

CLOSE ENCOUNTERS

with
PHP

What Are We Going to Cover?

Here is the structure of our course for each level.

Level One: HTTP requests with GET & POST

Level Two: Using includes and requires

Level Three: Custom validation and input security

Level Four: Composer package management and autoloading

Level Five: Using a validation package

Before We Begin

Here are the some suggested prerequisites for this course.



Basic HTML & CSS

Front-end Foundations & Front-end Formations



Basic PHP

Try PHP

CLOSE
ENCOUNTERS
with
PHP

What Are We Going to Build?

Date of Sighting

Your Email

Describe the Sighting

Submit

Array ([0] => "yes" must be a valid date)

Level 1

Requests & Forms


GET & POST

What Is a GET Request?

Here are some details about HTTP requests using the GET method.

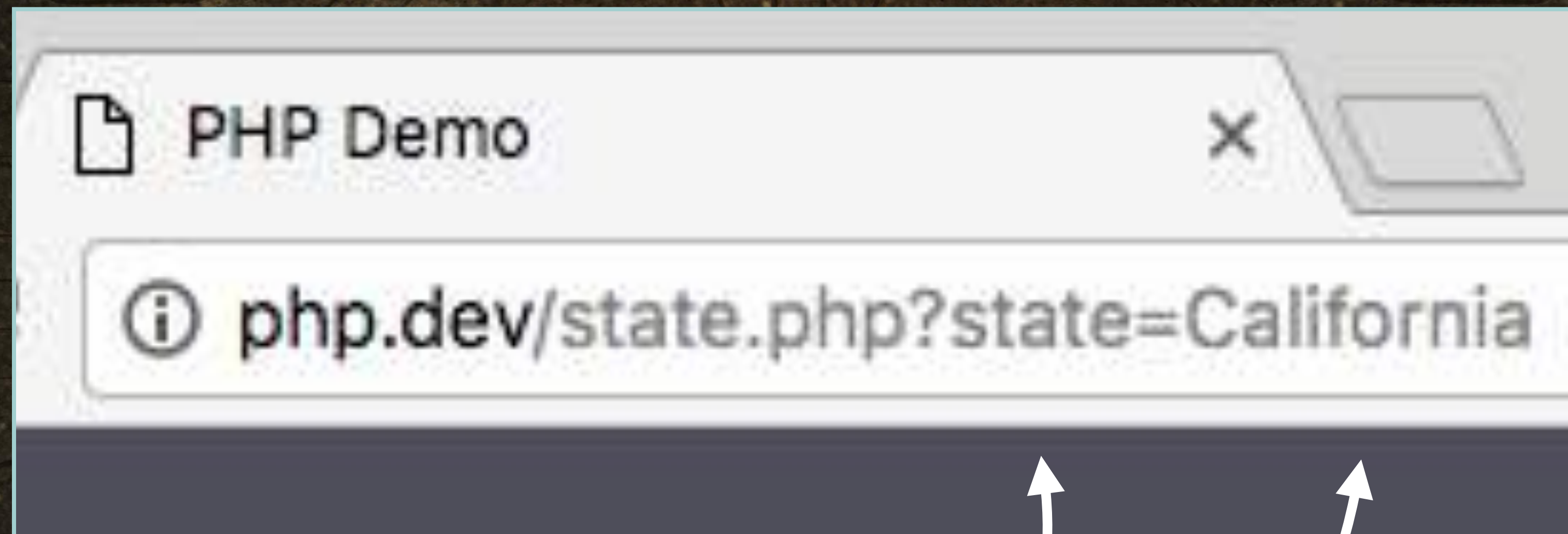
- GET is used to request data from a resource
- GET requests show query strings and values in the URL
- GET requests can be bookmarked
- GET requests can be cached
- GET requests remain in your history
- GET requests are only for retrieving data

Let's look at an example of this



UFO Sightings by State

Using a GET request, we will pass a query var containing the state.



Query variable

Variable value

CLOSE
ENCOUNTERS
with
PHP

Inspecting the Request

Here is an inspection of the headers of the GET request.



▼ General

Request URL: `http://php.dev/state.php?state=California`

Request Method: GET

Status Code: ● 200 OK

Remote Address: 127.0.0.1:80

► Response Headers (6)

► Request Headers (9)

▼ Query String Parameters [view source](#) [view URL encoded](#)

state: California


Using the GET request method

Our query var 'state' and value of 'California'

What Is a POST Request?

Here are some details about HTTP requests using the POST method.

- POST is used to send data to a resource
- POST requests show query names and values only in the body of the request
- POST requests are **never** bookmarked
- POST requests are **never** cached
- POST requests will not remain in your history



Let's look at an example of this

Simple Form With POST Method

A simple form to request data about a state.

The form will submit data to state.php

state.php

```
<form class="" action="state.php" method="post">
<label for="state">State Name</label><br>
<input type="text" name="state" value="">
<hr>
<br>
<input type="submit" value="submit">
</form>
```

Use the POST method for the form

Simple Form With POST Method

A simple form to request data about a state.

state.php

```
<form class="" action="state.php" method="post">
<label for="state">State Name</label><br>
<input type="text" name="state" value="">
<hr>
<br>
<input type="submit" value="submit">
</form>
```

"state" is how we can access the data the user entered after it is passed to state.php

Inspecting the Request

Here is an inspection of the headers of the POST request.



The screenshot displays the 'General' tab of a web browser's developer tools. It shows the following details:

- Request URL:** `http://php.dev/state.php`
- Request Method:** `POST`
- Status Code:** ● `200 OK`
- Remote Address:** `127.0.0.1:80`

Below the 'General' tab, there are expandable sections for 'Response Headers (6)', 'Request Headers (13)', and 'Form Data'. The 'Form Data' section is expanded, showing a single entry:

Field	Value
state	California

No exposed data in the URL

Using the POST request method

Our query 'state' and value of 'California'

CLOSE ENCOUNTERS

with
PHP

Level 1

Requests & Forms

Working With Form Data

What Are We Going to Build?

A date (16-11-1977)

Date of Sighting

Your Email

Describe the Sighting

Submit

Array ([0] => "yes" must be a valid date)

We will also need an area for errors or messages

Looking at the Sighting Form

index.php

```
<form class="" action="index.php" method="post">
```

```
<label for="date">Date of Sighting</label><br>
```

```
<input type="text" name="date" value="">
```

```
<hr>
```

```
<label for="email">Your Email</label><br>
```

```
<input type="text" name="email" value="">
```

```
<hr>
```

```
<label for="desc">Describe the Sighting</label><br>
```

```
<textarea name="desc" rows="8" cols="40"></textarea>
```

```
<br>
```

```
<input type="submit" value="submit">
```

```
</form>
```

Set the method to POST

Send the request to index.php

POST Data Returned From Form Submission



The screenshot displays a web browser's developer console with the following sections:

- General**
 - Request URL: `http://php.dev/`
 - Request Method: `POST`
 - Status Code: ● `200 OK`
 - Remote Address: `127.0.0.1:80`
- Response Headers (6)**
- Request Headers (13)**
- Form Data** (with links for `view source` and `view URL encoded`)
 - `date: 2016-1-1`
 - `email: you@example.com`
 - `desc: 'Ancient alien choral castle ancient alien clearly extraterrestrial, v
eiroglyph contend DNA manipulation, legendary times Mayan space time worm ho
ma Punku earth mound mystery Annunaki vimana, burmuta triangle Vymaanika-Sha
ic current, ancient alien theorists elongated skull golden disk cover up.'`

We need to access this data in PHP

How Do We Access the POST Data?

index.php

```
<?php
```

```
//Print All Data in POST
```

```
var_dump($_POST);
```

\$_POST is a PHP superglobal variable



```
?>
```

```
<!DOCTYPE html>
```

```
<html>
```

```
...
```

```
<form> ... </form>
```

```
...
```

```
</html>
```


Viewing the Output

index.php

```
/var/www/php/public/index.php:8:
array (size=3)
  'date' => string '2016-01-01' (length=10)
  'email' => string 'you@example.com' (length=15)
  'desc' => string 'Ancient alien choral castle ancient alien clearly
extraterrestrial, vimana extraterrestrial helicopter heiroglyph
contend DNA manipulation, legendary times Mayan space time worm hole.
Foo fighter electromagnetic Puma Punku earth mound mystery Annunaki
vimana, burmuta triangle Vymaanika-Shaashtra sun disc anti-gravity
magnetic current, ancient alien theorists elongated skull golden disk
cover up.' (length=397)
```


Accessing Each Post Variable Independently

index.php

```
<?php
//Echo each item in POST
echo $_POST['date'];
echo $_POST['email'];
echo $_POST['desc'];
?>
```

```
<!DOCTYPE html>
<html>
...
  <form>...</form>
...
</html>
```


Errors on Refresh?!

(!) Notice: Undefined index: date in /var/www/php/app/public/index.php on line 2


Call Stack

#	Time	Memory	Function	Location
1	0.0008	357024	{main}()	.../index.php:0
2	0.0067	362152	require('/var/www/php/app/public/index.php')	.../index.php:3

(!) Notice: Undefined index: email in /var/www/php/app/public/index.php on line

Call Stack

#	Time	Memory	Function	Location
---	------	--------	----------	----------



This tells us that `$_POST['date']` does not exist!

Does POST Data Exist?

index.php

```
<?php
if($_SERVER['REQUEST_METHOD'] === 'POST') {
```

```
    // Echo each item in POST
```

```
    echo $_POST['date'];
```

```
    echo $_POST['email'];
```

```
    echo $_POST['desc'];
```

```
}
```

```
?>
```

```
<!DOCTYPE html>
```

```
<html>
```

```
...
```

```
    <form>...</form>
```

```
...
```

```
</html>
```



First, check if the server request method is POST

Cleaning Up a Bit

index.php

```
<?php
if($_SERVER['REQUEST_METHOD'] === 'POST') {
    $date = $_POST['date'];
    $email = $_POST['email'];
    $description = $_POST['desc'];

    echo "<p>Date: $date</p>";
    echo "<p>Email: $email</p>";
    echo "<p>$description</p>";
}
?>
```

For now, we will just print the data to the page

```
<!DOCTYPE html>
<html>
...
    <form>...</form>
...
```


Viewing the Results

Date: 2016-1-1

Email: you@example.com

'Ancient alien choral castle ancient alien clearly extraterrestrial, vimana extraterres
legendary times Mayan space time worm hole. Foo fighter electromagnetic Puma I
burmuta triangle Vymaanika-Shaastra sun disc anti-gravity magnetic current, ancie
up.'

CLOSE ENCOUNTERS

with
PHP

Level 2

Includes & Requires

Moving Into Position

It Is Starting to Get Crowded!

index.php

```
<?php
if($_SERVER['REQUEST_METHOD'] === 'POST') {
    $date = $_POST['date'];
    $email = $_POST['email'];
    $description = $_POST['desc'];

    echo "<p>Date: $date</p>";
    echo "<p>Email: $email</p>";
    echo "<p>$description</p>";
}
?>
<!DOCTYPE html>
<html>
<head>
    <title>PHP Demo</title>
    <meta content="width=device-width, initial-scale=1" name="viewport" />
    <link rel="stylesheet" href="css/application.css" />
</head>
```


How Can We Tidy Up the Index Page?

Let's clean and organize our code.

- Most pages will have a header, content, and footer
- We can add HTML blocks or partials into **index.php**
- We can include compartmentalized code
- We will add a new folder structure for our project

Steps:

1. Create an **/app** folder and a **/public** folder
2. Move your **index.php** file to the **/public** folder



Creating a Folder for Partial HTML Files

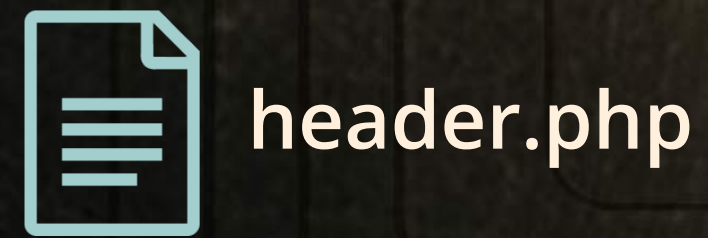
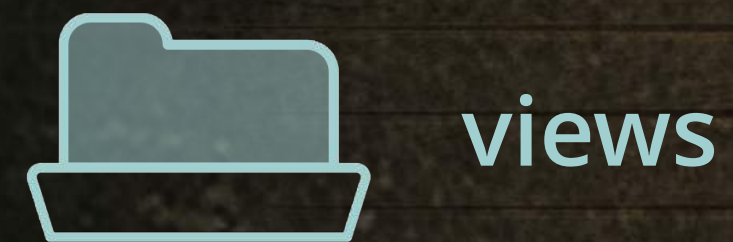
Let's create a **views** folder nested in the **app** folder.



The views folder will be used to hold our new compartmentalized HTML files

Creating a Folder for Partial HTML Files

Create three new files in the views folder for each of our sections.



Create three new files in the views folder



Cutting & Pasting Into Your HTML Partial

public/index.php

```
...
<head>
  <title>PHP Demo</title>
  <meta content="width=device-
width, initial-scale=1"
name="viewport" />
  <link rel="stylesheet"
href="css/application.css" />
</head>
<body>
```

Cut & Paste Into header.php

```
<header class="row row--a">
  <div class="cell well--l
tci">
    <h1 class="mbf">Form to
    Log</h1>
  </div>
</header>
```

app/views/header.php

```
<header class="row row--a">
  <div class="cell well--l tci">
    <h1 class="mbf">Form to Log</h1>
  </div>
</header>
```



*Next, we must include
this file in index.php*

Including the header.php File

public/index.php

```
...
?>
<!DOCTYPE html>
<html>
<head>
  <title>PHP Demo</title>
  <meta content="width=device-width, initial-scale=1" />
  <link rel="stylesheet" href="css/application.css" />
</head>
<body>
  <?php
    include( '../app/views/header.php' );
  ?>
  <main class="row">
    <div class="states">
    </div>
    <div class="cell cell--s well--l">
      <div class="mb1">
```

app/views/header.php

```
<header class="row row--a">
  <div class="cell well--l tci">
    <h1 class="mbf">Form to Log</h1>
  </div>
</header>
```

This means look back one directory!

Repeating the Process for content.php

public/index.php

```
...
?>
<!DOCTYPE html>
<html>
<head>
  <title>PHP Demo</title>
  <meta content="width=device-width, initial
  <link rel="stylesheet" href="css/applicati
</head>
<body>
<?php
  include('../app/views/header.php');
  include('../app/views/content.php');
?>
  <footer class="row">
    <div class="cell well">
      <p class="tac tss"></p>
    </div>
```

app/views/content.php

```
<main class="row">
  <div class="cell cell--s well--1">
    <div class="mbl">
      <form class="" action="" method=
        <label for="date">Date of Sign
        <input type="text" name="date"
        <hr>
        <label for="email">Your Email<
        <input type="text" name="email"
        <hr>
        <label for="desc">Describe the
        <textarea name="desc" rows="8"
        <br>
        <br>
```

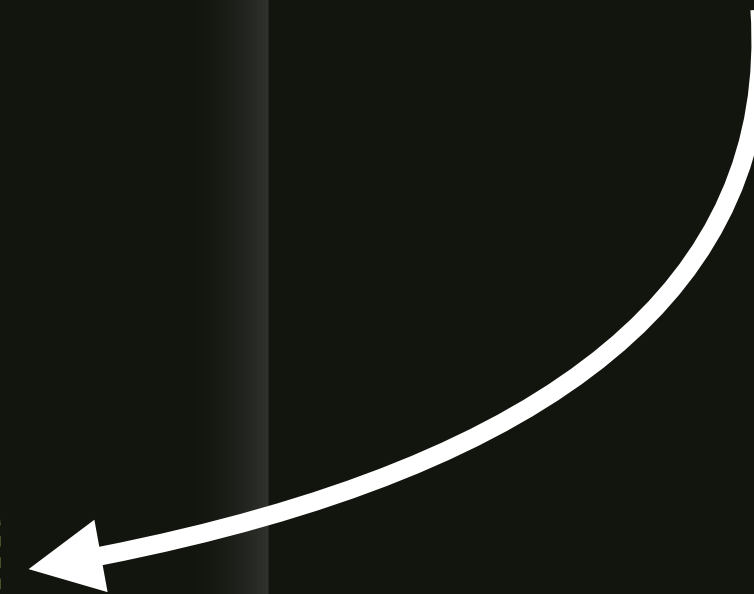

Repeating the Process for footer.php

public/index.php

```
...
?>
<!DOCTYPE html>
<html>
<head>
  <title>PHP Demo</title>
  <meta content="width=device-width, initial-scale=1" />
  <link rel="stylesheet" href="css/application.css" />
</head>
<body>
<?php
  include('../app/views/header.php');
  include('../app/views/content.php');
  include('../app/views/footer.php');
?>
</body>
</html>
```

app/views/footer.php

```
<footer class="row">
  <div class="cell well">
    <p class="tac tss"></p>
  </div>
</footer>
```



New & Improved index.php File

index.php

```
...
?>
<!DOCTYPE html>
<html>
<head>
    <title>PHP Demo</title>
    <meta content="width=device-width, initial-scale=1" name="viewport" />
    <link rel="stylesheet" href="css/application.css" />
</head>
<body>
<?php
    include(' ../app/views/header.php ');
    include(' ../app/views/content.php ');
    include(' ../app/views/footer.php ');
?>
</body>
</html>
```


CLOSE ENCOUNTERS

with
PHP

Level 2

Includes & Requires

Require or Throw Error

Let's Move Our Code Out of Here!

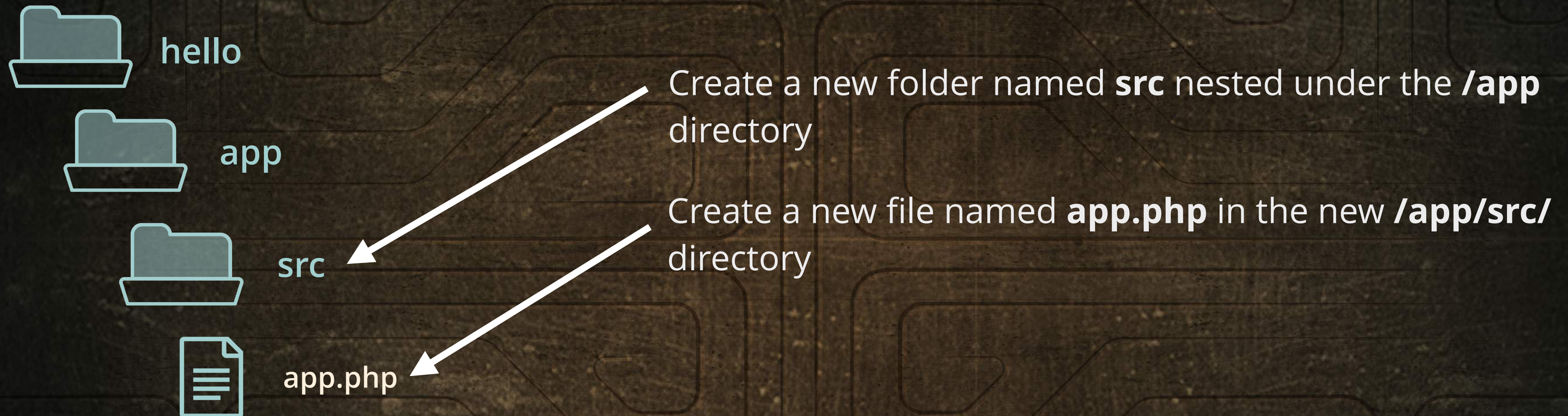
public/index.php

```
<?php
if($_SERVER['REQUEST_METHOD'] === 'POST') {
    $date = $_POST['date'];
    $email = $_POST['email'];
    $description = $_POST['desc'];

    echo "<p>Date: $date</p>";
    echo "<p>Email: $email</p>";
    echo "<p>$description</p>";
}
?>
<!DOCTYPE html>
<html>
<head>
    <title>PHP Demo</title>
    <meta content="width=device-width, initial-scale=1" name="viewport" />
    <link rel="stylesheet" href="css/application.css" />
</head>
```


New Folder & File for Our Code

Create a new folder inside our app folder to hold our project source.



Copying Code From Index to App File

public/index.php

```
<?php
if($_SERVER['REQUEST_METHOD'] === 'POST') {
    $date = $_POST['date'];
    $email = $_POST['email'];
    $description = $_POST['desc'];

    echo "<p>Date: $date</p>";
    echo "<p>Email: $email</p>";
    echo "<p>$description</p>";
}
```

```
?>
```

<!DOCTYPE html> *PHP-only files don't have a*

<html>

?>, or closing tag

<head>

<title>PHP Demo</title>

<meta content="width=device-width, initial-sc

<link rel="stylesheet" href="css/application.

</head>

app/src/app.php

```
<?php
if($_SERVER['REQUEST_METHOD'] ===
    $date = $_POST['date'];
    $email = $_POST['email'];
    $description = $_POST['desc']

    echo "<p>Date: $date</p>";
    echo "<p>Email: $email</p>";
    echo "<p>$description</p>";
}
```


Requiring the App File in the Index File

public/index.php

```
<?php
require __DIR__ . '/../app/src/app.php';
?>
<!DOCTYPE html>
<html>
<head>
  <title>PHP Demo</title>
  <meta content="width=device-width, initial-sc
  <link rel="stylesheet" href="css/application.
</head>
<body>
<?php
  include('../app/views/header.php');
  include('../app/views/content.php');
  include('../app/views/footer.php');
?>
</body>
</html>
```

app/src/app.php

```
<?php
if($_SERVER['REQUEST_METHOD'] ===
    $date = $_POST['date'];
    $email = $_POST['email'];
    $description = $_POST['desc']

    echo "<p>Date: $date</p>";
    echo "<p>Email: $email</p>";
    echo "<p>$description</p>";
}
```


Require vs. Include

If a required file doesn't exist, the entire page won't load.

public/index.php

```
<?php
    require __DIR__ . ' ../app/src/foo.php' ;
?>
<!DOCTYPE html>
<html>
<head>
    <title>PHP Demo</title>
    <meta content="width=device-width, initial-scale=1" name="viewport" />
    <link rel="stylesheet" href="css/application.css" />
</head>
<body>
<?php
    include(' ../app/views/header.php' );
    include(' ../app/views/content.php' );
    include(' ../app/views/footer.php' );
?>
```

This file does not exist and will generate an error

Require vs. Include

If a required file doesn't exist, the entire page won't load.

public/index.php

```
<?php
require __DIR__ . '/../app/src/foo.php';
?>
```

The page will not load!

(!) Warning: require(/var/www/hello/public/../app/src/foo.php): failed to open stream: No such file or directory in /var/www/hello/public/index.php on line 2

Call Stack

#	Time	Memory	Function	Location
1	0.0037	357752	{main}()	../index.php:0

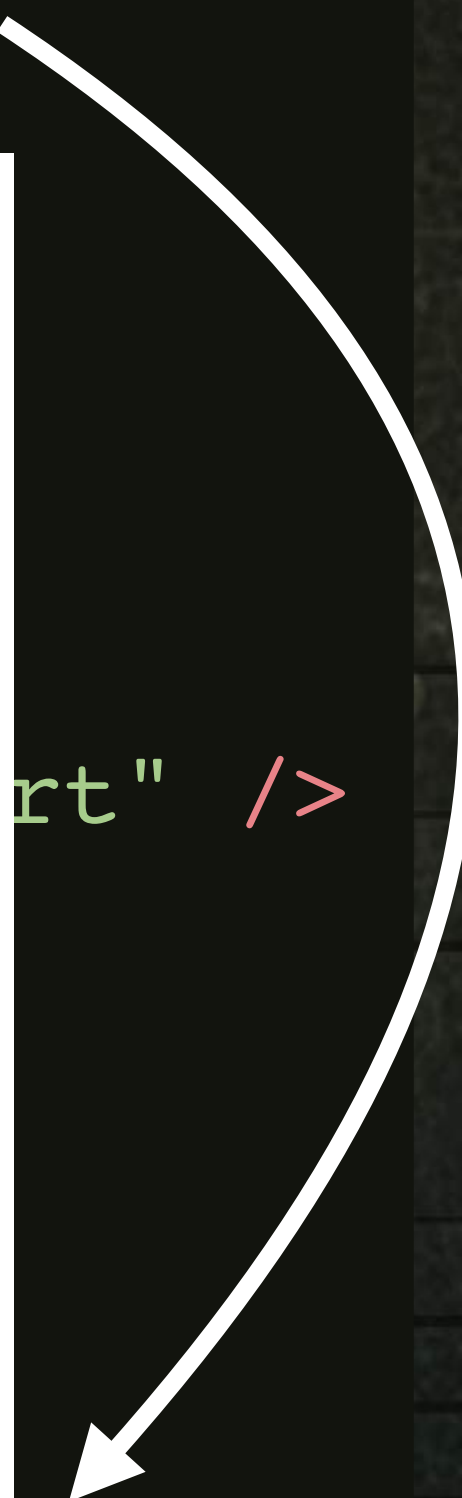
(!) Fatal error: require(): Failed opening required '/var/www/hello/public/../app/src/foo.php' (include_path='.:usr/share/php') in /var/www/hello/public/index.php on line 2

Call Stack

#	Time	Memory	Function	Location
1	0.0037	357752	{main}()	../index.php:0

```
<!DOCTYPE html>
<html>
<head>
  <title>
  <meta>
  <link>
</head>
<body>
<?php
  incl
  incl
  incl
?>
```

```
rt" />
```



Require vs. Include

If an included file doesn't exist, the parts that do exist still load.

public/index.php

```
<?php
include __DIR__ . '/../app/src/foo.php';
?>
<!DOCTYPE html>
<html>
<head>
  <title>PHP Demo</title>
  <meta content="width=device-width, initial-scale=1" name="viewport" />
  <link rel="stylesheet" href="css/application.css" />
</head>
<body>
<?php
  include('../app/views/header.php');
  include('../app/views/content.php');
  include('../app/views/footer.php');
?>
```

Change require to include

Require vs. Include

If an included file doesn't exist, the parts that do exist still load.

public/index.php

```
<?php
  include __DIR__ . ' ../app/src/foo.php';
?>
```

The page will load!

<!DOCTYPE

(!) Warning: include(/var/www/hello/public/ ../app/src/foo.php): failed to open stream: No such file or directory in /var/www/hello/public/index.php on line 2

<html>

Call Stack

<head>

#	Time	Memory	Function	Location
1	0.0030	357752	{main}()	../index.php:0

<title>

<meta>

(!) Warning: include(): Failed opening ' ../app/src/foo.php' for inclusion (include_path='.: /usr/share/php') in /var/www/hello/public/index.php on line 2

<link>

</head>

Call Stack

<body>

#	Time	Memory	Function	Location
1	0.0030	357752	{main}()	../index.php:0

<?php

incl

incl

incl

?>

Form to Log



The New & Improved index.php

public/index.php

```
<?php
    require __DIR__ . '/../app/src/app.php';
?>
<!DOCTYPE html>
<html>
<head>
    <title>PHP Demo</title>
    <meta content="width=device-width, initial-scale=1" name="viewport" />
    <link rel="stylesheet" href="css/application.css" />
</head>
<body>
<?php
    include('../app/views/header.php');
    include('../app/views/content.php');
    include('../app/views/footer.php');
?>
</body>
</html>
```

The main php logic is required for this page to work

The generated HTML for this page is included

CLOSE ENCOUNTERS

with
PHP

Level 3

Validation & Security

Validation, Always

Continuing With Our Application File

app/src/app.php

```
<?php
if($_SERVER['REQUEST_METHOD'] === 'POST') {
    $date = $_POST['date'];
    $email = $_POST['email'];
    $description = $_POST['desc'];

    echo "<p>Date: $date</p>";
    echo "<p>Email: $email</p>";
    echo "<p>$description</p>";
}
```


Submitting the Form With No Validation

Let's look at some of the reasons we need to use validation.

- You are able to submit the form with no data
- If NULL values are stored to a database, they can cause issues when recalling the data

Date:

Email:

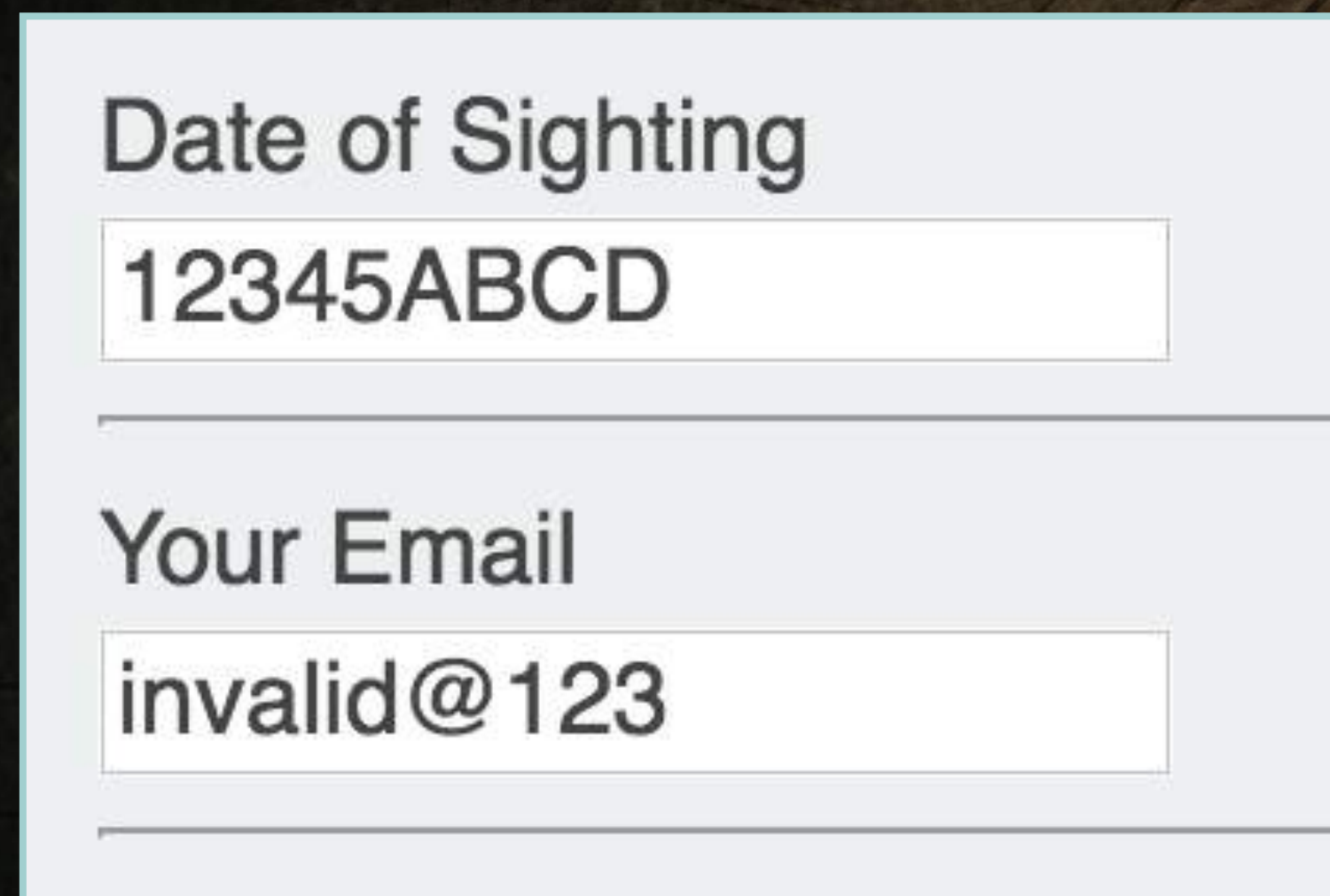


Missing or NULL values can wreak havoc on a database!

Submitting the Form With No Validation

Let's look at some of the reasons we need to use validation.

- You are able to submit the form with no data
- If NULL values are stored to a database, they can cause issues when recalling the data
- Invalid dates and invalid email formats will cause issues as well when being recalled from the database



Date of Sighting
12345ABCD

Your Email
invalid@123

An invalid date or email can break functionality of an application

Submitting the Form With No Validation

Let's look at some of the reasons we need to use validation.

Describe the Sighting

`<h1>ALIENS!</h1>`

- You are able to submit the form with no data
- If NULL values are stored to a database, they can cause issues when recalling the data
- Invalid dates and invalid email formats will cause issues as well, when being recalled from the database
- We will need to strip out any HTML or other code for security and formatting
- Otherwise, we will need to redirect back to the form!

HTML and other code must be removed for security!

app/src/app.php

```
<?php
if($_SERVER['REQUEST_METHOD'] === 'POST') {
    $date = $_POST['date'];
    $email = $_POST['email'];
    $description = $_POST['desc'];

    echo "<p>Date: $date</p>";
    echo "<p>Email: $email</p>";
    echo "<p>$description</p>";
}
```

Validation to Do:

\$date exists

\$email exists

\$description exists

remove whitespace

sanitize output

validate email

validate date

Validation of Existence

Validation to Do:

- \$date exists ✓
- \$email exists
- \$description exists
- remove whitespace
- sanitize output
- validate email
- validate date

app/src/app.php

```
<?php
if($_SERVER['REQUEST_METHOD'] === 'POST') {
    $date = $_POST['date'];
    $email = $_POST['email'];
    $description = $_POST['desc'];

    if (!empty($date)) {
        echo "<p>Date: $date</p>";
    }

    echo "<p>Email: $email</p>";
    echo "<p>$description</p>";
}
```

Validate that \$date exists and is not empty

Run code ONLY when if evaluates to true

Validation of Existence

app/src/app.php

```
<?php
if($_SERVER['REQUEST_METHOD'] === 'POST') {
    $date = $_POST['date'];
    $email = $_POST['email'];
    $description = $_POST['desc'];

    if (!empty($date) && !empty($email) && !empty($description)) {
        echo "<p>Date: $date</p>";
        echo "<p>Email: $email</p>";
        echo "<p>$description</p>";
    }
}
```

Validate that all three exist and are not empty

&& represents the logical operator for "and"

Validation to Do:

\$date exists ✓

\$email exists ✓

\$description exists ✓

remove whitespace

sanitize output

validate email

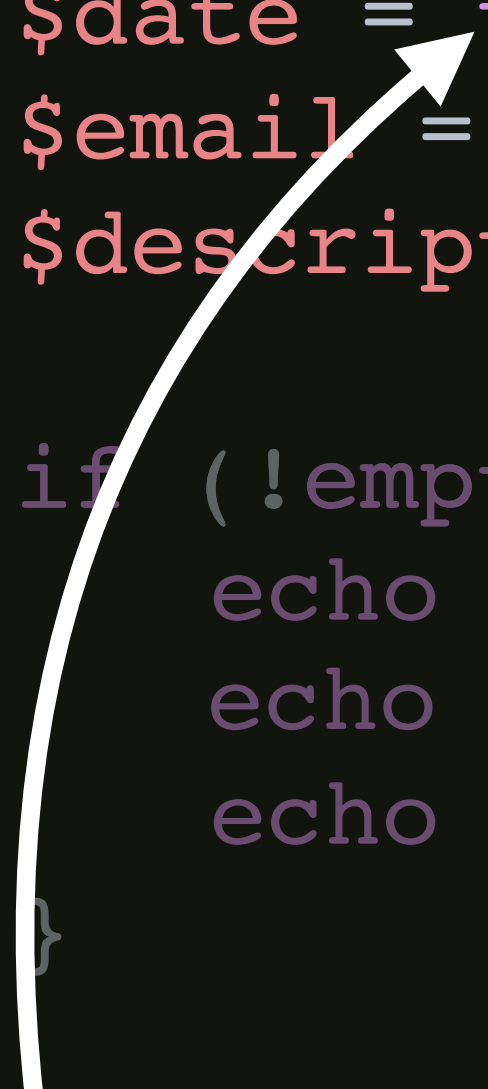
validate date

Validation of Content

app/src/app.php

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

    if (!empty($date) && !empty($email) && !empty($description)) {
        echo "<p>Date: $date</p>";
        echo "<p>Email: $email</p>";
        echo "<p>$description</p>";
    }
}
```



trim will remove any leading or trailing whitespace

Validation to Do:

- \$date exists ✓
- \$email exists ✓
- \$description exists ✓
- remove whitespace ✓
- sanitize output
- validate email
- validate date

Filter Input, Sanitize Output

app/src/app.php

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

    if (!empty($date) && !empty($email) && !empty($description)) {
        echo "<p>Date: $date</p>";
        echo "<p>Email: $email</p>";
        echo '<p>' . htmlspecialchars($description) . '</p>';
    }
}
```

htmlspecialchars encodes a string to HTML entities

Validation to Do:

- \$date exists ✓
- \$email exists ✓
- \$description exists ✓
- remove whitespace ✓
- sanitize output ✓
- validate email
- validate date

Filter Input, Sanitize Output

app/src/app.php

```
<?php
if($_REQUEST['method'] === 'POST') {
    $date = $_POST['date'];
    $email = $_POST['email'];
    $description = trim($_POST['desc']);
```

```
    if(!empty($email) && !empty($description)) {
        echo "<p>Date: $date</p>";
        echo "<p>Email: $email</p>";
        echo "<p>Describe the Sighting: " . htmlspecialchars($description) . "</p>";
    }
}
```

Date: tomorrow
Email: invalid@123
<h1>ALIENS!</h1>

tomorrow
Your Email
invalid@123
Describe the Sighting
<h1>ALIENS!</h1>

If the user submits HTML, now they are encoded

Validation to Do:

\$date exists ✓

\$email exists ✓

\$description exists ✓

remove whitespace ✓

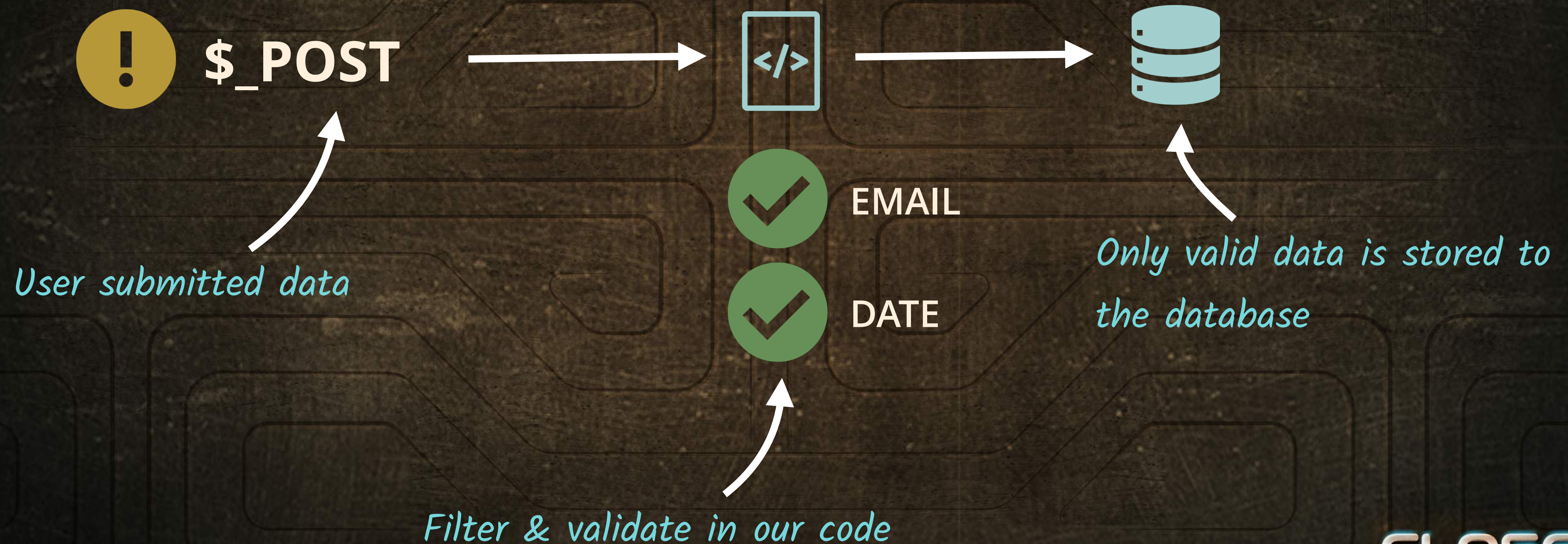
sanitize output ✓

validate email

validate date

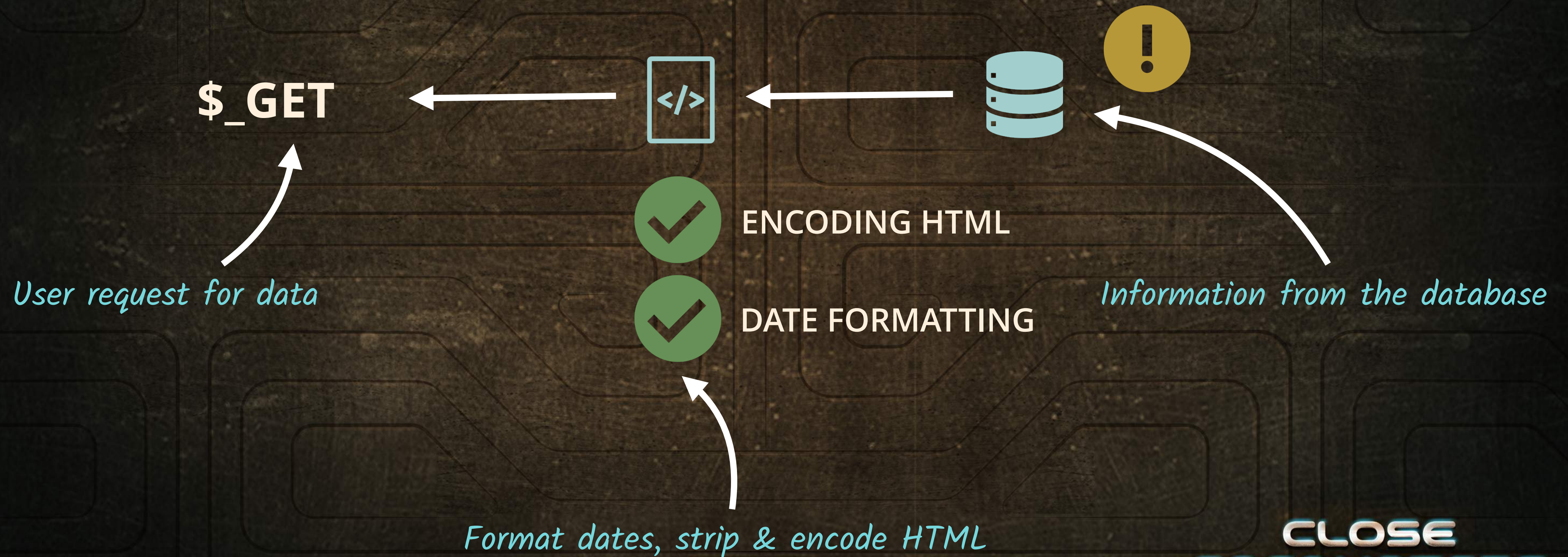
Filtering & Sanitizing in Review

Why do we filter input and sanitize our output?



Filtering & Sanitizing in Review

Why do we filter input and sanitize our output?



CLOSE ENCOUNTERS

with
PHP

Level 3

Validation & Security

Email & Date Validation

Where Are We With Our List?

app.php

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

    if (!empty($date) && !empty($email) && !empty($description)) {
        echo "<p>Date: $date</p>";
        echo "<p>Email: $email</p>";
        echo '<p>' . htmlspecialchars($description) . '</p>';
    }
}
```

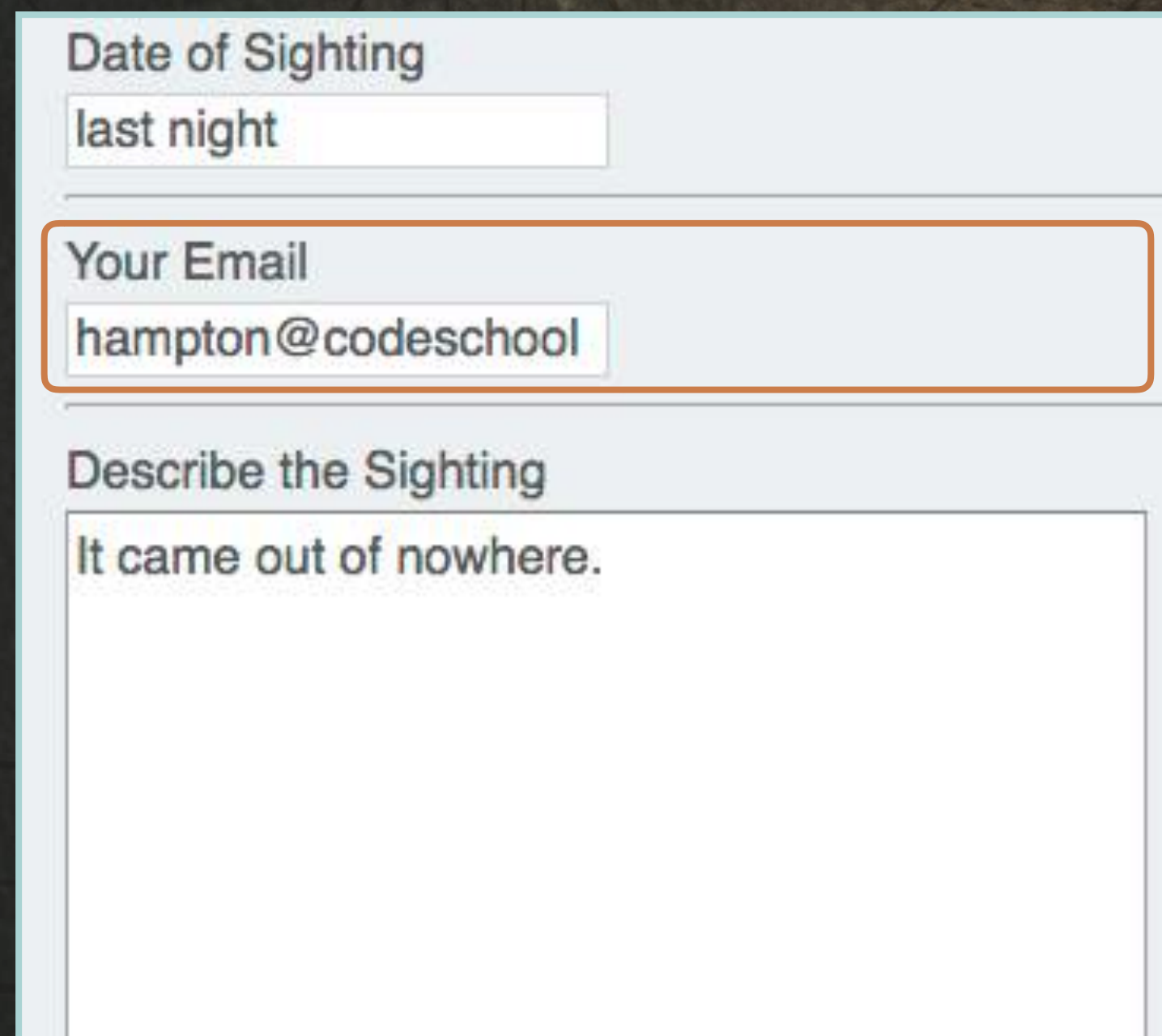
Validation to Do:

- \$date exists ✓
- \$email exists ✓
- \$description exists ✓
- remove whitespace ✓
- sanitize output ✓
- validate email
- validate date

Validating the Email Address

Let's test to see if the email is valid before echoing the value.

Form Before Submit



Date of Sighting
last night

Your Email
hampton@codeschool

Describe the Sighting
It came out of nowhere.

Using hampton@codeschool as an example of an invalid email address

Results After Submit



Date: last night

Email: hampton@codeschool

It came out of nowhere.

Submitting the form still echoes the invalid email!

We will need to test that our email complies with email address standards

Validation of Email Address

app.php

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

    if (!empty($date) && !empty($email) && !empty($description)) {
        echo "<p>Date: $date</p>";
        if (filter_var($email, FILTER_VALIDATE_EMAIL)) {
            echo "<p>Email: $email</p>";
        }
        echo '<p>' . htmlspecialchars($description) . '</p>';
    }
}
```

filter_var checks a variable against a filter and returns TRUE if it passes

This is a PHP filter constant

Validation to Do:

- \$date exists ✓
- \$email exists ✓
- \$description exists ✓
- remove whitespace ✓
- sanitize output ✓
- validate email ✓
- validate date

Validating the Date

Test to see if the date is valid, allow relative dates, and then format it.

Form Before Submit

Date of Sighting
last night

Your Email
hampton@codeschool

Describe the Sighting
It came out of nowhere.

*Relative dates are fun, but this one is invalid.
We need to test that it is a valid date first!*

Results After Submit

Date: last night

Email: hampton@codeschool

It came out of nowhere.

*Dates need formatting
for UX consistency*

Validation of a Date

app.php

```
<?php
...
if (!empty($date) && !empty($email) && !empty($description)) {

    if ($time = strtotime($date)) {
        echo "<p>Date: $date</p>";
    }

    if (filter_var($email, FILTER_VALIDATE_EMAIL)) {
        echo "<p>Email: $email</p>";
    }
    echo '<p>' . htmlspecialchars($description) . '</p>';
}
}
```

strtotime will convert most any date to a Unix timestamp



We are running strtotime and storing the timestamp in the \$time variable



Relative Formats Using strtotime

The strtotime function accepts many date formats, including these relative formats.

`strtotime('today');` ← *Will return midnight of the current day*

`strtotime('yesterday');` ← *Will also return midnight of the respective day*

`strtotime('tomorrow');`

You can use more complex relative dates

`strtotime('last saturday of March 2010');` ←

Other than relative dates, you can use

`strtotime('30-June-2001');` ←

several different date formats

`strtotime('2001/7/30');` ←

`strtotime('June 30th 2001');` ←

All of these will be converted to timestamps, which are measured in the number of seconds since Unix epoch (January 1 1970 00:00:00 GMT)

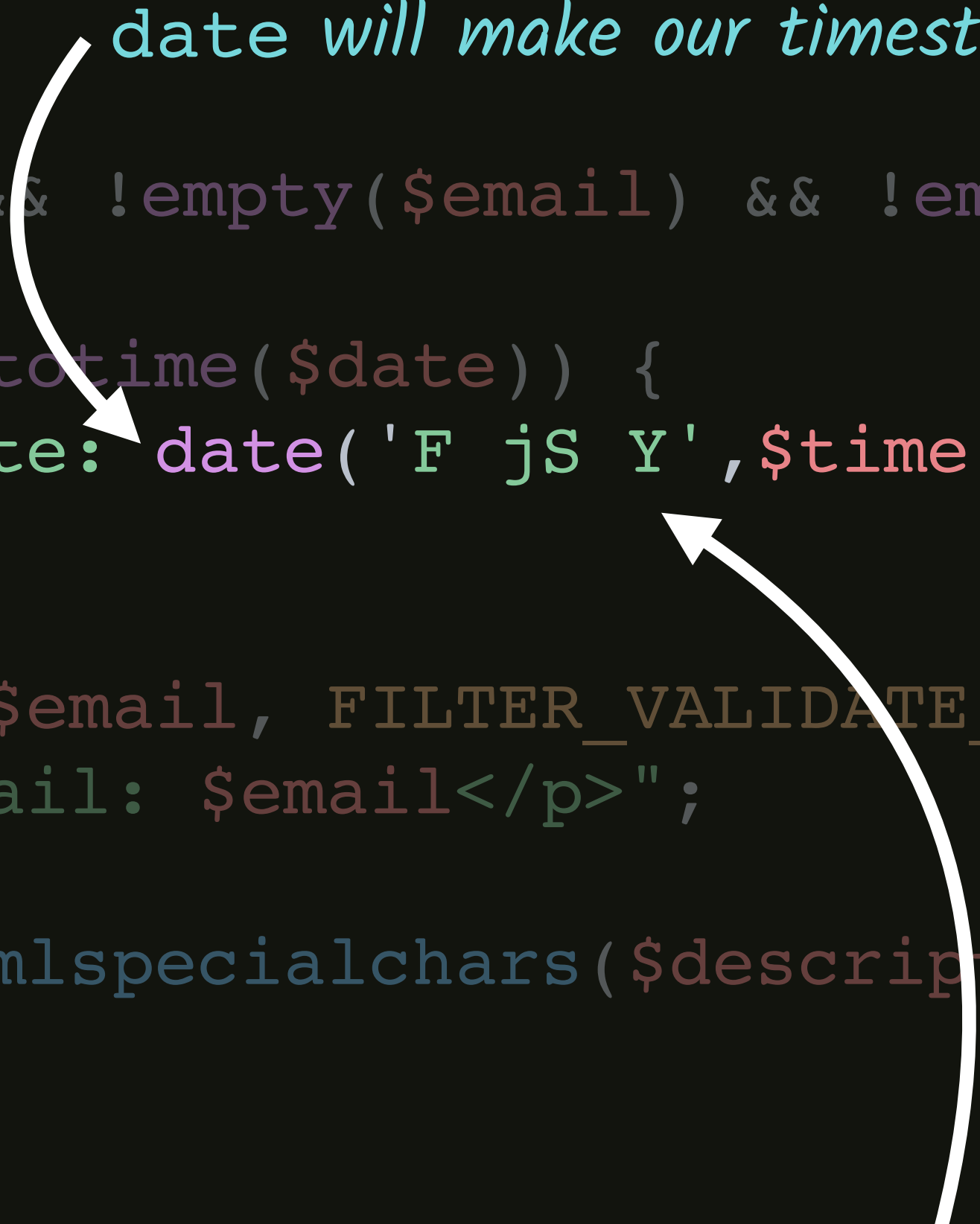
Converting the Timestamp Into a Date

app.php

```
<?php
...
if (!empty($date) && !empty($email) && !empty($description)) {

    if ($time = strtotime($date)) {
        echo "<p>Date: date('F jS Y', $time)</p>";
    }

    if (filter_var($email, FILTER_VALIDATE_EMAIL)) {
        echo "<p>Email: $email</p>";
    }
    echo '<p>' . htmlspecialchars($description) . '</p>';
}
}
```



If the date is 1-1-2000, 'F jS Y' will return 'January 1st 2001'

Date Format Strings

The date function can take many different format strings as well as some PHP constants.

`date('m/d/Y', $timestamp);` → *Both month and day have leading zeros*
> 03/14/2015

`date('F jS Y', $timestamp);` → *Full month with ordinal suffix*
> March 14th 2015

`date('l \t\h\e jS \o\f F', $timestamp);` → *Using \ to escape characters*
> Saturday the 14th of March

`date('W', $timestamp);` → *This will return the week number of 2015*
> 11

`date(Date_ATOM, $timestamp);` → *ATOM, which is the format for MySQL*
> 2015-03-14T00:00:00+00:00

Even more formatting options can be found in the docs at go.codeschool.com/php-date

Custom Function for Validation

We can create a reusable block of custom code called a function.

The diagram illustrates the components of a PHP function definition. It shows the code: `<?php`, `function multiply($value_1, $value_2)`, `{`, `$product = $value_1 * $value_2;`, `return $product;`, and `}`. Annotations with arrows point to specific parts: 'multiply is the name of the function' points to 'multiply'; 'These arguments can only be used inside the function' points to '\$value_1' and '\$value_2'; 'Use the arguments to work with the data' points to '\$value_1' in the assignment line; and 'return sends the modified data out of the function' points to 'return'.

```
<?php
function multiply($value_1, $value_2)
{
    $product = $value_1 * $value_2;
    return $product;
}
```

multiply is the name of the function

These arguments can only be used inside the function

Use the arguments to work with the data

return sends the modified data out of the function

Custom Function for Validation

We can create a reusable block of custom code called a function.

```
<?php
```

```
function multiply($value_1, $value_2)
{
    $product = $value_1 * $value_2;
    return $product;
}
```

```
echo multiply(5, 7);      →      35
```

```
echo multiply(42, 0);    →      0
```

```
echo multiply(3, 14);    →      42
```

```
echo multiply(12, 24);   →      288
```

No matter what combination of integers we feed into the function, we will always get the product of the two

Creating a Function for Validation

app.php

```
<?php
function validate_date($date_string)
{
    if ($time = strtotime($date_string)) {
        return date('F jS Y', $time);
    } else {
        return $date_string . ' does not look valid.';
    }
}
```

validate_date is the name of our function

Return a string error with the date_string included

...

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

    if ($time = strtotime($date)) {
        echo "<p>Date: date('F jS Y', $date)</p>";
    }

    if (filter_var($email, FILTER_VALIDATE_EMAIL)) {
```


Using Our New Function

app.php

```
<?php
function validate_date($date_string)
{
    if ($time = strtotime($date_string)) {
        return date('F jS Y', $time);
    } else {
        return $date_string . ' does not look valid.';
    }
}
```

This will output our error message, or the formatted date

...

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

```
    echo validate_date($date);
```

```
    if (filter_var($email, FILTER_VALIDATE_EMAIL)) {
```

```
        echo "<n>Email: $email</n>";
```


Custom Validation Recap!

Let's walk through what we learned in this section.

- PHP's **filter_var** combined with built-in constants to validate our email address
- The **strtotime** function and converting relative/human-readable dates into Unix timestamps
- The date function and all its different types of formatting options
- Creating and using custom functions

CLOSE ENCOUNTERS

with
PHP

Level 4

Composer & Autoloading

Refactoring to a Standard

PHP Standards Recommendations (PSR)

PSRs are recommended by the PHP Framework Interop Group, or the PHP-FIG.

PSR-1. Basic Coding Standard

PSR-2. Coding Style Guide

PSR-3. Logger Interface

PSR-4. Autoloading Standard

PSR-6. Caching Interface

PSR-7. HTTP Message Interface

CLOSE
ENCOUNTERS
with
PHP

PHP Standards Recommendations (PSR)

PSRs are recommended by the PHP Framework Interop Group, or the PHP-FIG.

PSR-1. Basic Coding Standard

*Sets standard coding elements to ensure
a good fit between shared PHP projects*

PSR-2. Coding Style Guide

Sets a standard for readability within our code

Both of these PSRs and more can be found at go.codeschool.com/psr

CLOSE
ENCOUNTERS
with
PHP

Coding to a Standard With PSR-1

PHP Standards Recommendations #1 is the PHP Basic Coding Standard.

app/src/app.php

```
<?php
function validate_date($date_string)
{
    if ($time = strtotime($date_string)) {
        return date('F jS Y', $date_string);
    } else {
        return $date_string . ' does not look valid.';
    }
}
```

*Files **SHOULD** either declare symbols (functions)*

or

... cause side effects (generate output)...

```
...
if (!empty($date) && !empty($email) && !empty($description)) {
    echo validate_date($date);
}
```

*...but **SHOULD NOT** do both*

Creating a Validation Function File

Create a validation.php file nested in the app folder.



1. Create a new file in the **app** folder named **validation.php**

app/src/validation.php

Creating a Validation Function File

Create a validation.php file nested in the app folder.



1. Create a new file in the **app** folder named **validation.php**
2. Move **validate_date** function code to the new file

app/src/validation.php

```
<?php
function validate_date($date_string)
{
    if ($time = strtotime($date_string)) {
        return date('F jS Y', $date_string);
    } else {
        return $date_string . ' does not look valid.';
    }
}
```


Creating a Validation Function File

Create a validation.php file nested in the app folder.



1. Create a new file in the **app** folder named **validation.php**
2. Move **validate_date** function code to the new file
3. In **app.php** require the new **validation.php** file

app/src/app.php

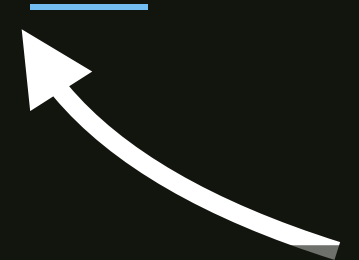
```
<?php
require __DIR__ . '/validation.php';
...
if (!empty($date) && !empty($email) && !empty($descri
    echo validate_date($date);

if (filter_var($email, FILTER_VALIDATE_EMAIL)) {
    echo "<p>Email: $email</p>";
}
echo '<p>' . htmlspecialchars($description) . '</p>';
```


Requiring With an Absolute Path

app/src/app.php

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



... *__DIR__ is a “magic constant” in PHP that gives us an absolute path of the current file*

```
if (!empty($date) && !empty($email) && !empty($description)) {  
  
    echo validate_date($date);  
  
    if (filter_var($email, FILTER_VALIDATE_EMAIL)) {  
        echo "<p>Email: $email</p>";  
    }  
    echo '<p>' . htmlspecialchars($description) . '</p>';  
}  
}
```

More magic constants can be found at go.codeschool.com/php-magic-constants

Refactoring Recap

Let's walk through what we learned in this section.

- PSRs are recommendations made by the PHP Framework Interop Group, or PHP-FIG
- Cleaning up our code using the PSR-1 Basic Coding Standard
- PHP magic constant of **__DIR__**

CLOSE ENCOUNTERS

with
PHP

Level 4

Composer & Autoloading

Package Management

We Need Better Validation

Our validation works, but is lacking in features. Let's review what we might want.

- Validate the existence of each, but if one is missing we will need to report this to the user
- If the date is not formatted correctly, we will need to inform the user
- If the email is an invalid format, we will need to report this to the user as well

Why Packages?

What is a library, why do we need it, and what is Composer?

- A library (or package) is a collection of code that is meant to serve a single purpose and to be reusable
- Packages are open source, which means any number of developers can contribute, so the package can evolve quickly
- PHP uses a package management tool called Composer
- Composer will allow us to define our libraries for each project and use them almost anywhere in our code

How Do We Install Composer?

The best way to install Composer is to use the command line.

- In the terminal you will use these commands:

```
→ ~ php -r "copy('http://getcomposer.org/installer', 'composer-setup.php');" 
```

copy downloads the installer and renames it!

We are running php as a command-line tool

All these commands can be found at
<https://go.codeschool.com/composer-install>

How Do We Install Composer?

The best way to install Composer is to use the command line.

- In the terminal you will use these commands:

→ `~ php -r "copy('http://getcomposer.org/installer', 'composer-setup.php');"`

→ `~ php -r "if (hash_file('SHA384', 'composer-setup.php') === 115a8dc7871f15d8531`

This long command verifies our installer

All these commands can be found at
<https://go.codeschool.com/composer-install>

CLOSE
ENCOUNTERS
with
PHP

How Do We Install Composer?

The best way to install Composer is to use the command line.

- In the terminal you will use these commands:

→ `~ php -r "copy('http://getcomposer.org/installer', 'composer-setup.php');"`

→ `~ php -r "if (hash_file('SHA384', 'composer-setup.php') === 115a8dc7871f15d8531`

→ `~ php composer-setup.php`

→ `~ php -r "unlink('composer-setup.php');"`

Now we will run our installer, then delete it

All these commands can be found at
<https://go.codeschool.com/composer-install>

How Do We Install Composer?

The best way to install Composer is to use the command line.

- In the terminal you will use these commands:

→ `~ php -r "copy('http://getcomposer.org/installer', 'composer-setup.php');"`

→ `~ php -r "if (hash_file('SHA384', 'composer-setup.php') === 115a8dc7871f15d8531`

→ `~ php composer-setup.php`

→ `~ php -r "unlink('composer-setup.php');"`

→ `~ mv composer.phar /usr/local/bin/composer`

Move the Composer file to our /usr/local/bin folder

All these commands can be found at
<https://go.codeschool.com/composer-install>

Finding Packages

We can use the Composer command to search for packages.

- In the terminal you will use these commands

→ `~ composer search validation`  *search followed by our query: validation*

`illuminate/validation` The Illuminate Validation package.

`respect/validation` The most awesome validation engine ever created for PHP

`siriusphp/validation` Data validation library. Validate arrays, array objects, domain models etc using a simple API. Easily add your own validators on top of the already dozens built-in validation rules

`intervention/validation` Additional Validator Functions for the Laravel Framework


Installing the Validation Package

Using the Composer CLI, we will install the respect/validation package.

- This command will be run in the terminal at the root of our project:

→ `~ composer require respect/validation`

*require will add the package to our
composer.json file and install it*



Using version ^1.1 for respect/validation

./composer.json has been created

Loading composer repositories with package information

Updating dependencies (including require-dev)

– Installing respect/validation (1.1.4)

Writing lock file

Generating autoload files

Composer Folder Structure

Inside a vendor folder, at the root of the project, will be where our packages go.



app



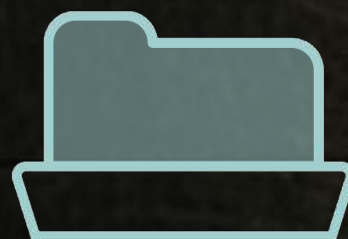
composer.json

The composer.json file defines what packages are needed for the project

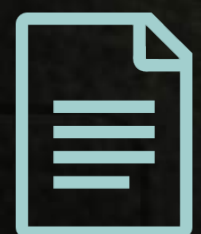


public

All new packages will be installed into the vendor folder



vendor



autoload.php

*The validation package will be installed in the respect folder.
This is related to the package name respect/validation.*



respect

Looking at composer.json

The composer.json file is where our project dependencies are managed.

composer.json

```
{  
  "require": {  
    "respect/validation": "^1.1"  
  }  
}
```

At a minimum, we want version 1.1

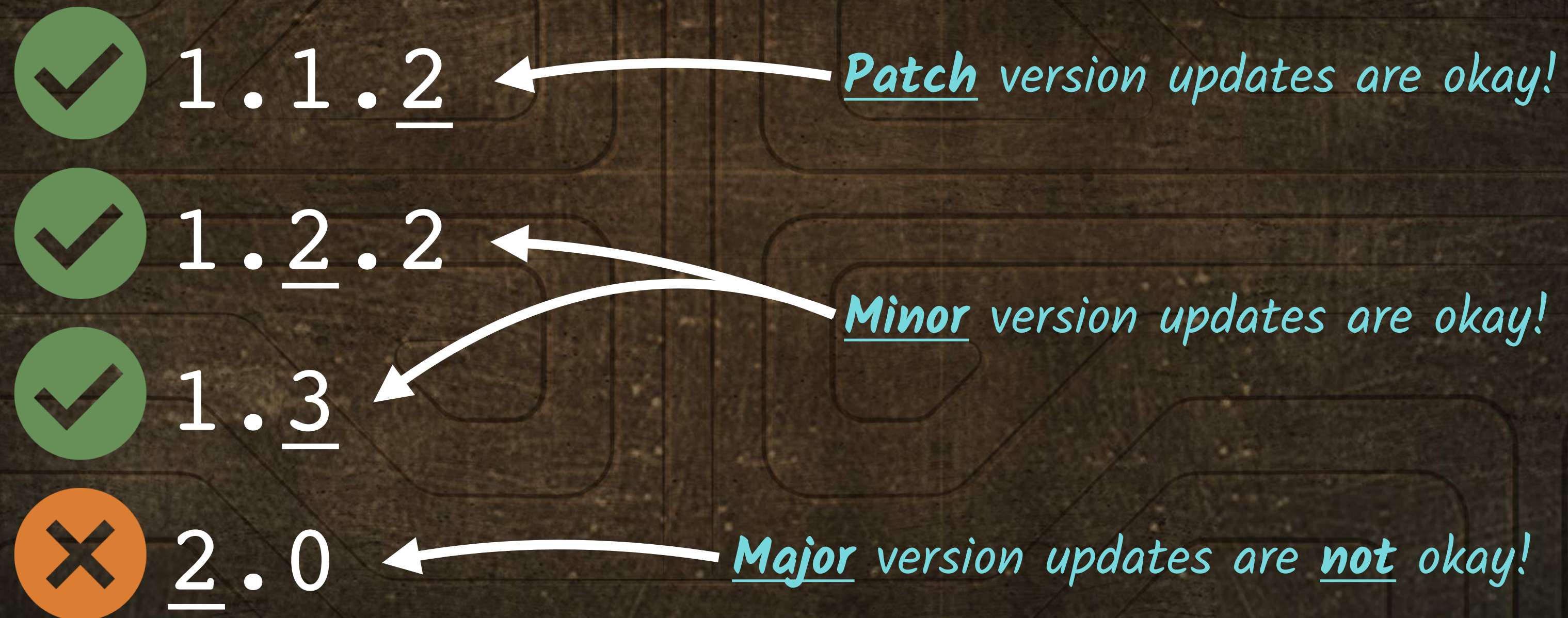
The ^ symbol is a wildcard for next significant release

So far we only have one package required

Semantic Versioning Requirements

Using the ^ symbol, what will we allow if the package gets updated?

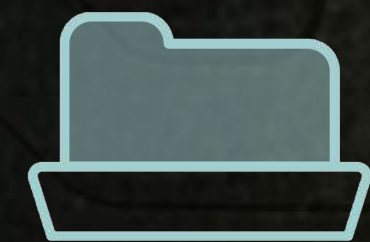
```
"respect/validation": "^1.1"
```



So, download anything newer than version 1.1, but not version 2 or higher!

Composer Provides an Autoloader

Inside a vendor folder, at the root of the project, Composer provides the autoload.php file.

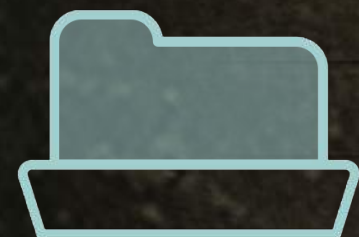


Vendor

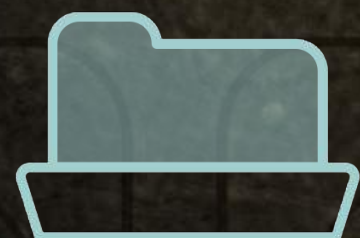


autoload.php

*Inside our vendor folder, we will find the autoload.php file.
This file alone will load all packages in the vendor folder.*



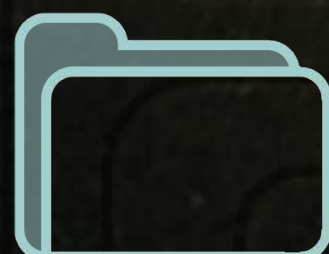
Respect



Validation



docs



library

*The files we will be needing are actually down here, in the
library folder, and autoload will take care of all that!*

CLOSE
ENCOUNTERS
with
PHP

Adding Autoloading From Composer

Requiring the autoload file in our project will give us access to all of the packages.

Add the autoload.php file from our vendor directory.

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



The autoload.php file will automatically give us access to all of the packages within the Composer vendor directory.

Adding Autoloading

app/src/app.php

```
<?php
require __DIR__ . '/../../vendor/autoload.php';
require __DIR__ . '/validation.php';
...
if (!empty($date) && !empty($email) && !empty($description)) {
    echo validate_date($date);

    if (filter_var($email, FILTER_VALIDATE_EMAIL)) {
        echo "<p>Email: $email</p>";
    }
    echo '<p>' . htmlspecialchars($description) . '</p>';
}
}
```

Add the autoload.php file from our vendor directory

We can now use packages anywhere below, including in our validation file

Using Respect/Validation


The use command is how we are able to load libraries.

app/src/validation.php

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

function validate_date($date_string)
{
    if ($time = strtotime($date_string)) {
        return date('F jS Y', $date_string);
    } else {
        return $date_string . ' does not look valid.';
    }
}
```

*Using the namespace of Respect\Validation,
we can access the Validator class*



Using Respect/Validation

The use command is how we are able to load libraries.

app/src/validation.php

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

```
$v = new Validator;
```

The new keyword creates a Validator object named \$v

```
function validate_date($date_string)
{
    if ($time = strtotime($date_string)) {
        return date('F jS Y', $date_string);
    } else {
        return $date_string . ' does not look valid.';
    }
}
```


Using Respect/Validation

The use command is how we are able to load libraries.


app/src/validation.php

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

```
$v = new Validator;
```

```
var_dump($v);
```

*Let's see what the \$v object looks like
with a var_dump!*



```
function validate_date($date_string)  
{  
    if ($time = strtotime($date_string)) {  
        return date('F jS Y', $date_string);  
    } else {  
        return $date_string . ' does not look valid.';  
    }  
}
```


Var Dump of Our Validator

The Validator is an object type, with a protected array of rules. What is all this?!

```
/var/www/hello/app/src/app.php:8:  
object(Respect\Validation\Validator)[3]  
  protected 'rules' =>  
    array (size=0)  
      empty  
  protected 'name' => null  
  protected 'template' => null
```

- We can run validation commands with the validator to test against custom rules
- Each instance of a validator can have a unique name
- Each instance of a validator can also have a template that allows us to customize our error strings
- We can now add some rules to our empty rules array on the Validator

Composer & Autoloading Review

Let's take a quick look back over this lesson in review.

- Composer is a package manager for PHP
- We used the Composer CLI to search and install packages to our application
- We gained access to the package through the use of the **autoload.php** file
- The **use** keyword allows us to access a class through a Namespace/ClassName pattern
- We create new validator instances with the **new** keyword

CLOSE ENCOUNTERS

with
PHP

Level 5

Validation With Respect

Object-oriented Validation

A Closer Look at the Validator Class

validation.php

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

$v = new Validator;
var_dump($v);

function validate_date($date_string)
{
    if ($time = strtotime($date_string)) {
        return date('F jS Y', $date_string);
    } else {
        return $date_string . ' does not look valid.';
    }
}
```

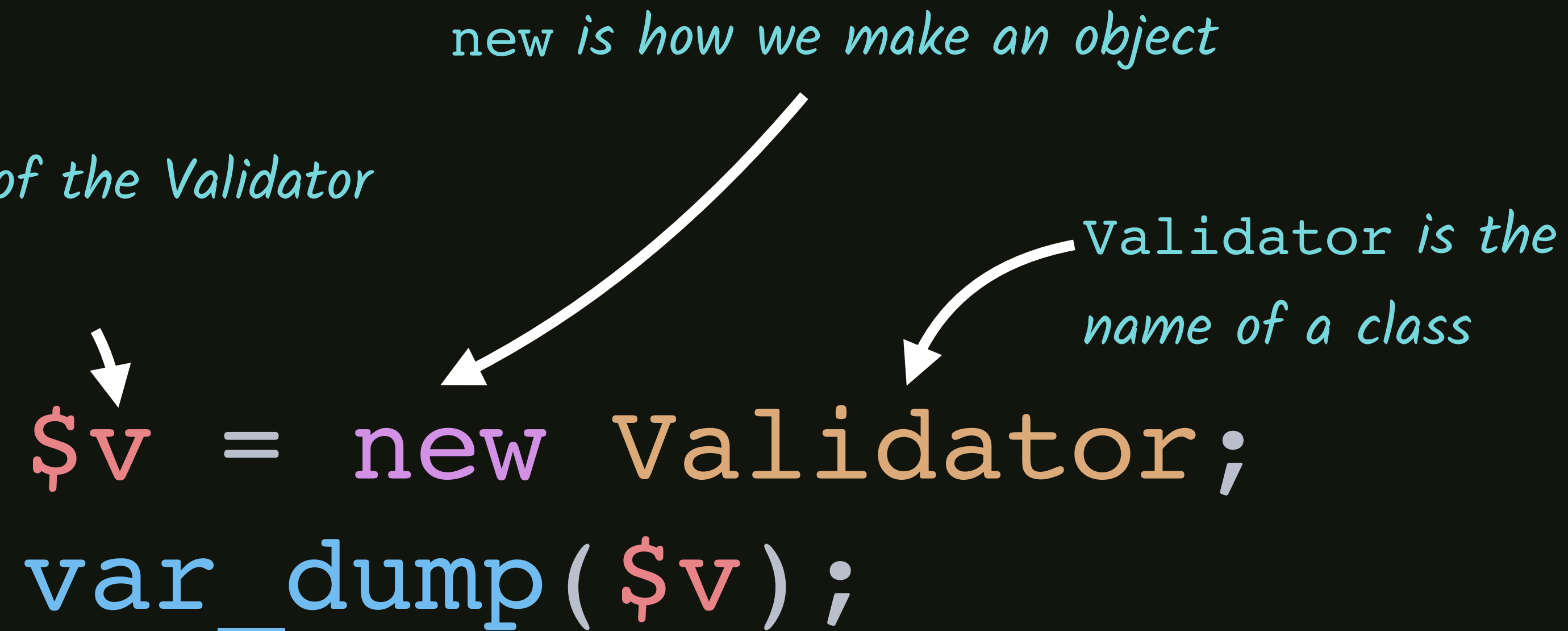

A Closer Look at the Validator Class

validation.php

new is how we make an object

*\$v is now an object of the Validator
type*

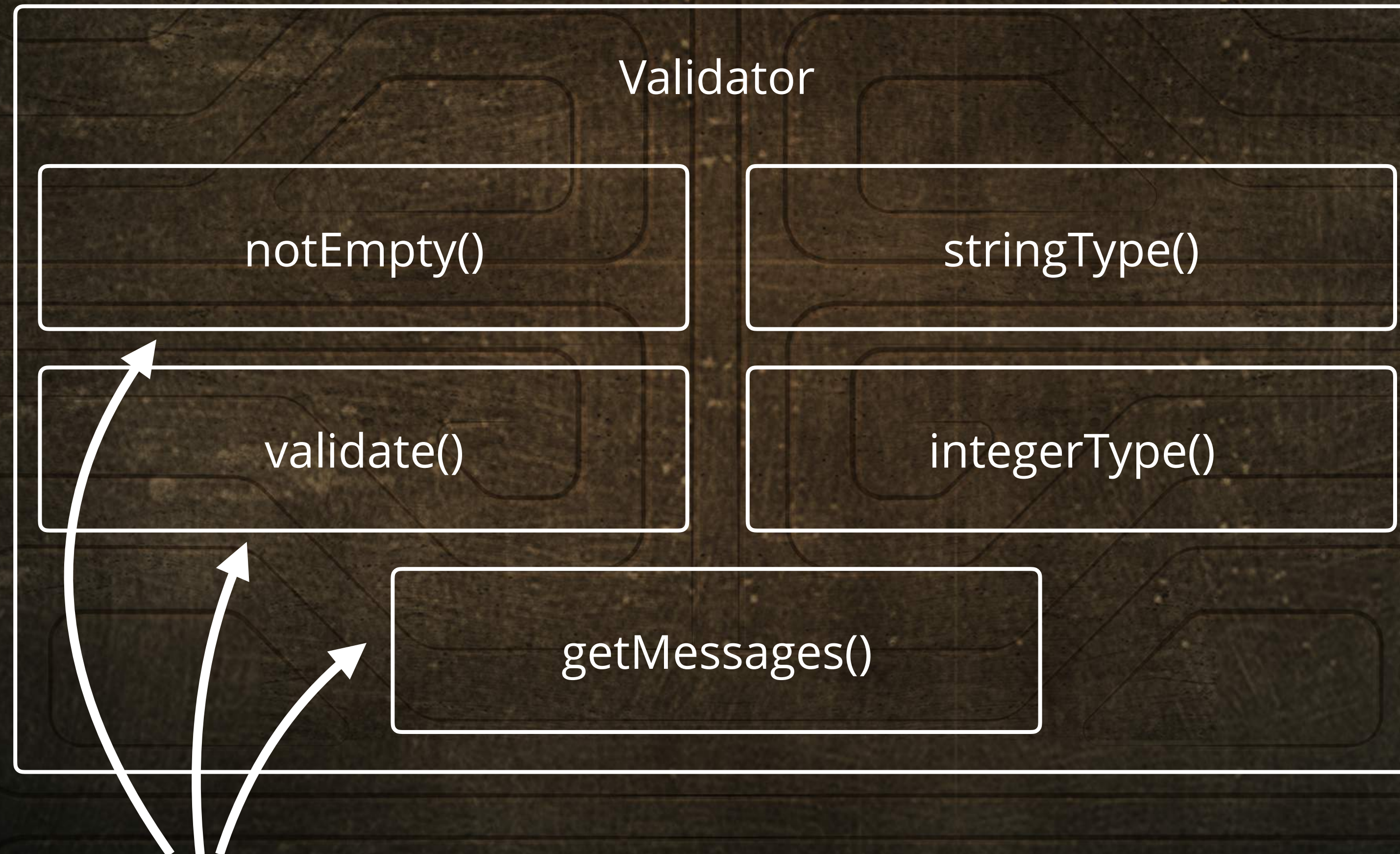
*Validator is the
name of a class*



```
$v = new Validator;  
var_dump($v);
```


What Is a Class?

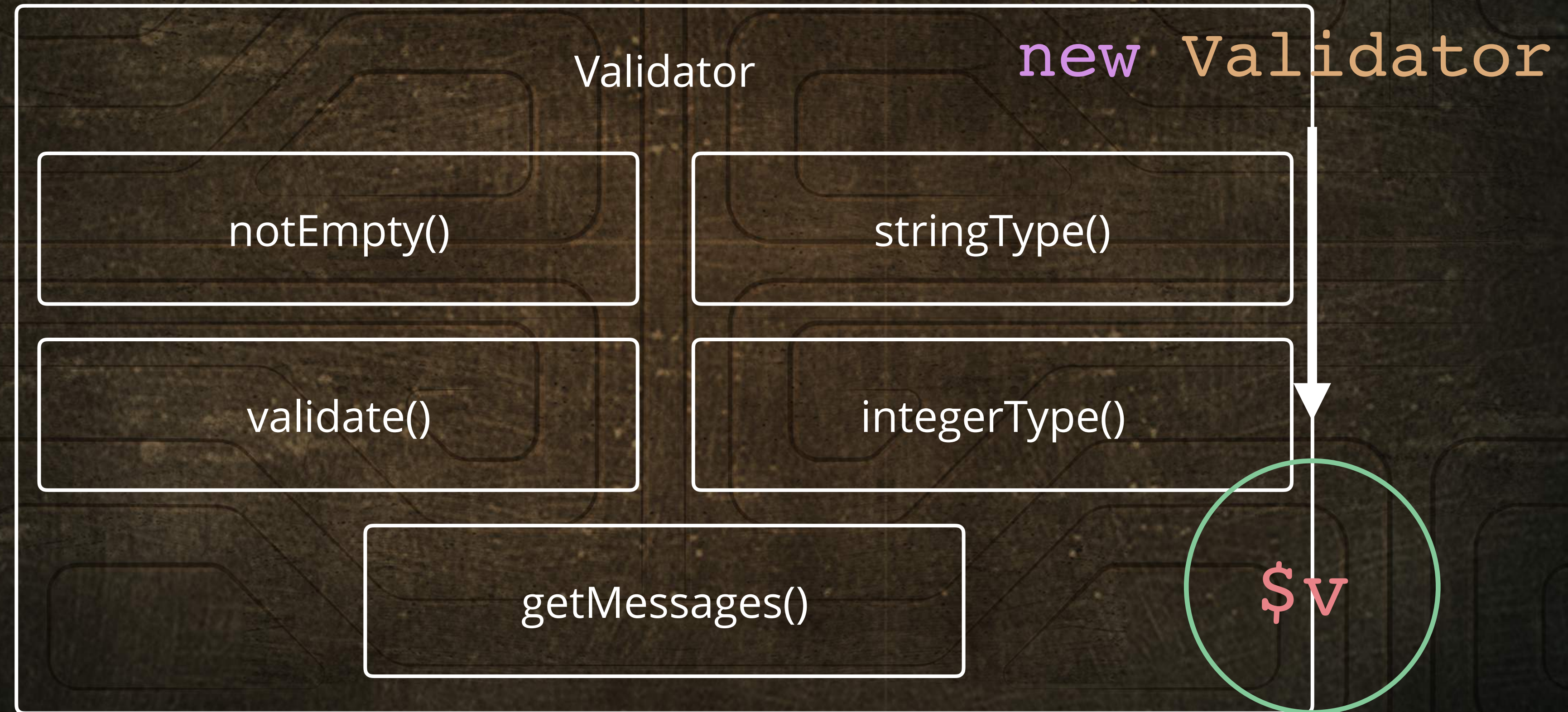
A class is a way to group our code and provide a blueprint for a single-purpose object.



Methods are functions inside of a class

Creating an Object

With the new keyword, we create an object that is a single instance of the Validator class.



How Can We Use Validator Here?


validation.php

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

$v = new Validator;
var_dump($v);

function validate_date($date_string)
{
    if ($time = strtotime($date_string)) {
        return date('F jS Y', $date_string);
    } else {
        return $date_string . ' does not look valid.';
    }
}
```

*We need to replace all of the code inside the
validate_date function with something
from the Validator class*



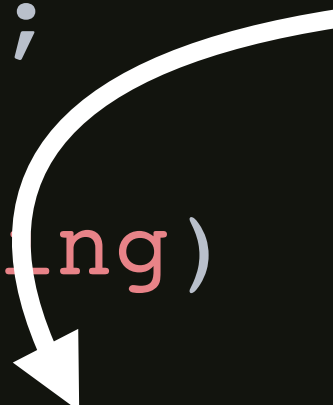
Using Respect/Validation for the Date

validation.php

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

function validate_date($date_string)
{
    if(Validator::notEmpty()->validate($date_string)) {
        return date('F jS Y', $date_string);
    }
}
```

This validate call will return true or false, so it is okay to use it as a conditional



There is a lot going on here! Let's take a closer look.

Inspecting the Validator Class

This line of code means, “If the \$date_string is not empty, return true; otherwise, return false.”

validate is a method that tests our data against the rules — in this case, notEmpty



```
Validator::notEmpty()->validate($date_string);
```

The diagram consists of two white arrows. One arrow originates from the text 'validate is a method that tests our data against the rules — in this case, notEmpty' and points to the 'validate' method call in the code snippet. The second arrow originates from the text 'notEmpty is a method (function) in the Validator class, which validator calls a rule' and points to the 'notEmpty' method call in the code snippet.

*notEmpty is a method (function) in the
Validator class, which validator calls a rule*

Validation of a Date

We have validated that our date string is not empty, but how do we know it is a date?

First, we are evaluating that \$date_string is not empty!



```
Validator::notEmpty()->validate($date_string);
```

```
Validator::date()->validate($date_string);
```



date() verifies that the string entered is in a valid date format


Chaining Validation Methods

With Respect/Validation, we can chain rules together for simplicity and clarity.

Now we're validating that we have some data and that it looks like a date in one line

```
Validator::date()->notEmpty()->validate($date_string);
```

*We can chain the validation rules together
by using an object operator ->*



This is great, but can we do more?

Creating Custom Validators

Clarifying our code is easy when we create a custom validator variable.

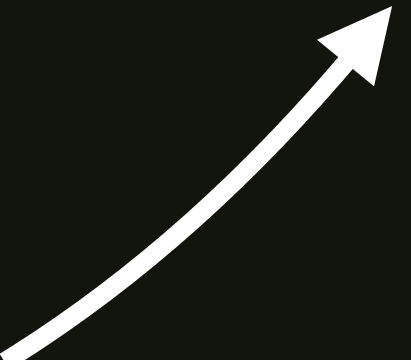
We can assign a Validator class with rules to a variable



```
$date_validator = Validator::date()->notEmpty();
```

```
$date_validator->validate($date_string);
```

Now we only have to run the validate method on the \$date_validator variable we created!



Validation With Our Custom Validator

validation.php

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

Create our validator for the date

```
function validate_date($date_string)
{
    $date_validator = Validator::date()->notEmpty();
```

```
    if ($date_validator->validate($date_string)) {
        $date_time = strtotime($date_string);
        return date('F jS Y', $date_time);
    }
}
```

Use a conditional to test our validator

Requiring a Date Format

We can set a format inside our date validator call, using any format from the PHP date function.


validation.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);
    }
}
```

The format d-m-Y requires that the date be entered like this: 30-12-2017



Returning a Message on Error


validation.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.';
    }
}
```

If the validator fails, return a message about the format!



Validator Class Review

What we have gone over in this section

- How to create an object
- Using our **Validator** class and methods
- The **validate** method with the **notEmpty** rule
- Chaining rules with the object operator
- Storing custom validators in a variable
- Using custom validators in a conditional

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
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.

```
try {  
      
} catch (Exception $e) {  
      
}
```

Inside the try brackets, we will add our code. If this fails and throws an exception, we will want to catch it.

Exception is a class that is built into PHP. Here, it is assigning the exception to an object variable called \$e.

Inside the catch brackets, we will want to do something useful with the new \$e object, which will be of the Exception type

Using Respect's Exception Classes

app/src/validation.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
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)
          '00000000535eff730000000006c81c734' =>
            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)
                      '00000000535eff720000000006c81c734' =>
                        object(Respect\Validation\Rules\Date)[5]
                          ...
                          '00000000535eff710000000006c81c734' =>
                            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
...
$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
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  
require __DIR__ . '/../../vendor/autoload.php';
```

Include our validation library

```
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']);
```

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

```
    $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);  
        $email_validator->assert($email);
```


A Better Approach, With Validation

app/src/app.php

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

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

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

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

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

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