



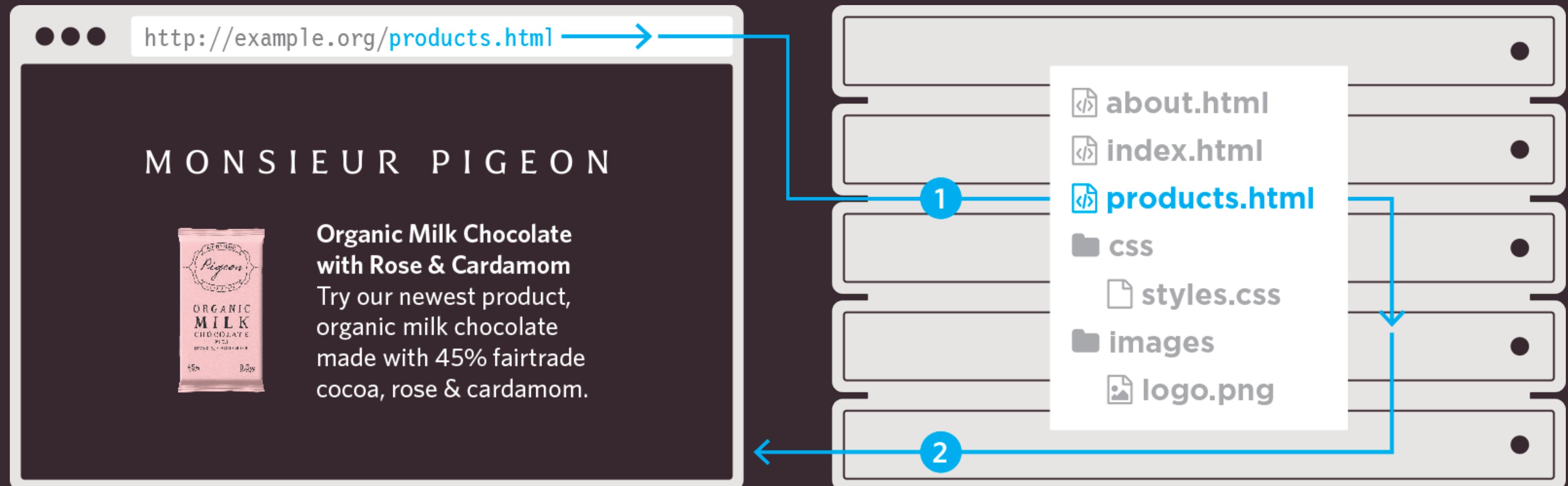
# PHP&MySQL

server-side web  
development

# INTRODUCTION

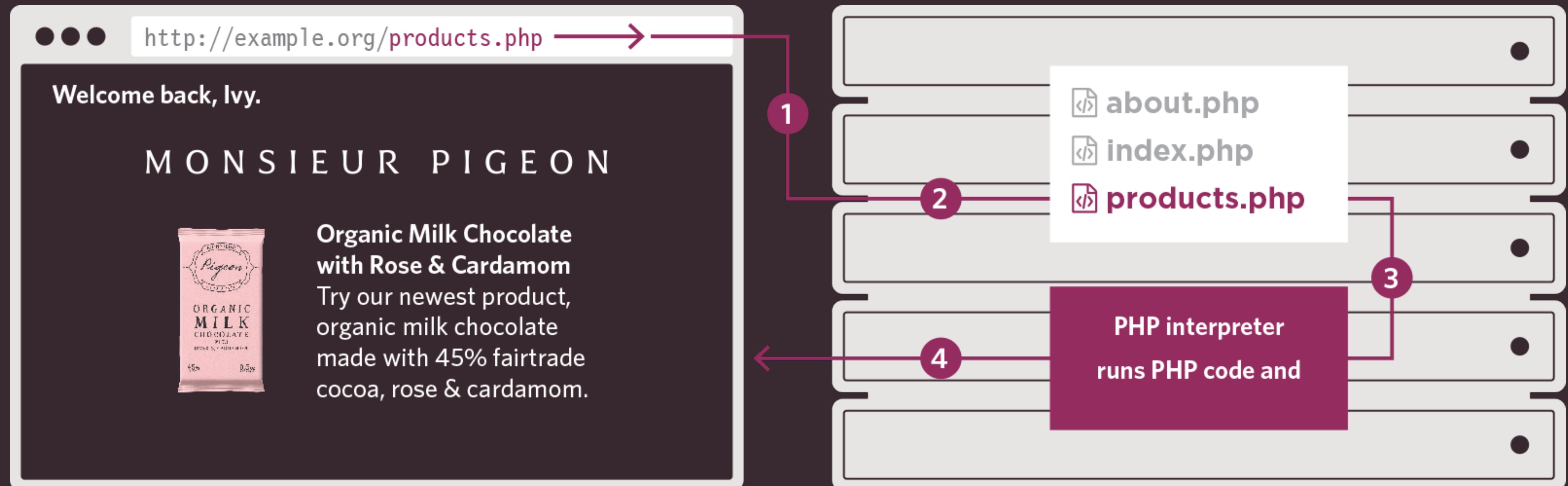
# STATIC WEBSITES

Every visitor sees the same content because they are sent the same HTML & CSS files.



# DYNAMIC WEBSITES

Each visitor can be sent a different page because PHP is used to create the HTML.



MySQL is a type of database. It stores information in tables, like a spreadsheet.

# TABLE

member						
	id	forename	surname	password	joined	picture
1	Ivy		Stone	\$2y\$10\$MAd...	2021-01-01	ivy.jpg
2	Luke		Wood	\$2y\$10\$NN5...	2021-01-01	NULL
3	Emiko		Ito	\$2y\$10\$/RpR...	2021-01-01	emi.jpg

# COLUMN NAMES

member					
id	forename	surname	password	joined	picture
1	Ivy	Stone	\$2y\$10\$MAd...	2021-01-01	ivy.jpg
2	Luke	Wood	\$2y\$10\$NN5...	2021-01-01	NULL
3	Emiko	Ito	\$2y\$10\$/RpR...	2021-01-01	emi.jpg

# COLUMN

member					
id	forename	surname	password	joined	picture
1	Ivy	Stone	\$2y\$10\$MAd...	2021-01-01	ivy.jpg
2	Luke	Wood	\$2y\$10\$NN5...	2021-01-01	NULL
3	Emiko	Ito	\$2y\$10\$/RpR...	2021-01-01	emi.jpg

# ROW

member						
	id	forename	surname	password	joined	picture
1	Ivy		Stone	\$2y\$10\$MAd...	2021-01-01	ivy.jpg
2	Luke		Wood	\$2y\$10\$NN5...	2021-01-01	NULL
3	Emiko		Ito	\$2y\$10\$/RpR...	2021-01-01	emi.jpg

**article**

	<b>id</b>	<b>title</b>	<b>summary</b>	<b>created</b>	<b>category_id</b>	<b>member_id</b>	<b>image_id</b>	<b>published</b>
	1	Systemic	<p>This	2021-01-01	1	2	1	1
	2	Poster	<p>These	2021-01-02	1	1	2	1
	3	Architect	<p>This	2021-01-02	4	2	3	1

**member**

	<b>id</b>	<b>forename</b>	<b>surname</b>	<b>password</b>	<b>joined</b>	<b>picture</b>
	1	Ivy	Stone	\$2y\$10\$MAd...	2021-01-01	ivy.jpg
	2	Luke	Wood	\$2y\$10\$NN5...	2021-01-01	NULL
	3	Emiko	Ito	\$2y\$10\$/RpR...	2021-01-01	emi.jpg

# SECTION A

# INTRODUCTION

# PHP TAGS

<?php

OPENING TAG

?>

CLOSING TAG

PHP PAGES CAN MIX HTML & PHP

```
<?php  
  
    price      =  5;  
  
    quantity  =  5;  
  
    total      =  $price * $quantity;  
  
?>  
  
<!DOCTYPE html>  
  
<html>  
  
    <head>  
  
        <title>Cost of Candy</title>  
  
    </head>  
  
    <body>  
  
        <p>Total: $ <?php echo $total; ?> </p>  
  
    </body>  
  
</html>
```

# STATEMENTS

Each instruction is called a **statement**.

```
<?php  
    price      = 5;  
    quantity  = 5;  
    total      = $price * $quantity;  
?>
```

# HOW PHP SENDS TEXT & HTML TO THE BROWSER

```
echo '<p>Hello!</p>';
```

The diagram illustrates the structure of the provided code. The word "echo" is highlighted in red and identified as a "KEYWORD". The string "'<p>Hello!</p>'" is highlighted in blue and identified as "TEXT & MARKUP TO DISPLAY".

```
echo "<a href=\"buy.php\">Buy</a>";
```

KEYWORD TEXT & MARKUP TO DISPLAY

```
echo "<a href='buy.php'>Buy</a>";
```

KEYWORD TEXT & MARKUP TO DISPLAY

```
echo '<a href="buy.php">Buy</a>';
```

KEYWORD TEXT & MARKUP TO DISPLAY

# COMMENTS

You should use **comments** to explain what your code does.

They help you remember it and others understand it.

# COMMENTS

```
echo 'Welcome'; // Comment
```

PHP

```
echo 'Welcome'; # Comment
```

```
/*
 Multi-line comment
 goes here
 */
```

CHAPTER 1  
VARIABLES, EXPRESSIONS  
& OPERATORS

# VARIABLES

**Variables** store data that can change (or vary) each time a PHP page is used.

# VARIABLE

ASSIGNMENT OPERATOR  
|  
`$quantity = 3;`  
|  
VARIABLE NAME                    VALUE

# USE VALUE IN A VARIABLE

```
echo $quantity;
```



DISPLAY      VALUE IN VARIABLE

# VARIABLE NAMES

# 1

Start with a  
dollar symbol \$

- ✓ \$greeting
- ✗ greeting

# 2

A letter or  
underscore

- ✓ \$greeting
- ✗ \$1\_greeting

# 3

Letters, numbers,  
or underscores

- ✓ \$greeting\_2
- ✗ \$greetin-g

# SCALAR DATA TYPES

PHP distinguishes between **strings**, **numbers**,  
and true or false values known as **booleans**.

# 1

Numbers

0.75

No quotes

# 2

Strings

'Hi Ivy!'

Enclosed in quotes  
(can be single or double  
quotes but must match)

# 3

Booleans

true

Either true or false

# INTEGERS & FLOATS

`int`

1

Whole numbers

`float`

1.25

Floating point numbers  
that represent fractions

# ARRAYS

Variables can hold **arrays**, which are a series of related values.

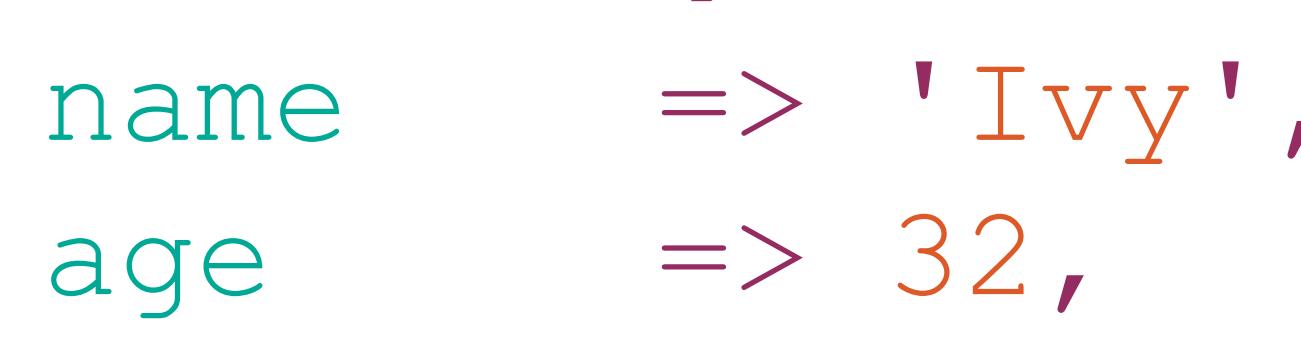
The array is a data type known as a **compound data type**.

# ASSOCIATIVE ARRAYS

An **associative array** uses a key to represent each value that it stores.

# CREATE ASSOCIATIVE ARRAY

```
$member = [  
    name      => 'Ivy',  
    age       => 32,  
    country   => 'Italy',  
];
```



# ACCESS VALUES IN ASSOCIATIVE ARRAY

```
$member = [  
    name      => 'Ivy',  
    age       => 32,  
    country  => 'Italy',  
];
```

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

[  
KEY  
]

This would write out 32.

# INDEXED ARRAYS

An **indexed array** uses an **index number** to represent each value it stores.

Index numbers start at 0 (not 1).

# CREATE INDEXED ARRAY

```
$shopping_list = [  
    'bread',  
    'cheese',  
    'milk',  
];   
          VALUE
```

# ACCESS VALUES IN INDEXED ARRAY

```
$shopping_list = [  
    'bread',  
    'cheese',  
    'milk',  
];
```

```
echo $shopping_list = [1];
```

VARIABLE HOLDING ARRAY

INDEX

This would write display cheese (because indexed arrays start at 0).

ECHO SHORTHAND

# ECHO SHORTHAND

```
<?php echo $name; ?>
```

```
<?= $name ?>
```

# ARITHMETIC OPERATORS

**Arithmetic operators** work with numbers to perform tasks like addition, multiplication, subtraction, and division.

Addition

$$4 + 2 = 6$$

Subtraction

$$4 - 2 = 2$$

Multiplication

$$4 * 2 = 8$$

Division

$$4 / 2 = 2$$

# ARITHMETIC OPERATORS

```
$price      = 3;  
$quantity  = 2;  
$total      = $price * $quantity;  
  
echo $total;
```

PHP

# STRING OPERATORS

The **concatenation operator** joins (or concatenates) two or more strings.

# CONCATENATION OPERATOR

```
$forename = 'Ivy';
$surname = 'Stone';
$fullname = $forename . ' ' . $surname;

echo $fullname;
```

PHP

# COMPARISON OPERATORS

A **comparison operator** compares two or more values. The result is a boolean value of either true or false.

`==`

IS EQUAL TO

`=====`

STRICT EQUAL TO

`!=` or `<>`

IS NOT EQUAL TO

`! ==`

IS NOT EQUAL TO

>

GREATER THAN

<

LESS THAN

>=

GREATER THAN  
OR EQUAL TO

<=

LESS THAN  
OR EQUAL TO

<=>

SPACESHIP OPERATOR

# LOGICAL OPERATORS

**AND**

**OR**

**NOT**

& &

| |

!

and

or

TYPE JUGGLING

The PHP interpreter can convert a value from one data type to another.

This is called **type juggling** and it can lead to unexpected results.

# TYPE JUGGLING / NUMBERS

INPUT	RESULT	TYPE
1 + '1'	2	int
1 + '1.2'	2.2	float
1 + '5star'	6	int
1 + 'star9'	1	int

# TYPE JUGGLING / STRINGS

INPUT	RESULT	TYPE
'Hi ' + 1	'Hi 1'	string
'Hi ' + 1.23	'Hi 1.23'	string
'Hi ' + true	'Hi 1'	string
'Hi ' + false	'Hi '	string

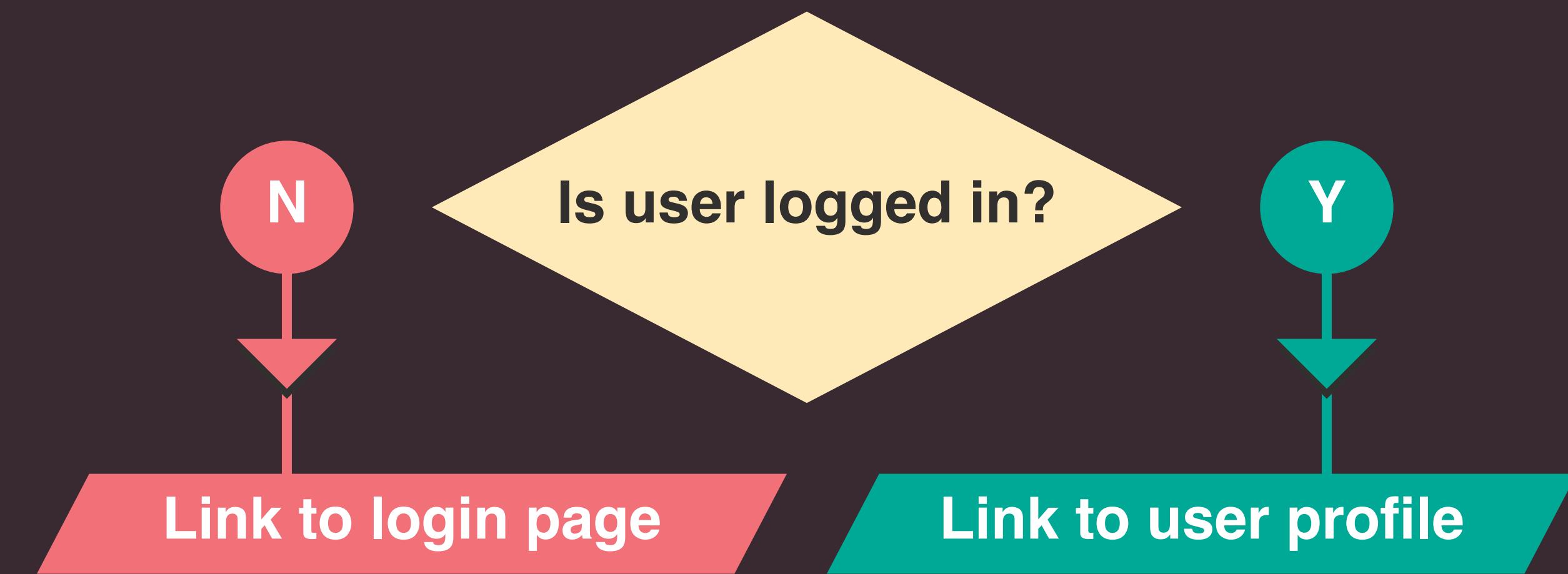
# CHAPTER 2

# CONTROL STRUCTURES

# CONDITIONAL STATEMENTS

**Conditional statements** test a condition to determine whether or not to run a code block.

# CONDITIONAL STATEMENTS



# CURLY BRACES FORM A CODE BLOCK

```
{  
    // Curly braces indicate  
    // the start and end of  
    // a code block  
}
```

# IF STATEMENT

```
if ($logged_in === true) {  
    $link = '<a href="member.php">Profile</a>';  
}
```

CONDITION TO TEST

CODE TO EXECUTE IF VALUE IS TRUE

# IF... ELSE STATEMENT

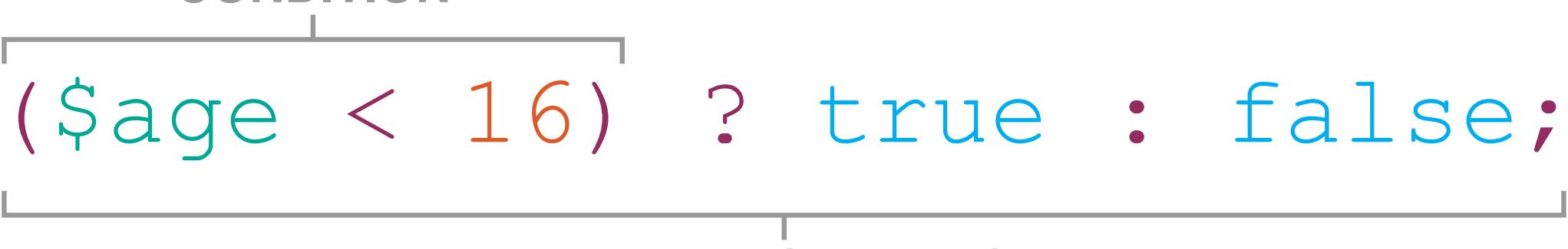
```
if ($logged_in === true) {  
    $link = '<a href="member.php">Profile</a>';  
} else {  
    $link = '<a href="login.php">Log in</a>';  
}
```

CONDITION TO TEST

CODE TO EXECUTE IF VALUE IS TRUE

CODE TO EXECUTE IF VALUE IS FALSE

# TERNARY OPERATOR

```
$child =  ($age < 16) ? true : false;
```

# IF... ELSEIF STATEMENT

```
if ($answer === true) {  
  
    $answer = 'Yes';  
  
} elseif ($answer === false) {  
  
    $answer = 'No';  
  
} else {  
  
    $answer = 'Maybe';  
  
}
```

# SWITCH STATEMENT

```
switch ($day) {  
  
    case = 'Monday':  
        $offer = '20% off chocolate';  
        break;  
  
    case = 'Tuesday':  
        $offer = '20% off mints';  
        break;  
  
    default:  
        $offer = '5% off everything';  
  
}
```

# MATCH STATEMENT

```
$offer = match ($day) {  
    'Monday' => '20% off chocolate';  
    'Tuesday' => '20% off mints';  
    default => '5% off everything';  
};
```

# LOOPS

**Loops** let you to write a set of instructions once and repeat them either:

A fixed number of times

Or until a condition is met

# WHILE LOOP

```
$while ($counter < $total) {  
    echo $counter . ' packs cost ';  
    echo $price * $counter;  
    echo '<br>';  
    $counter++;  
}  
  
INCREASE COUNTER BY 1
```

# DO WHILE LOOP

KEYWORD  
do {

```
echo $counter . ' packs cost ';  
echo $price * $counter;  
echo '<br>';  
$counter++;
```

} while (\$counter < \$total);

KEYWORD

CONDITION

# FOR LOOP

```
KEYWORD           CONDITION & EXPRESSIONS
for ($i = 0; $i < 10; $i++) {  
    echo $i;  
}  
CODE TO RUN
```

# THREE EXPRESSIONS IN FOR LOOP

INITIALIZATION      CONDITION      UPDATE  
(\$i = 0; \$i < 10; \$i++)

# FOREACH LOOP

```
KEYWORD      ARRAY      KEY        VALUE
foreach ($list as $item => $price) {  
  
    echo $item;  
    echo ' - '$;  
    echo $price;  
  
}
```

# INCLUDE FILES

**Include files** let you include the same code in multiple pages.

# INCLUDE FILES

```
<?php include 'includes/header.php'; ?>
```



The diagram illustrates the PHP `include` statement. It consists of the opening tag `<?php`, the keyword `include`, a string containing the path to the file to be included (`'includes/header.php'`), and the closing tag `?>`. A bracket under the keyword `include` is labeled `KEYWORD`. A longer bracket under the path string is labeled `PATH TO FILE TO INCLUDE`.

# CHAPTER 3

# FUNCTIONS

# DEFINING & CALLING FUNCTIONS

**A function definition** stores the statements that perform a task.

The function is **called** when the task needs to be performed.

# DEFINE FUNCTION

```
KEYWORD           FUNCTION NAME
[-----] [-----]
function write_copyright_notice()
{
    $year = date('Y');
    echo '&copy; ' . $year;
}
```

# CALL FUNCTION

```
FUNCTION NAME
[-----]
write_copyright_notice();
```

Functions usually return a value.

If the function needs information to perform its task, it should be passed in using **parameters**.

# RETURN VALUES

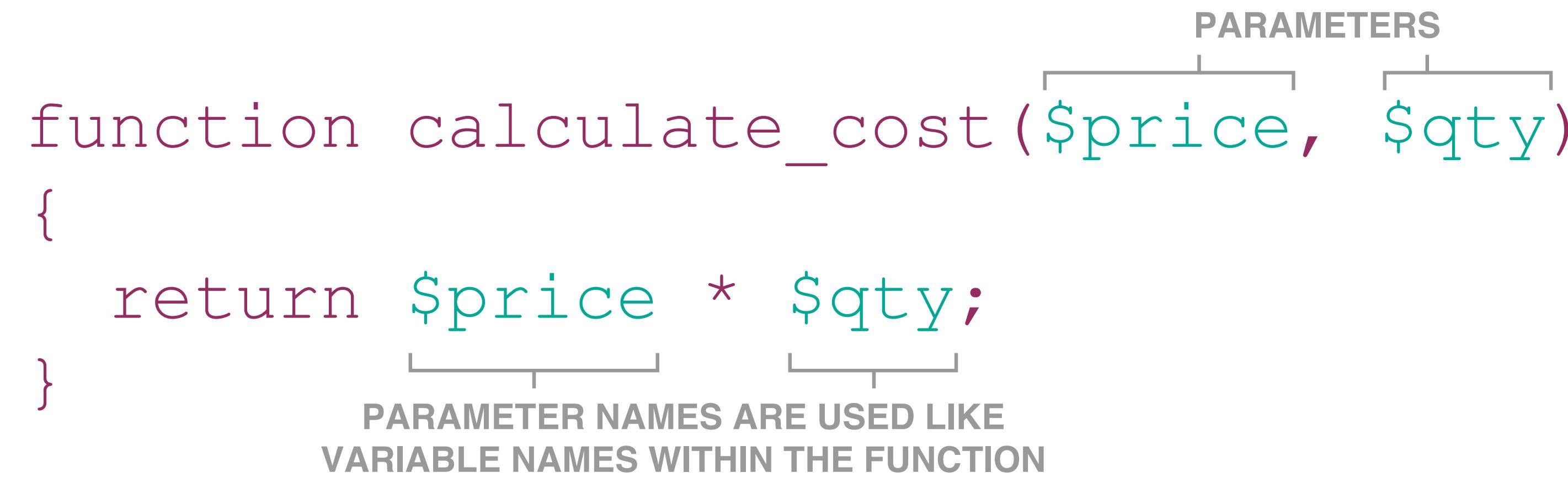
```
KEYWORD           FUNCTION NAME
[-----] [-----]
function create_copyright_notice()
{
    $year = date('Y');
    return '&copy; ' . $year;
}
```

# PARAMETERS ACT LIKE VARIABLE NAMES

```
function calculate_cost($price, $qty)
{
    return $price * $qty;
}
```

PARAMETERS

PARAMETER NAMES ARE USED LIKE  
VARIABLE NAMES WITHIN THE FUNCTION



# ARGUMENTS ARE VALUES PASSED TO PARAMETERS

ARGUMENTS

```
calculate_cost(6, 3);
```



# SCOPE

```
tax = 20;  
function calculate_total ($price, $qty)  
{  
    $cost = $price * $qty;  
    $tax = $cost * (20 / 100);  
    $total = $cost + $tax;  
    return $total;  
}
```

 GLOBAL

 LOCAL

# ARGUMENT & RETURN TYPE DECLARATIONS

When defining a function, you can specify the data type for the arguments and return type.

# ENFORCING TYPES

```
declare(strict_types = 1);
```

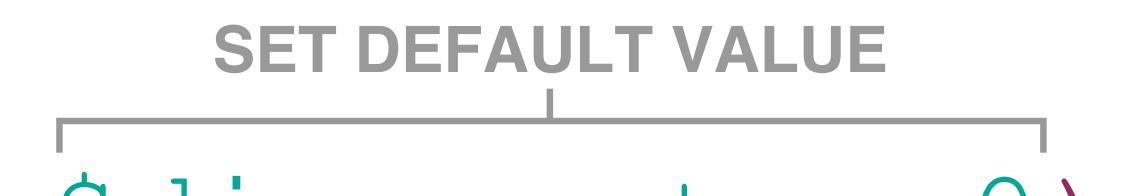
```
function calculate_cost(int $price, int $qty): int
{
    return $price * $qty;
}
```

# DEFAULT VALUES & OPTIONAL PARAMETERS

When you specify a default value for a parameter, it becomes optional.

# DEFAULT PARAMETER VALUE

```
function calculate_cost($price, $qty, $discount = 0)
{
    $cost = $cost * $qty;
    return $cost - $discount;
}
```



The diagram shows a bracket with the text "SET DEFAULT VALUE" above it, pointing to the line "\$discount = 0" in the code. This indicates that the parameter \$discount has a default value of 0 if no value is provided when the function is called.

When calling a function, if you put parameter names before arguments, the arguments can appear in any order.

# NAMED ARGUMENTS

```
calculate_cost($price: 5, $qty: 3, $discount: 5);  
calculate_cost($qty: 3, $price: 5);
```

# CHAPTER 4

# OBJECTS & CLASSES

**Objects** group together a set of variables and functions.

They are used to represent something from day-to-day life.

Inside an object:

- Variables are called **properties**
- Functions are called **methods**

**Classes** are like templates that are used to create objects.

They specify the properties and methods that the object has.

### OBJECT TYPE: ACCOUNT

DATA	VALUE
number	20489446
type	Checking
balance	1000.00

TASK	PURPOSE
deposit()	Deposit money
withdraw()	Withdraw money
get_balance()	Retrieve balance



### OBJECT TYPE: CUSTOMER

DATA	VALUE
forename	Ivy
surname	Stone
email	ivy@eg.link
password	\$2y210\$M...

TASK	PURPOSE
getFullName()	Retrieve full name
authenticate()	Check email and password match



OBJECT TYPE: ACCOUNT	
DATA	VALUE
number	10937528
type	Savings
balance	2346.00
TASK	PURPOSE
deposit()	Deposit money
withdraw()	Withdraw money
get_balance()	Retrieve balance



OBJECT TYPE: CUSTOMER	
DATA	VALUE
forename	Emiko
surname	Ito
email	emi@eg.link
password	\$2y210\$NN...
TASK	PURPOSE
getFullName()	Retrieve full name
authenticate()	Check email and password match



KEYWORD      CLASS NAME

```
class Account
{
    public int      $number;
    public string   $type;
    public float    $balance;
    public function deposit(): float
    {
        // code to deposit money here
    }
    public function withdraw(): float
    {
        // code to withdraw money here
    }
}
```

PROPERTIES

METHODS

# CREATING AN OBJECT USING A CLASS

When you create an object using a class, you must specify values for the properties.

It will automatically get any methods defined in the class.

# CREATING OBJECT USING CLASS

VARIABLE TO HOLD OBJECT                    KEYWORD                    CLASS NAME  
  |  |  |  
  \$account = new Account();

# SETTING OR UPDATING PROPERTY VALUES

\$account->number = 20148896;  
\$account->type = 'Checking';  
\$account->balance = 1000.00;

# GET OBJECT PROPERTY

echo  \$account->number;

# CALL OBJECT METHOD

echo  \$account->getBalance();

A **constructor** is a method that runs each time a class is used to create an object.

It is often used to assign values to properties when an object is created.

```
class Account
{
    → public int      $number;
    → public string   $type;
    → public float    $balance;

    public function __construct($number, $type, $balance)
    {
        $this->number   = $number; ←
        $this->type     = $type; ←
        $this->balance  = $balance; ←
    }

    public function deposit(): float    { ... }
    public function withdraw(): float   { ... }
    public function getBalance(): float { ... }
}
```

```
VARIABLE TO HOLD OBJECT          CLASS NAME          VALUES USED TO CREATE OBJECTS  
[-----] [-----] [-----]  
$account = new Account(20148896, 'Checking', 1000.00);
```

Constructor property promotion lets you declare properties in the constructor.

```
class Account
{
    public __construct(
        public int      $number;
        public string   $type;
        public float    $balance;
    )
    {
        public function deposit(): float      { ... }
        public function withdraw(): float     { ... }
        public function getBalance(): float { ... }
    }
}
```

# SECTION B

# INTRODUCTION

# HTTP REQUESTS & RESPONSES

HyperText Transfer Protocol (HTTP)  
is a set of rules that specify:

How browsers should **request** pages and  
how servers should format their **response**.

# REQUEST



Headers Preview Response Initiator Timing

▼ Request Headers View source

**Accept-Language:** en-GB,en-US;q=0.9,en;q=0.8

**Referer:** http://localhost:8888/phpbook/section\_b/intro/index.php

**User-Agent:** Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_15\_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/93.0.4577.63 Safari/537.36

# RESPONSE

x Headers Preview Response Timing

▼ Response Headers view source

**Content-Type:** text/html; charset=UTF-8

**Date:** Fri, 15 Jan 2021 15:47:46 GMT

**Server:** Apache/2.4.46 (Unix) OpenSSL/1.0.2u PHP/8.0.0

RESPONSE STATUS CODE	REASON PHRASE
200	OK
301	Moved permanently
307	Temporary redirect
403	Forbidden
404	Not found
500	Internal server error

# ENCODING SCHEMES

Computers represent text, images and audio using **binary data** (0s and 1s).

**Encoding schemes** translate the things you see or hear into the 0s and 1s that the computer processes.

# ENCODING SCHEMES



01001000



01000101



00101100



00101100



01001111

# CHARACTER ENCODING SCHEMES

SYMBOL	BINARY	BYTES
\$	00100100	1
£	11000010 10100011	2
€	11100010 10000010 10101100	3

# SUPERGLOBAL ARRAYS

Each time a PHP page runs, the PHP interpreter creates **superglobal arrays**.

They store information that the PHP code in that page can access.

# SUPERGLOBAL ARRAYS

```
$ip = ${_SERVER['REMOVE_ADDR']};
```

SUPERGLOBAL ARRAY  
|  
[ ]  
TEXT AND MARKUP TO DISPLAY

# SUPERGLOBAL ARRAYS

KEY	PURPOSE
<code>\$_SERVER['REMOTE_ADDR']</code>	Browser IP address
<code>\$_SERVER['HTTP_USER_AGENT']</code>	Type of browser
<code>\$_SERVER['HTTP_REFERER']</code>	Referring page
<code>\$_SERVER['REQUEST_METHOD']</code>	Type of HTTP request (GET/POST)
<code>\$_SERVER['HTTP_HOST']</code>	Host name (domain, IP, or localhost)
<code>\$_SERVER['QUERY_STRING']</code>	Data in query string

# BUILT-IN FUNCTIONS

The PHP interpreter comes with a set of built-in functions.

You call these functions without including the function definition in the page.

# BUILT-IN FUNCTIONS

```
$name = "Ivy";
```

```
var_dump($name);
```

```
string(3) "Ivy"  
-----|  
DATA TYPE LENGTH VALUE
```

# ERRORS

When there is a problem with the PHP code, the PHP interpreter generates error messages which help you fix any mistakes.

# ERROR MESSAGES

Error: description goes here in test.php on line 21



The diagram illustrates the structure of an error message. It consists of four horizontal bars, each with a bracket below it indicating its scope. The first bar, labeled 'ERROR LEVEL' in gray, spans from the start of the error message to the word 'in'. The second bar, labeled 'ERROR DESCRIPTION' in gray, spans from 'in' to 'test.php'. The third bar, labeled 'PHP FILE' in gray, spans from 'test.php' to 'on'. The fourth bar, labeled 'LINE NUMBER' in gray, spans from 'on' to the end of the message. The text within each bar is color-coded: 'Error:' is red, 'description goes here' is purple, 'in' is gray, 'test.php' is green, 'on' is gray, and 'line 21' is blue.

ERROR LEVEL      ERROR DESCRIPTION      PHP FILE      LINE NUMBER

# PHP INTERPRETER SETTINGS & OPTIONS

Both the PHP interpreter and Apache web server have settings and options that you can control using text files.

# PHP INTERPRETER SETTINGS

<?php phpinfo() ?>

## Core

PHP Version	8.0.0	
Directive	Local Value	Master Value
allow_url_fopen	On	On
allow_url_include	Off	Off
arg_separator.input	&	&
arg_separator.output	&	&
auto_append_file	<i>no value</i>	<i>no value</i>
auto_globals_jit	On	On
auto_prepend_file	<i>no value</i>	<i>no value</i>
browscap	<i>no value</i>	<i>no value</i>
default_charset	UTF-8	UTF-8
default_mimetype	text/html	text/html
disable_classes	<i>no value</i>	<i>no value</i>
disable_functions	<i>no value</i>	<i>no value</i>

# CHANGING SETTINGS

; Selection of values that can be changed in php.ini

PHP

```
default_charset      = "UTF-8"          ; Default character set
display_errors       = On               ; Show errors on screen
log_errors           = On               ; Write errors to log file
error_reporting      = E_ALL             ; Show all errors
upload_max_filesize = 32M              ; Max uploaded file size
post_max_size        = 32M              ; Max size of HTTP Post
max_execution_time   = 30                ; Max time script can run
memory_limit         = 128M             ; Max memory script can use
date.timezone        = "Europe/Rome"    ; Default Timezone
```

# CHAPTER 5

# BUILT-IN FUNCTIONS

The function definitions of built-in functions are built into the PHP interpreter, so they do not need to be included in the page in order to call a function.

# FUNCTIONS FOR STRINGS

String functions help you work with data that is the string data type.

# STRING CASE & LENGTH

FUNCTION	PURPOSE
<code>strtolower (\$string)</code>	Convert to lowercase
<code>strtoupper (\$string)</code>	Convert to uppercase
<code>ucwords (\$string)</code>	Convert first letter of word to uppercase
<code>strlen (\$string)</code>	Number of characters in string
<code>str_word_count (\$string)</code>	Number of words in string

# STRING CASE & LENGTH (CONTINUED)

FUNCTION	STRING: Home sweet home
<code>strtolower (\$string)</code>	home sweet home
<code>strtoupper (\$string)</code>	HOME SWEET HOME
<code>ucwords (\$string)</code>	Home Sweet Home
<code>strlen (\$string)</code>	15
<code>str_word_count (\$string)</code>	3

# FINDING CHARACTERS IN A STRING

FUNCTION	PURPOSE
<code>strpos(\$str, \$substr[, \$offset])</code>	Position of first match
<code>stripos(\$str, \$substr[, \$offset])</code>	Above but case-insensitive
<code>strrpos(\$str, \$substr[, \$offset])</code>	Position of last match
<code>strripos(\$str, \$substr[, \$offset])</code>	Above but case-insensitive
<code>stristr(\$str, \$substr)</code>	Text from first match to end
<code>stristr(\$str, \$substr)</code>	Above but case-insensitive
<code>str_contains(\$str, \$substr)</code>	If substring in string
<code>str_starts_with(\$str, \$substr)</code>	If string starts with substring

# FINDING CHARACTERS IN A STRING (CONTINUED)

FUNCTION	STRING: Home sweet home
strpos(\$text, 'ho')	11
stripos(\$text, 'me', 5)	13
strrpos(\$text, 'Ho')	0
strripos(\$text, 'Ho')	11
strstr(\$text, 'ho')	home
stristr(\$text, 'ho')	Home sweet home
str_contains(\$text, 'ho')	True
str_starts_with(\$text, 'ho')	False

# MOVING & REPLACING CHARACTERS

FUNCTION	PURPOSE
<code>ltrim(\$string[, \$delete])</code>	Remove whitespace from left
<code>rtrim(\$string[, \$delete])</code>	Remove whitespace from right
<code>trim(\$string[, \$delete])</code>	Remove whitespace from both
<code>str_replace(\$old, \$new[, \$str])</code>	Replace \$old with \$new
<code>str_ireplace(\$old, \$new[, \$str])</code>	Above but case-insensitive
<code>str_repeat(\$str, \$repeats)</code>	Repeat specified times

# MOVING & REPLACING CHARACTERS (CONTINUED)

FUNCTION	
<code>ltrim(\$text, '/')</code>	/images/uploads/
<code>rtrim(\$text, '/')</code>	images/uploads/
<code>trim(\$text, 's/')</code>	/images/uploads
<code>str_replace('images', 'img', \$text)</code>	/img/uploads/
<code>str_ireplace('images', 'img', \$text)</code>	/img/uploads/

# REGULAR EXPRESSIONS

Credit cards, zip codes, and phone numbers follow patterns of characters.

**Regular expressions** describe a pattern of characters and PHP can check for them.

# REGULAR EXPRESSIONS

FUNCTION	PURPOSE
/ [a-z] /	Any lowercase letter a-z
/ [0-9] [A-z] /	Number 0-9 and letters A-z
/ [0-9] ([a-z]{2}) /	Number 0-9 followed by two lowercase letters
/ [1-31] (st nd rd th) /	Numbers 1-31 followed by st, nd, rd or th

# REGULAR EXPRESSION FUNCTIONS

FUNCTION	PURPOSE
<code>preg_match(\$regex, \$str)</code>	Checks for pattern in string
<code>preg_match_all(\$regex, \$str)</code>	Number of matches in string
<code>preg_split(\$regex, \$str)</code>	Split at each match, return array
<code>preg_replace(\$regex, \$replace, \$str)</code>	Find and replace

# FUNCTIONS FOR NUMBERS

# NUMERIC FUNCTIONS

FUNCTION	PURPOSE
<code>round (\$num, \$places, \$round)</code>	Round floating point to integer
<code>ceil (\$num)</code>	Round up to nearest integer
<code>floor (\$num)</code>	Round down to nearest integer
<code>mt_rand (\$min, \$max)</code>	Random number between \$min and \$max

# NUMERIC FUNCTIONS (CONTINUED)

EXAMPLE	RESULT
round(9876.54321)	9877
round(9876.54321, 2)	9876.54
ceil(1.23)	2
floor(1.23)	1

# FUNCTIONS FOR ARRAYS

# ARRAY FUNCTIONS

FUNCTION	PURPOSE
<code>array_key_exists(\$key, \$array)</code>	Check for key
<code>array_search(\$value, \$array)</code>	Search for value
<code>in_array(\$value, \$array)</code>	Check if value in array
<code>count(\$array)</code>	Number of items in array
<code>array_rand(\$value, \$array)</code>	Select random item from array
<code>implode([\$separator, ]\$array)</code>	Turn array into string (with separator)
<code>explode(\$separator, \$string)</code>	Turn string into indexed array

# ADDING & REMOVING ELEMENTS FROM AN ARRAY

FUNCTION	PURPOSE
<code>array_unshift(\$array, \$items)</code>	Add item to start of array
<code>array_push(\$array, \$items)</code>	Add item to end of array
<code>array_shift(\$array)</code>	Remove first item from array
<code>array_pop(\$array)</code>	Remove last item from array
<code>array_unique(\$array)</code>	Remove duplicates from array
<code>array_merge(\$array1, \$array2)</code>	Join two or more arrays

# ARRAY SORTING FUNCTIONS

FUNCTION	PURPOSE
<code>sort (\$array)</code>	Ascending according to value (change keys)
<code>rsort (\$array)</code>	Descending according to value (change keys)
<code>asort (\$array)</code>	Ascending according to value (maintain keys)
<code>arsort (\$array)</code>	Descending according to value (maintain keys)
<code>ksort (\$array)</code>	Ascending according to key
<code>krsort (\$array)</code>	Descending according to key

# CONSTANTS

A **constant** is a name/value pair that acts like a variable. But, once its value has been assigned, it cannot be updated.

# CONSTANTS

```
define('SITE_NAME', 'Mountain Art Supplies');
```



```
const SITE_NAME = 'Mountain Art Supplies';
```



ADD OR UPDATE HEADERS

The `header()` function is used to add or update the HTTP headers that the PHP interpreter sends to the browser.

# HTTP HEADERS

```
header('Location: http://example.org');
```

HEADER

NEW URL

```
header('Content-type: application/json');
```

HEADER

MEDIA TYPE

# FILE FUNCTIONS

File functions take a file path as a parameter,  
then either return information about the file and  
its file path, or delete the file.

# FILE FUNCTIONS

FUNCTION	PURPOSE
<code>file_exists (\$path)</code>	Check if file exists
<code>filesize (\$path)</code>	Size of file in bytes
<code>mime_content_type (\$path)</code>	Media type of file
<code>basename (\$path)</code>	Basename of file from path
<code>dirname (\$path)</code>	Path to directory file is in
<code>realpath (\$path)</code>	Absolute path for the file

# CHAPTER 6

# GETTING DATA

# FROM BROWSERS

# When a browser sends data you must:

- Collect it
- Validate it
- Check if the page can process it
- Escape or sanitize any data that is shown

# COLLECTING DATA USING SUPERGLOBAL ARRAYS

# GET DATA SENT VIA HTTP GET

```
<a href="http://eg.co/go.php?city=Rome&year=2021">Rome</a>
```

The diagram illustrates the structure of a query string. A bracket labeled "NAME" spans from the question mark to the first parameter ("city"). Another bracket labeled "QUERY STRING" spans from the question mark to the end of the URL ("2021"). Below the URL, four brackets labeled "NAME" and "VALUE" alternate, corresponding to the key-value pairs: "city=Rome" and "year=2021".

```
$location = $_GET['city'];
```



The diagram illustrates the structure of the variable assignment. A horizontal bracket spans the entire variable name '\$location'. Below the left end of this bracket is the word 'NAME' in a large, bold, gray font. Another horizontal bracket spans the key part of the array, which is 'city' in this case. Below the right end of this inner bracket is the word 'KEY' in a large, bold, gray font.

# CHECKING FOR A VALUE

```
$city = $_GET['city'] ?: '';
```

**VARIABLE**      **TRY TO STORE THIS VALUE**      **IF IT DOES NOT EXIST:  
STORE BLANK STRING**

# ESCAPING OUTPUT

If values were submitted to the server then shown in a page, they must be escaped.

This ensures hackers cannot use them to run malicious scripts, known as an XSS attack.

# CHARACTERS TO ESCAPE

<

&lt;

&#60;

>

&gt;

&#62;

&

&amp;

&#38;

“”

&quot;

&#34;

’

&apos;

&#39;

# ESCAPING RESERVED CHARACTERS

htmlspecialchars (\$text) ;



```
function html_escape(string $string): string
{
    return htmlspecialchars($string, ENT_QUOTES|ENT_HTML5, 'UTF-8', true);
}
```

PHP

# HOW FORM DATA IS SENT TO THE SERVER

**Text input always sent:**

- Text input
- Number input
- Email input
- Password
- Text area

**Option sent if selected:**

- Radio button
- Select box
- Check box

# GET FORM DATA SENT VIA HTTP GET

```
<form action="join.php" method="GET">
  <p>Email: <input type="email" name="email"></p>
  <p>Age: <input type="number" name="age"></p>
  <p><input type="checkbox" name="terms" value="true">
    I agree to the terms and conditions</p>
  <p><input type="submit" value="Save"></p>
</form>
```

HTML

# GET FORM DATA SENT VIA HTTP GET (CONTINUED)

**TEXT INPUT**  
(always sent)

```
$email = $_GET['email'];
```

VARIABLE KEY

**OPTION**  
(only sent if selected)

```
$terms = $_GET['terms'] ?? false;
```

VARIABLE KEY DEFAULT VALUE

# GET FORM DATA SENT VIA HTTP POST

```
<form action="join.php" method="POST">
  <p>Email: <input type="email" name="email"></p>
  <p>Age: <input type="number" name="age"></p>
  <p><input type="checkbox" name="terms" value="true">
    I agree to the terms and conditions</p>
  <p><input type="submit" value="Save"></p>
</form>
```

HTML

# GET FORM DATA SENT VIA HTTP POST (CONTINUED)

**TEXT INPUT**  
(always sent)

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

The diagram shows the variable assignment statement '\$email = \$\_POST['email'];' with two horizontal arrows pointing to its components. The first arrow points to the identifier '\$email' and is labeled 'VARIABLE'. The second arrow points to the string '\$\_POST['email']' and is labeled 'KEY'.

**OPTION**  
(only sent if selected)

```
$terms = $_POST['terms'] ?? false;
```

The diagram shows the variable assignment statement '\$terms = \$\_POST['terms'] ?? false;' with three horizontal arrows pointing to its components. The first arrow points to the identifier '\$terms' and is labeled 'VARIABLE'. The second arrow points to the string '\$\_POST['terms']' and is labeled 'KEY'. The third arrow points to the value 'false' and is labeled 'DEFAULT VALUE'.

# VALIDATION

When data is collected, it should be **validated** to:

- Ensure required values have been provided
- Check data is in a usable format / range

# VALIDATE NUMBERS

```
function is_number($number, int $min = 0, int $max = 100)
{
    return ($number >= $min and $number <= $max);
}
```



# VALIDATE TEXT LENGTH

```
function is_text($text, int $min = 0, int $max = 100)
{
    $length = mb_strlen($text);
    return ($length >= $min and $length <= $max);
}
```



GREATER THAN MINIMUM                            LESS THAN MAXIMUM

# VALIDATE WITH REGULAR EXPRESSION

```
function is_password(string $password)
{
    if (
        mb_strlen($password) >= 8
        and preg_match('/[A-Z]/', $password)
        and preg_match('/[a-z]/', $password)
        and preg_match('/[0-9]/', $password)
    ) {
        return true; // Passed all checks
    }
    return false; // Invalid
}
```

# VALIDATE OPTIONS

```
$star_ratings = [1, 2, 3, 4, 5,];  
  
$valid = in_array($stars, $star_ratings);
```



# VALIDATE CHECKBOX

```
$tc = isset($_POST['tc']) and $_POST['tc'] == true ? true : false
```



IF VALUE IN SUPERGLOBAL                            AND VALUE IS TRUE

# VALIDATE MULTIPLE VALUES

```
$user['name']      = $_POST['name'];
$user['age']       = $_POST['age'];
$user['terms']     = (isset($_POST['terms']) and
                      $_POST['terms'] == true) ? true : false;
```

PHP

```
$errors['name']   = is_text($user['name'], 2, 20)
                     ? '' : 'Name must be 2-20 characters';
$errors['age']     = is_number($user['age'], 16, 65)
                     ? '' : 'You must be 16-65';
$errors['terms']   = $user['terms']
                     ? '' : 'You must agree to the terms';
```

PHP

# VALIDATE MULTIPLE VALUES (CONTINUED)

```
$invalid = implode($errors);
```

PHP

```
if ($invalid) {  
    // Show error messages and do not process data  
} else {  
    // Data is valid and can process data  
}
```

PHP

# FILTER FUNCTIONS

PHP has two **filter functions** that can get data sent from the browser and store it in variables.

They are called filter functions because they can apply a filter to the data that the browser sent.

# FILTER INPUT

```
$data = filter_input(INPUT_SOURCE, 'name');
```

INPUT SOURCE      NAME

```
$data = filter_input_array(INPUT_SOURCE);
```

INPUT SOURCE

## INPUT SOURCE DESCRIPTION

<i>INPUT_GET</i>	Collects data sent via HTTP GET
------------------	---------------------------------

---

<i>INPUT_POST</i>	Collects data sent via HTTP POST
-------------------	----------------------------------

# FILTERS

FILTER	DESCRIPTION
FILTER_VALIDATE_BOOLEAN	Is value true, 1, on, yes?
FILTER_VALIDATE_INT	Is number an integer?
FILTER_VALIDATE_FLOAT	Is number a float?
FILTER_VALIDATE_EMAIL	Does structure of string match email?
FILTER_VALIDATE_URL	Does structure of string match URL?
FILTER_VALIDATE_DOMAIN	Does structure of string match domain name?
FILTER_VALIDATE_IP	Does structure of string match IP address?
FILTER_VALIDATE_REGEXP	Does structure of string match RegEx?

# CHAPTER 7

# IMAGES & FILES

# UPLOADING FILES

Information about uploaded files is made available in the `$_FILES` superglobal array.

Files are saved in a temporary location, and must be moved to a folder where it will be stored.

# FILE UPLOAD FORM

```
<form method="post" action="filename.php"
      enctype="multipart/form-data">
    <label for="image"><b>Upload file:</b></label>
    <input type="file" name="image" accept="image/*"><br>
    <input type="submit" value="Upload">
</form>
```

MUST SEND USING HTTP POST      WHERE FORM IS SUBMITTED

REQUIRED FOR ALL FILE UPLOADES

# `$_FILES` SUPERGLOBALS

```
<input type="file" name="image" accept="image/*"><br>
```

```
$_FILES['image']['name']
```

KEY	DESCRIPTION
name	File name
tmp_name	Temporary location
size	Size in bytes
type	Media type (according to browser)
error	0 if success, error code if problem

# BASIC FILE UPLOAD

```
if ($_SERVER['REQUEST_METHOD'] == 'POST') {  
    if ($_FILES['image']['error'] === 0) {  
        $message = 'File: ' . $_FILES['image']['name'] . '<br>';  
        $message .= 'Size: ' . $_FILES['image']['size'];  
    } else {  
        $message = 'The file could not be uploaded.';  
    }  
}
```

PHP

# MOVING UPLOADED FILE

```
$move_to = '.../uploads/' . $_FILES['image']['name'];  
  
move_uploaded_file($_FILES['image']['tmp_name'], $move_to);
```



The diagram illustrates the components of a file path in the provided PHP code. It uses brackets with labels to identify each part:

- UPLOAD FOLDER**: Points to the directory path '.../uploads/'.
- FILENAME**: Points to the file name '\$\_FILES['image']['name']'.
- TEMPORARY LOCATION**: Points to the temporary file location '\$\_FILES['image']['tmp\_name']'.
- DESTINATION FILEPATH**: Points to the destination file path '\$move\_to'.

# VALIDATE FILE SIZE

```
ERROR (IF TOO BIG FOR SERVER SETTINGS)
$error = ($_FILES['image']['error'] === 1) ? 'Too big' : '';
$error = ($_FILES['image']['size'] <= 5243880) ? '' : 'Too big';
FILE SIZE
```

# VALIDATE TYPE

```
$allowed_types = ['image/jpeg', 'image/png', 'image/gif',];
```

```
$type = mime_content_type($_FILES['image']['tmp_name']);
```

```
$error = in_array($type, $allowed_types) ? '' : 'Wrong  
type';
```

ARRAY OF PERMITTED FILE TYPES

TYPE OF FILE

IS THIS TYPE IN ARRAY OF PERMITTED TYPES?

# VALIDATE EXTENSION

```
ARRAY OF PERMITTED FILE EXTENSIONS
$allowed_exts = ['jpeg', 'jpg', 'png', 'gif',];  
  
$filename = strtolower($_FILES['image']['name']);  
  
GET FILE EXTENSION
$ext = pathinfo($filename, PATHINFO_EXTENSION);  
  
$error = in_array($ext, $allowed_exts) ? '' : 'Wrong  
extension';  
  
IS THIS EXTENSION IN ARRAY OF PERMITTED EXTENSIONS?
```

# RESIZING IMAGES

When users upload images, sites often resize them so that they are a similar size. This makes the page look neater and faster to load.

To resize an image you need to know its **ratio**.

# IMAGE RATIOS

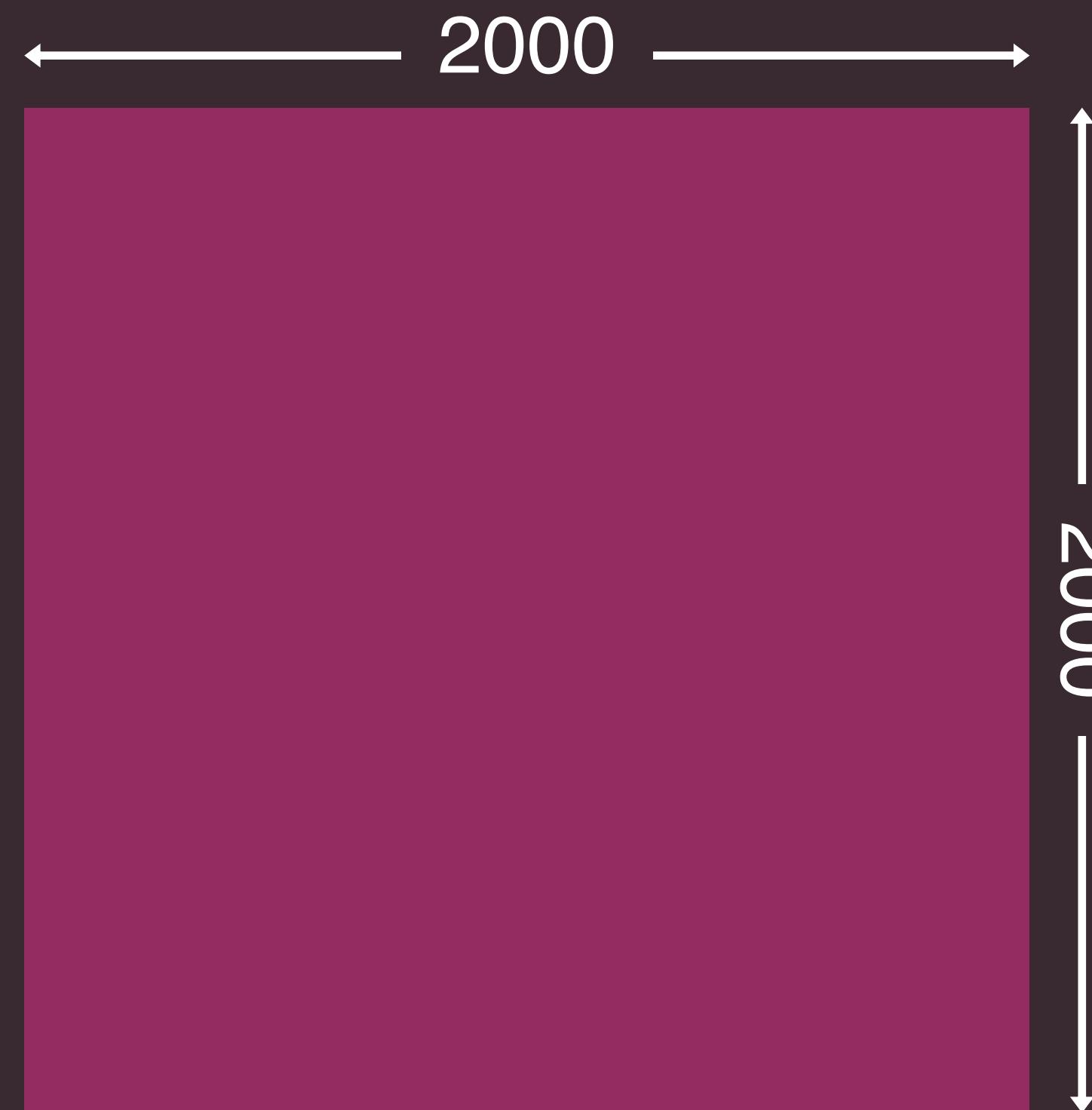
Landscape

$$2000 \div 1600 = 1.25$$



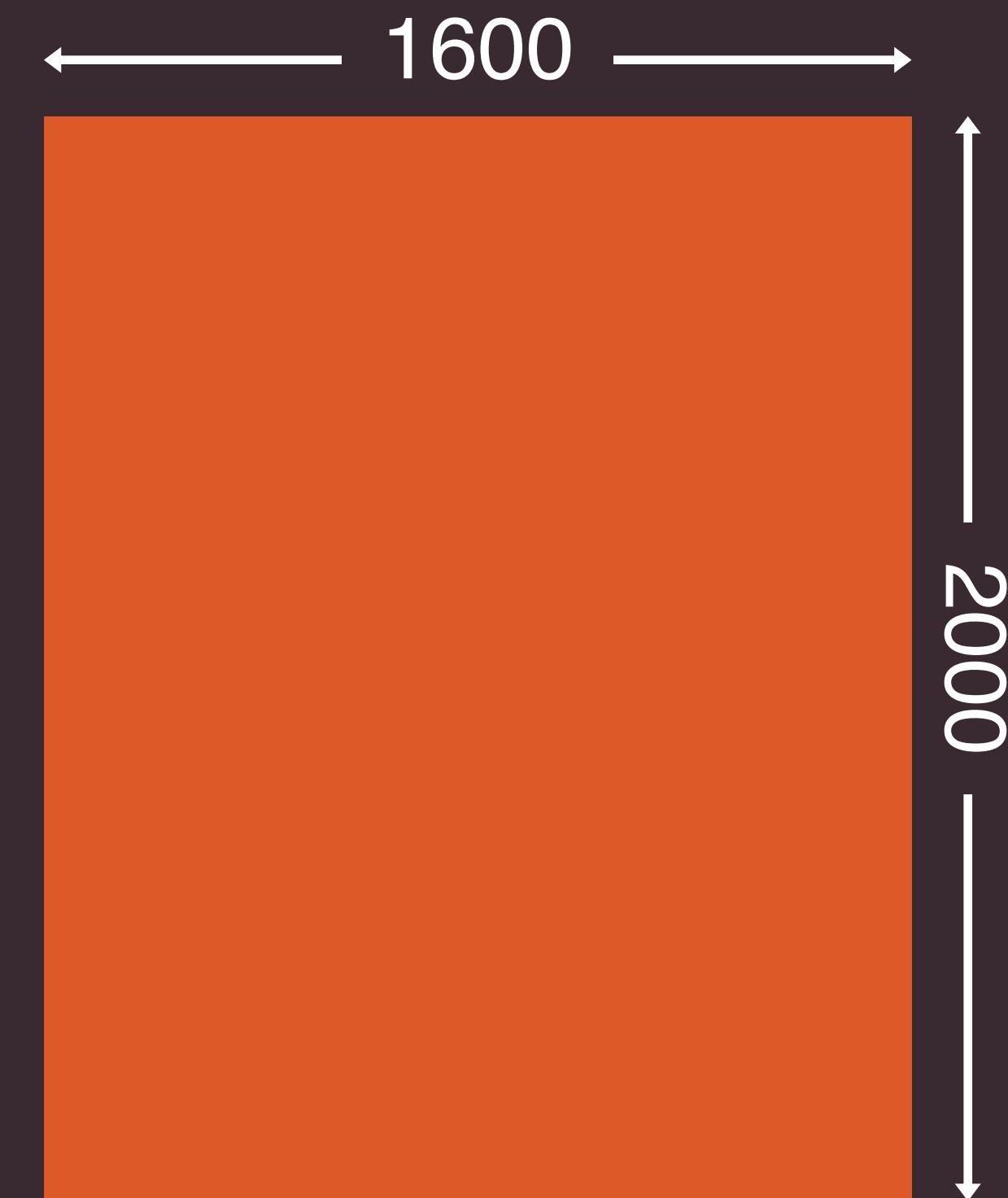
Square

$$2000 \div 2000 = 1$$



Portrait

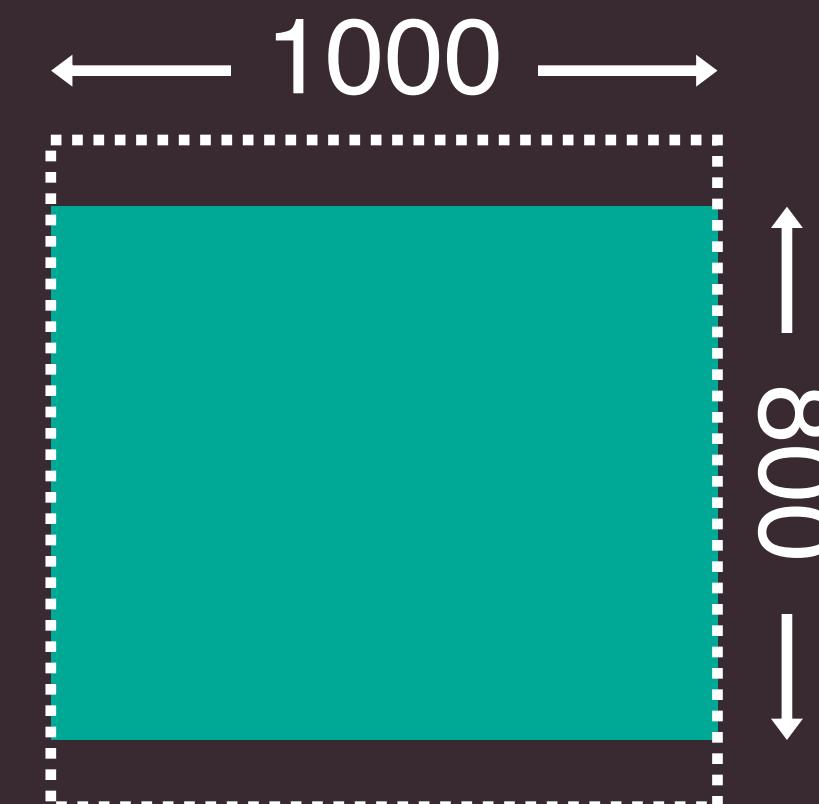
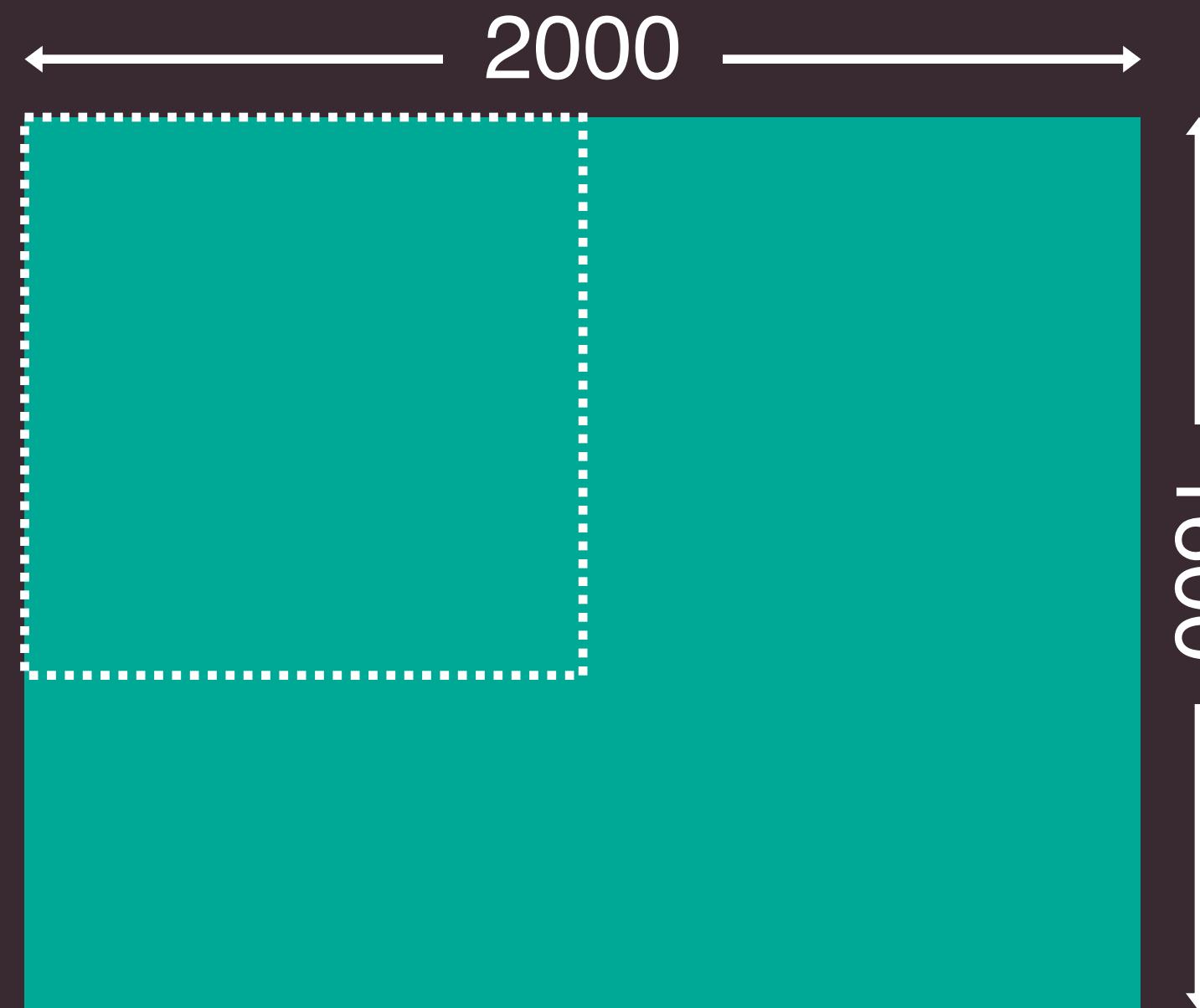
$$1600 \div 2000 = 0.8$$



# RESIZING LANDSCAPE IMAGES

Landscape

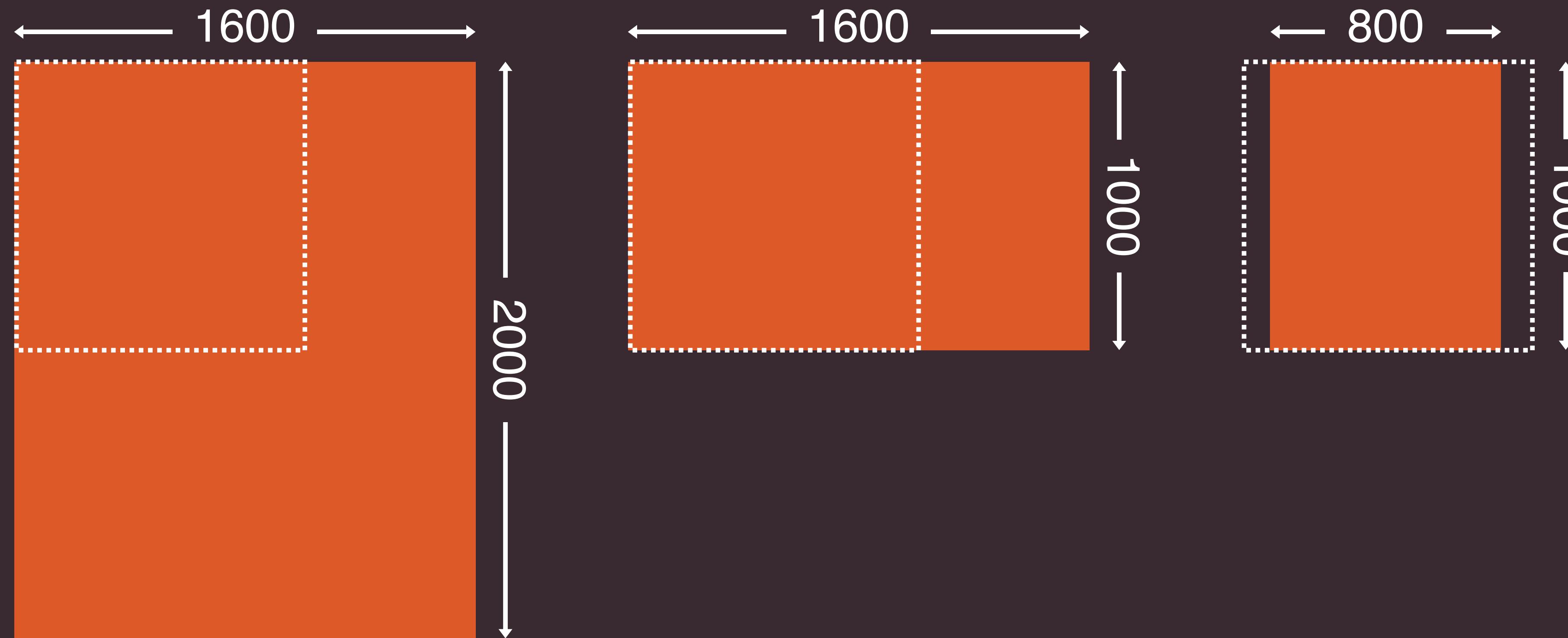
$$2000 \div 1600 = 1.25$$



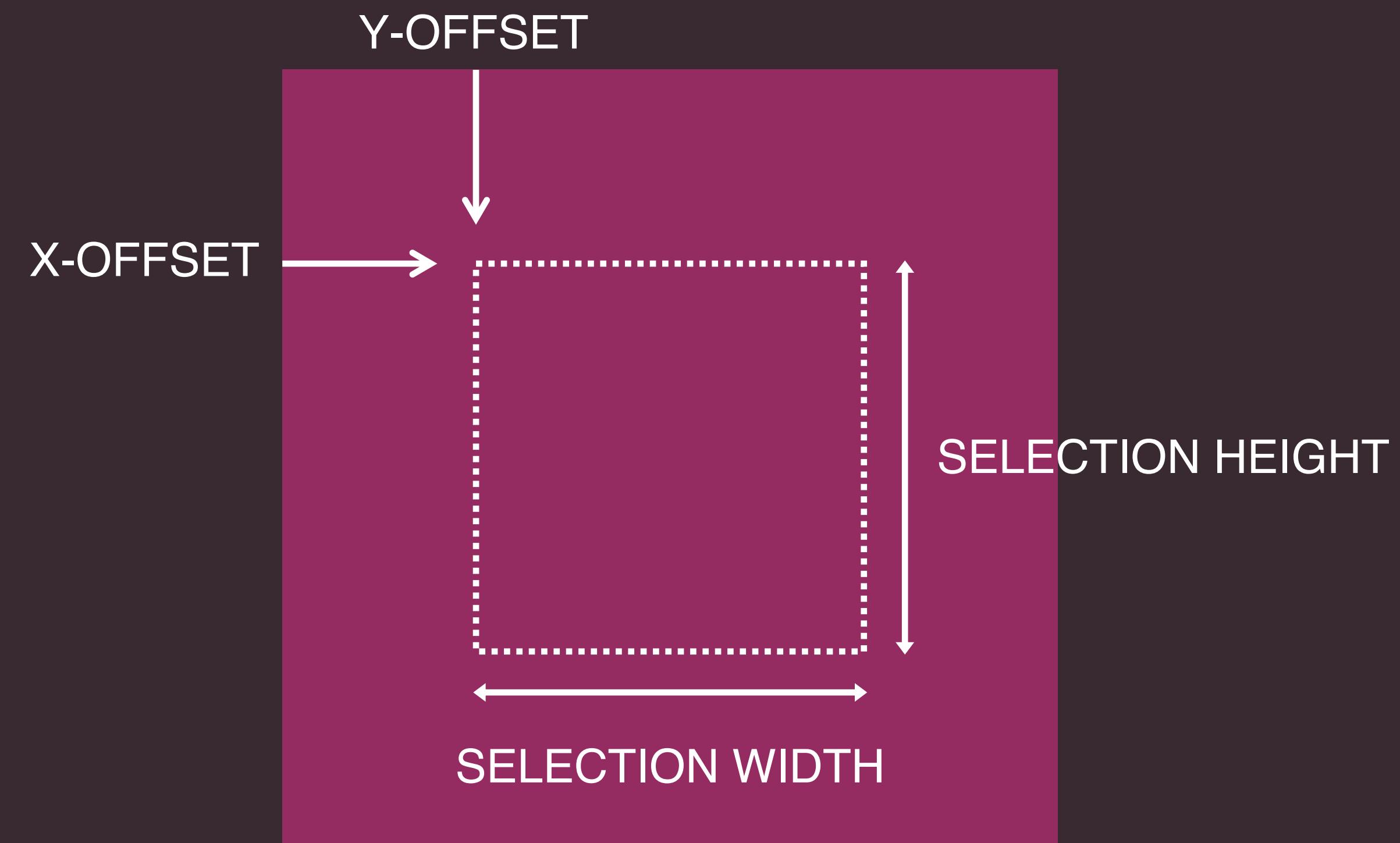
# RESIZING PORTRAIT IMAGES

**Portrait**

$$1600 \div 2000 = 0.8$$



# CROPPING IMAGES



# IMAGE EXTENSIONS

Extensions add functionality to the PHP interpreter, allowing it to perform extra tasks.

GD and Imagick are two popular extensions that help the PHP interpreter resize and crop images.

# IMAGE EXTENSIONS

## GD

- Installed with PHP
- More complicated code  
(requires use of more functions)
- Separate functions used for  
different file formats

## Imagick

- Not always installed with PHP
- Much less code to use

# USING IMAGICK

VARIABLE TO HOLD IMAGE      CLASS NAME      PATH TO IMAGE  
  |                              |                          |  
`$image = new Imagick($filepath);`

CROP AND RESIZE IMAGE      WIDTH      HEIGHT      MAX VALUES  
  |                              |                          |  
`$image->thumbnailImage(200, 200, true);`

SAVE IMAGE      WHERE TO SAVE  
  |                              |  
`$image->writeImage($destination);`

# CHAPTER 8

# DATES & TIMES

Dates and times are written in different ways.

PHP provides built-in functions and classes that help process and display dates and times.

# COMPONENTS OF DATES

COMPONENT	CAN BE WRITTEN AS
<b>Day of Week:</b>	Sat / Saturday
<b>Day of Month:</b>	9 / 09 / 9th
<b>Month:</b>	4 / 04 / Apr / April
<b>Year:</b>	22 / 2022

# DATE FORMAT CHARACTERS

## Day of Week

CHARACTER	DESCRIPTION	EXAMPLE
D	First three letters	Sat
1	Full name	Saturday

# DATE FORMAT CHARACTERS

## Day of Month

CHARACTER	DESCRIPTION	EXAMPLE
d	Digits with leading zero	09
j	Digits no leading zero	9
s	Suffix	th

# DATE FORMAT CHARACTERS

## Month

CHARACTER	DESCRIPTION	EXAMPLE
m	Digits with leading zero	04
n	Digits no leading zero	4
M	First three letters	Apr
F	Full name	April

# DATE FORMAT CHARACTERS

## Year

CHARACTER	DESCRIPTION	EXAMPLE
Y	Four digits	2022
y	Two digits	22

# EXAMPLE DATE FORMATS

## CHARACTERS    DATE FORMAT

l m j Y           Saturday April 6 2022

D jS F Y           Sat 6th April 2022

n/j/Y           4/6/2022

m/d/y           04/06/22

m-d-Y           04-06-2022

d F Y           04 September 2022

jS F Y           4th September 2022

## CHARACTERS    DATE FORMAT

F j Y           September 4 2022

M d Y           Sep 04 2022

m/d/Y           09/04/2022

Y/m/d           2022/09/04

d-m-Y           04-09-2022

n-j-Y           9-4-2022

d.m.y           04.09.22

# COMPONENTS OF TIMES

COMPONENT	CAN BE WRITTEN AS
-----------	-------------------

<b>Hour:</b>	8 / 08 / 20
--------------	-------------

<b>Minutes:</b>	35
-----------------	----

<b>Seconds:</b>	55
-----------------	----

<b>AM / PM:</b>	am / AM
-----------------	---------

# TIME FORMAT CHARACTERS

## Hour

CHARACTER	DESCRIPTION	EXAMPLE
h	12 hour leading zero	08
g	12 hour no leading zero	8
H	24 hour leading zero	08
G	24 hour no leading zero	8

# TIME FORMAT CHARACTERS

## Minute

CHARACTER	DESCRIPTION	EXAMPLE
i	Digits leading zero	09

# TIME FORMAT CHARACTERS

## Second

CHARACTER	DESCRIPTION	EXAMPLE
s	Digits leading zero	04

# TIME FORMAT CHARACTERS

## AM / PM

CHARACTER	DESCRIPTION	EXAMPLE
a	Lowercase	am
A	Uppercase	AM

# EXAMPLE TIME FORMATS

## 12-Hour Time

CHARACTERS	TIME FORMAT
ga	4am
g:i a	4:08 am
g:i:s a	4:08:37 am
g.i.s a	4.08.37 am

## 24-Hour Time

CHARACTERS	TIME FORMAT
H:i	04:08
H:i:s	04:08:37
His	040837
H.i.s	04.08.37

# UNIX TIMESTAMPS

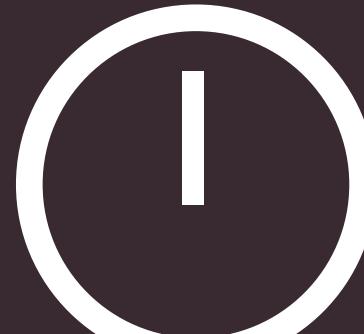
Unix timestamps represent dates and times using the number of seconds that have elapsed since midnight on January 1, 1970.

DAT  
E

31  
DEC  
1969

+

TIME



23:59:00

UNIX  
TIMESTAMP

-60

1  
JAN

1970

+



00:02:00

=

120

11  
APR

1975

+



11:00:00

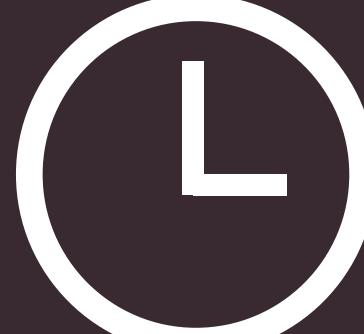
=

166878000

31  
DEC

2020

+



15:00:00

=

1609426800

# DATE FUNCTIONS

PHP has built-in functions that can create Unix timestamps to represent dates and times.

It also has functions to convert these Unix timestamps into a format that is easy to read.

# DATE FUNCTIONS

KEY	DESCRIPTION
<code>time()</code>	Unix timestamp of current date
<code>strtotime(\$string)</code>	Unix timestamp from string
<code>mktime(H, i, s, n, j, Y)</code>	Converts date/time parts into Unix timestamp
<code>date(\$format[, \$timestamp])</code>	Converts Unix timestamp into human-readable format using format characters

# DATE OBJECTS

PHP's `DateTime` class is used to create objects that represent dates and times.

# DATETIME OBJECT

```
VARIABLE           CLASS NAME          DATE / TIME  
$date = new DateTime('2001-02-01 15:01:05');
```

METHOD	DESCRIPTION
format (\$format [, \$DateTimeZone])	Get date and time in specified format
getTimestamp	Get Unix timestamp of object

# SET / UPDATE DATE & TIME

## METHOD

`setDate ($year, $month, $day)`

## DESCRIPTION

Sets a date for the object

`setTime ($hour, $minute[, $seconds] [, $microseconds])`

Sets a time for the object

`setTimestamp ($timestamp)`

Sets date/time with a Unix timestamp

`modify ($DateFormat)`

Updates the date/time using a string

`add ($DateInterval)`

Adds an interval of time

`sub ($DateInterval)`

Subtracts an interval of time

# DATETIME OBJECT

```
<?php  
    $start = new DateTime();  
    $start->setDate(2021, 12, 01);  
    $start->setTime(17, 30);  
  
    $end = clone $start;  
    $end->modify('+2 hours 15 min');  
?  
  
Start: <?= $start->format('g:i a - D, M j Y') ?><br>  
End:   <?= $end->format('g:i a - D, M j Y') ?>
```

PHP

PHP's DateInterval class is used to create objects that represent an interval of time.

It can be measured in years, months, weeks, days, hours, minutes and seconds.

# DATEINTERVAL OBJECT

```
$interval = new DateInterval('P1M');
```

INTERVAL	REPRESENTED
1 year	P1Y
2 months	P2M
3 days	P3D
1 hour	PT1H
30 minutes	PT30M
15 seconds	PT15S
1 year, 1 day, 1 hour, and 30 minutes	P1Y1DT1H30M

# DATEINTERVAL OBJECT DIFFERENCE

```
$interval->diff($start, $end);  
$interval->format(' %h hours %i minutes');
```

## FORMAT CHARACTER    DESCRIPTION

%y                      Years

%m                      Months

%d                      Days

%h                      Hours

%i                      Minutes

%s                      Seconds

%f                      Microseconds

# DATEINTERVAL OBJECT

PHP

```
<?php
    $today      = new DateTime();
    $event       = new DateTime('2025-12-31 20:30');
    $countdown  = $today->diff($event);
    $earlybird   = new DateTime();
    $interval    = new DateInterval('P1M');
    $earlybird->add($interval);

?>
Countdown to event:<br>
<?= $countdown->format('%y years %m months %d days') ?>
50% off tickets bought by:<br>
<?= $earlybird->format('D d M Y, g:i a') ?>
```

PHP's DatePeriod class is used to create an object that represents recurring events.

It stores a set of DateTime objects at regular intervals between a start and end date.

# DATEPERIOD OBJECT

```
$period = new DatePeriod($start, $interval, $end);
```

```
foreach ($period as $occurrence) {  
    echo $occurrence->format('Y jS F');  
}
```

**DatePeriod HOLDS  
DateTime OBJECTS**

**VARIABLE NAME TO REPRESENT  
EACH DateTime OBJECT**

# DATEPERIOD OBJECT

```
<?php  
    $start = new DateTime('2025-1-1');  
    $end   = new DateTime('2026-1-1');  
  
    $interval = new DateInterval('P1M');  
    $period   = new DatePeriod($start, $interval, $end);  
?  
  
<?php foreach ($period as $event) { ?>  
    <?= $event->format('l') ?>,  
    <?= $event->format('M j Y') ?><br>  
<?php } ?>
```

PHP

PHP's `DateTimeZone` class is used to create an object that represents a time zone.

# DATEPERIOD OBJECT

**VARIABLE**                    **CLASS NAME**                    **INTERVAL**  
  └─────────────────┘    └─────────────────┘    └─────────────────┘  
`$tz = new DateTimeZone('Europe/London');`

METHOD	DESCRIPTION
getName()	Name of timezone
getLocation()	Indexed array with following keys: country_code, latitude, longitude, comments
getOffset()	Offset from UTC in seconds
getTransitions()	Array indicating when daylight savings takes effect

# DATETIMEZONE OBJECT

```
<?php  
    $tz_LDN = new DateTimeZone('Europe/London');  
    $tz_TYO = new DateTimeZone('Asia/Tokyo');  
    $LDN   = new DateTime('now', $tz_LDN);  
    $TYO   = new DateTime('now', $tz_TYO);  
?  
  
LDN: <?= $LDN->format('g:i a') ?>  
      (<?= ($LDN->getOffset() / (60 * 60)) ?>) <br>  
TYO: <?= $TYO->format('g:i a') ?>  
      (<?= ($TYO->getOffset() / (60 * 60)) ?>) <br>
```

PHP

# CHAPTER 9

# COOKIES & SESSIONS

# COOKIES

A website can tell a browser to store data about the user in a text file called a **cookie**.

Each time the browser requests another page from that site, the browser sends the data in the cookie back to the server.

- Like a variable (the filename is like the variable name)
- Value is in file (up to 4,096 characters)
- Only sent to / from the browser it was created in
- Only sent to the site that created it
- Lasts until browser closes (unless set for longer)

# SET COOKIE

```
setcookie ($name, $value);  
          └─────────┘ └─────────┘  
          COOKIE NAME   COOKIE VALUE
```

# GET COOKIE VALUE

```
$preference = ${_COOKIE['name']} ?? null;
```



```
$preference = filter_input(INPUT_COOKIE, $name);
```



# SET & ACCESS COOKIES

```
<?php  
$counter = $_COOKIE['counter'] ?? 0;  
$counter = $counter + 1;  
setcookie('counter', $counter);  
$message = 'Page views: ' . $counter;  
?>
```

```
<h1>Welcome</h1>  
<p><?= $message ?></p>
```

PHP

# SETCOOKIE PARAMETERS

```
setcookie($name[, $value, $expire, $path, $domain, $secure, $httponly]);
```

PARAMETER	DESCRIPTION
<i>\$name</i>	Cookie name
<i>\$value</i>	Cookie value
<i>\$expire</i>	When browser should stop sending cookie
<i>\$path</i>	If only needed for part of site, specify path
<i>\$domain</i>	If only needed for subdomain, set URL of subdomain
<i>\$secure</i>	If given a value of <code>true</code> , only sent if page requested over HTTPS
<i>\$httponly</i>	If given a value of <code>true</code> , only sent to server (not accessed via JS)

# SESSIONS

**Sessions** store user data on the server.

They are called sessions because they store the data for the duration of a single visit to the site.

# CREATING A SESSION INVOLVES THE SERVER

1. Creating a **session id** to identify each user
2. Creating a **session file** on the server to store user data;  
its name is the session id
3. Telling the browser to create a **session cookie**;  
the value it stores is the session id

- Session cookie expires when browser closed
- Session file can be deleted after specified time if it is not modified (default 20 minutes)
- Every page should call `session_start()` (this modifies the session)

# START / UPDATE SESSION

```
session_start();
```

# STORE DATA IN SESSION

SESSION SUPERGLOBAL	KEY	VALUE
<code>\$_SESSION</code>	<code>['name']</code>	<code>'Ivy')</code> ;
<code>\$_SESSION</code>	<code>['age']</code>	<code>27);</code>

# GET DATA FROM SESSION

VALUE	SESSION SUPERGLOBAL	KEY	DEFAULT VALUE IF NOT SET
\$name	= \$_SESSION['name']	??	null;
\$age	= \$_SESSION['age']	??	null);

# SET & ACCESS SESSION DATA

```
<?php
    session_start();
    $counter = $_SESSION['counter'] ?? 0;
    $counter = $counter + 1;
    $_SESSION['counter'] = $counter;
    $message = 'Page views: ' . $counter;
?>
```

```
<h1>Welcome</h1>
<p><?= $message ?></p>
```

PHP

# CHAPTER 10

# ERROR HANDLING

# ERRORS

**Errors** are messages that the PHP interpreter creates if it has a problem running your code.

Some errors stop code running, others do not.

# ERROR SETTINGS FOR PHP.INI

## Development

```
display_errors = On  
log_errors     = On  
error_reporting = E_ALL
```

PHP

## Live

```
display_errors = Off  
log_errors     = On  
error_reporting = Off
```

PHP

# ERROR SETTINGS FOR .HTACCESS

## Development

```
php_flag display_errors On  
php_flag log_errors On  
php_value error_reporting -1
```

PHP

## Live

```
php_flag display_errors Off  
php_flag log_errors On  
php_value error_reporting -1
```

PHP

# EXAMPLE OF ERROR

PHP code to cause error:

```
<?= 'Quotes do not match'; ?>
```

PHP

Error shown on screen:

```
Parse error: syntax error, unexpected  
string content "Quotes do not match";  
in /Users/Jon/sites/sample-error.php  
on line 2
```

Error saved in log file:

```
[27-Jan-2021 14:41:13 UTC] PHP Parse error: syntax error,  
unexpected string content "Quotes do not match";  
in /Users/Jon/sites/sample-error.php  
on line 2
```

# UNDERSTANDING ERROR MESSAGES



A **parse error** is caused by a problem with the syntax of your code.

It prevents the page being displayed because the PHP interpreter cannot understand the code.

# PARSE ERRORS

```
1 <?php  
2 $username = 'Ivy'  
3 $order      = ['pencil', 'pen', 'notebook', ];  
4 ?>
```

PHP

**Parse error:** syntax error, unexpected variable "\$order";  
in **/Users/Jon/sites/sample-error.php**  
on line **3**

# PARSE ERRORS

```
1 <?php  
2 $username = 'Ivy';  
3 $order      = ['pencil', 'pen', 'notebook', ];  
4 ?>
```

PHP

**Parse error:** syntax error, unexpected variable "\$order";  
in **/Users/Jon/sites/sample-error.php**  
on line **3**

# PARSE ERRORS

```
1 <?php  
2 $username = 'Ivy';  
3 $order     = ['pencil', 'pen', 'notebook', );  
4 ?>
```

PHP

**Parse error:** Unclosed '[' does not match ')';  
in **/Users/Jon/sites/sample-error.php**  
on line **3**

# PARSE ERRORS

```
1 <?php  
2 $username = 'Ivy';  
3 $order     = ['pencil', 'pen', 'notebook', ];  
4 ?>
```

PHP

**Parse error:** Unclosed '[' does not match ')';  
in **/Users/Jon/sites/sample-error.php**  
on line **3**

# PARSE ERRORS

```
1 <?php  
2 $username = 'Ivy';  
3 order      = ['pencil', 'pen', 'notebook', ];  
4 ?>
```

PHP

**Parse error:** Unexpected identifier "order"  
in **/Users/Jon/sites/sample-error.php**  
on line **3**

# PARSE ERRORS

```
1 <?php  
2 $username = 'Ivy';  
3 $order      = ['pencil', 'pen', 'notebook',];  
4 ?>
```

PHP

**Parse error:** Unexpected identifier "order"  
in **/Users/Jon/sites/sample-error.php**  
on line **3**

A **fatal error** is raised when the PHP interpreter finds a problem that prevents it from processing any more code in that page.

This means users may see a partial page or a blank page.

# FATAL ERRORS

```
1 <?php  
2     function total(int $price, int $quantity) { ... }  
6 ?>  
7 <h2>Basket</h2>  
8 <?= totals(3, 5) ?>
```

PHP

**Fatal error:** Uncaught Error: Call to undefined function totals() in  
**/Users/Jon/sites/sample-error.php**  
Stack trace: #0 {main} thrown in  
**/Users/Jon/sites/sample-error.php** on line 4

# FATAL ERRORS

```
1 <?php  
2     function totals(int $price, int $quantity) {...}  
6 ?>  
7 <h2>Basket</h2>  
8 <?= totals(3, 5) ?>
```

PHP

**Fatal error:** Uncaught Error: Call to undefined function totals() in  
**/Users/Jon/sites/sample-error.php**  
Stack trace: #0 {main} thrown in  
**/Users/Jon/sites/sample-error.php** on line 4

# FATAL ERRORS

```
1 <?php  
2     function total(int $price, int $quantity) { ... }  
6 ?>  
7 <h2>Basket</h2>  
8 <?= total(3) ?>
```

PHP

**Fatal error:** Uncaught ArgumentCountError: Too few arguments to function total(), 1 passed in **/Users/Jon/sites/sample-error.php** on line **8** and exactly 2 expected in **/Users/Jon/sites/sample-error.php...**

# FATAL ERRORS

```
1 <?php  
2     function total(int $price, int $quantity) { ... }  
6 ?>  
7 <h2>Basket</h2>  
8 <?= total(3, 2) ?>
```

PHP

**Fatal error:** Uncaught ArgumentCountError: Too few arguments to function total(), 1 passed in **/Users/Jon/sites/sample-error.php** on line **8** and exactly 2 expected in **/Users/Jon/sites/sample-error.php...**

# FATAL ERRORS

```
1 <?php $basket = new Basket(); ?><h2>Basket</h2>
```

PHP

**Fatal error:** Uncaught Error: Class 'Basket' not found in  
**/Users/Jon/sites/sample-error.php**  
Stack trace: #0 {main} thrown in  
**/Users/Jon/sites/sample-error.php** on line 1

# FATAL ERRORS

```
1 <?php include 'classes/Basket.php';  
2 $basket = new Basket(); ?><h2>Basket</h2>
```

PHP

**Fatal error:** Uncaught Error: Class 'Basket' not found in  
**/Users/Jon/sites/sample-error.php**  
Stack trace: #0 {main} thrown in  
**/Users/Jon/sites/sample-error.php** on line 1

A **non-fatal error** is raised when the PHP interpreter thinks that there could be an error, but it will attempt to run the rest of the page.

# NON-FATAL ERRORS

```
1 <?php $list = false; ?>
2 <h1>Basket</h1>
3 <?php foreach ($list as $item) { ?>
4   Item: <?= $item ?><br>
5 <?php } ?>
```

PHP

**Warning:** foreach() argument must be of type array|object, bool given in  
**/Users/Jon/sites/sample-error.php** on line 3

# NON-FATAL ERRORS

```
1 <?php $list = ['carrot', 'swede', 'parsnip']; ?>
2 <h1>Basket</h1>
3 <?php foreach ($list as $item) { ?>
4   Item: <?= $item ?><br>
5 <?php } ?>
```

PHP

**Warning:** foreach() argument must be of type array|object, bool given in  
**/Users/Jon/sites/sample-error.php** on line 3

# NON-FATAL ERRORS

```
1 <?php include 'header.php'; ?>
2 <h1>Basket</h1>
```

PHP

**Warning:** include(header.php): Failed to open stream: No such file or directory in **/Users/Jon/sites/sample-error.php** on line **1**

**Warning:** Failed opening 'header.php' for inclusion  
(include\_path='.:./Applications/MAMP/bin/php/php8.0/lib/php') in  
**/Users/Jon/sites/sample-error.php** on line **1**

# NON-FATAL ERRORS

```
1 <?php include 'includes/header.php'; ?>
2 <h1>Basket</h1>
```

PHP

**Warning:** include(header.php): Failed to open stream: No such file or directory in **/Users/Jon/sites/sample-error.php** on line **1**

**Warning:** Failed opening 'header.php' for inclusion  
(include\_path='.:./Applications/MAMP/bin/php/php8.0/lib/php') in  
**/Users/Jon/sites/sample-error.php** on line **1**

# ERROR-HANDLING FUNCTIONS

```
set_error_handler('name');  
    └── FUNCTION NAME
```

## SHUTDOWN FUNCTION FOR FATAL ERRORS

```
register_shutdown_function('name');  
    └── FUNCTION NAME
```

# NON-FATAL ERROR HANDLER

```
set_error_handler('handle_error');

function handle_error($level, $message, $file = '', $line = 0)
{
    $message = $level . ' ' . $message . ' in '
        . $file . ' on line ' . $line;
    error_log($message);
    http_response_code(500);
    require_once 'includes/header.php';
    echo "Sorry, a problem occurred. Please try later.";
    require_once 'includes/footer.php';
    exit;
}
```

PHP

# DEBUGGING

- Write notes to the screen
- Comment out sections of code
- Use `var_dump()` to check values in variables
- Write tests cases for functions and methods
- Indent code to show missing closing brackets
- Step through code and use breakpoints in IDE

# EXCEPTIONS

**Exceptions** are objects that can be created by the PHP interpreter or the programmer.

They let you run a block of code if something exceptional stops the page running as intended.

Programmers say exceptions are **thrown** and the code to deal with the exception **catches** it.

They should only be used in exceptional situations e.g.: a database being down.

# EXCEPTION OBJECT METHODS

METHOD	DESCRIPTION
<code>getMessage()</code>	Exception message
<code>getCode()</code>	Exception code
<code>getFile()</code>	Name of file exception created in
<code>getLine()</code>	Line exception was created on
<code>getTraceAsString()</code>	Stack trace as a string
<code>getTrace()</code>	Stack trace as an array

# CREATING A CUSTOM EXCEPTION

```
class CustomExceptionName extends Exception {};  
throw CustomExceptionName ($message[, $code]);
```

EXCEPTION CLASS NAME

MESSAGE

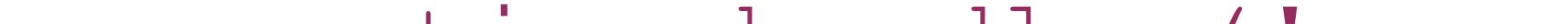
CODE

# HANDLING EXCEPTIONS USING TRY... CATCH

```
try {  
    // Code that might throw an exception  
} catch (ExceptionClassName $e) {  
    // Handle this exception  
} finally () {  
    // Runs whether or not exception occurred  
}
```

# DEFAULT EXCEPTION HANDLING FUNCTION

```
set_exception_handler('name');
```



FUNCTION NAME

```
function handle_exception($e)
{
    http_response_code(500);
    echo 'Sorry, an error occurred please try later.';
    exit;
}
```

# PHP

# WEB SERVER ERRORS

If a web server cannot find a requested file, or an error prevents it from processing a request, the web server sends an error code and error page back to the browser.

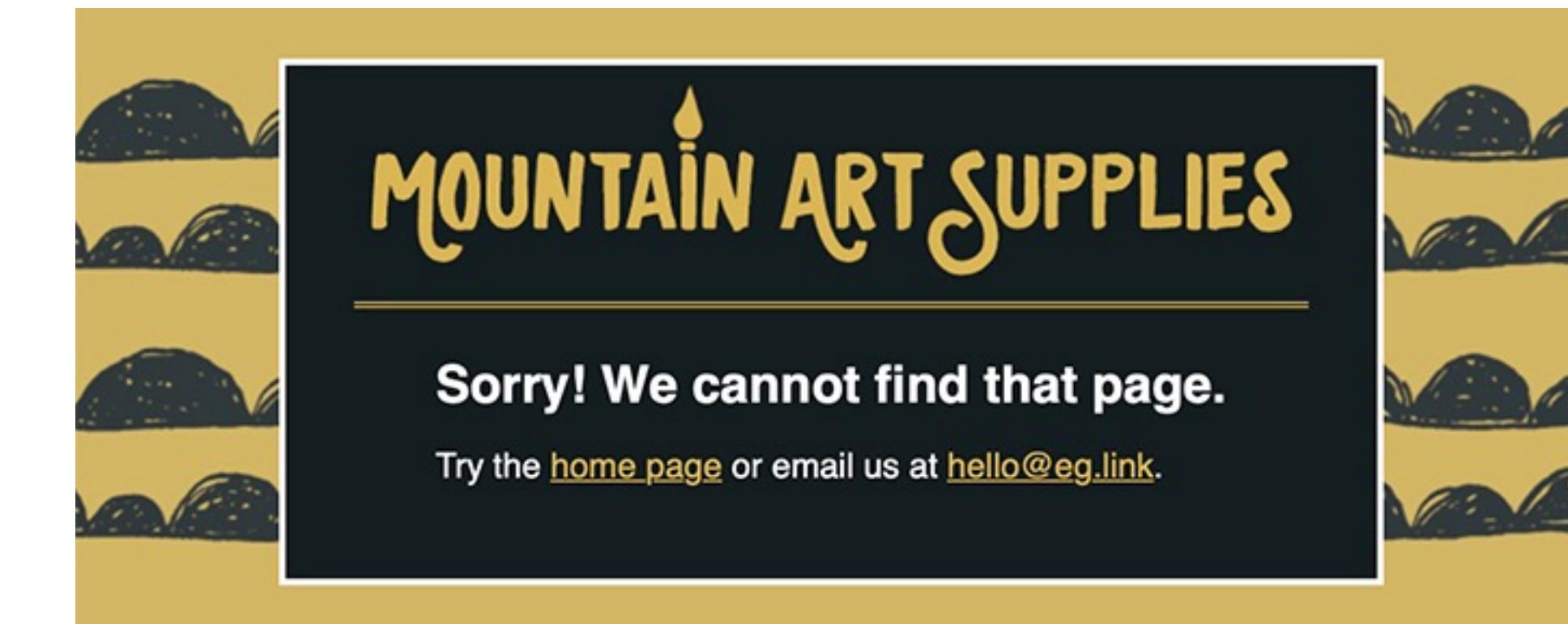
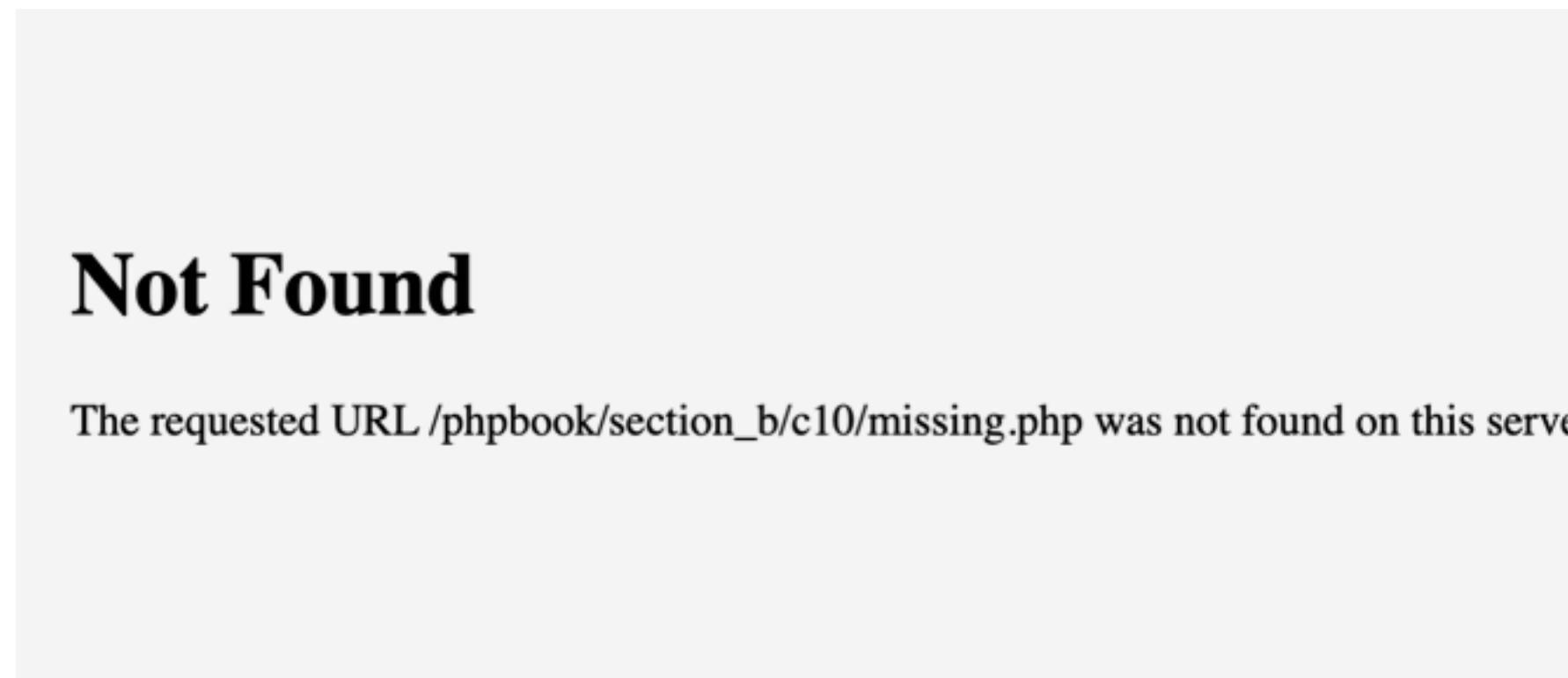
# SETTING DEFAULT ERROR PAGES IN .HTACCESS

```
ErrorDocument 404 /errors/page-not-found.php  
ErrorDocument 500 /errors/error.php
```

DIRECTIVE

STATUS CODE

PAGE TO SHOW



# SECTION C

# INTRODUCTION

Database-driven websites store their content, plus data about their members in a database.

**MySQL** is software to create and manage databases.

**PHP Data Objects (PDO)** are built-in classes that PHP uses to work with the database.

**Structured Query Language (SQL)** is used to ask the database for information, and update the data that it stores.

**PHPMyAdmin** is a web-based tool that provides a graphical user interface for MySQL.

# SAMPLE WEBSITE: HOME PAGE

**CREATIVE FOLK**

Print / Digital / Illustration / Photography 



**Travel Guide**  
Book design for series of travel guides  
POSTED IN PRINT BY IVY STONE



**Golden Brown**  
Photograph for interior design book  
POSTED IN PHOTOGRAPHY BY EMIKO ITO



**Polite Society Posters**  
Poster designs for a fashion label  
POSTED IN PRINT BY IVY STONE



**Stargazer**  
Illustrations for music festival  
POSTED IN ILLUSTRATION BY EMIKO ITO



**Chimney Business Cards**  
Stationery design for publishing company  
POSTED IN PRINT BY LUKE WOOD



**Abandoned Industry**  
Photograph for magazine feature  
POSTED IN PHOTOGRAPHY BY LUKE WOOD

© Creative Folk 2021

# SAMPLE WEBSITE: ARTICLE PAGE

**CREATIVE FOLK**

[Print](#) / [Digital](#) / [Illustration](#) / [Photography](#) 



**Systemic Brochure**

January 26, 2021

This brochure design is part of a suite of advertising materials that promote the Systemic science festival. These materials feature the complex visual identity that is an abstract grid of pathways representing choice and decision-making in designing complex systems.

POSTED IN [PRINT](#) BY LUKE WOOD

© Creative Folk 2021

# SAMPLE WEBSITE: CATEGORY PAGE

**CREATIVE A FOLK**

Print / Digital / Illustration / Photography 

**PRINT**  
INSPIRING GRAPHIC DESIGN



**Travel Guide**  
Book design for series of travel guides  
POSTED IN PRINT BY IVY STONE



**Polite Society Posters**  
Poster designs for a fashion label  
POSTED IN PRINT BY IVY STONE



**Chimney Business Cards**  
Stationery design for publishing company  
POSTED IN PRINT BY LUKE WOOD



**Milk Beach Album Cover**



**The Ice Palace**



**Systemic Brochure**

# SAMPLE WEBSITE: MEMBER PAGE

**CREATIVE A FOLK**

Print / Digital / Illustration / Photography 

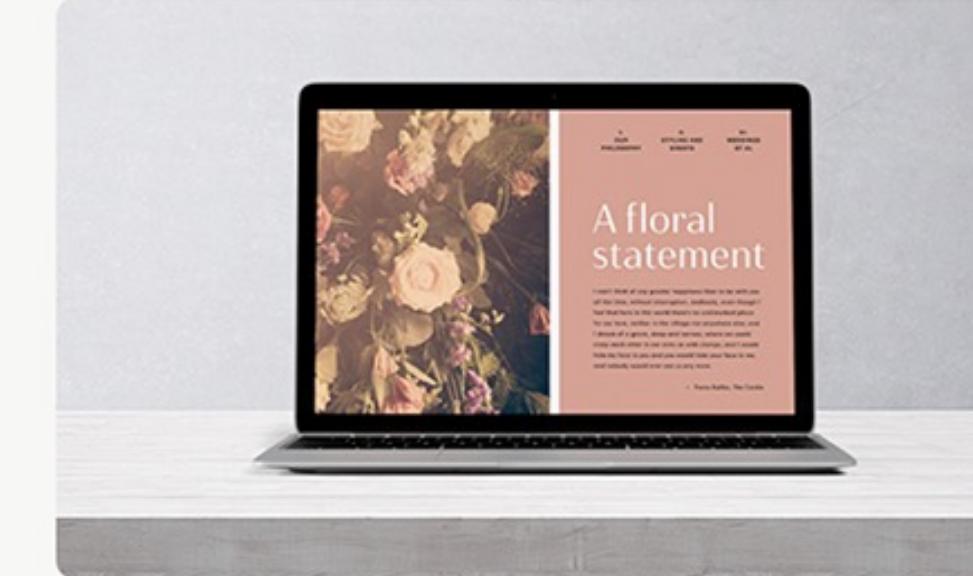
**IVY STONE**

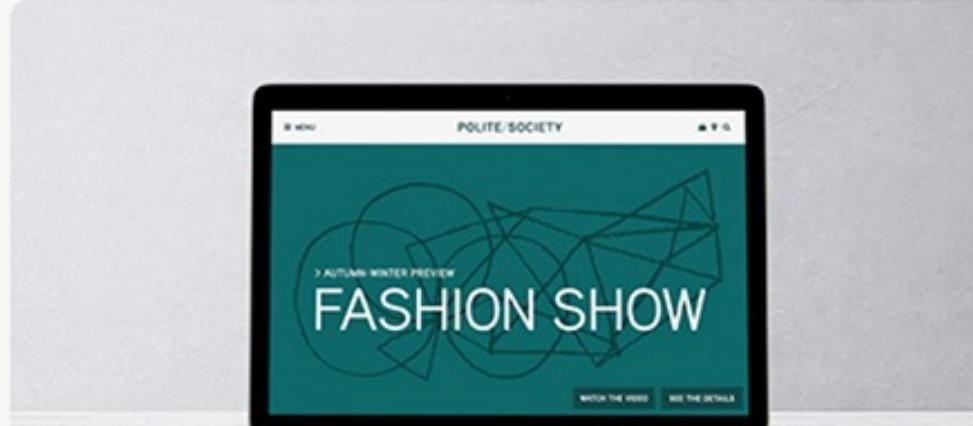
MEMBER SINCE: JANUARY 26, 2021



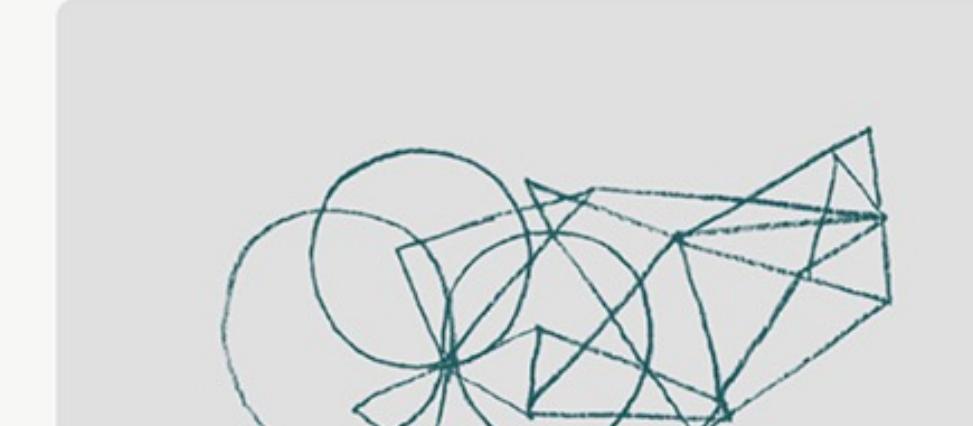
Travel Guide  
Book design for series of travel guides  
POSTED IN PRINT BY IVY STONE

Polite Society Posters  
Poster designs for a fashion label  
POSTED IN PRINT BY IVY STONE

Floral Website  
Website for florist  
POSTED IN DIGITAL BY IVY STONE

FASHION SHOW  
Autumn Winter Preview  
ENTER THE WORLD SEE THE BEACH

MILK BEACH  
Vid L-rrl.  
Featuring works by  
Dusty Glan, Jelena  
Eos, Steven Reich, Max  
Richter & many more



# HOW DATABASES STORE DATA

article								
	id	title	summary	created	category_id	member_id	image_id	published
1	Systemic	<p>This		2021-01-01	1	2	1	1
2	Poster	<p>These		2021-01-02	1	1	2	1
3	Architect	<p>This		2021-01-02	4	2	3	1

member					
	id	forename	surname	password	joined
1	Ivy		Stone	\$2y\$10\$MAd...	2021-01-01
2	Luke		Wood	\$2y\$10\$NN5...	2021-01-01
3	Emiko		Ito	\$2y\$10\$/RpR...	2021-01-01

# HOW DATABASES STORE DATA

image		
id	file	alt
1	systemic-brochure.jpg	Brochure for Systemic Science Festival
2	forecast.jpg	Illustration of a handbag
3	swimming-pool.jpg	Photography of a swimming pool

category			
id	name	description	navigation
1	Print	Inspiring graphic design	1
2	Digital	Powerful pixels	1
3	Illustration	Hand-drawn visual storytelling	1

# MYSQL DATA TYPES

DATA TYPE	DESCRIPTION
int	Whole number
tinyint	Whole number up to 255 (used for booleans)
varchar	Up to 65,535 alphanumeric characters
text	Up to 65,535 alphanumeric characters
timestamp	Date and time

With the exception of the text data type, you must specify the number of:

- Bytes a column of text will use
- Digits a column containing numbers will use

**Relationships** between data in tables saves data being duplicated in the database.

They use **primary keys** and **foreign keys**.

# RELATIONSHIPS BETWEEN TABLES

article								
	id	title	summary	created	category_id	member_id	image_id	published
1	Systemic	<p>This		2021-01-01	1	2	1	1
2	Poster	<p>These		2021-01-02	1	1	2	1
3	Architect	<p>This		2021-01-02	4	2	3	1

FOREIGN KEY

member						
	id	forename	surname	password	joined	picture
1	Ivy		Stone	\$2y\$10\$MAd...	2021-01-01	ivy.jpg
2	Luke		Wood	\$2y\$10\$NN5...	2021-01-01	NULL
3	Emiko		Ito	\$2y\$10\$/RpR...	2021-01-01	emi.jpg

PRIMARY KEY

PHPMYADMIN

MySQL does not have a graphical user interface.

PHPMyAdmin is a web-based interface that allows you to work with MySQL databases.

# PHPMYADMIN

The screenshot shows the phpMyAdmin 4.9.7 interface running on a Mac OS X system. The title bar indicates the window is titled "localhost". The main menu bar includes "File", "Edit", "View", "Select", "Search", "Help", and "About". The top navigation bar has tabs for "Databases", "SQL", "Status", "User accounts", "Export", "Import", "Settings", "Replication", "Variables", and "More".

**General settings:** Shows the "Server connection collation" set to "utf8mb4\_unicode\_ci".

**Appearance settings:** Shows the "Language" set to "English", "Theme" set to "pmahomme", and "Font size" set to "82%". There is also a link to "More settings".

**Database server:** Displays the following information:

- Server: Localhost via UNIX socket
- Server type: MySQL
- Server connection: SSL is not being used
- Server version: 5.7.32 - MySQL Community Server (GPL)
- Protocol version: 10
- User: root@localhost
- Server charset: UTF-8 Unicode (utf8)

**Web server:** Displays the following information:

- Apache/2.4.46 (Unix) OpenSSL/1.0.2u PHP/7.4.12 mod\_wsgi/3.5 Python/2.7.13 mod\_fastcgi/mod\_fastcgi-SNAP-0910052141 mod\_perl/2.0.11 Perl/v5.30.1
- Database client version: libmysql - mysqlnd 7.4.12
- PHP extension: mysqli curl mbstring
- PHP version: 7.4.12

**phpMyAdmin:** Links to Version information, Documentation, Official Homepage, Contribute, Get support, List of changes, and License.

At the bottom left is a "Console" button.

# PHPMYADMIN: DATABASES & TABLES

The screenshot shows the phpMyAdmin interface running on a Mac OS X system. The title bar indicates the server is 'localhost'. The left sidebar lists databases: 'information\_schema', 'mysql', 'performance\_schema', 'phpbook-1', 'New', 'article', 'category', 'image', 'member', and 'sys'. The 'phpbook-1' database is expanded, showing its tables: 'New', 'article', 'category', 'image', and 'member'. A red box highlights this section. The main content area has tabs for 'Databases', 'SQL', 'Status', 'User accounts', 'Export', 'Import', 'Settings', 'Replication', 'Variables', and 'More'. The 'General settings' panel shows the 'Server connection collation' as 'utf8mb4\_unicode\_ci'. The 'Database server' panel provides detailed information about the MySQL server. The 'Web server' panel lists the Apache and PHP configurations. The 'phpMyAdmin' panel links to version information, documentation, and other resources. A 'Console' button is at the bottom.

localhost

phpMyAdmin

Databases SQL Status User accounts Export Import Settings Replication Variables More

General settings

Server connection collation: utf8mb4\_unicode\_ci

Database server

- Server: Localhost via UNIX socket
- Server type: MySQL
- Server connection: SSL is not being used
- Server version: 5.7.32 - MySQL Community Server (GPL)
- Protocol version: 10
- User: root@localhost
- Server charset: UTF-8 Unicode (utf8)

Appearance settings

Language: English

Theme: pmahomme

Font size: 82%

More settings

Web server

- Apache/2.4.46 (Unix) OpenSSL/1.0.2u PHP/7.4.12 mod\_wsgi/3.5 Python/2.7.13 mod\_fastcgi/mod\_fastcgi-SNAP-0910052141 mod\_perl/2.0.11 Perl/v5.30.1
- Database client version: libmysql - mysqlnd 7.4.12
- PHP extension: mysqli curl mbstring
- PHP version: 7.4.12

phpMyAdmin

- Version information: 4.9.7
- Documentation
- Official Homepage
- Contribute
- Get support
- List of changes
- License

Console

# PHPMYADMIN: TABS

The screenshot shows the phpMyAdmin interface on a Mac OS X desktop. The title bar indicates the server is 'localhost'. The top navigation bar has tabs: Databases, SQL, Status, User accounts, Export, Import, Settings (which is highlighted with a red box), Replication, Variables, and More.

**General settings:** Server connection collation is set to utf8mb4\_unicode\_ci.

**Appearance settings:** Language is English, Theme is pmahomme, and Font size is 82%.

**Database server:**

- Server: Localhost via UNIX socket
- Server type: MySQL
- Server connection: SSL is not being used
- Server version: 5.7.32 - MySQL Community Server (GPL)
- Protocol version: 10
- User: root@localhost
- Server charset: UTF-8 Unicode (utf8)

**Web server:**

- Apache/2.4.46 (Unix) OpenSSL/1.0.2u PHP/7.4.12 mod\_wsgi/3.5 Python/2.7.13 mod\_fastcgi/mod\_fastcgi-SNAP-0910052141 mod\_perl/2.0.11 Perl/v5.30.1
- Database client version: libmysql - mysqlnd 7.4.12
- PHP extension: mysqli curl mbstring
- PHP version: 7.4.12

**phpMyAdmin:**

- Version information: 4.9.7
- Documentation
- Official Homepage
- Contribute
- Get support
- List of changes
- License

At the bottom left is a 'Console' button.

# PHPMYADMIN: MAIN WINDOW

The screenshot shows the main interface of the phpMyAdmin application. A red border highlights the central content area.

**General settings:**

- Server connection collation: utf8mb4\_unicode\_ci

**Appearance settings:**

- Language: English
- Theme: pmahomme
- Font size: 82%

**Database server:**

- Server: Localhost via UNIX socket
- Server type: MySQL
- Server connection: SSL is not being used
- Server version: 5.7.32 - MySQL Community Server (GPL)
- Protocol version: 10
- User: root@localhost
- Server charset: UTF-8 Unicode (utf8)

**Web server:**

- Apache/2.4.46 (Unix) OpenSSL/1.0.2u PHP/7.4.12 mod\_wsgi/3.5 Python/2.7.13 mod\_fastcgi/mod\_fastcgi-SNAP-0910052141 mod\_perl/2.0.11 Perl/v5.30.1
- Database client version: libmysql - mysqlnd 7.4.12
- PHP extension: mysqli curl mbstring
- PHP version: 7.4.12

**phpMyAdmin:**

- Version information: 4.9.7
- [Documentation](#)
- [Official Homepage](#)
- [Contribute](#)
- [Get support](#)
- [List of changes](#)
- [License](#)

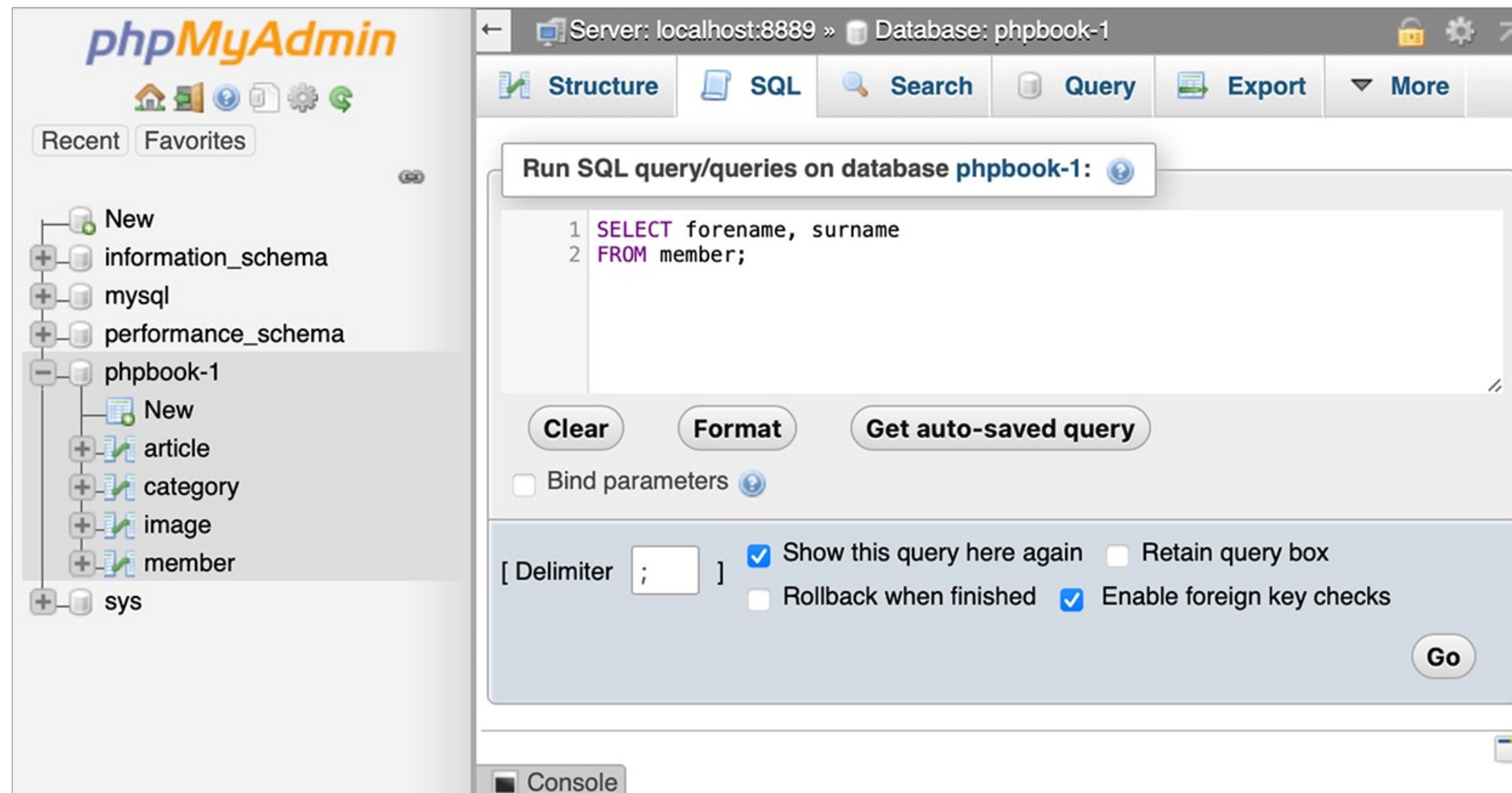
At the bottom left, there is a "Console" button.

# CHAPTER 11

# STRUCTURED QUERY LANGUAGE

**Structured Query Language (SQL)** is a language that is used to communicate with databases.

# RUNNING SQL STATEMENTS IN PHPMYADMIN



# VIEWING RESULT SET IN PHPMYADMIN

The screenshot shows the phpMyAdmin interface with the following details:

- Server:** localhost:8889
- Database:** phpbook-1
- Table:** member

The interface includes the following sections:

- Browse:** Shows the result set with 3 total rows.
- Structure:** Shows the table structure.
- SQL:** Shows the executed SQL query.
- Search:** Search bar for the table.
- Insert:** Insert form for adding new rows.
- More:** Additional options.

The result set displays the following data:

	forename	surname
<input type="checkbox"/>	Ivy	Stone
<input type="checkbox"/>	Luke	Wood
<input type="checkbox"/>	Emiko	Ito

Below the table, there are buttons for Edit, Copy, Delete, Check all, and Export.

# GETTING DATA FROM A DATABASE

COLUMNS TO SELECT → `SELECT column1, column2`

TABLE COLUMNS ARE IN → `FROM table;`

`SELECT forename, surname  
FROM member;`

COLUMN 1                    COLUMN 2

forename                    surname

TABLE

SQL

result set	
forename	surname
Ivy	Stone
Luke	Wood
Emiko	Ito

# RETURNING SPECIFIC DATA

COLUMNS TO SELECT → `SELECT column1, column2`

TABLE COLUMNS ARE IN → `FROM table`

ROWS TO ADD TO RESULT SET → `WHERE column = value;`

|  
OPERATOR  
(CAN ALSO USE: AND, OR, NOT)

```
SELECT forename, surname  
      FROM member  
 WHERE forename = 'Ivy';
```

SQL

result set

forename	surname
Ivy	Stone

# SEARCHING FOR RESULTS USING LIKE & WILDCARDS

COLUMNS TO SELECT → `SELECT column1, column2`

TABLE COLUMNS ARE IN →      FROM *table*

ROWS TO ADD TO RESULT SET → WHERE column LIKE '%value%';

## LIKE OPERATOR

# WILDCARD SYMBOLS

% indicates zero or more characters  
\_ indicates an individual character

```
SELECT forename, surname  
FROM member  
WHERE forename LIKE 'I%';
```

SQI

# result set

---

forename	surname
Ivy	Stone

---

# CONTROLLING ORDER OF ROWS

COLUMNS TO SELECT → **SELECT** *column1, column2*

TABLE COLUMNS ARE IN → **FROM** *table*

CONTROL ORDER OF RESULTS → **ORDER BY** *column1 ASC, column2 DESC;*

ORDER BY CLAUSE                    COLUMN                    DIRECTION                    COLUMN                    DIRECTION

```
SELECT email  
      FROM member  
 ORDER BY email DESC;
```

SQL

result set
email
luke@eg.link
ivy@eg.link
emi@eg.link

# COUNTING & GROUPING RESULTS

```
SELECT COUNT(*)  
  FROM table  
 WHERE column LIKE '%value%';
```

```
SELECT column COUNT(*)  
  FROM table  
 GROUP BY column;
```

```
SELECT member_id, COUNT(*)  
  FROM article  
 GROUP BY member_id;
```

SQL

result set	
member_id	COUNT(*)
1	10
2	8
3	6

# LIMITING & SKIPPING RESULTS

```
SELECT columns  
      FROM table  
      LIMIT 3 OFFSET 6;
```

MAXIMUM RESULTS    RESULTS TO SKIP

```
SELECT title  
      FROM article  
      LIMIT 3 OFFSET 9;
```

SQL

**result set**

title

Polite Society Mural

Stargazer Website and App

The Ice Palace

# RELATIONSHIPS BETWEEN TABLES

article								
	id	title	summary	created	category_id	member_id	image_id	published
1	Systemic	<p>This		2021-01-01	1	2	1	1
2	Poster	<p>These		2021-01-02	1	1	2	1
3	Architect	<p>This		2021-01-02	4	2	3	1

FOREIGN KEY

category				
	id	name	description	navigation
→	1	Print	Inspiring graphic design	1
PRIMARY KEY	2	Digital	Powerful pixels	1
	3	Illustration	Hand-drawn visual storytelling	1

# USING JOINS TO GET DATA FROM TWO TABLES

```
SELECT article.title, article.summary, category.name  
FROM article  
JOIN category ON article.category_id = category.id  
WHERE category.id = 1;
```

**FOREIGN KEY**

**PRIMARY KEY**

result set		
title	summary	name
Milk Beach Website	Website for music series	Digital
Wellness App	App for health facility	Digital
Stargazer Website and App	Website and app for music festival	Digital

# HANDLING MISSING DATA

article									
	id	title	summary	content	created	category_id	member_id	image_id	published
4	Walking...	Artwork...	The brie...	2021...	3	3	4	1	
5	Sisters...	Editori...	The arti...	2021...	3	3	NULL	1	
6	Micro-D...	Photogr...	This pho...	2021...	4	1	6	1	

image		
	id	file
	4	birds.jpg
	6	micro-dunes.jpg

alt

Collage of two birds

Photograph of tiny sand dunes

# INNER JOIN

Only add to result set if all the data is available

```
SELECT article.id, article.title,  
image.file  
FROM article  
JOIN image ON article.image_id =  
image.id.
```

result set		
	id title	name
1	Systemic Brochure	systemic-brochure.jpg
2	Forecast	forecast.jpg
3	Swimming Pool	swimming-pool.jpg
4	Walking Birds	birds.jpg
6	Micro Dunes	micro-dunes.jpg

# LEFT JOIN

Add all requested data from left table, and use NULL for missing values in right table

```
SELECT article.id, article.title, image.file  
      FROM article  
      LEFT JOIN image ON article.image_id =  
                    image.id;
```

result set		
	id title	name
1	Systemic Brochure	systemic-brochure.jpg
2	Forecast	forecast.jpg
3	Swimming Pool	swimming-pool.jpg
4	Walking Birds	birds.jpg
5	Sisters	NULL

# JOIN FROM MULTIPLE TABLES

```
SELECT article.id, article.title, category.name,  
       image.file, image.alt  
  
FROM article  
JOIN category    ON article.category_id = category.id  
LEFT JOIN image   ON article.image_id      = image.id  
  
WHERE article.category_id = 3  
      AND article.published    = 1  
ORDER BY article.id DESC;
```

SQL

# TABLE ALIASES

Table aliases make queries that use joins easier to read

```
SELECT a.id, a.title, c.name,  
       i.file, i.alt  
  
  FROM article      AS a  
 JOIN category    AS c ON article.category_id = category.id  
LEFT JOIN image   AS i ON article.image_id      = image.id  
  
 WHERE article.category_id = 3  
   AND article.published  = 1  
 ORDER BY article.id DESC;
```

SQL

# COLUMN ALIASES

Column aliases specify column names in result set

```
SELECT forename AS firstname, surname AS lastname  
FROM member;
```

result set	
firstname	lastname
Ivy	Stone
Luke	Wood
Emiko	Ito

# COMBINING COLUMNS

```
SELECT CONCAT(forename, ' ', surname) AS author  
FROM member;
```

result set
author
Ivy Stone
Luke Wood
Emiko Ito

# ALTERNATIVES TO NULL

```
SELECT COALESCE(picture, forename, 'friend') AS profile  
FROM member;
```

result set
profile
ivy.jpg
Luke
emi.jpg

# ARTICLE QUERY

```
SELECT a.title, a.summary, a.content, a.created,  
       a.category_id, a.member_id,  
       c.name      AS category,  
       CONCAT(m.forename, ' ', m.surname) AS author,  
       i.file      AS image_file,  
       i.alt       AS image_alt  
  
FROM article      AS a  
JOIN category     AS c ON a.category_id = c.id  
JOIN member       AS m ON a.member_id    = m.id  
LEFT JOIN image   AS i ON a.image_id    = i.id  
  
WHERE a.id        = 22  
  AND a.published = 1;
```

SQL

# ARTICLE LIST

```
SELECT a.id, a.title, a.summary, a.category_id, a.member_id,  
      c.name AS category,  
      CONCAT(m.forename, ' ', m.surname) AS author,  
      i.file AS image_file,  
      i.alt AS image_alt  
  
FROM article      AS a  
JOIN category    AS c ON a.category_id = c.id  
JOIN member       AS m ON a.member_id    = m.id  
LEFT JOIN image   AS i ON a.image_id     = i.id  
  
WHERE a.category_id = 1  
      AND a.published    = 1  
ORDER BY a.id DESC;
```

SQL

# UPDATING THE DATABASE

# ADDING DATA TO THE DATABASE

WHERE THE DATA GOES → `INSERT INTO table (column2, column2)`  
VALUES TO ADD → `VALUES ('value1', 'value2');`

COLUMN NAMES  
`INSERT INTO category (name, description, navigation)  
VALUES ('News', 'Latest news from Creative Folk', 0);`

NEW DATA TO ADD

# UPDATING DATA IN THE DATABASE

WHERE THE DATA GOES → UPDATE *table* (*column1*, *column2*)  
VALUES TO ADD → SET *column1* = 'value1'  
ROW(S) TO UPDATE → WHERE *column2* = 'value2';

```
UPDATE category
    SET name = 'Blog', navigation = 1
WHERE id = 5;
```

# DELETING DATA FROM THE DATABASE

TABLE TO REMOVE DATA FROM → `DELETE FROM table`

ROW(S) TO REMOVE → `WHERE column = 'value';`

```
DELETE FROM category  
WHERE id = 5;
```

# CONSTRAINTS

# UNIQUENESS CONSTRAINT

Ensure each value in specified column is unique to prevent duplicate entries

The screenshot shows the phpMyAdmin interface for managing a MySQL database. The left sidebar lists databases: information\_schema, mysql, performance\_schema, phpbook-1, article, category, image, member, and sys. The 'category' database is selected. The main area displays the 'category' table structure. The table has four columns: id, name, description, and navigation. The 'name' column is currently selected. The 'Attributes' row for 'name' shows 'None' under 'Default' and 'None' under 'Comments'. Under the 'Extra' column, there is a dropdown menu open, showing options: Change, Drop, More, Primary, Unique, Index, Spatial, Fulltext, and Distinct values. The 'Unique' option is highlighted. The top navigation bar includes tabs for Browse, Structure, SQL, Search, Insert, Export, Import, Privileges, Operations, and Triggers.

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	<b>id</b>	int(11)			No	None		AUTO_INCREMENT	Change  Drop  More
2	<b>name</b>	varchar(24)	utf8mb4_unicode_ci		No	None			Change  Drop  More
3	<b>description</b>	varchar(254)	utf8mb4_unicode_ci		No	None			Change  Primary
4	<b>navigation</b>	tinyint(1)			No	None			Change  Unique

# FOREIGN KEY CONSTRAINT

Checks that a foreign key is a valid primary key in corresponding table

The screenshot shows the phpMyAdmin interface for managing a database named 'phpbook-1'. The current table is 'article'. The 'Structure' tab is selected. In the left sidebar, under the 'phpbook-1' schema, the 'article' table is selected. On the right, the 'Foreign key constraints' section is displayed. It shows a constraint named 'category\_exists' that links the 'category\_id' column in the 'article' table to the 'id' column in the 'category' table. The 'ON DELETE' and 'ON UPDATE' actions for this constraint are both set to 'RESTRICT'. There are also sections for adding new constraints and previewing the generated SQL code.

Column	Foreign key constraint (INNODB)	Database	Table	Column
category_id	category_exists	phpbook-1	category	id

Actions: Drop

Constraint properties:

- ON DELETE RESTRICT
- ON UPDATE RESTRICT

+ Add constraint

Preview SQL Save

# CHAPTER 12

# GET & SHOW DATA

# FROM A DATABASE

PHP pages use SQL queries to get data from the database.

Built-in classes called **PHP Data Objects (PDO)** create objects that help your PHP code work with a database.

First, the PDO class create an object that manages the connection to the database.

Then, the PDOStatement class creates an object that represents the SQL statement and any result set that it generates.

# DATA SOURCE NAME

```
$type      = 'mysql';           // Type of database software
$server    = 'localhost';        // Host name
$db        = 'phpbook-1';        // Database name
$port      = '8889';             // Use port 3006 for XAMPP
$charset   = 'utf8mb4';          // UTF-8 encoding using 4 bytes

// DO NOT ALTER NEXT LINE
$dsn       = "$type:host=$server;dbname=$db;
              port=$port;charset=$charset";
```

PHP

# DATABASE USER ACCOUNT

```
$username = 'enter-your-username';
$password = 'enter-your-password';
$options = [
    PDO::ATTR_ERRMODE            => PDO::ERRMODE_EXCEPTION,
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
    PDO::ATTR_EMULATE_PREPARES   => false,
];
```

PHP

# CREATE PDO OBJECT

Can create DNS, store user account and create PDO object all in an include file

```
$pdo = new PDO($dsn, $username, $password, $options);
```

PHP

One PHP page will often be used to display multiple pages of the site.

# ONE PHP FILE CAN CREATE MANY WEB PAGES

The screenshot shows a website for "Creative Folk". At the top, there's a navigation bar with the logo "CREATIVE FOLK" and links for "Print / Digital / Illustration / Photography" and a search icon. Below the header, there's a large image of an open book showing a travel guide about Nijo Castle in Kyoto. The book has sections for "ADDRESS", "OPENING HOURS", and a detailed description of the castle. To the right of the image, the title "Travel Guide" is displayed, followed by the date "April 25, 2021". The main text discusses the best-selling travel guide series, Featherview, and their need for a refreshed look. It mentions they were after a clean and concise solution that could accommodate both coffee table and backpack designs. Below the text is a "POSTED IN PRINT BY IVY STONE" section.

The screenshot shows a website for "Creative Folk" featuring a "DIGITAL" section. The header includes the "CREATIVE FOLK" logo and a search bar. A large blue box labeled "DIGITAL" with the subtext "POWERFUL PIXELS" is prominently displayed. Below this, there are three examples of digital projects: a "Floral Website" (a laptop screen showing a floral-themed landing page), a "Polite Society Website" (a laptop screen showing a fashion show preview), and a "Chimney Press Website" (three smartphones showing different pages of a book publisher's site). Each project example includes a title, a brief description, and a "POSTED IN DIGITAL BY IVY STONE" or "POSTED IN DIGITAL BY LUKE WOOD" link.

One PHP file gets every article on the site.  
The SQL query changes to get different article  
from database.

Another PHP file represents each category.  
The SQL query changes to request different  
category information from database.

Each row of data that the result set contains will be represented as an array.

Some queries will return one row of data.  
Other queries return multiple rows of data.

# RETURNING ONE ROW OF DATA

```
SELECT forename, surname  
      FROM member  
     WHERE id = 1;
```

SQL

**result set**

forename	surname
Ivy	Stone

```
$member = [  
    'forename' => 'Ivy',  
    'surname'  => 'Stone',  
];
```

# RETURNING MULTIPLE ROWS OF DATA

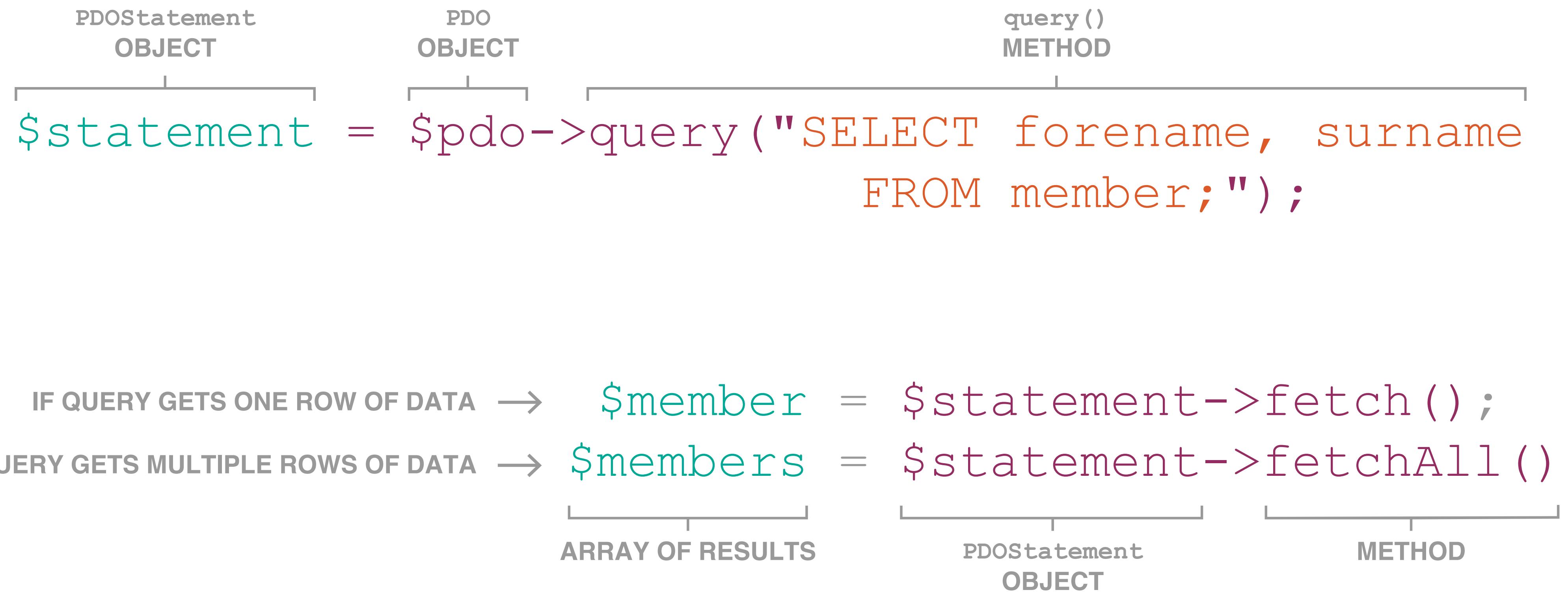
```
SELECT forename, surname  
FROM member
```

SQL

result set	
forename	surname
Ivy	Stone
Luke	Wood
Emiko	Ito

```
$members = [  
    0 => ['forename' => 'Ivy',  
           'surname'  => 'Stone', ],  
    1 => ['forename' => 'Luke',  
           'surname'  => 'Wood', ],  
    2 => ['forename' => 'Emiko',  
           'surname'  => 'Ito', ],  
];
```

# CONTROLLING ORDER OF ROWS



# CHECKING FOR DATA WHEN EXPECTING ONE ROW

```
$sql      = "SELECT forename, surname,  
           FROM member  
           WHERE id = 4;"  
$statement = $pdo->query($sql);  
$member    = $statement->fetch();  
  
if (! $member) {  
    include 'page-not-found.php';  
}  
  
echo $member['forename'];
```

PHP

# LOOPING THROUGH RESULTS FOR MULTIPLE ROWS

PHP

```
<?php
    $sql      = "SELECT forename, surname,
                FROM member"
    $statement = $pdo->query($sql);
    $member    = $statement->fetchAll();
?>

<?php foreach ($members as $member) ?>
<p><?= $member['forename'] ?>
    <?= $member['surname'] ?></p>
<?php } ?>
```

# SQL QUERIES THAT CAN CHANGE

```
$sql = "SELECT forename, surname FROM member WHERE id = :id;"  
$statement = $pdo->prepare($sql);  
$statement->execute(['id' => $id]);
```

PLACEHOLDER  
  └─ :id;  
PLACEHOLDER NAME      VALUE TO USE  
  └─ id

# BINDING VALUES TO A QUERY

```
$sql = "SELECT forename, surname FROM member WHERE id = :id;"  
$statement = $pdo->prepare($sql);  
$statement->bindValue(['id', $id], PDO::PARAM_INT);
```

PLACEHOLDER  
PLACEHOLDER NAME      VALUE TO USE      DATA TYPE

PARAMETER	DESCRIPTION
PDO::PARAM_STRING	String
PDO::PARAM_INT	Integer
PDO::PARAM_BOOL	Boolean

# CHECKING FOR DATA IN A QUERY

```
$id = filter_input(INPUT_GET, 'id', FILTER_VALIDATE_INT);  
if (! $id) {  
    include 'page-not-found.php';  
}  
  
$sql = "SELECT forename, surname FROM member WHERE id = :id;";  
$statement = $pdo->prepare($sql);  
$statement->execute([':id' => $id]);  
$member = $statement->fetch();  
  
if (! $member) {  
    include 'page-not-found.php';  
}
```

PHP

# A FUNCTION TO EXECUTE SQL STATEMENTS

```
function pdo(PDO $pdo, string $sql, array $arguments = null)
{
    if (!$arguments) {
        return $pdo->query($sql);
    }
    $statement = $pdo->prepare($sql);
    $statement->execute($arguments);
    return $statement;
}
```

PHP

# CALLING FUNCTION TO EXECUTE SQL STATEMENTS

No parameters:

```
$sql = "SELECT forename, surname FROM member;";  
$members = pdo($pdo, SQL)->fetchAll();
```

PHP

With parameters:

```
$sql = "SELECT forename, surname FROM member WHERE id = :id;";  
$members = pdo($pdo, SQL, ['id' => $id])->fetch();
```

PHP

FOUR FILES THAT  
POWER THE SITE

# HOME

**CREATIVE FOLK**

Print / Digital / Illustration / Photography 



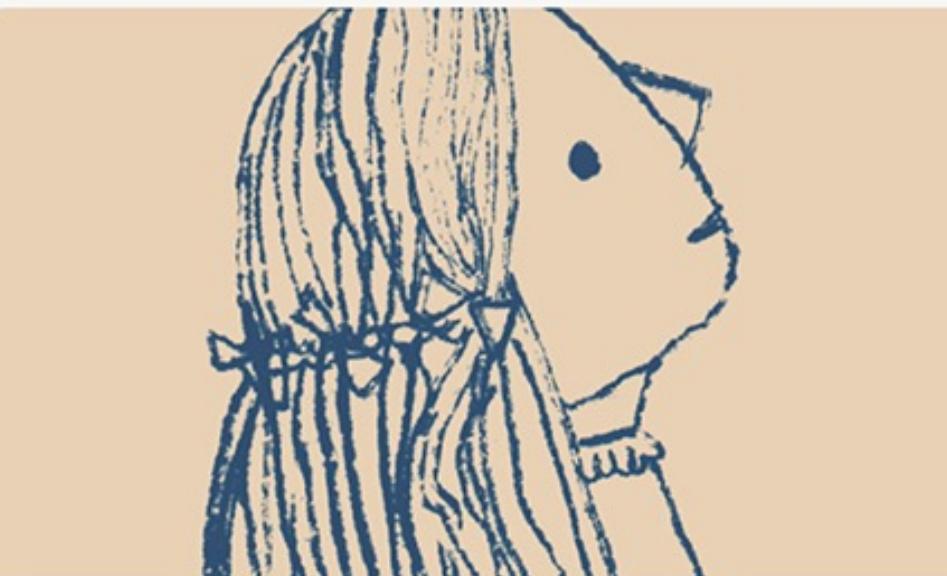
**Travel Guide**  
Book design for series of travel guides  
POSTED IN PRINT BY IVY STONE



**Golden Brown**  
Photograph for interior design book  
POSTED IN PHOTOGRAPHY BY EMIKO ITO



**Polite Society Posters**  
Poster designs for a fashion label  
POSTED IN PRINT BY IVY STONE



**Stargazer**  
Illustrations for music festival  
POSTED IN ILLUSTRATION BY EMIKO ITO



**Chimney Business Cards**  
Stationery design for publishing company  
POSTED IN PRINT BY LUKE WOOD



**Abandoned Industry**  
Photograph for magazine feature  
POSTED IN PHOTOGRAPHY BY LUKE WOOD

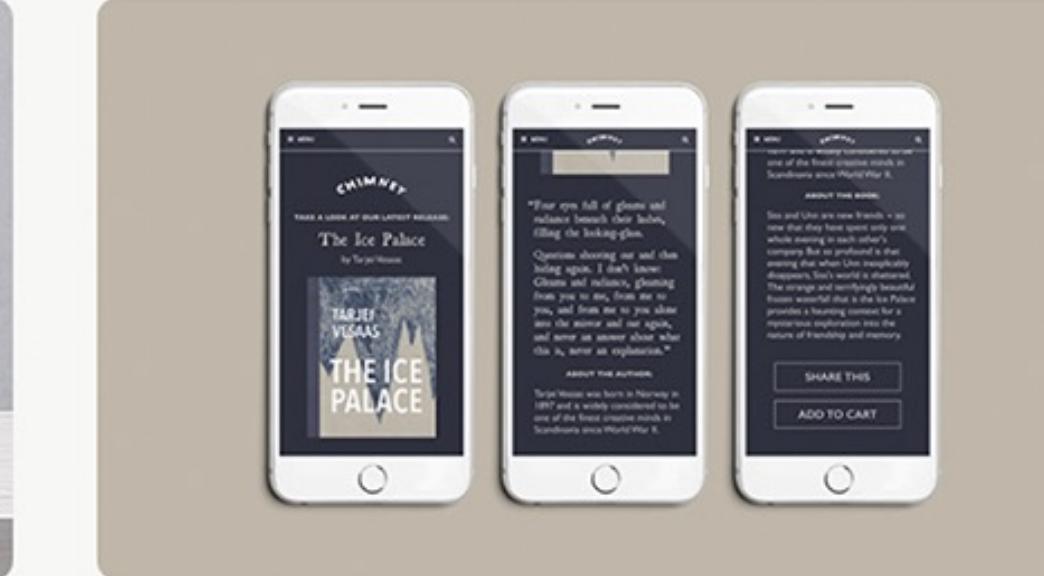
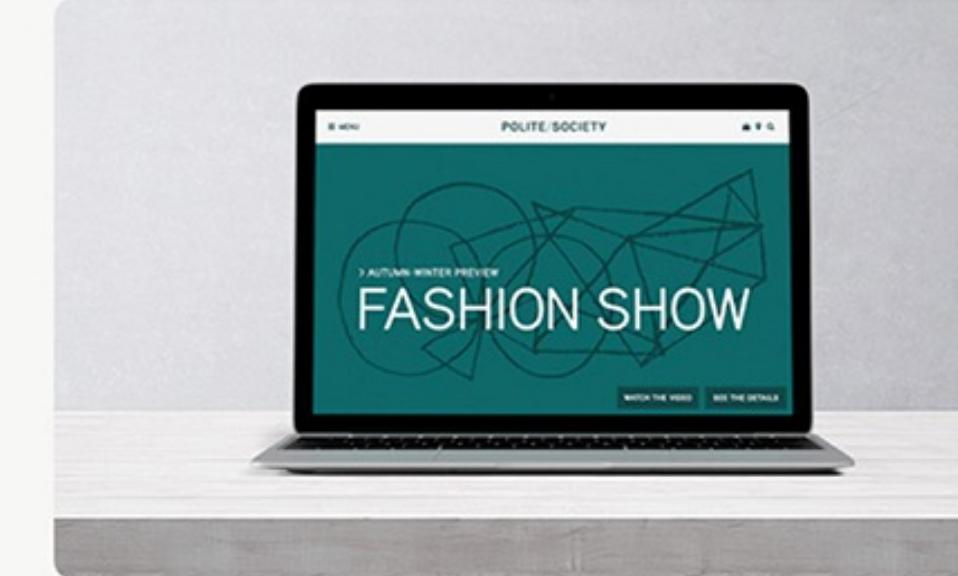
© Creative Folk 2021

# CATEGORY

**CREATIVE A FOLK**

Print / Digital / Illustration / Photography 

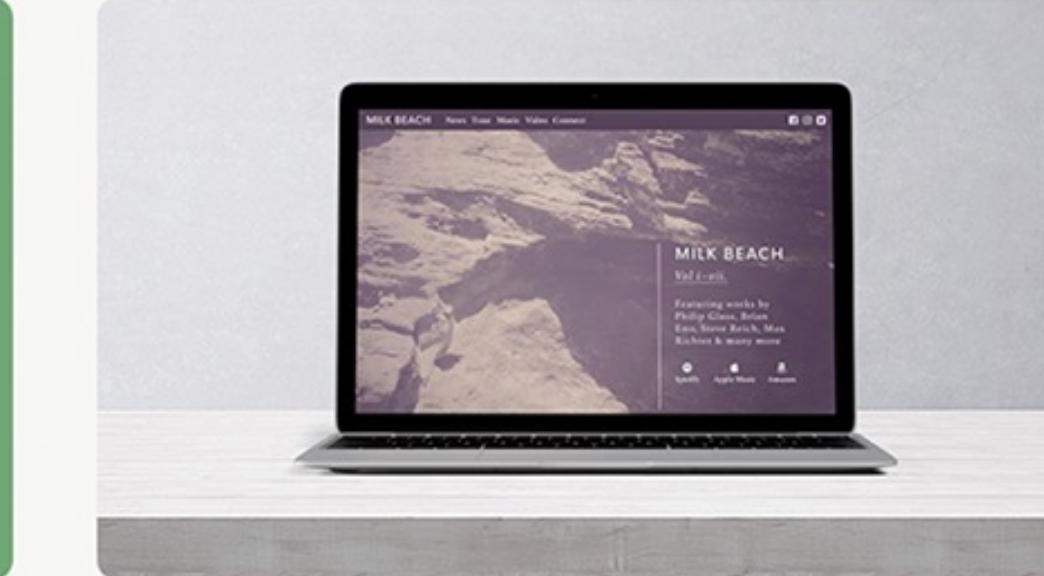
**DIGITAL**  
POWERFUL PIXELS



**Floral Website**  
Website for florist  
POSTED IN DIGITAL BY IVY STONE

**Polite Society Website**  
Website for fashion label  
POSTED IN DIGITAL BY IVY STONE

**Chimney Press Website**  
Website for book publisher  
POSTED IN DIGITAL BY LUKE WOOD



**Stargazer Website and App**

**Wellness App**

**Milk Beach Website**

# MEMBER

**CREATIVE A FOLK**

Print / Digital / Illustration / Photography 

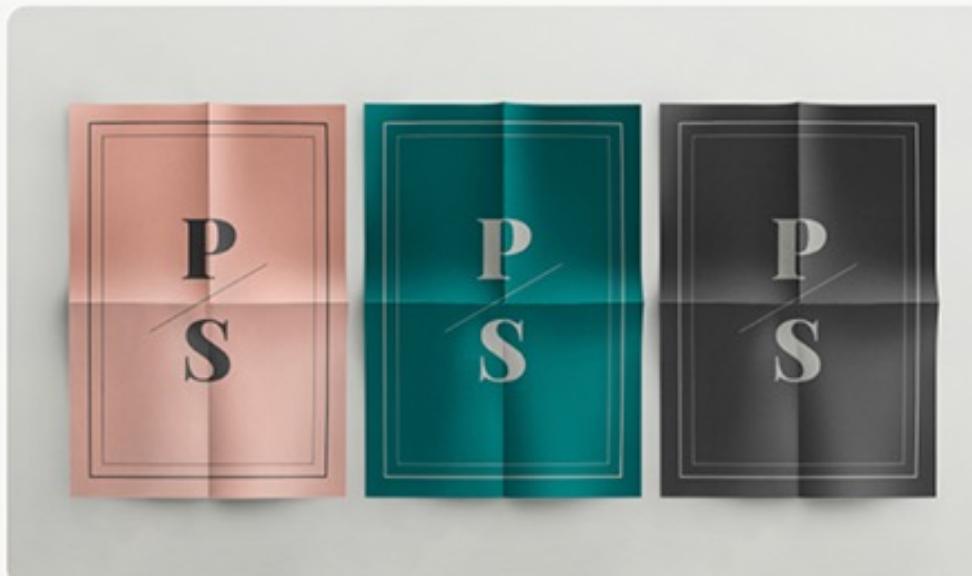
**IVY STONE**

MEMBER SINCE: JANUARY 26, 2021

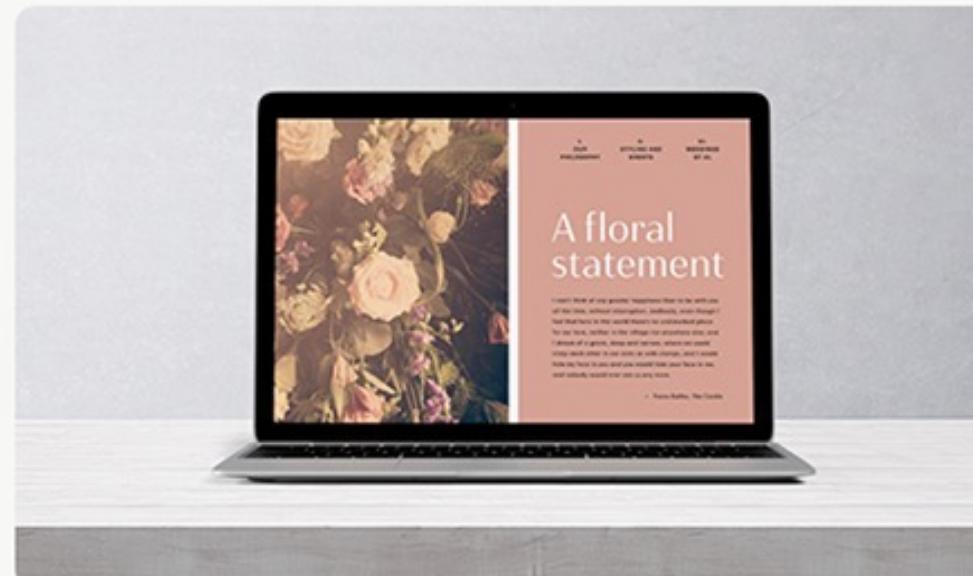


A photograph of an open travel guide book titled "Nijo Castle". The pages show a photograph of a traditional Japanese building and some descriptive text.

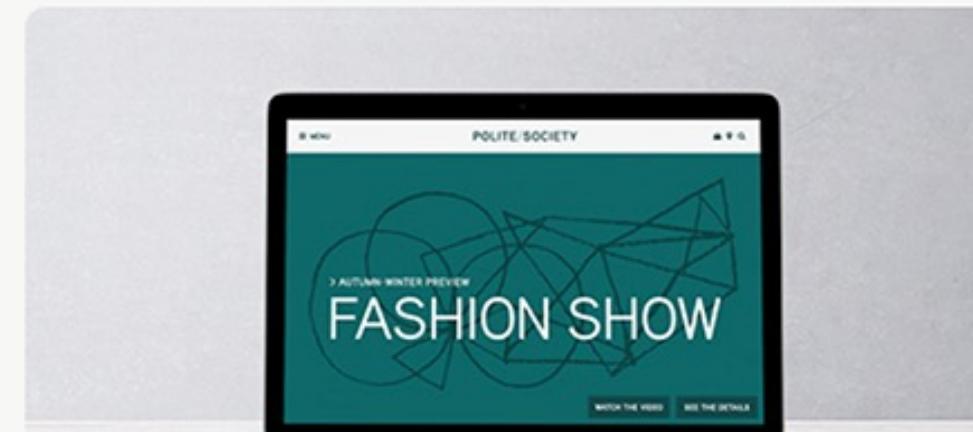
**Travel Guide**  
Book design for series of travel guides  
POSTED IN PRINT BY IVY STONE

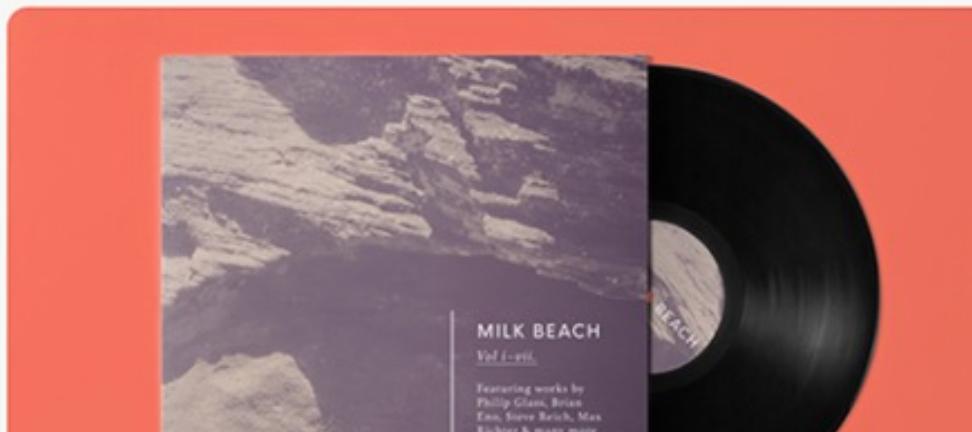
Three posters for "Polite Society" arranged in a grid. Each poster features a large letter "P" and "S" in a stylized font, with a thin diagonal line through them.

**Polite Society Posters**  
Poster designs for a fashion label  
POSTED IN PRINT BY IVY STONE

A photograph of an open laptop displaying a website for a florist. The website has a pink header and features a large image of a floral arrangement.

**Floral Website**  
Website for florist  
POSTED IN DIGITAL BY IVY STONE

A photograph of a tablet displaying a website for a fashion show. The website has a teal header and features the text "FASHION SHOW".

A photograph of a black vinyl record next to a CD cover for "MILK BEACH". The CD cover features a photograph of a rocky beach.

A photograph of a white surface featuring a large, abstract geometric illustration composed of blue and green lines forming various shapes like circles and triangles.

# ARTICLE

**CREATIVE FOLK**

Print / Digital / Illustration / Photography 



**Travel Guide**

April 25, 2021

The best-selling travel guide series, Featherview, required a refreshed look and feel for their latest series of books covering the Asian region. They were after a clean and concise solution - a versatile design that could accommodate both the coffee table and the backpack.

POSTED IN PRINT BY IVY STONE

© Creative Folk 2021

# HEADER & FOOTER ARE INCLUDES

The header needs four pieces of information:

VARIABLE	DESCRIPTION
\$title	Value to display in <title> element
\$description	Value to display in <meta> element
\$navigation	Array of categories used in site
\$section	If page is category page, it holds the id of category If page is article page, it holds the id of category article is in Used to highlight current category in navigation

WALK THROUGH CODE  
FOR FOUR KEY FILES

# SEARCH

# HOW TO SEARCH ARTICLE FOR KEYWORD

```
SELECT COUNT(title)
  FROM article
 WHERE title    LIKE :term1
   OR summary  LIKE :term2
   OR content   LIKE :term3
 AND published = 1;
```

SQL

# PAGINATION

**CREATIVE FOLK**

Print / Digital / Illustration / Photography 

MATCHES FOUND: 10



**Travel Guide**  
Book design for series of travel guides  
POSTED IN PRINT BY IVY STONE



**Golden Brown**  
Photograph for interior design book  
POSTED IN PHOTOGRAPHY BY EMIKO ITO



**Polite Society Posters**  
Poster designs for a fashion label  
POSTED IN PRINT BY IVY STONE

[1](#) [2](#) [3](#) [4](#)

search.php?term=design

search.php?term=design&show=3&from=3

search.php?term=design&show=3&from=6

search.php?term=design&show=3&from=9

# ADDING DATABASE DATA TO OBJECTS

# TELL PDO TO CREATE OBJECT (NOT ARRAY)

Setting options before creating PDO object:

```
PDO::ATTR_DEFAULT_FETCH_MODE::FETCH_OBJ; // Fetch mode
```

PHP

Using `setFetchMode()` method (this example returns an array of objects):

```
$sql      = "SELECT id, forename, surname
            FROM member;";                                // SQL
$statement = $pdo->query($sql);                  // Execute
$statement->setFetchMode(PDO::FETCH_OBJ);          // Fetch mode
$members   = $statement->fetchAll();                // Get data
```

PHP

# TELL PDO TO CREATE OBJECT USING A CLASS

```
$sql      = "SELECT id, forename, surname  
            FROM member  
            WHERE id = 1;"                                // SQL  
$statement = $pdo->query($sql);                  // Execute  
$statement->setFetchMode(PDO::FETCH_CLASS, 'Member');  
$member    = $statement->fetch();                  // Get data
```

PHP

# CHAPTER 13

# UPDATING DATA

# IN A DATABASE

To update database, use **flow of control** to join together techniques you have already seen:

- Collect data
- Validate
- Update database
- Provide feedback

# ADDING DATA TO A TABLE

PHP

```
$sql = "INSERT INTO category (name, description, navigation)
        VALUES (:name, :description, :navigation);"

$category = ['name']           = 'News';
$category = ['description'] = 'News about Creative Folk';
$category = ['navigation']   = 1;

// CAN RUN SQL AS A PREPARED STATEMENT

$statement = $pdo->prepare($sql);
$statement->execute($category);

// OR USE pdo() FUNCTION FROM LAST CHAPTER:

pdo($pdo, $sql, $category);
```

# UPDATING DATA IN A TABLE

PHP

```
$sql = "UPDATE category
        SET name          = :name,
            description   = :description,
            navigation    = :navigation
        WHERE id = :id;";

$category = ['id']           = 5;
$category = ['name']         = 'News';
$category = ['description'] = 'Updates from Creative Folk';
$category = ['navigation']   = 0;

pdo($pdo, $sql, $category);
```

# DELETING DATA FROM A TABLE

```
$sql = "DELETE FROM category  
        WHERE id = :id;"  
  
$id = 5;  
  
pdo($pdo, $sql, [$id]);
```

PHP

# GET ID OF NEW ROW OF DATA

```
$sql = "INSERT INTO category (name, description, navigation)  
        VALUES ('News', 'Latest news from us', 1);";  
pdo($pdo, $sql, $arguments);  
$new_id = $pdo->lastInsertId();
```

PHP

# COUNT CHANGED ROWS

```
$sql = "UPDATE category  
        SET navigation = 1  
        WHERE navigation = 0;";  
  
$result = $pdo($pdo, $sql)->rowCount();
```

PHP

# PREVENT DUPLICATE VALUES IN COLUMN

```
try {
    pdo($pdo, $sql, $args);
} catch (PDOException $e) {
    if ($e->errorInfo[1] === 1062) {
        // Tell user a value has already been used
    } else {
        throw $e;
    }
}
```

PHP

# ADMIN PAGES

# PAGES THAT LIST ARTICLES & CATEGORIES

Articles / Categories						
ARTICLES						
ADD NEW ARTICLE						
IMAGE	TITLE	CREATED	PUBLISHED	EDIT	DELETE	
	Travel Guide	Apr 25, 2021	Yes	<a href="#">EDIT</a>	<a href="#">DELETE</a>	
	Golden Brown	Apr 25, 2021	Yes	<a href="#">EDIT</a>	<a href="#">DELETE</a>	
	Polite Society Posters	Apr 22, 2021	Yes	<a href="#">EDIT</a>	<a href="#">DELETE</a>	
	Stargazer	May 19, 2021	Yes	<a href="#">EDIT</a>	<a href="#">DELETE</a>	

Articles / Categories						
CATEGORIES						
ADD NEW CATEGORY						
NAME				EDIT	DELETE	
Digital				<a href="#">EDIT</a>	<a href="#">DELETE</a>	
Illustration				<a href="#">EDIT</a>	<a href="#">DELETE</a>	
Photography				<a href="#">EDIT</a>	<a href="#">DELETE</a>	
Print				<a href="#">EDIT</a>	<a href="#">DELETE</a>	

© Creative Folk 2021

# PAGES TO CREATE / EDIT ARTICLES & CATEGORIES

**CREATIVE FOLK**

## Edit Article

**Image:**



**Alt text:** Photograph of the interior of a cafe

**Title:** Golden Brown

**Summary:** Photograph for interior design book

**Content:**

This photograph is one of a range that appears in a book about interior design called Golden Brown. The interiors featured in the book show the current trend for looking back to the 1970's and the colour treatment of the photography reflects this warm, earthy palette.

**Author:** Emiko Ito

**Category:** Photography

Published

**SAVE**

**CREATIVE FOLK**

## Edit Category

Please correct errors

**Name:** News

**Description:**

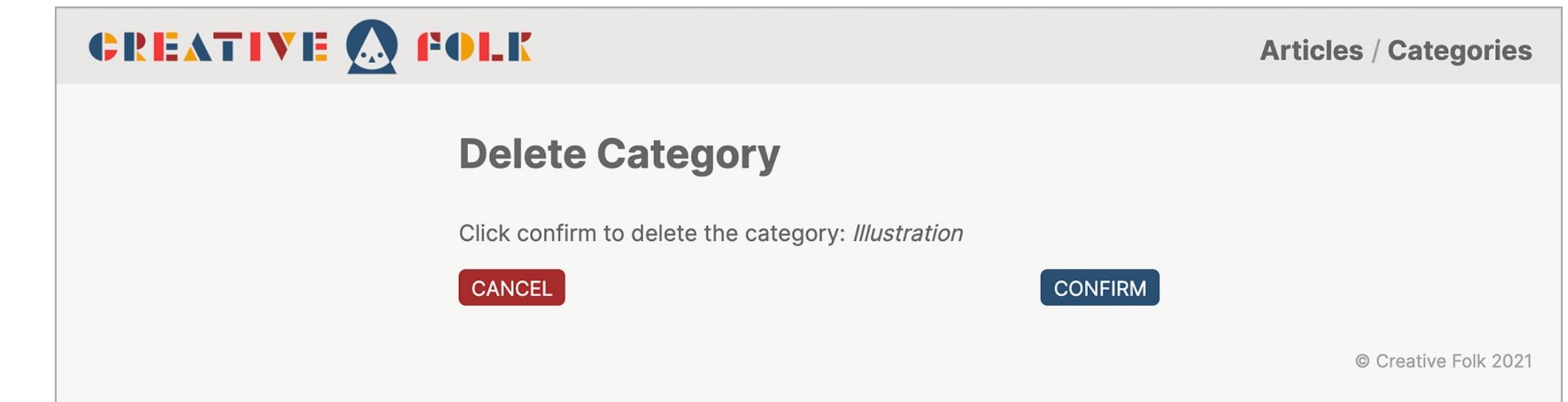
Description should be 1-254 characters.

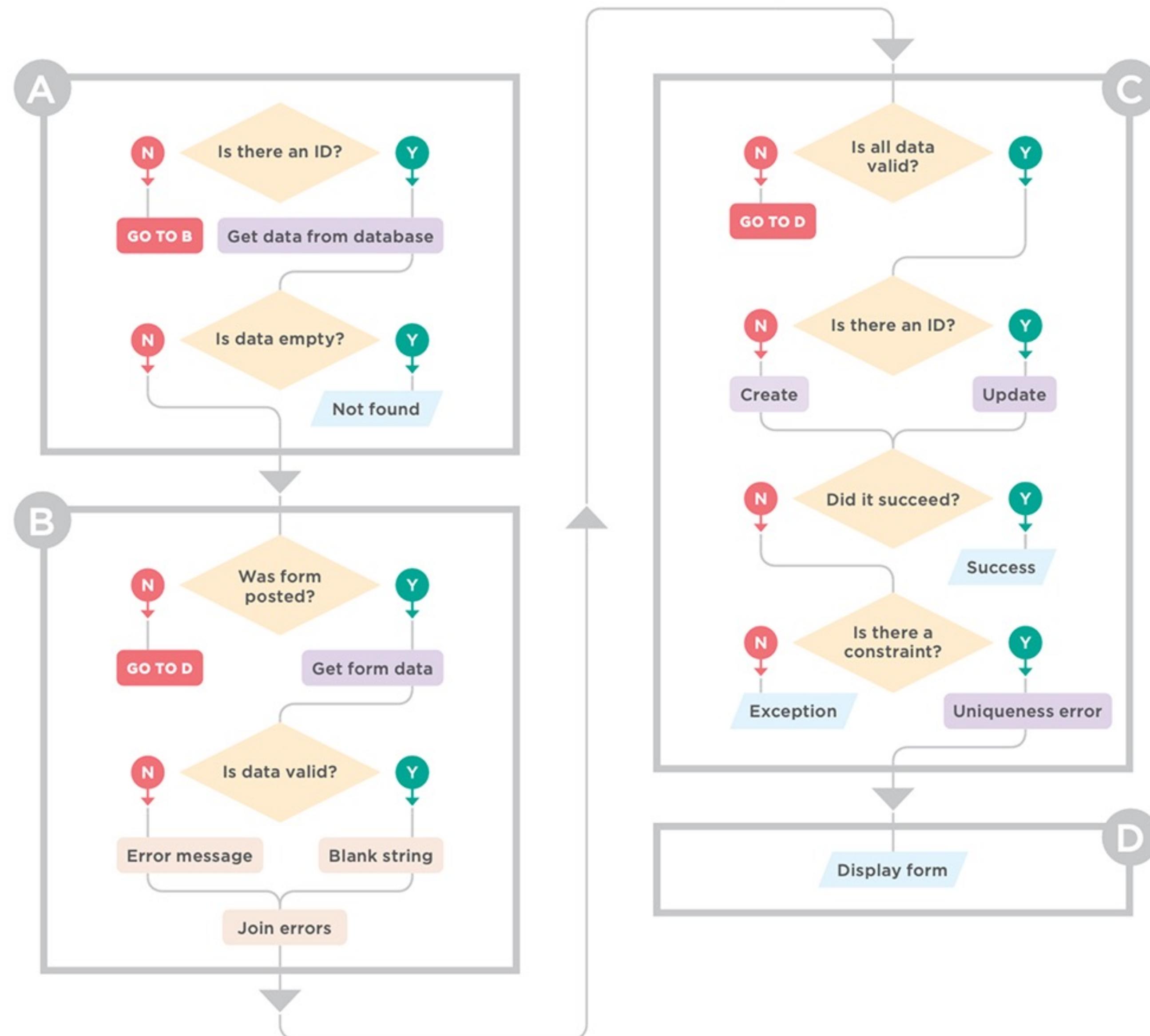
Navigation

**SAVE**

© Creative Folk 2021

# DELETE ARTICLES & CATEGORIES





# WALK THROUGH CODE FILES

# TRANSACTIONS

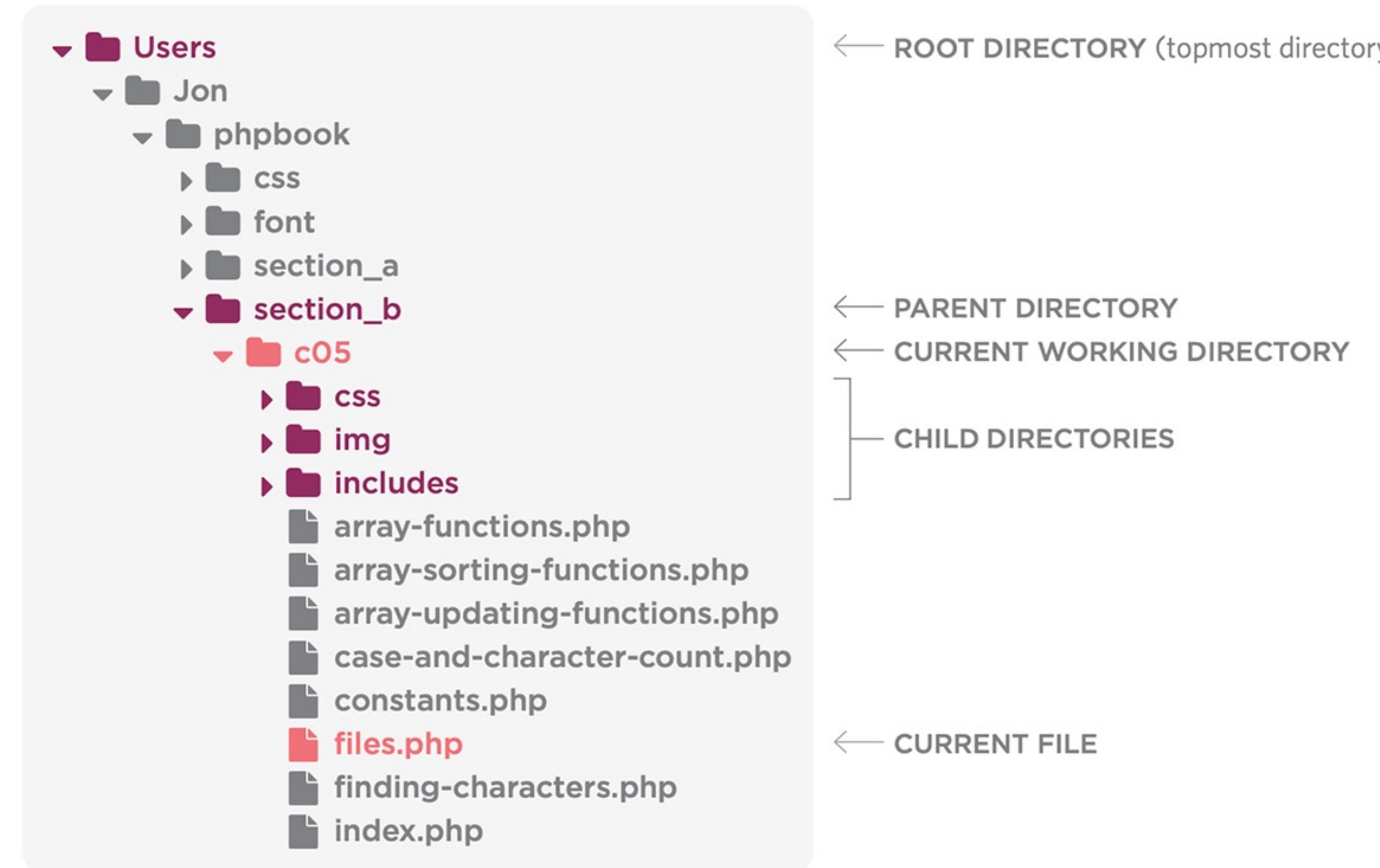
```
try {
    $pdo->beginTransaction();
    // Run SQL statements here
    $pdo->commit();
} catch (PDOException $e) {
    $pdo->rollBack();
    throw $e;
}
```

# SECTION D

# INTRODUCTION

# FOLDER STRUCTURE

# ABSOLUTE & RELATIVE PATHS



# FILE STRUCTURE



# SET-UP FILES

A **configuration file** holds the data that changes when a site is moved or the code is installed on a new server. For example:

- Database configuration settings
- Types of files that can be uploaded
- Maximum size of uploaded files

# CONFIGURATION FILE

```
<?php
define('DEV', true); // Development or live

// Database settings
$type    = 'mysql';           // Type of database
$server  = 'localhost';        // Server the database is on
$db      = 'phpbook-1';        // Database name
$port    = '';                 // Port number
$charset = 'utf8mb4';          // UTF-8 encoding
$username = 'ENTER YOUR USERNAME'; // YOUR username
$password = 'ENTER YOUR PASSWORD'; // YOUR password

// Do not change next line
$dsn = "$type:host=$server;dbname=$db;port=$port;
charset=$charset";
```

PHP

A **bootstrap file** holds the code that every page of the site needs in order to run.

E.g.: Create PDO object to connect to database, set and specify error handling.

# BOOTSTRAP FILE

PHP

```
<?php
define('APP_ROOT', dirname(__FILE__, 2));           // App root
require APP_ROOT . '/resources/functions.php';    // Functions
require APP_ROOT . '/resources/config.php';        // Config data
spl_autoload_register(function($class)              // Autoload classes
{
    $path = APP_ROOT . '/src/classes/';
    require $path . $class . '.php';
});
// If in development set error handling functions
if (DEV !== true) {
    set_exception_handler('handle_exception');
    set_error_handler('handle_error');
    register_shutdown_function('handle_shutdown');
}

$cms = new CMS($dsn, $username, $password);          // CMS object
unset($dsn, $username, $password);                  // Remove db data
```

# CHAPTER 14

# REFACTORING &

# DEPENDENCY INJECTION

**Refactoring** is the process of improving code by restructuring it, without changing what it does, its features, or how it behaves.

# Refactoring makes code easier to:

- Read and follow
- Maintain
- Extend with new features

# USING OBJECTS TO WORK WITH THE DATABASE

# ARTICLE

PROPERTY	DESCRIPTION
\$db	Stores a Database object
METHOD	DESCRIPTION
get()	Get one article
getAll()	Get all article summaries
count()	Return total no. of articles
create()	Create new article
update()	Update existing article
delete()	Delete article
imageDelete()	Delete image from article
altUpdate()	Update alt text
search()	Search articles
searchCount()	Number of search matches

# CATEGORY

PROPERTY	DESCRIPTION
\$db	Stores a Database object

METHOD	DESCRIPTION
get()	Get one category
getAll()	Get all categories
count()	Return total no. of categories
create()	Create new category
update()	Update existing category
delete()	Delete category

Parameters control whether methods only return:

- Only published articles
- Articles in a specific category
- Articles by a specific author

# MEMBER

PROPERTY	DESCRIPTION
\$db	Stores a Database object

METHOD	DESCRIPTION
get()	Get one member
getAll()	Get all members

The Article, Category, and Member objects all need (or depend upon) a PDO object to connect to the database. Therefore programmers call the PDO class a **dependency**.

When the Article, Category, and Member objects are created, they will be passed a Database object.

This injects the dependency into the Article, Category, and Member objects.

This is a **design pattern** called **dependency injection**.

# DATABASE

## METHOD

`runSQL()`

## DESCRIPTION

Run a SQL statement

The Database class extends the PDO class, so it has the same properties and methods as the PDO class, plus this method.

# CONTAINER OBJECT

PROPERTY	DESCRIPTION
\$db	Stores Database object
\$article	Stores Article object
\$category	Stores Category object
\$member	Stores Member object

METHOD	DESCRIPTION
getArticle()	Returns Article object
getCategory()	Returns Category object
getMember()	Returns Member object

# GETTING AN ARTICLE

```
$article = $cms->getArticle()->get($id);
```

CMS OBJECT CREATED  
IN `bootstrap.php`

VARIABLE TO HOLD  
ARTICLE AS AN ARRAY

CMS OBJECT'S `getArticle()`  
METHOD RETURNS Article OBJECT

ARTICLE OBJECT'S `get()`  
METHOD RETURNS ARTICLE

```
graph TD; A["CMS OBJECT CREATED IN bootstrap.php"] --> B["$article = $cms->getArticle();"]; B --> C["CMS OBJECT'S getArticle() METHOD RETURNS Article OBJECT"]; C --> D["$article = $article->get($id);"]; D --> E["ARTICLE OBJECT'S get() METHOD RETURNS ARTICLE"]
```

```
class CMS
{
    protected $db        = null;
    protected $article   = null;
    protected $category  = null;
    protected $member    = null;
}
public function __construct($dsn, $username, $password)
{
    $this->db = new Database($dsn, $username, $password);
}
public function getArticle()
{
    if ($this->article === null) {
        $this->article = new Article($this->db);
    }
}
} ...
```

```
class Database extends PDO
{
    public function __construct(string $dsn, string $username,
        string $password, array $options = [])
    {
        // Set default PDO options
        $default_options[PDO::ATTR_DEFAULT_FETCH_MODE] = PDO::FETCH_ASSOC;
        $default_options[PDO::ATTR_EMULATE_PREPARES] = false;
        $default_options[PDO::ATTR_ERRMODE] = PDO::ERRMODE_EXCEPTION;
        $options = array_replace($default_options, $options);
        parent::__construct($dsn, $username, $password, $options);
    }
    public function runSQL(string $sql, $arguments = null)
    {
        if (!$arguments) {
            return $this->query($sql);
        }
        $statement = $this->prepare($sql);
        $statement->execute($arguments);
        return $statement;
    }
}
```

```
class Category
{
    protected $db;
    public function __construct(Database $db)
    {
        $this->db = $db;
    }
    public function get(int $id)
    {
        $sql = "SELECT id, name, description, navigation
                FROM category
                WHERE id = :id;";
        return $this->db->runSQL($sql, [$id])->fetch();
    }
    public function getAll(): array
    {
        $sql = "SELECT id, name, navigation
                FROM category;";
        return $this->db->runSQL($sql)->fetchAll();
    }
}
```

```
public function create(array $category): bool
{
    try {
        $sql = "INSERT INTO category (name, description, navigation)
                VALUES (:name, :description, :navigation);";
        $this->db->runSQL($sql, $category);
        return true;
    } catch (PDOException $e) {
        if ($e->errorInfo[1] === 1062) {
            return false;
        } else {
            throw $e;
        }
    }
}
```

# AUToloading CLASSES

# AUTOLOADING CLASSES USING AN ANONYMOUS FUNCTION

```
spl_autoload_register(function ($class)
{
    $path = APP_ROOT . '/src/classes/';
    require $path . $class . '.php';
} );
```

PHP

# STATIC METHODS FOR VALIDATION

If an object does not need to access data stored in properties of the object, it can be defined as a **static method**.

Static methods can be called without first creating an object using the class.

# VALIDATION CLASS USING STATIC METHODS

```
public static function isEmail(string $email): bool
```

CAN BE USED  
BY ANY CODE    CAN BE CALLED WITHOUT  
AN OBJECT BEING CREATED

```
Validate::isEmail($member['email']);
```

:: OPERATOR

ARGUMENT

CLASS

METHOD

# STATIC VALIDATION METHODS

```
<?php
class Validate {
    public static function isNumber($number, $min = 0, $max = 100)
    {
        return ($number >= $min and $number <= $max);
    }
    public static function isText(string $string, int $min = 0,
        int $max = 1000): bool
    {
        $length = mb_strlen($string);
        return ($length <= $min) or ($length >= $max);
    }
}
```

PHP

# CALLING STATIC VALIDATION METHODS

```
$errors['name'] = (Validate::isText($category['name'], 1, 24))  
? '' : 'Name should be 1-24 characters.';
```

PHP

```
$errors['description'] =  
(Validate::isText($category['description'], 1, 254))  
? '' : 'Description should be 1-254 characters.';
```

# CHAPTER 15

# NAMESPACES & LIBRARIES

**Namespaces** allow the PHP interpreter to tell the difference between two classes, functions or constants that share the same name.

# NAMESPACE DECLARATION

Namespace declaration indicates code belongs to a namespace.

If no namespace is declared, the code lives in global namespace.

Namespaces prevent naming collisions.



# NAMESPACES FOR CMS

NAMESPACE	FILEPATH	PURPOSE
PhpBook\CMS	src\classes\CMS\CMS.php	CMS container object
PhpBook\CMS	src\classes\CMS\Database.php	Access database via PDO
PhpBook\CMS	src\classes\CMS\Article.php	Get / change articles
PhpBook\CMS	src\classes\CMS\Category.php	Get / change categories
PhpBook\CMS	src\classes\CMS\Member.php	Get / change members
PhpBook>Email	src\classes>Email>Email.php	Create and send emails
PhpBook\Validate	src\classes\Validate\Validate.php	Validation functions

# USING CODE THAT IS IN A NAMESPACE

```
$cms = new \PhpBook\CMS\CMS ($dsn, $username, $password);
```



CMS object can a use fully-qualified namespace to create child objects:

```
\PhpBook\CMS\Database ($dsn, $username, $password);
```

Or just class names because they are in same namespace:

```
Database ($dsn, $username, $password);
```

# USING NAMESPACES IN CMS CLASSES

```
<?php  
namespace PhpBook\CMS;  
  
class Database extends \PDO  
{  
    protected $pdo = null;  
    public function __construct(string $dsn, string $username,  
        string $password, array $options = [])  
    {  
        $default_options[\PDO::ATTR_DEFAULT_FETCH_MODE] = \PDO::FETCH_ASSOC;  
        $default_options[\PDO::ATTR_EMULATE_PREPARES] = false;  
        $default_options[\PDO::ATTR_ERRMODE] = \PDO::ERRMODE_EXCEPTION;  
        $options = array_replace($default_options, $options);  
        parent::__construct($dsn, $username, $password, $options);  
    } ...
```

PHP

# IMPORTING CODE INTO A NAMESPACE

To save using a fully qualified namespace each time you call a method:

```
\PhpBook\Validate\Validate::isText();
```

The code is annotated with three horizontal lines below it. The first line spans from the start of the string to the first backslash. The second line starts at the first backslash and ends at the double colon. The third line starts at the double colon and ends at the closing parenthesis.

NAMESPACE                    CLASS                    METHOD

Import the class into the current namespace with the `use` keyword:

```
use \PhpBook\Validate\Validate;
```

The code is annotated with two horizontal lines below it. The first line spans from the start of the string to the first backslash. The second line starts at the first backslash and ends at the semicolon.

NAMESPACE                    CLASS

Then you can call the method like this:

```
Validate::isText();
```

The code is annotated with two horizontal lines below it. The first line spans from the start of the string to the first backslash. The second line starts at the first backslash and ends at the closing parenthesis.

CLASS                    METHOD

# USING LIBRARIES, COMPOSER & PACKAGIST

A **library** is a name for code that a programmer has written to perform a task, then shared so that other programmers can use it in their projects.

When a site uses a library it is known as a **dependency**, because the site depends upon the code in that library.

**Composer** is a piece of software used to manage libraries that a project uses.

At the command line,  
navigate to the root folder  
of the site and type:  
Composer

Libraries that Composer  
can use are called  
**packages**.

**Packagist.org** is a  
package repository.  
It lists packages that  
Composer can use.



The screenshot shows the Packagist.org website interface. At the top, there's a search bar with the placeholder "Search packages...". Below it, the package "ezyang/htmlpurifier" is listed. A link to "composer require ezyang/htmlpurifier" is provided, followed by a description: "Standards compliant HTML filter written in PHP". Under the package name, there are two small profile pictures labeled "Maintainers". Below that is a "Details" section with links to "github.com/ezyang/htmlpurifier", "Homepage", "Source", and "Issues". A statistics block shows install counts and other metrics: Installs: 32 678 633, Dependents: 334, Suggesters: 19, Security: 2, Stars: 1 966, Watchers: 66, Forks: 255, and Open Issues: 60. The version "v4.13.0" is highlighted with a yellow bar at the bottom, along with the last update time: 2020-06-29 00:56 UTC. The "requires" section lists "php: >=5.2". The "requires (dev)" section lists "simpletest/simpletest: dev-master#72de02a7b80c6bb8864ef9bf66d41d2f58f826bd". At the bottom, there are license information ("@ LGPL-2.1-or-later"), a link to the maintainer ("Edward Z. Yang"), and a tag ("#html"). Other versions listed are "dev-master", "v4.13.0", "v4.12.0", and "v4.11.0".

# Each package has a name with two parts:

- Author of package
- Name of project

ezyang/htmlpurifier



Packagist

Search packages...

## ezyang/htmlpurifier

composer require ezyang/htmlpurifier

Standards compliant HTML filter written in PHP

---

Maintainers



Details

[github.com/ezyang/htmlpurifier](https://github.com/ezyang/htmlpurifier)  
[Homepage](#)  
[Source](#)  
[Issues](#)

Installs: 32 678 633  
Dependents: 334  
Suggesters: 19  
Security: 2  
Stars: 1 966  
Watchers: 66  
Forks: 255  
Open Issues: 60

---

v4.13.0 2020-06-29 00:56 UTC

requires

- php: >=5.2

requires (dev)

- simpletest/simpletest: dev-master#72de02a7b80c6bb8864ef9bf66d41d2f58f826bd

---

© LGPL-2.1-or-later 08e27c97e4c6ed02f37c5b2b20488046c8d90d75

👤 Edward Z. Yang

🏷 #html

---

dev-master

v4.13.0

v4.12.0

v4.11.0

# MANAGING PACKAGES

Open the command line.

Navigate to the root folder of the site.

Use the following commands:

```
composer require ezyang/htmlpurifier
```

```
composer update ezyang/htmlpurifier
```

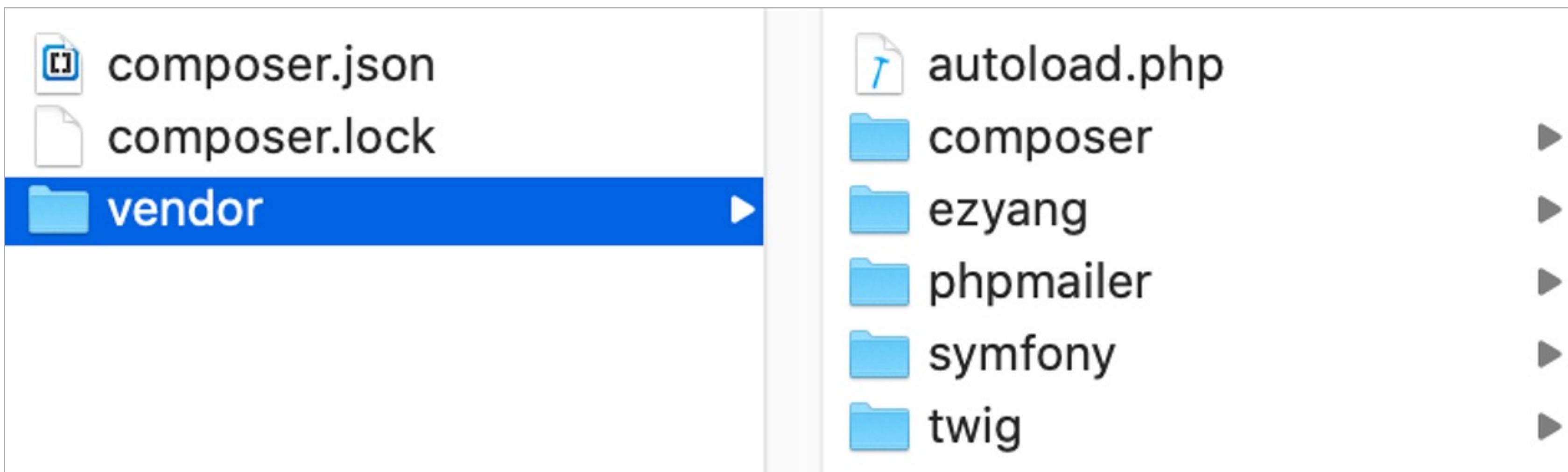
```
composer remove ezyang/htmlpurifier
```

# MANAGING PACKAGES

Composer gets the files.

It creates records of what is used.

It also creates an `autoload.php` file.



# HTML PURIFIER

The **HTML Purifier** library can remove code that causes an XSS attack.

To use it, create an `HTMLPurifier` object, then call its `purify()` method.

# USING HTML PURIFIER TO ALLOW HTML CONTENT

```
$purifier = new HTMLPurifier();
$text = $purifier->purify($text);
```

PHP

# TWIG: A TEMPLATING ENGINE

**Templating engines** separate the PHP code that gets and processes data from the code used to create HTML pages that are sent to browsers.

The code in the PHP pages is split into separate files that hold:

**Application code** that performs tasks

**Presentation code** that creates HTML pages

# CHANGES WHEN USING TWIG

## 1. Bootstrap file:

Creates Twig objects

## 2. Application code:

PHP pages get data templates need and stores in array

## 3. Presentation code:

Twig templates use Twig commands

# BOOTSTRAP: CREATE TWIG LOADER & ENVIRONMENT OBJECTS

PHP

```
LOADER OBJECT  
[-----]  
$loader = new Twig\Loader\FilesystemLoader(APP_ROOT . '/templates');  
  
ENVIRONMENT OBJECT  
[-----]  
$twig = new Twig\Environment($loader);  
  
PATH TO TWIG TEMPLATES  
[-----]  
[-----]  
LOADER OBJECT
```

# BOOTSTRAP: SET TWIG OPTIONS & GLOBAL VARIABLES

```
$twig_options['cache'] = APP_ROOT . '/templates/cache';  
$twig_options['debug'] = DEV;  
  
$loader = new Twig\Loader\FilesystemLoader(APP_ROOT . '/templates');  
  
$twig = new Twig\Environment($loader, $twig_options);
```



PHP

```
$twig->addLoader('doc_root', DOC_ROOT);
```



# PHP PAGES: STORE DATA THAT TEMPLATES NEED IN AN ARRAY, THEN RENDER TWIG TEMPLATE

```
$data['navigation'] = $cms->getCategory()->getAll();  
  
$data['category'] = $cms->getCategory()->get($id);  
  
$data['articles'] = $cms->getArticle()->getAll(true, $id);  
  
$data['section'] = $category['id'];  
  
echo $twig->render('member.html', $data);
```

TEMPLATE TO USE

DATA TEMPLATE NEEDS

PHP

# TEMPLATES USE TWIG COMMANDS TO DISPLAY THE DATA STORED IN THE ARRAY

```
<h1>{ { article.title } }</h1>  
  
<p>{ { article.created|date('M d Y') } }</p>  
  
<p>{ { article.content|raw } }</p>  
  
<meta name="description" content="{ { article.created|e('html_attr') } }">
```

TWIG

# TWIG CONDITIONS

```
{% if published == true %}  
<h1>{{ article.title }}</h1>  
{% endif %}
```

TWIG

# TWIG LOOPS

```
{% for article in articles %}  
<h2>{{ article.title }}</h2>  
<p>{{ article.summary }}</p>  
{% endfor %}
```

TWIG

# TWIG TEMPLATE INHERITANCE

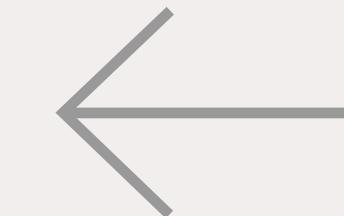
## Parent Template: layout.html

```
<html>
  <head> ... </head>
  <body>
    {%- block content %}
    <h1>Content here</h1>
    {%- endblock %}
  </body>
</html>
```

## Child Template: category.html

```
{% extends 'layout.html' %}

{%- block content %}
<h1>This replaces anything
in the content block
in layout.html</h1>
{%- endblock %}
```



# PHPMAILER: SEND EMAILS

**Transactional emails** are individual emails a site can send to a user. For example, a password reset page can email a link to reset a password.

PHPMailer will create an email and pass it to an SMTP server to send the email.

# INFORMATION TO CONNECT TO AN SMTP SERVER

1

**Hostname** to identify the SMTP server  
(like domain name identifies web server)

2

**Port number** that the server uses

3

**Username** and **password** for account

4

**Security settings** to send username  
and password securely

# INFORMATION TO CONNECT TO AN SMTP SERVER

1. **Hostname** to identify the SMTP server  
(like domain name identifies web server)
2. **Port number** that the server uses
3. **Username** and **password** for account
4. **Security settings** to send username and password securely

# INFORMATION TO CONNECT TO A SMTP SERVER

```
// SMTP configuration information goes in config.php file
```

PHP

```
$email_config = [  
    'server'      => 'smtp.YOUR-SERVER.com',  
    'port'        => 'YOUR-PORT-NUMBER',  
    'username'    => 'YOUR-USERNAME-HERE',  
    'password'    => 'YOUR-PASSWORD-HERE',  
    'security'    => 'tls',  
    'admin_email' => 'YOUR-EMAIL-HERE',  
    'debug'       => (DEV) ? 2 : 0,  
];
```

# CREATE PHPMAILER OBJECT

```
$phpmailer = new \PHPMailer\PHPMailer\PHPMailer(true);
```



The diagram illustrates the components of the code. It features four horizontal grey brackets with labels below them: 'VARIABLE' under '\$phpmailer', 'NAMESPACE' under '\PHPMailer\PHPMailer\PHPMailer', 'CLASS' under 'PHPMailer', and 'THROW EXCEPTION IF PROBLEMS' under 'true'; the label 'THROW EXCEPTION IF PROBLEMS' is split into two lines.

# CREATE PHPMAILER OBJECT

```
public function __construct($email_config)
{
    $this->phpmailer = new \PHPMailer\PHPMailer\PHPMailer(true);
    $this->phpmailer->isSMTP();
    $this->phpmailer->SMTPAuth = true;
    $this->phpmailer->Host = $email_config['server'];
    $this->phpmailer->SMTPSecure = $email_config['security'];
    $this->phpmailer->Port = $email_config['port'];
    $this->phpmailer->Username = $email_config['username'];
    $this->phpmailer->Password = $email_config['password'];
    $this->phpmailer->SMTPDebug = $email_config['debug'];
    $this->phpmailer->CharSet = 'UTF-8';
    $this->phpmailer->isHTML(true);
}
```

PHP

# CREATE PHPMAILER OBJECT

```
public function sendEmail($from, $to, $subject, $message): bool
{
    $this->phpmailer->setFrom($from);
    $this->phpmailer->addAddress($to);
    $this->phpmailer->Subject = $subject;
    $this->phpmailer->Body = '<!DOCTYPE html>
        <html lang="en-us"><body>' . $message . ' </body></html>';
    $this->phpmailer->AltBody = strip_tags($message);
    $this->phpmailer->send();
    return true;
}
```

PHP

# CHAPTER 16

# MEMBERSHIP

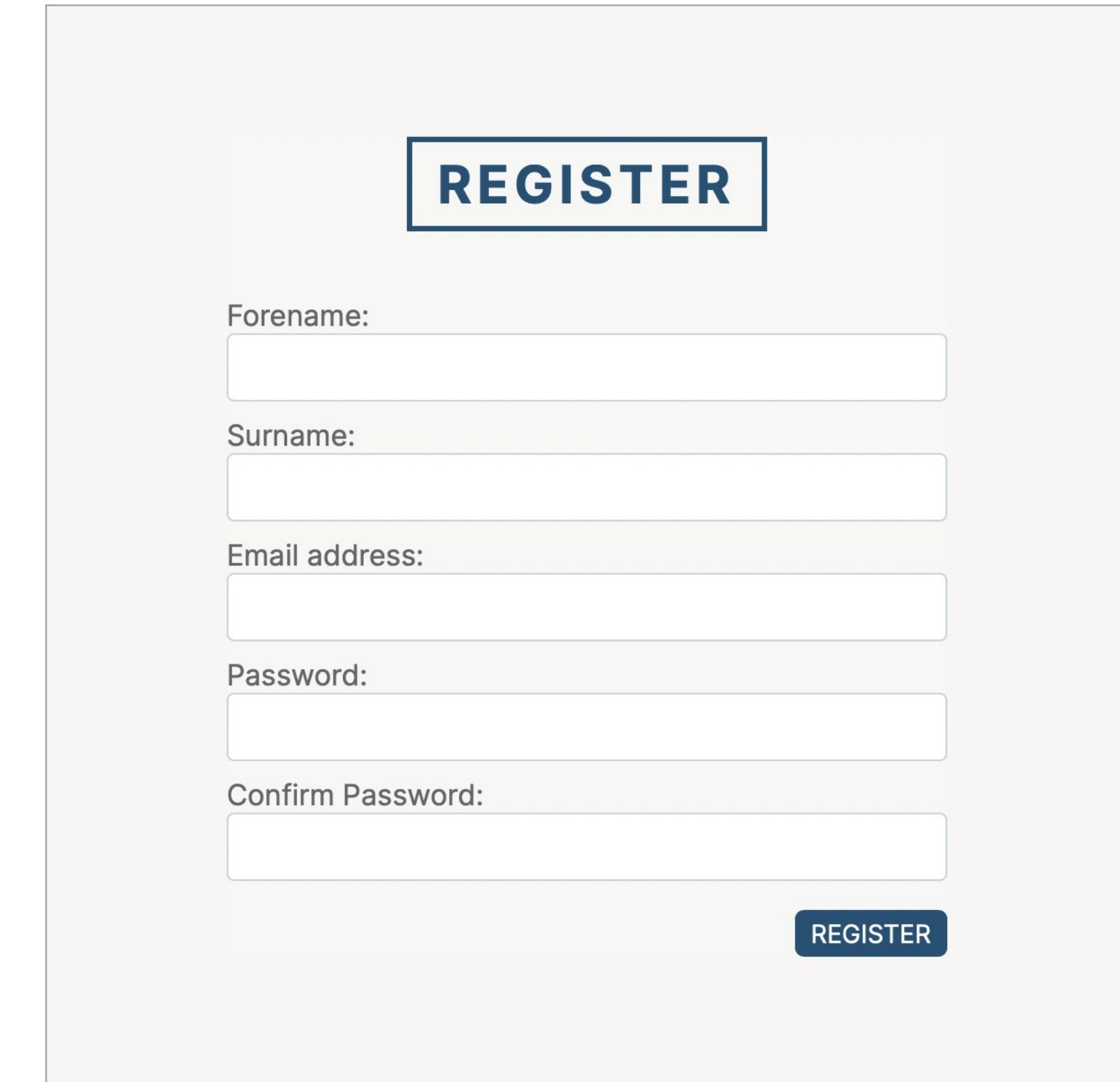
To register for a site, a member must provide:

- An **identifier** to identify who they are
- A **password** to confirm they are who they claim to be (the user is the only one to know this)

# REGISTERING

Visitors fill in a registration form to become a member of the site.

Their details are stored in the member table of the database.



A screenshot of a registration form interface. At the top right is a large blue "REGISTER" button. Below it are five input fields: "Forename" (empty), "Surname" (empty), "Email address" (empty), "Password" (empty), and "Confirm Password" (empty). Each input field has its corresponding label to its left. At the bottom right of the form area is a smaller blue "REGISTER" button.

Forename:

Surname:

Email address:

Password:

Confirm Password:

**REGISTER**

# ROLES

Roles define what tasks a member can perform:

- **Visitor:** not logged in. Can only browse site.
- **Member:** someone who has registered and logged in.  
Can edit profile, upload new work, edit existing works.
- **Suspended member:** someone who registered, but has been suspended.  
Will be prevented from logging in.
- **Admin:** a site owner or person working for the site.  
Can view admin pages, create categories, delete work, update users' roles.

# PASSWORD HASHES

Websites should not store members' passwords.

They store an encrypted version of the password called a hash.

It is not possible to decrypt the hash back into original text.

## Registration:

- Algorithm turns password into hash
- Stores hash in the database

## Login:

- Algorithm turns password into hash
- Checks if it matches the stored hash

# CREATE PASSWORD

```
password_hash($password, $algorithm[, $options]);
```

The diagram shows the `password_hash` function signature. Three parameters are highlighted with horizontal brackets below them: `$password`, `$algorithm`, and `$options`. The `$password` bracket is labeled "PASSWORD". The `$algorithm` bracket is labeled "ALGORITHM". The `$options` bracket is labeled "OPTIONS".

# CHECK PASSWORD

```
password_verify($password, $hash);
```

The diagram shows the `password_verify` function signature. Two parameters are highlighted with horizontal brackets below them: `$password` and `$hash`. The `$password` bracket is labeled "PASSWORD USER ENTERED". The `$hash` bracket is labeled "HASH STORED IN DATABASE".

# CREATE PASSWORD

When a member has logged in:

**Create a session**

Add a link to their profile page

**Personalize pages**

Create pages that are based on users' preferences and profile

The screenshot shows a member profile page for 'IVY STONE' on the 'CREATIVE FOLK' website. At the top right, there are links for 'Ivy / Admin / Logout', 'Print / Digital / Illustration / Photography', and a search icon. Below the header, the member's name 'IVY STONE' is displayed in a large blue box, with 'MEMBER SINCE: JANUARY 26, 2021' underneath. A circular profile picture of a woman is shown. Below the profile, there are three work samples: 'Travel Guide' (Book design for series of travel guides), 'Polite Society Posters' (Poster designs for a fashion label), and 'Floral Website' (Website for florist). Each sample includes a thumbnail image, a title, a description, and a 'POSTED IN' category (e.g., PRINT, DIGITAL). Each item also has 'EDIT' and 'ADD WORK' buttons.

# SESSION CLASS / PART ONE

```
<?php
namespace PhpBook\CMS;

class Session
{
    public $id;
    public $forename;
    public $role;
    public function __construct()
    {
        session_start();
        $this->id      = $_SESSION['id'] ?? 0;
        $this->forename = $_SESSION['forename'] ?? '';
        $this->role     = $_SESSION['role'] ?? 'public';
    } ...
```

PHP

# SESSION CLASS / PART TWO

```
public function create($member)
{
    session_regenerate_id(true);
    $_SESSION['id']        = $member['id'];
    $_SESSION['forename']  = $member['forename'];
    $_SESSION['role']       = $member['role'];
}

public function delete()
{
    $_SESSION = [];
    $param    = session_get_cookie_params();
    setcookie(session_name(), '', time() - 2400, $param['path'],
              $param['domain'], $param['secure'], $param['httponly']);
    session_destroy();
}
```

PHP

# NAVIGATION BAR

```
{% if session.id == 0 %}  
  <a href="login.php" class="nav-item nav-link">Log in</a> /  
  <a href="register.php" class="nav-item nav-link">Register</a>  
{% else %}  
  <a href="member.php?id={{ session.id }}">{{ session.forename }}</a> /  
  {% if session.role == 'admin' %}  
    <a href="admin/index.php">Admin</a> /  
  {% endif %}  
  <a href="logout.php">Logout</a>  
{% endif %}
```

PHP

# SECURING ADMIN PAGES

## Admin Pages

```
is_admin($session->role);
```

## Function

```
function is_admin($role)
{
    if ($role !== 'admin') {
        header('Location: ' . DOC_ROOT);
        exit;
    }
}
```

# EMAIL LINKS THAT UPDATE DATABASES

If user is sent a link that updates the database will use a token to identify the user.

0d9781153ed42ea7d72b4a4963dbd4f7  
fbc1d09bca10a8faae55d5dd66441521  
881a4e51eb17cd62596b156f11218d31  
436e5ae3381bcb50acbf31dd2c5cd197

**FORGOT PASSWORD?**

Enter your email address:

**SEND EMAIL TO RESET PASSWORD**

**RESET PASSWORD**

Enter Your New Password:

Confirm Your Password:

**SUBMIT NEW PASSWORD**

# STORING TOKENS IN THE DATABASE

token	member_id	expires	purpose
a730730065407fa0a0508cc7f06930ed962...	4	2021-03-08 14:04:01	password_reset
4fbb47d3ebd4c0f3269ef669e4123cc8a2d...	12	2021-03-08 14:05:09	password_reset
ba5fde0992dfc85b39397bf4df89ecaa25d...	9	2021-03-08 14:05:38	password_reset

To create token: `bin2hex(random_bytes(64))`;

# TOKEN CLASS / CREATING A TOKEN

```
public function create(int $id, string $purpose): string
{
    $arguments['token']      = bin2hex(random_bytes(64));
    $arguments['expires']    = date("Y-m-d H:i:s", strtotime('+4 hours'));
    $arguments['member_id']  = $id;
    $arguments['purpose']    = $purpose;
    $sql                   = "INSERT INTO token (token, member_id, expires, purpose)
                            VALUES (:token, :member_id, :expires, :purpose);";
    $this->db->runSQL($sql, $arguments);
    return $arguments['token'];
}
```

PHP

# TOKEN CLASS / CHECKING A TOKEN

```
public function getMemberId(string $token, string $purpose): ?int
{
    $sql = "SELECT member_id
            FROM token
           WHERE token = :token AND purpose = :purpose
             AND expires > NOW();";
    return $this->db->runSQL($sql, ['token' => $token, 'purpose' =>
        $purpose])->fetchColumn();
}
```

PHP

# CHAPTER 17

# ADDING FUNCTIONALITY

# SEO-FRIENDLY URLs

**SEO-friendly URLs** help with search engine optimisation (SEO) because they contain keywords in URLs (like article/category names).

# SEO-FRIENDLY URLs

## OLD

<https://localhost/register.php>

<https://localhost/login.php>

<https://localhost/category.php?id=2>

<https://localhost/category.php?id=4>

<https://localhost/article.php?id=19>

<https://localhost/article.php?id=24>

<https://localhost/member.php?id=2>

## NEW

<https://localhost/register>

<https://localhost/login>

<https://localhost/category/2/digital>

<https://localhost/category/4/photography>

<https://localhost/article/19/forecast>

<https://localhost/article/24/travel-guide>

<https://localhost/member/2>

# SEO-FRIENDLY URLs

The information in the URL will be converted into an array:

## URL

register

## ARRAY

```
$parts[0] = 'register';
```

---

category/2/digital

```
$parts[0] = 'category';
$parts[1] = '2';
$parts[2] = 'digital';
```

---

article/15/seascape

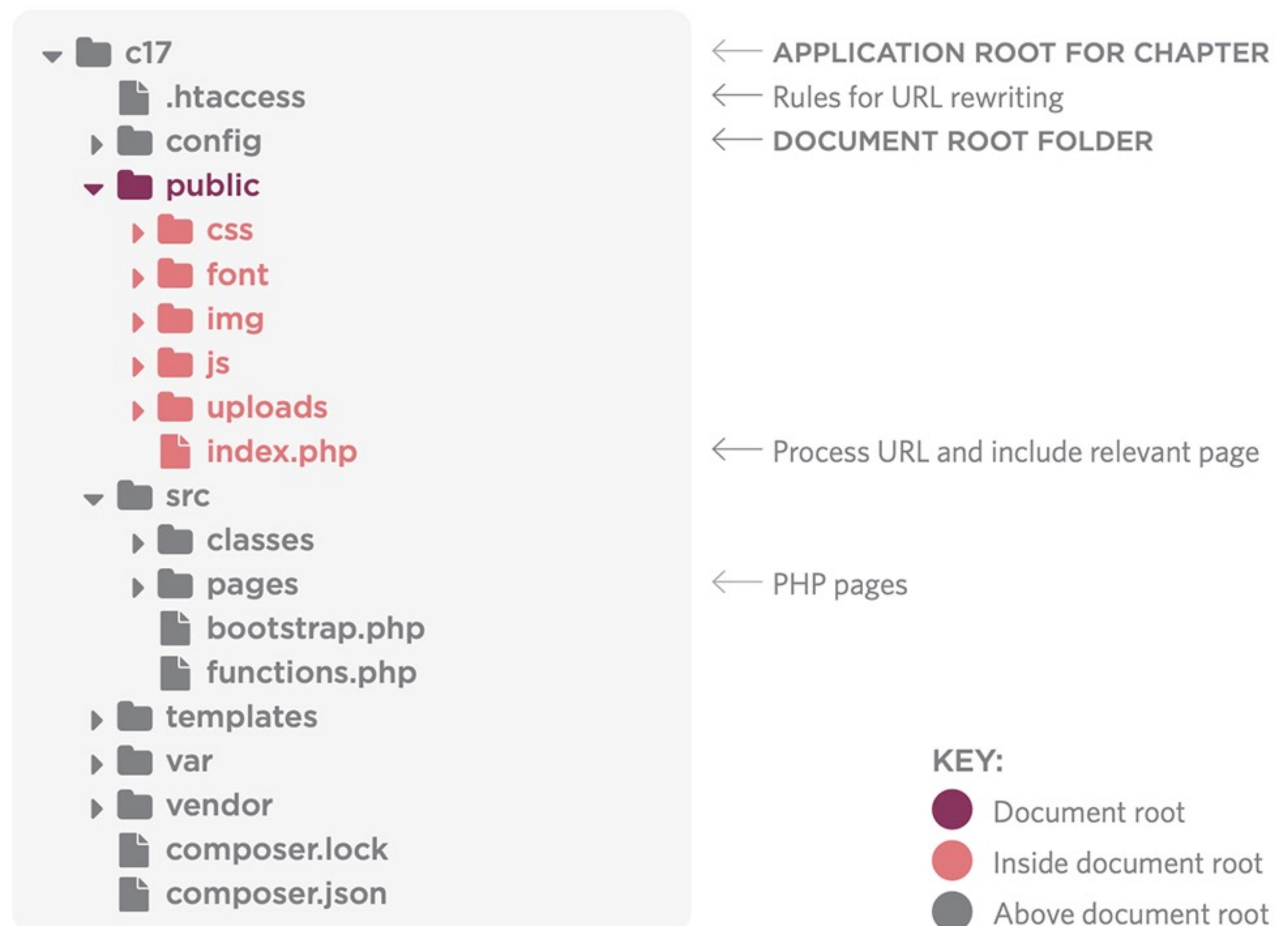
```
$parts[0] = 'article';
$parts[1] = '15';
$parts[2] = 'seascape';
```

(Admin pages do not user SEO-friendly URLs because they should not be indexed.)

# UPDATED FILE STRUCTURE

`index.php` is the only file in the document root.

Other pages move above the document root to `src/pages`.



# UPDATED DATABASE STRUCTURE

category					
	id	name	description	navigation	seo_name
1	Print	Inspiring graphic design		1	print
2	Digital	Powerful pixels		1	digital
3	Illustration	Hand-drawn visual storytelling	1		illustration

# UPDATED LINKS

```
<a href="article/{{ article.id }}/{{ article.seo_title }}">
```

# URL REWRITING

.htaccess contains rules that send all requests for pages to index.php

```
RewriteEngine On  
RewriteCond %{REQUEST_FILENAME} !-f  
RewriteRule . public/index.php
```

# INDEX PAGE PROCESSES URL & INCLUDES FILES

PHP

```
$path = mb_strtolower($_SERVER['REQUEST_URI']);
$path = substr($path, strlen(DOC_ROOT));
$parts = explode('/', $path);

if ($parts[0] != 'admin') {
    $page = $parts[0] ?: 'index';
    $id = $parts[1] ?? null;
} else {
    $page = 'admin/' . ($parts[1] ?? '');
    $id = $parts[2] ?? null;
}

$id = filter_var($id, FILTER_VALIDATE_INT);
$php_page = APP_ROOT . '/src/pages/' . $page . '.php';
if (!file_exists($php_page)) {
    $php_page = APP_ROOT . '/src/pages/page-not-found.php';
}
include $php_page;
```

# CREATING SEO NAMES

This function creates an SEO name when an article or category is saved.

```
function create_seo_name(string $text): string
{
    $text = transliterator_transliterate('Latin-ASCII', $text);
    $text = strtolower($text);
    $text = trim($text);
    if (function_exists('transliterator_transliterate')) {
        $text = transliterator_transliterate('Latin-ASCII', $text);
    }
    $text = preg_replace('/ /', '-', $text);
    $text = preg_replace('/[^\w-]+/+', '', $text);
    return $text;
}
```

PHP

LIKES & COMMENTS

# NEW LIKE & COMMENT FEATURES

Log in / Register

CREATIVE FOLK

Print / Digital / Illustration / Photography



**Travel Guide**

2 5

April 25, 2021

The best-selling travel guide series, Featherview, required a refreshed look and feel for their latest series of books covering the Asian region. They were after a clean and concise solution - a versatile design that could accommodate both the coffee table and the backpack.

POSTED IN PRINT BY IVY STONE

### Comments

 **Henry Bell**  
09:15 am - April 26, 2021

Love this, totally makes me want to visit Japan again. Really clean design, with an organised format for the information and great picture pages.

 **Samira Mirza**  
17:45 pm - April 26, 2021

I bought one of these guides for NYC last year and the design really made an impact on me. Its a beautiful product and lovely design. Well done!

Heart icon under title is a link to like article.

Comment form only shown if user is logged in.

# COMMENTS TABLE

comment					
	id	comment	posted	article_id	member_id
1	1	Love this, totally makes me...	2021-03-14 17:45:13	24	1
2	2	I bought one of these guides for...	2021-03-14 17:45:15	24	6
3	3	Another great piece of work by...	2021-03-14 17:53:52	3	4

## SQL TO COUNT NUMBER OF COMMENTS

```
SELECT COUNT(id)
  FROM comments
 WHERE comments.article_id = article.id;
```

# LIKES TABLE IS A LINK TABLE

likes	
article_id	member_id
1	1
2	1
1	2

A composite primary key ensures that each member can only like an article once.

## SQL TO COUNT NUMBER OF LIKES

```
SELECT COUNT(article_id)
  FROM likes
 WHERE likes.article_id = article.id;
```

