones de tipo Error.

1. Control de errores.

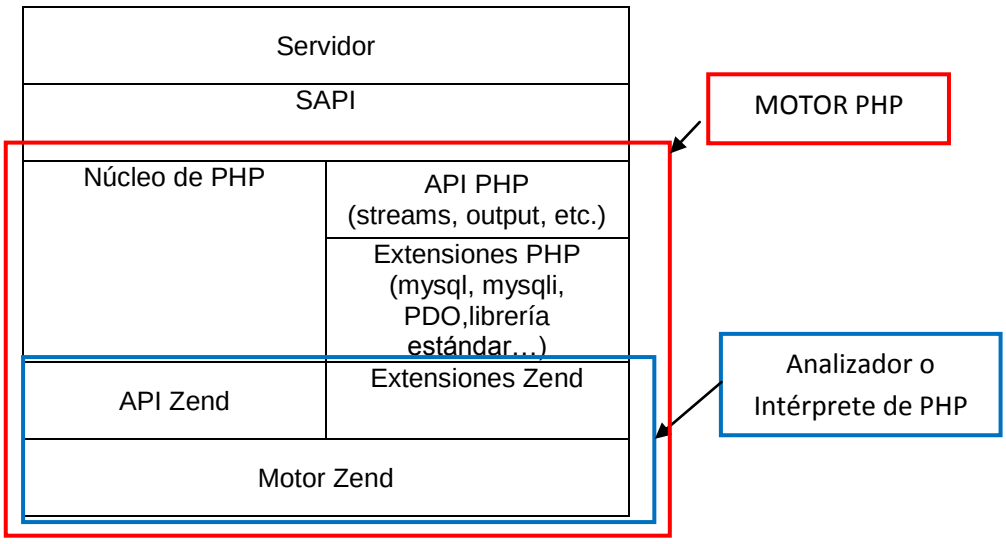
De forma resumida los errores pueden clasificarse de mayor a menor gravedad en:

Errores críticos (Fatal error): si se producen se aborta la ejecución y el programa deja de funcionar por completo (*ej: ejecutar una función que no ha sido definida*).

Errores *parse*: son errores detectados por el analizador del núcleo de PHP; no se lleva a cabo la ejecución de programa. Los más habituales son errores sintácticos. (ej: olvidar $ en una variable, no cerrar un paréntesis,…)

Advertencias (Warning): No provocan la finalización del programa pero pueden ser la causa de que no funcione correctamente o de producirse errores graves en algún momento.*(ej: include "errores.php"; y el fichero que no exista. Genera un warning pero producirá un error grave al intentar utilizarle, dividir por 0,..)*

Notificaciones o avisos (Notice): No ocasionan problemas serios en la ejecución aunque pueden hacer que el programa no funcione como se esperaba. (*ej: querer utilizar una variable sin ser definida).*



En comportamiento de PHP frente a los errores se controla a través de ciertas **directivas del fichero ini.php**:

- *error_reporting:* indica qué tipo de errores se notifican. Lo habitual es *error_reporting* = E_ALL; es decir, todos los errores o *error_reporting*= E_ALL &~E_NOTICE. (es decir, todos menos los avisos)
- *display_error:* por defecto está activada; hace que los mensajes aparezcan en la salida estándar y por lo tanto se ven en el navegador. Generalmente, se desactiva en servidor en producción por seguridad.
- *log_errors:* si se desactiva display_error en un servidor de producción, se debe tener activa está otra directiva con el fin de que los errores después se almacenen en un fichero.
- *error_log:* con esta directiva se establece la ruta del archivo en el cual se registran los errores.
- https://www.php.net/manual/es/errorfunc.configuration.php obtendrás información sobre otras muchas directivas asociadas al control de errores.

Actividad 1: Consulta el fichero php.ini y localiza las directivas comentadas.

Existen diversas funciones que pueden ser utilizadas por el programador para el control de errores:

- *error_reporting()*: permite establecer qué tipo de errores se desean controlar; con ella se modifica el valor de la directiva *error_reporting.*
- *set_error_handler()*: permite al programador definir y con ello reemplazar la gestión de errores producidos en tiempo de ejecución establecida en PHP, es decir, por el gestor de errores interno. Será necesario que defina una función callback, la cual requiere de forma obligatoria, dos parámetros, que serán el nivel del error (constante preestablecida en PHP) y el mensaje a mostrar cuando se produzca. Los parámetros a esta función son proporcionados por el intérprete de PHP. El nombre de la función será el parámetro de *set_error_handler()* y así se informará a PHP de que debe ejecutarla cuando encuentre un error y no seguir con el sistema de control de errores del gestor interno.
- *restore_error_handler():* restablece el mecanismo de control de errores preestablecido por PHP.
- *Trigger_error():* se puede utilizar con el gestor interno de control de errores o con una nuevo definido con set_error_handler(). Es útil para generar una respuesta en particular a una excepción en tiempo de ejecución. Sólo necesita dos parámetros: el mensaje a visualizar ante el error de usuario y la constante predefinida del error.

```php
<?php

set_error_handler("micontrolador");
$dividir=5/0;
restore_error_handler();
function micontrolador($nivel, $mensaje)
{
    switch ($nivel)
    {
    case E_WARNING:
        $mensaje="no se puede dividir por cero";
        echo "Error de tipo WARNING: $mensaje";
        break;
    default:
        echo "error no identificado.";
    }
}
?>
```

Los errores que pueden ser controlados definiendo un controlador propio son:

- E_WARNING
- E_NOTICE
- E_USER_ERROR, E_USER_WARNING, E_USER_NOTICE, E_USER_DEPRECATED
- E_RECOVERABLE_ERROR

Notas:

- El nivel de error y la constante o número asociado a cada tipo de errores, es lo mismo.
- Una función callback es aquella cuyo nombre puede ser parámetro de entrada de otra función.

Actividad 2: Crea tu propio controlador de errores. Para ello, simula que una variable no es definida y el archivo especificado en un include, no existe.

Actividad 3: Los errores controlados utilizando set_error_handler() no son incluidos en el fichero log especificado en la directiva error_log del fichero ini.php. ¿Cómo podrías conseguirlo?.

2. Manejo de excepciones.

En PHP, a partir de la versión 5, se incluye el manejo de excepciones siendo similar al existente en otros lenguajes como es Java. Básicamente el funcionamiento es el siguiente:

- Si un código puede producir algún tipo de error, es incluido dentro de un bloque try.
- Cuando se produce algún error, se lanza una excepción a través de la sentencia throw.
- A continuación del bloque try debe existir como mínimo un bloque catch que procesará el error y evitará que el programa se detenga.
- Se puede añadir un bloque finally que se ejecuta tras try/catch, se genere o no el error, siendo un uso habitual en asociado a tareas de limpieza (cierre de conexión de una BD, limpiar variables, cerrar ficheros,…)

Por ejemplo,

```php
<?php
$divisor=5;
$dividendo=0;
try
{
if ($dividendo==0)
    throw new Exception ("Dividiendo por 0");
$cociente=$divisor/$dividendo;
}
catch (Exception $e)
{
        echo "Error: ".$e->getMessage();
}
?>
```

Para ello se utiliza una clase predeterminada *Exception* (https://www.php.net/manual/es/class.exception.php)

Se pueden personalizar las excepciones definiendo clases que extienden de la clase Exception; se crea la clase y heredará todo de ella.

```php
try {
    // init bootstrapping phase

    $config_file_path = "config.php";

    if (!file_exists($config_file_path))
    {
      throw new ConfigFileNotFoundException("Configuration file not found."
    }

    // continue execution of the bootstrapping phase
} catch (ConfigFileNotFoundException $e) {
    echo "ConfigFileNotFoundException: ".$e->getMessage();
    // other additional actions that you want to carry out for this excepti
    die();
} catch (Exception $e) {
    echo $e->getMessage();
    die();
}
?>
```

https://code.tutsplus.com/es/tutorials/php-exceptions-try-catch-for-error-handling--cms-32013

Al igual que para el control de errores, se cuenta con una serie de funciones para el manejo de las excepciones. Una de ellas es *set_exception_handler()* que permite crear un controlador de excepciones no detectadas.

```php
<?php
function myException($exception) {
  echo "<b>Exception:</b> " . $exception->getMessage();
}

set_exception_handler('myException');

throw new Exception('Uncaught Exception occurred');
?>
```

https://www.w3schools.com/php/php_exception.asp

En este código se lanza una excepción (throw) pero no se recoge con un catch(); se encarga la función definida y el controlador de excepciones no detectadas (sin catch()).

En **php 7** surgen tipos nuevos de excepciones para el control de los errores pero no heredan de la clase Exception sino de la clase Error. Se pueden utilizar para manejar errores como excepciones (https://www.php.net/manual/es/class.error.php)

Muchos de los errores fatales (E_ERROR) o recuperables (E_RECOVERABLE_ERROR) pueden ser capturados a través de esta clase Error.

Fatal error: Uncaught Error: Call to undefined function suma() in E:\WDES\UT3\errores.php:4 Stack trace: #0 {main} thrown in **E:\WDES\UT3\errores.php** on line **4**

Conclusiones.

- Todo programa debería iniciarse con un código que incluyera el control de errores y excepciones.
- No es lo mismo realizar un control de errores que de excepciones. Hay muchos errores que detienen la ejecución de un programa y que escapa del control del programador ya que su control recae en el núcleo de php o en su intérprete y cuya recuperación no es posible a través de código. Sin embargo las excepciones son generadas dentro del código y es posible recuperarse de ellas a través de las sentencias especificadas en catch.
- Por otra parte las excepciones no quedan registradas en un fichero log; para conseguirlo, deberíamos incluir dentro del catch el tratamiento que deseemos con la excepción incluyendo en el fichero log.text que se esté utilizando el mensaje generado en la excepción (tampoco quedan registrados los errores si se utilizado un controlador de errores propio a través de set_error_handler()).

Actividad 4: Crea tu propio control de excepciones para los casos recogidos en la actividad 2.

Actividad 5: Incluir un controlador de excepciones en la actividad anterior y el código necesario para que se ejecute.

Actividad 6: Los errores controlados por excepciones tampoco quedan recogidos en el registro de errores. Consíguelo. Anula los mensajes derivados del gestor interno utilizando la función ini_set.

Actividad 7: Crea una clase que herede de la clase Exception para extender su función y controlar diferentes excepciones producidas en un código.

Actividad 8: Utiliza las clases de errores de php 7 en algún ejemplo de control de excepciones.