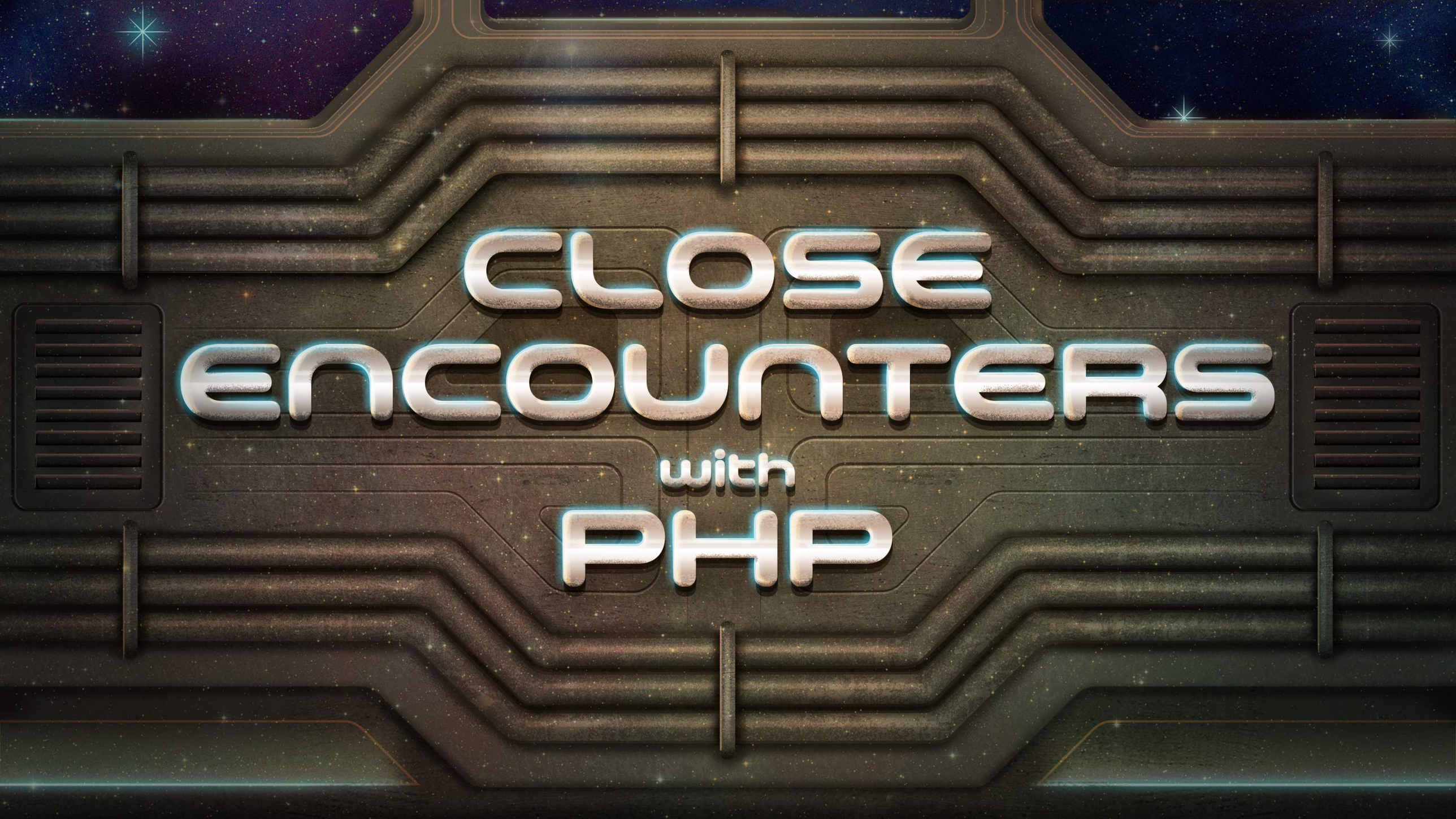

CLOSE
ENCOUNTERS
with
PHP

Level 5

Validation With Respect

Try to Validate; Catch the Exceptions

Returning Messages With If/Else

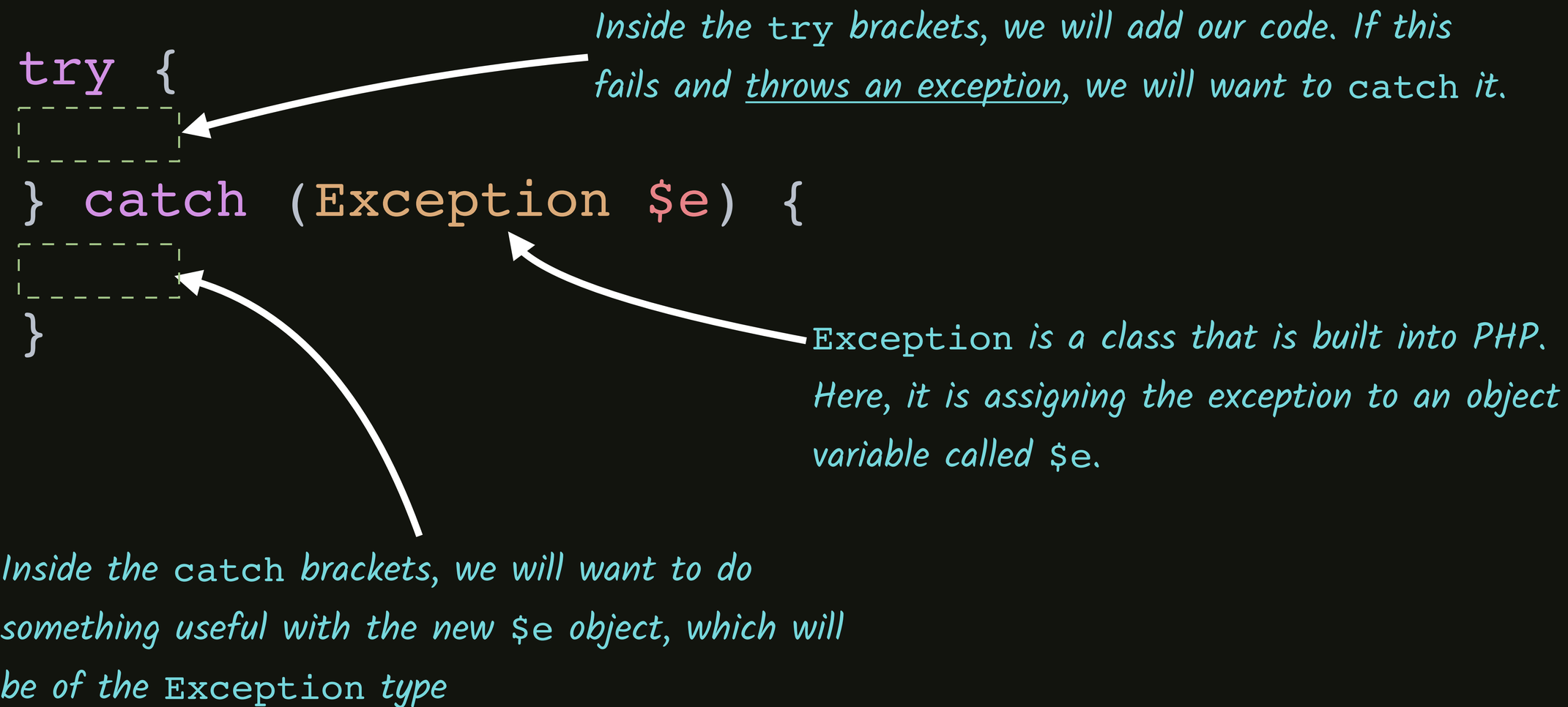
app/src/validation.php

```php
<?php
use Respect\Validation\Validator;

function validate_date($date_string)
{
    $date_validator = Validator::date('d-m-Y')->notEmpty();

    if ($date_validator->validate($date_string)) {
        $date_time = strtotime($date_string);
        return date('F jS Y', $date_time);
    } else {
        return 'The date must be in a DD-MM-YYYY format.';
    }
}
```

Objects have exceptions, which can also return messages!

Try to Validate, Catch the Exception

In PHP we have a try/catch **statement, which will allow us to work with object errors.**



Using Respect's Exception Classes

app/src/validation.php

```php
<?php
use Respect\Validation\Validator;
use Respect\Validation\Exceptions\NestedValidationException;

function validate_date($date_string)
{
    $date_validator = Validator::date('d-m-Y')->notEmpty();

    if ($date_validator->validate($date_string)) {
        $date_time = strtotime($date_string);
        return date('F jS Y', $date_time);
    } else {
        return 'The date must be in a DD-MM-YYYY format.';
    }
}
```

Include a class with the `use` *keyword.*

Now `NestedValidationException` *is available to our code.*

Creating the Try/Catch for Validation

app/src/validation.php

```php
<?php
use Respect\Validation\Validator;
use Respect\Validation\Exceptions\NestedValidationException;

function validate_date($date_string)
{
    $date_validator = Validator::date('d-m-Y')->notEmpty();

    try {
        $date_validator->assert($date_string);
        $date_time = strtotime($date_string);
        return date('F jS Y', $date_time);
    } catch (NestedValidationException $e) {
        var_dump($e);
    }
}
```

Where `validate` *returned true or false, we need to use* `assert`*, which will fail with an* `Exception`

Failures will throw a `NestedValidationException`

Submitting an Invalid Date Format

```
/var/www/hello/app/src/validation.php:14:
object(Respect\Validation\Exceptions\AllOfException)[7]
  private 'exceptions' (Respect\Validation\Exceptions\NestedValidationException) =>
    object(SplObjectStorage)[8]
      private 'storage' =>
        array (size=1)
          '00000000535eff73000000006c81c734' =>
            array (size=2)
              ...
  protected 'id' => string 'allOf' (length=5)
  protected 'mode' => int 1
  protected 'name' => string '"someday"' (length=9)
  protected 'template' => string 'These rules must pass for {{name}}' (length=34)
  protected 'params' =>
    array (size=7)
      'name' => null
      'template' => null
      'rules' =>
        array (size=2)
          '00000000535eff72000000006c81c734' =>
            object(Respect\Validation\Rules\Date)[5]
              ...
          '00000000535eff71000000006c81c734' =>
            object(Respect\Validation\Rules\NotEmpty)[6]
```

Here is our `Exception` *object*

Here is the data we submitted and the rules that it must pass

How Can We Use the Exception?

app/src/validation.php

```
<?php
...
    $date_validator = Validator::date('d-m-Y')->notEmpty();

    try {
        $date_validator->assert($date_string);
        $date_time = strtotime($date_string);
        return date('F jS Y', $date_time);
    } catch (NestedValidationException $e) {
        var_dump($e->getMessages());
    }
}
```

The `NestedValidationException` *has a method named* `getMessages` *that will return some useful information*

Let's submit a 'someday' as our date and see what happens

Viewing the getMessages Method

`getMessages` *is returning an array, with one error*

```
/var/www/hello/app/src/validation.php:14:
array (size=1)
  0 => string '"someday" must be a valid date. Sample format: "30-12-2005"' (length=59)
```

Our item is a string with some very helpful information we can give the user!

Returning the Errors to the User

app/src/validation.php

```php
<?php
...
    $date_validator = Validator::date('d-m-Y')->notEmpty();

    try {
        $date_validator->assert($date_string);
        $date_time = strtotime($date_string);
        return date('F jS Y', $date_time);
    } catch (NestedValidationException $e) {
        return $e->getMessages();
    }
}
```

We will return the array of errors!

One Method for Presenting the Errors

app/src/app.php

```php
<?php
    require __DIR__ . '/../../vendor/autoload.php';
    require __DIR__ . '/validation.php';
    ...

    if (!empty($date) && !empty($email) && !empty($description)) {

        $value = validate_date($date);

        if (is_array($value)) {
            foreach ($value as $error) {
                echo "<span class='error'>$error</span>";
            }
        } else {
            echo $value;
        }
```

We will now get a formatted date, or an array of errors

`is_array` returns true for an array

Now we can loop through the errors and present each one with a `span` tag

A Better Approach, With Validation

app/src/app.php

```php
<?php
require __DIR__ . '/../../vendor/autoload.php';

use Respect\Validation\Validator;
use Respect\Validation\Exceptions\NestedValidationException;

if($_SERVER['REQUEST_METHOD'] === 'POST')) {
    $date = trim($_POST['date']);
    $email = trim($_POST['email']);
    $description = trim($_POST['desc']);

    $date_validator = Validator::date('d-m-Y')->notEmpty();
    $email_validator = Validator::email()->notEmpty();
    $desc_validator = Validator::stringType()->length(1, 750);

    try {
        $date_validator->assert($date);
```

Include our validation library

Create custom validators for all of the form fields — each will have its own types of validation

A Better Approach, With Validation

app/src/app.php

```php
try {
    $date_validator->assert($date);
    $email_validator->assert($email);
    $desc_validator->assert($description);

    echo date('F jS Y', strtotime($date));
    echo $email;
    echo $description;

} catch (NestedValidationException $e) {
    echo'<ul>';
    foreach ($e->getMessages() as $message) {
        echo"<li>$message</li>";
    }
    echo'</ul>';
}
```

Using a `try/catch`, we will assert each of our custom validators — each will have their own `Exception` if it fails

Here, we are echoing — in a real-world application, you might write this to a database!

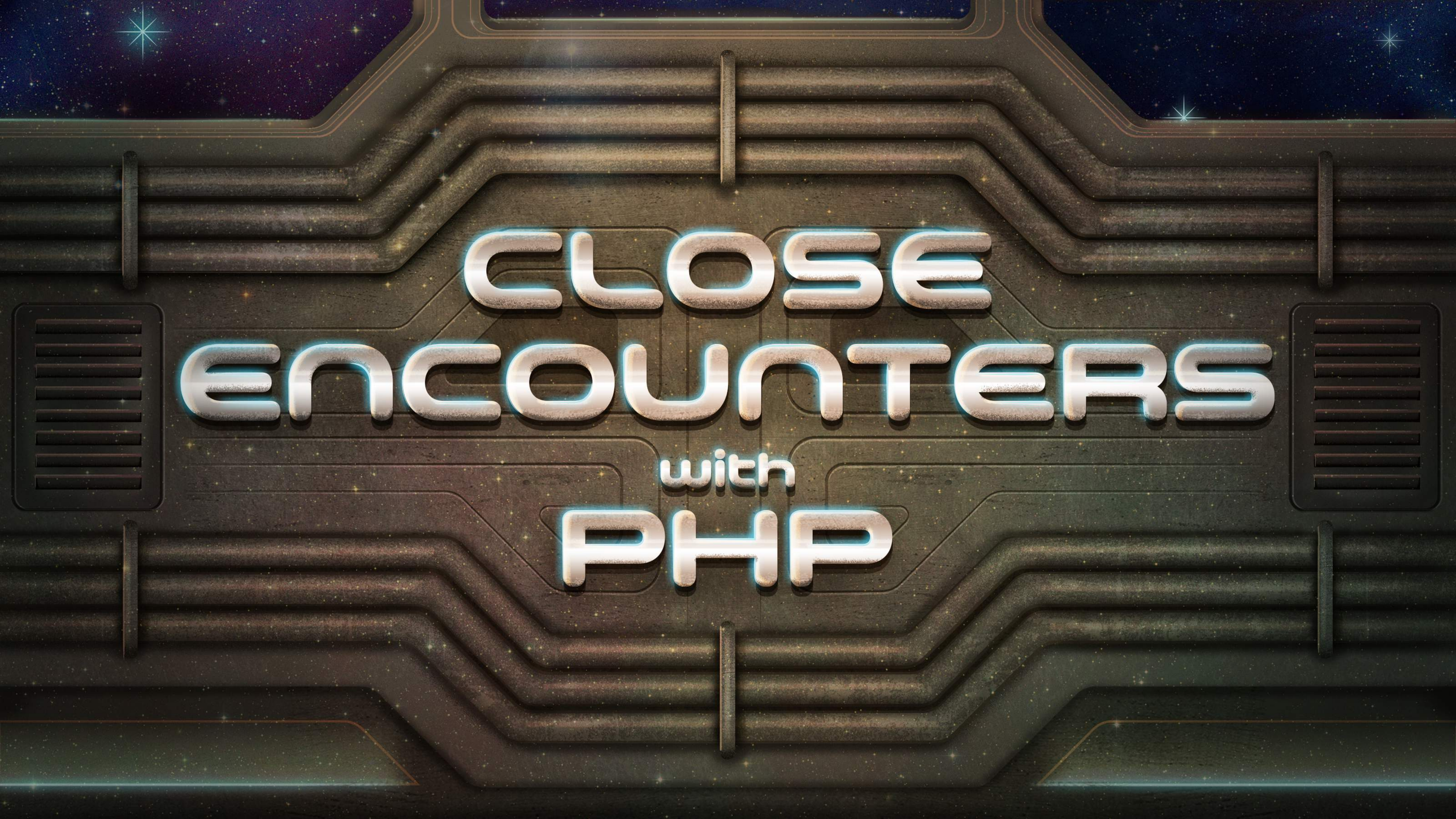
If an `Exception` occurs, we will loop through each of them and output to an unordered list

Validation Review

What have we learned in this section?

- Using a **try/catch** block
- Working with exceptions and custom exceptions
- The **getMessages** method for validation error messages
- Returning errors as an array
- Returning errors by refactoring, using a single **try/catch** block

CLOSE ENCOUNTERS with PHP


CLOSE
ENCOUNTERS
with
PHP