# 1.- Errores.

PHP tiene una clasificación de los distintos errores que pueden dar lugar cuando se tiene un programa en ejecución. Esta clasificación se hace mediante el uso de constantes, las cuales comienzan por "*E_*". Algunas de las más importantes son:

- **E_ERROR:** indica errores fatales en tiempo de ejecución y que no se pueden recuperar.
- **E_NOTICE:** realiza un aviso e tiempo de ejecución en el que un script podría indicar un error.
- **E_CORE_ERROR:** se trata de errores fatales que ocurren durante el arranque inicial de php.

El archivo que se encarga de indicar como se van a tratar los errores producidos es *php.ini*. Dos de los parámetros más importantes son:

- **error_reporting:** establece el nivel de notificación de errores, es decir, los errores que se mostrarán al usuario. El valor predeterminado es *E_ALL & ~E_NOTICE & ~E_STRICT & ~E_DEPRECATED*, que no muestra los errores de las constantes *E_NOTICE*, *E_STRICT* y *E_DEPRECATED*.
- **display_errors:** indica si los errores se muestran por pantalla al usuario o se ocultan.
- **log_errors:** indica si los mensajes de error se han de registrar en el servidor o en *error_log* (nombre del fichero donde los errores del script deberían ser registrados).

Desde código existe la posibilidad de usar los parámetros anteriormente mencionados para indicar, por ejemplo, el nivel de notificación en un momento dado.

```
...
error_reporting(E_ALL & ~E_NOTICE & ~E_STRICT & ~E_DEPRECATED & ~E_WARNING);
$resultado = $dividendo / $divisor;
error_reporting(E_ALL & ~E_NOTICE & ~E_STRICT & ~E_DEPRECATED);
...
```

Una forma más efectiva de gestionar los errores es mediante el uso de la función *set_error_handler.* Con ella se puede realizar una gestión personalizada de los errores que se produzcan durante la ejecución de programa.

```
<?php
set_error_handler("miGestorDeErrores");
$resultado = $dividendo / $divisor;

// función de gestión de errores
function miGestorDeErrores($errno, $errstr)
{
    switch ($errno) {
    case E_USER_ERROR:
        echo "<b>Mi ERROR</b> [$errno] $errstr<br />\n";
        echo "  Error fatal en la línea $errline en el archivo $errfile";
        echo ", PHP " . PHP_VERSION . " (" . PHP_OS . ")<br />\n";
        break;
    case E_USER_WARNING:
        echo "<b>Mi WARNING</b> [$errno] $errstr<br />\n";
        break;
}
?>
```

La función *restore_error_handler* restaura el manejador de errores de php que está definido en el archivo *php.ini.*

## 2.- Excepciones.

Php ha incorporado a partir de su versión 5.0 un modelo de excepciones, similar al que hay disponible en otros lenguajes de programación, mediante el uso de *try – catch.*

- **try:** contiene el código en el que se puede producir un error.
- **catch:** es el encargado de gestionar el error. Puede existir más de un *catch* para un *try.*
- **throw:** una excepción es lanzada mediante *throw* cuando se ha producido un error. Si se ha finalizado el conjunto de instrucciones delo bloque *try* y no se ha lanzado ninguna excepción, la ejecución del programa prosigue después de los bloques *try – catch.*

```
<?php
      function inverso($x)
      {
          if (!$x)
          {
              throw new Exception('División por cero.');
          }
          return 1/$x;
      }

      try
      {
          echo inverso(5) . "\n";
          echo inverso(0) . "\n";
      }
      catch (Exception $e)
      {
          echo 'Excepción capturada: ',  $e->getMessage(), "\n";
      }

      // Continuar la ejecución
      echo 'Hola Mundo\n';
?>
```

*Exception* es la clase base para todas las excepciones en PHP5, y para todas las excepciones de usuario en PHP7. Las propiedades que posee son:

- **message:** el mensaje de la excepción.
- **code:** el código de la excepción.
- **file:** el nombre del fichero donde se originó la excepción.
- **line:** la línea donde se originó la excepción.

A estas propiedades se puede acceder mediante el uso de los métodos *get*, por ejemplo *getMessage.*