

CT4009

Error Handling & Debugging

Andy Bell

CT4009

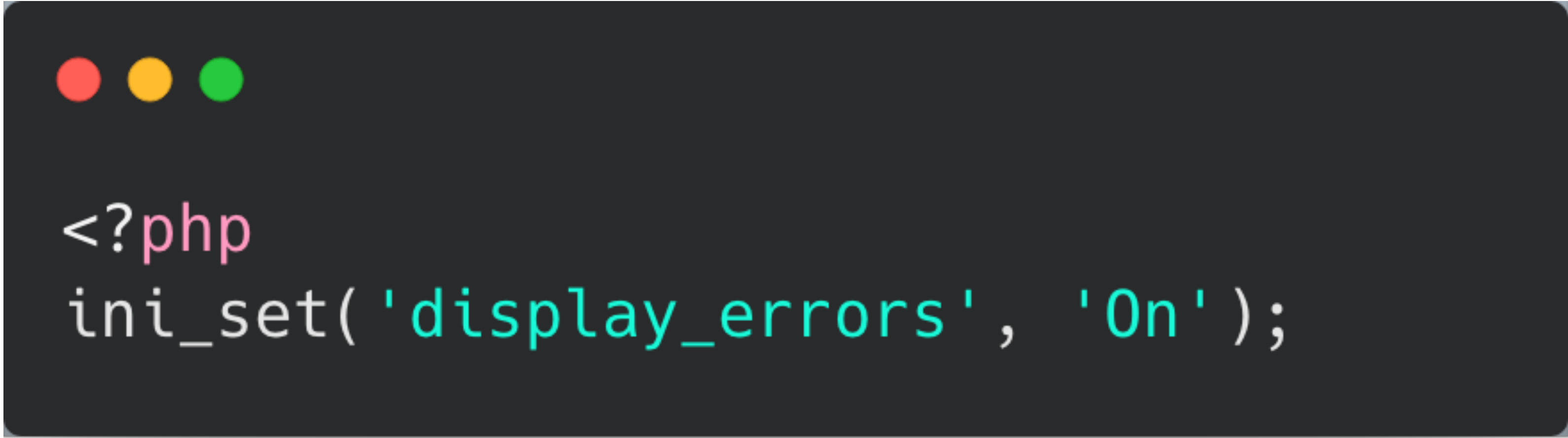
Error Handling & Debugging

What we'll cover today

- How to see errors
- PHP error types
- Debugging options

Enable errors

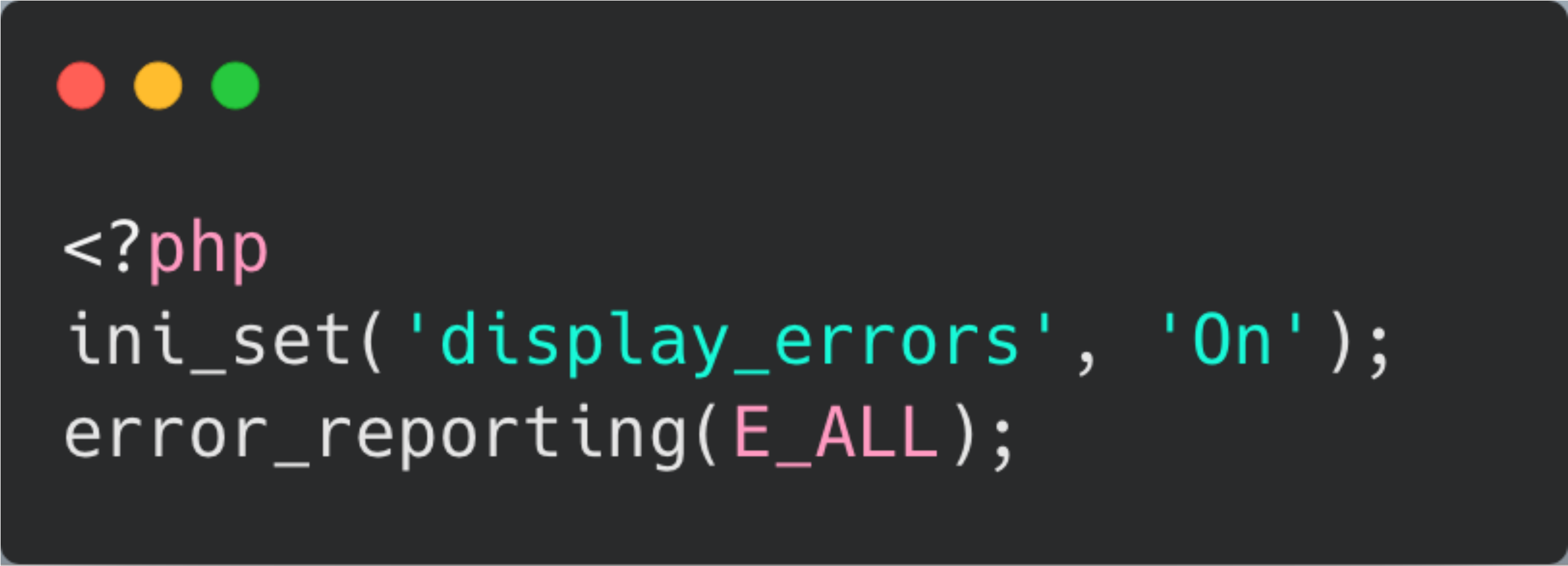
By default, a lot of servers will have errors disabled, so add this at the **entry point** (usually index.php) of your PHP application.



```
<?php  
ini_set( 'display_errors', 'On' );
```

Enable all errors

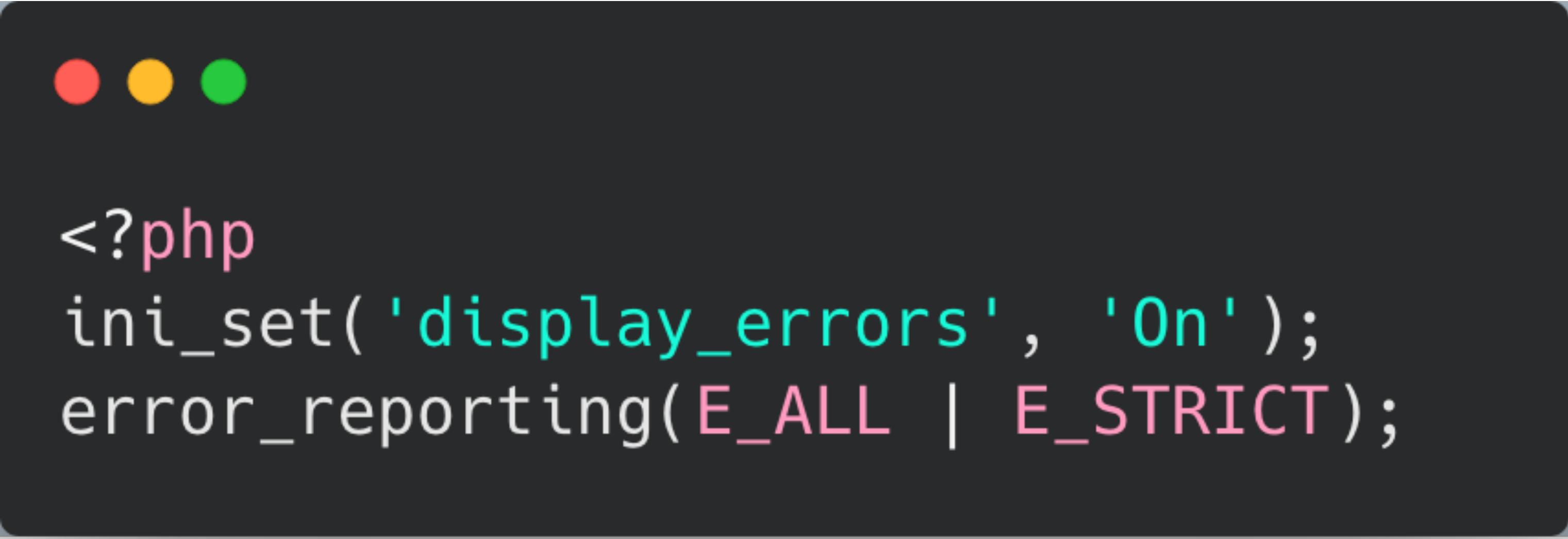
Sometimes, less severe errors like **notices** won't show, so amend our example to this:



```
<?php
ini_set( 'display_errors', 'On' );
error_reporting( E_ALL );
```

Strict mode

This mode will show **everything** which is ideal for development



```
<?php
ini_set( 'display_errors', 'On' );
error_reporting(E_ALL | E_STRICT);
```

⚠️ WARNING ⚠️

You **shouldn't** enable errors on a production server. It's fine on your student sites websites, but on a real environment enabling errors **slows your server down**.

Error Types

CT4009

Error Handling & Debugging

Error type: Syntax / Parse Error

Syntax errors or parse errors are generally caused by a typo in your code. They will be **breaking errors** that stop your code from continuing.

Typos could include:

- Missing semicolon (“;”)
- Unmatched quotation marks
- Missing brace (“{” or “}”)
- Missing parenthesis (“(” or “)”)

Example



```
<?php  
$myVariable = "Hello, everyone"  
echo $myVariable;
```

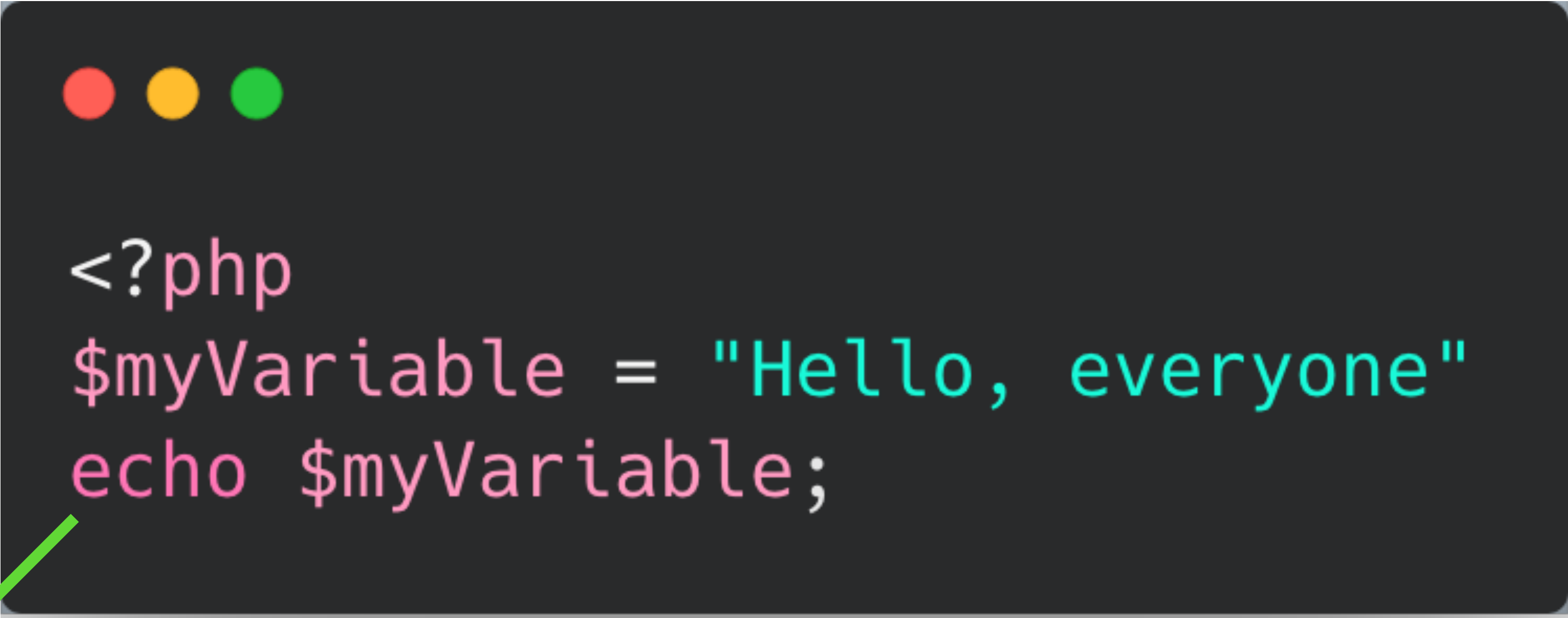
Example: Error message

Parse error: syntax error, unexpected 'echo' (T_ECHO) in /Users/andybell/Desktop/index.php on line 3

Let's break down each part:

- “**unexpected 'echo'**” means that some code started to execute before PHP was ready
- “**/Users/andybell/Desktop/index.php**” is the path to the file that threw the error
- “**line 3**” is the line where the error happened


Example



```
<?php  
$myVariable = "Hello, everyone"  
echo $myVariable;
```

Line 3

Example



```
<?php  
$myVariable = "Hello, everyone"  
echo $myVariable;
```

Where the actual error is

CT4009

Error Handling & Debugging

If it's not clear where the error is,
look at the line above

Error type: Warning

Warnings are less severe, and in some cases, will not break your code.

Warnings could include:

- Headers already sent. (*Check for white space at the head of your code or in files you're including.- Unmatched quotation marks*)
- Incorrect number of parameters to a function
- Incorrect path names when including files

Example



```
<?php
```

```
include '/include-file.php';
```

Example: Error message

Warning: include(/include-file.php): failed to open stream: No such file or directory in /Users/andybell/Desktop/index.php on line 2

I got the include path wrong, so PHP helpfully tells me that in a warning.

If I remove the '/' from the beginning of the include path, it'll fix the issue.

Error type: Notice

A notice is almost the same as a **warning**. They are just one level less severe.

Notices will **always** be reported, wether it's to screen or to logs (unless a server is explicitly configured), so it's worth trying to remove them before pushing code live.

Example



```
<?php
$arrayExample = [
    'name' => 'Andy',
    'email' => 'andy@email.com'
];

echo $arrayExample['age'];
```

Example: Error message

Notice: Undefined index: age in /Users/andybell/Desktop/index.php on line 9

Let's break down each part:

- **“Undefined index: age”** means that we tried to use an array index that wasn't there
- **“/Users/andybell/Desktop/index.php”** is the path to the file that threw the error
- **“line 9”** is the line where the error happened

Notice how the error starts with “Notice:” (pun intended)

Error type: Fatal

Although the most dramatic sounding, they are often the easiest to fix because they break on the exact failure point. Your application will **stop working** with a 500 error.

The usual suspect is a call to a function that doesn't exist.

Example



```
<?php  
echo aFunctionThatDoesntExist();
```

Example: Error message

Fatal error: Uncaught Error: Call to undefined function aFunctionThatDoesntExist() in /Users/andybell/Desktop/index.php:2

Stack trace: #0 {main} thrown in /Users/andybell/Desktop/index.php on line 2

The error shows the exact breaking point. It also shows a **Stack Trace** which is a list of all of the code that was executed before this error. This is handy for giving you context about what the error is.

Debugging options

CT4009

Error Handling & Debugging

die(\$message)

- **die()** does exactly what it says on the tin
- It's should only be used as a debug tool and shouldn't be in your **live** codebase
- It'll show whatever **\$message** you pass in and stop the **entire application**

Example



```
<?php
$aVariable = 5;

if($aVariable < 10) {
    die('The variable was less than 10');
}

echo 'You won't see this if $aVariable is less than 10';
```


var_dump()

- **var_dump()** is very similar to **console.log** in JavaScript
- It's recommended that you wrap your output in `<pre>` tags so the formatting is maintained
- Your code will continue as usual after a **var_dump()**

var_dump()

- **var_dump()** is very similar to **console.log** in JavaScript
- It's recommended that you wrap your output in `<pre>` tags so the formatting is maintained
- Your code will continue as usual after a **var_dump()**
- It's a great way for seeing what a variable or function's result is at any given point

Example



```
<?php
$arrayExample = [
    'name' => 'Andy',
    'email' => 'andy@email.com'
];

echo '<pre>';
var_dump($arrayExample);
echo '</pre>';
```

CT4009

Error Handling & Debugging

Result

Output:

```
array(2) {  
    ["name"]=>  
    string(4) "Andy"  
    ["email"]=>  
    string(14) "andy@email.com"  
}
```

XDebug

Xdebug is a PHP extension that aims to lend a helping hand in the process of debugging your applications. Xdebug offers features like:

- Automatic stack trace upon error
- Function call logging
- Display features such as enhanced `var_dump()` output and code coverage information.

Link to more information: <https://xdebug.org/>

Debugging tools as a service

There are a number of tools that extend debugging that we use in the industry. The main aim of these tools is to turn your log files into something very human readable and actionable.

They are very useful for monitoring your **live** code!

Popular services:

- Sentry: <https://sentry.io>
- Rollbar: <https://rollbar.com/>
- Raygun: <https://raygun.com/>

Recap

- We learned how to see errors
- We learned about PHP error types
- We learned about Debugging options