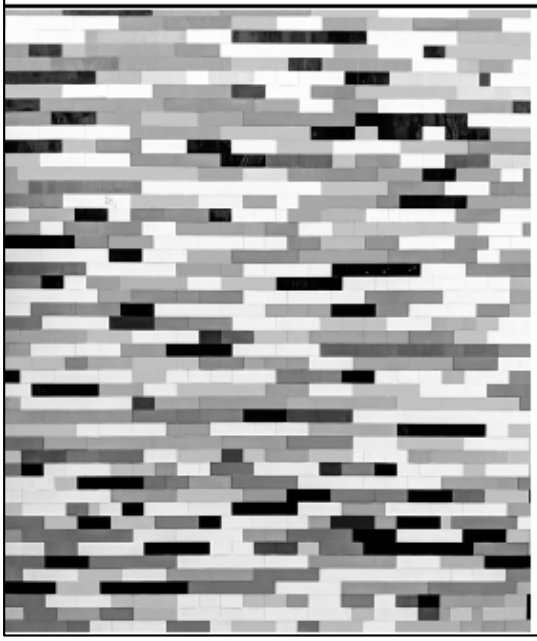
DEEPer

PHP Development
Week 3 Session 1

TODAY'S SESSION



01 PHP

PHP

PHP: Hypertext Preprocessor

01

What is PHP

- Stands for PHP: Hypertext Preprocessor
- Created in 1994 by Rasmus Lerdorf
- Allows us to create dynamic server-driven pages
- Open source general purpose scripting language
- PHP files can contain HTML (with nested CSS/JS) and PHP code
- PHP files run **on the server**, so the HTML output from PHP is returned to the user's browser – they never see the PHP code

Basic Syntax


```php
<?php

// Single line comment

/*
  Multi-line comment
*/

$variable = true;

?>
```

Opening/closing tags, attributes and values, content. The content of self-closing tags is either pre-defined, or defined by its attributes

Variables & Data Types


```php
<?php

$string = 'Hello World!';
$int = 30;
$floatingPoint = 3.14159;
$boolean = true;
$array = ['apple', 'pear', 'banana'];

$reassignedVariable = $int; // 30

$concatenatedString = 'Hello' . ' World'; // "Hello World"
```

Arrays

- Arrays are collections of **values**, each of which has a **key**
- Keys can be either specified or generated automatically
- Keys can either be strings or integers
- An array may not contain the same key twice
- If no key is provided when an element is added to an array, the next available integer is used

Arrays – Examples


```php
$arrayWithStringKeys = [
  'name' => 'John Smith',
  'age' => 25,
];
$example = $arrayWithStringKeys['name']; // John Smith

$arrayWithIntegerKeys = [
  5 => true,
  9 => false,
];
$example = $arrayWithIntegerKeys[5]; // true

// [0 => 'apple', 1 => 'pear', 2 => 'banana']
$arrayWithGeneratedKeys = ['apple', 'pear', 'banana'];
$example = $arrayWithGeneratedKeys[1]; // pear
```

Array & String Functions

- PHP includes several built-in functions to manipulate arrays and strings
- They perform various commonly required operations like;
  - Reversing or sorting values
  - Comparing values
  - Adding or removing values
  - Applying logic to all elements in array

Array & String Functions - Examples


```php
<?php

$string = 'hello world!';
$array = ['apple', 'orange', 'pear'];
$array2 = ['plum', 'banana'];

// https://www.php.net/manual/en/ref.strings.php
$leftTrimmedString = ltrim($string, 'H') // 'ello World'
$explodedString = explode(' ', $string); // ['hello', 'world']
$stringLength = strlen($string); // 12
$upperCasedFirst = ucfirst($string); // 'Hello world'

// https://www.php.net/manual/en/ref.array.php
$reversedArray = array_reverse($array);
$mergedArrays = array_merge($array, $array2);
$lastElement = array_pop($array);
$firstElement = array_shift($array);
$arrayLength = count($array); // 3
$mappedArray = array_map(function ($value) {
  return 'Foo: ' . $value;
}, $array);
```

Basic Output

Simple variable types can be outputted using `echo`

```
1  <?php $message = 'Hello world!'; ?>
2  <!DOCTYPE html>
3  <html>
4    <head>
5      <title>PHP Example</title>
6    </head>
7    <body>
8      <p><?php echo $message; ?></p>
9    </body>
10 </html>
```

Students to replicate

Basic Output

More complicated data types like arrays or objects can be outputted to the screen using `var_dump`. This is generally only used for debugging and testing purposes.

```
array(3) {
  [0]=>
  int(1)
  [1]=>
  int(2)
  [2]=>
  int(3)
}
```

Variables & Data Types

- PHP is a loosely typed language (like JavaScript!)
- Variables can change their type on-the-fly
- Variables always begin with a dollar sign
- Types include string, integer, floating point, boolean, array, object, null

Operators

- There are three types of operator
  - **Arithmetic** – mathematic operations
  - **Assignment** – store a value in a variable
  - **Comparison** – compare the values of variables

Arithmetic Operators

Operator	Name	Example	Result
+	Add	$z + $x	Sum of $z and $x
-	Subtract	$z - $x	Difference of $z and $x
*	Multiply	$z * $x	Product of $z and $x
/	Divide	$z / $x	Quotient of $z and $x
%	Modulus	$z % $x	Remainder of $z divided by $x
**	Exponentiate	$z ** $x	Result of raising $z to the $x'th power

https://www.bitdegree.org/learn/php-operators

Assignment Operators

Assignment	Same as...	Description
z = x	z = x	The variable on the left gets value of the variable on the right
z += x	z = z + x	Adds the value on the left of the operand to the value on the right
z -= x	z = z - x	Subtracts the value on the left of the operand from the value on the right
z *= x	z = z * x	Multiplies the value on the left of the operand by the value on the right
z /= x	z = z / x	Divides the value on the left of the operand by the value on the right
z %= x	z = z % x	Displays the modulus of the value on the left of the operand by the value on the right

https://www.bitdegree.org/learn/php-operators

Comparison Operators

Operator	Name	Example	Result
==	Equal to	$z == $x	Returns true if $z is equal to $x
===	Identical to	$z === $x	Returns true if $z is equal to $x, and they are of the same type
!=	Not equal to	$z != $x	Returns true if $z is not equal to $x
<>	Not equal to	$z <> $x	Returns true if $z is not equal to $x
!==	Not identical to	$z !== $x	Returns true if $z is not equal to $x, or they are not of the same type
>	Greater than	$z > $x	Returns true if $z is greater than $x
<	Less than	$z < $x	Returns true if $z is less than $x
>=	Greater than or equal to	$z >= $x	Returns true if $z is greater than or equal to $x
<=	Less than or equal to	$z <= $x	Returns true if $z is less than or equal to $x

https://www.bitdegree.org/learn/php-operators

Conditionals & Loops

- We may perform alternate actions or sequences based on certain conditions being met
- PHP offers several basic constructs to achieve this
- Types of conditionals:
  - if / elseif / else
  - switch

Conditionals – if/elseif/else


```php
<?php

$monthNum = date('n'); // 'n' gives us the month as a numeric value (1-12)

if ($monthNum <= 3) {
    echo 'You are in the first quarter of the year';
} elseif ($monthNum <= 6) {
    echo 'You are in the second quarter of the year';
} elseif ($monthNum <= 9) {
    echo 'You are in the third quarter of the year';
} else {
    echo 'By process of elimination, you must be in the last quarter of the year';
}
```

Conditionals - switch


```php
$monthNum = date('n'); // 'n' gives us the month as a numeric value (1-12)

switch ($monthNum) {
    case 1:
    case 2:
    case 3:
        echo 'You are in the first quarter of the year';
        break;
    case 4:
    case 5:
    case 6:
        echo 'You are in the second quarter of the year';
        break;
    case 7:
    case 8:
    case 9:
        echo 'You are in the third quarter of the year';
        break;
    default:
        echo 'By process of elimination, you must be in the last quarter of the year';
}
```

Loops

- There are times when we may want to perform the same action more than once, either;
  - For a set number of times
  - For each item that exists in an array
- Types of loops:
  - for / foreach
  - do / do while

Very similar to JavaScript

For Loops

Used when we know exactly how many iterations are required

```
for ($i = 1; $i < 10; $i++) {
    echo $i;
}

// output: 123456789
```

Foreach Loops

Run code for each element in an array

```
$fruits = ['apple', 'pear', 'banana'];

foreach ($fruits as $fruit) {
    echo $fruit;
}

// output: applepearbanana
```

While Loops

Used when we want to perform an action until a condition is met

```
$string = '';

while (strlen($string) < 5) {
  $string .= 'a';
}

echo $string;

// output: aaaaa
```

Do While Loops

Used when we want to perform an action until a condition is met (but always at least once!)

```
$performAction = false;

do {
    echo 'hello!';
} while ($performAction === true);

// output: hello!
```

Functions

Reusable code can be extracted to functions, like JavaScript

```
<?php

function addNumbers($a, $b) {
  return $a + $b;
}

$sum = addNumbers(3, 5); // 8
```

Variable Scope

- Variables defined **outside** of any functions have **global** scope
- Variables defined **within** a function have **local** scope

```php
<?php

$foo = 'bar';

function myFunction() {
    $bar = 42;

    echo $foo;  // doesn't work
    echo $bar;  // works
}

echo $foo;  // works
echo $bar;  // doesn't work
```

Classes

- Classes can be thought of as **blueprints for objects**
- You can create **instances** of a **class** with the `new` keyword

```php
<?php
  class User {
    public $id;
    public $name;
    public $age;
  }
?>
```

```php
<?php
  $me = new User();
?>
```

Classes

- Classes have **properties** (variables) and **methods** (functions)
- Both properties and methods have **visibility**
- For now, all properties will be **public**, meaning they can be accessed in any code which has access to an instance

```
class User {
  public $name;
}
```

Classes – Setting Properties

- Public properties can have default values, and the value can be set or overridden from outside

```php
<?php
  class User {
    public $name = 'Joe Bloggs';
    public $age = 20;
  }

  $me = new User();
  $me->name = 'Jane Doe';
  $me->age = 25;
?>
```

Classes – Getting Properties

- Public properties can be referenced using similar syntax to setting them

```php
<?php
  class User {
    public $name = 'Joe Bloggs';
    public $age = 20;
  }

  $me = new User();
  $name = $me->name; // Joe Bloggs
```

Classes – Hydration

- The process of filling object instances with data is often referred to as **Hydrating Objects**
- There are libraries which can automate this process, but for now we'll look at some manual examples

Classes – Hydration


```php
<?php
  class User {
    public $name;
  }

  // $users could come from a database, file(s), an API call etc
  $users = [
    [
      'name' => 'Joe Bloggs',
      'age' => 20,
    ], [
      'name' => 'Jane Doe',
      'age' => 25,
    ],
  ];

  $userObjects = [];

  foreach ($users as $user) {
    $instance = new User();
    $instance->name = $user['name'];
    $instance->age = $user['age'];

    $userObjects[] = $instance;
  }

  // $userObjects is now an array of User objects containing the user data
  var_dump($userObjects);
```

```
array(2) {
  [0]=>
  object(User)#1 (2) {
    ["name"]=>
    string(10) "Joe Bloggs"
    ["age"]=>
    int(20)
  }
  [1]=>
  object(User)#2 (2) {
    ["name"]=>
    string(8) "Jane Doe"
    ["age"]=>
    int(25)
  }
}
```

Form Handling

- When a form with an action of GET or POST is submitted to a PHP file, PHP can access the submitted values
- Submitted data is available in PHP's **Superglobals** in array form
- If a form has an empty `action` attribute, it will submit back to itself
- In PHP, `$_GET` and `$_POST` contain the submitted values respectively
- GET submits through the URL, POST through the request body

Form Handling - POST


```php
<?php
  if (!empty($_POST)) {
    // The form was submitted as there is POST data
    echo '<pre>';
    var_dump($_POST);
    echo '</pre>';
  }
?>

...
<body>
  <form action="" method="post">
    <label for="my-username">Username:</label>
    <input type="text" name="username" id="my-username">

    <button type="submit">Submit Form</button>
  </form>
</body>
```

Username: JoeBloggs109 Submit Form

```
array(1) {
  ["username"]=>
  string(12) "JoeBloggs109"
}
```

Username: Submit Form

Form Handling - GET


```php
<?php
  if (!empty($_GET)) {
    // There are GET parameters in the URL
    echo '<pre>';
    var_dump($_GET);
    echo '</pre>';
  }
?>

...
<body>
  <form action="" method="get">
    <label for="search-term">Search Term:</label>
    <input type="text" name="searchTerm" id="search-term">

    <button type="submit">Search</button>
  </form>
</body>
```

Search Term: my search term Search

```
array(1) {
  ["searchTerm"]=>
  string(14) "my search term"
}

...
```

Search Term: Search

Form Handling – Multiple Selection


```html
<form action="" method="post">
  <h2>Book Form</h2>
  <label for="genreHorror">
    Horror
    <input
      id="genreHorror"
      type="checkbox"
      name="genre[]"
      value="horror"
    >
  </label>
  <label for="genreFantasy">
    Fantasy
    <input
      id="genreFantasy"
      type="checkbox"
      name="genre[]"
      value="fantasy"
    >
  </label>
  <label for="genreBiography">
    Biography
    <input
      id="genreBiography"
      type="checkbox"
      name="genre[]"
      value="biography"
      >
  </label>

  <button type="submit">Submit Form</button>
</form>
```

Book Form

Horror ☑ Fantasy ☐ Biography ☑ Submit Form

```
array(1) {
  ["genre"]=>
  array(2) {
    [0]=>
    string(6) "horror"
    [1]=>
    string(9) "biography"
  }
}
```

Book Form

Horror ☐ Fantasy ☐ Biography ☐ Submit Form

Splitting Up PHP

- So far all of our PHP code has been in the same file as HTML
- Some code we have written so far, we will want to be reusable
- A key example of this is the classes we have defined

Splitting Up PHP


```php
<?php

// Includes/loads the specified file and runs/evaluates it
// Emits a PHP Warning if not found
include 'path/to/file.php';

// Same as include, however if this line runs more than once the file will
// only be loaded once
include_once 'path/to/file.php';

// Same as include, but emits a PHP Error if not found
require 'path/to/file.php';

// Same as require_once, but emits a PHP Error if not found
require_once 'path/to/file.php';
```

Splitting Up PHP – Example


```
// ----------------------------------------
// mainFile.php

<?php
  require_once 'myFile.php';
  echo 'Hello from the main file!';
?>
```

Hello from the required file!
Hello from the main file!

```
// ----------------------------------------
// myFile.php

<?php
  echo 'Hello from the required file!<br>';
?>
```

Splitting Up PHP – Class Example


```
// ----------------------------------------
// index.php

<?php
  require_once 'Product.php';

  $product = new Product();
  // ...
?>


// ----------------------------------------
// Product.php

<?php
  class Product {
    public $title;
  }
?>
```

- Product.php can now be included in any script requiring it, preventing duplication

Object Serialisation

- Serialising a value in PHP generates a representation which can be stored, and later retrieved and unserialised
- This is often used for storing object references for later consumption by another script, for example in a file or database
- PHP provides two simple functions for this process
  - `serialize()` – converts a PHP value to its byte-stream representation
  - `unserialize()` – converts a stored representation back into its original PHP value

Object Serialisation - Example


```php
<?php
  class Product {
    public $title;
  }

  $product = new Product();
  $product->title = 'My Product';

  $serialisedProduct = serialize($product);

  echo '<pre>' . $serialisedProduct . '</pre>';
?>
```

```
O:7:"Product":1:{s:5:"title";s:10:"My Product";}
```

Object Unserialisation - Example


```php
<?php
  // The original class of the serialised object must be loaded in scope
  class Product {
    public $title;
  }

  // Serialised project likely loaded from storage (DB/File etc)
  $serialisedProduct = 'O:7:"Product":1:{s:5:"title";s:10:"My Product";}';

  $product = unserialize($serialisedProduct);

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

```
object(Product)#1 (1) {
  ["title"]=>
  string(10) "My Product"
}
```

Files With PHP

- PHP offers several methods of reading and writing to files on the same server
- A common example is reading and writing CSVs
- Today, we'll be reading and writing serialised data
- It is possible to append to a file or start writing at a specific location
- Files are commonly accessed via a `resource` variable

Files With PHP

- There are several functions available to perform actions on files:
  - `fopen` – creates a resource handle to access the file
  - `fread` – reads a specified amount of data from an open file
  - `fwrite` – writes data to a file
  - `fseek` – moves the internal pointer (cursor)
  - `fclose` – closes the file's resource handle
- Using these methods move the internal pointer – think of this as your cursor's location in VS Code, but controlled programmatically!

Files With PHP

- The `fopen` function takes two parameters:
- `$filename` – the name and path of the file (relative to the script)
- `$mode` – the way in which to open the file. Common examples are:
  - `r` – read-only access, no ability to `fwrite`
  - `w` – write-only access, no ability to `fread`
  - `w+` – read and write access, ***overwriting*** existing contents
  - `a+` – read and write access, ***appending*** to existing contents
- More modes exist and can be reviewed here: https://www.php.net/manual/en/function.fopen.php

Run some polls after this slide?

Files With PHP


```php
$filename = 'file.txt'; // file contents: "Hello world!"
$file = fopen($filename, 'w+'); // overwrite file contents

fwrite($file, 'Hello files!'); // writes "Hello files!" to the file

fseek($file, 0); // move our cursor back to the start of the file

echo fread($file, filesize($filename)); // output: "Hello files!"

fclose($file); // close the file handle
```

Files With PHP

- A common interaction is to read the entire contents of a file into a string
- Based on the previous example, the interaction would look as follows:
  - `fopen` with a mode of `r`
  - `fread` the full length of the file (using `filesize`)
  - `fclose`
- A simpler alternative is to use a single method:
  - `file_get_contents`

Files With PHP


```
$file = fopen($filename, 'r');
$fileContents = fread($file, filesize($filename));
fclose($file);

$fileContents = file_get_contents($filename);
```

Files With PHP

- Similarly, we may commonly just want to replace the contents of a file with something we have in a string
- Based on the previous example, the interaction would look as follows:
  - `fopen` with a mode of `w`
  - `fwrite`
  - `fclose`
- A simpler alternative is to use a single method:
  - `file_put_contents`

Files With PHP


```
$file = fopen($filename, 'w');
fwrite($file, 'Hello files!');
fclose($file);

file_put_contents($filename, 'Hello files!');
```

Files With PHP

- If we're using multiple files to store data, we may wish to scan the contents of a directory
- There are several methods to achieve this:
  - `scandir` – get an array of files in a directory, including dot (hidden) files and relative paths ('.' and '..')
  - `glob` – get an array of files matching a pattern in a directory
  - `opendir` / `readdir` / `closedir` – iterate through a directory

Files With PHP


```php
$files = scandir('.'); // scans the directory in which this script resides
/**
 * Sample output:
 * array(5) {
 *   [0]=>
 *   string(1) "."
 *   [1]=>
 *   string(2) ".."
 *   [2]=>
 *   string(9) "file1.txt"
 *   [3]=>
 *   string(9) "file2.txt"
 *   [4]=>
 *   string(9) "file3.txt"
 * }
 **/
```

Files With PHP


```
$files = glob('*.txt'); // scans the current directory for any .txt files
/**
 * Sample output:
 * array(3) {
 *   [0]=>
 *   string(9) "file1.txt"
 *   [1]=>
 *   string(9) "file2.txt"
 *   [2]=>
 *   string(9) "file3.txt"
 * }
 **/
```

Templating

- PHP can be used to dynamically generate HTML output based on available data
- In its simplest form dynamic values can be outputted, such as a product's title
- Most logic in PHP is available when templating, including conditionals and loops
- Try to keep your templating logic separate to all other logic, such as loading records

Templating – Syntax


```
// Echo a variable
<?= $myVariable; ?>

// Conditional
<?php if ($booleanVariable): ?>
  <p>Output If True</p>
<?php elseif ($anotherBoolean): ?>
  <p>Output If True</p>
<?php else: ?>
  <p>Output Else</p>
<?php endif ?>

// Foreach
<?php foreach ($arrayVariable as $item): ?>
  <?= $item; ?>
<?php endforeach ?>
```

Templating – Example


```php
<?php
  class Product {
    public $title = 'Macbook Pro';
    public $isInStock = true;
    public $reviews = ['Very good!', 'Better than Windows!'];
  }

  $product = new Product(); // e.g. loaded from a file
?>

<body>
  <h1>
    <?= $product->title; ?>
  </h1>

  <p>
    <?php if ($product->isInStock): ?>
      Product is in stock!
    <?php else: ?>
      Product is out of stock!
    <?php endif ?>
  </p>

  <h2>Reviews</h2>
  <ul>
    <?php foreach ($product->reviews as $review): ?>
      <li><?= $review; ?></li>
    <?php endforeach ?>
  </ul>

</body>
```

Macbook Pro

Product is in stock!

Reviews

- Very good!
- Better than Windows!

Redirecting

- In many cases when processing PHP logic, we want to redirect the user to another PHP script or URL
- For example after creating a new record, on successful save we can redirect the user to a dashboard page or that product's details page

Redirecting


```php
<?php

// Some other logic...

header("Location: my-file.php");
// or
header("Location: http://google.co.uk");

// Kill the script
die();
```

Date & Time

- In PHP there are various ways of accessing and manipulating date and time
- Several functions are provided along with a DateTime object

Date & Time – Function Examples


```php
<?php

/* Key formatting characters;
  Y: 4 digit year
  m: 2 digit month
  d: 2 digit day
  H: 2 digit hours
  i: 2 digit minutes
  s: 2 digit seconds
  l: day of the week (e.g. Sunday)
  F: month of the year (e.g. July)
*/

// time() returns the current timestamp
echo time(); // e.g. 1597610372

// date() by default uses the current time() value
echo date('Y-m-d H:i:s'); // e.g. 2020-08-17 12:30:05

// strtotime() converts various textual date descriptions to a timestamp
echo strtotime('+1 week 2 days 3 hours'); // 1598399029

// The above can then be combined as required
$oneWeekToday = date('d/m/Y', strtotime('+1 week'));
```

Date & Time – Object Examples


```php
<?php

// Defaults to now
$dateTime = new DateTime();
echo $dateTime->format('Y-m-d');

// Optional date can be provided on creation
$dateTime = new DateTime('2020-05-12');

// If the format of the required date is known;
$dateTime = DateTime::createFromFormat('j-M-Y', '15-Feb-2009');
```