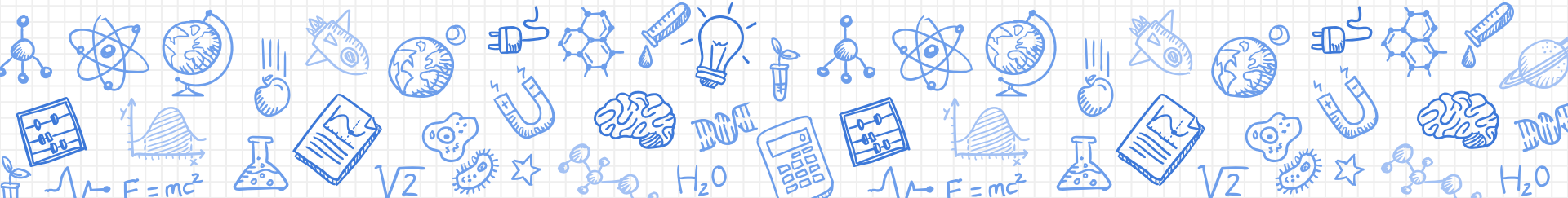
PHP Forms

Error Handling & Logging



Error Handling and Logging

PHP has functions to deal with errors and logs.

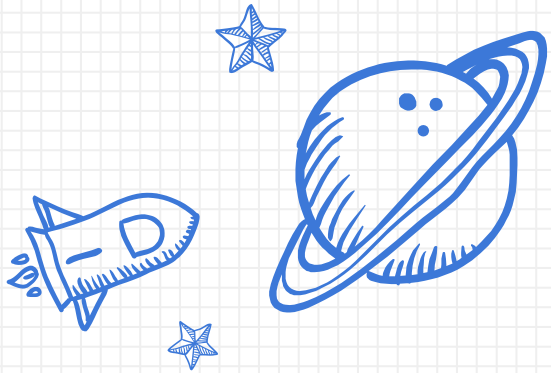
They allow you to define your own error handling rules, as well as modify the way the errors can be logged.

This allows you to change and enhance error reporting to suit your needs.

Configuring

There are not external libraries or installation needed, they are part of the **PHP core**.

The behaviour of these functions is affected by settings in **php.ini**.



PHP support Error Handling & Logging
by default in the PHP core.

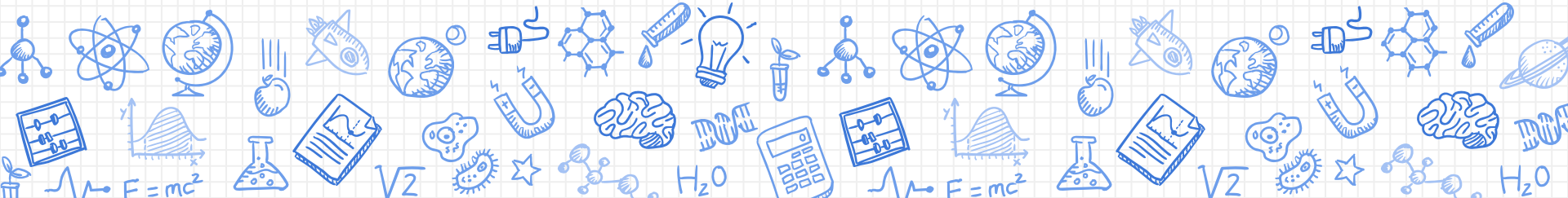
```php
<?php

    // display all errors - in code config overwrite
    error_reporting(E_ALL);
    ini_set('display_errors', 1);

?>
```

Set the php config to show the errors.

Errors and Logging Configuration Options	
error_reporting	Sets which PHP errors are reported
display_errors	This determines whether errors should be printed to the screen as part of the output or if they should be hidden from the user.
display_startup_errors	Even when display_errors is on, errors that occur during PHP's startup sequence are not displayed. It's strongly recommended to keep display_startup_errors off, except for debugging.
log_errors	Tells whether script error messages should be logged to the server's error log or error_log. This option is thus server-specific.
log_errors_max_len	Set the maximum length of log_errors in bytes. In error_log information about the source is added. The default is 1024 and 0 allows to not apply any maximum length at all. This length is applied to logged errors, displayed errors and also to **$php_errormsg**.
ignore_repeated_errors	Do not log repeated messages. Repeated errors must occur in the same file on the same line unless ignore_repeated_source is set true.
ignore_repeated_source	Ignore source of message when ignoring repeated messages. When this setting is On you will not log errors with repeated messages from different files or sourcelines.
report_memleaks	If this parameter is set to On (the default), this parameter will show a report of memory leaks detected by the Zend memory manager.

Errors and Logging Configuration Options	
track_errors	If enabled, the last error message will always be present in the variable **$php_errormsg**.
html_errors	Turn off HTML tags in error messages. The new format for HTML errors produces clickable messages that direct the user to a page describing the error or function in causing the error. These references are affected by docref_root and docref_ext.
xmlrpc_errors	Turns off normal error reporting and formats errors as XML-RPC error message.
xmlrpc_error_number	Used as the value of the XML-RPC faultCode element.
docref_root	The new error format contains a reference to a page describing the error or function causing the error.
docref_ext	docref_root extension
error_prepend_string	String to output before an error message.
error_append_string	String to output after an error message.
error_log	Name of the file where script errors should be logged. The file should be writable by the web server's user.

PHP Forms



PHP Forms

One of the most powerful features of PHP is the way it handles HTML forms.

The basic concept that is important to understand is that any form element will automatically be available to your PHP scripts.

The PHP superglobals **$_GET** and **$_POST** are used to collect form-data.

$_GET vs $_POST

$_GET

- ✘ To request data
- ✘ To send non-sensitive data
- ✘ All data is visible via the URL
- ✘ Limited to the max URL characters
- ✘ Possible to bookmark the page

$_POST

- ✘ To create, update and delete data
- ✘ To send sensitive data
- ✘ Invisible to others embedded within the HTTP request
- ✘ Support for multi-part binary input while uploading

```
<form action="action.php" method="post">
 <p>Your name: <input type="text" name="name" /></p>
 <p>Your age: <input type="text" name="age" /></p>
 <p><input type="submit" /></p>
</form>
```

```
<!-- form.php -->
<form action="action.php" method="post">
 <p>Your name: <input type="text" name="name" /></p>
 <p>Your age: <input type="text" name="age" /></p>
 <p><input type="submit" /></p>
</form>

<!-- action.php-->
Hi <?php echo htmlspecialchars($_POST['name']); ?>.
You are <?php echo (int)$_POST['age']; ?> years old.
```

htmlspecialchars — Convert special characters to HTML entities

isset()

Determine if a variable is set and is not NULL.

If a variable has been unset with **unset()**, it will no longer be set. **isset()** will return FALSE if testing a variable that has been set to NULL.

```php
<?php

    if(isset($_POST['inputName'])){
        // code
    }

?>
```

isset() is useful to determinate if a value is set.

empty()

Determine whether a variable is considered to be empty. A variable is considered empty if it does not exist or if its value equals FALSE. empty() does not generate a warning if the variable does not exist.

```php
<?php

    if(empty($_POST['inputName'])){
        echo 'Name is required.';
    }

?>
```

empty() determinates id the value/variable is empty or not defined.