



IT TECHNICAL TRAINING

# **GET INTO TECH PROGRAMME**

DELEGATE GUIDE

V3.0

Visit your learning portal:

**QA.COM/myQA**



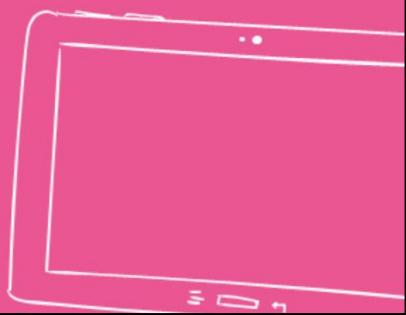
sky



GET INTO TECH



QA



## CHAPTER OVERVIEW

- Program introduction
- Program outline
- Course aims and objectives
- Introductions
- Housekeeping

## PROGRAM OUTLINE: STANDARD

- Week 0: Setup
- Week 1:  
Immersion week
- Week 2 to 14:  
  
Evening training  
session  
Evening hangout  
session
- Week 15: Showcase

Introduction to Programming

HTML / Cascading Style Sheets /  
JavaScript

PHP

MySQL

## PROGRAM OUTLINE: YOUNG WOMEN

- Week 0: Setup
- Week 1:  
Immersion week
- Week 2 to 8:  
  
Evening training  
session  
Evening hangout  
session
- Week 9: Showcase

Introduction to Programming

HTML / Cascading Style Sheets /  
JavaScript

PHP

MySQL

## COURSE AIMS AND OBJECTIVES

- Understand software development
- Create web applications using HTML, CSS, JavaScript and PHP
- Understand Object Oriented Programming (OOP)
- Interact with a database using MySQL

# INTRODUCTIONS





# HOUSEKEEPING!



# AN INTRODUCTION TO PROGRAMMING

## CHAPTER OVERVIEW

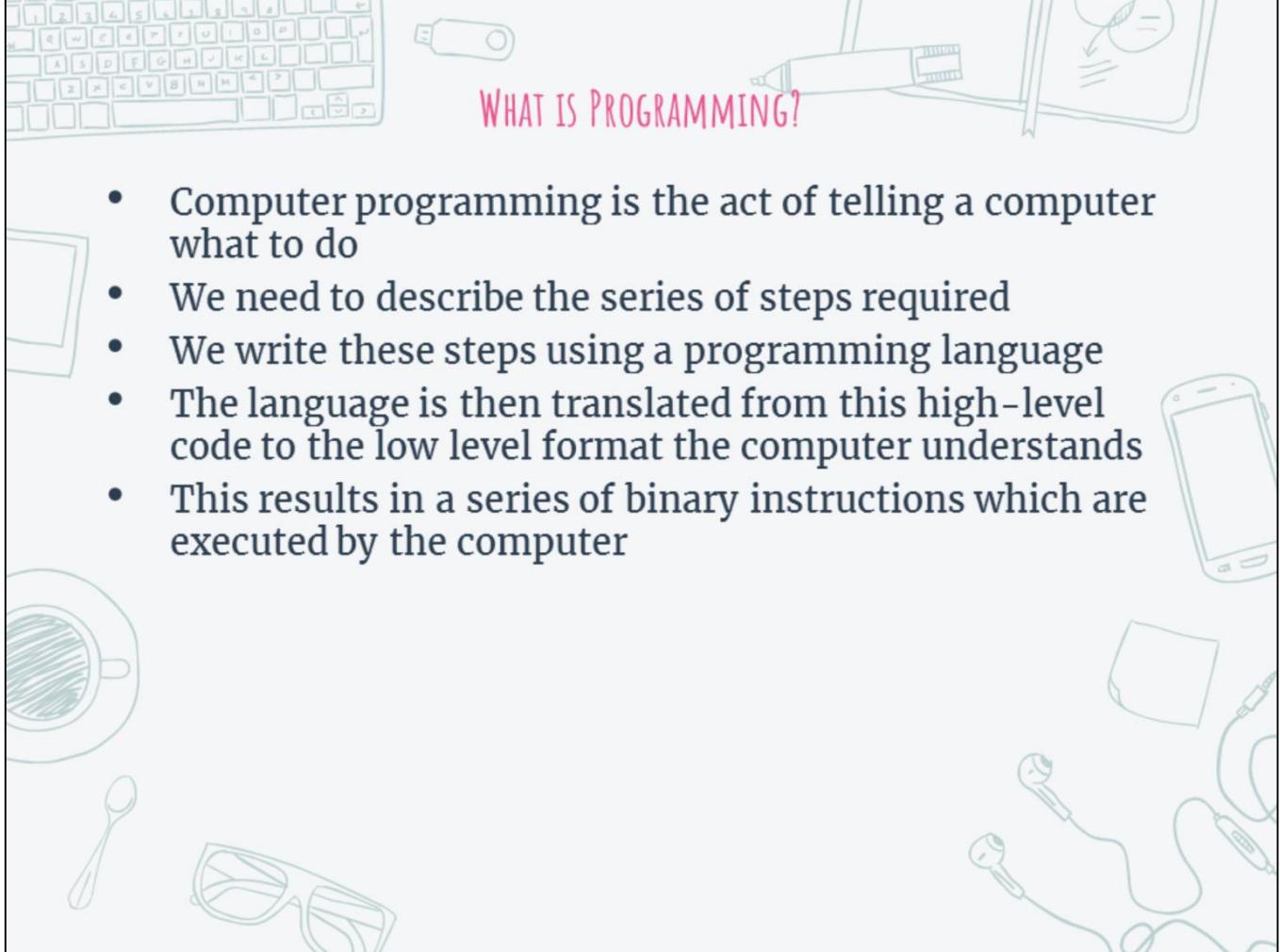
- What is a program?
- What is programming?
- Programmer activities
- How do programmers work?
- Agile and Scrum
- Essential programmer traits

## WHAT IS A PROGRAM?



A program is a sequence of instructions. A program is not necessarily for a computer. It could be a set of instructions for a human to undertake or a refrigerator or even a car engine. Programs describe the process of doing something. Often this involves taking some inputs and acting on them to produce some output. You can think of this as like baking a cake; you take the raw ingredients such as flour, butter, sugar and eggs and sift, beat and mix them together. You bake your mixture for a certain time at a certain temperature and your output from this program is a cake.

Computer programs follow a similar process. You can take inputs, such as data from another system or gathered from a user and manipulate it to suit your requirements. In order to give your instructions to a computer you need to be able to speak a language the computer can understand. Computers understand binary data which is a collection of zeros and ones. Fortunately there are many “high level” languages to choose from that are more human-readable which will translate into the binary values the computer requires.



## WHAT IS PROGRAMMING?

- Computer programming is the act of telling a computer what to do
- We need to describe the series of steps required
- We write these steps using a programming language
- The language is then translated from this high-level code to the low level format the computer understands
- This results in a series of binary instructions which are executed by the computer

There are many programming languages to choose from. Can you list any? The language that a developer chooses often depends on what kind of program they are creating as well as corporate standards and practices. Even though there are many languages to choose from many of them share similarities in the words and grammar that they use. Ultimately the language that you code in will be translated into machine code. This translation process is sometimes referred to as “interpreted” or “compiled” code. Interpreted code is code that is run dynamically. In other words, the computer is performing the translation as it goes along. Languages that are compiled rather than interpreted are translated first, before execution begins.



# ACTIVITY

Your instructor will ask you to work in pairs and to write a program to make toast.  
You will discuss your programs once completed.



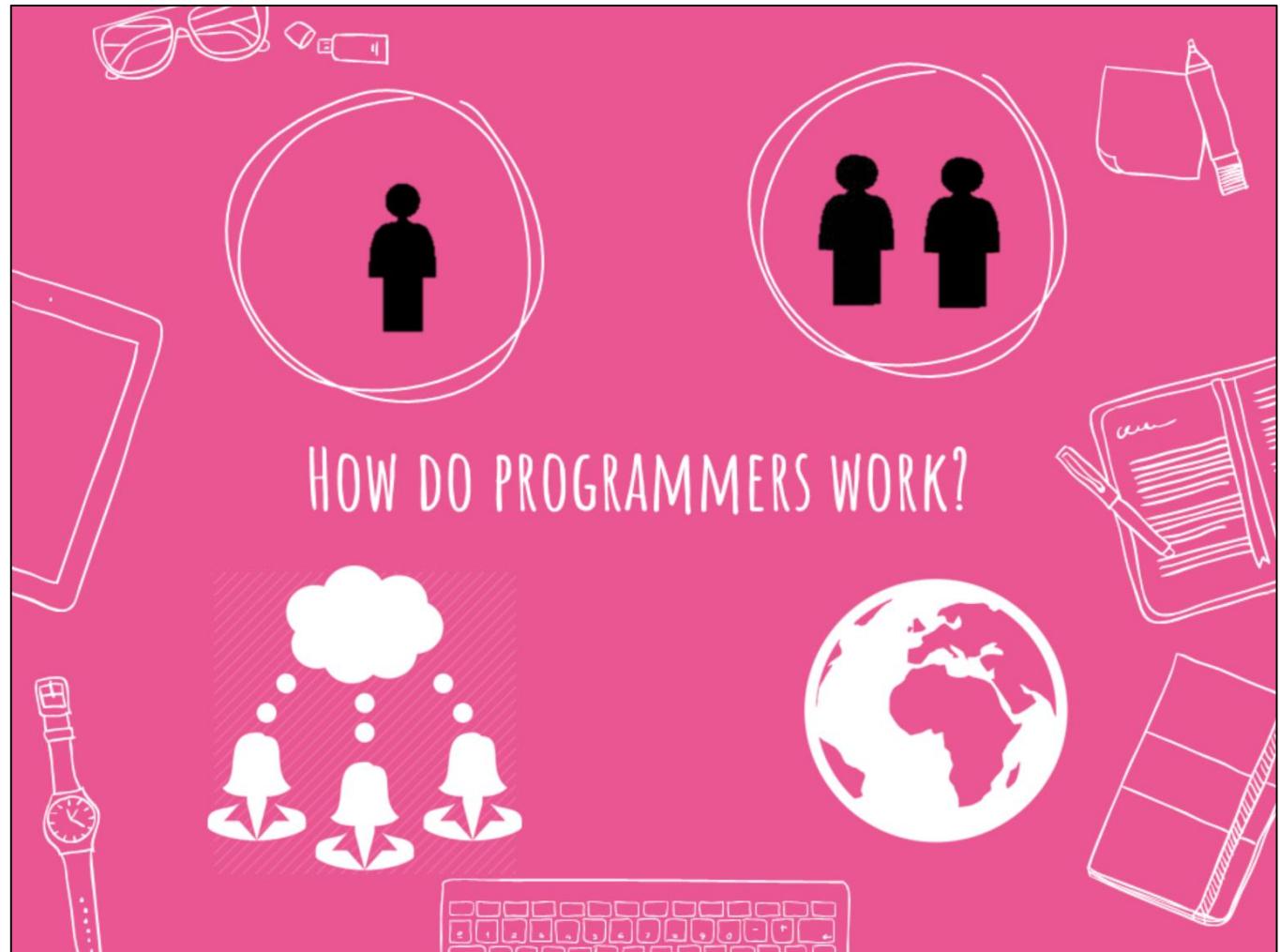
## BRAINSTORM

As a class, brainstorm a list of all of the tasks you think are performed by a programmer.

## PROGRAMMER ACTIVITIES

- Discover user requirements
- Research existing systems
- Modify existing systems
- Write computer software
- Test software
- Provide support and training
- Create operating manuals / documentation

This list describes some of the activities performed by software engineers. It is by no means exhaustive. Can you think of anymore activities to add to the list?



## HOW DO PROGRAMMERS WORK?

- Alone
- Pair programming
- Team work
- Virtual teams

Programmers work in a variety of ways. Depending on your corporate culture you may spend time as a developer working alone. Even if you work alone on certain coding tasks your work will normally form part of a larger team effort. Many organisations practise a technique known as pair programming. This technique involves two programmers working together actively on a task and taking turns to be the driver or the navigator. Some organisations have large non-co-located software teams that need to work together. There are various tools and techniques that organisations use to support these virtual teams.

Many software teams use a development methodology based on iterative development known as Agile.

# AGILE

- Group of software development methodologies
- Aligns development with evolving customer requirements
- Based on iterative development
- Collaboration within cross-functional teams is key
- Encourages frequent inspection and adaptation
- Aims to enable rapid delivery of high-quality software
- The most widely used agile framework is Scrum

Agile development refers to any development process that is aligned to the values and principles laid out within the Agile Manifesto.

The Agile Manifesto: <http://agilemanifesto.org/>

Agile software development methodologies are based on collaboration within disciplined, cross-functional teams that work alongside the customer to continuously gather requirements and prioritise functionality through iterative cycles often referred to as sprints. One of the key features of agile development is the inspection of how the team is working and adapting as necessary to improve the process. The aim of Agile is to enable the rapid delivery of high-quality software whilst reducing any unnecessary work.

There are various Agile frameworks such as Lean, Kanban, XP (Extreme Programming) and Scrum. The most widely used framework is Scrum.



SCRUM



<https://www.youtube.com/watch?v=2Vt7Ik8Ublw>

This video 'Scrum in under 5 minutes' is from Scrum Company.

## SCRUM ROLES

### Product Owner

The main project stakeholder responsible for giving direction to the team and liaising with the customer and other stakeholders to gather requirements.

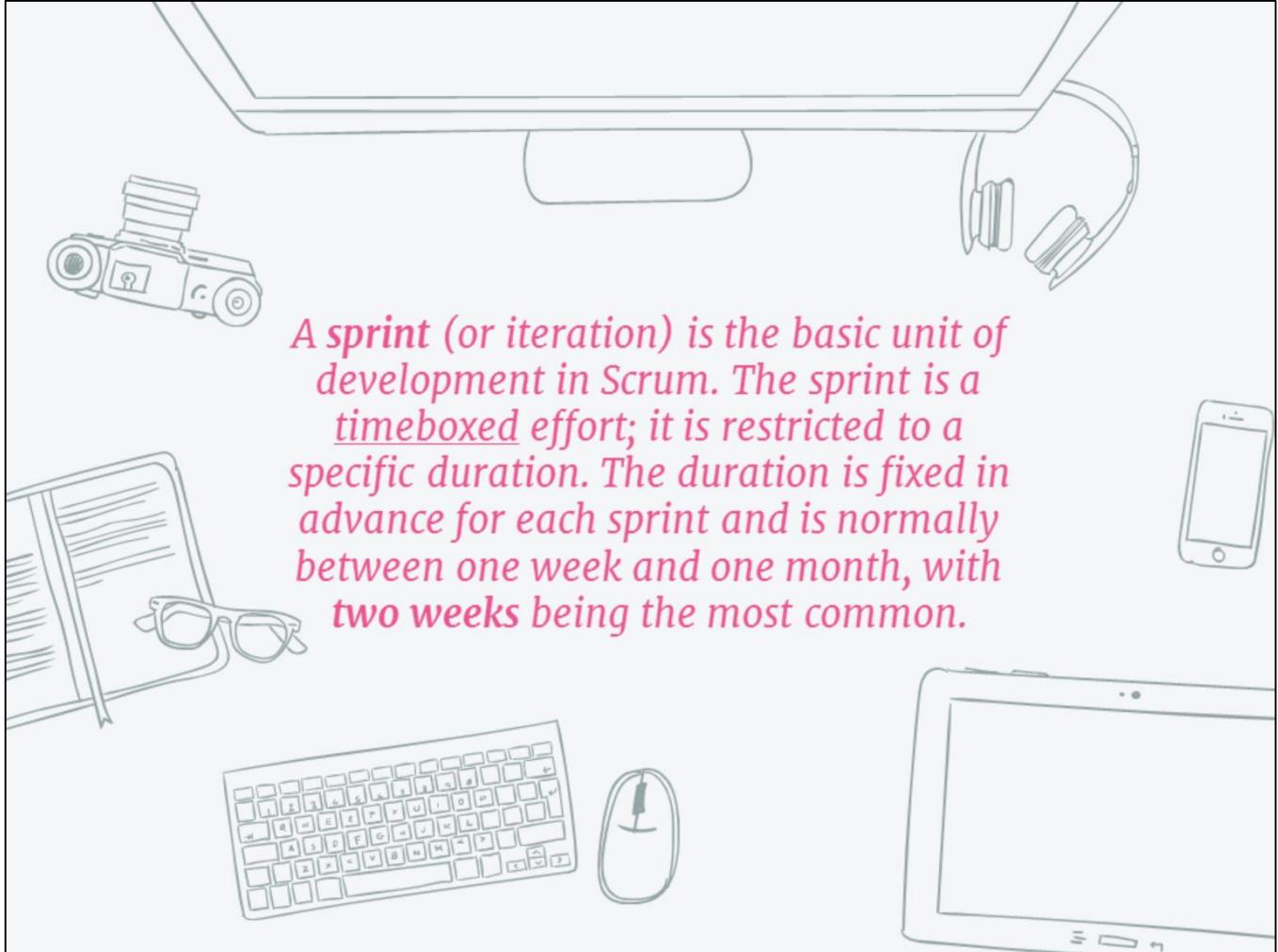
### Team Member

Each team member takes joint responsibility in completing the work they have collectively committed to complete during each sprint.

### Scrum Master

The team facilitator or coach who's main focus is on the project process and to aid in removing distractions that come from outside of the team. Sometimes referred to as a servant leader.

There are three roles within a scrum team: product owner, team member and scrum master.

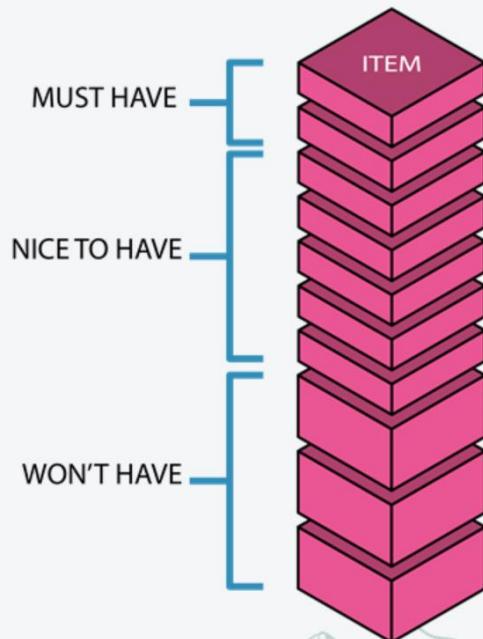


*A sprint (or iteration) is the basic unit of development in Scrum. The sprint is a timeboxed effort; it is restricted to a specific duration. The duration is fixed in advance for each sprint and is normally between one week and one month, with two weeks being the most common.*

## SCRUM ARTIFACTS: PRODUCT BACKLOG

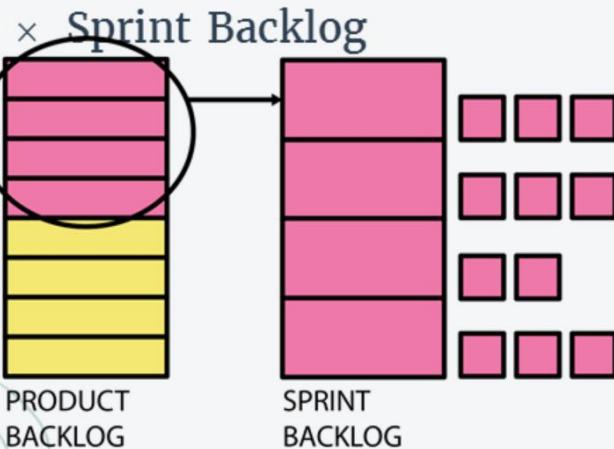
### Product Backlog

A to-do list of all the things that need to be done within the project. These items can have a technical nature or can be user-centric e.g. in the form of user stories.



Items higher up the prioritised list are broken down into detailed user stories. Items further down the list may be left as large, currently undetailed user stories, often referred to as epics. In order to prioritise the list some teams use the MOSCOW technique: Must Have, Should Have, Could Have, Won't Have.

## SCRUM ARTIFACTS: SPRINT BACKLOG



A list of tasks identified by the Scrum team about which functionality will be completed during the next sprint and the work needed to deliver that functionality into a “Done” state.

The Sprint Backlog is a list of tasks identified by the Scrum team about which functionality will be completed during the next sprint and the work needed to deliver that functionality into a “Done” state. During the sprint planning meeting, the team selects some number of product backlog items, usually in the form of user stories, and identifies the tasks necessary to complete each user story. The team also estimates the effort involved in each task to ensure they neither over or under-commit to work during the sprint. The Sprint Backlog makes visible all of the work that the scrum team identifies as necessary to meet the Sprint Goal.



## SCRUM ARTIFACTS: SCRUM BOARD



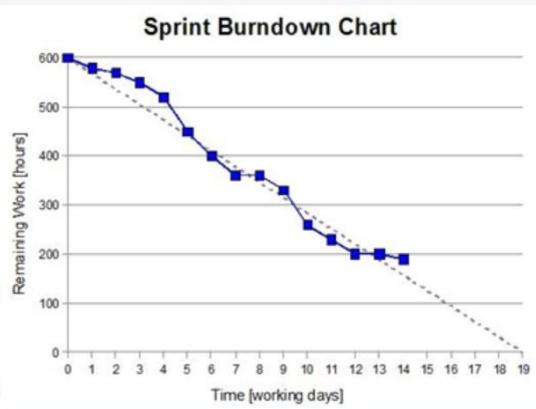
### Scrum Board

A visual display of the progress of the team during a sprint. It presents a snapshot of the current sprint backlog allowing everyone to see which tasks remain to be started, which are in progress and which are done.



In Scrum, the task board is a visual display of the progress of the Scrum team during a sprint. It presents a snapshot of the current sprint backlog, allowing everyone to see which tasks remain to be started, which are in progress and which are done. The board can take many physical and virtual forms, but it performs the same function regardless of how it looks.

## SCRUM ARTIFACTS: BURNDOWN CHART



### Burndown Chart

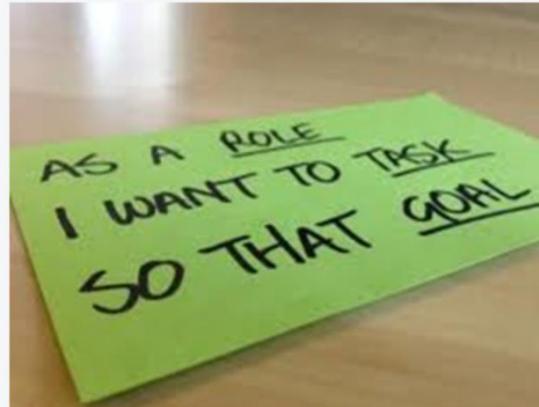
A graphical representation of work left to do versus time. It shows how quickly you are *burning* through your customer's user stories. It helps to predict when all the work will be completed.

A burn down chart is a graphical representation of work left to do versus time. The chart shows how quickly you and your team are burning through your customer's user stories. It shows the total effort against the amount of work you deliver each iteration. The outstanding work (or backlog) is often on the vertical axis, with time along the horizontal axis. It is a run chart of outstanding work and is useful for predicting when all of the work will be completed.

## AGILE: USER STORIES

### User Story

A short, simple description of a software feature from an end-user perspective. The story describes the type of user, what they want and why.



A user story is a tool used in Agile software development to capture a short, simple description of a software feature from an end-user perspective. The user story describes the type of user, what they want and why. A user story helps to create a simplified description of a requirement.

They typically follow a simple template:

As a < type of user >, I want < some capability > so that < some goal >.

This is sometimes expressed more simply as: as who, I want what, so that why.

User stories are often written on index cards or sticky notes and arranged on walls or tables to facilitate planning and discussion. As such, they strongly shift the focus from writing about features to discussing them. In fact, these discussions are more important than whatever text is written.

# SCRUM MEETINGS / RITUALS

## Sprint Planning



What is achievable within this sprint.  
What work does the scrum team commit to.

## Sprint Review



Deliver results, discuss or showcase completed work and receive feedback.

## Daily Stand-up



What did you do yesterday?  
What are you going to do today?  
What impedes you?

## Sprint Retrospective



Reflect on sprint and look for team improvements.  
What went well / badly/ needs improvement?

Scrum has many different meetings, which some teams call rituals. The sprint planning meeting occurs at the start of a sprint, the daily stand-up (or scrum) occurs every day during the sprint and the sprint review and sprint retrospectives happen at the end of a sprint. The sprint planning meeting is normally attended by the whole scrum team but often only the development team and scrum master attend the daily stand-ups.

The sprint review is a time to inspect and adapt the product and therefore external stakeholders attend this meeting. These stakeholders could be subject matter experts, other corporate teams, legal or regulatory bodies or business executives who have a stake in the project. The sprint retrospective is an opportunity to inspect and adapt the process and is therefore normally attended by only the scrum team (product owner, scrum master and development team). During the retrospective, teams are free to examine what's happening, analyse the way they work, identify ways to improve, and make plans to implement these improvements. Anything that affects how the team creates the product is open to scrutiny and discussion, including processes, practices, communication, environment, artifacts and tools.

# PAIR PROGRAMMING



Pair programming is an agile software development technique in which two programmers work together at one workstation. One, the driver, writes code while the other, the observer or navigator, reviews each line of code as it is typed in. The two programmers switch roles frequently.

Pair programming is an important technique for quickly developing higher quality code, while also reducing risk and spreading knowledge in an organisation. With pair programming, two software developers work on one computer, collaborating on the same design, algorithm, code, or test.

It is counter intuitive, but two people working at a single computer will add as much functionality as two working separately, except that it will be much higher in quality. With increased quality comes big savings later in the project.



## ACTIVITY

As a group, discuss the skills and personality traits you feel are essential for a software engineer to succeed

The purpose of this exercise is to get you thinking about the skills and traits a successful software developer needs.

As you are discussing the characteristics think about which of these skills you possess and try to identify how your own attributes and traits will be beneficial within a programming role.



# Any questions?

# THE INTERNET AND ITS TECHNOLOGIES

## CHAPTER OVERVIEW

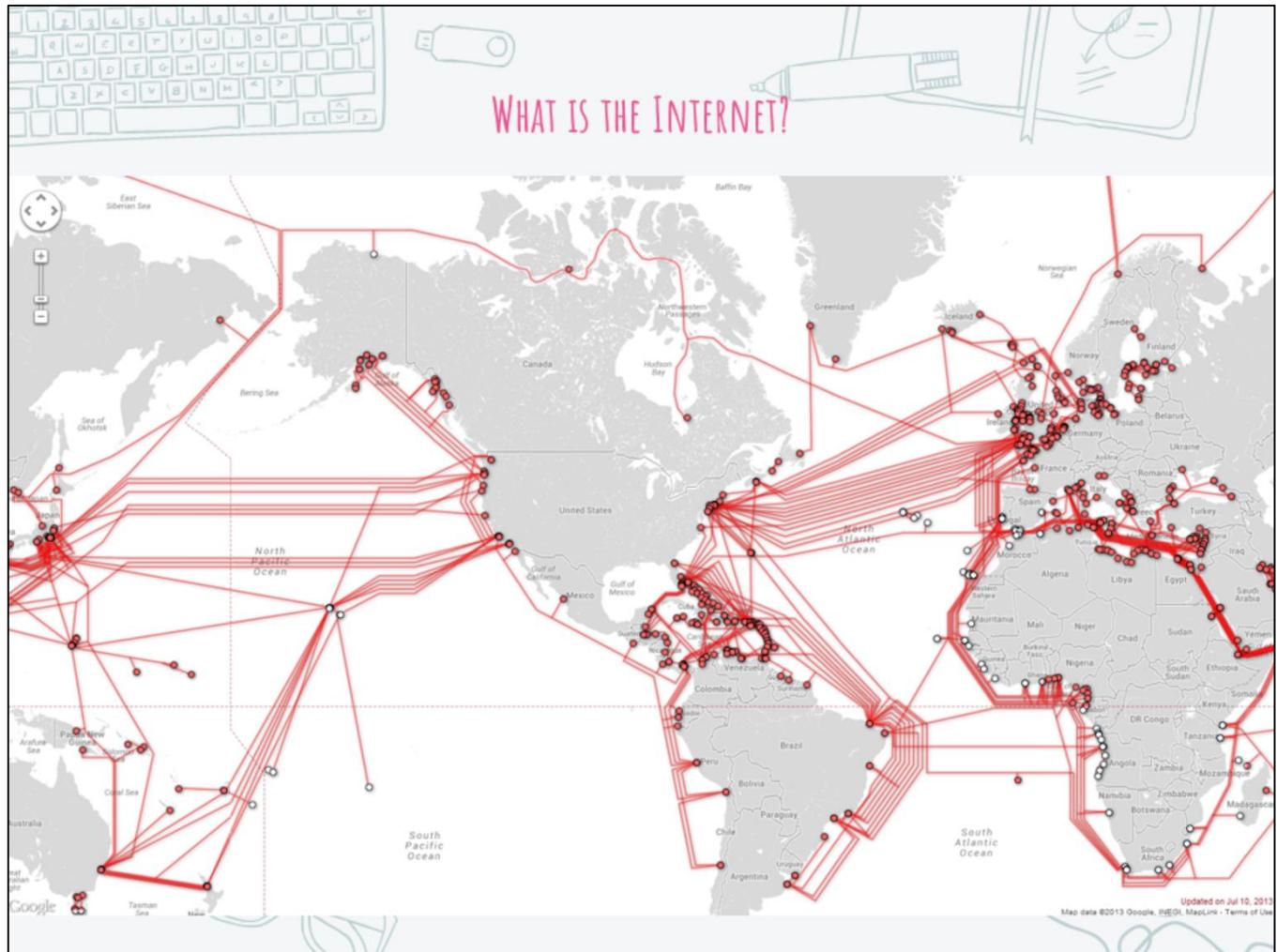
- What is the internet?
- Clients, Servers and HTTP
- The Domain Name System
- HTTP Request Methods
- HTTP Requests and Responses
- Web Architecture and PHP

### Exercise

Group research into key internet technologies

By the end of this chapter you should have a better understanding of the term "the internet", what kinds of machines connect to each other and why they do.

There are many "levels" of view – at the lowest level there are zeros and ones sent across wires. However, web development requires only a broad perspective: a little about the language of web communication and a little about each machine's responsibilities.



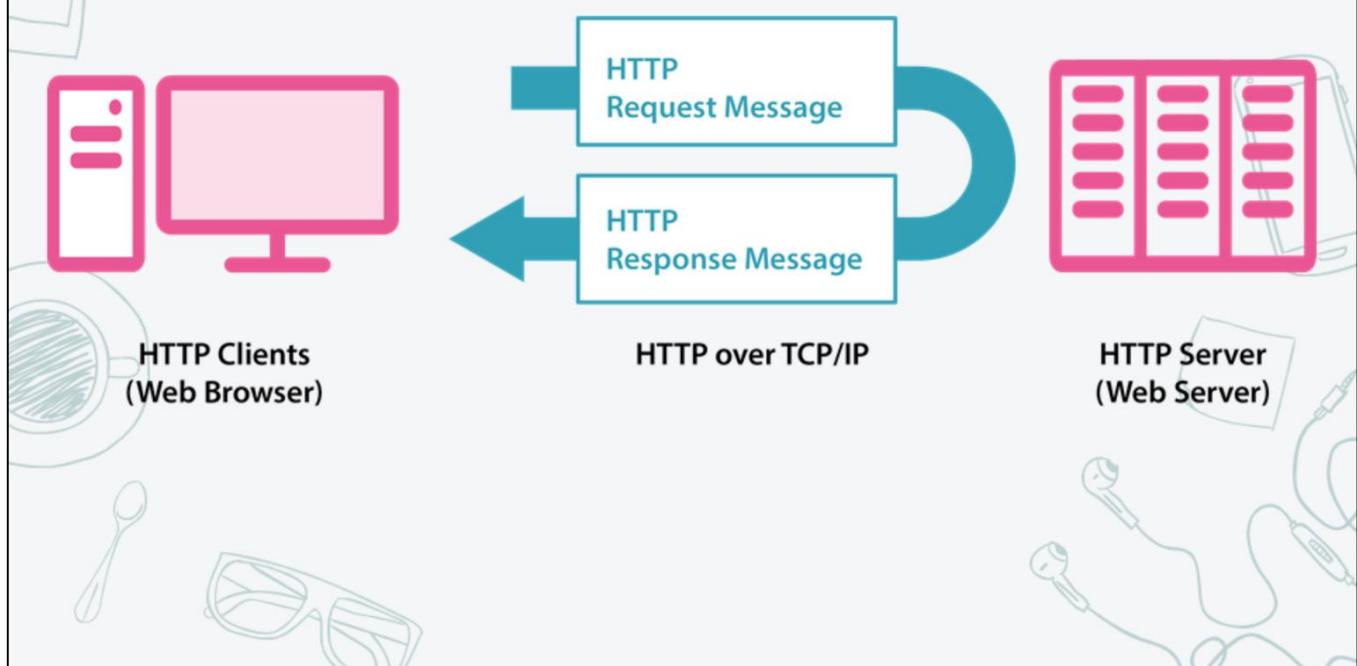
Not all networks of connected machines are "The Internet". "The Internet" refers to machines which meet three conditions: they are interconnected, they talk using the hypertext transfer protocol (HTTP) and they are all connected (directly or indirectly) to "The Internet Backbone". This backbone is a series of large connection points around the world which coordinate a lot of communication.

Intranets are networks of machines, talking to each other with HTTP but not connected to the outside world. They are private and can only be accessed by machines within the network. To gain access to an intranet site using a web browser you have to be plugged into that particular network.

Internet networks are filled with machines each which serve a different purpose. Some are firewalls, some are routers, some are nameservers. Firewalls determine whether or not communication should pass further along the network. Routers redirect communication traffic from one machine to another. Nameservers associate unique resource locators (URLs) with specific machine addresses (IP addresses).

The term client is very general and simply refers to whomever initiates a request. This request is made to another machine which is called a server. Web servers serve up the data requested back to the client that initiated the request. Often this data is in the form of a web page and the client that initiated the request is a web browser.

# HTTP: THE HYPERTEXT TRANSFER PROTOCOL



Imagine a network of only two computers. In this network one computer will issue requests to another, in particular, it will ask the other computer for a website.

The computer which asks for the website is called the client and the other computer which provides the website is called the server. These terms apply exactly as they would at a coffee shop: the customer (client) sends an order to the barista (server) and hopefully they deliver it.

In coffee shops clients and servers speak the same language, in the United Kingdom that's typically English. On the internet however there is a much more precise and formally specified language – or "protocol" – which has its own commands. This protocol is HTTP: The HyperText Transfer Protocol.

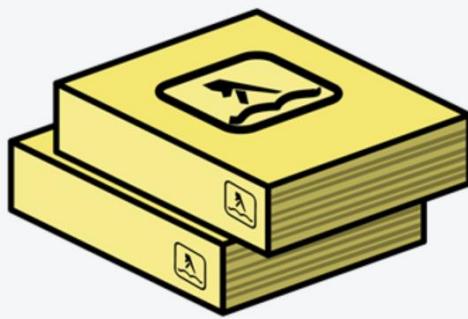
Requests and responses sent using HTTP have to be converted for transmission over a real, physical wire. The Transmission Control Protocol (TCP) defines how the HTTP commands will be structured for transmission.

The internet then is several protocols together: TCP/IP (a networking protocol which uses "Internet Protocol" addresses) and HTTP (a language for how web clients and servers communicate).

## THE DOMAIN NAME SYSTEM

### Domain Name System

The DNS is a central part of the Internet, providing a way to match names (e.g. www.sky.com) to numbers (the address for the webserver that hosts the website). Anything connected to the Internet - laptops, tablets, mobile phones, websites - has an Internet Protocol (IP) address made up of numbers.



Internal networks of a few machines are known as intranets. You can use IP addresses to access them. In the wider internet you do not often use IP addresses as you would not be able to remember all of the addresses you need.

When you enter www.sky.com in a browser the webserver asks a DNS (Domain Name System) server for the IP address of that domain.

DNS servers are typically provided by your Internet Service Provider (ISP) and provided when you connect to the internet.

The Domain Name System makes it easier to use the internet because it maps friendly URLs to an IP address which locates a machine connected to the internet. It is therefore a "lookup" system that relates domains to IP addresses for particular machines.



Go to a command prompt / terminal and type the following command:

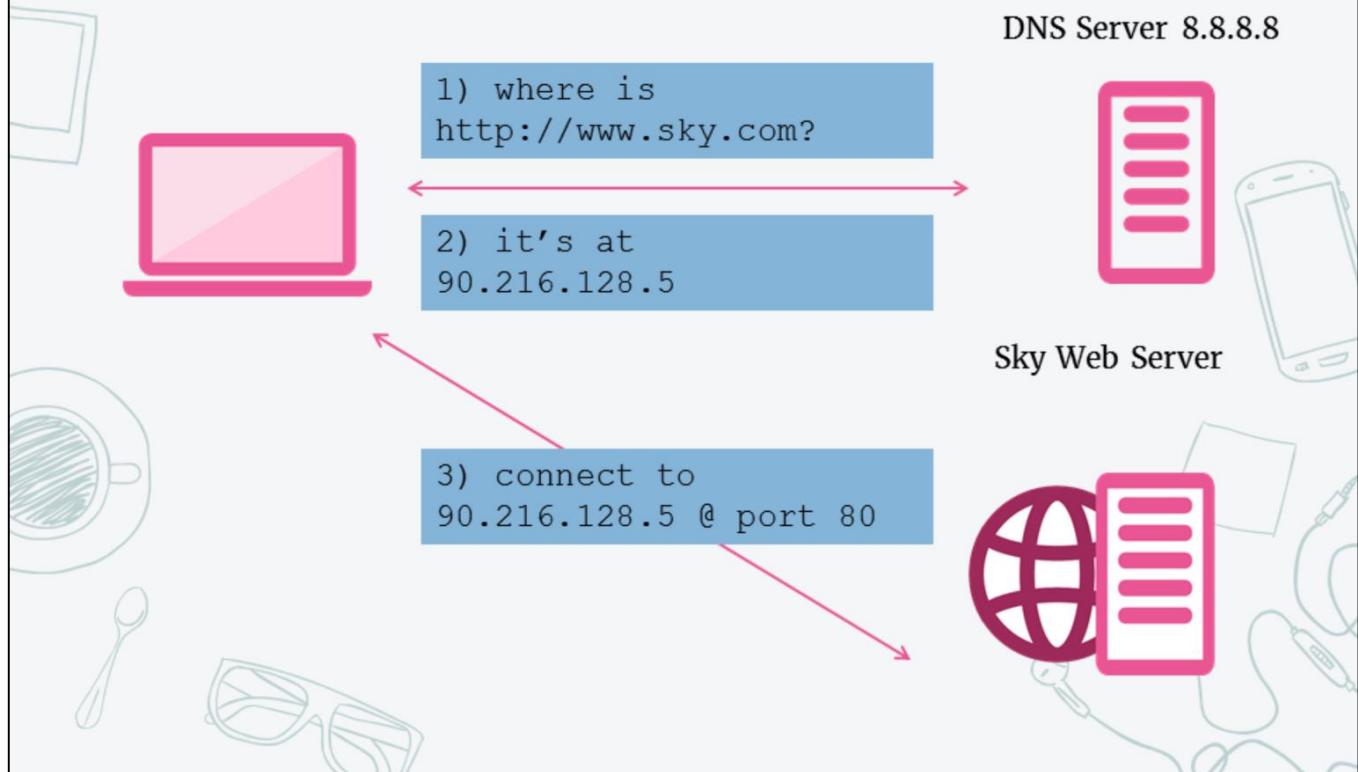
```
ping sky.com
```

Make a note of the internet protocol address. It will be in the format of 4 'octets' such as xxx.xxx.xxx.xxx. An example IP address is 216.58.214.14.

Then go to your browser and in the address bar type in the IP address you just discovered with the help of DNS:

```
http://XXX.XXX.XXX.XXX
```

## DNS IN ACTION



This diagram describes the typical situation for a browser requesting a site. The user types in a URL ([www.sky.com](http://www.sky.com)) and a request is sent – but to where? The browser doesn't yet know to which machine sky.com corresponds.

It first sends a request to a DNS server which replies with the IP address of the machine which serves the website, in this case, the IP address of sky.com.

Operating systems and browsers typically cache these responses for later use so browsers will typically not need to issue many DNS requests.

When the address is known the browser can send a request to the actual server of interest.

The default port on which a webserver listens for requests is port 80. Sometimes an administrator will change this to port 8080 if port 80 is already in use by another program. In programming, a port is a connection point and the way a client program specifies which server program it wishes to target. Port numbers range from 0 to 65535. Ports 0 to 1024 are reserved for use by certain privileged services.

## HTTP REQUEST METHODS

### HTTP GET

Requests data from a specified resource. Any data sent with the request is visible within the URL. Do not use with sensitive data. Only use to retrieve data. The request can be cached and bookmarked and has some restrictions on data length.



### HTTP POST

Submits data to be processed to a specified resource. Any data is sent in the message body not the URL. The request is never cached, cannot be bookmarked and has no restrictions on data length.



The two commonly used methods for a request-response between a client and server are: GET and POST.

GET - Requests data from a specified resource.

POST - Submits data to be processed to a specified resource.

#### The GET Method

Note that the query string (name/value pairs) is sent in the URL of a GET request:

/test/demo\_form.php?name1=value1&name2=value2

#### The POST Method

Note that the query string (name/value pairs) is sent in the HTTP message body of a POST request:

POST /test/demo\_form.php HTTP/1.1

Host: sky.com

name1=value1&name2=value2

## HTTP REQUESTS AND RESPONSES

- A client first connects to a server <http://www.sky.com> using its IP address at port 80. The browser sends this request to the server machine

```
GET /folder/file.html HTTP/1.1
```

- The web server software on this machine (Apache / IIS) parses this request and returns a response

```
HTTP/1.1 201 OK
Date: Thu, 31 Mar 2016 23:59:59 GMT
Content-Type: text/html
Content-Length: 1354
<html><body><h1>The File!</h1></body></html>
```

- The web browser parses this response and renders a page for you to view

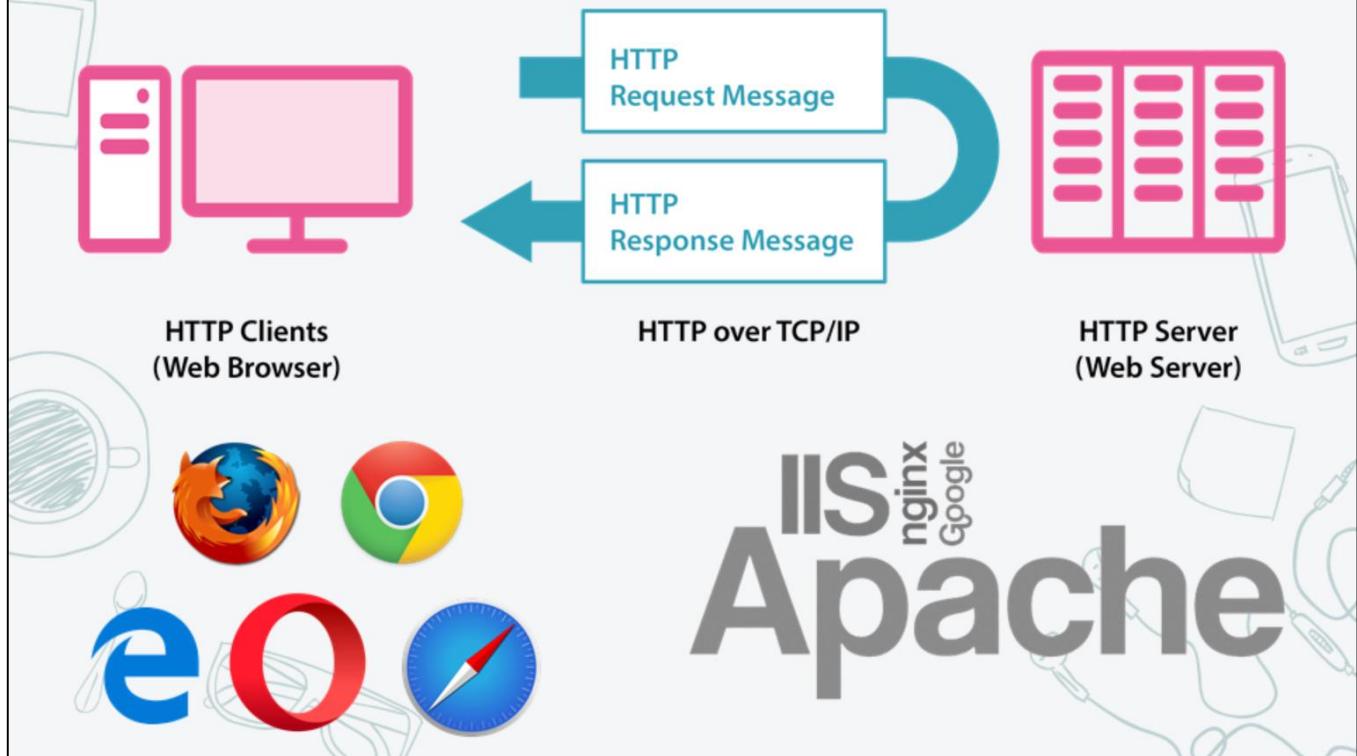
HTTP is an ordinary plain text protocol, so it's quite human readable. The first request a browser sends to a server is typically a GET request which means "serve me this particular resource". This resource is often data or a webpage. GET requests contain the URL of the resource the client is requesting.

The communication from the client to the server is called a request. The communication that then follows from the server to the client is called a response.

HTTP Requests and Responses have a common structure: a type, some key-value pairs of metadata called "headers" and a request body which can be empty.

HTTP responses have a similar structure to HTTP requests but will include some different information. The common HTTP response for a GET request is a 201 OK response. which includes the HTML (CSS, JavaScript, etc.) as its response body and headers describing the content type and how many bytes it is in size.

## WEB CLIENTS AND WEB SERVERS



You probably know some web client software: for example Chrome, Firefox, Safari, Internet Explorer and Opera. The purpose of this software is to send HTTP requests and to display the response. If the response is a web page it is the browser's responsibility to parse the page and display its contents to the user.

Web server software is less well known, but the most commonly used product is Apache. Others include Microsoft's Internet Information Server (IIS), Google Web Server (GWS) and nginx (pronounced Engine X). Apache is a web server program that receives HTTP requests and determines what HTTP response ought to be sent.

Apache lives on machines we call "web servers" and browsers such as Chrome, live on machines we call clients.

## WHAT IS PHP?

- PHP is an acronym for PHP Hypertext Pre-processor
- Original acronym was Personal Home Page
- PHP is a widely used, open source, server-side scripting language
- PHP is free to download and use
- PHP is designed to work with HTTP and integrates with many web servers
- PHP provides many useful features for creating and building websites
- PHP lives on the server and works with the web server to supply information to clients often in the form of web pages

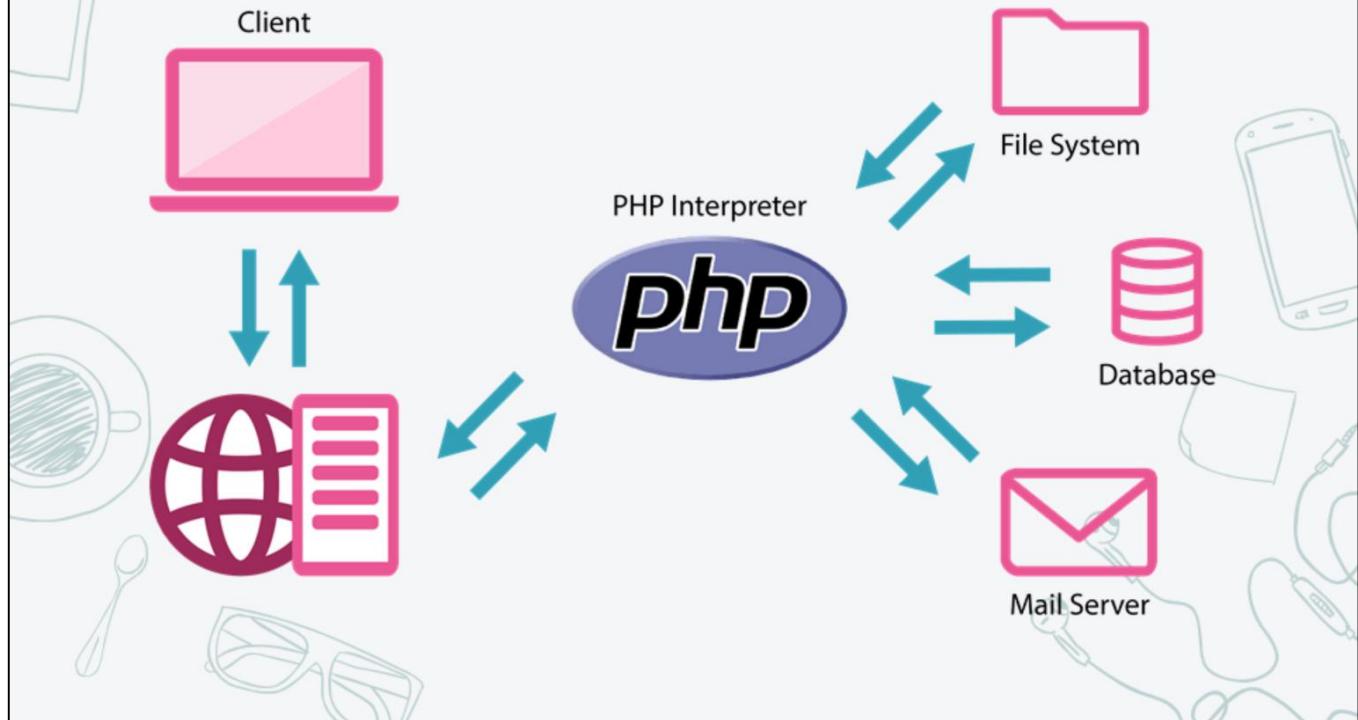


PHP is server-side scripting language. It sits on the web server and interprets and runs PHP scripts in response to requests from clients.

The PHP interpreter assists the web server. You write your web application using the PHP programming language and the webserver will ask the PHP interpreter to follow the instructions you write and collect their output. This output will then be sent back to the client.

PHP is most often used to dynamically create HTTP responses, for example web pages, that web servers can send back to clients.

## WEB ARCHITECTURE AND PHP



This diagram shows the anatomy of a request and response and where PHP sits within the architecture. On a server machine there will classically be installed four pieces of technology: an operating system, a web server, a programming language interpreter and a database. In the PHP world these four are often denoted "LAMP": Linux, Apache, PHP, MySQL.

When a client sends a request to a server, the operating system receives the bytes sent across the network. The operating system then passes these on to the web server (e.g. Apache) to handle.

If it is a request for a PHP website, Apache will ask the PHP interpreter to run the commands associated with this request. PHP may then itself query a database to gather or save data relevant to the web site. PHP may also process files from the file system such as reading or writing files and PHP may interact with a mail server to send an email. When PHP is done it sends its output back to the web server which sends it (via the operating system) across the network back to the client.

The request starts at the client machine, flows through the web server's operating system, through PHP which may request services from other parts of the system and then PHP formulates a response which it sends back to the web server which in turn sends the response to the client where the browser software will parse and display the results.

## CLIENT SERVER COMMUNICATION WITH PHP

1. The web browser sends an HTTP request to the web server.
2. Because the request is for a PHP page the web server asks PHP to fulfil the request.
3. PHP may interact with other services such as a database, mail server or the file system or external sources such as a credit card processing system.
4. PHP sends the response back to the web server which sends the HTTP response back to the client.
5. The client's browser parses the response and displays any results.

The client-server architecture with one browser, one server, one request and one response is simplified but actually complete enough to describe the majority of simple and personal websites.

In a large enterprise environment the architecture could be spread across multiple different servers. You could place a database on its own machine, you could even place PHP on its own machine.

In the most high demand networks a single machine, known as a Load Balancer, distributes HTTP requests across a network of other machines, so that a single machine rarely gets overloaded.

## WHY PHP?

- Easy to learn
- Available from nearly every hosting company
- Designed for the web
- Out-of-the-box support for databases and other typical website software
- Large number of libraries and tools for web development
- PHP is one of the most popular programming languages of all time
  - It powers the web's biggest blogging system -Wordpress
  - It is estimated that approximately 80% of the web is powered by PHP including Facebook, Yahoo, .gov.uk

You only need a few simple features of the PHP language to create a basic web site. With other languages, such as Java or C#, there are a lot of advanced concepts which appear immediately when you try to write some simple software.

PHP used to be limited to a small feature set, however, it has grown considerably as a language and now rivals many others in terms of the complexity of programs which can be written using it.

Even though PHP can be applied to large scale, professional environments, its hobbyist roots remain and therefore make it easy enough to be a beginner's first server-side language.

PHP sits quite naturally next to web servers: it handles request and responses and begins and ends its "life" just as a web server expects: you start, do your job, finish and send back the response.

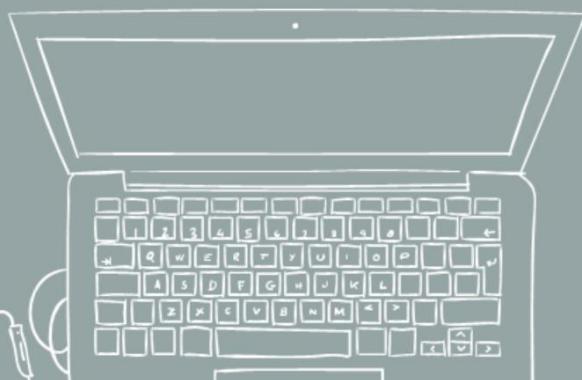
Also, the language is filled with libraries, tools and technologies that work well with the web (HTTP and HTML) and make it a natural fit for web development.



# Any questions?



## EXERCISE 3A



# AN INTRODUCTION TO PHP

## CHAPTER OVERVIEW

- An Introduction to PHP
- Online Documentation
- PHP Files
- PHP Basics
- An Introduction to NetBeans
- Configuring PHP

### Exercise

- Create a NetBeans project
- Run the Apache Web Server
- Run different PHP files
- Modify PHP files and compare their output
- Change the project run configuration

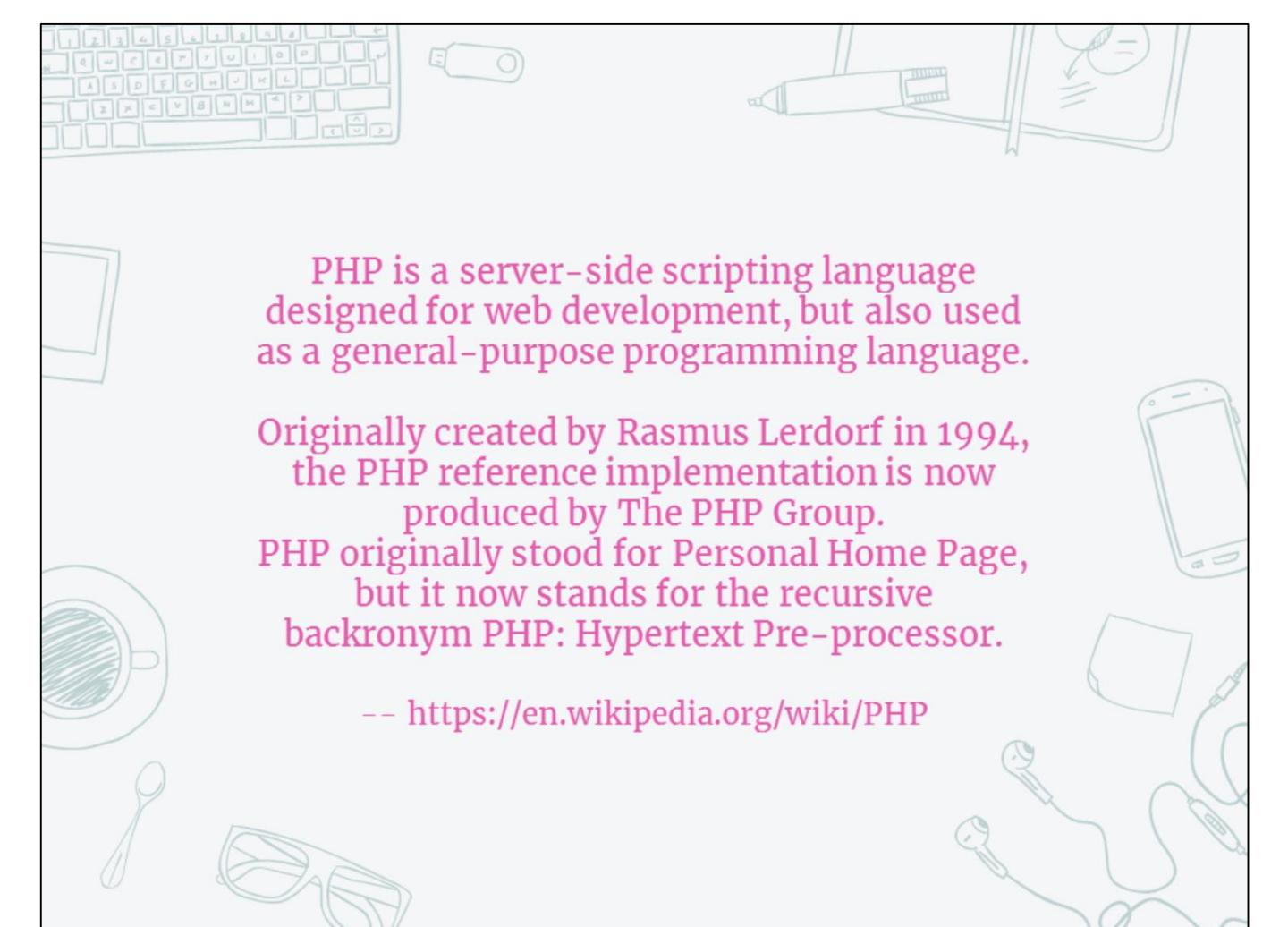
The goal of this chapter is to get you comfortable looking at PHP files and running them to see their output. You will not go into the details regarding PHP's syntax as you will explore this in subsequent chapters.

In the introduction to the internet the PHP interpreter was conceptually placed on the server side. This indicates a lot about the PHP language: it is primarily designed for assisting web servers in their goals of delivering appropriate HTTP responses to clients. It is, in other words, a language designed for building websites.

If you already have experience programming in other languages, PHP might seem a little strange. It mostly proceeds from the view that everything is text, and that text can be used quite directly in all sorts of ways. That, for example, '0' is false or that '12abcde' can be treated just like the number 12. This is because PHP begins and ends with text: the text of HTTP requests. PHP is designed to understand the HTTP message format.

This chapter focuses on some PHP background and the program that begins the whole process of learning and developing with PHP: the PHP executable. In Microsoft Windows you call this "php.exe".

The PHP executable is downloaded from PHP.net and includes several subprograms, or features. The primary feature of php.exe is the PHP interpreter which accepts instructions in the language called PHP.



**PHP is a server-side scripting language designed for web development, but also used as a general-purpose programming language.**

Originally created by Rasmus Lerdorf in 1994, the PHP reference implementation is now produced by The PHP Group.

PHP originally stood for Personal Home Page, but it now stands for the recursive backronym PHP: Hypertext Pre-processor.

-- <https://en.wikipedia.org/wiki/PHP>

Take a look at the claims of this quote:

"PHP is server-side" – the interpreter and your PHP files will be sitting on a webserver waiting for a request to come through.

"scripting language" – the word scripting tends to refer to languages which are interpreted rather than compiled. Compiled languages first send their code to a compiler which makes a program file, this program file is then the one which you run. With PHP you just run the code directly, no extra hassle.

"web development" – the kind of programming which concerns building websites. This is actually quite specialised, as far as programming goes. Web development is often dealing with data that has the same kind of simple, relational structure. It focuses on HTML, HTTP, validating forms and user input. Plus cookies, sessions, AJAX and techniques which do not reappear in any other programming area. PHP is a popular language, in part, because it specialised in this area.

"also used as a general-purpose programming language" – the phrase also used is quite appropriate: PHP isn't designed for general use and lacks some key features that might be expected in traditional languages (such as threading).

## HISTORY OF PHP

Version	RELEASE DATE	SUPPORTED UNTIL
1.0 – 3.0	June 1995	October 2000
4.0 – 5.0	May 2000	September 2005
5.0 – 5.2	July 2004	January 2011
5.2 – 6.0	June 2009	December 2018
7.0 +	December 2015	December 2018

PHP was originally released in June 1995 and has evolved significantly with each new version. Note: the release of version 6.0 was abandoned.

During this course you will be using PHP version 7.0.

## ADVANTAGES OF USING PHP FOR WEB DEVELOPMENT

- The PHP language is comparatively simple and can be learnt piece by piece
- Hobbyists can be productive with simple code yet professionals can write powerful applications
- PHP runs scripts for the life of an HTTP request: it "dies" quickly which makes it memory efficient and scalable
- PHP provides a wealth of libraries specifically designed for the web
- PHP parses HTTP Requests into data and makes it easily available for use



There are several important features which make PHP suitable for web development.

The mascot of the PHP project is the elePHPant, a blue elephant with the PHP logo on its side.

→ C i php.net/manual/en/getting-started.php

**php** Downloads Documentation Get Involved Help

PHP 7.1.8 Released

PHP Manual

# ONLINE DOCUMENTATION: PHP.NET

Change language: English Ed

## Getting Started

- [Introduction](#)
  - [What is PHP?](#)
  - [What can PHP do?](#)
- [A simple tutorial](#)
  - [What do I need?](#)
  - [Your first PHP-enabled page](#)
  - [Something Useful](#)
  - [Dealing with Forms](#)
  - [Using old code with new versions of PHP](#)
  - [What's next?](#)

### User Contributed Notes

There are no user contributed notes for this page.

Becoming familiar with PHP's online documentation, or manual, is critical and you should refer to it constantly throughout the course. The manual can be found at:

<http://php.net>

The PHP language has many idiosyncrasies and even experienced developers frequently use the documentation as a resource. Code examples are provided along with commentary and definitions.

## PHP FILES

### PHP Files

All PHP files start with  
`<?php`

If the file is entirely PHP  
do not use an end tag

If the file contains other  
content surrounding the  
php code use an end tag  
`?>`

PHP short output tags

`<?= ?>`

```
<?php
```

```
echo "This is a sample php file.";
```

```
<?php
```

```
echo "This uses an end tag.";
```

```
?>
```

```
<?= "These are short output tags.";
```

```
?>
```

What do PHP files look like?

There are several ways a PHP file might be structured. You can use the standard start tag `<?php` and the end tag `?>`. Or you can use a start tag without an end tag. You can also use what are known as short tags which start with `<?=` and end with `?>`. To understand why you might use the different formats, let's discuss how the PHP interpreter works.

Wherever it sees either `<?php` or `<?=` it assumes there will be PHP code. Whenever it sees `?>` it assumes the PHP code is ending.

The interpreter will not touch any non-PHP code. For example, if your file is all HTML and you send it to the interpreter you will get all the same HTML back out.

The PHP short tags `<?= ?>` are special. They are specifically output tags; anything in between them will be output. Whereas the full tags `<?php ?>` can include instructions that have no output, or which you do not wish to output.

Files which are entirely made up of PHP code should begin only with `<?php` and not include a final `?>`. This is incase there is any hidden whitespace after `?>` which would be returned by the interpreter as valid output. Often times you don't want that whitespace in your HTTP response. Whitespace within PHP tags is ignored, whitespace outside of them isn't.

## PHP BASICS

All PHP statements end with a semicolon ;

The echo language construct outputs one or more strings (text).

PHP contains useful **functions** (named groups of instructions) such as date( )

PHP short output tags <?= ?> are a shorthand for the echo language construct.

```
The date today is <?php echo date('d-m-Y'); ?>  
The date today is <?= date('d-m-Y'); ?>
```

The examples above use the date() function. A function is a named group of instructions. You will learn more about functions in a later chapter.

The second example uses the PHP short tags as an equivalent to the standard start tag plus the echo language construct.

All PHP statements end with a semicolon (;).

## HOW TO CREATE PHP FILES

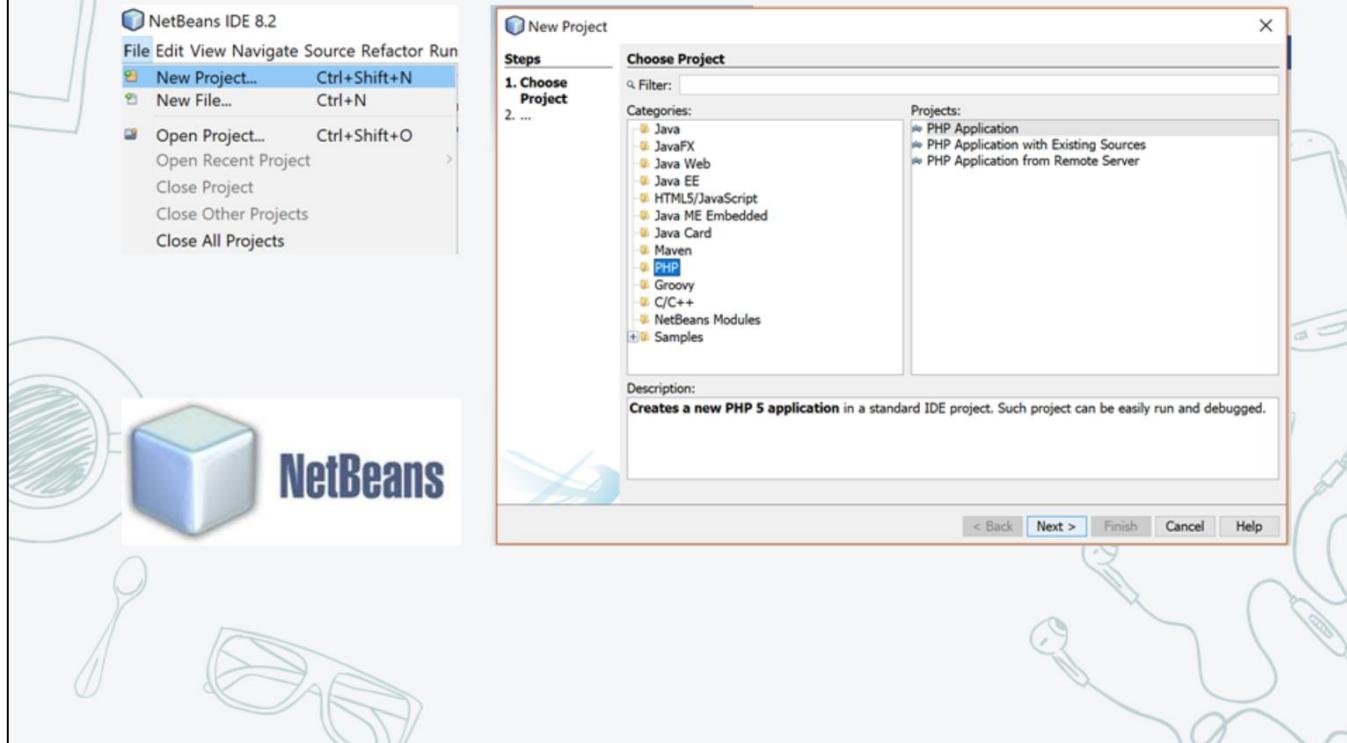
Software developers use a tool called an Integrated Development Environment (IDE):

An **integrated development environment (IDE)** is a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of a **source code editor**, build automation tools and a **debugger**. Most modern IDEs have **intelligent code completion**.

[https://en.wikipedia.org/wiki/  
Integrated\\_development\\_environment](https://en.wikipedia.org/wiki/Integrated_development_environment)

You could use a simple text editor such as Notepad to create your code files. However, an Integrated Development Environment (IDE) will give you many productivity gains, such as automatic code completion and error detection.

# NETBEANS: CREATE A PROJECT



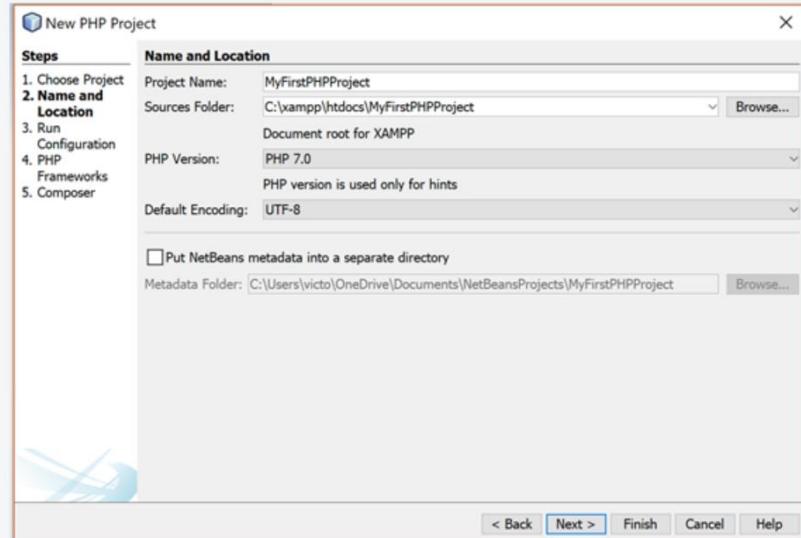
You will be using the NetBeans IDE during this course. It is a free and open source integrated development environment for application development on Windows, Mac, Linux, and Solaris operating systems.

The IDE simplifies the development of web, enterprise, desktop, and mobile applications that use the Java and HTML5 platforms. The IDE also offers support for the development of PHP and C/C++ applications.

NetBeans uses the popular concept of projects to organise code files. Use the File -> New Project menu option to create a new project and select PHP from the Categories list. You will typically use the PHP Application project type.

Note: ignore the reference to PHP 5 in the dialog box. You can select the version of PHP on the next screen.

## NETBEANS: NAME A PROJECT



Type in a name for your project (but don't use spaces) and check you are using PHP 7.0 as the PHP version. To run a PHP web application you will need a web server to host and serve up your PHP files. As part of the course setup you installed XAMPP (pronounced 'Zamp') which includes an Apache server. The XAMPP Apache server expects its files to be stored in the `htdocs` folder of the xampp installation directory.

On Windows this is typically:

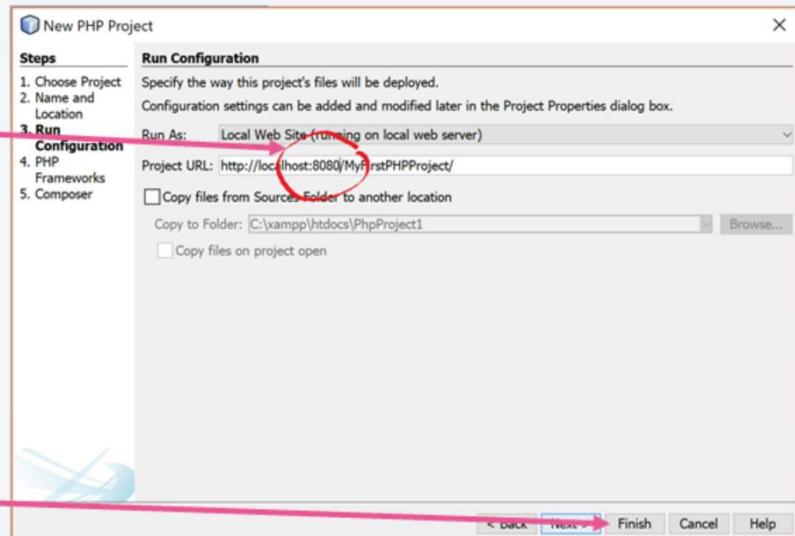
`C:\xampp\htdocs\`

On a Mac this is typically:

`/Applications/XAMPP/htdocs`

# NETBEANS: PROJECT CONFIGURATION

Check your port number is correct:



Choose Finish:

You will not be using any PHP frameworks or composers on this course so choose Finish after you have verified you have specified the correct port number. If your Apache server is running on the default port (Port 80) your Project URL will be:

<http://localhost/MyFirstPHPProject>

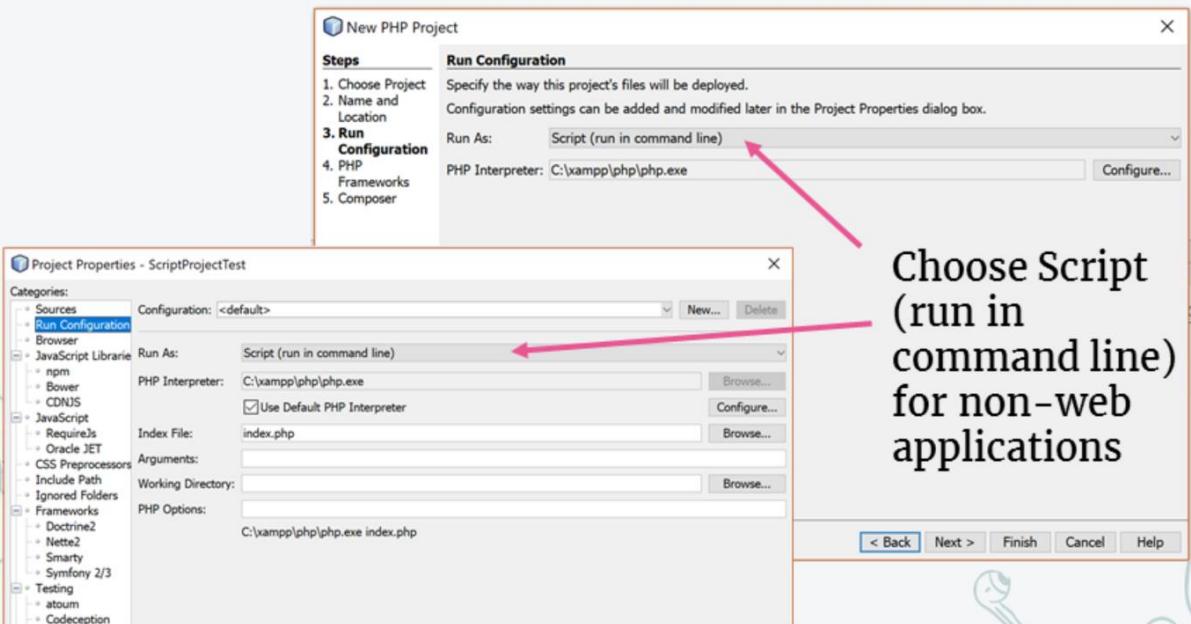
If your Apache server is running under a different port (for example Port 8080) your Project URL will be:

<http://localhost:8080/MyFirstPHPProject>

If you are creating a command line / terminal application you will have to change the Run As value to: Script (run in command line) and choose an appropriate path to the PHP Interpreter. On Windows this is typically

C:\xampp\php\php.exe

# NETBEANS: PROJECT CONFIGURATION COMMAND LINE APPLICATIONS



Choose Script  
(run in  
command line)  
for non-web  
applications

If you want to create a non-web project such as a command line / terminal application that does not run in the browser then change the configuration. You can do this in the New PHP Project wizard Run Configuration screen or after a project has been created by right-clicking on the project name and choosing Properties -> Run Configuration.

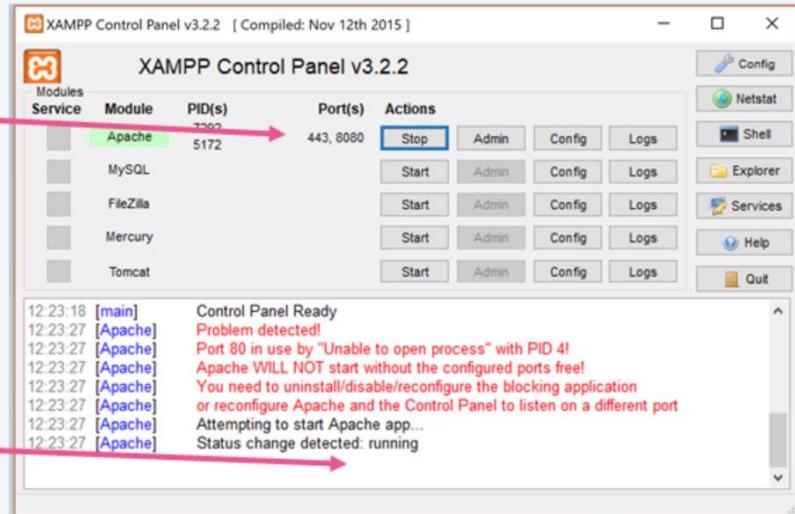
Non-web applications will output their content to the NetBeans Output window rather than to a browser.

For command line applications on a Mac you have already specified where your installation of PHP resides so all you need to specify is:

"Run As - > Script( run in command line )"

# STARTING YOUR APACHE WEB SERVER

Your port number  
is the second  
number here:



Confirm the  
status is  
running:

To run a PHP web project your Apache Web Server must be running. Launch the XAMPP control panel and ensure the status of the server is running. You can also verify the port number the server is using.

Note: PHP also has its own built-in web server that can be started from a command line / terminal using the following command:

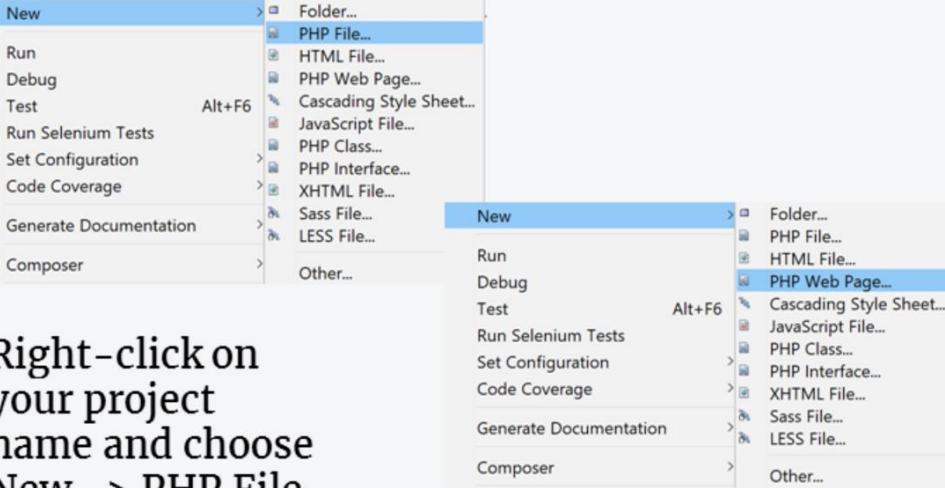
```
php -S localhost:8080
```

You only need to pass the host and port number if you are using a non-default port.

You will be using the Apache server that is part of your XAMPP installation rather than the built-in PHP web server.

To locate the XAMPP control panel on a Mac press **cmd+space** and type "**manager-osx**". It is located in /Applications/XAMPP.

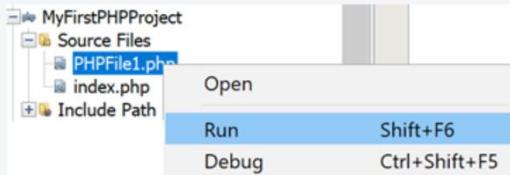
## NETBEANS: ADD FILES TO A PROJECT



Right-click on your project name and choose New -> PHP File / PHP Web Page.

To add files to your project right-click on the project name and choose New -> PHP File for command line projects or New -> PHP Web Page for web-based projects. The PHP Web Page will include HyperText Markup Language (HTML) that can be interpreted by your browser. You will learn more about HTML in a later chapter.

## NETBEANS: RUN YOUR FILE / PROJECT



To run a specific file right-click on your file and choose **Run** or use the keyboard shortcut **Shift + F6**.



To run the default file for your project choose the **Run Project** button or use keyboard shortcut **F6**.

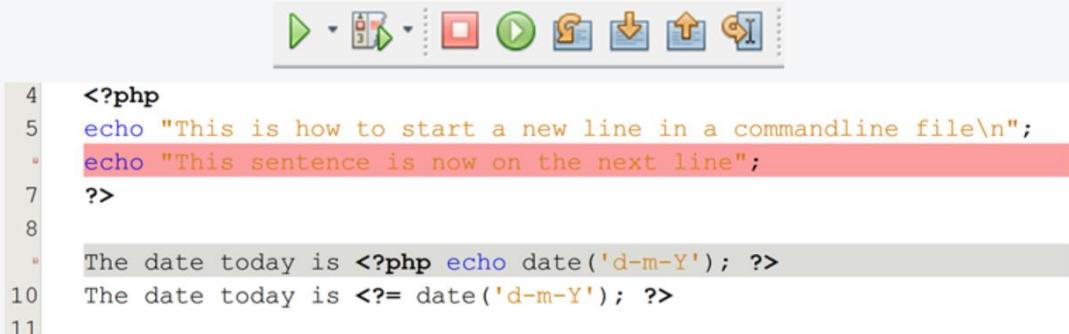
To run the default file for your project choose the Run Project button or on a Windows machine use keyboard shortcut F6. The default file is normally set as index.php but can be changed in your run configuration.

To run a specific file, right-click on your file and choose Run or use the keyboard shortcut Shift + F6.

On the NetBeans toolbar you can choose which browser the file will run in for each project or you can configure the setting globally by using the Tools -> Options ->General tab, and configuring the Web Browser setting. It is recommended you use Chrome as your development browser as it offers developer tools to help with the client-side aspects of your webpages.

For information on NetBeans shortcuts for use on a Mac see: <https://netbeans.org/kb/articles/mac.html>

## NETBEANS: DEBUGGING



```
4 <?php
5 echo "This is how to start a new line in a commandline file\n";
6 echo "This sentence is now on the next line";
7 ?>
8
9
10 The date today is <?php echo date('d-m-Y'); ?>
11 The date today is <?= date('d-m-Y'); ?>
```

Set breakpoints and then choose **Debug Project** or use the keyboard shortcut **Ctrl+ F6**.

To run each line of code in debug mode choose the **Step Into** button or use keyboard shortcut **F7**.

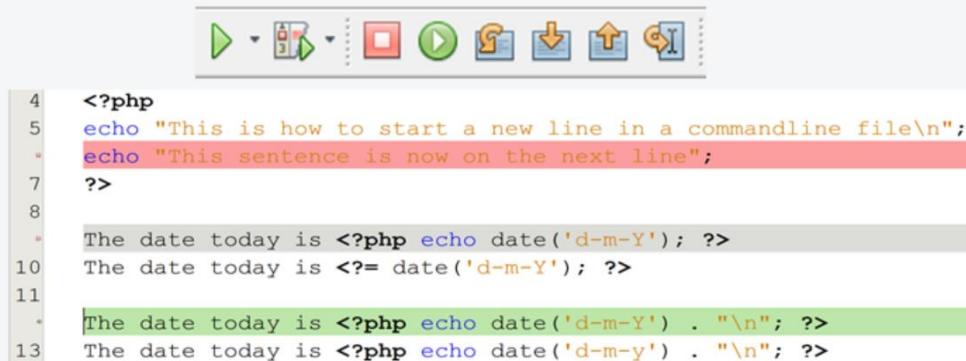
Debugging is the process of finding and resolving defects that prevent the correct operation of your application. NetBeans provides debugging support. Before you start to debug your application you should set a breakpoint on a line of code that you would like to pause the application on. The debugging features of most IDEs enable you to pause your code and then step into each line of code, at your own pace, whilst inspecting your code as it runs.

To set a breakpoint in NetBeans click into the grey margin well to the left of a line of code. If this line of code is PHP, the breakpoint will be set and the line will appear in red. If the line of code contains other text plus some inline PHP the breakpoint will be set and the line will appear in grey.

To start to debug click the Debug Project button on the toolbar or use the keyboard shortcut Ctrl + F5.

You can now use the debug buttons, such as Step Into (F7) to run each line of code in debug mode.

## NETBEANS: DEBUGGING



```
4 <?php
5 echo "This is how to start a new line in a commandline file\n";
6 echo "This sentence is now on the next line";
7 ?>
8
9
10 The date today is <?php echo date('d-m-Y'); ?>
11 The date today is <?= date('d-m-Y'); ?>
12
13 The date today is <?php echo date('d-m-Y') . "\n"; ?>
14 The date today is <?php echo date('d-m-y') . "\n"; ?>
```

The line of code in green is the line that will run next.

To stop debugging choose the **Finish Debugging Session** button or use keyboard shortcut **Shift + F5**.

You will make extensive use of the debugging feature as your code becomes more sophisticated and complex.



Create a new NetBeans Project called `MyFirstPHPCmdProject`.

This is not a web project. It should be a command line / terminal application.

Add a PHP file to the project called `PHPCmdFile` and experiment with the PHP examples you have seen so far within this chapter.

Practise running the file within the NetBeans output window.

Set a breakpoint on your first line of PHP code and debug the project. Make use of the Step Into feature.

## THE ECHO LANGUAGE CONSTRUCT

```
<?php  
echo "First line of output\n";  
echo "Second line of output\n";  
  
//The echo statement outputs text strings  
echo "Hello World!";  
  
echo "PHP in a web page <br>";  
echo "This is ", "text made from ", "multiple parameters";
```

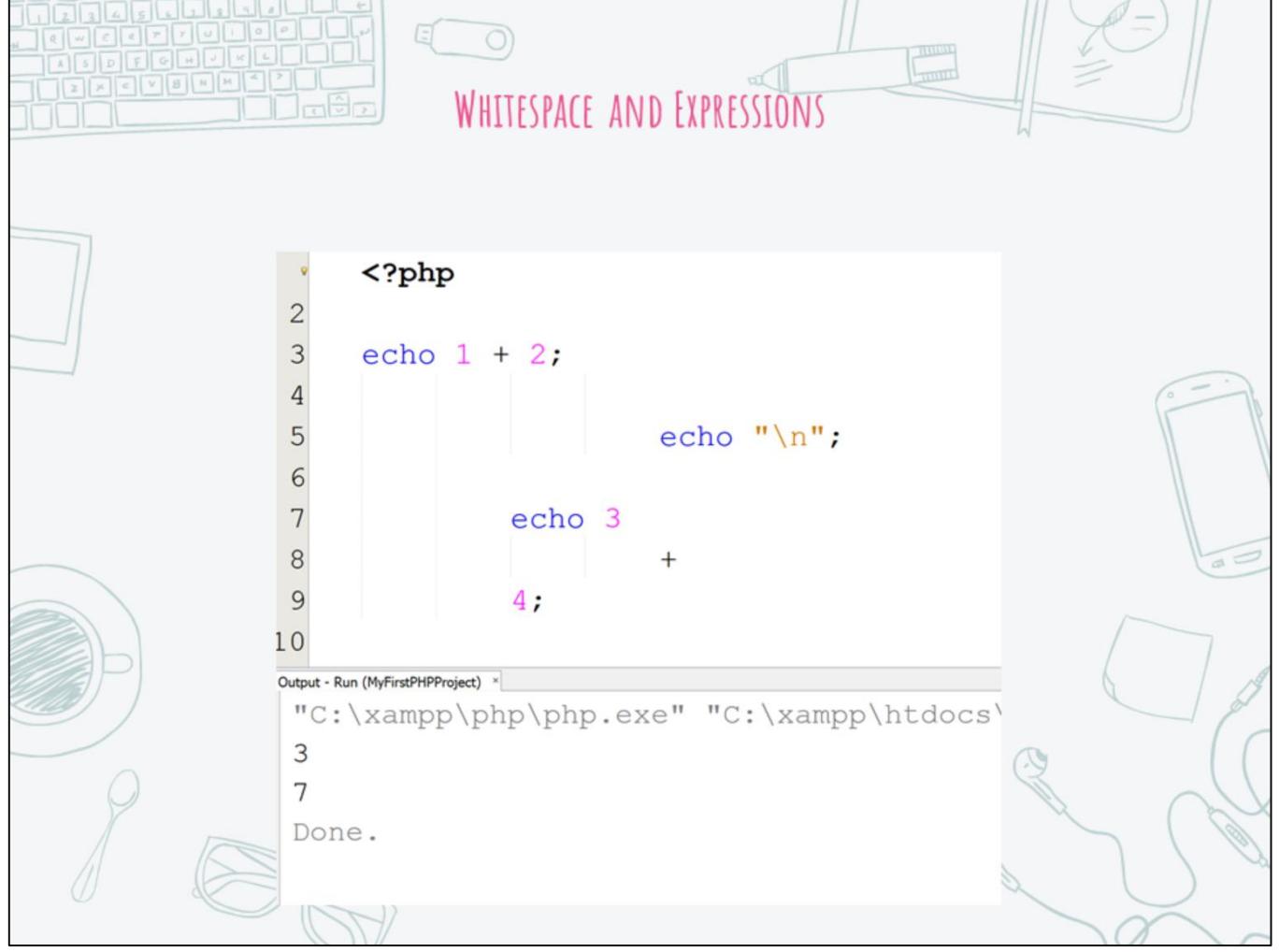
The echo statement in PHP is one way to get output to the screen. The echo statement can take multiple parameters and it can be used with or without brackets: echo or echo(). The above examples show how to output text with the echo statement.

You can use PHP to run scripts from the command line or to return web pages. To print multiple lines of text output from the command line you can use multiple echo statements with the special new line instruction "\n".

To output text in a web page you use HTML tags. HTML is the syntax, or markup, you use to construct web pages. HTML stands for HyperText Markup Language. You will learn more about HTML in the next chapter. To produce a line break in a HTML page you use the <br> tag.

As you can see you can also pass multiple values, or parameters, to the echo statement however, you will rarely see this.

# WHITESPACE AND EXPRESSIONS



```
<?php
2
3 echo 1 + 2;
4
5         echo "\n";
6
7 echo 3
8     +
9 4;
10

Output - Run (MyFirstPHPProject) ×
"C:\xampp\php\php.exe" "C:\xampp\htdocs\"
```

3  
7  
Done.

Whitespace, such as tabs and spaces, does not typically affect the execution of a PHP program. Above is an example PHP file. It is an entire file which begins with `<?php` and ends with an empty line.

There are three instructions or "statements", each of which is terminated with a semicolon. In PHP, statements always finish with semicolons.

These particular kinds of instructions are known as echo statements. The echo command (technically, "language construct") is followed by a value. The echo command outputs whatever value(s) it is given.

In this case one echo is given a simple value, a new line. In the other cases it is given an expression, which adds two numbers together.

Expressions are special kinds of instructions which have values; `1 + 2` has the value 3. Statements do not have values. There is no such thing as "the value of an echo statement". Echo is a command.

The values output here are therefore 3, a new line and 7.

The whitespace will be ignored by the PHP interpreter because it is inside of the PHP script. Remember, if the whitespace was outside of the PHP script, it may be sent to the output.

## PHP AND HTML

```
<!DOCTYPE html>
<html>
    <head>
        <title>Web Page</title>
    </head>
    <body>
        <?php echo '<h1>Hello World!</h1>'; ?>
        <p>Some other HTML content</p>
        <?= '<h1>Hello World from short tags!</h1>'; ?>
    </body>
</html>
```

**Hello World!**

Some other HTML content

**Hello World from short tags!**

Previously you saw a file that was entirely PHP. In this case you see a file that has HTML and PHP mixed together.

What will the interpreter do with this file? Recall that the PHP interpreter will not touch any non-PHP code. The file then will be returned with all the HTML in the same place, except in the `<body>` element there will be two new tags, both `<h1></h1>` header tags, which come from PHP.

The need to echo values is so common in PHP that there are special PHP tags for it, the short tags: `<?= ?>` Any value placed between these tags will be echoed. You can think of the equal sign as implying an echo.

You will see short tags used throughout various examples. You will see more HTML in the next chapter.

## CONFIGURING PHP

The PHP Interpreter is highly configurable and can be configured using an external file: **php.ini** or commands within your PHP script.

```
<?php  
  
error_reporting(E_ALL); //report all errors  
ini_set('display_errors', 'on'); //on the screen
```

PHP is an unusual language in many respects and one of its most peculiar features is how heavily configurable the interpreter is with an external ini file. The ini file can prevent certain operations from running, turn off features of the language, and even inject code before and after scripts run.

This was very useful in the earlier hobbyist days of PHP when the company who owned the webserver ("hosting company") typically offered shared hosting: your website would live together with many others on one machine. To prevent you from being a troublesome neighbour the ini file could be used to restrict what your application could do.

However, this also caused huge problems: code which would run successfully given one configuration file would not run successfully given another. This made programming for the language overly constrained. Today hosting is very cheap and ubiquitous and it's easy to get ownership of a machine without needing to share.

Most of the important configuration options can now be set in the PHP script itself, using, for example, the `ini_set()` function. Sometimes there are specific functions for specific configuration options. `error_reporting()` is one such function which sets how sensitive PHP will be to programming mistakes and errors.

## EDITING PHP.INI

You can access the php.ini file from the XAMPP Control Panel.



To edit the php.ini file you can right-click on the Apache Server's Config button in the XAMPP Control Panel.

On a Mac, the php.ini file is located here: **/usr/local/etc/php/7.0**



## PHP's Consistency



### PHP's Consistency

PHP is the product of a community project and therefore has a mix of different styles and approaches. This results in some inconsistency within the syntax and grammar.

Use the online documentation for clarification.

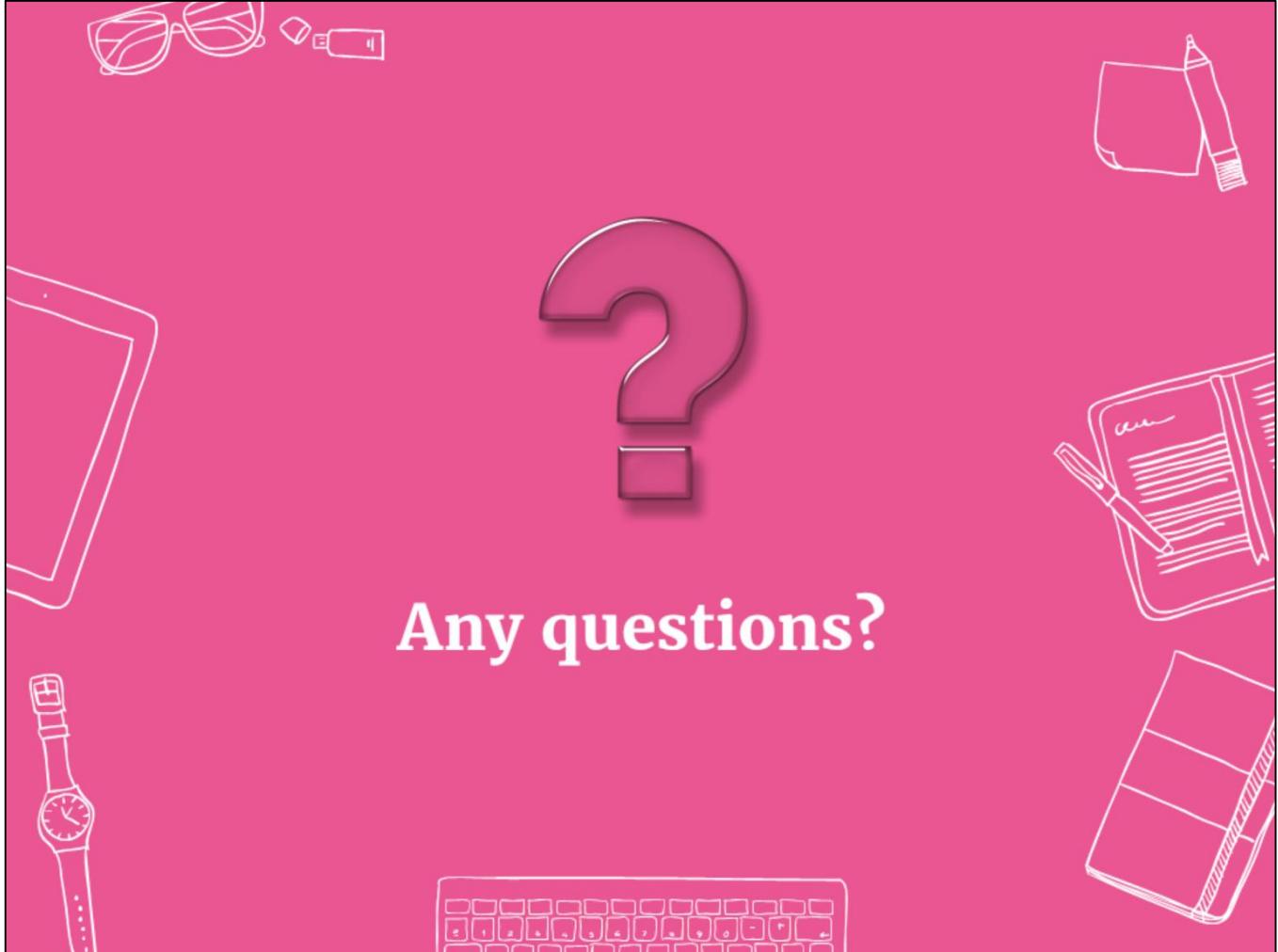


Some programming languages have a clear vision and a consistent set of approaches (Python, C#) however PHP is the product of a community project and its contributors. Each contributor responsible for a feature decides what it will look like and often contributors copy features from other languages and use their styles to do it, creating a mishmash. As a result of this the language has a mix of different styles and approaches. You will therefore come across some inconsistencies within the syntax and grammar.

Here are some examples of PHP's quirks:

- Some functions have underscores (\_) in their names — while others do not
- Some types are capitalized (NULL) and the rest lower case (boolean)
- We can use (bool) to cast a boolean — and indeed (boolean) too!
- Echo is a 'language construct' even though it behaves exactly like a function.

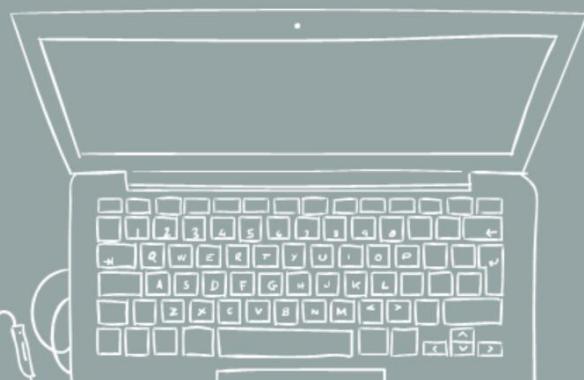
Therefore, learning PHP is a matter of learning its different idiosyncrasies and relying heavily on the online documentation and IDE.



**Any questions?**



## EXERCISE 4



# HTML: HYPERTEXT MARKUP LANGUAGE

## CHAPTER OVERVIEW

- An Introduction to HTML
- Headings
- Links and images
- File paths
- HTML tables
- Structural elements
- Lists
- Entities

### Exercise

Create html pages using basic tags, structural tags, tables, lists, links, comments, entities and images

This chapter is an introduction to HTML: Hyper Text Markup Language. The focus of this chapter is using HTML to produce static web page content. During the rest of your training you will create more dynamic HTML as a result of running PHP scripts. This chapter does not focus on the appearance or styling of the HTML. You will learn how to separate your styles from your content by using cascading style sheets in the next module.

# A SIMPLE HTML PAGE

```
<!DOCTYPE html>
<html>
  <head>
    <title>Page Title</title>
  </head>
  <body>

    <h1>This is a Heading</h1>
    <p>This is a paragraph.</p>

  </body>
</html>
```



HTML stands for Hyper Text Markup Language. It is the standard markup language for creating Web pages.

HTML describes the structure of Web pages using markup. This markup is in the form of elements which are represented by tags. These tags label pieces of content such as "heading", "paragraph", "table", and so on.

Browsers do not display the HTML tags, but use them to render the content of the page. It is the browser's job to interpret the markup and contents of the web page and decide how to display it.

The current version of HTML is version 5. The `<!DOCTYPE html>` tag is how you specify you are using this version.

The `<html>` element is the root element of an HTML page and all other HTML content must be enclosed within this element. It has two child elements: `head` and `body`.

The `<head>` element contains meta information about the document. The meta information in the head section typically defines the document title, character set, styles, links, scripts, and other metadata. The `<title>` element specifies a title for the document which, in most browsers, appears on the tab of the browser page. Most meta information however, is not shown within the browser. You will learn more about this metadata in subsequent topics.

The `<body>` element contains the visible page content. There are numerous tags that represent different structural elements such as `<h1>` and `<p>` tags. The `<h1>` element defines a large heading and the `<p>` element defines a paragraph. Most browsers add some whitespace before and after a paragraph.

# ELEMENTS AND ATTRIBUTES



HTML is comprised of elements and attributes. Elements usually consists of a start tag, content and an end tag.

`<h1>This is a H1 Heading</h1>`

The start tag is `<h1>`

The content is "This is a H1 Heading"

The end tag is `</h1>`

Some HTML tags are empty tags (sometimes called self-closing tags) such as `<br>` (line break) and `<img>` (image) tags. HTML 5 does not require empty tags to have a closing tag so these tags can be omitted. You can include a forward slash if preferred.

`<br>`

`<br />`

HTML elements can be nested and form a hierarchy of nodes known as the Document Object Model (DOM).

HTML elements can contain attributes which are name="value" pairs that add extra information to the element.

The language attribute informs search engines and accessibility tools such as screen readers of the language and dialect that is in use on the web page. The above example specifies the language as English. If you want to also specify the dialect use an additional two character code e.g. en-US or en-GB.

## HEADINGS

```
<!DOCTYPE html>
<html>
  <head>
    <title>Page Title</title>
  </head>
  <body>
    <h1>This is a H1 Heading</h1>
    <h2>This is a H2 Heading</h2>
    <h3>This is a H3 Heading</h3>
    <h4>This is a H4 Heading</h4>
    <h5>This is a H5 Heading</h5>
    <h6>This is a H6 Heading</h6>
  </body>
</html>
```

**This is a H1 Heading**

**This is a H2 Heading**

**This is a H3 Heading**

**This is a H4 Heading**

**This is a H5 Heading**

**This is a H6 Heading**

HTML headings are defined with the `<h1>` to `<h6>` tags.

`<h1>` defines the most important heading. `<h6>` defines the least important heading. How the headings are rendered depends on your browser unless you have specified some styling for your web page. You style your web pages using Cascading Style Sheets (CSS). You will learn more about CSS in the subsequent topics.

Use headings appropriately so that users can navigate your site content. Screen readers also use headings to index the structure and content of your pages.

## LINKS AND IMAGES

```
<!DOCTYPE html>
<html>
  <head>
    <title>Page Title</title>
  </head>
  <body>
    <a href="http://www.sky.com">This is a link</a>
    

    <a href="http://www.sky.com">
      
    </a>

  </body>
</html>
```



This is a link

HTML links are defined with the `<a>` tag (anchor tag). The link's destination is specified in the `href` attribute. An attribute is a `name="value"` pair that is embedded within an element tag. Attributes are used to provide additional information about HTML elements.

HTML images are defined with the `<img>` tag. The source file (`src`), alternative text (`alt`), width, and height are provided as attributes. The alternative text is displayed if the image can not be shown and is also read by accessibility tools such as screen readers.

HTML elements can be nested so that elements can contain other elements. All HTML documents consist of nested HTML elements as the head and body elements are nested within the root `html` element. In the example above an `image` element is nested within an `anchor` tag to form a clickable image rather than merely clickable text. The first image of the Sky logo is just an image and does nothing when clicked. The second logo is clickable and takes the user to the `sky.com` website once clicked.

## HTML FILE PATHS

FILE PATH	DESCRIPTION
	Logo.png is located in the same folder as the current page
	Logo.png is located in the images folder located in the current folder
	Logo.png is located in the images folder located at the root of the current web
	Logo.png is located in the folder one level up from the current folder

A file path describes the location of a file in a web site's folder structure. File paths are used when linking to external files such as images and other web pages. Later in the course you will use file paths to link to style sheets and JavaScript files.

File paths can be relative or absolute. A relative file path points to a file relative to the current page. An absolute file path is the full URL to an internet file such as:

```

```

If possible it is a best practice to use relative file paths because your web pages will not be bound to your current base URL. All links will work on your own development computer (localhost) as well as on your current public domain and your future public domains.

# TABLES

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>HTML Tables</title>
  </head>
  <body>
    <h1>This is a Table</h1>
    <table>
      <tr>
        <th>Firstname</th>
        <th>Lastname</th>
        <th>Age</th>
      </tr>
      <tr>
        <td>Jane</td>
        <td>Doe</td>
        <td>30</td>
      </tr>
      <tr>
        <td>Jo</td>
        <td>Bloggs</td>
        <td>94</td>
      </tr>
    </table>
  </body>

```

## This is a Table

Firstname	Lastname	Age
Jill	Smith	50
Eve	Jackson	94

<tr> Table Row  
<th> Table Header  
<td> Table Data

The `<table>` container defines a table. The table's contents are defined by creating table rows `<tr>`, table headings `<th>` and table data (cells) `<td>`.

## COMPLEX TABLES: VERTICAL

```
<table border="1">
<tr><th rowspan="3">Spring </th>
<td>March</td></tr>
<tr><td>April</td></tr>
<tr><td>May</td></tr>
<tr><th rowspan="3">Summer</th>
<td>June</td></tr>
<tr><td>July</td></tr>
<tr><td>August</td></tr>
<tr><th rowspan="3">Autumn</th>
<td>September</td></tr>
<tr><td>October</td></tr>
<tr><td>November</td></tr>
<tr><th rowspan="3">Winter</th>
<td>December</td></tr>
<tr><td>January</td></tr>
<tr><td>February</td></tr>
</table>
```

Seasons

<b>Spring</b>	March
	April
	May
<b>Summer</b>	June
	July
	August
<b>Autumn</b>	September
	October
	November
<b>Winter</b>	December
	January
	February

The `<table>` element above contains a `border` attribute. This border is visible around the table, the headings and the table cells. The `rowspan` attribute within the `<th>` elements makes the cell span more than one row. In this example the heading will span 3 rows.

## COMPLEX TABLES: HORIZONTAL

```
<table border="1">
  <tr>
    <th colspan="3">Spring </th>
    <th colspan="3">Summer</th>
    <th colspan="3">Autumn</th>
    <th colspan="3">Winter</th>
  </tr>
  <tr>
    <td>March</td>
    <td>April</td>
    <td>May</td>
    <td>June</td>
    <td>July</td>
    <td>August</td>
    <td>September</td>
    <td>October</td>
    <td>November</td>
    <td>December</td>
    <td>January</td>
    <td>February</td>
  </tr>
</table>
```

→

Spring			Summer			Autumn			Winter		
March	April	May	June	July	August	September	October	November	December	January	February

The colspan attribute within the <th> elements makes the cell span more than one column. In this example the heading will span 3 columns.

Tables should be used for tabular data and not as a technique to layout your pages.

## COMMENTS

```
<body>
  <h3>Comments</h3>
    <h4>Regular Comments</h4>
      <!-- This is a regular comment -->
    <h4>Conditional Comments</h4>
      <!-- The below is a conditional comment -->
      <!--[if IE 9]>
      .... some HTML here ....
      .... used to specify HTML tags for Internet Explorer only
      <! [endif]-->
  </body>
```

A document can be annotated with the comment element, denoted by the start tag `<!--` and end tag `-->`. Comments can appear in any section. Text within the comment is generally ignored by the browser. Comments cannot be nested within other comments.

Comments may be as long as you wish, but consider the performance ramifications of very large comments as they are downloaded by the server to the client even though they are not displayed on screen. Long comments make for wasted bandwidth.

Conditional comments are used to provide special instructions to Internet Explorer which often has non-standard behaviour. A conditional comment contains an `[if]` and `[endif]` start and end block. You will use these later in the course.

# BLOCK AND INLINE ELEMENTS

```
<body>
  <div style="background-color:#ea5692;color:white;padding:20px;">
    <h3>Block-level elements</h3>
    <p>A div (division) is a block-level element.</p>
    <p>It acts like a container for other elements.</p>
    <p>Other block-level elements include the heading tags
      and paragraph tags.</p>
  </div>
  <h3>Inline elements</h3>
  <p>A span is an <span style="color:red">inline</span> element</p>
  <p>Other inline elements include image and anchor tags.</p>
  <h3>Both divs and spans are used to style areas of a web page</h3>
</body>
```

#### Block-level elements

A div (division) is a block-level element.

It acts like a container for other elements.

Other block-level elements include the heading tags and paragraph tags.

#### Inline elements

A span is an inline element

Other inline elements include image and anchor tags.

Both divs and spans are used to style areas of a web page

Every HTML element has a default display value depending on what type of element it is. The default display value for most elements is block or inline. A block-level element always starts on a new line and takes up the full width available. An inline element does not start on a new line and only takes up as much width as necessary.

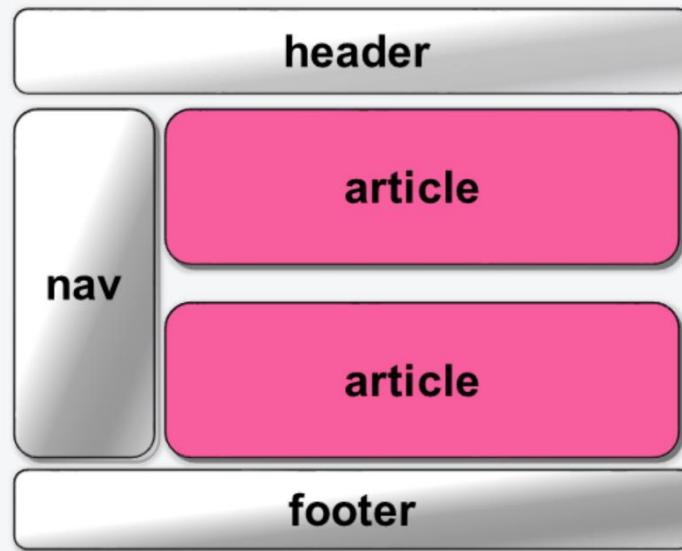
Examples of block-level elements are `<div>`, `<p>`, `<h1>` to `<h6>` and `<form>`. You will see the `<form>` tag later in the course. Examples of inline elements are `<span>`, `<img>` and `<a>`.

The `<div>` element is used to structure a HTML document into sections or divisions. It groups content together and allows you to apply certain attributes to this group. The `<div>` element can be used to apply a style to the sections of the document to control its appearance. The `span` element is an inline element that is also used to apply styles.

Both div and span are used to create areas of a webpage for styling and structural purposes if there is no semantic meaning to the area. If the area of the page you are creating has a semantic meaning use one of the new HTML 5 structural tags.

Note: the example above use the `style` attribute. You will see more on this later.

# STRUCTURAL ELEMENTS



HTML5 defines many new semantic elements. A semantic element clearly describes its meaning to both the browser and the developer. These are the structural semantic elements:

- <header> - Defines a header for a document or a section
- <nav> - Defines a container for navigation links
- <section> - Defines a section in a document
- <article> - Defines an independent self-contained article
- <aside> - Defines content aside from the content (like a sidebar)
- <footer> - Defines a footer for a document or a section
- <details> - Defines additional details
- <summary> - Defines a heading for the <details> element

Use the above elements instead of the non-semantic <div> and <span> elements whenever possible. Using these semantic sectioning elements will give the content meaning as well as provide a structural outline for assistive technologies and search engine indexes.



## LISTS



```
<h2>An Unordered HTML List</h2>
<ul>
  <li>Item 1</li>
  <li>Item 2</li>
  <li>Item 3</li>
</ul>
```

```
<h2>An Ordered HTML List</h2>
<ol>
  <li>Item 1</li>
  <li>Item 2</li>
  <li>Item 3</li>
</ol>
```

<ul>      Unordered list  
<ol>      Ordered list  
<li>      List item

### An Unordered HTML List

- Item 1
- Item 2
- Item 3

### An Ordered HTML List

1. Item 1
2. Item 2
3. Item 3

Basic lists in HTML can be ordered or unordered. An unordered list uses the `<ul>` tag whereas the ordered list uses a `<ol>` tag. Both types of list use the list item `<li>` element tag.

# LISTS: DESCRIPTION AND NESTED

```
<h2>A Description HTML List</h2>
] <dl>
    <dt>Item 1</dt>
    <dd>- a description of item 1</dd>
    <dt>Item 2</dt>
    <dd>- a description of item 2</dd>
</dl>
<h2>A Nested HTML List</h2>
] <ul>
    <li>Item 1</li>
    <li>Item 2
        <ul>
            <li>Sub-item A</li>
            <li>Sub-item B</li>
        </ul>
    </li>
    <li>Item 3</li>
</ul>
```

<dl>      Description list  
<dt>      Description term  
<dd>      Description data

## A Description HTML List

Item 1  
- a description of item 1  
Item 2  
- a description of item 2

## A Nested HTML List

- Item 1
- Item 2
  - Sub-item A
  - Sub-item B
- Item 3

HTML also provides description lists. A description list is a list of terms, with a description of each term.

The `<dl>` tag defines the description list, the `<dt>` tag defines the description term (name), and the `<dd>` tag describes each term (description data).

HTML also supports nested lists (lists within lists). These are useful for menus and navigation.

## ENTITIES

CHARACTER	ENTITY NAME	ENTITY #	DESCRIPTION
&nbsp;	&nbsp;	&#160;	Inserts a non-breaking space
<	&lt;	&#60;	Less than
>	&gt;	&#62;	Greater than
©	&copy;	&#169;	Copyright
&	&amp;	&#38;	Ampersand

Some characters are reserved in HTML such as the less than (<) and greater than (>) characters. To display these reserved characters you use character entities. Entities are used to implement reserved characters or to express characters that cannot easily be entered with the keyboard. Character entities begin with an ampersand followed either by the decimal index of the character or a name. This is then terminated by a semicolon.

Some common entities are:

- <              &lt;
- >              &gt;
- &              &amp;
- "              &quot;

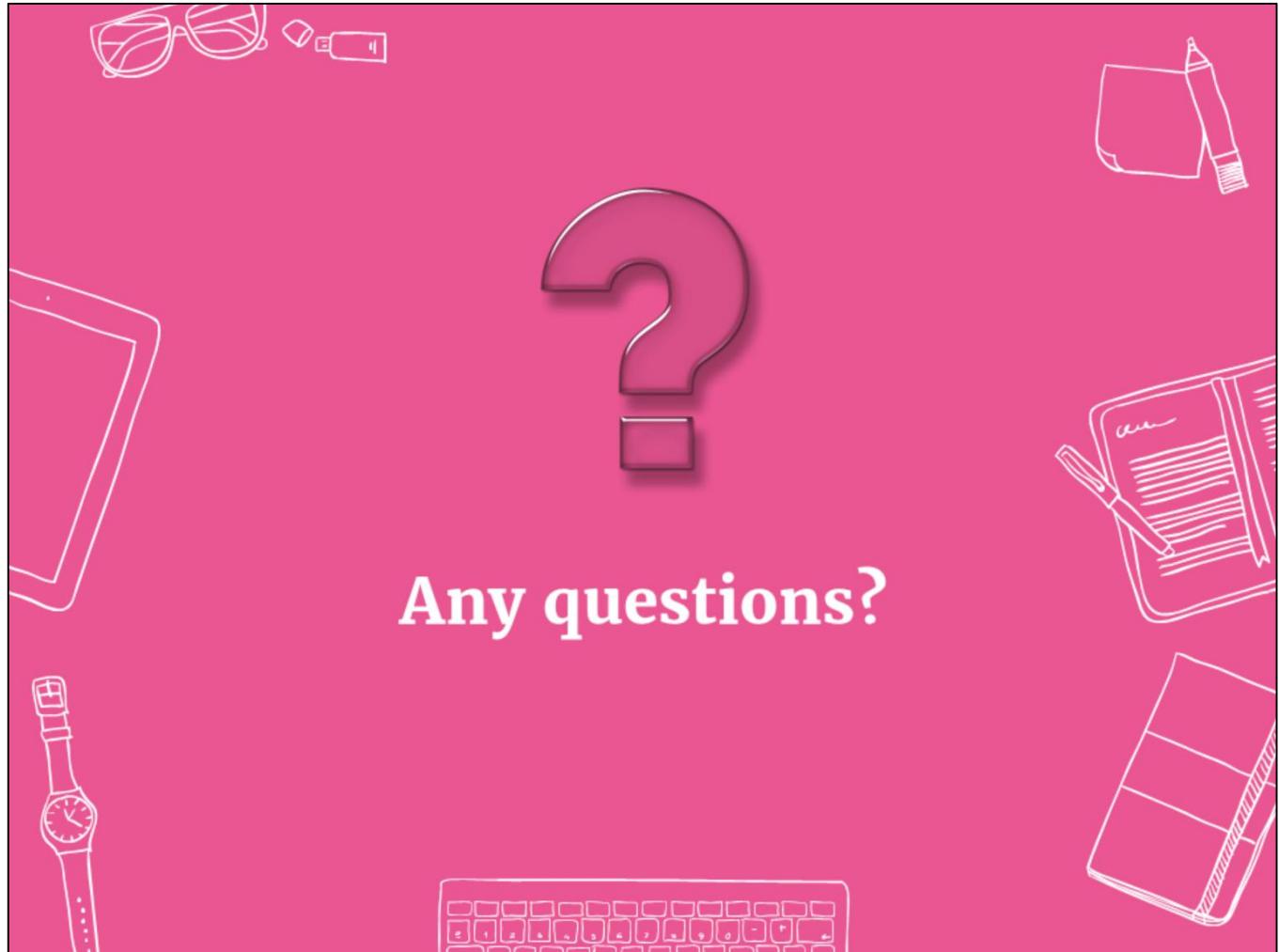
Another common character entity used in HTML is the non-breaking space: &nbsp;

A non-breaking space is a space that will not break into a new line. Two words separated by a non-breaking space will stick together. This is useful when breaking the words might be disruptive such as:

10 km/h

10 PM

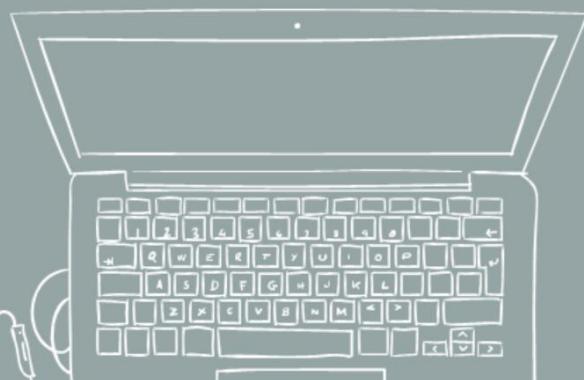
Another common use of the non-breaking space is to prevent the truncation of spaces performed automatically by browsers. If you write 10 spaces in your text, the browser will remove 9 of them. To add real spaces to your text, you can use the &nbsp; character entity.

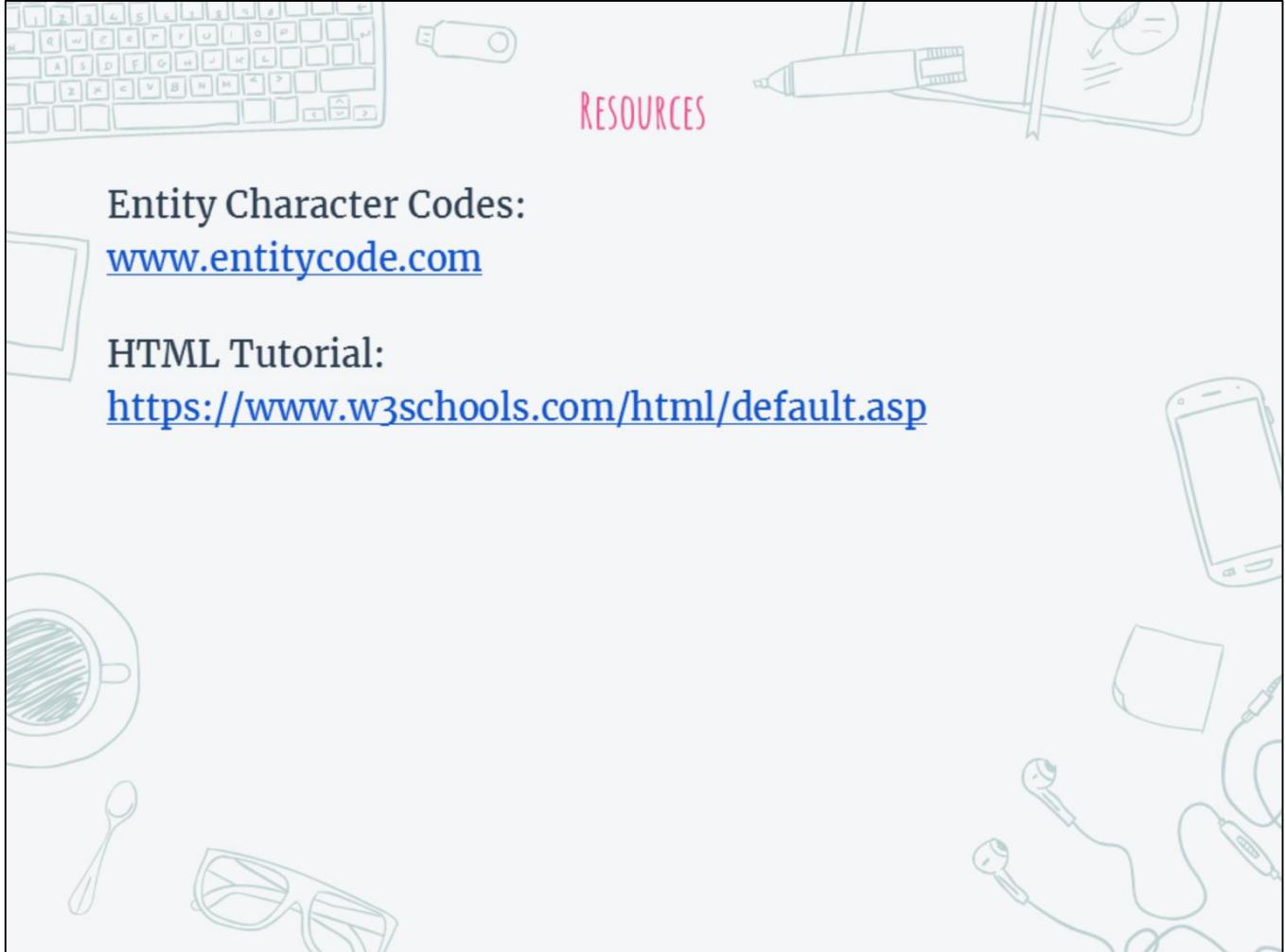


**Any questions?**



## EXERCISE 5





## RESOURCES

Entity Character Codes:

[www.entitycode.com](http://www.entitycode.com)

HTML Tutorial:

<https://www.w3schools.com/html/default.asp>

# CSS: CASCADING STYLE SHEETS

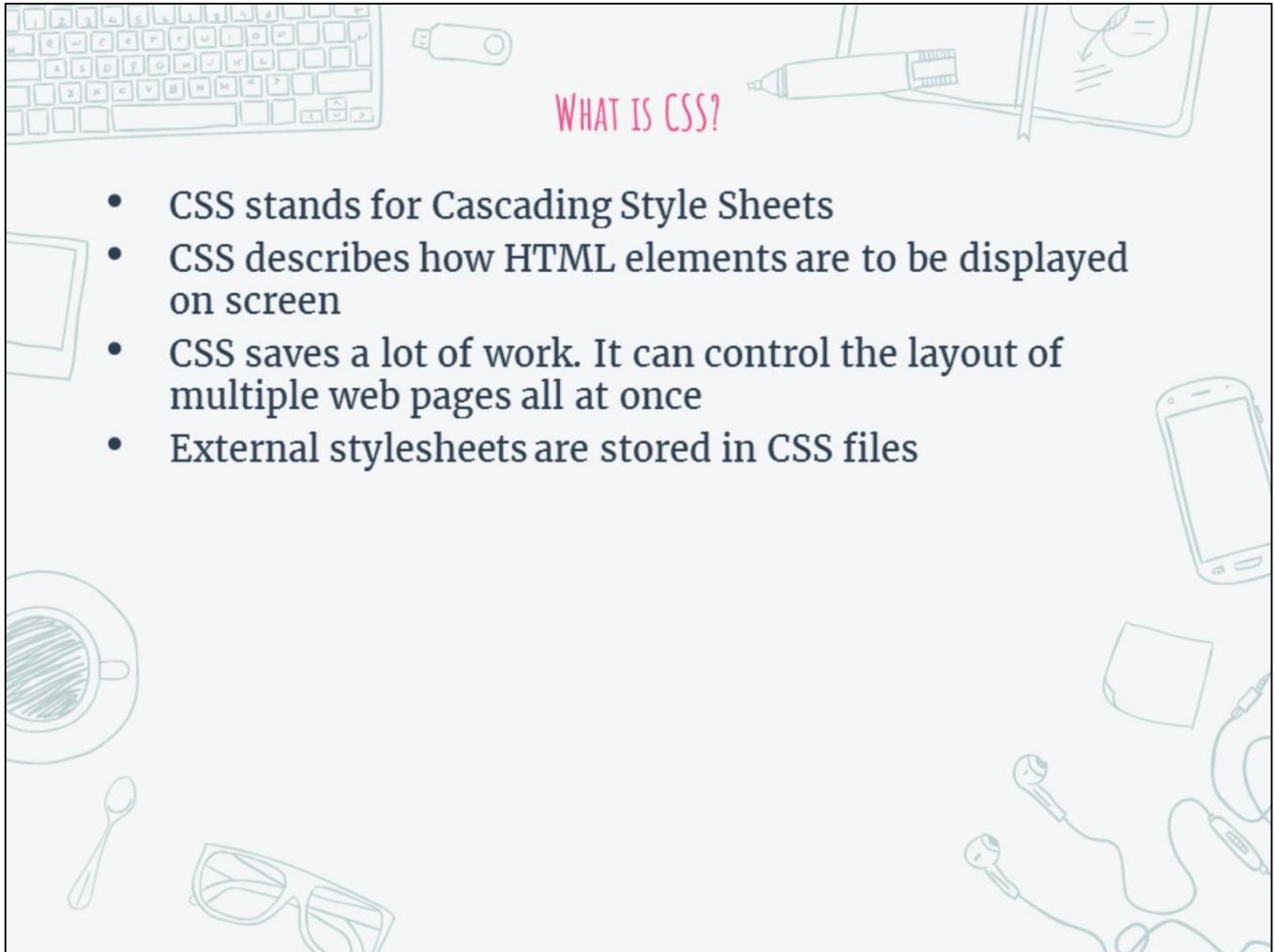
## CHAPTER OVERVIEW

- An Introduction to CSS
- CSS Selectors
- Applying Styles
- Inheritance and Specificity
- Responsive Web Design with Bootstrap

### Exercise

- Create external, internal and inline styles
- Style a web page using Bootstrap

This chapter is an introduction to CSS: Cascading Style Sheets. Stylesheets are used to control the appearance and layout of your web pages. The objective of this chapter is to understand basic CSS and to introduce the Bootstrap Framework. Bootstrap is the most popular HTML, CSS, and JavaScript framework for developing responsive, mobile-first web sites. It is completely free to download and use.



## WHAT IS CSS?

- CSS stands for Cascading Style Sheets
- CSS describes how HTML elements are to be displayed on screen
- CSS saves a lot of work. It can control the layout of multiple web pages all at once
- External stylesheets are stored in CSS files

CSS is used to define styles for your web pages, including the design, layout and variations in display for different devices and screen sizes. HTML was created to describe the content of a web page. To separate the styles from the content CSS was created. The ethos of CSS is to prevent too much clutter in your webpages by separating the styles from the elements. The style definitions are normally saved in external .css files. With an external stylesheet file, you can change the look of an entire website by changing just one file.

Cascading means that styles can fall (or cascade) from one style sheet to another, enabling multiple style sheets to be used on one HTML document. For example, you might have a global style sheet for your organisation plus site specific sheets that need to be merged (cascaded) together.

# SIMPLE CSS STYLE RULES

```
body {  
    font: 16px Verdana, sans-serif;  
    line-height: 1.8;  
    color: black;  
}  
h1 {font: 18px; color: darkmagenta;}  
p {color: darkgray;}  
span {color: magenta;}
```

## An Introduction to CSS

This paragraph has styling applied via an external CSS file.

## An Introduction to CSS

This paragraph has styling applied via an external **CSS** file.

A CSS rule-set consists of a selector and a declaration block. For example:

```
h1 {font: 18px; color: darkmagenta;}
```

The selector is h1. The declaration block sits within a pair of braces {} and each declaration includes a CSS property name and a value, separated by a colon. Each declaration ends with a semicolon.

The example above shows how the web page is rendered without the styles and then how it is displayed once the style sheet has been linked to the web page.

The style attributes that can be specified in the style rules are numerous. Those shown above include font, line-height and color.

## CSS SELECTORS

### SELECTOR EXAMPLE

### DESCRIPTION

p {color: blue;}

All elements of a particular type: all p tags

h1, h2, h3, h4 {color: blue;}

Multiple selectors can be combined with a comma: all h1 to h4 tags

\* {color: blue;}

The universal selector matches anything: all text will be blue

.blueit {color: blue;}

A class called blueit is created  
<p class="blueit">I am blue</p>

#blueit {color: blue;}

A tag with a given id is given a style  
<p id="blueit">I am blue</p>

CSS uses selectors to select (find) all matching elements in order to then apply the style rules. There are many different types of selectors used including selecting HTML elements based on their element name, their id, their class, and an attribute.

In the above table the first code sample is applying a blue font colour all the <p> tags on the page. The second example is applying the same styling to several HTML elements. Specifically, in this case the heading styles 1 to 4. Additional styling can be added to the heading styles in their own definitions. The third example will change the text colour to blue for any element in the page that contains text. The fourth example creates a class using the dot syntax. Any element given the class="blueit" attribute will have the style rule applied. The fifth example shows a style being created for an element with a given id attribute. This syntax uses a hash to denote it is an id style rule. Any element with the id="blueit" will automatically have the style applied.

## CSS ADVANCED SELECTORS

SELECTOR EXAMPLE	DESCRIPTION
p.blueit {color: blue;}	All tags in a style class of a certain type. All <p> tags with class="blueit"
p#blueit {color: blue;}	A specific tag with a given ID. All <p> tags with id="blueit"
p span {color: blue;}	Any descendant tag: child, grandchild, great grandchild etc. All spans within <p> tags
p > span {color: blue;}	A direct descendant tag <p>Text <span>blue text</span> text</p>
h1 + p {color: blue;}	An immediate sibling (adjacent selector) <h1>Heading</h1><p>Text is blue</p>

In the table above the first example will only apply the style to <p> tags with the class attribute applied. If the element type is not a <p> tag this style will not be applied even if the class attribute is defined on the element. The second example specifies the type of the element must be a <p> tag as well as the element having the correct value for the id attribute. The third example style will affect any span at any level within a paragraph tag.

The fourth example will affect only a span within a paragraph tag and not spans within any nested tags within the paragraph. The fifth example is known as an adjacent selector and will only affect paragraphs immediately following a h1 tag. i.e. at the same level in the document tree (sibling).

## CSS ATTRIBUTE SELECTORS

### SELECTOR EXAMPLE

### DESCRIPTION

a[href] {color: blue;}

All <a> tags with a href attribute present

img[src="logo.jpg"] {width: 5px;}

All <img> tags with a src attribute matching the value logo.jpg

table[title~="important"] {width: 100%;}

All tables with a title that includes the word important

[lang|=en]

All elements with a lang attribute value starting with "en"

In the table above the first example will only apply the style to <a> tags with a href attribute present. This style rule does not check for a value for the attribute – just its existence. The second example checks for the attribute value and will select all image tags with the src attribute with a value of logo.jpg. The third example selects all tables with a title attribute that includes the word important. This syntax uses the ~ (tilde) symbol. The last example selects all elements with a lang attribute value starting with en. This syntax uses the pipe () symbol.

## CSS COMMENTS

```
/* body style */
body {
    font: 16px Verdana, sans-serif;
    line-height: 1.8;
    color: black;
}
/* h1 style */
h1 {font: 18px; color: darkmagenta;}
p {color: darkgray;}
/*
    span {color: magenta;}
*/
```

### An Introduction to CSS

This paragraph has styling applied via an external CSS file.

You can use CSS comments within your style sheets. These start with /\* and end with \*/. CSS comments can span multiple lines. Use CSS comments to add descriptions / annotations to your style sheets and to comment out particular style rules for testing and troubleshooting.

In the above example the style rule for the span has been commented out so the text within the span is styled dark gray rather than dark magenta. As the span is within the paragraph it inherits the paragraph styles unless overridden by a more specific style rule.

# APPLYING STYLE SHEETS

## External

```
<link rel="stylesheet" type="text/css" href="styles.css" />
```

## Imported

```
<style type="text/css">
  @import url("styles.css");
</style>
```

## Internal

```
<style type="text/css">
  p {color:blue}
</style>
```

## Inline

```
<p style="color:blue">
  ...
</p>
```

There are four ways of inserting a style sheet:

- External style sheet
- Imported style sheet
- Internal style sheet
- Inline style

## EXTERNAL STYLE SHEETS

```
<head>
    <title>Applying Style Sheets</title>
    <link rel="stylesheet" type="text/css" href="ExternalStyles.css">
</head>

/* body style */
body {
    font: 16px Verdana, sans-serif;
    line-height: 1.8;
    background-color: linen;
}

/* h1 style */
h1 {font: 18px; color: darkmagenta;}
/* p style */
p {color: darkgray;}
```

### External Style Sheets

This paragraph has styling applied via an external CSS file.

With an external style sheet, you can change the look of an entire website by changing just one file. Each page must include a reference to the external style sheet file inside the `<link>` element. The `<link>` element goes inside the `<head>` section. The style sheet file must be saved with a `.css` extension and should not contain any html tags. NetBeans enables you to add css files to your project. You can easily link to this file by dragging the file from the Projects pane into the `<head>` section of your web page. NetBeans will then create the `<link>` element for you.

# IMPORTING STYLE SHEETS

```
<head>
    <title>Applying Style Sheets</title>
    <link rel="stylesheet" type="text/css" href="mystyles.css">
</head>

@import url("stylesD.css");
/* body style */
body {
    font: 16px Verdana, sans-serif;
    line-height: 1.8;
    background-color: linen;
}

/* h1 style */
h1 {font: 18px; color: darkmagenta;}
/* p style */
p {color: darkgray; }

p {background-color: white;}
```

## Imported Style Sheets

This paragraph has styling applied via an imported CSS file.

A style sheet can be imported into another style sheet using the `@import` rule. This rule must be the first rule within your css file. You can use this technique to load external fonts, such as Google Fonts, for use in your style sheets.

In the above example, the web page links to an external style sheet (`mystyles.css`) which in turn links to another style sheet (`stylesD.css`). This imported style sheet includes a rule that paragraphs should be styled with a white background colour. This results in the paragraph in the web page being styled with dark grey text (`mystyles.css`) and a white background (`stylesD.css`).

## INTERNAL STYLE SHEETS

```
<!DOCTYPE html>
<html>
  <head>
    <title>Applying Style Sheets</title>
  <style>
    body {
      font: 16px Verdana, sans-serif;
      line-height: 1.8;
      background-color: linen;
    }
    h1 { font: 18px; color: darkmagenta; }
    p { color: darkgray;
        background-color: white;
    }
  </style>
  </head>
  <body>
    <h1>Internal Styles</h1>
    <p>This paragraph has styling applied via internal styles.</p>
  </body>
</html>
```

### Internal Styles

This paragraph has styling applied via internal styles.

An internal style sheet may be used if one single page has a unique style. Internal styles are defined within the `<style>` element, inside the `<head>` section of an HTML page.

# INLINE STYLES

```
<!DOCTYPE html>
<html>
  <head>
    <title>Applying Style Sheets</title>
  </head>
<body style="font: 16px Verdana, sans-serif; line-height: 1.8; background-color: linen;">
  <h1 style="font: 18px; color: darkmagenta;">Inline Styles</h1>
  <p style="color: darkgray; background-color: white;">This paragraph has
    styling applied via inline styles.</p>
  <p>This paragraph has no specific inline styles.</p>
</body>
</html>
```

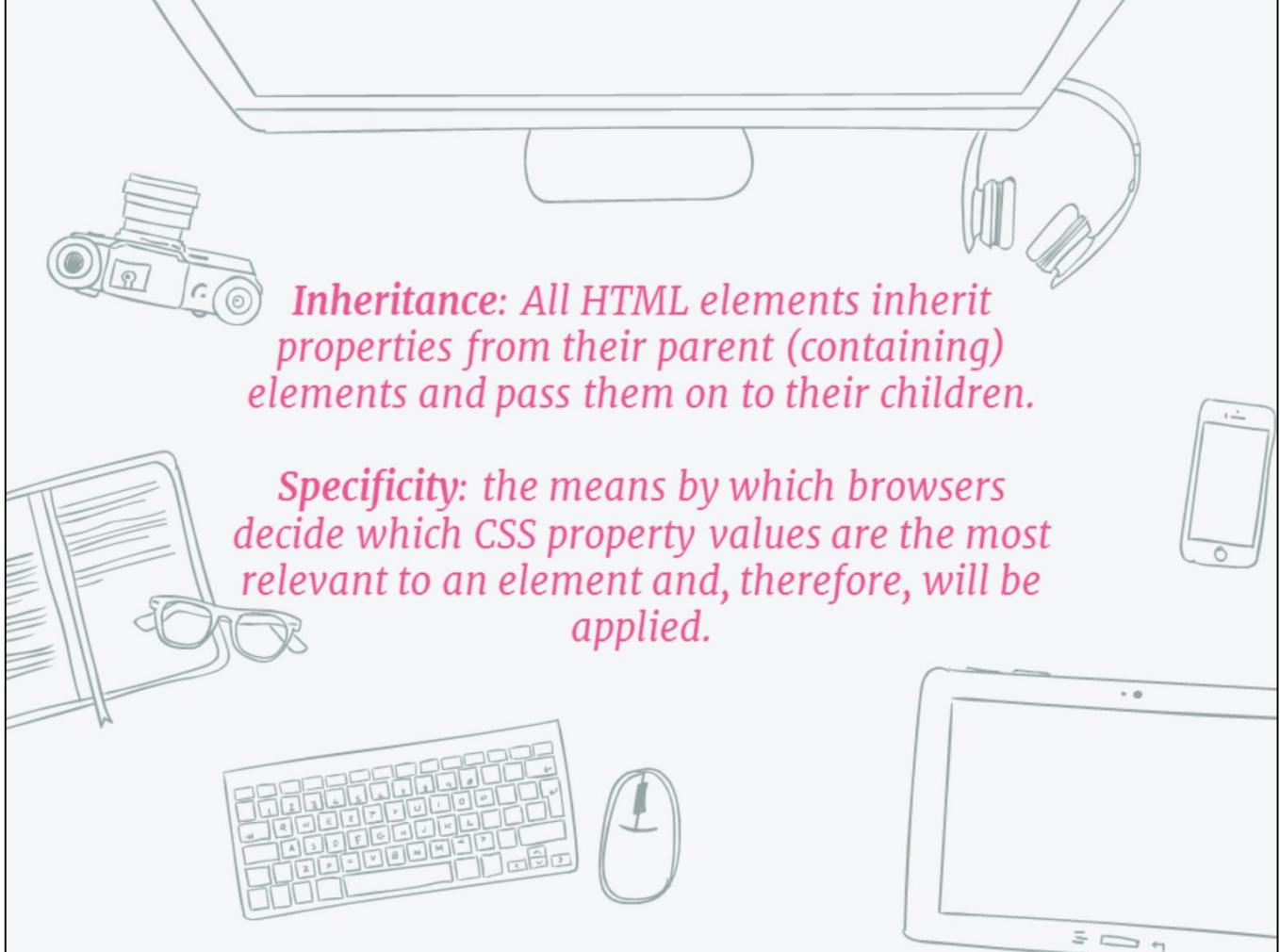
## Inline Styles

This paragraph has styling applied via inline styles.

This paragraph has no specific inline styles.

An inline style may be used to apply a unique style for a single element. To use inline styles, add the style attribute to the relevant element. The style attribute can contain any CSS property. The above example shows how to specify the font, line height and background colour for the body of the page plus specific styles for the heading and one of the paragraphs. Note that the second paragraph inherits the styles from body as this is its parent (containing) element but has no specific inline styles of its own so it is not styled with a white background or dark gray text like the first paragraph.

An inline style loses many of the advantages of an external or internal style sheet and results in the mixing of content with presentation therefore it is recommended that you use this method sparingly.



**Inheritance:** All HTML elements inherit properties from their parent (containing) elements and pass them on to their children.

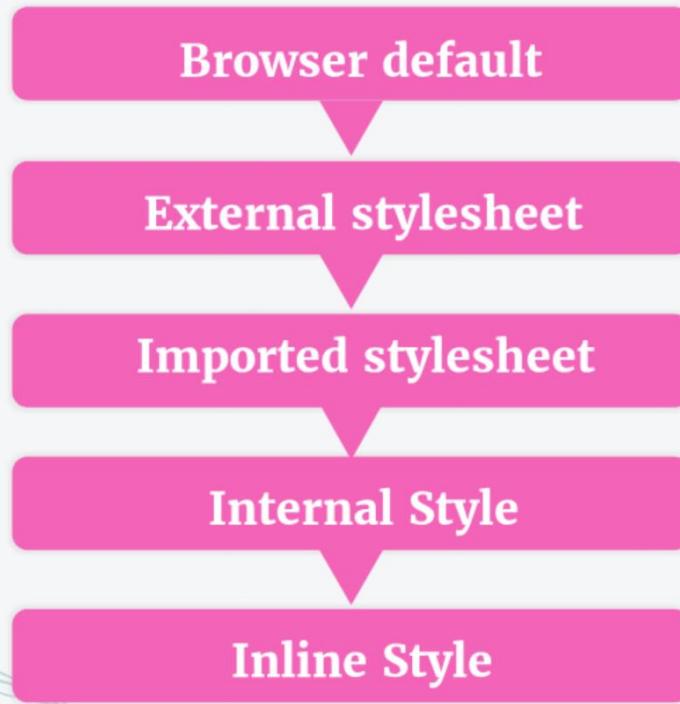
**Specificity:** the means by which browsers decide which CSS property values are the most relevant to an element and, therefore, will be applied.

CSS styles can be specified in many ways (inline, internal, imported and external) but they will all cascade into one virtual style sheet in the end. All HTML elements inherit properties from their parent (containing) elements and pass them on to their children. Sometimes conflicts will arise between the style sheets that influence how styles are merged. If multiple conflicting CSS rules are applied the specificity rules decide which styles are applied or overridden. CSS Specificity is the means by which browsers decide which CSS property values are the most relevant to an element and, therefore, will be applied. Specificity is based on the matching rules which are composed of different sorts of CSS selectors. The rules can be complex to understand but are only used if there are conflicting, contradictory styles. The simple order of style cascading (overriding) is:

- External stylesheet - overrides the browser default
- Imported stylesheet – overrides the linked stylesheet
- Internal style – overrides the imported stylesheet
- Inline style attribute – overrides internal styles

Note: not all styles are inherited. Margin, border and padding styles are not inherited. To override an inherited style specify a style rule with a higher specificity e.g. within an internal style sheet or as an inline style or using a more specific selector as per the specificity rules. When multiple declarations have equal specificity, the last declaration found in the CSS is applied to the element. Remember, specificity only applies when the same element is targeted by multiple declarations. As per CSS rules, directly targeted elements will always take precedence over rules which an element inherits from its ancestor.

## CSS SPECIFICITY



The simple order of style cascading (overriding) is:

- External stylesheet - overrides the browser default
- Imported stylesheet – overrides the linked stylesheet
- Internal style – overrides the imported stylesheet
- Inline style attribute – overrides internal styles

## CSS SPECIFICITY RULE ORDER

```
<style type="text/css">
  p          {color: red;}
  .blue      {color: blue;}
  #green     {color: green;}
</style>
</head>
<body>
  <p id="green" class="blue" style="color:orange;">Oranges are orange</p>
  <p id="green" class="blue">Oranges are green</p>
  <p class="blue">Oranges are blue</p>
  <p>Oranges are red</p>
</body>
</html>
```

Oranges are orange

Oranges are green

Oranges are blue

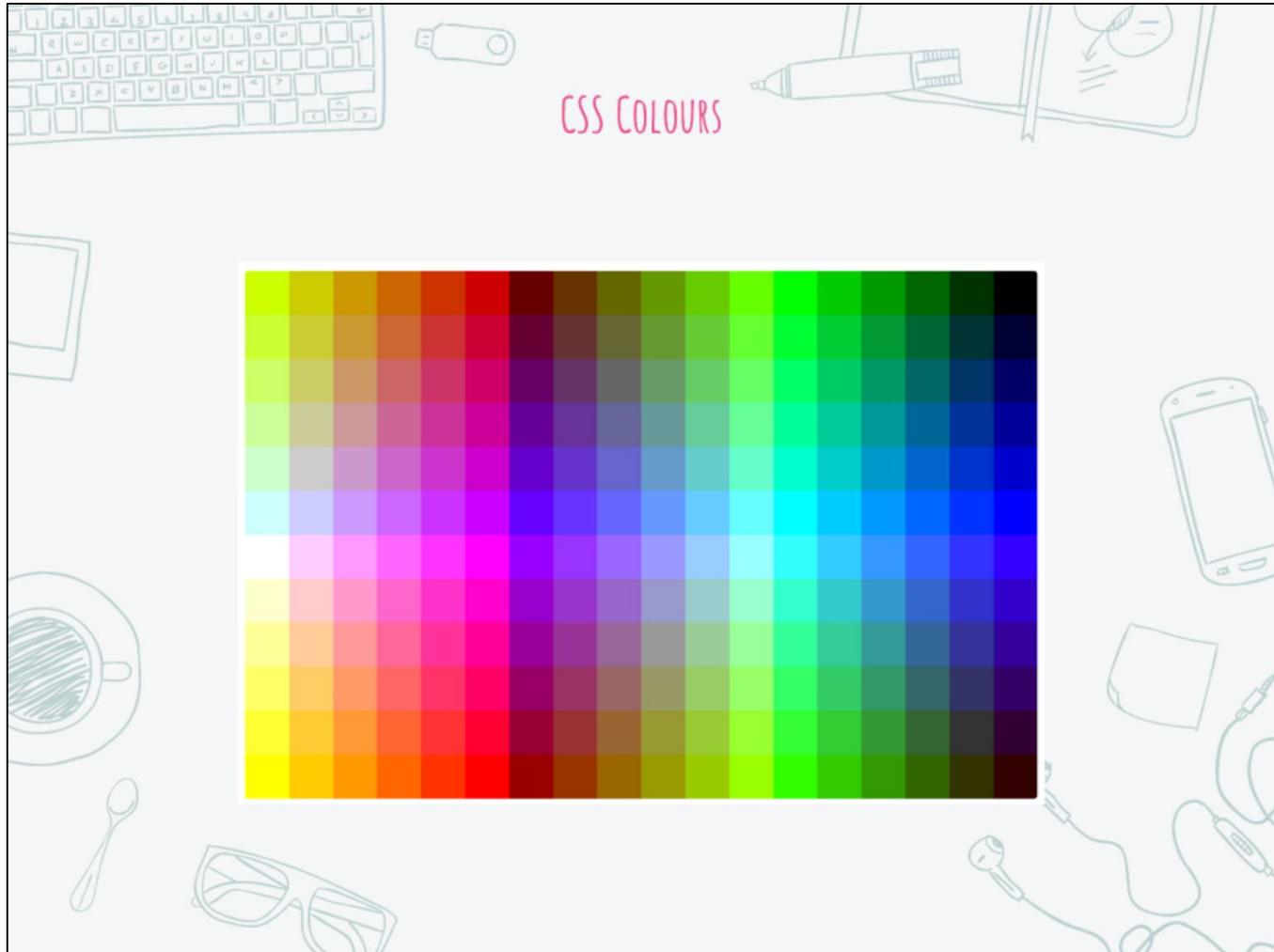
Oranges are red

In the cascade, style rules are sorted by specificity:

- A style applied with `style="..."` is more specific than
- A style applied by `id`, which is more specific than
- A style applied by `class`, which is more specific than
- A style applied by `element`

The above example demonstrates the specificity rules of selectors. The most specific style for the first paragraph is the inline style defined with `style="..."`. The most specific style for the second paragraph is the rule with the `id` attribute. The most specific style for the third paragraph is the `class` style rule. The final paragraph has no inline styles so the internal style sheet style is applied. This style rule overrides any styles defined in an imported or external style sheet. If this internal style was not present in the web page then an imported style rule would be applied.

Note: rules of the same specificity that appear later in the style sheet are more specific than earlier rules.



## CSS COLOURS

Colours are displayed combining RED, GREEN, and BLUE light (RGB). With CSS, colours can be specified in different ways:

By colour names (hundreds of supported names: red, crimson, indianred, firebrick, salmon, darkred etc.)

- As RGB values (pattern 0-255, 0-255, 0-255 e.g. `rgb(255,0,0)` is red)
- As hexadecimal values (pattern `#RRGGBB` e.g. `#ff0000` is red)
- As HSL values (CSS3: Hue, Saturation, and Lightness e.g. `hsl(0, 100%, 50%)` is red)
- As HWB values (CSS4: Hue, Whiteness, Blackness is a suggested standard for CSS4 e.g. `hwb(0, 0%, 0%)` is red)

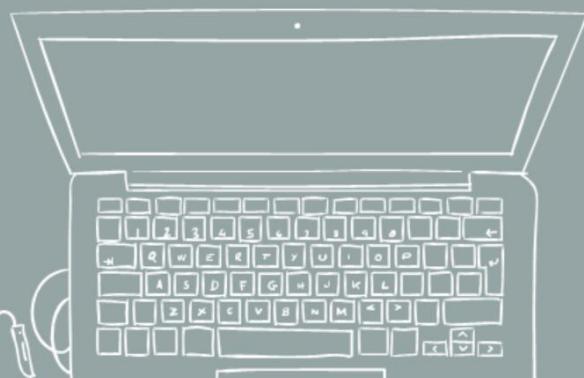
You do not have to memorise these colour codes as there are lots of free colour pickers and colour palette generators available. Plus IDEs such as NetBeans support the colour names and include a colour chooser.

You can find a colour palette generator here: <https://coolors.co/>

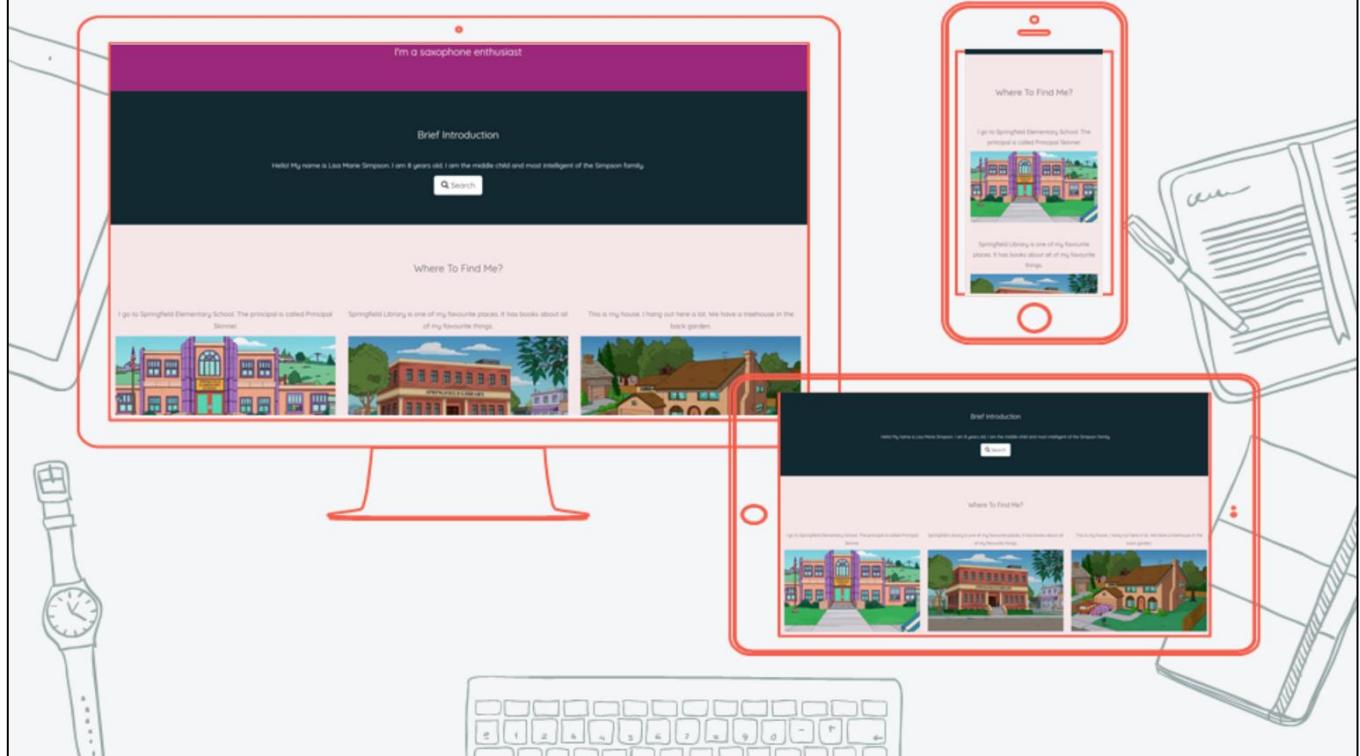
Use the spacebar to spin for more colours and lock in any colours you like. The generator will suggest complimentary colours for your palette.



## EXERCISE 6A



# RESPONSIVE WEB DESIGN



Responsive web design is the approach that suggests that design and development should respond to the user's behaviour and environment based on screen size, platform and orientation. The practice consists of a mix of flexible grids and layouts, images and an intelligent use of CSS media queries. The aim of Responsive Web Design is to make your web page look good on all devices: desktops, tablets, and phones.

Responsive web design is about using CSS and HTML to resize, hide, shrink, enlarge, or move the content to make it look good on any screen. You can create your own responsive design using a combination of HTML, CSS and media queries. A media query is used to define different style rules for different media types/devices. Media queries look at the capability of the device, and can be used to check many things, such as: width and height of the viewport. Most developers however, use a pre-built responsive framework to speed the development process and create a consistent approach for each application they develop. The most widely used responsive framework is Bootstrap.



# BOOTSTRAP



## Bootstrap

Build responsive, mobile-first projects on the web with the world's most popular front-end component library.

Bootstrap is an open source toolkit for developing with HTML, CSS, and JS.

[Get started](#)[Download](#)

Currently v4.0.0-beta



Bootstrap was developed by Mark Otto and Jacob Thornton at Twitter, and released as an open source product in August 2011 on GitHub. Bootstrap is the most popular HTML, CSS, and JavaScript framework for developing responsive, mobile-first web sites. It is completely free to download and use. It speeds up front-end development and includes HTML and CSS based design templates for typography, forms, buttons, tables, navigation, modals, image carousels and many other elements. Bootstrap gives you the ability to easily create responsive designs.

- Easy to use: anybody with just basic knowledge of HTML and CSS can start using Bootstrap
- Responsive features: Bootstrap's responsive CSS adjusts to phones, tablets, and desktops
- Mobile-first approach: in Bootstrap 3, mobile-first styles are part of the core framework
- Browser compatibility: Bootstrap is compatible with all modern browsers

Note: HTML is used to define the content of web pages. CSS is used to specify the layout of web pages. JavaScript is used to program the behaviour of web pages.

JavaScript is a cross-platform, object-oriented scripting language. It is a small and lightweight language. Client-side JavaScript extends the core language by supplying objects to control a browser and its Document Object Model (DOM). For example, client-side extensions allow an application to place elements on an HTML form and respond to user events such as mouse clicks, form input, and page navigation.

## Bootstrap CDN

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />
    <!-- Bootstrap CSS -->
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-beta/css/bootstrap.min.css" integrity="sha384-/Y6pD6FV/Vv2HJnA6t+vslU6fwYXjCFtcEpHbNJ0zUpv3K/odUJf8eCI7NQkQ=" crossorigin="anonymous">
  </head>
  <body>
    <h1>Hello, world!</h1>

    <!-- Optional JavaScript -->
    <!-- jQuery first, then Popper.js, then Bootstrap JS -->
    <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js" integrity="sha384-hgoWltywGZLxW0O/ZlXoJZJ47JgE6GA4jlZUW0X8Z9JL4dUWZBj0qjZmOOGDZP" crossorigin="anonymous">
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.12.9/dist/umd/popper.min.js" integrity="sha384-ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsE4f2WpO40M2qR8Q9r9DUZI43wM&lt;!-- Bootstrap JS -->
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-beta/js/bootstrap.min.js" integrity="sha384-jqchDLeZrdKU6kZu9yu7TfGK6t+o8ZqtmfZPnVZuJyJ07ZuWZP+q8vWZa+u0" crossorigin="anonymous">
  </body>
</html>
```

To use Bootstrap you can either download it or reference it from a Content Delivery Network (CDN). Bootstrap has dependencies on various JavaScript libraries so you need to reference those as well. You must put the link to the Bootstrap CSS before any other style sheet link in the `<head>` section of your page. It is recommended that you reference the JavaScript libraries at the bottom of your file just above the closing `</body>` tag to improve page load performance. Ensure the links to jQuery and Popper.js appear first, as the JavaScript plugins depend on them.

One advantage of using the Bootstrap CDN is that many users already have downloaded Bootstrap from MaxCDN when visiting another site. As a result, it will be loaded from cache when they visit your site, which leads to faster loading time. Also, most CDNs will make sure that once a user requests a file from it, it will be served from the server closest to them, which also leads to faster loading time.

Bootstrap uses HTML elements and CSS properties that require the HTML5 doctype so always include the HTML5 doctype at the beginning of the page, along with the lang attribute and the correct character set. Bootstrap 3 is designed to be responsive to mobile devices. To ensure proper rendering and touch zooming, add the following `<meta>` tag inside the `<head>` element:

```
<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
```

You can find a Bootstrap starter template on the Bootstrap website: <https://getbootstrap.com/docs/4.0/getting-started/introduction/>



# BOOTSTRAP GRIDS



col-sm-1	col-sm-1	col-sm-1	col-sm-1	col-sm-1	col-sm-1	col-sm-1	col-sm-1	col-sm-1	col-sm-1	col-sm-1	col-sm-1
col-sm-2		col-sm-10									
col-sm-3		col-sm-3			col-sm-3		col-sm-3		col-sm-3		col-sm-3
col-sm-4		col-sm-4					col-sm-4		col-sm-4		col-sm-4
col-sm-4		col-sm-8									
col-sm-6		col-sm-6					col-sm-6				
col-sm-12											

```
<div class="row">
    <div class="col-sm-2" style="background-color: #c83d8f;">
        col-sm-2
    </div>
    <div class="col-sm-10" style="background-color: #e55bad;">
        col-sm-10
    </div>
</div>
```

Place rows inside a **container** or **container-fluid**



Bootstrap's grid system allows up to 12 columns across the page. If you do not want to use all 12 column individually, you can group the columns together to create wider columns. The grid system is responsive, and the columns will re-arrange depending on the screen size. On a big screen it might look better with the content organized in three columns, but on a small screen it would be better if the content items were stacked on top of each other. Ensure that your grid columns add up to twelve for a row otherwise they will be stacked no matter what the size of the viewport.

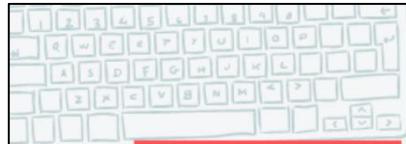
The Bootstrap grid system has four classes:

- xs (extra small - for phones)
- sm (small - for tablets)
- md (medium - for desktops)
- lg (large - for larger desktops)

You can combine these classes to create more dynamic and flexible layouts.

- Rows must be placed within a .container (fixed-width) or .container-fluid (full-width) for proper alignment and padding
- Use rows to create horizontal groups of columns
- Content should be placed within columns, and only columns may be immediate children of rows
- Predefined classes like .row and .col-sm-4 are available for quickly making grid layouts
- Grid columns are created by specifying the number of 12 available columns you wish to span. For example, three equal columns would use three .col-sm-4

Note: Each class scales up, so if you wish to set the same widths for xs and sm, you only need to specify xs.



## FONTS



### Google Fonts

#### Roboto

Christian Robertson (12 styles)

#### Archivo

Omnibus-Type (8 styles)

#### Saira Semi Condensed

Omnibus Type (9 styles)

All their equipment  
and instruments are  
alive.

A red flair  
silhouetted the  
jagged edge of a  
wing.

I watched the storm,  
so beautiful yet  
terrific.

#### Open Sans

Steve Matteson (10 styles)

Almost before we  
knew it, we had  
left the ground.

#### Lato

Lukasz Dziedzic (10 styles)

A shining crescent  
far beneath the  
flying vessel.

#### Slabo 27px

John Hudson (1 style)

It was going to be a  
lonely trip back.

Bootstrap will change your font from the browser default. Bootstrap's global default font-size is 14px, with a line-height of 1.428. This is applied to the <body> element and all paragraphs (<p>). You can change the font within your own external style sheet. You can also import new fonts from Google Fonts: <https://fonts.googleapis.com/>

Use the import directive at the top of your css file to import the new font and then use it within your css file:

```
@import url('https://fonts.googleapis.com/css?family=Quicksand');
```

```
body {
```

```
    font: 20px Quicksand, sans-serif;
```

```
    line-height: 1.8;
```

```
    color: #f5f6f7;
```

```
}
```



# Tutorials

w3schools.com



There are numerous free Bootstrap tutorials available on the web:

<https://www.w3schools.com/bootstrap/default.asp>

<https://www.tutorialspoint.com/bootstrap/>

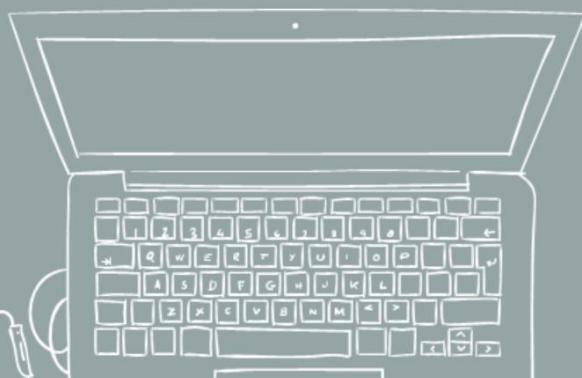
<https://mdbootstrap.com/basic-constructions/>



# Any questions?



## EXERCISE 6B



## RESOURCES

30 CSS Selectors:

<https://code.tutsplus.com/tutorials/the-30-css-selectors-you-must-memorize--net-16048>

CSS Selector Tutorial:

[https://developer.mozilla.org/en-US/docs/Learn/CSS/Introduction\\_to\\_CSS/Simple\\_selectors](https://developer.mozilla.org/en-US/docs/Learn/CSS/Introduction_to_CSS/Simple_selectors)

CSS Colours

[https://www.w3schools.com/cssref/css\\_colors.asp](https://www.w3schools.com/cssref/css_colors.asp)

Bootstrap Starter Template

<https://getbootstrap.com/docs/4.0/getting-started/introduction/>

# PHP SYNTAX

## CHAPTER OVERVIEW

- PHP Basic Syntax
- Comments
- Case-sensitivity
- Variables
- Interpolation and Concatenation
- Echo and Print
- Data Types
- Constants
- Casting
- Operators

### Exercise

Practice with variables and operators

The PHP language may be divided into two feature sets: procedural and object-oriented.

In the first set are those features which support simple statements and instructions which build up to small groupings of instructions which are called procedures or functions. The goal of this chapter is to learn more about the PHP syntax in order to understand the procedural nature of PHP. You will learn more about PHP's object-oriented capabilities later in the course.

## PHP BASIC SYNTAX RECAP

- PHP scripts have a **.php** extension
- PHP can be mixed with html
- Scripts start with **<?php**
- All statements end with a semicolon ;

```
<body>
    <h1>Basic PHP Syntax Explained</h1>
    <?php
        echo "Remember me? I print output to the page";
        echo "<br>"; //This is a web page line break
        echo "Don't forget - PHP statements end with a semicolon";
    ?>
</body>
</html>
```



A PHP script is normally contained in a file with a '.php' extension. The script is executed on the server, and the HTML result is sent back to the browser.

The PHP script can be placed anywhere in the document and starts with **<?php**. The script may also end with **?>** if it is contained within HTML.

The echo statement outputs the text to the web page. Remember, all PHP statements end with a semicolon (;).

## COMMENTS

```
<?php
//Here is a comment on one line
# Here is another one-line comment

] /*
Here is a block comment which
spans multiple lines
- */

//Below is a comment that omits part of the line of code
$result = 2 /* + 10 */ + 2;
echo $result;

] /**
Here is a doc comment
- /
?>
```



A comment is a line of code that is not executed as part of the program. It is intended to be read by anyone that works with the code to aid in their understanding. The very best code is straightforward to read: you can see by how the developer has laid it out what it does. However code never tells you why it was written: what was the developer aiming for?

Code very often doesn't do exactly what was intended and you'll be tasked to fix it. Without knowing what was intended you'll not know how broken it is. Comments help with this problem.

You can add them to code and they will be completely ignored by the interpreter. One line comments are prefixed by `#` or `//` and PHP ignores any text appearing after them. Multiline comments begin with `/*` and end with `*/` and any text appearing in between is ignored. These different syntaxes allow you to put comments anywhere in your code, in convenient places where the person reading it needs to understand something more about why it's written that way. Comments can also aid the original developer by refreshing their memory after time away from writing the code.

Sometimes developers add comments to code that explains syntax, or describes nothing more than what the code does, for example

`1 + 2 //add two numbers.` While you're learning these might be a good idea but in real code they can be counterproductive, since people often change the code without changing the comments so they become misleading.

The last kind, one which begins with `/**` (i.e., two stars) is recorded as a kind of code annotation. You can use these DocBlock comments to provide verbose information about an element in your code. IDEs can use this information to assist with auto-completion and the `phpDocumentor` can use these comments to generate documentation. See <https://www.phpdoc.org/> for more information.

## VARIABLES

- A symbolic reference in memory to a piece of data

```
<?php  
  
$result = 2 + 2;  
echo $result;  
$colour = "blue";  
$i = 5;  
$msg = "Good morning";  
$j = 99.99;
```

- Variables are defined with a \$
- Only letters and underscores are allowed
- Variables names are case-sensitive

A variable is a named placeholder in memory for storing a piece of data. Variables are called as such because as your program executes the data that is stored can be changed, hence it varies.

PHP is not a strongly-typed language. This means that as a developer you are not forced to declare your variables and specify the type of data they will store. To create a variable in PHP you simply assign a value into it. All PHP variables start with a dollar (\$) sign, followed by the name of the variable. The name of the variable must start with either a letter or an underscore (\_). Do not attempt to put other punctuation characters in your variable names; these are not allowed. Only alpha-numeric values and underscores are valid. Variable names are case-sensitive. \$colour and \$COLOUR refer to two different variables.

You assign a value to a variable using a single equals operator (=). Assignments are read right-to-left, meaning that the expression on the right of the equals sign is evaluated first then whatever value it produces is put into the variable named on the left.

In PHP variables that have not been defined have an initial value of null.

Note: use quotes when the value is a string (text).

## CASE SENSITIVITY

```
// All classes, functions and keywords are not case-sensitive  
echo "The echo statement is NOT case sensitive";  
ECHO "echo eChO ECHO are all okay";  
  
// Variable names are case-sensitive  
$colour = "red";  
echo "My dress is " . $colour;  
echo "Your trousers are " . $Colour;  
echo "His hair is " . $COLOUR;
```



My dress is red  
Your trousers are  
His hair is

In PHP, all classes, functions, user-defined functions and keywords, such as if, else, while, and echo, are not case-sensitive.

So echo, eChO or ECHO are all perfectly valid statements.

Variable names, however, are case-sensitive. So the \$colour variable is not recognised when referred to as either \$Colour or \$COLOUR. So the output from the above statements would display:

My dress is red

Your trousers are

His hair is

## INTERPOLATION AND CONCATENATION

- We can use a variable directly inside any double-quoted string. PHP casts it to a string first

```
$username = 'Student';
$email = 'student@sky.com';

echo "$username is available at $email";
Student is available at student@sky.com
```

- Strings can be concatenated (joined together) to produce one larger string

```
$message = ' This ' . ' is ';
$message = $message . 'concatenation!';
echo $message;

This is concatenation!
```

Variables are interpolated (substituted) within double quoted strings. Before interpolating a variable, PHP will cast it to a string.

Dot '.' is the concatenation operator and joins strings together into one larger string. PHP will also cast anything concatenated into a string before it joins them together.

A cast simply makes it explicit that it is OK to use a variable of one type as another type in this instance.

## OUTPUTTING VARIABLE VALUES WITH ECHO

- The dot is used for explicit concatenation
- The variable values are converted to strings first
- The double-quoted string is interpolated

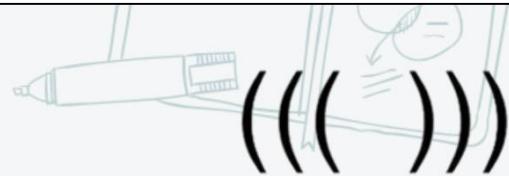
```
<?php  
$colour = "red";  
echo "My dress is " . $colour;  
  
// or  
  
echo "My dress is $colour";
```



The above examples show variables being used with the echo statement. A variable is used with concatenation and also within a double-quoted string i.e. it is interpolated.



## ECHO EXAMPLES



```
echo true;
```

```
1
```

```
echo [1, 2, 3];
```

```
Array
```

```
echo 'Hello\nWorld';
echo "\nHello\nWorld";
```

```
Hello\nWorld
Hello
World
```

```
echo 'Here is an "echo"
of a sum ', 1 + 2;
```

```
Here is an "echo" of a sum 3
```

Echo will take an argument of any type and cast it to a string and then output that string. This can give you unexpected results. If you echo true you get 1, if you echo false you get an empty string " (no output) because PHP defines the string version of false to be ". If you echo a collection of data (an array) you get the string 'Array' – a peculiar behaviour unique to PHP.

Echo can take more than one argument in a comma separated list. If you give it several, it will convert each to a string and output them in turn.

In between each comma you can write expressions. PHP understands that commas separate out expressions. As commas have a lower "precedence", they are evaluated last.

Precedence rules are complex and very few people learn them by rote, but the general rule is that the lower the precedence of an operator the more it separates out expressions in between it. In mathematics BODMAS is the precedence order for brackets/orders/division/multiplication/addition/subtraction.

The expression `2 * 3 + 3 * 5` should be read  $(2 * 3) + (3 * 5)$  because addition is a lower precedence, or "separates" expressions more than \* does.

A single quoted string is a literal string and will display exactly as seen.

## DATA TYPES

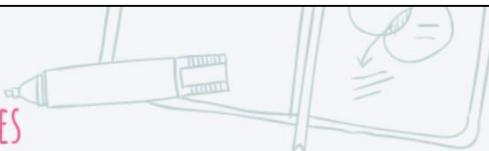


Variables can store data of different types, and different data types can do different things. PHP supports 8 different data types:

- String
- Integer
- Float
- Boolean
- Array
- Object
- Null
- Resource



## DATA TYPE EXAMPLES



DATA TYPE	EXAMPLE	COMMENTS
String	"Hello\nWorld";	Interpolated string
String	'Hello\nWorld';	Literal string
Integer	1	Any whole number
Integer	99	
Float	5.99	Any decimal number

A string is a sequence of characters, like "Hello world!". A string can be any text inside quotes. You can use single or double quotes. Strings using double quotes are "interpolated" meaning that substitutions are performed on them: what you write isn't exactly what the computer stores. For example, in the first string you write `\n` but PHP interpolates a newline here instead, so if you echoed this string, you would get a newline – not a backlash-n. Strings with single quotes are known as literal: meaning that they are literally what you type, here no newline is inserted by PHP.

There are two kinds of numbers in PHP: integers and floats (sometimes called doubles). Integers are any whole number, floats are any decimal number. The number 'one' can be represented as an integer or as a float. Floats are stored differently than integers and are less precise – even though both say '1' they have different characteristics. If you're doing financial calculations with PHP you should use a library such as bcmath.

An integer data type is a non-decimal number between -2,147,483,648 and 2,147,483,647.

An integer must have at least one digit and must not have a decimal point. An integer can be either positive or negative.

A float (floating point number) is a number with a decimal point or a number in exponential form.



## DATA TYPE EXAMPLES

DATA TYPE	EXAMPLE
Boolean	true
Boolean	false
Array	\$colours = ["Red", "Green", "Blue"];
Object	myCar
Object	yourCar



A Boolean represents two possible states: TRUE or FALSE. Boolean values are mostly used to represent either success or failure in an application or tell you whether conditions such as "age < 18" are met. These values are created with a keyword syntax, you type true not "true"; they aren't strings since they do not have quotes.

An array stores multiple values in one single variable. Arrays are collections of data, they're shown here for completeness. You will see more on arrays later.

An object is a data type which stores data and information on how to process that data. Objects are constructed in code from a class template. A class is a structure that can contain properties and methods. You will discover more about objects in the Object Oriented chapters later in the course.

## DATA TYPE EXAMPLES

Data Type	Example
Null	null
Null	\$x = null;

Use `var_dump()` to display information about your variables:

```
var_dump($colours);  
var_dump($x);
```

```
array(3) {
    [0] =>
    string(3) "Red"
    [1] =>
    string(5) "Green"
    [2] =>
    string(4) "Blue"
}
C:\xampp\htdocs\Chapter7\E.php:9:
NULL
Done.
```

Null is a special data type which can have only one value: NULL. Null means "Nothing" and is mostly used when something doesn't yet have a value. A variable of data type NULL is a variable that has no value assigned to it. If a variable is created without a value, it is automatically assigned a value of NULL. Variables can also be emptied by setting the value to null.

The special resource type is not an actual data type. It is the storing of a reference to functions and resources external to PHP. A common example of using the resource data type is a database call.

The PHP `var_dump()` function returns the data type and value of a variable.

## CONSTANTS

- A named value that gives meaning to your code
- Defined using the word const followed by a name
- Convention is to use all caps and underscores

```
const WordsPerPicture = 1000;  
  
echo 'A picture is worth ',  
    WordsPerPicture, ' words';  
echo 'The result is ', 6E7 - 1E5;  
  
const UK_POPULATION = 6E7;  
const UK_EMMIGRATION = 1E5;  
  
echo "\nThe result is ",  
    UK_POPULATION - UK_EMMIGRATION;
```

A picture is  
worth 1000 words

The result is  
59900000  
The result is  
59900000

Constants are like variables except that once they are defined they cannot be changed or undefined. A constant is an identifier (name) for a simple value. The value cannot be changed during the script. A valid constant name starts with a letter or underscore. You do not use a dollar (\$) sign before the constant name.

Often you will want to put numbers or text into your application. For example, you may want to test for an age being below a threshold: "age < 18". Here, 18 is an integer and the significance of the number is unclear. If 18 represents the maximum age allowed then "age < MAXIMUM\_AGE" is more meaningful.

Constants should be used wherever possible to give meaning to your code. Whenever you are typing in a string, integer, float, or Boolean, ask yourself if it would be better to use a constant.

Consider,

```
const ALLOW_CHILDREN = false;  
const MINIMUM_HEIGHT = 1.5;  
const ERROR_MESSAGE = "This rollercoaster is restricted to adults.:";
```

The syntax for a constant is straightforward: they keyword const, followed by the name of the constant, equals, whatever its value is going to be. A common naming convention for constants is all-caps separating words with underscores.

## CONSTANTS: USING DEFINE

- An older syntax is to use the define() function to create a constant

```
const WordsPerPicture = 1000;
```

```
define('WORDS_PER_PICTURE', 1000); //older
```

- The define function can also create constants using non-literal values: data can be manipulated with function calls within the definition

```
define('NAME', trim(' Sherlock Holmes '));
```

- Use this sparingly however as function calls tend to imply a value that will vary and is therefore not constant

You can also use the define() function to create constants. The syntax is a little older than using the const keyword.

Note: A function is a named group of instructions that can be called (used) repeatedly. You can create your own functions in PHP and there are lots of built-in functions to use too.

The trim() function removes any leading and trailing whitespace surrounding a string so in the example above.

' Sherlock Holmes ' becomes 'Sherlock Holmes' before being assigned as the value to the 'NAME' constant.

## 'PREDEFINED' CONSTANTS

PHP provides lots of built-in constants. Some useful ones are:

DIRECTORY\_SEPARATOR

/ on linux, \ on windows

PHP\_EOL

\n on linux, \r\n on windows

PATH\_SEPARATOR

: on linux, ; on windows

\_\_DIR\_\_

the present file directory

\_\_FILE\_\_

the present file path

PHP\_EOL will appear throughout the course examples.

This is the end of line constant.

```
echo ( 2 + 3 ), PHP_EOL;  
echo ( 2 ** 3 ), PHP_EOL;
```

5

8

PHP predefines a lot of constants.

For example, PHP\_EOL, which is defined to be the newline character on your operating system. On UNIX (Mac, Linux..) its \n on windows its \r\n.

DIRECTORY\_SEPARATOR is another useful one which is / on UNIX and \ on windows. A common trick however is just to use UNIX-style paths on both, since windows will often work with them.

Many of the constants PHP predefines are used for specific libraries, and for example, the constant PASSWORD\_DEFAULT is used with the password hashing library to specify using the default hashing system.

Note: The \_\_DIR\_\_ and \_\_FILE\_\_ constants use double underscores at the beginning and end.

## CASTING

PHP defines a correspondence between values of one type and values of another

- For example, we may ask for the boolean version of '0' or '1'

```
echo (int) 1.0;  
echo (int) 2.8;
```

1

2

```
echo (float) 1;
```

1.0

```
echo (bool) 0;  
echo (bool) 1;
```

false

true

- Casting is achieved by preceding a value of one type with the name of another enclosed in parentheses

PHP provides a way to move a value from one data type to another. This is called casting. If you cast a float to an integer PHP will throw away any decimal part of the number (it won't round it).

If you cast an integer to a Boolean, you get false for 0 and true for any other number.

In PHP almost all our input comes in as text from HTTP requests. We may need to perform mathematical calculations on this text so we would need to convert our data to an appropriate numeric type. PHP tries to cast values on your behalf as much as possible. If you use a number like it is a Boolean it will treat it as one and figure out if it means true or false.

PHP's casting rules will take a little while to learn and they're unintuitive if you come from most other programming languages. Since PHP is specialized for the web it tries harder than most other languages to make strings (text) easy to use.

## OPERATORS

TYPE	EXAMPLE
Arithmetic	<code>1 + 1;</code>
Comparison	<code>3 &lt; 5;</code>
Assignment	<code>\$a = \$b;</code>
Logical	<code>!true;</code>
Increment / Decrement	<code>\$x++;</code>

Operators are used to perform operations on variables and values. PHP has many categories of operators including arithmetic and comparison operators.

The most basic operations on types are 'operators'. For example, addition uses the plus '+' operator. Operators take operands or arguments which they use to determine a value. In the statement '1 + 2', 1 and 2 are the arguments or operands to the + operator.

PHP has another term for "operators with names": language constructs. The word "operator" tends to be reserved for symbols (e.g. +), so echo is known as a language construct.

Operators and language constructs will be introduced throughout this course.

## ARITHMETIC OPERATORS



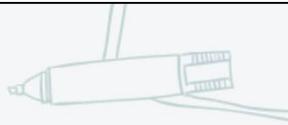
- Many of the usual mathematical operations are available

echo 1 + 1;	2
echo 1.2 / 3;	0.4
echo 2 - 2;	0
echo 3 * 3;	9
echo 8 ** 2; //PHP 5.6+	64

- In PHP 5.6 and onwards there is an exponentiation operator `\*\*` meaning "raise to the power"

The PHP arithmetic operators are used with numeric values to perform common arithmetical operations, such as addition, subtraction, multiplication, division, modulus and exponentiation.

PHP has a dedicated operator for exponentiation, `**` which is not seen in many other languages.



NOT(!)  
AND(&&)  
OR(||)

## Compare

- Values can be compared to return a Boolean result
- Boolean values can be combined using standard logical operators

```
!false;                                //NOT false
!!true;                                 //NOT (NOT true)
true && true;                            //AND
true || false;                           //OR
false || false;                          //Comparison Operators

3 < 5;
10 < 1 == false;
1 >= 1 == true;
2 <= 0 == false;
True == TRUE;                           //case-insensitive
FaLSE == false;
true && false == false;
```

Mathematical operators work on numerical values. Logical operators work on boolean values: && is Logical AND; || is Logical OR; ! isLogical NOT.

&& is true if both arguments are true, otherwise it is false. || is true if any of its arguments are true, otherwise false. And ! is true if its argument is false, or false if its argument is true (it flips it).

Logical operators are mostly used to combine conditions such as "be 18 AND over 5ft" or "have £10 OR a voucher" or invert a condition, "NOT under 18".

Comparison operators work on any value however they always evaluate to a Boolean: true, or false: < Less Than; > Greater Than; >= Greater than or equal to; <= Less than or equal to.

The most important comparison operators concern equality: == equal to; === identically equal to (equal in value and type).

PHP's equality operator, ==, will try all sorts of casts on your behalf, so that '0' == 0 == false is actually true, because '0' is 0 which is false if you cast them.

The identity operator compares value and type, so that '0' which is a string isn't 0 which is an integer.

All of the examples above return true.



## ASSIGNMENT OPERATORS



Many simple operators have a shortened syntax known as **compound** operators.

LONG-HAND	SHORTHAND	OPERATION
<code>\$a = \$b</code>	<code>\$a = \$b</code>	Assignment
<code>\$a = \$a + \$b</code>	<code>\$a += \$b</code>	Addition and assignment
<code>\$a = \$a - \$b</code>	<code>\$a -= \$b</code>	Subtraction and assignment
<code>\$a = \$a * \$b</code>	<code>\$a *= \$b</code>	Multiplication and assignment
<code>\$a = \$a / \$b</code>	<code>\$a /= \$b</code>	Division and assignment
<code>\$a = \$a % \$b</code>	<code>\$a %= \$b</code>	Modulus and assignment

The PHP assignment operators are used with numeric data to write a value to a variable. The assignment operator takes the value from the expression on the right-hand side of the equals and sets it into the variable on the left-hand side.

There are compound operators which enable you to perform an operation and a reassignment in a single short-hand statement.

`$a = $a + $b` which adds the value of `$b` to `$a` and then updates the value into `$a` is equivalent to the compound statement `$a += $b`

Other operators of this kind include,

- `$x += 10;`
- `$x -= 10;`
- `$x *= 10;`
- `$x /= 10;`
- `$x %= 10;`

## ASSIGNMENT OPERATOR EXAMPLES

```
$name = "Sherlock";           //assignment
$newN = $name . " Holmes";   //newN = Sherlock Holmes

$age = 26;                   //assign
$age = $age + 1;             //reassign, age = 26 + 1
$age +=1;                    //compound operator
$age++;                      //increment operator

$brName = "Mycroft";
$brName .= " Holmes";        //append Holmes

$brAge = 33;
$brAge += 1;                 //add-one to brAge

$brAge++;
$brAge--;                    //increment operator
                            //decrement operator
```



The \$newN variable in the example above is set to the value of \$name concatenated to the string " Holmes".

The statement \$brName .= " Holmes" is a concatenation and assignment compound operation.

To increase the \$age variable by 1 you can use the long-hand syntax \$age = \$age + 1 or the short-hand syntax \$age +=1 or the even shorter increment operator \$age++.

To decrease the variable by one, use the decrement operator – (minus minus).

## INCREMENT / DECREMENT OPERATORS

- Increment and decrement operators increase or decrease their value by 1 respectively
- There are two forms of operator:
  - Pre-fix               $++\$x$
  - Post-fix             $\$x++$

**++ --**

```
$x = 10;
```

echo --\$x; //subtract	<b>then</b>	echo # \$x - 1,	echo	9
echo ++\$x; //add	<b>then</b>	echo # \$x + 1,	echo	10
echo \$x++; //echo	<b>then</b>	add # echo \$x,	\$x + 1	10
echo \$x--; //echo	<b>then</b>	subtract # echo \$x,	\$x - 1	11
echo \$x;				10

Increment and decrement operators come in two forms: pre-fix and post-fix.  $\$x++$  and  $++\$x$  both increment  $\$x$  by one. However, if these operations are combined into a wider expression the order of operations is different.

$++\$x$  is a pre-fix operator and the value of  $x$  will be increased by one before it is used in the wider expression.

$\$x++$  is a post-fix operator and the value of  $x$  will be used in the wider expression before it is increased by one.



## \$ OPERATOR



- The \$ symbol has one further use for variables, it will take a string argument and look up a variable of that name
- Braces are sometimes used to be explicit about using the value of a variable as an argument to \$

```
$id = 'C';
$A = 'Joe';
$B = 'Fred';
$C = 'Lucy';
echo "{$id}<br>";
echo $$id; //shorter alternative
```

Lucy

```
$name = 'student';
$studentLocation = 'UK';

echo $$name . 'Location';
```

UK

The dollar sign (\$) is not only the first element of a variable name, it's an operator. If you pass a string argument to \$ it will look up a variable by that name. You can pass arguments to \$ most clearly by putting them in braces ({}).

So \${'id'} means the variable called \$id.

If you have a variable whose value is 'id' we can also use that.

```
$nameOfVariable = 'id';
```

\$\$nameOfVariable is then \$id.

You can even omit the braces: \$\$nameOfVariable.

Whilst this feature is not that often used, it demonstrates the power of PHP to treat strings as anything it likes: it can even treat them as the names of variables.

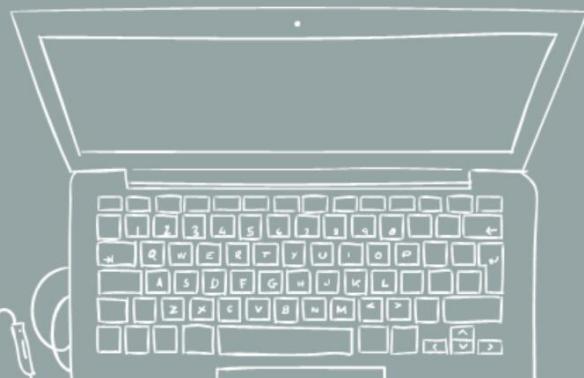


**Any questions?**





## EXERCISE 7



## RESOURCES

PHP Manual:

<http://php.net/>

PHP Documentor:

<https://phpdoc.org/docs/latest/references/phpdoc/basic-syntax.html>

# PHP CONDITIONALS

## CHAPTER OVERVIEW

- PHP Conditional Statements
- Introduction to functions
- Using `gettype()`
- Mixing types
- Conditionals and control flow
- If statement
- Switch statement
- Ternary, Short-Ternary and the NULL-coalescing operator

### Exercise

Create a Calculator program

Applications need to make decisions, and they do so by testing whether certain conditions are true or false. A common conditional test is to check whether user input data is valid. Web applications, including PHP, are particularly sensitive to user input: has it been provided? Is it valid? PHP has a very expansive notion of an "empty" or false value. It considers both 0 and "" to be false. This is useful as these kinds of values are often incorrect if entered by a user and PHP makes testing for them easy.

This chapter will introduce conditional syntax which provides the capability to test conditions and make decisions. To write successful conditional code it is necessary to understand what PHP considers true and what it considers to be false. This is often described as truth-like and false-like or truthy and falsey.

This chapter introduces the syntax for creating conditional statements.

Note: There is a chart at the end of the chapter which defines whether one value is equal (==) to another value. Try to get a feel for this chart and why PHP defines equality in this particularly loose way.



## PHP CONDITIONAL STATEMENTS



Used to perform different actions for different contexts or situations

- if statement
- if...else statement
- if...elseif...else statement
- switch statement

```
$d = date("D");
if($d == "Fri"){
    echo "Yay! It's nearly the weekend!";
}
```

**DETOUR**



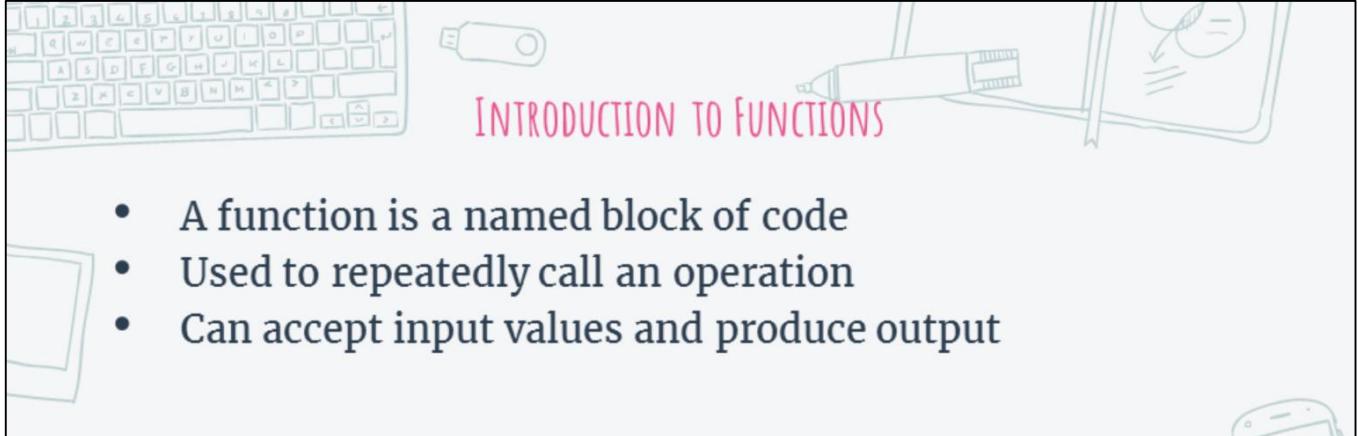
Conditional statements are used to perform different actions based on different conditions. Very often when you write code, you want to perform different actions for different situations or context. You can use conditional statements in your code to do this.

PHP has the following conditional statements:

- if statement - evaluates a single conditional test and executes some code if the condition is true
- if...else statement - executes a code block if a condition is true and another code block if that condition is false
- if...elseif....else statement - evaluates multiple conditions and executes a code block for the first true condition it evaluates
- switch statement - tests a value against a list of matches and executes the matching code block

The above example uses the date( ) function to extract the day portion of the current date. This value is then tested to see if it matches the condition "Is it Friday?". If today is Friday, an echo statement outputs a message. If today is not Friday, there is no output.

Conditional statements change the direction of your code and allow you to perform different actions depending on the current context.



## INTRODUCTION TO FUNCTIONS

- A function is a named block of code
- Used to repeatedly call an operation
- Can accept input values and produce output

```
echo ucwords('this function will capitalize words!');
```

```
This Function Will Capitalize Words!
```

```
echo str_replace('.', '!', 'Good Morning.');
```

```
Good Morning!
```

You have already seen a few PHP functions including date( ) and gettype( ). A function is a named block of code statements that can be used repeatedly in a program. To execute the code within the function you call the function.

PHP has more than one thousand built-in functions. You can also create your own functions which you will do in a later chapter. You will see examples of function calls in this chapter.

Examples of functions:

ucwords() takes a string and capitalizes the first letter of each word in that string.

str\_replace() replaces its first argument with its second argument inside the string that is passed as the third argument.



## INSPECTING THE TYPE OF VALUES

echo gettype("Hello\nWorld"); //simple types	string
echo gettype('Hello\nWorld');	string
echo gettype(1);	integer
echo gettype(1.0);	double
echo gettype(5E6);	double
echo gettype(true);	boolean
echo gettype(false);	boolean
echo gettype(null);	NULL
//complex types	
echo gettype([1, 2, 3]);	array
echo gettype(array(1, 2, 3));	array
echo gettype( function (\$input) { return \$input; } ) ;	object

You can use the built-in PHP function `gettype()` to discover information about the type of data passed as the parameter.

Above are examples of seven of PHP's eight datatypes. You will learn more about functions and arrays in later chapters.

The eight types are strings (textual data), integers (whole numbers), doubles (floating point numbers), booleans (true, false), NULL, arrays (collections of data), objects (groupings of data and behaviour) and the resource type. The resource type is a place-holder type which is only ever created by the PHP virtual machine and represents access to some resource (typically something like a file).

```
echo gettype(fopen('foo', 'w')), PHP_EOL; // resource
```



## MIXING TYPES

- PHP will try as hard as possible to make the code you write work, for example:
  - Combine strings and numbers using the add '+' operator
  - Combine booleans with numbers using the logical AND operator '&&'

```
echo '12a' + '1.2b';
echo "false" && true && 1;
echo true . 'Hello' . null . 1.2;
```

```
13.2
1
1Hello1.2
```

- PHP defines what will happen when you mix these types by following casting and operator precedence rules
  - Since concatenation (.) is an operation on strings, all concatenated values are first cast to strings
  - Since Logical-AND (&&) is a boolean operation, all operands are first cast to booleans

PHP will do its best to make sense out of your code, even if what you are trying to do isn't particularly sensible. PHP is a very loose language and is incredibly forgiving. Other programming languages are more strict and would return errors if you tried to do the above operations. In most scenarios it doesn't make sense to try to add the values of 12a to 1.2b however PHP will produce a value for you.

Note: it is obviously best to try and write sensible code and not rely on PHP to make sense of any nonsensical code.

## CONDITIONALS AND CONTROL FLOW

- So far programs have been executed linearly: one instruction at a time, every instruction
- Conditional or “conditional execution flow” can optionally perform instructions given a condition is true or false

```
$condition = true;  
  
if($condition) {  
    $value = "Yes";  
} else {  
    $value = "No";  
}  
  
echo $value;
```

Yes



```
$symbol = "Green";  
  
if($symbol=="Green")  
    $value = "Cross"  
} else {  
    $value = "Wait";  
}  
  
echo $value;
```

Cross

Between each set of curly-braces ( {} ) is a sequence of instructions. These instructions may or may not be executed.

If the condition evaluates to true the block that follows 'if' will be executed, otherwise the block following 'else' will be executed.

Other examples are:

- \$condition = \$age > 18;
- \$condition = \$password == "SECRET";
- \$condition = \$height > 1.5 && \$age > 13;

## CONDITIONS

Any value may be used as a condition for if-else statements

- The IF-branch is chosen if casting to a bool is **true**
- And the ELSE-branch if casting to a bool is **false**

```
if(1) {  
    echo "Y";  
} else {  
    echo "N";  
}
```

Y

```
if('Hello') {  
    echo "Y";  
} else {  
    echo "N";  
}
```

Y

```
if(1.1) {  
    echo "Y";  
} else {  
    echo "N";  
}
```

Y

```
if(0) {  
    echo "Y";  
} else {  
    echo "N";  
}
```

N

```
if('') {  
    echo "Y";  
} else {  
    echo "N";  
}
```

N

```
if('0') {  
    echo "Y";  
} else {  
    echo "N";  
}
```

N

Every value in PHP can evaluate to either true or false. Values which behave like `true` are sometimes called **truthy** and values that behave like false are called **falsey**.

For example, the empty string "", is a falsey value. If you put it in an if/else statement the else part will run. It behaves as though you had put false as the condition.

Most intuitively "there's something there" values are truthy: 1, 'Hello', 1.1 – all these values seem like a user has input something into a form.

Falsey values are those that feel like "there's nothing there", e.g. 0 and the empty string "". Somewhat peculiarly string zero '0' is also falsey. This is because 0 is falsey and PHP will cast strings to integers whenever possible.

It is quite important in PHP to know what is falsey (everything else is truthy), since the notion of something being "empty" or "not there" recurs when dealing with user data.

Note: the above examples use nonsensical conditions and are here to demonstrate the importance of understanding what PHP deems to be false-like (falsey);

## IF STATEMENT

- **else if** and **elseif** both allow for additional conditions
- Placement of the braces **{}** varies according to style

```
if(null) {  
    $value = "n";  
} else if (false) {  
    $value = "f";  
} elseif (0) {  
    $value = "z";  
} elseif (true) {  
    $value = "t";  
}  
echo $value;
```

```
if(null)  
{  
    $value = "n";  
}  
else if (false)  
{  
    $value = "f";  
}  
else if (true)  
{  
    $value = "t";  
}  
echo $value;
```

Both the keyword `elseif` and the keywords `else if` can be used to provide additional conditional branches in the if-statement.

The first option that evaluates to true has its code block executed. All other code blocks from the if statement are then ignored.

In both these examples null and false are skipped because they are falsey.



## SWITCH STATEMENT

The **switch-case** statement is useful for conditionally executing code when comparing one value to a list of possible values.

```
$testValue = get_from_user(); //hypothetical value

switch($testValue) {           //what is $testValue?
    case 'q':                //is it q?
        quit();               //code to perform if q
        break;                 //break out of the switch block

    case 'h':                 //is it h?
    case 'H':                 //OR is it H?
        help();                //code if h or H
        break;                 //break out of the switch block

    default:                  //if nothing has matched
        error();               //code if not matched to any value
        break;                 //break out of the switch block
}
```

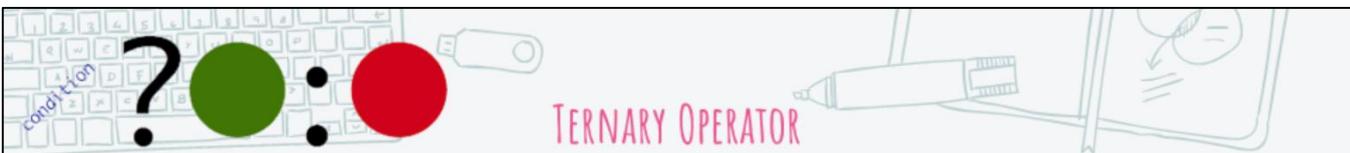
The switch-case statement is an alternative to the if-else statement. Whereas if-else tests any condition, switch-case is testing a value for a match.

The code above "switches" on \$testValue, comparing it to 'q', then 'h', then 'H'. If no match is found, "default" matches.

The code in a switch-case runs from the first match until the first break. So if \$testValue were 'h' or 'H', help() would run.

Once 'break;' is hit, PHP jumps out of the switch statement and continues the program from the final brace onward.

Note: get\_from\_user( ), quit( ), help( ) and error( ) are example function calls.



## TERNARY OPERATOR

- A more concise version of the if...else statement
- It has three operands:
  - A condition
  - A result for true
  - A result for false

```
$condition ? $trueValue: $falseValue;
```

echo true ? "Yes" : "No";	Yes
echo false ? "Selected" : "Not Selected";	Not Selected
echo "Hello" == 'Hello' ? 'Yes' : 'No';	Yes
echo (2 < 1 ? 'Two<One' : 'One<Two');	One<Two

The ternary operator is a more concise version of the if-else statement. It is called the ternary operator because it takes three operands - a condition, a result for true, and a result for false.

Some times you wish to select a value from two options given a condition. For example:

```
$message = null; //not yet set  
if($height > 1.5) {  
    $message = "you can go on the ride!";  
} else {  
    $message = "you cannot go on this ride!";  
}
```

Here all you are really doing is determining what the value of \$message should be. In this case, PHP provides the ternary operator:

```
$message = $height > 1.5 ? "you can go" : "you cannot go!"
```

The value of a ternary expression is the left-value if the condition is true, otherwise it is the value on the right.

## SHORT-TERNARY, TRUTHY AND FALSEY

- There is a shortened ternary operator which takes two arguments (despite ternary meaning three)
  - The expression evaluates to the first argument if it casts to true (truthy) and the second argument if it is falsey

```
echo 'No' ?: 'NotSeen';
echo '1' ?: 'Not Seen';
echo $username ?: "guest"

echo false ?: 'Yes';
echo null ?: 'Yes';
echo 0 ?: 'Yes';
echo '0' ?: 'Yes';
```



No	
1	
Student	
Yes	
Yes	
Yes	
Yes	

“0”, “”, null, 0 are all false in boolean contexts: falsey

The short ternary operator ?: is a form of the conditional operator which was previously available only as:

```
expr ? val_if_true : val_if_false
```

From PHP 5.3 onward it is possible to leave out the middle part, e.g. expr ?: val\_if\_false which is equivalent to:

```
expr ? expr : val_if_false
```

Expression expr1 ?: expr3 returns expr1 if expr1 evaluates to TRUE, and expr3 otherwise.

Consider a series of values: 1, 2

In this series the first truthy value is 1

Now consider this series: 0, 3

The first truthy value is 3. If you put zero '0' as a condition in an if statement it would be interpreted as false.

The short ternary operator does exactly what you've done above, except it isn't a comma, it is a ?:

```
echo 1 ?: 2;           // 1
echo 0 ?: 3;           //3
```

## THE NULL-COALEScing OPERATOR

The NULL-coalescing operator ?? can be chained in a series

- The value of the expression is the first non-null value
- Since the coalesce operator may be chained, it is easier to use than the short-ternary operator

```
echo null ?? null ?? 1;  
echo '' ?? 'NotShown';
```

1

```
echo null ?: 2;  
echo false ?: 3;  
echo '' ?: 'Shown';
```

2

3

Shown

- An empty string "" is non-null but evaluates to false

The above examples show the difference between the null-coalescing operator and the short-ternary operator. The null-coalescing operator can be chained together as many times as required however, the short-ternary operator is restricted to two outcomes. Notice however, that the two operators treat an empty string differently. It is classed as non-null by the coalescing operator but as a false value for the short-ternary operator. Be sure to test for an empty string in your code as it may be returned by the null-coalescing operator. Despite this feature the null-coalescing operator is typically easier to use than the short-ternary operator.

The goal behind the short-ternary operator is to let you select a default value if a variable is empty.

```
$username = $name ?: "guest";
```

However, it has the drawback that if the first argument is undefined it will throw a NOTICE error, e.g.

```
$username = $_GET['user'] ?: 'guest';
```

This will throw an error if there is no 'user' element of \$\_GET.

The null-coalescing operator does a much better job. Given a sequence of values it will pick the first truthy value without causing any errors:

```
$username = $_GET['username'] ?? $_POST['username'] ?? "Guest";
```

\$username will either be \$\_GET['username'] or \$\_POST['username'] or 'Guest'. If GET is empty it will be POST, if POST is empty it will be Guest.

Note: You will use these superglobals (\$\_GET and \$\_POST) in later chapters to extract information from a web page.

## NULL OPERATORS AND ERRORS

- The coalesce operator will silence error-prone arguments
- The short-ternary operator will not

```
<?php //eg.php  
error_reporting(E_ALL);  
  
echo $undefined ?: "Shown with Error\n";
```

```
echo $undefined ?? "Shown without Error\n";
```

```
Notice: Undefined variable: undefined in eg.php on line 4  
Shown with Error  
Shown without Error
```



The short ternary operator ( ?: ) is of limited use, since the first argument is often empty because it's an array index or undefined variable which causes a NOTICE error.

The null-coalescing operator will not throw a notice error so its use is preferred.

The `error_reporting(E_ALL)` statement above instructs PHP to emit all errors that are raised by the code.



## TYPE COMPARISON TABLE: X == Y



**TRUE column is truthy, FALSE column is falsey, NULL column is nully**

X, Y	TRUE	FALSE	1	0	-1	"1"	"0"	"-1"	NULL	array()	"php"	""
TRUE	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE
FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE
1	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
0	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE
-1	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
"1"	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
"0"	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
"-1"	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
NULL	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	TRUE
array()	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE
"php"	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE
""	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE

In PHP a value will be considered to be true or truthy if it "feels like it's data". And false or falsey if "it feels like its empty or false".

The table above compares values of all types against each other. For example, read down the left column, find the string "php" (second from bottom). Now read right. The title of each column is what the string "php" is being compared to. So "php" is TRUE, it isn't FALSE, it isn't 1, it is 0, it isn't -1 ....

You might be surprised that "php" is 0, however try "php" + 2 and you'll get 2. PHP will convert nonsense strings to 0. This will raise a minor error in PHP7 but not in earlier versions.

To find out what is truthy and what is falsey, read across the TRUE and FALSE rows. The top row, reading left to right tells you which values are truthy: true, 1, -1, "-1", "php"

The second row tells you which are falsey: FALSE, 0, '0', NULL, array(), "

If you look at the falsey set they all "feel empty": false, zero, null, an empty array, an empty string.

## OPTIONAL BRACES {}

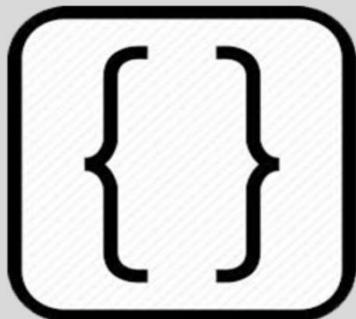
- In PHP control flow statements the use of braces is optional
- Within if, elseif and else statements the braces may be omitted if there is only one statement within them

```
if(3 > 1)
    echo 'Yes, 3 > 1';
else
    echo 'No, 3 < 1!';

if(3 > 1)
{
    echo 'Yes, 3 > 1';
}
else
{
    echo 'No, 3 < 1!';
}
```

Yes, 3 > 1

//braces  
Yes, 3 > 1



Above you can see the same code twice. However, in the first example the optional braces are not used but in the second example the braces are used. In this instance you would not see any difference in the program as there is only a single line of code that should be executed based on the conditional test. However, if a program needs multiple lines of code to run based on a condition then the use of braces is recommended. See overleaf.

## THE USE OF BRACES { }

- With braces {...} is recommended



```
if(false) {  
    mustBeAdmin();          //this does not run  
    doAdminThing();         //this does not run  
}
```



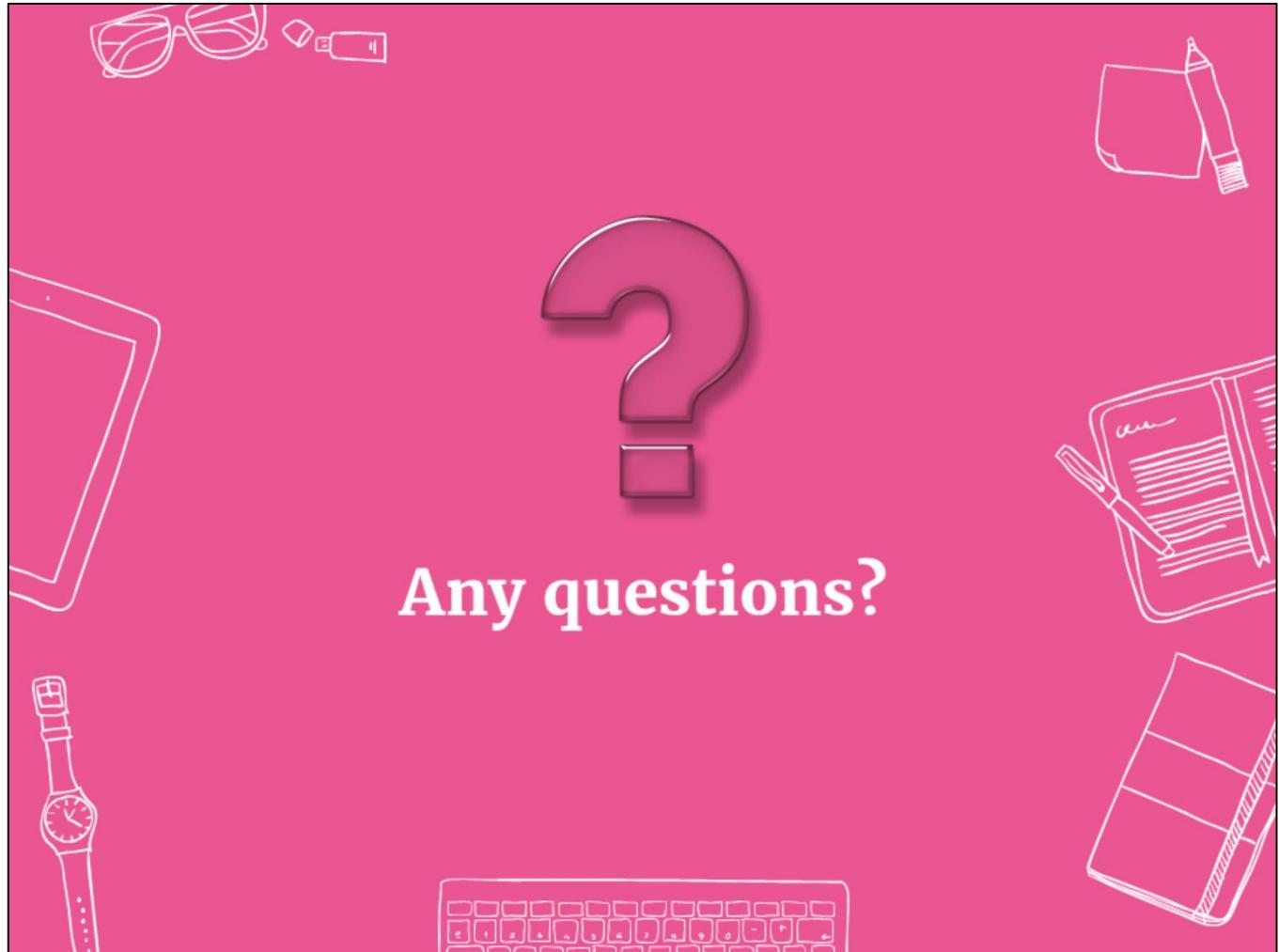
- Without braces – not recommended



```
if(false)  
    mustBeAdmin();          //this does not run  
    doAdminThing();         //this does!
```



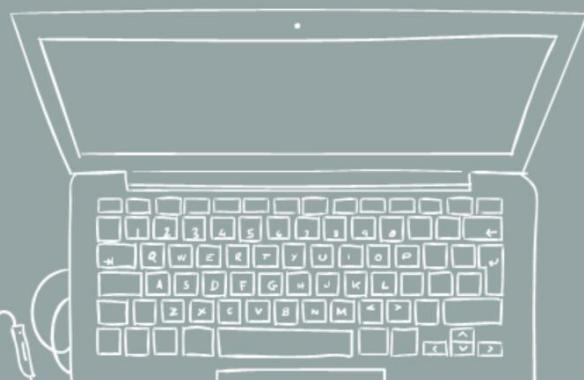
Braces are not needed for a single line of execution but are for a block of code that you would like to conditionally execute. You can see from the above example that a user would be able to perform administrator tasks even if they were not an administrator if the braces are omitted.



**Any questions?**



## EXERCISE 8



## RESOURCES

Type Comparisons

<http://php.net/manual/en/types.comparisons.php>

Ternary Operator

[https://wiki.php.net/rfc/isset\\_ternary](https://wiki.php.net/rfc/isset_ternary)

Control Structures

<http://php.net/manual/en/control-structures.if.php>

# PHP ARRAYS AND LOOPS

## CHAPTER OVERVIEW

- Arrays
- Sequential and Associative Arrays
- Multidimensional Arrays
- Loops
- Array Iteration
- exit() and die()

### Exercise

#### Loops and Arrays

A variable stores a single value. An array is a special variable, which can hold multiple values. PHP provides arrays to manage the various simple ways data can be grouped. Once data is bundled together in an array you will need techniques to unbundle the data. For this PHP provides loops. The foreach loop is especially helpful with arrays.

This chapter focuses on introducing the syntax for arrays and loops.

By the end of the chapter you should be able to describe what an array is and how you can iterate over an array.



## VAR\_DUMP()



- var\_dump() is a function which dumps information about a variable in a formatted manner
- It is mostly used during development for debugging

```
$l = "Baker Street";  
$m = null;
```

```
var_dump($l);  
var_dump($m);
```

```
$b = 3.1;  
$c = true;  
var_dump($b, $c);
```

```
var_dump(isset($m), empty($m));
```

```
var_dump('1' == 1, '1' === 1);
```

```
string(12) "Baker Street"  
NULL
```

```
float(3.1) bool(true)
```

```
bool(false)  
bool(true)
```

```
bool(true) bool(false)
```

The var\_dump() function displays structured information about one or more expressions that includes its type and value. Arrays and objects are explored recursively with values indented to show structure.

The function displays more information than echo. For example, "echo true" displays the value '1' whereas var\_dump(true) displays the value "bool(true)" .

For a string variable the function also displays the length of the string.

Note the use of the equality operators "==" and "===" , double equals and triple equals. The "==" operator is the equals operator and returns true if both operands are equal. The triple equals is known as the identical operator and returns true if both operands are equal and are of the same type.



## EMPTY AND ISSET



empty() and isset() are functions that can be used to test the value of a variable

- **empty()** returns true if the variable is an empty string, false, array(), "0", or is unset
- **isset()** returns true if a variable is set and is not null: even when its contents are falsey

Undefined variables are null

```
$l = "Main Street"; $m = null;  
$z= 0;  
var_dump($l);  
var_dump($m);  
var_dump($z);
```

```
var_dump(isset($m), empty($m));  
var_dump(isset($z), empty($z));
```

```
string(11) "Main Street"  
NULL  
int(0)
```

```
bool(false), bool(true)  
bool(true), bool(true)
```

empty() - Determines whether a variable is considered to be empty. A variable is considered empty if it does not exist or if its value equals FALSE.

empty() does not generate a warning if the variable does not exist. The following things are considered to be empty:

"" (an empty string), 0 (0 as an integer), 0.0 (0 as a float), "0" (0 as a string), NULL, FALSE, array() (an empty array), \$var; (a variable declared, but without a value)

isset() — Determines if a variable is set and is not NULL. isset() returns TRUE if the variable exists and has a value other than NULL. Otherwise, it returns FALSE.

empty() determines whether a value is falsey

isset() determines whether a value is not null or null-like

A variable can be deemed to be both set and empty. For example the values FALSE and "0" are both set and classed as being empty.

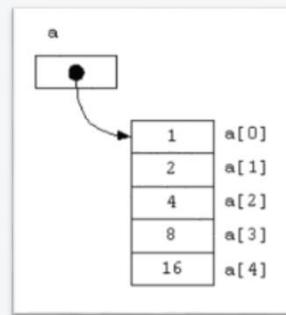


## ARRAYS



A special variable, which can hold more than one value at a time

- In PHP, there are three types of arrays:
- **Indexed arrays** - Arrays with a numeric index
- **Associative arrays** - Arrays with named keys
- **Multidimensional arrays** - Arrays containing one or more arrays



```
$numbers = array(1, 2, 4, 8, 16);
```

In PHP, there are three types of arrays:

Indexed arrays - Arrays with a numeric index

Associative arrays - Arrays with named keys

Multidimensional arrays - Arrays containing one or more arrays

Indexed arrays are zero-based; they start at index 0 by default.



## INDEXED ARRAYS



PHP has two main complex (multi-valued) types: array and object

- Arrays associate keys with values
- By default these keys are increasing integers from 0
- Known as indexed or sequential arrays
- Two syntaxes: **[x,x,x]** or the **array()** function

```
$pens = ['Parker', 'Waterman', 'Sheaffer'];           //new
$pens = array('Parker', 'Waterman', 'Sheaffer');       //old

$animals = ['Dog', 'Cat', 'Rabbit'];

$animals[2] = 'Mouse';      //Overwrite 3rd Element

echo $animals[0];          //Increasing integer keys
echo $animals[1];
echo $animals[2];           Dog
                            Cat
                            Mouse
```

An array is a collection of values. Each value is placed in a comma-separated list between square brackets. Alternative syntax is to use the `array` keyword followed by parentheses. The square-bracket notation is used through these guides.

The \$pens array contains three elements. Each element is assigned an integer index by default. The first element is 0 and each subsequent element increases by 1.

To access elements of the \$pens array follow the variable with the subscript operator ( [ ] ) providing the index you want to access. In the example above the element of \$animals at position 2 (the third element) is modified before each element is displayed.

You should pay careful attention to the square brackets here. When square brackets are at the end of a variable they are the subscript or "lookup" operator, and are accessing a value at a given index. When the square brackets surround a comma separated list they are defining an array.

It is important that you can distinguish between looking up a key in an array (subscript operator) versus defining an array. The square-bracket syntax was introduced in PHP5.3 (Dec 2009).



## ASSOCIATIVE ARRAYS

- The key for each value may be a value of any simple type
- arrays with non-sequential (e.g. string) keys are called associative arrays

```
$person = [  
    'name' => 'Sherlock Holmes',  
    'age'  => 27  
];
```

```
$brother = [];  
$brother['name']= 'Mycroft Holmes';  
$brother['age'] = 34;
```

```
echo $person['name'];  
echo ' is ', $person['age'];  
  
echo $brother['name'];  
echo ' is ', $brother['age'];
```



Sherlock Holmes is 27

Mycroft Holmes is 34

Arrays need not have integer indexes. You can specify the index type. To do so you use the fat arrow (`=>`) shown above.

Here `$person` is defined to have two elements, 'Sherlock Holmes' and 27 – however they are not at 0, 1 as with the previous array but at the indexes 'name' and 'age'.

This creates a kind of dictionary... or lookup table. The entry 27 is at the index age.

To access the name key in the `$person` array specify the key using the subscript operator ([ ]) e.g. `$person['name']`

To overwrite it, or create it if it doesn't exist, use the assignment operator (=) e.g. `$person['name'] = 'newName'`;

There's a lot of terminology surrounding arrays which is mostly interchangeable: index/key/lookup tend to mean the same thing; as so do: value / element / entry.

## MULTIDIMENSIONAL ARRAYS

A multidimensional array is an array of arrays.

```
$people = [  
    'Ben' => ['Location' => 'London', 'Height' => 1.81],  
    'Lucy' => ['Location' => 'Edinburgh', 'Height' => 1.65]  
];
```

Multidimensional Arrays

A multidimensional array is an array whose elements are themselves arrays. There are a number of ways of defining this array. You could also write:

```
$people = [];  
        //initialize to empty  
  
$people['Ben'] = [];  
        //initialize Ben element to empty  
  
$people['Ben']['Location'] = 'London';  
        //assign London at Location key of Ben  
$people['Ben']['Height'] = 1.81;  
        //assign 1.81 to Height key of Ben  
  
$people['Lucy'] = [];  
        //initialize Lucy element to empty  
  
$people['Lucy']['Location'] = 'Edinburgh';  
        //assign Edinburgh at Location key of Lucy  
$people['Lucy']['Height'] = 1.65;  
        //assign 1.65 to Height key of Lucy
```



## ALL ARRAYS ARE ASSOCIATIVE



In PHP all arrays are associative: you can mix the type of keys.

```
$seq = ['a', 'b', 'c'];  
  
//or you could specify integer keys  
$assoc = [  
    0 => 'a', 1 => 'b', 2 => 'c',  
];  
  
$mixed = [  
    0 => 'a',  
    1 => 'b',  
    'c', //PHP keeps counting!  
    'string' => '?!',  
    'd'  
];  
  
print_r($seq); print_r($assoc);  
print_r($mixed);
```

```
Array (  
    [0] => a  
    [1] => b  
    [2] => c  
)  
Array (  
    [0] => a  
    [1] => b  
    [2] => c  
)  
Array (  
    [0] => a  
    [1] => b  
    [2] => c  
    [string] => ?!  
    [3] => d  
)
```

PHP's arrays are actually a single unified data structure. If you do not specify a key for an element PHP will always assign it the integer that is next in sequence.

Look at the \$mixed array. The "next index in sequence" for the "c" element is 2. The array then contains a key called string! The last array element 'd' is automatically assigned then next sequential index value of 3.

PHP arrays which are sequential perform a little faster than arrays with mixed types of keys, so it is still worth distinguishing between "sequential" and "associative" types. However from the developer's point of view all arrays are basically the same.

An 'associative array', such as dictionaries or hashmaps, are distinct types in other languages. In PHP, the array type can be used as a simple sequential array, a dictionary or multidimensional array. PHP arrays are extremely flexible.

The print\_r( ) function is used to print human-readable information about a variable.

## ARRAY FUNCTIONS

- **print\_r()** - prints human-readable information about a variable

```
print_r(['Tea', 'Milk']);
```

```
Array  
(  
    [0] => Tea  
    [1] => Milk  
)
```

- **var\_dump()** - also prints information and provides type information
- **count()** – calculates the length of the array

```
echo count(['a', 'b']);
```

2

- The length or size of an array is how many elements it contains
- **sizeof()** is an alias of **count()** but rarely seen

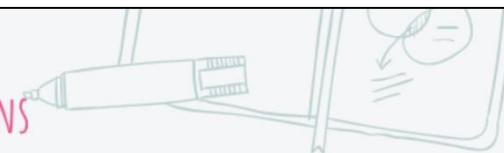
Here are some useful functions which operate on arrays.

Firstly, print\_r() prints the contents of an array in a human readable format (read as: print readable). print\_r() displays the values and the keys. var\_dump() is similar and tends to be more useful for debugging because it gives you a little more information that includes type information and if the value is a string it will also provide the length of the string.

The count() function tells you how many elements are in an array. The last element of a sequential array is therefore count() – 1 , because they are 0-indexed.



## ARRAY OPERATIONS



### Appending to an array

- Use an empty subscript operator [ ] on an array variable to append

```
$names = [];  
$names[] = "Sherlock";  
$names[] = "Mycroft";  
print_r($names);
```

```
Array (  
    [0] => Sherlock  
    [1] => Mycroft  
)
```

### Adding arrays

- Adding two arrays preserves the first array and adds in anything missing from the second. Useful for default configuration values

```
print_r(  
[ 'host'      => 'db.example',  
  'database'  => 'users' ] +  
  
[ 'port'      => 3306,  
  'host'      => 'localhost' ]);
```

```
Array (  
    [host] => db.example  
    [database] => users  
    [port] => 3306  
)
```



If you do not specify any key when you assign to an array php will use the "next in sequence" numerical index. In the example above \$names is created empty, then two elements are appended to the array, in sequence.

Arrays may also be combined. If you add two arrays \$a + \$b the resulting array will be \$a including any missing key-value pairs provided by \$b. That is, all the key-values in \$a will be in the final array and only the ones not in \$a from \$b.

Array addition is therefore not symmetric like numerical addition: whereas  $1 + 2 = 2 + 1 = 3$  with arrays \$a + \$b is not \$b + \$a .

You should read array addition as \$finalArray + \$missingKeys.



## UNPACKING ARRAYS



The **list()** language construct assigns items from an array into variables.

```
list($user, $location) = ['Student', 'UK'];  
echo "$user @ $location";
```

Student @ UK

The **explode()** function splits strings into sequential arrays, breaking them apart using the provided separator.

```
$name = "Sherlock Holmes";  
$names = explode(' ', $name);  
  
echo "$names[1], Mr. $names[0]";      Holmes, Mr. Sherlock  
list($first, $last) = explode(' ', "Sherlock Holmes");
```

You can combine **list( )** and **explode( )**

The **list( )** construct unpacks sequential arrays into variables. It assigns the zeroth element to the first variable in the comma-separated list, and the first element to the second, and so on.

The **list** construct is a form of multiple assignment, that is, creating and initializing several variables at once.

The **explode( )** function makes a sequential array from a string. In the example, the name "Sherlock Holmes" is split on the space so that "Sherlock" is the 0-element of **\$names** and "Holmes" is the 1-element.

The last expression assigns "Sherlock" to **\$first** and "Holmes" to **\$last** after splitting the string using **explode**.

## ISSET() AND EMPTY() WITH ARRAY KEYS

- Use **isset()** to determine whether a key exists

```
$item = ['name' => 'Camera', 'price' => 0];  
  
if(isset($item['price'])) {  
    echo $item['price'];  
}  
if(isset($item['sales_price'])) {  
    echo $item['sales_price'];  
}  
if(empty($item['price'])) {  
    echo 'No Price!';  
}
```

0

No Price!

- PHP will raise an error if you refer to a key that does not exist
- Use **isset()** to test for existence and **empty()** to test for a falsey value

There is a PHP function to test whether a key exists: `array_key_exists()`. However, this is a function call and a little more expensive than `isset()`.

`isset()` is a language construct that takes an expression between its parentheses and tests whether it is not null without issuing any errors.

`empty()` tests whether a value is falsey, `isset()` whether it isn't null. There's a subtle difference: see the example above. PHP considers 0 empty but not-null so 0 is empty and is set.

If your application does not need to distinguish between an array key that does not exist and one whose value happens to be NULL you should use `isset()` because it happens to be a little faster than the `array_key_exists()` function. However, if a NULL array key value and a non-existent array key are something you need to differentiate between, use `array_key_exists()`.



## CONSTANT ARRAYS



Arrays may be defined as constants.

```
const FILES = ['/home/qa/httpd.conf',
    '/etc/apache2/httpd.conf'];
```

```
print_r(FILES);
```

```
const ANIMALS = array('dog', 'cat', 'bird');
//or
define('ANIMALS', ['dog', 'cat', 'bird']);

echo ANIMALS[1]; // outputs "cat"
```

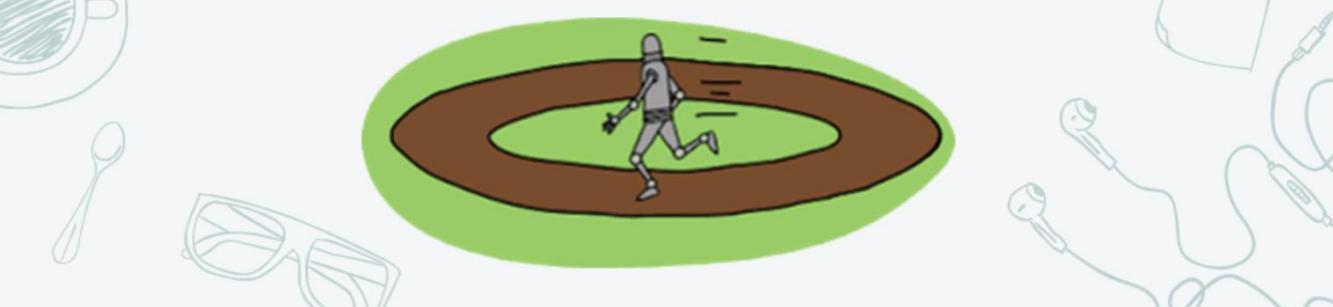


Since PHP5.6 it has been possible to define constant arrays.

## PHP LOOPS

There are four types of loops in PHP

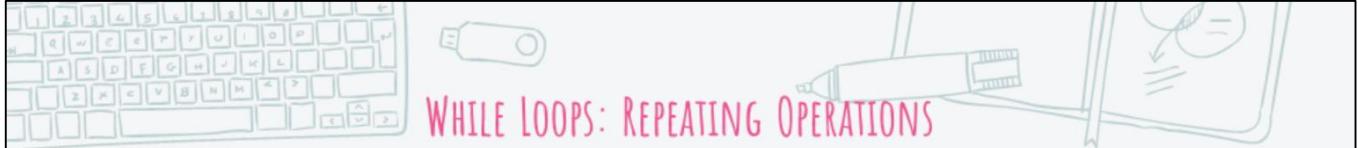
- **while** – loops through a code block as long as the condition is true
- **do...while** – loops through the block once, and then repeats as long as the condition is true
- **for** – loops through a code block a specified number of times
- **foreach** – loops through a block of code for each element in an array



In programming you often wish to repeat operations. Instead of copying and pasting code, you can use loops to perform a task like this.

In PHP, we have the following looping statements:

- while - loops through a block of code as long as the specified condition is true
- do...while - loops through a block of code once, and then repeats the loop as long as the specified condition is true
- for - loops through a block of code a specified number of times
- foreach - loops through a block of code for each element in an array



## WHILE LOOPS: REPEATING OPERATIONS

- The **while** loop executes its body (the code within braces, **{}**) while a condition is true.

```
$counter = 0;  
$max = 10;  
  
while($counter < $max) {  
    $counter++;  
    echo " $counter";  
}  
1 2 3 4 5 6 7 8 9 10
```

- Or until the keyword **break** is reached.

```
while(true) {  
    if(!$counter--) {  
        break;  
    } else {  
        echo " $counter";  
    }  
}  
9 8 7 6 5 4 3 2 1 0
```

The body of a while loop ( the code block **{}** ) will be repeated as long as the condition is true.

Therefore, the condition of a while loop is usually a variable or function call that will eventually return false (or go to 0, which is falsey). However, you could just use `true` as the condition in which case the while loop would go on forever (or until the program terminates).

You may recall from switch statements that the break keyword "jumps" a scope, here it does too: if you use break within a loop the loop will terminate and the normal flow of the program will continue.

## FOR LOOPS: REPEATING OPERATIONS

- Use a for loop to repeat a block of code an exact number of times
  - `for(initialization; condition; expression)  
{code to repeat...}`

```
for($counter = 0; $counter < 3; $counter++) {  
    echo " $counter";  
}
```

0 1 2

- You can use a for loop to iterate through an array and apply the same operation to each array element

```
$basket = ['Tea', 'Milk', 'Eggs'];  
for($i = 0; $i < count($basket); $i++) {  
    echo " $basket[$i]";  
}
```

Tea Milk Eggs

Use a for loop when you want to repeat a block of code an exact number of times. The for-loop structure is `for(initialization; condition; expression){code to repeat...}`

PHP will run the "initialization" once and it will continue looping while the condition is true. After each loop it will run the "expression" clause, that for example, will increment your counter variable, `$i` by one. It will then test the condition to decide whether to loop again. For loops can be used to iterate over sequential arrays since the loop counter you create exactly mirrors the sequential indexes of an array. The body of the for loop is then executed for every element of the array in turn, since `$i` changes with each execution.

Recall `count()` is a function which tells you how many elements are in an array. Here `count($basket)` is 3.

## FOREACH: ITERATING OVER ARRAYS

The **foreach** statement is the standard way of iterating over arrays in PHP

- The syntax is more natural than other loops for array iteration

```
$numbers = [1, 3, 5];  
  
foreach ($numbers as $number) {  
    echo $number * 2;  
}
```

2610

- The body of the foreach statement is executed once for each element of the array



In PHP while() loops and for() loops are not used very often, and very rarely with arrays. For arrays, PHP provides a specialised loop, the foreach loop.

In the example given above, the first variable in the foreach clause is the array: \$numbers, and the second variable is the name you wish to give each element as you iterate over the array: \$number.

\$number is then the value of each element of the array, a different one on every execution of the body of the foreach. The foreach loop always starts at the beginning of the array and finishes when it reaches the last element.

The \$number variable starts at 1:  $1 * 2 = 2$

On the next iteration \$number is 3:  $3 * 2 = 6$

On the next iteration \$number is 5:  $5 * 2 = 10$

As there are no line breaks or spaces in the echoed output, the result is 2 6 10 without any spaces or breaks.



## FOREACH WITH ASSOCIATIVE ARRAYS



To access the keys and values of the associative array specify variables in the foreach loop syntax.

**key => value**

For sequential arrays this key would be an integer.

```
$people = [  
    'Ben' => 'London',  
    'Lucy'=> 'Edinburgh'  
];  
  
foreach($people as $name => $location) {  
    echo $name, ' lives in ', $location;  
}  
Ben lives in London  
Lucy lives in Edinburgh
```

Associative arrays are made up of a named key and a value known as key-value pairs. You specify variable names for the key and the value in the foreach syntax. In the example above, the first variable in the foreach clause is the array you wish to iterate over, \$people. Then there are the names you wish to give to the key and value e.g. \$name => \$location. \$name is the key, \$location is the value. You can choose any valid names for the key and value such as \$k => \$v.

Upon each execution of the foreach block the next key-value pair is iterated over until the end of the array is reached.

Note that if \$people were a sequential array, then \$key would just be an integer index. In PHP associative and sequential arrays are effectively the same.

## FOREACH WITH MULTIDIMENSIONAL ARRAYS

To access the values in a multidimensional array use  
\$value[key].

```
$people = [  
    'Ben' => ['Location' => 'London', 'Height' => 1.81],  
    'Lucy' => ['Location' => 'Edinburgh', 'Height' => 1.65]  
];  
// foreach($array as $key => $value)  
foreach($people as $name => $attributes) {  
    echo $name, ' lives in ', $attributes['Location'];  
}  
Ben lives in London  
Lucy lives in Edinburgh
```

A foreach loop works in exactly the same way as for a single dimensional array except that the value (\$attributes) is itself an array. On the first iteration \$attributes is:

['Location' => 'London', 'Height' => 1.81]

On the second iteration \$attributes is:

['Location' => 'Edinburgh', 'Height' => 1.65]

To access one of the values of the \$attributes array use the key:

\$attributes['Location'] or \$attributes['Height'].



## CONTROL FLOW EXTRAS

**exit()** and **die()** terminate the script and report a value

- they are interchangeable however by convention die() is more often used when exiting with a string message and exit() with a number to represent a status code

```
//with computer processes:  
//0 means success, non-zero means failure  
exit(0);  
  
//strings are more useful on the web  
die("an error message");
```

- **continue** moves a loop forward skipping any code below it

```
foreach([1,2,3] as $number) {  
    if($number % 2) { continue; }  
    echo $number;  
}
```

2

The exit() and die() functions halt the execution of the script as soon as they are called.

die() is typically used with a string message as a temporary error handler. The exit function is used in production code by convention as it indicates a deliberate and permanent action, rather than a placeholder to-be-fixed later.

Note:

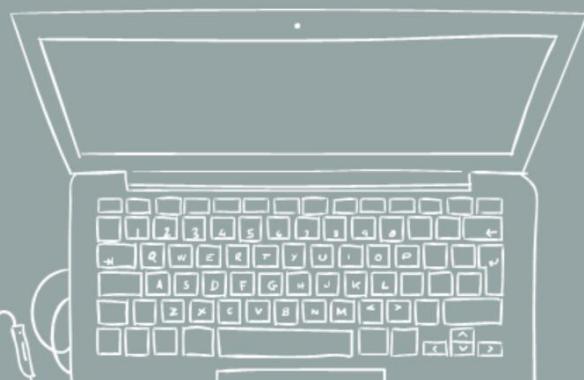
\$x % 2 means "the remainder when \$x is divided by 2" – this remainder is 0 (fasley) when a number is even so with any even number the if statement evaluates to a false value so the continue statement is not called and the code executes the echo statement. For any odd number there will be a remainder which is interpreted as a truthy value and therefore the if condition is satisfied so the continue statement is called. Continue skips the loop to the next iteration.



# Any questions?



## EXERCISE 9



## RESOURCES

Var\_dump()

<http://php.net/manual/en/function.var-dump.php>

Arrays

<http://php.net/manual/en/language.types.array.php>

List Function

<http://php.net/manual/en/function.list.php>

For loops

<http://php.net/manual/en/control-structures.for.php>

Foreach loops

<http://php.net/manual/en/control-structures.foreach.php>

# PHP FUNCTIONS

## CHAPTER OVERVIEW

- Create functions
- Function arguments
- Returning values
- Parameter passing
- Variadic functions
- Variable lifetimes and scope
- Appendix

### Exercise

#### Functions

You have seen many examples of functions, such as `print_r()` and `count()`. Procedural programs are programs which use these kinds of functions connected together by data. Procedural programming is about procedures. These procedures are known as functions in PHP. In this chapter you will discover how to define your own functions.

A function is a named block of code that can be called to repeat that operation whenever necessary within your logic. But PHP functions can also be passed around, just like any other value. This technique is very popular within the PHP community and some of the newest web frameworks make good use of this technique.

This chapter will explore each variety of function PHP has to offer and how you can use it.

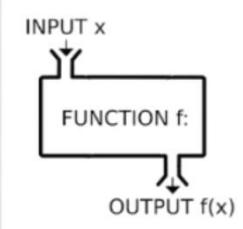
## PHP FUNCTIONS

- PHP's power comes from its functions
- There are more than 1000 built-in functions
- A named block of statements that can be used repeatedly in a program
- Does not execute immediately when the page loads
- Only executed by a call to the function

```
//function signature
```

```
function somefunction($arg1, $arg2) //argument list
    return $arg1 . $arg2;
}
```

```
echo somefunction('Hello', 'World'); //passing the args
```



A function is a lot like a recipe. Imagine a recipe to bake a cake: it would describe how much flour goes in, the number of eggs, and what processes to perform to make the cake. However, what if you wanted to make a much bigger cake? You would not want all of the measurements to be hard-coded. Instead, you could replace all of the fixed quantities (e.g. 100g flour) with named placeholders (\$flour) that tell the person reading the recipe they need to fill something in. When they've filled in all the values and baked it, they get out a cake.

Above you can see an example of the structure of a function plus an example of how you would invoke (or call) that function.

A PHP function only executes when a script explicitly invokes it.

## USER DEFINED FUNCTIONS

- User Defined Functions can be used when no built-in function can be found to do the job
- Declaration starts with the function keyword

```
function functionName () {  
    // Code to be executed;  
}
```

- Function names can start with a letter or an underscore, not a number
- Give your functions a name that reflects what they do
- Function names are not case-sensitive

A function is defined using the function keyword. Function names should represent the purpose of the function and usually start with a letter. Sometimes you will see developers use an underscore as the first character of their function name. This is allowed although numbers are not. Function names are not case-sensitive; myfunction, MyFunction and myFUNCTION all refer to the same function.

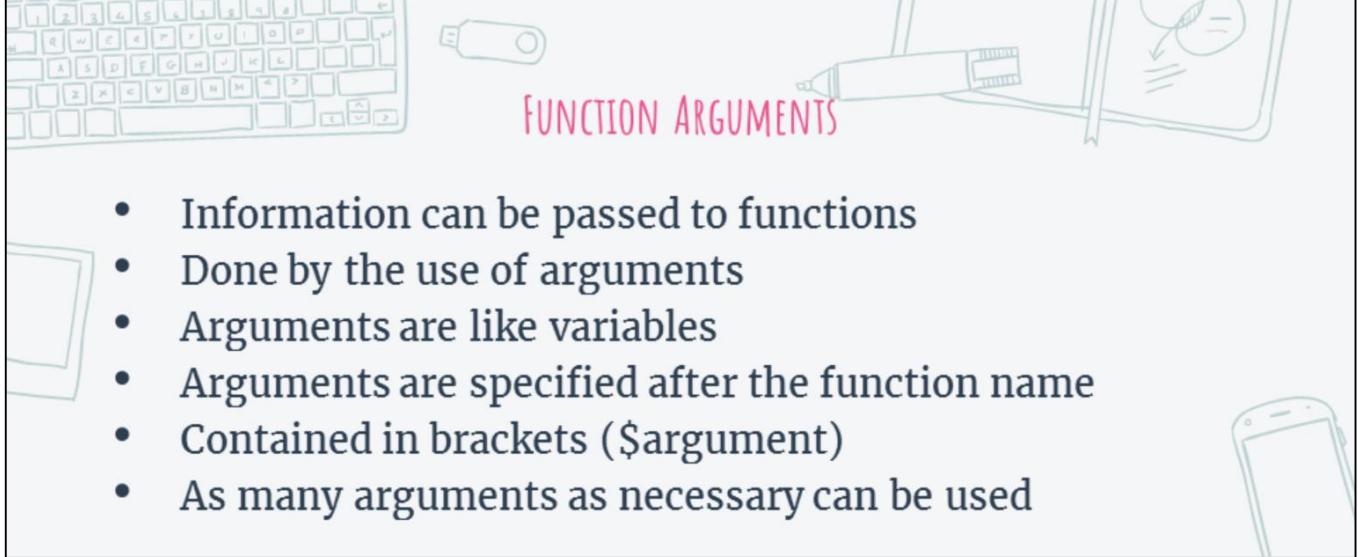
## USER DEFINED FUNCTIONS EXAMPLE

- Function called writeMsg is created
- Open brace { indicates start of code
- Close brace } indicates end of code
- writeMsg( ) indicates method call
- Program outputs “Hello world!” as function is called

```
<?php
function writeMsg() {
    echo "Hello world!";
}

writeMsg(); // Call a function
?>
```

The contents of a function are enclosed in braces { }. To call a function you use brackets ( ) after the function name.



## FUNCTION ARGUMENTS

- Information can be passed to functions
- Done by the use of arguments
- Arguments are like variables
- Arguments are specified after the function name
- Contained in brackets (\$argument)
- As many arguments as necessary can be used

```
function writeMsg() {  
    echo "No arguments";  
}  
function writeMsgWithArg($arg) {  
    echo "One argument";  
}  
function writeMsgWithArgs($arg1, $arg2) {  
    echo "Multiple arguments";  
}
```

Above you can see three different functions, each with a different argument list. The first function writeMsg() expects zero arguments. The second function writeMsgWithArg() expects a single argument (\$arg). The final function expects two arguments (\$arg1, \$arg2). Notice that you separate each argument with a comma.

You would typically use the arguments (values) passed in within your function code. For example:

```
function writeMsgWithArg($arg){  
    echo "This is the argument: " . $arg;  
}
```

## FUNCTION ARGUMENTS EXAMPLE

- The function has 1 argument (\$fname)
- When the function is called, a different name is passed in
- This name is used inside the function to add the first name to the last name on output

```
function familyName ($fname) {  
    echo "$fname Smith.<br>";  
}  
  
familyName ("Jane");  
familyName ("John");  
familyName ("Jenny");  
familyName ("Jack");
```

Above you can see the function being called repeatedly to add the last name of 'Smith' to each first name passed in. The function echoes their full name.

## MULTIPLE ARGUMENTS EXAMPLE

- This function uses multiple arguments
- The second argument is added with a preceding comma in both the function definition and the function call

```
function familyName($fname, $year) {  
    echo "$fname Smith. Born in $year<br>";  
}  
  
familyName("Jane", 1981);  
familyName("John", 1975);  
familyName("Jenny", 2008);  
familyName("Jack", 2011);
```

Above you can see the function being called repeatedly to add the last name of 'Smith' to each first name passed in plus their year of birth. You use a comma within the function definition between each argument (\$name , \$year) and also within the function call to separate the argument values ("Jane" , 1981).

## DEFAULT ARGUMENTS EXAMPLE

- If no value is given in the call, the default will be used
  - Default arguments must be the final ones in the argument list

```
function setHeight($min=50) {  
    echo "Height is: $min <br>";  
}  
  
setHeight(30);  
  
setHeight(); // Height will use default of 50
```

• **setHeight() has no argument therefore the default of 50 is used**

You can provide a default value for an argument within the function definition, for example, \$min=50.

If the function is called with a specified value, that value will be used. If the function is called without a value, the default value is used.

## RETURNING VALUES FROM FUNCTIONS

- Once a function has run, you often want to use the result
- The **return** statement is used to make the result of the function available to other parts of the code

```
function sum($x, $y) {  
    $z = $x + $y;  
    return $z;  
}  
  
echo "5 + 10 = " . sum(5,10) . "<br>";  
echo "7 + 13 = " . sum(7,13) . "<br>";  
echo "2 + 4 = " . sum(2,4);
```

You use the **return** keyword to pass data back out of the function, to the code that called the function.



## DEFINING FUNCTIONS: SUMMARY



- A function is a named block of code which denotes a sequence of instructions
- A function receives arguments as input and returns a single value as output
- When a function is called the values passed are assigned to the argument variables defined in the function signature
- The return value is specified by an expression which follows the return keyword



```
//function signature
function prep($prefix, $message) { //argument list
    return $prefix . $message;
}

$msg = 'hello world!';

echo prep('Logging: ', $msg); //passing the arguments
//becomes: $prefix      $message
```

The example above is for a function called prep. You can think of a function as a code recipe. The function has two placeholders: \$prefix and \$message. These placeholders are known as parameters. When you want to call the code recipe you need to specify the value of these parameters. In the example, the values passed to the function are 'Logging: ', as the first parameter, and 'hello world' as the second. It turns out prep() is just a code recipe for joining strings together. Its output is determined by the expression following the return keyword.

In PHP all function definitions begin with the function keyword followed by a name and a list of parameters. The parameters are placed in parentheses and separated by commas.



## DEFAULT ARGUMENTS WITH CONSTANTS

Arguments may be given default values supplied by a constant.

```
const PREFIX = 'Log: ';
$msg = 'hello world!';

function prepA($message, $prefix = 'Logging: ') {
    return $prefix . $message;
}//or
function prepB($message, $prefix = PREFIX) {}

echo prepA($msg);
echo prepA($msg, 'Printing: ');
echo prepB($msg); //implicit use of constant value
echo prepB($msg, PREFIX); //explicit use of constant value

Logging: hello world!
Printing: hello world!
Log: hello world!
Log: hello world!
```

Function arguments may be given default values so if they are not supplied when the function is called, PHP will use the default instead. Default parameters must always be last in the parameter list.

In the example above, if you call the prepA function without a second parameter then the default parameter value of 'Logging:' is assumed.

Default arguments are especially elegant when coupled with a constant or series of constants. In this manner you can access the value of the default argument via the const definition and you can use the constant explicitly if you want to make it clear that the constant value should be used.

```
echo prepA($msg, PREFIX);
```

## PARAMETER PASSING: BY VALUE AND BY REFERENCE

There are two principle ways PHP passes arguments

- By value – PHP supplies a copy of the value to the function, so the function cannot modify the original
- By reference – PHP supplies a reference to the original value, so the function can modify it

```
function byValue ($x) {  
    $x *= 2;  
}  
function byReference (&$x) {  
    $x *= 2;  
}  
  
$y = 10;  
byValue($y);  
echo $y;  
  
byReference($y);  
echo $y;
```

COPY

REFERENCE

10

20

The parameters of a function provide a way to pass data into the function and also to pass changed data out of the function. There are two mechanisms in PHP for passing parameters: by value and by reference. The default passing mechanism is by value. In this case, the parameter values are copied into the function. Even if the called function changes the parameter this will not impact on the original variable as it was merely a copy of the variable that was passed.

This is not so with pass by reference. If you prepend an argument with an ampersand (&), you indicate to PHP that whichever variable was passed when calling the function will be the actual variable modified by the function. In other words, the function operates on the existing variable you have defined rather than a copy of it.

Compare the `byValue()` and `byReference()` functions above. `byValue()` can not change the value of `$y` as the function gets passed a copy of it. The `byReference()` function, however, can change the value since the & "ties" the parameter `$x` to the actual variable `$y`. This "tie" is known as a reference.

## PHP NATIVE FUNCTIONS & BY REFERENCE

- Several of PHP's predefined functions use by reference argument passing semantics
- If you pass a variable by reference that does not exist, PHP will create it

```
function makeVariable(&$originalArray) {  
    $originalArray = ["Fill",  
                    "With",  
                    "Results"];  
}  
  
makeVariable($newVariable);  
  
print_r($newVariable);
```



```
Array (  
    [0] => Fill  
    [1] => With  
    [2] => Results  
)
```

Several native PHP functions take arguments by reference. A nice quirk of PHP's syntax is that if you pass an undefined variable to a function which takes an argument by reference, PHP will not issue an error. In the example above the variable \$newVariable is never defined, but it is passed to the makeVariable function and modified without error. This is referred to as an output parameter in other languages.

An example of a built-in function that uses by reference parameters is the sort function. This function sorts the array passed in rather than returning a new sorted array.

An example of a PHP function that creates and populates an undefined variable is the preg\_match() function. This function performs a regular expression match. It searches for a pattern in a given string and returns any matches it finds. If you pass a variable that does not exist PHP will create it and fill it with your matches.



## VARIADIC FUNCTIONS



- To enable a function to receive any number of arguments
- The packing operator ...\$x packs the arguments into an array
- Your variadic parameter should be the last parameter

```
function println(...$strings) {  
    echo implode($strings);  
}  
  
function printlnPrefix($prefix, ...$strs) {  
    echo $prefix . implode("\n{$prefix}", $strs);  
}  
  
println('This ', ' is ', ' a ', 'test', PHP_EOL);  
printlnPrefix('Recorded: ', 'An Error Message',  
'Another Error Message!');  
  
This is a test  
Recorded: An Error Message  
Recorded: Another Error Message!
```



The final parameter in a function definition may be proceeded by three dots (an ellipsis). This instructs PHP to collect the "extra" arguments that have been passed, put them into a sequential array and assign the array to the named elliptical parameter.

In the example above `println()` takes any number of arguments. Irrespective of how many are passed in, they are collected into an array called `$strings`, which is passed to the `implode` function. The `implode` function joins array elements with a glue string. The glue string defaults to an empty string and is therefore optional.

`printlnPrefix()` takes at least one argument but possibly more. Any additional arguments beyond the first are collected in to the `$strs` array.

In this manner functions may take a varying number of arguments and are thus known as variadic. The packing operator is known as the **splat operator**.

Note: `Implode()` joins an array of strings into one string using an optional glue separator. In a very unusual fashion, `implode()`'s optional argument comes first. When the `printlnPrefix` function is called with the parameter of 'Recorded' the `implode` function uses this as glue and will place it in-between each of the strings in the string array. The function also prepends the same value to the start of the output.

## OLD-STYLE VARIADICS

All functions in PHP are technically variadic:

- Use the `func_get_args()` function to create an array from the supplied parameters

```
function printlnOld($fixed) {  
    echo $fixed, PHP_EOL;  
    echo implode(',', func_get_args());  
}  
  
printlnOld(1, 2, "String!");  
1  
1,2,String!
```



Before PHP5.6 (Aug 2014) PHP lacked an argument packing operator / splat operator (...). However, from within the body of a function you can call `func_get_args()` which returns the passed arguments as a sequential array which you can then act on. You may see the `func_get_args()` function used in examples which is why it is included here.



## MAGIC CONSTANTS



PHP predefines some "magic constants" whose values change depending on where they are used.

```
function magic() {  
    print_r([__DIR__,  
            __FILE__,  
            __LINE__,  
            __FUNCTION__]);  
}  
magic();  
  
Array  
(  
    [0] => C:\...\Guides\07-PHP.iv  
    [1] => C:\...\Guides\07-PHP.iv\MagicConst.php  
    [2] => 6  
    [3] => magic  
)
```



PHP provides a large number of predefined constants to any script which it runs. There are eight magical constants that change depending on where they are used. For example, the value of `__LINE__` depends on the line that it's used on in your script. These special constants are case-insensitive and are `__DIR__`, `__FILE__`, `__LINE__`, `__FUNCTION__`, `__CLASS__`, `__NAMESPACE__`, `__METHOD__`, and `__TRAIT__`.

Each constant listed above acquires a different value depending on where it is used.

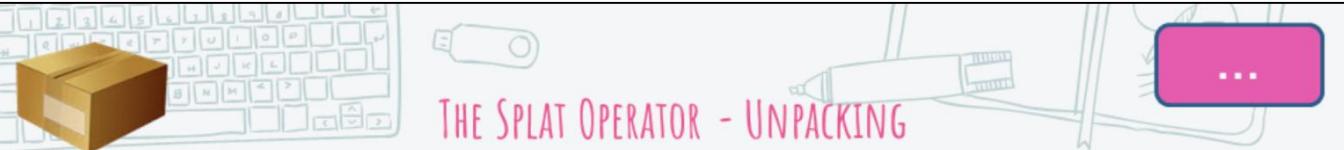
`__DIR__` and `__FILE__` are the directory and the full path and filename of the file they are used in.

`__LINE__` is the line number of the file this constant appears on.

`__FUNCTION__`, if used from within a function, is the name of the function.

The other magic constants refer to items you will see in the object oriented sections of the course.

The magic constants use double underscores in their names.



## THE SPLAT OPERATOR - UNPACKING

- Function definition: splat means pack
- Function call: splat means unpack

```
const SecretKey = 12;

function authenticate($user, $password) {
    if($password == SecretKey) {
        echo "$user is authenticated!";
    } else {
        echo "$user is not authenticated!";
    }
}

$details = ['Student', 12];

authenticate(...$details);      //syntax is PHP 5.6
Student is authenticated!
```

The splat operator, when used within a function definition, is used to pack up arguments into an array. The splat operator, when used within a function call, unpacks (or spreads) the arguments from an array.

Above, \$details is a sequential array. Calling the authenticate() function with "...\$details" instructs PHP to treat each element as an independent argument, i.e. as \$user and then \$password in turn.

This operator exists only to save you a little typing and isn't used very often.



## VARIABLE LIFETIMES AND SCOPE: GLOBAL

Variables defined inside functions live for the duration of the function call.

```
$count = 0;                                //the global $count
function counter() {
    $count++;          //this is a different variable!
                        //counter()'s $count
}
counter(); //create & destroy counters()'s $count
echo $count;
```

0

```
$count = 0;
function counter() {
    global $count;
    $count++;          //changing the global count
}
counter(); //this operates on the global
echo $count;
```

1

Whenever a function is called all of its variables are recreated. When the function call ends, all the function's variables are destroyed.

This "lifetime" boundary is known as a scope: anything within a scope is really in a private little world and may be destroyed more quickly than you expect.

Everything has a scope. The scope of the entire program, the outermost scope, is called "global" and it is created when the script runs and is destroyed when the script finishes.

In the example above \$count is defined in the global scope (the world of globals). Then again a variable with the same name is defined inside counter(). However, just as John Smith, London, UK is not the same as John Smith, New York, USA, neither are these two variables the same. The \$count in counter only exists while the function is being called.

Data, if it needs to leave a function should do so via the return keyword.

However, if you wish to allow a function to access the outermost, global scope, you can use the global keyword as shown above.

## VARIABLE LIFETIMES AND SCOPE: STATIC

- **global** changes the scope
- **static** changes the lifetime – so that variables which live inside functions are not destroyed when their function returns

```
$counter = 0;

function counter() {
    static $counter = 0; //initial value

    echo "counter()'s is ", $counter++;
}

counter();
counter();
counter();

echo "global is ", $counter, PHP_EOL;
```



```
counter()'s is 0
counter()'s is 1
counter()'s is 2
global is 0
```

global is a very powerful keyword within functions. It violates their standard lifetime characteristics (they refer to data that lives longer than a function call) and they also violate the privacy of scope: a variable named the same is the same.

The static keyword is a little less extreme. It forces a particular variable within a function to stay around for as long as the script is running, i.e. it ensures it is not deleted when the function returns.

However, unlike 'global', it does not violate the privacy of a scope: the \$counter in the function above is not the same variable as the \$counter outside it.

## STATICS AND GLOBALS ARE (TYPICALLY) BAD

- Functions specify their input using arguments and their output with a return value
  - When using globals and statics, the return value of a function depends on "hidden" state
  - This makes it difficult to reason about the behaviour of a function
- Globals "connect" several functions together, or regions of code, in a hidden manner
  - If several functions rely on a global or if a region of code modifies a global a function depends on, the entire behaviour of your codebase becomes difficult to predict
  - Small changes in one area "cascade out" unpredictably
- Globals and statics may be used very sparingly but it is almost always possible and better to avoid them

In contemporary PHP projects there are unlikely to be any uses of either the global keyword or static keyword within functions.

Functions are designed to take input, perform some operation and produce output. All the information a function needs to calculate its output should be clear from its input.

Globals and statics weave webs of state between functions. If functions do more than return values, but remember them, then you cannot predict from the input alone what their output will be.

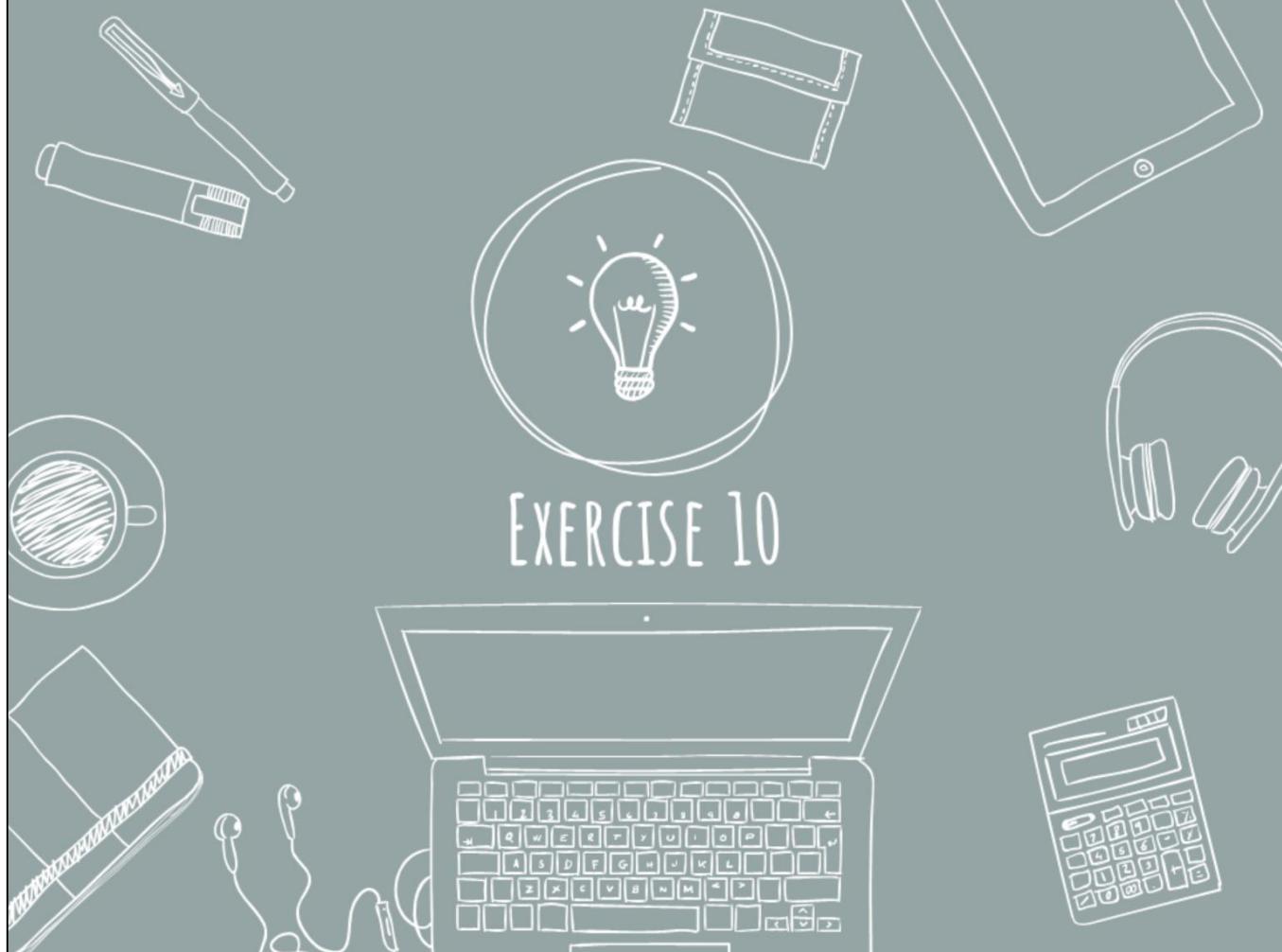
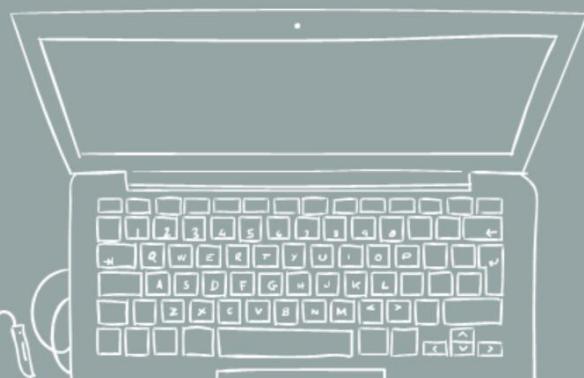
And if you use 'global' to dissolve the sphere of privacy around a function, to make its variables those of the script, you tie it closely to how that script is written and in general, make it very hard to understand. Why not, in any case, just pass the parameters a function needs?



# Any questions?



## EXERCISE 10



## RESOURCES

Functions

<http://php.net/manual/en/language.functions.php>

Arguments

<http://php.net/manual/en/functions.arguments.php>

Variable Functions

<http://php.net/manual/en/functions.variable-functions.php>

Returning Values

<http://php.net/manual/en/functions.returning-values.php>

# Appendix

## ANONYMOUS FUNCTIONS AND CLOSURES

An anonymous function has no name.

These functions can use variables defined outside themselves and capture their value. These are called closures.

```
$func = function ($str) {  
    return 'Log: ' . $str;  
};
```

```
echo $func("Error!");
```

Log: Error!

```
$prefix = 'Log: ';  
$func = function ($str) use($prefix) {  
    return $prefix . $str;  
};
```

//semicolon ;

```
$prefix = 'New Value';           //no effect!  
echo $func("Error!");
```

Log: Error!

An ordinary function definition creates a global identifier (the functions name) and assigns it a block of code, so that when we use the identifier a block of code is executed. For example, `function g() {}` defines the identifier 'g' which we can use directly: `g()`. Anonymous functions denote the same kinds of executable code: they are called, take arguments and otherwise behave like functions. However, they have no name. To actually call them you have to assign them to a variable.

In the example, the `$func` variable holds a function-object (i.e. anonymous function, or closure). A single variable holds this function, it is not bound to a global identifier. To call this function you must use the variable, as the call to `$func` shows.

Anonymous functions may also "capture" values from the scope they are defined in. When they do this they are called "closures". This capturing process copies in the value of the captured variable at the time the function is defined. The `use` keyword provides `$func` with a local copy of `$prefix` that is totally independent to the one outside.

Note: the function "closes over" (hence closure) or captures the `$prefix` value at the moment it is used. Subsequent changes to the variable have no impact. This should be familiar to javascript developers, however in JavaScript no `use` is required.

## CALLBACKS AND HIGHER-ORDER FUNCTIONS

Functions may take other functions as arguments, in PHP these are called **callbacks** since you call back to another function.

```
function formatter($str){ return 'Logging '.trim($str); }

function write($string, $loc, $callback) {
    file_put_contents($loc, $callback($string));
    echo $callback($string);
}
write('a message ', 'test.txt', 'formatter');

Logging a message
```

- For defined functions a string of the function name can be used
- Whenever parentheses follow a string variable in this way, it is treated as the function named by the string

```
$fn = 'formatter';
echo $fn('Error!');
```

Logging Error!

One of the main advantages of being able to treat functions like a variable is that you can pass them around. This means you can pass a function to another function by using the function name like a variable. Functions which take other functions as arguments (or return functions) are known as higher-order functions. The functions they take as parameters are called callbacks. In the above example, the function `write()` accepts three arguments. The first is a string to be logged, the second is a file name and the third should be the name of a function. The function will be called to manipulate the string before it is logged to the file. The built-in function `file_put_contents()` is used to create, write to, and close a file in a single operation. The echo statement outputs the formatted string to the output stream.

In PHP, if you treat a string variable as though it were a function (by following it with parentheses) it will actually call the function named by the string.

Therefore, the call to `write` passes, as its third argument, a string of 'formatter'. When the variable `$callback` is treated like a function (called with arguments in parentheses) the real function `formatter` is called instead. The PHP manual uses the terms "callback" and "callable" interchangeably, however, "callback" traditionally refers to a string or array value that acts like a function pointer, referencing a function (or class method) for future invocation.

## CALLBACKS: ALTERNATIVES

- We can use an anonymous function directly

```
function write($string, $loc, $callback) {  
    file_put_contents($loc, $callback($string));  
    echo $callback($string);  
}  
  
write(' a message', 'test.txt', function ($str) {  
    return 'Logging ' . trim($str);}  
);
```

Logging a message

- And PHP's predefined functions

```
write(' this is a message ', 'test.txt', 'trim');  
this is a message
```

The method of using strings as "pretend" functions is particularly characteristic of older PHP. In later versions the existence of anonymous functions makes passing functions around much easier.

In the above example, a function has been passed as the third parameter so there is no need for a separate 'formatter' method. This is an anonymous function; it doesn't have a name.

Note that PHP's mechanism for calling anonymous functions and stringly-named functions is the same: put parentheses after a variable and it will be treated as though it is a function.

Functions which require callbacks can be passed either function values you define, or the string name of the built-in function you wish to call.

This callback is useful as it allows the data to be manipulated before being written and the type of processing that occurs is decided by the caller of the function rather than being hard-coded within the function. The write() function knows it needs to write to a log file but wishes to allow the caller to specify a function to correctly format the message. It could, for example, ask for 15 different parameters each which pertain to some specific formatting option (e.g. colour, width, ...). However, asking for a function is much more flexible.



## ASIDE: RECURSIVE FUNCTIONS

Functions may call themselves

- However they need to stop at some point!
- When writing a recursive function start with the break-condition

```
function fact($number) {  
    if ($number < 2) {  
        return 1; //break condition  
    } else {  
        //recursive call  
        return $number * fact($number - 1);  
    }  
}  
  
echo fact(5); //5 * fact(4) == 5 * 4 * 3 * 2 * fact(1)  
120
```

Consider the function, `function f() { f(); }`. This is just an infinite loop; it continues to call itself forever. How can f refer to itself? Here you should think of the act of defining a function different to executing its body: by the time the function body executes (i.e. f gets called), it has already been defined. When defining a recursive function, you should start by specifying a condition which causes the recursion to stop. In the above example, there are two branches: an ordinary return and a return which calls the function itself. The return which calls itself will eventually hit the branch that doesn't, and thus stop the recursive process.

Note: As far as PHP web development goes, recursive functions are mostly a curiosity. They used to be required to perform a recursive traversal down a folder system but their usage has now been superseded by the introduction of the SPL's `RecursiveDirectoryIterator`. SPL is the Standard PHP Library.

# NAMESPACES & INCLUDE FILES

## CHAPTER OVERVIEW

- Using multiple PHP files
- Includes and the include path
- Namespacing

### Exercise Namespacing

PHP code is rarely limited to one file alone, but spans multiple: even in very simple applications there are files for templating, files for querying the database, files for processing data.

As applications become more complex there are files which merely define functions for other files to use, called libraries. PHP uses a very simple mechanism to coordinate multiple files, it simply allows you to join them together as though they were one big one. However, suppose in fileA you define a set of functions with the same names as those in fileB. If you were to merely join these files together PHP would produce an error: you cannot have multiple functions of the same name.

This is not a rare occurrence: every kind of data collection, for example a collection of users and a collection of products, tends to have create(), read(), update() and delete() capabilities. It would be convenient to have all the user-related functions in user.php and all the product-related functions in products.php but if you needed to use both files at once then you'd have two create()s!

In this chapter you will look at how PHP joins files together and how it provides a solution to the problem of clashing names. There are two main elements to this chapter: the include or "file joining" functionality and namespacing. The former is vital to understand for every kind of PHP application regardless of when it was written. PHP's include mechanism is one of its oldest and its use is ubiquitous. Namespacing arrived later in PHP (5.3, 2009) and it is ubiquitous in contemporary object-oriented PHP application development.



## INCLUDE



We often wish to separate code across files to logically group related functionality

- Use the **include** statement to syntactically join files together
- They are executed as if all the code appears in the same file

```
<?php // fileB.php  
  
const B = 'In File B!';  
const Name = 'Mary';  
  
<?php //fileC.php  
  
include 'fileA.php';  
include 'fileB.php';  
  
echo A . PHP_EOL;  
echo B . PHP_EOL;  
echo Name . PHP_EOL;
```

```
<?php //fileA.php  
  
const A = 'In File A!';  
const Name = 'Chandra';
```

**NOTICE**

In File A!  
In File B!  
Chandra

In PHP you can separate your code into multiple files and include those code files (with the `include` statement) to import their data and functionality into the currently executing script.

In the above example, two files (`fileA.php` and `fileB.php`) are being included in `fileC.php`. The use of the `include` statement brings the constants `A` and `B` into scope and enables you to reference them from within the script. Notice that the constant '`Name`' is duplicated across both included files. This results in a PHP Notice error because when PHP imports `fileB` it identifies that the constant '`Name`' has already been defined. You can turn off the PHP errors from inside the script (use `error_reporting(0)`), to see the above output. PHP uses the original value of the '`Name`' constant and outputs the value '`Chandra`' instead of '`Mary`'.

Each file you introduce should group related "items": functions, constants, and variables. A primary motivation for separating code out into several files is often that the file you are reading has grown so large and over-purposed that it is difficult to understand. In the example, `fileA.php` and `fileB.php` are passed as operands to the '`include`' language construct within `fileC`. There are several metaphors for the operation of the `include` statement: it is mostly correct to treat it as literally lifting all the code in, say, `fileA` and pasting it at the `include` point into `fileC`. Thus everything defined in `fileA` gets defined in `fileC` and likewise for `B`. Remember, if there are conflicts then there will be errors.

Note: `fileC` is the only file sent to the PHP interpreter (`PHP -f`). `fileA` and `fileB` merely live alongside it.

## INCLUDING FILES - ALTERNATIVES

- **Include** – file included if found
- **Require** – stops script if file not found
- **Include\_once** – safe for duplicated file names
- **Require\_once** – safe for duplicated file names but file must exist

( ! ) Fatal error:

```
include 'fileA.php'; //fileA included if found  
require 'fileB.php'; //fileB must be found or error
```

```
include_once 'fileA.php';  
include_once 'fileA.php';
```

As an alternative to the **include** statement you can use the **require** statement. The require statement is identical to the include statement except that upon failure to find the file it will produce a fatal error which will stop the script executing. The include statement only emits a warning if the file is not found and this warning does not prevent the script from executing.

To avoid PHP generating errors you can use the **include\_once** and **require\_once** statements.

The **require\_once** statement is identical to **require** except PHP will check if the file has already been included, and if so, not include (**require**) it again.

The **include\_once** statement is similar to the **include** statement, with the only difference being that if the code from a file has already been included, it will not be included again, and **include\_once** returns TRUE. As the name suggests, the file will be included just once.

In the above example, the second **include\_once** statement for **fileA** will not attempt to import the same file again. This is useful for code which spans so many files you may accidentally include a file more than once or when you might conditionally include something.

NOTE: If PHP cannot find a function it will issue a fatal error.

## THE INCLUDE PATH

The include statement accepts

- Full paths (e.g. `/home/qa/lib.php`)
- Relative paths (e.g. `../lib/until.php`)
- Or include-path relative paths

PHP defines an include path (in the ini)

- A series of folder names separated by `PATH\_SEPARATOR`
- Each folder will be searched in turn until the file is found
- If no file is found a PHP warning or error is raised

Modifying the include path is a very common part of "bootstrapping" an application

- E.g. to have include statements all relative to a 'lib' directory use `set_include_path()` and `get_include_path()`

```
$new = dirname(__DIR__) . '/' . 'my_lib_dir';
set_include_path(
    get_include_path() . PATH_SEPARATOR . $new
);
```

PHP needs to know where to find your included and required files. Files are included based on the file path given or, if none is given, the include path specified. The include path is defined in the configuration (ini) file. If the included file isn't found in the include path, PHP will finally check in the calling script's own directory and the current working directory before failing. If a path is defined, whether absolute (starting with a drive letter or slash), or relative to the current directory (starting with `.` or `..`), the include path will be ignored altogether. For example, if a filename begins with `..`, the parser will look in the parent directory to find the requested file.

The include path configuration option is a list of directories separated with a colon (for Unix) or a semicolon (for Windows). PHP considers each entry in the include path separately when looking for files to include. It will check the first path, and if it doesn't find it, check the next path, until it either locates the included file or returns with a warning or error. You may modify or set your include path at runtime using the `set_include_path()` function. The example uses `dirname(__DIR__)` to retrieve the full path of the parent's directory.

`PATH_SEPARATOR` is the OS specific separator which is a colon (`:`) on linux and a semicolon (`;`) on windows. Note: You can use `stream_resolve_include_path()` to return a string containing the resolved absolute filename (or `FALSE` on failure). The code example above uses the `set_*` and `get_*` include path functions to add an entry to the lookup locations. Updating the include path so that it contains the root directory of your application allows you to write "neat" include statements such as,

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

which may fully resolve to /home/qa/some/annoying/location/myfolder/file.php.



## USING INCLUDE FOR TEMPLATES

- Variables, functions, and constants defined in one file are immediately available when included
- HTML and non-PHP text is also output as usual

```
<?php //fileB.php ?>  
Hello <?php echo $name; ?>
```

```
<?php //fileA.php  
$name = 'Sherlock';
```

```
<?php //fileC.php  
  
include 'fileA.php';  
include 'fileB.php';
```

Hello Sherlock!

- This is useful for templating
  - Templates can use variables (e.g. \$name) defined elsewhere
  - Include the files which define the template variables and functions before the template

As a consequence of the cut-and-paste-like behaviour of the include statement, variables defined in one file are available as soon as that file has been included.

Thus in the above example, fileA defines a \$name variable and fileB echoes a \$name variable. fileC could define a \$name variable itself and then include fileB, or it could include fileA which provided a \$name variable. Either way, fileB will display whatever \$name variable it can find.

This suggests another motivation for separating code into files: one file has all the HTML templating syntax and the other defines the data upon which this file depends.

Note that you could swap out fileA with another file that defines a \$name variable and fileB would remain the same.

This principle is called the "separation of concerns"; a design approach to separate a computer program into distinct sections whereby each section is responsible for a specific set of information or functionality.

## USING INCLUDE INSIDE FUNCTIONS

- The file is included in the same scope as the include statement

```
<?php //fileA.phtml ?>  
Hello <?php echo $name; ?>
```

```
$name = 'Mycroft';
```

```
function display($template) {  
    $name = 'Sherlock';  
    include "$template.phtml";  
}
```

```
display('fileA');
```

Hello Sherlock

If you include inside a function, the "template" (included file) only sees the variables defined inside that function

Include is a very powerful operation: it references the entire contents of the defined file. Care must be taken to ensure conflicts and errors do not arise. To reduce conflicts you can control the scope of the included elements by using the include statement from within a function. The code from the included file is then contained by the function's boundary and shares its scope.

The file name referenced in the example is '**phtml**' rather than '**php**'. This is a convention used by many developers to signify that the file contains mainly html with little, if any, php code and in the most part it is presentation related.

Note: The above example could use the extract function to create variables within the scope of the function. The extract() function takes an associative array and makes variables of each key, e.g. extract(['age' => 27]); creates the variable \$age and sets it to 27.

The variables will be created within the scope of the function if you use the include statement within that function.

The example on the slide could be rewritten as:

```
function display($template, $vars)  
{ extract($vars); include "$template.phtml"; }  
  
display('fileA', ['name' => 'Sherlock']);
```

## NAMESPACES: THE MOTIVATION

- Programmes can contain thousands of identifiers: variables, constants, functions , classes, interfaces
    - sometimes it will make sense to give the same name to two distinct items
    - for example, if there is a "speak" function for cats and a "speak" function for dogs it seems as though we will require two "speak()"s
  - To avoid conflicts programmers used conventions to differentiate similar items
    - e.g. cat\_speak(), dog\_speak()
    - e.g. file\_write(), network\_write(), doc\_write()
- Can result in unwieldy and inconsistent identifiers
- CatSpeak or cat\_speak ?
  - blog\_view\_post\_pretty\_template()

**WOOF!**



A namespace is a way of encapsulating items to avoid name collisions. A name collision occurs when two or more

developers create a re-usable code element such as a function or a constant with the same name. The name you specify for an element may even clash with an internal PHP element. The use of

namespaces enables you to disambiguate between entities of the same name.

Namespaces also enable you to create a short-name, or alias, for a type with a long name, resulting in more readable code.

Suppose you put all your user data functions in user.php and all your product data functions in product.php. Then you include both in your script. Now you have two create()s; one for creating users and one for creating products. To avoid this clash you could give your functions more specific and unique names: create\_user() and create\_product(). Using this technique, however, can lead

to long and complex names as you may have to include the organisation name and application name within the identifier.

PHP introduces namespaces to solve this problem.

## CREATING A NAMESPACE

- Every name is defined in a scope
  - a name is an identifier for an item such as a function or constant
  - a scope is the *lexical region* (area of code) where the identifier can be accessed
- You can create your own named scopes
  - called namespaces: a *space for names*

```
namespace Inner {  
    const Name = 'Student';  
}
```

```
echo \Inner\Name;
```

Student

To refer to identifiers inside namespaces you use a path with the backslash character '\ as a separator

To define a namespace you write the `namespace` keyword followed by a name which will become your prefix. Everything within the namespace is automatically prefixed with the namespace's name.

In the above example, the `const Name` is actually called `\Inner\Name`. When you are outside the scope of the namespace where the constant is defined, you have to use its full name: `\Inner\Name`.

PHP namespaces look like windows-style paths as they use a backslash as a separator. Instead of creating a function called `create_user()`, you can create a function called `\User\create()` with the help of a namespace.

Why `\User\create` and not `\Create\user`? Well, it depends on how you wish to group code, but since a namespace "groups" everything within it by applying a common prefix, it typically makes more sense to group user-related functions than create-related functions. For example: `\User\Create`, `\User>Delete` and `\Product\Create`, `\Product\Delete`, etc.

## NAMESPACES AS PREFIXES

Namespaces are an easier way to apply a consistent prefix to your functions and constants than conventions

- Everything inside the namespace gets a prefix
- But you don't have to use this prefix *inside* the namespace

namespace Dog {		
function speak() {		
echo 'Woof';		
}		
speak();	//no prefix	
}	//speak() is \Dog\speak()	Woof
namespace Cat {		
function speak() {		MEOW!
echo 'Meow!';		
}		WOOF!
speak();		
\Dog\speak();	//prefix	
}		Meow Woof

Namespaces prefix functions and constants (but not variables which makes including templates easier).

Consider the Dog namespace above: the call to speak() is automatically a call to \Dog\speak(). PHP will always look in the current namespace to see if it can find a matching function.

Compare this with the Cat namespace. To call Dog's speak() you need to specify its full name, \Dog\speak() or else you would call Cat's version.

Here you can see the power of namespaces at work in differentiating functions (and other identifiers) that share the same name. When you define a function you give it a simple, clear name (e.g. create) and you let the namespace declaration (e.g. User) provide a prefix thus eliminating name collisions.

## NESTED NAMESPACES

- You can create a namespace to contain all of your code: `Outer`
  - Inside `Outer` you create an additional namespace: `Inner`
  - We place the constant `Name` in the namespace `\Outer\Inner`
  - From inside `Outer`, `Name` is at `Inner\Name` -- a relative path!

```
namespace Outer {  
    namespace Inner {  
        const Name = 'Student';  
    }  
  
    echo Inner\Name;  
} //the Outer prefix is implied!
```



Student

- Braces may be omitted to make the entire file one namespace

```
<?php  
namespace Outer\Inner;          //everything in this file  
                                //is prefixed  
const Name = 'Student';  
//Namespaces can be nested within one another, so that in the code above the full name of the  
//constant is \Outer\Inner\Name. Note however that from within the Outer namespace, the  
//constant can be referred to as just Inner\Name.
```

Omitting the first slash means that PHP will automatically look in the present namespace to find the item, i.e. it should add "\Outer" to Inner\Name. Echoing \Inner\Name would result in an error: \Inner\Name means a Name constant defined in the Inner namespace – there is no such constant. The actual constant is \Outer\Inner\Name . The first backslash is then like an absolute path, e.g. /home/uploads/file and without it it's a relative path, uploads/file .

If there is a single brace-less namespace statement at the top of a PHP file the entire file will be considered within that namespace, i.e. everything will be thus prefixed. To facilitate having nested namespaces using this syntax, it is valid (and encouraged) to specify an extended namespace path. For example:

```
namespace ExampleOrg\Cart\User;
```

Note that the leading slash is always omitted in namespace declarations. If they are at the top of the file then \ is implied.



## A NAMESPACE PER FILE

- On large code bases with lots of pure PHP files developers often use a unique namespace per file
- This prevents conflicts that result from including files that contain duplicated identifiers

```
<?php // fileA.php  
namespace QA\A;  
  
const Name = 'John A';
```

```
<?php //fileC.php  
  
include 'fileA.php';  
include 'fileB.php';  
  
echo \QA\A\Name . PHP_EOL;  
echo \QA\B\Name . PHP_EOL;
```

```
<?php //fileB.php  
namespace QA\B;  
  
const Name = 'Helen B';
```

John A  
Helen B

In large PHP applications it is usual to give every file its own namespace. Although multiple files can have the same namespace, this is rare.

With this convention, the include statement is a little more friendly. When you include fileA, everything defined therein is prefixed by its particular namespace, and so for B. Therefore, there will be no name clashes.

Note that fileC itself has no namespace. Namespaces are typically given to code which defines names (identifiers) rather than merely uses them and fileC here defines no types.

Note: You can see `include` is not a literal cut-and-paste, otherwise it would paste two namespace statements. Its actual mechanism is not easily described and pertains to how the PHP interpreter itself works. However, the cut-and-paste analogy is largely accurate.

## UNPACKING NAMESPACES

- A long prefix in front of every identifier can be cumbersome e.g. QA\People\Employees\CEO
- PHP provides the **use** keyword which *imports* a namespace

```
<?php // fileA.php  
namespace QA\People\A;  
  
const Name = 'John A';
```

```
<?php //fileB.php  
namespace QA\People\B;  
  
const Name = 'Helen B';
```

```
<?php //fileC.php  
include 'fileA.php';  
include 'fileB.php';
```

```
use QA\People\A; //removes the need for QA\People  
  
echo A\Name . PHP_EOL;  
echo \QA\People\B\Name;
```



John A  
Helen B

You have seen how namespaces can be used to avoid name collisions and also how a single namespace declaration provides prefixes on every identifier in a file once defined. You can also import a namespace to remove the need to fully-qualify an identifier.

The `use` statement, if it is given the name of a namespace, will remove everything but its last part. This eliminates the need to fully-qualify an identifier; you can now refer to an item by using the last part of the namespace and the identifier name only. In the above example, QA\People\A can be referred to as just A. Every mention of A from now on will be taken to mean QA\People\A.

Namespace B has not been imported with a use statement therefore you will have to refer to the name constant defined within by its full name: \QA\People\B\Name.

PHP has decided to preserve this "last part" because it is often the very last part which is sufficient for avoiding clashes, e.g. User>Create() vs. Product>Create() which both might exist within \Example\Cart\.



## UNPACKING NAMESPACES: PARTICULAR ITEMS

- Sometimes partially removing the prefix of every item in a namespace is not specific enough
- PHP allows you to `use` particular functions, constants, etc. to remove the entire prefix only for those items

```
<?php // fileA.php  
namespace QA\People\A;  
  
const Name = 'John A';
```

```
<?php //fileB.php  
namespace QA\People\B;  
  
const Name = 'Helen B';
```

```
<?php //fileC.php  
include 'fileA.php';  
include 'fileB.php';
```

```
use const QA\People\A\Name; //removes QA\People\A  
  
echo Name . PHP_EOL;  
echo QA\People\B\Name;
```

John A  
Helen B

The `use` statement can be very specific. You can write **use const** or **use function** with the full path of the constant or function and its namespace component will be entirely discarded.

In the example, because the `use` statement specifically refers to the QA\People\A\Name constant you may use the identifier Name to refer to the same value.

Compare these two kinds of `use`: if given a namespace it shortens it to its final element, if given a particular function or constant it provides that function or constant without any prefix whatsoever.

## THE GLOBAL NAMESPACE

- The default (unnamed) namespace is called the global namespace
  - It can be referenced using a leading backslash
  - Some built-in php identifiers can be overwritten within a namespace

```
<?php  
const Location = 'UK';  
  
echo \Location;  
echo Location;
```

UK  
UK

- You can define multiple namespaces per file using the brace '{ }' - style syntax. An unnamed namespace declaration is global
- Everything in `namespace {}` is global

```
<?php  
namespace QA { const Name = 'John A'; }  
  
namespace { echo QA\Name; }
```



John A

If you do not specify a namespace in your file, you are actually in "the global namespace". The global namespace is the root namespace from which everything else proceeds. Its name is the leading slash which has so far prefixed all your identifiers. So to access something specifically in the global namespace use a single backslash. However, since whenever you are in a namespace you do not have to specify its name, in the global namespace you do not need to specify the leading backslash .

If you want to have multiple namespaces per file, and you wish to have some code in the global namespace (i.e. unprefixed) then you can use a namespace declaration without a name, as shown above. This is very rarely done in practice. If you want code without a namespace prefix, simply create a file without one.

## ACROSS NAMESPACES

- Variables are not namespaced!
  - There is no `\A\B\C\$d` variable
  - Constants, Functions, Classes, Interfaces, Abstracts, Traits are namespaced
- The full path of any namespaced item begins from the global namespace, `\`
  - If you are in the global namespace you can omit the first slash
  - If you are in the same namespace as the identifier you can omit everything

```
namespace Parent {  
    function command() {}  
}  
  
namespace Parent\Child {  
    command(); //ERROR!  
    \Parent\command(); //CORRECT  
}
```



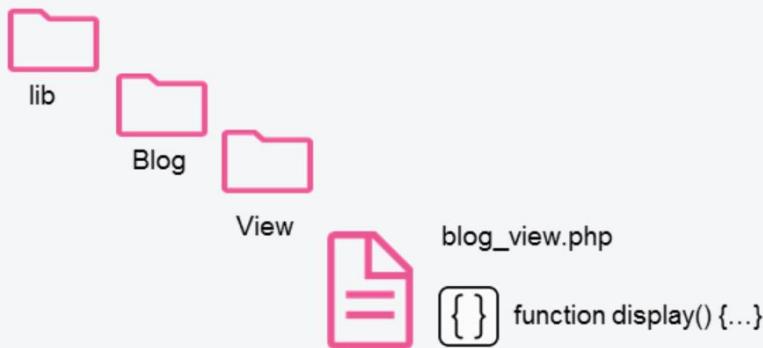
Recall that variables are not namespaced. Only definitions such as constants, functions, classes, interfaces, and traits are prefixed. You will encounter some of these types later during the object-oriented part of the course.

Note: Programmers from other languages might imagine there is a parent-child inheritance relationship between namespaces. In the above example, some might imagine that if PHP cannot find `command()` in `Parent\Child` it will look for it in `Parent`. Alas, it will not as they are two totally different namespaces.

This chapter has consistently used the term "prefix" to describe namespaces but this is to describe how PHP uses them. In PHP namespacing is essentially just a prefixing mechanism. Namespaces are merely additions to the names of identifiers, such as constants and functions.

## NAMESPACES AND DIRECTORIES

- Namespace paths look a lot like file paths
  - Connect namespaces to file paths to assist in locating files and functions
- Blog\View\display()



It is recommended that you repeat the namespace structure within the file path structure

Recall that `include` glues files together. Namespaces were introduced to keep this gluing safe: to partition code so that all of your identifiers had distinguishing prefixes.

However, including arbitrarily named files, e.g. "user.php" can be confusing. If you include user.php, what namespace do you get? You cannot tell from the file name. It is therefore a good idea to marry-up filenames (and paths) to the namespaces used within them.

Consider a file user.php which has the namespace ExampleOrg\Cart\User and within it a create() function. Do you really expect to infer the function name (ExampleOrg\Cart\User\create()) from the include statement alone (include user.php)? However, if you create a directory ExampleOrg with subdirectories of Cart and User to match the namespace structure then your include statement matches the namespace structure:

```
include 'ExampleOrg/Cart/User/user.php';
```

This makes it clear to see which namespace you might want to import:

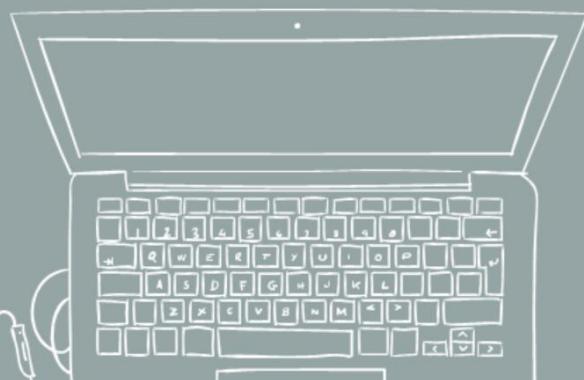
```
use ExampleOrg\Cart\User;
```



# Any questions?



## EXERCISE 11



## RESOURCES

Include

<http://php.net/manual/en/function.include.php>

Namespaces

<http://php.net/manual/en/language.namespaces.basics.php>

<http://php.net/manual/en/language.namespaces.php>

<http://php.net/manual/en/language.namespaces.rationale.php>

# PHP STRINGS

## CHAPTER OVERVIEW

- Strings as arrays
- Handling and formatting strings
- Comparing strings
- Using regular expressions in PHP
- Splitting strings

### Exercise

#### String Manipulation

Strings are one of PHP's most fundamental and most widely used data types. Everything which comes into a program is a string (user input via HTTP requests) and everything leaves a program as a string, for example as an echo statement.

It is therefore essentially impossible to construct a working PHP web application without a basic understanding of strings. If your website is to do anything beyond the trivial, you will need yet more sophisticated knowledge of this data type. This chapter covers syntax related to strings, functions related to strings and some additional numeric functions. The chapter also introduces regular expressions, often referred to as regex statements. A regex statement is a tool for parsing and understanding string data. Regex statements can be complex and an entire book would be needed to cover every detail. You will however, as a PHP developer, come across problems which require their use.

By the end of this chapter you should understand all the syntax surrounding strings and have a strong sense of what each string-related function does.



## STRINGS ARE ARRAY-LIKE



- Access single characters using [ ] notation

```
$name = 'logfile.txt';
echo $name[2];
```

g

```
$name[2] = 'c';
echo $name;
```

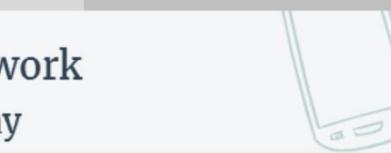
locfile.txt

- Functions which operate on arrays *might* work
  - use str\_split() to convert to an actual array

```
print_r(array_unique('aabbcc')); //ERROR!
print_r(array_unique(str_split('aabbcc')));
```

Warning: array\_unique() expects ...

```
Array (
    [0] => a
    [2] => b
    [4] => c
)
```



In PHP strings are like arrays of characters and can be accessed by position like an array, even on the left of an assignment.

However, even though strings might look like arrays in a certain light, you should not use array functions on them. For example, sizeof() gives the number of elements in an array, but does not give the number of characters in a string - use strlen() for that.

The array\_unique() function removes duplicate values from an array. It expects an array as a parameter and therefore does not work with strings unless you turn them into an array first. You can achieve this by using the str\_split() function. This function converts a string into an array.

## INTERPOLATION

- Braces {} may be used within strings to clarify which variable you mean to include

```
$location = "Italy";
```

```
echo "I am in ${location}";  
echo "I am in {$location}";
```

```
$animal = "dog";  
$animals = "sheep";
```

```
echo "I have many ${animal}s";  
echo "I have many $animals";
```

```
I am in Italy  
I am in Italy
```

```
I have many dogs  
I have many sheep
```



There are two principle kinds of strings in PHP: literal and interpolated. Literal strings use single quotes and are "read exactly as they are typed" and double-quoted strings interpolate (substitute) values. Interpolated, double-quoted strings, allow you to use variables within them. You may optionally use braces {} surrounding variables within strings.

In the example above, the variable \$animals is used in a string so that its value, "sheep" is interpolated. However suppose you wished to use the variable \$animal followed by an s. In this scenario braces can be used to specify exactly which variable you intend.

Braces may be positioned around variables as in, \${\$animal} or immediately after the dollar sign, as in \${animal}.

## STRING-RELATED FUNCTIONS

There are numerous string functions in the PHP library

- String Format Functions
  - `nl2br`, `printf`, `sprintf`
- String Information Functions
  - `str_word_count`, `strlen`, `strpos`
- String Manipulation Functions
  - `trim`, `str_replace`, `join`, `str_split`, `strtolower`, `strtoupper`, `substr`



There are numerous string handling functions in the PHP library. Many of them are prefixed by 'str'.

Some are simple and won't need much explanation. For example, `strtoupper()`, which converts a string to uppercase characters. The following pages explain some of these functions in more detail.

PHP's string handling capabilities are very powerful compared to many other languages. For example, there is a function to turn a string into title case that is not seen in many other languages. This function, `ucwords()`, is just one example of the power of PHP string-handling functions.

## STRING FORMATTING: NL2BR

- Searches for newline characters in a string
- Inserts an HTML line break in front of each newline
  - Requires a string as parameter
  - Returns the updated string with `<br />` in front of each "`\n`"

```
echo nl2br("First line.\nSecond line."); First line.<br />  
Second line.
```

- Useful for processing `<textarea>` data which contains "`\n`" and should often be displayed with HTML breaks

`<br>`

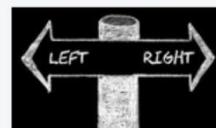
The `nl2br` function returns a string with '`<br />`' inserted before all new lines in the string. This function is commonly used to display any submitted message or data containing line breaks in the same spacing format.

Note that the new lines are preserved; `nl2br` does not replace newlines with HTML line breaks. Of course, the newlines are invisible on a web browser, since they are ignored.

## STRING MANIPULATION: TRIM

Strips characters, such as whitespace, tabs, and new lines, from the start and end of a string

- Takes a string to trim and optionally, a list of characters to strip
- Returns the trimmed string
- There are also **ltrim()** and **rtrim()** functions



```
var_dump(' £10.00 ');  
var_dump(trim(' £10.00 '));
```

```
string(7) " 10.00 "  
string(5) "10.00"
```

```
echo rtrim('example.com/path//', '/');  
echo rtrim('example.com/path', '/');
```

```
example.com/path  
example.com/path
```

- Useful for removing spaces from user input
- And removing "optional" characters (e.g. commas, slashes) to make input uniform

`trim()` is one of the most common string operations you will perform. User data is often full of leading or terminating extraneous spaces which `trim()` will remove.

You may also pass a specific set of characters to the trim functions (`trim`, `rtrim` – right trim, `ltrim` – left trim) such as slashes, exclamation points and commas and the function will remove those characters.

In the example above, `rtrim()` is used to remove any slashes on the right side of the url path. Note that `rtrim()` returns the same path regardless of whether the slash is present. Therefore, this technique can be used to make data with optional prefixes/suffixes uniform.

Note: If you do not specify the optional argument, the `trim` function strips the following characters:

- " " : An ordinary space.
- "\t" : A tab.
- "\n" : a new line (line feed).
- "\r" : A carriage return.
- "\0" : The NUL-byte.
- "\x0B" : A vertical tab.

PHP also contains the `ltrim` and `rtrim` functions, which strip the above-mentioned characters from the start and end of a string, respectively.

## STRING HANDLING: STRPOS

Enables you to find:

- The position of the first occurrence of a string inside another string
- Takes two parameters:
  - The main string (haystack) and the string whose position you need to find (needle)
- Returns the position as an integer

Useful for determining whether one string is inside another

```
echo strpos("Hello world!", "r");
```

8

```
var_dump(strpos("hello", "h") != false);  
var_dump(strpos("hello", "h") !== false);
```

bool(false)  
bool(true)



The strpos() function finds the position of the first occurrence of a substring in a string. The PHP manual calls these parameters the haystack and the needle. The function returns the numeric position where the needle exists within the haystack. Be aware that the string positions start at zero not at one. If the needle is not found the function returns false. In the second example on the slide, you are searching for the needle "h" in the haystack "hello". The function returns zero as "h" is in the zeroth position however a zero is also falsey. To accurately test whether "h" exists within the string you must use the identical operators (==) or (!==), rather than the equality operators.

== and != are the identically equals operators. The operands must match on both value and type to be considered equal.

## FUNCTIONS RETURNING 0 ('', ...)

Some functions can return *both* 0 and false

- **strpos()** returns the position of one string within another
- this is 0 if its position is at the beginning of the string
- and false if it does not occur in the string at all

In PHP if you wish to distinguish between the false and 0 cases use strict equality === or !==

```
var_dump(0 === false);  
var_dump(0 !== false);
```

```
bool(false)  
bool(true)
```

```
var_dump(0 == false);  
var_dump(0 != false);
```

```
bool(true)  
bool(false)
```

The strpos() function and others like it can cause a few debugging issues. The strpos() function tells you the position of one string within another, for example the position of 'a' in 'abc'. The position of 'a' is 0 . However, if the string is not present, the function returns false.

0 is a falsey value. Meaning that if you write

```
if(strpos('abc', 'a')) {  
    //...  
}
```

your code will not run. 'a' is in 'abc' at the zeroth position, however 0 will evaluate to false!

Therefore, if you are testing to see if one string is within another the recommended approach is to write,  
if(strpos(\$haystack, \$needle) !== false) {}

which uses the !== operator which compares value and type: it says "is not identically false" so that 0 will be a valid value.

## EQUALITY AND IDENTICAL OPERATORS

Equality operator `==` is often used to compare values

- Could be unexpected for numeric strings

```
$area_code = "0000000816";
```

```
if ($area_code == "816") {  
    echo "Kansas City!";  
} else {  
    echo "I'm not in Kansas!";  
}
```

Kansas City!

Identical operator `==` both sides have to be identical in value and type

- No "magical" interpretations (e.g. it won't strip zeros)



The most common way of comparing strings is to use double equals as a test for equality. As you can see, this works *most* of the time but not *every* time. PHP does some hidden magic with strings containing only numeric characters. Namely, it strips leading zeros. In some applications, such as telephone numbers, or a PIN, a leading zero could be significant, so `==` is not always exactly what you want.

The identically-equal-to operator, `==`, performs none of these magical interpretations and requires exact equality of value and type.



## STRING MANIPULATION: CHANGING CASE

- **strtolower()**
  - Convert the alphabetic characters of a string to lowercase
- **strtoupper()**
  - Convert the alphabetic characters of a string to uppercase
- **ucwords()**
  - Uppercase the first letter of each word in a string ("title case")
- **ucfirst()**
  - Uppercase first letter

```
echo strtoupper('hello brianna!');  
echo strtolower('Hello MARK!');
```

```
HELLO BRIANNA!  
hello mark!
```

```
echo ucwords('sherlock holmes');  
echo ucfirst('Sherlock holmes');
```

```
Sherlock Holmes  
Sherlock holmes
```

Both the strtoupper and strtolower functions accept the string to convert and return the converted string.

The ucwords() and ucfirst() functions are useful for tidying up user input, for example, blog titles that were not formatted correctly.



## STRING HANDLING: STRLEN

- Enables you to find the length of a string
- ```
$str = "As we know, there are known knowns; ".  
      "there are things we know we know. We also ".  
      "know there are known unknowns; that is to ".  
      "say we know there are some things we do not ".  
      "know. But there are also unknown unknowns --".  
      " the ones we don't know we don't know.";
```

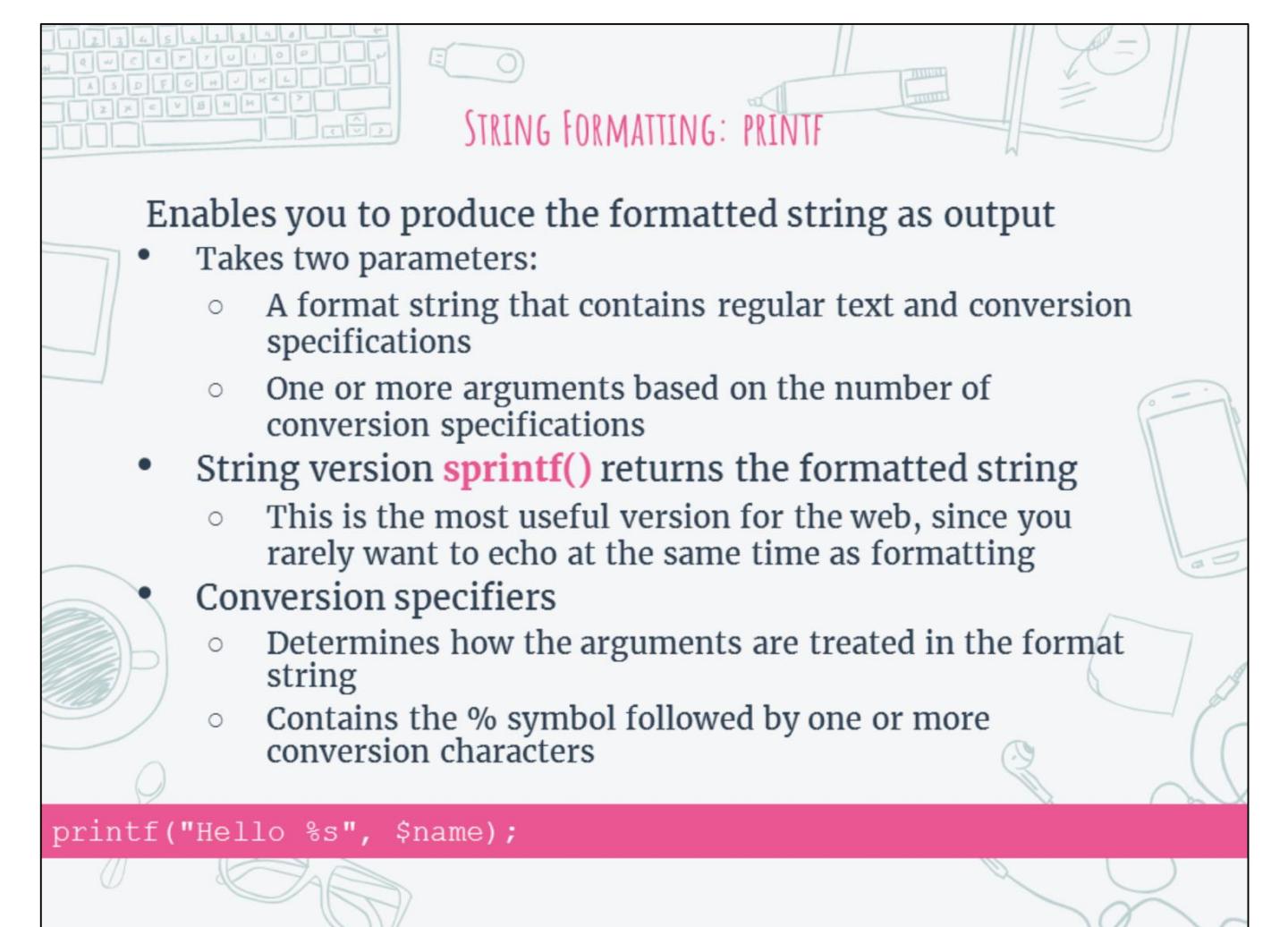
```
$length = strlen($str);  
echo "The speech is equal to $length characters";
```

The speech is equal to 246 characters



The `strlen()` function returns the number of characters in the string: letters, punctuation, numbers, etc. are included.

NOTE: The above quote is attributed to Donald Rumsfeld, former US Secretary of Defense



## STRING FORMATTING: PRINTF

Enables you to produce the formatted string as output

- Takes two parameters:
  - A format string that contains regular text and conversion specifications
  - One or more arguments based on the number of conversion specifications
- String version **sprintf()** returns the formatted string
  - This is the most useful version for the web, since you rarely want to echo at the same time as formatting
- Conversion specifiers
  - Determines how the arguments are treated in the format string
  - Contains the % symbol followed by one or more conversion characters

```
printf("Hello %s", $name);
```

The printf function produces a formatted string as output and returns either the length of the output string or a negative value if an output error has occurred.

The return value of printf() is rarely, if ever used since PHP will issue errors if you enter an incorrectly formatted string.

printf() and related functions see most of their use in command-line oriented PHP applications since on the web HTML and CSS are used to format text appropriately.

## STRING FORMATTING: printf (2)

Common conversion specifiers:

- **%d**: Treats the argument as an **integer** number
- **%s**: Treats the argument as a **string**
- **%f**: Treats the argument as a **float** (and rounds)

```
$planets = [ 'Mercury' => 57.91,
              'Venus'     => 108.2,
              'Earth'      => 149.597870,
              'Mars'       => 227.94 ];

/format = "%-10s %06.2f Gm\n";

foreach($planets as $name => $dist) {
    printf($format, $name, $dist);
}
```



|         |        |    |
|---------|--------|----|
| Mercury | 057.91 | Gm |
| Venus   | 108.20 | Gm |
| Earth   | 149.60 | Gm |
| Mars    | 227.94 | Gm |

The conversion specifiers shown on the slides are type specifiers. They specify how the argument should be treated.

The example format string contains the following specifiers:

**%-10s**      Left-justified, at least 10 character string, space padding

**%06.2f**      Right-justified, at least 6 characters, zero padding. One of the six characters is the decimal point, and the argument is rounded to two digits after the decimal point (see also the round() function)

Note that the justifications and widths use space padding by default, which is not much use with HTML. The example was generated on the command line, where it is significant. You might use printf with other media, like a report or output file. PHP also contains a sprintf function. You can also use this function for formatting strings. Both the printf and sprintf functions follow the same formatting rules but differ in the string types they return. While printf produces the formatted string as output and returns the number of characters of the string, sprintf returns only the formatted string. Use printf if you need to output a formatted string. If you need a formatted string for processing and not for output, use the sprintf function instead.

NOTE: The above example displays the distance in gigameters from the sun to the various planets.

## STRING HANDLING: SUBSTR

- Enables you to retrieve a part of a string
- Takes three parameters:
  - the string
  - a start position, zero is the leftmost, -1 is the rightmost
  - an optional length (defaults to end of string)
- Returns the sub-string specified by the arguments

```
// 0 1 2 3 4 5 6 7 8 9 10 11  
// H e l l o   W o r l d  !
```

```
$str = 'Hello World!';  
echo substr($str, 2, 3);  
echo substr($str, 2);  
echo substr($str, -2);
```

```
llo  
llo World!  
d!
```

The substr() function enables you to retrieve a portion of a string. You must provide two parameters. The first should be the string you are searching within, the second should be the start position. To extract a string from the left-hand side use a positive number. Zero is the leftmost position. To extract a string from the right-hand side use a negative number. Minus one is the rightmost position.

The third parameter, length, is optional. If you do not specify the length parameter, the substr function creates a substring from the start position to the end of the main string.

## STRING MANIPULATION: STR\_REPLACE

Searches a main string for a sub string

Replaces all occurrences of the substring

- Takes three parameters:
  - sub string to replace, the replacement string, the main string
  - or, array of substrings, array of replacements, array of strings

Returns the updated string (s)

```
$str = "We also know there are known unknowns; that\n".
       "is to say we know there are some things we\n".
       "do not know.".

echo str_replace('know', 'dig', $str),"\n";
```

```
We also dig there are dign undigns; that
is to say we dig there are some things we
do not dig.
```



The `str_replace` function can also take arrays—a search array, a replace array, or both—as parameters. If both the search and replace arrays are specified, the function takes a value from each array and uses the values to perform search and replace on the main string.

`str_replace` with arrays is often used as a basic templating (e.g. merge-style) placeholder-and-replace mechanism.

```
str_replace(['#date#', '#name#'], ['Monday', 'Sherlock'], $details)
```

This will replace the string `#date#`, if it occurs in `$details`, with the string 'Monday'. And so on for each element of the array.

Note: An optional fourth parameter specifies the number of replacements to make, but it is usually defaulted, which means “all occurrences”.

The function `str_ireplace` is a case insensitive version.

## COMPARING STRINGS

### String comparison functions

- Designed to be used as usort() helper functions
  - Return 1 if string1 > string2
  - Return -1 if string1 < string2
  - Return 0 if string1 and string2 are equal
- **strcmp(\$s1, \$s2)**
- **strncmp(\$s1, \$s2, \$maxlength)**

### Case insensitive comparisons:

- **strcasecmp(\$s1, \$s2)**
- **strncasecmp(\$s1, \$s2, \$maxlength)**

If you want a more sophisticated test use a Regular Expression



More predictable than the conventional operators (==) are the strcmp family of functions. These comparison operators are used to assist the sorting function: in order to sort data you must be able to compare the data items to make a decision on which data item should appear first and which should appear second.

The strcmp function is case-sensitive, so strcmp('PHP','php') will return a negative value. If you want to compare strings without considering case, use the strcasecmp function instead.

The strncmp and strncasecmp functions both have a maximum length argument that restricts the length of the comparison (from the left of the string).

These functions are, however, rarely used and somewhat defeat the spirit of PHP which is to rely on its liberal interpretation of values, its "added magic". However when you need to be exact, these functions exist.

Additionally, PHP7 introduces the "spaceship operator" <=> (combined comparison operator) which has similar semantics to strcmp(). See the link in the resources section for more information.

## REGULAR EXPRESSIONS IN PHP

- A Regular Expression:
  - Enables you to find instances of one string in another
  - Is a pattern that defines a portion of a string
  - Is used by a function to search and replace strings
- PHP uses PCRE – Perl Compatible Regular Expressions



Regular expressions, also called regexs, or REs, enable you to specify a pattern against which a string can be searched for substrings matching the pattern.

In many cases user input will be "regular", that is, it will follow some pattern.

For example, an article is a regular repartition of sentences each delineated by a fullstop. You could use this fact to pull apart an article sentence-by-sentence.

Many cases will be specific to your business or problem domain: for example, if you are writing a financial application then all financial input may follow the pattern CURRENCY SYMBOL + NUMBER, e.g. £10.00.

Regular expressions can be used to validate input: check that user input matches your expectations; break-apart input: retrieve only the hostname from an email address; or substitute input: replace all UK dates with US dates.

# PHP REGULAR EXPRESSION PATTERNS

## Elementary extended RE meta-characters

|           |                                            |                   |
|-----------|--------------------------------------------|-------------------|
| .         | match any single character                 | Character Classes |
| [a-zA-Z]  | match any char in the [...] set            |                   |
| [^a-zA-Z] | match any char <i>not</i> in the [...] set |                   |
| ^         | match beginning of text                    | Anchors           |
| \$        | match end of text                          |                   |
| x?        | match 0 or 1 occurrences of x              | Quantifiers       |
| x+        | match 1 or more occurrences of x           |                   |
| x*        | match 0 or more occurrences of x           |                   |
| x{m, n}   | match between m and n x's                  |                   |
| abc       | match abc                                  | Alternation       |
| abc xyz   | match abc or xyz                           |                   |

The examples listed above cover the most common regular expression meta-characters. If you are new to regular expressions, this is a good table to remember.

Regular expressions require hours of explication in themselves so this chapter will illustrate their basic usage to familiarize you with their capabilities.

## FINDING STRINGS: PREG\_MATCH

- Searches a string matching a regular expression
- Takes three parameters:
  - A PCRE regular expression
  - A string to search
  - An optional array variable that stores the matches

```
$string = 'marcus@sky.com';

if (preg_match('/.+@.+\.com/', $string, $results)) {
    echo "$string: valid email";
} else {
    echo "$string: invalid email";
}

print_r($results);
marcus@sky.com: valid email.
Array ( [0] => marcus@sky.com )
```



In the example on the slide, the preg\_match function uses a regular expression to search for 'one or more characters'(.+), followed by '@', followed by 'one or more characters'. This is followed by a full-stop character - 'escaped' so that the special meaning (one of any character) is ignored, then the characters 'com'.

The preg\_match() expression therefore returns true if a string contains an at-symbol (@) and ".com" in order. This is quite a particular test and only illustrates something you may wish to accomplish with regex.

When using the PCRE functions, it is required that the pattern is enclosed by *delimiters*. A delimiter can be any non-alphanumeric, non-backslash, non-whitespace character. Often used delimiters are forward slashes (/), hash signs (#) and tildes (~). In the above example, slashes are used. It is also possible to use bracket style delimiters where the opening and closing brackets are the starting and ending delimiter, respectively.(), {}, [] and <> are all valid bracket style delimiter pairs.

The preg\_match function returns either the length of the matched string or it returns False if no matches are found or if an error occurs. If you do not specify the optional array variable, or if the length of the matched string is 0, then this function returns 1. The preg\_match function is commonly used to verify fields submitted via a form.

## REPLACING STRINGS: PREG\_REPLACE

- Searches a string matching a regular expression
  - Returns the matched string with a replacement string
- Does not alter the searched string
- Takes the following parameters:
  - A PCRE regular expression
  - A replacement string
  - A string to search

```
$string = 'copyright';
echo preg_replace('/right/', 'left', $string);      copyleft

$string = 'hit number 26';
echo preg_replace('/[a-zA-Z]/', '', $string);        26
```

You can use the preg\_replace function to search and replace text. For example, you can search for all URLs present in your script and replace them with hyperlinks. Similarly, you can replace all copyright information present in scripts with updated information.

The preg\_replace function shown in the example updates "copyright" to "copyleft" indicating a change of licensing.

The second preg\_replace function removes all alphabetical characters of a string by replacing them with an empty string.

The replacement is applied globally, for example:

```
$string = 'I like tea because tea is nice';
echo preg_replace('/tea/','coffee', $string);
```

returns:

I like coffee because coffee is nice

However, str\_replace would have been more appropriate here. Generally, using the regular expression functions is slower than using string functions, so only use them when necessary.

NOTE: In the above example the space between 'number' and '26' is not replaced by the preg\_replace function. To replace it include a space within the regex expression after the 'z' and before the close square bracket ']':

```
echo preg_replace('/[a-zA-Z ]/',' ', $string) .
```



## GROUPING AND BACK-REFERENCES

- Parentheses group characters and apply quantifiers
  - the optional third argument is used to store the matches

```
preg_match('/(.+)@(.+\..com)/', 'name@example.com', $ms);  
print_r($ms);
```

```
Array (  
    [0] => name@example.com  
    [1] => name  
    [2] => example.com  
)
```

- PHP also allows ‘back-references’
  - Based on the content of the text enclosed in parentheses
  - Indicated by \n, which represents the ‘nth’ set of parentheses

```
echo preg_replace('/(\w+) (\w+)/',  
                  '\2, Mr. \1',  
                  'S Holmes');
```

Holmes, Mr. S

`preg_match()` can also be used to extract information from a string. In the above example, the `$ms` array contains particular parts of the email string supplied. This is because certain elements of the regex have been placed within parentheses.

The regex groups in parentheses are also known as a capturing parentheses group. In the (optional) array, `$ms`, the printed elements are: 0 - the whole match, 1 - the text matching the first parentheses group, 2 - the text matching the second parentheses group, and so on.

Note: PHP and PCRE support the use of parentheses to group a number of characters or regular expressions into a single unit and then apply a regular expression quantifier to the entire group. This can be useful when the pattern consists of recurring blocks of text or words.

NOTE: The quantifiers are zero-based with the zeroth element being the part of the searched string that matches the supplied groupings.

## SPLITTING STRINGS: EXPLODE()

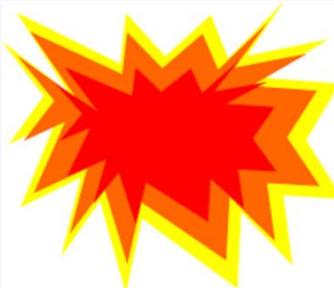
Split a string into array elements

- Takes two parameters:
  - A single string delimiter
  - The string to split

Returns an array containing the sub-strings

```
$date = '31/12/2017';  
$result = explode('/', $date);  
  
echo "$result[2]/$result[1]/$result[0]";
```

2015/12/31



The `explode()` enables you to split strings using a single delimiter. The above example uses the function to split a date string using the slash (/) delimiter.

If you need to use multiple delimiters use the `preg_split()` function which takes a regex as a parameter.

## SPLITTING STRINGS: PREG\_SPLIT

- Split a string into array elements
  - Similar to explode(), but uses regular expressions so is therefore slightly slower
- Takes two parameters
  - A PCRE regular expression to specify the field delimiters
  - The string to split
- Returns an array containing the sub-strings

```
$date = '11.11-1911';
$result = preg_split('/[.-]/', $date);

echo "$result[2]/$result[1]/$result[0]";
```

1911/11/11

Strings are often split into arrays in PHP. For example, you may wish to split URLs into their constituent parts or a comma-separated lists into its individual elements.

However, if the separator is a single character (e.g. a comma or a slash) then use the explode() function rather than a regex as it is much simpler and faster.

Note: The preg\_split function can also accept an optional int parameter that specifies the maximum number of elements that the array should contain. If you specify the optional parameter, the last element of the array will contain any remaining part of the string.

There is also a function to split a string by length, rather than pattern. The str\_split function contains an optional split-length parameter of type integer. If you specify the split-length parameter, the elements of the returned array will contain the number of characters specified by the split-length parameter.



## NUMBER FORMAT



Returns a number as a formatted string

```
number_format($number, decimalPlaces, decSymbol, thousandsSep);  
  
$number = 1234.56;  
  
// English notation (default)  
$english_format = number_format($number);  
// 1,235  
  
// French notation  
$french_format = number_format($number, 2, ',', ' ');  
// 1 234,56  
  
$number = 1234.5678;  
  
// notation without thousands separator  
$english_format = number_format($number, 2, '.', '');  
// 1234.57
```

The `number_format()` function will return a string of a number using the "numerical punctuation" typical of a particular locale. This function accepts either one, two, or four parameters: the number to be formatted, the number of decimal places, the decimal point character and the thousands separator character.

`number_format(1234.56, 2, ',', ' ')` means format number 1234.56 to 2 decimal places, using a comma to separate the decimal places and a space for a thousands separator.

- **bcadd** — Add two arbitrary precision numbers
- **bccomp** — Compare two arbitrary precision numbers
- **bcddiv** — Divide two arbitrary precision numbers
- **bcmod** — Get modulus of an arbitrary precision number
- **bcmul** — Multiply two arbitrary precision numbers
- **bcpow** — Raise an arbitrary precision number to another
- **bcpowmod** — Raise an arbitrary precision number to another, reduced by a specified modulus
- **bcscale** — Set default scale parameter for all bc math functions
- **bcsqrt** — Get the square root of an arbitrary precision number
- **bcsub** — Subtract one arbitrary precision number from another

PHP provides the `bcmath` library (Binary Calculator Math) for arbitrary precision mathematics. It supports numbers of any size and precision, represented as strings.

To perform calculations that must be accurate to a high number of decimal places, use the functions in this library.

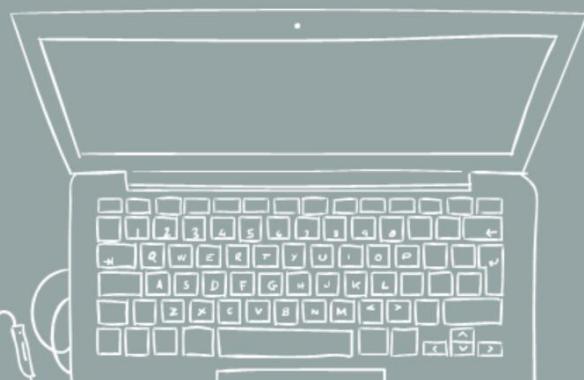
Never assume that operations on ordinary floating-point numbers are accurate. If you were to create a financial application in PHP, use the above library for improved accuracy.



**Any questions?**



## EXERCISE 12



## RESOURCES

String Functions

<http://php.net/manual/en/book.strings.php>

Regular Expressions

<http://php.net/manual/en/reference.pcre.pattern.syntax.php>

Explode Function

<http://php.net/manual/en/function.explode.php>

BCMath

<http://php.net/manual/en/book.bc.php>

Combined Comparison Operator

<https://wiki.php.net/rfc/combined-comparison-operator>

# FILE IO



## CHAPTER OVERVIEW

- Introduction to file I/O
- Opening a file
- Reading a file
- Writing to a file
- Working with XML
- Other useful file functions
- Serialization
- Output Buffering

### Exercise File IO

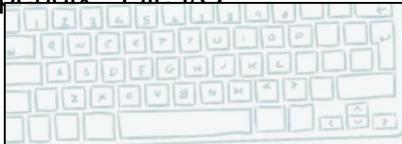
## INTRODUCTION TO FILE I/O

- Databases are relatively expensive to maintain
- Flat files are often used to store simple data
  - Supported by many programs and tools
  - Easy to transfer data between systems
  - Easy to maintain and check
  - Simple
- Most programming languages have functions to:
  - Open a file
  - Read or write to the file
  - Close a file

Most programming languages support some sort of file input/output, and PHP is no exception. Simple text files are suitable for relatively small amounts of data, or data held sequentially such as log files. Reading such files is normally sequential, that is you start at the beginning of the file then read each line until you find what you are looking for. Clearly this can be slow if you are dealing with a large file.

Writing text files from a user's perspective is easy - use a text editor. Where records (lines) in the file are variable length, updating programmatically can be rather more complex. This is because there is no inherent record structure: if a replacement line was longer than the original then it would overwrite part of the next line. Therefore, updating is often restricted to appending: adding new lines onto the end of the file. This has the advantage of being very fast, and is often the method of choice for error logging and audit trails.

All this is not to say that PHP cannot deal with binary files, or access files randomly. It is just that you will do this less often. More complex tasks are generally handled by relational database systems such as MySQL.



## OPENING A FILE

A file must be opened before use

- The operating system has to find the file
- Security checks must be performed before access is allowed
- This is abstracted to a *resource* returned by **fopen**
- Then used on subsequent IO requests

```
$resource = fopen($filename, $mode);
$fileResource = fopen('/etc/hosts.txt', 'r');
```

Available modes:

- **r** Open for read
- **w** Open for write. If the file does not exist, create it, otherwise truncate it (overwrite).
- **a** Open for append (write). If the file does not exist, create it.
- **x** Create and open for write; file must already exist.

C programmers will be familiar with **fopen** - the PHP version is very similar.

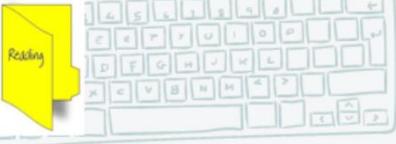
As well as the modes shown, adding a '+' will open the file for both read and write:

- |           |                                                                                        |
|-----------|----------------------------------------------------------------------------------------|
| <b>r+</b> | Open for read and write                                                                |
| <b>w+</b> | Open for read and write. If the file does not exist, create it, otherwise truncate it. |
| <b>a+</b> | Open for append (read and write); If the file does not exist, create it.               |
| <b>x+</b> | Create and open for read and write; file must already exist.                           |

On Microsoft Windows you can specify 't' to translate between "\n" (normal line endings) and "\r\n" (Microsoft line endings).

Microsoft Windows also distinguishes between text and binary files. The default modes open files as binary (from PHP 4.3.2). To open as a binary file add a 'b' to the mode, for example: '**rb**'.

Opening files from a web server can be problematic, since it is not always obvious which is the current directory. It is usually best to use fully qualified path names, rather than relative ones.





## READING A FILE BY LINE

- The fgets() function gets a line terminated by "\n"

```
$fp = fopen('hosts.txt', 'r'); //open for reading
                           // $fp resource value

while ( !feof($fp)           //while NOT at file-end
      && $line = fgets($fp)    //AND while
      )                      //fgets returns a line
{
    echo explode("\t", $line)[0]; // echo the IP
    echo PHP_EOL;              // echo a "\n"
}

fclose($fp);                //close the resource(file)
```

- Always test the result of I/O function calls; they return 'false' on error

The simplest way to read a text file is line-by-line in a loop. The function `fgets` (another stolen from C) reads the next line from a file, or up to the (optional) maximum length specified.

If you do not specify a maximum length reading ends when `length - 1` bytes have been read, or a newline or an end of file marker is reached. If no length is specified, it will keep reading from the stream until it reaches the end of the line.

`feof` — Tests for the end-of-file on a file pointer

The example reads text from a `hosts.txt` file. The hosts file is used by an operating system to map hostnames to IP addresses. The hosts file contains lines of text consisting of an IP address in the first text field followed by one or more host names. The `echo` statement above outputs the zeroth element from the exploded text which is therefore the IP address.

## OTHER READING METHODS

- Read n bytes from a file – **fread()**
  - File must be opened by fopen and closed by fclose

```
$data = fread($fp, $n);
```

- Read the whole file into a variable
  - File does not need to be opened / closed by fopen and fclose
  - **file\_get\_contents()** – reads entire file into a string

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

- **file()** – reads into an indexed array
- File does not need to be opened / closed by fopen and fclose

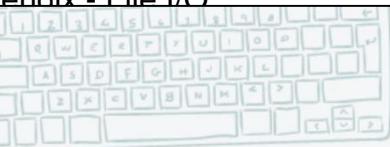
```
$lines = file($filename);
echo $lines[2];
```

There are other ways of reading a file, and three are shown: **fread**, **file\_get\_contents** and **file**.

**fread** (another function from C) enables us to read a specified number of bytes. It is usually used when reading binary files.

**file\_get\_contents** and **file** are similar, and are a much simpler way of accessing a file in one lump - so called *slurping*. **file\_get\_contents** slurps into a scalar variable, so the whole file is in one long stream, including the new-lines which delimit each record.

**file** is rather easier, since each record is placed as a separate element in an indexed array. With the exception of **fread()** above, none of the functions require an **fopen** or an **fclose**. Since the whole file is read into memory, these functions should only be used for files with a size of a few kilobytes.



## WRITING TO A FILE



### Write n bytes to a file – **fwrite( )**

- File must be opened by fopen for write or append
- fwrite is also known as fputs

```
$n = fwrite($fp, $data);
```

- Dump a variable into a file
  - **file\_put\_contents( )** – write from a variable
  - Same as fopen, then fwrite, then fclose

```
file_put_contents($filename, $data);
```

- Optional flags allow more control

```
file_put_contents($filename, $data, FILE_APPEND);
```

Writing to a file usually uses the function **fwrite**, or its alias **fputs**. The function returns the number of bytes written, which for normal files should be the same as that requested unless, for example, the disk is full. We can also specify the maximum number of bytes to write with an optional third parameter.

The function **file\_put\_contents** is the opposite of slurp - it puts the value of a variable (scalar or array) into a file. A number of optional flags are available and a common one is shown.

**FILE\_APPEND**: If file \$filename already exists, append the data to the file instead of overwriting it.



TEST BEFORE USE



## Test

- Whether you have permissions to access a file
- Whether you can read/write/etc. To it
- *Before accessing it*

```
if( file_exists($path)
&& is_readable($path)
&& is_writable($path)) {

    //use $path

}
```



- Otherwise there will be errors!



To avoid errors when reading and writing to files use the following functions to check file permissions and file existence:

`is_readable()` – checks read permissions

`is_writable()` – checks write permissions

`file_exists()` - checks whether a file or directory exists

## OTHER USEFUL FILE FUNCTIONS

|                          |                                                                                            |
|--------------------------|--------------------------------------------------------------------------------------------|
| is_readable              | Do I have read access? Also is_writeable, is_executable, is_file, is_dir, file_exists etc. |
| opendir/readdir/closedir | For iterating through a directory                                                          |
| filesize                 | Returns file size in bytes                                                                 |
| fileperms                | Return the UNIX style file permissions                                                     |
| filetype                 | Return the file type                                                                       |
| readfile                 | Read a file and write it to stdout                                                         |
| stream_get_line          | Read from a file, up to a delimiter                                                        |
| unlink                   | Delete a file                                                                              |

As you can see there are numerous functions for handling files and directories.

**fflush()** - This function forces a write of all buffered output to the resource pointed to by the file handle.

**fileperms()** - Gets permissions for the given file as a numeric mode value.

**highlight\_file()** - Prints out or returns a syntax highlighted version of the code contained in a file using the colours defined in the built-in syntax highlighter for PHP.

**touch()** - Attempts to set the access and modification times of the file named in the filename parameter to the value given in time. Note that the access time is always modified, regardless of the number of parameters. If the file does not exist, it will be created.

See [PHP.net](http://PHP.net) for more information on the above functions.

## DIRECTORIES

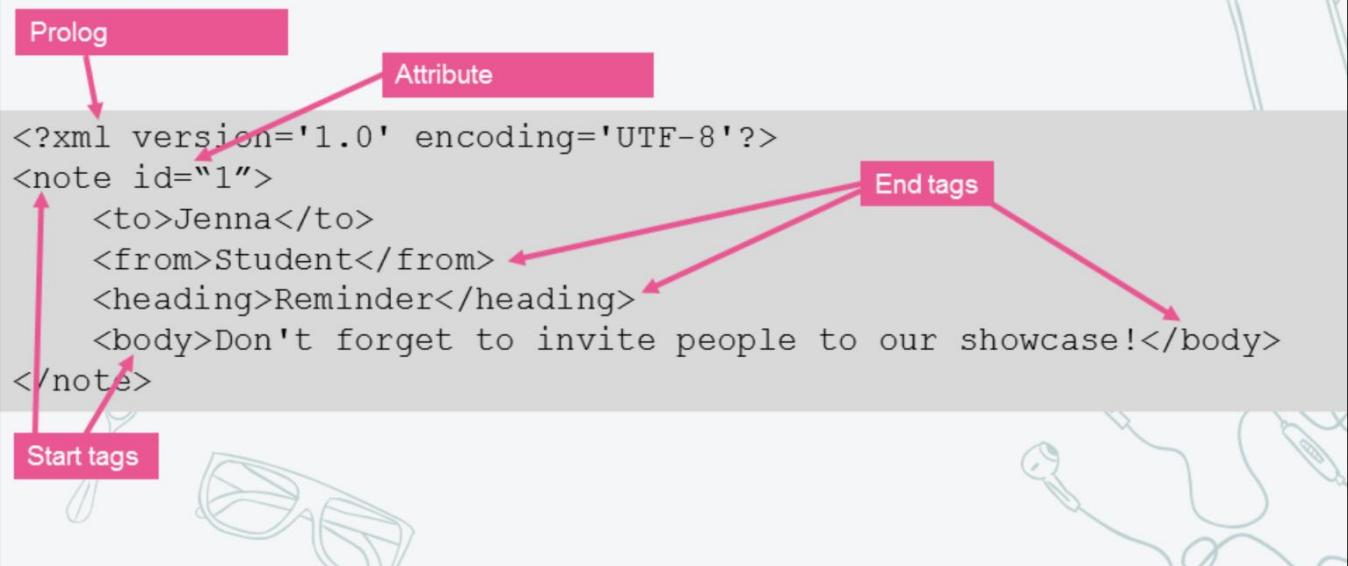
- `chdir` Change directory
- `chroot` Change the root directory
- `closedir` Close directory handle
- `dir` Return an instance of the Directory class
- `getcwd` Gets the current working directory
- `opendir` Open directory handle
- `readdir` Read entry from directory handle
- `rewinddir` Rewind directory handle
- `scandir` List files and directories inside the specified path

The above functions enable you to navigate and interrogate an operating system's directory structure.

See [PHP.net](http://PHP.net) for more information on the above functions.

## AN INTRODUCTION TO XML

- eXtensible Markup Language
- Data storage and transportation
- Both machine and human-readable
- XML has syntax rules



A popular data exchange format is XML – eXtensible Markup Language. The XML language is a way to structure data for storing and transporting across the web and between organisations. It is a markup language that consists of a hierarchical structure similar to HTML. Unlike HTML, XML is extensible which means you create your own tags to describe your data. Several web technologies like RSS Feeds and Podcasts are written in XML. XML was designed to be both human- and machine-readable.

XML documents are hierarchical, or tree-like, and consist of elements, attributes and text nodes. An XML tree starts at a **root element** and branches from the root to **child elements**.

All elements can have their own child elements. All elements are related in some way just as in a family tree: elements can be siblings, parents and children of other elements. All elements can have text content (“Student” or “Reminder”) and attributes (`id="1"`). The prolog defines the XML version and the character encoding.

XML syntax has some rules:

XML documents must contain one **root element** that is the **parent** of all other elements. In the above example this is `<note>`.

The XML prolog is optional. If it exists, it must come first in the document. The prolog is:

```
<?xml version='1.0' encoding='UTF-8' ?>
```

As XML documents can contain international characters, like acute accents in French, you should specify the encoding used, or save your XML files as UTF-8.

Unlike HTML, in XML, it is illegal to omit the closing tag. All elements **must** have a closing tag e.g. </to>, </note>.

XML tags are case sensitive. The tag <Note> is different from the tag <note>. Therefore opening and closing tags must be written with the same case.



## READING XML FROM A VARIABLE

Use **simplexml\_load\_string()** to read XML from a variable

```
<?php  
$myXMLData =  
"<?xml version='1.0' encoding='UTF-8'?>  
<note id='1'>  
    <to>Jenna</to>  
    <from>Student</from>  
    <heading>Reminder</heading>  
    <body>Don't forget to invite people to our  
showcase!</body>  
</note>";  
  
$xml=simplexml_load_string($myXMLData) or die("Error:  
Cannot create object");  
print_r($xml);  
?>
```

You may need to read data into your PHP applications from a variable that stores XML or from XML files. To read XML data as a string variable use the PHP function **simplexml\_load\_string()**. The **die()** function is used to trap any errors that may occur.

NOTE: The XML functions you use in PHP are object-oriented. They return an object or an array of objects of the type **SimpleXMLElement**. In object-orientated programming (OOP) objects store their own data and provide functionality to access or manipulate this data. In most OOP languages you access the data that is stored by accessing the object's properties and you access the object's functionality by accessing the object's methods. The term 'method' is an OOP way of referring to the functions of an object.

## READING XML FROM A FILE

Use **simplexml\_load\_file()** to read XML from a file

```
<?php  
  
$xmlNodes=simplexml_load_file("notes.xml") or die("Error:  
Cannot create object");  
  
//display all the returned SimpleXMLElement Objects  
print_r($xmlNodes);
```

To read XML data from a file use the `simplexml_load_file` function and pass it the name of the XML file.



## EXAMPLE XML FILE



## Example of an XML file containing book information

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
<book category="cooking">
    <title lang="en">Jamie's 15 Minute Meals</title>
    <author>Jamie Oliver</author>
    <year>2012</year>
    <price>8.00</price>
</book>
<book category="cooking">
    <title lang="en">How to be a Domestic Goddess</title>
    <author>Nigella Lawson</author>
    <year>1998</year>
    <price>16.00</price>
</book>
<book category="children">
    <title lang="en">Harry Potter...
```

Above is part of a sample XML document containing book information. The file is saved as books.xml.

The root of the document is <bookstore>. This is the parent element of the book (child / sub) elements. Each <book> element contains a category attribute and child elements for title, author, year and price. Each of these child elements has a text value.

## PROCESSING XML FROM A FILE

Use the **children()** method to access the book (child) elements of the bookstore

```
$xmlNodes=simplexml_load_file("books.xml") or die("Error:  
Cannot create object");  
  
echo "Read XML Nodes in a loop" . PHP_EOL;  
  
foreach ($xmlNodes->children() as $book) {  
    echo $book->title . ", ";  
    echo $book->author . ", ";  
    echo $book->year . ", ";  
    echo $book->price . PHP_EOL;  
}
```

To access the elements within the SimpleXMLElement use the children( ) method. In most OOP languages you use a dot (.) between the object and its properties and methods however in PHP the dot is used for concatenation so another operator was chosen: **->**

This operator is often referred to as the 'object operator', the 'skinny arrow' or the 'single arrow' operator. Note it is different to the double-arrow operator (**=>**) that you use with associative arrays.

The returned object in the example above is \$xmlNodes. It is typically an array of **SimpleXMLElement**. To access its methods or properties use the object name followed by the skinny arrow followed by either the property or method of your choice. To access the children method use the following:

```
$xmlNodes->children()
```

The children method finds the children of the given node.

You can then use the same operator to access each of the properties of the book, such as \$book->title and \$book->price.

## PROCESSING XML ATTRIBUTES

To access XML attributes ensure you are at the correct element level in the hierarchy and use array-like syntax:  
\$element['attributeName']

```
foreach ($xmlNodes as $book) {  
    echo $book['category'] . PHP_EOL;  
}  
  
foreach ($xmlNodes->children() as $book) {  
    echo "Language: " . $book->title['lang'] . " Title: ";  
    echo $book->title . PHP_EOL;  
}
```

To access the attributes within the elements ensure you are accessing the correct level of element and use array-like notation to access the attribute by name.

In the above example the category attribute exists within the book element and the lang element exists within the title sub element so you need to access the child level to gain access to the title element and its attribute.

## OUTPUT BUFFERING

- Output buffering is a technique to capture all output, such as echo statements, as a value
- It prevents any output being sent and enables the developer to determine when you would like to send the output

```
function display($template) {  
    ob_start();  
    include $template;  
    return ob_get_clean();  
}
```



- **ob\_start()** begins the output buffering
- **ob\_get\_clean()** finishes buffering and returns the buffered output

PHP's output control functions allow you to control when output is sent from the script. Without output buffering (the default), your HTML is sent to the browser in pieces as PHP processes through your script. With output buffering, your HTML is stored in a variable and sent to the browser as one piece at the end of your script. The main advantage of output buffering for Web developers is that it decreases the amount of time it takes to download and render your HTML because it's not being sent to the browser in pieces as PHP processes the HTML.

**ob\_start()** – This function will turn output buffering on. While output buffering is active no output is sent from the script (other than headers), instead the output is stored in an internal buffer.

The contents of this internal buffer may be copied into a string variable using **ob\_get\_contents()**. To output what is stored in the internal buffer, use **ob\_end\_flush()**. Alternatively, **ob\_end\_clean()** will silently discard the buffer contents.

**ob\_get\_clean()** - This function discards the contents of the topmost output buffer and turns off this output buffering. If you want to further process the buffer's contents you have to call **ob\_get\_contents()** before **ob\_end\_clean()** as the buffer contents are discarded when **ob\_end\_clean()** is called. **ob\_get\_clean()** essentially executes both

**ob\_get\_contents()** and **ob\_end\_clean()**. It returns a string.

**ob\_get\_contents()** - Gets the contents of the output buffer without clearing it.

**ob\_end\_flush()** - Flush (send) the output buffer and turn off output buffering. It returns a bool.

## SERIALIZATION

- Generates a storable representation of a value
- This is useful for storing or passing PHP values around without losing their type and structure
- To make the serialized string into a PHP value again, use **unserialize()**

```
$string = serialize(['name' => 'value']);  
  
file_put_contents('file.db', $string);  
  
$array = unserialize(file_get_contents('file.db'));
```

**Serialization** is the process of converting an object into a stream of bytes in order to store the object or transmit it to memory, a database, or a file. Its main purpose is to save the state of an object in order to be able to recreate it when needed. The reverse process is called **deserialization**.

PHP provides the `serialize` and `unserialize` functions.

The `serialize` function can take all types, except the resource-type as its input. It returns the type serialized as a string.

The `unserialize` function takes the serialized string as its input and returns the converted value. The type can be a Boolean, integer, float, string, array or object. In case the passed string is not unserializeable, `FALSE` is returned and an `E_NOTICE` is issued.

## FILE IO SUMMARY

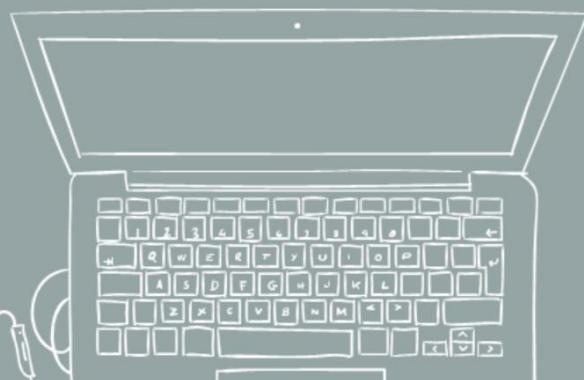
- To read a file:
  - Use fopen -> fgets -> fclose
  - Or use file\_get\_contents
- To write to a file
  - Use fopen -> fputs -> fclose
  - Or use file\_put\_contents
- PHP supports many other file-system functions
- PHP can manipulate XML files  
Output buffering can capture output and prevent it being displayed
- Serialization converts PHP values to a storable string representation



# Any questions?



## EXERCISE 13



## RESOURCES

Filesystem Functions

<http://php.net/manual/en/function.file.php>

Directory Functions

<http://php.net/manual/en/ref.dir.php>

W3Schools SimpleXML Parser

[https://www.w3schools.com/php/php\\_xml\\_simplexml\\_read.asp](https://www.w3schools.com/php/php_xml_simplexml_read.asp)

Output Buffering

<http://php.net/manual/en/book.outcontrol.php>

Serialization

<http://php.net/manual/en/function.serialize.php>

# ARRAY FUNCTIONS

## CHAPTER OVERVIEW

- Dereferencing Syntax
- Functions for Manipulating Arrays
- Functions for Manipulating Functions

### Exercise

#### Arrays & Functions

You have seen that data can be grouped together into collections called arrays. This data structure is extremely versatile and arrays are the workhorse of the PHP language.

There is a whole suite of functions dedicated to adding, modifying, combining, splitting and otherwise manipulating arrays. These predefined functions provided by PHP provide many of the common operations you will need when managing arrays. If arrays are workhorses, functions are their housemasters and they form an essential part of any PHP application. There are therefore a few predefined functions which help with functions themselves: functions that concern functions. This chapter will build on your syntactical knowledge of arrays and functions, introducing the predefined libraries that are helpful when working with them.

The array functions are ubiquitous in PHP application development and every developer needs to know they exist, and broadly, what they can do.

The PHP library functions are so numerous, diverse and unconventional that ultimately even the most professional PHP developer relies on PHP's documentation to supplement their knowledge. It is essential that you are aware of the functionality that exists so that if you are facing a new problem you know there are functions that can help you solve it.



## ARRAYS: MORE SYNTAX

### Two syntaxes for array definition

```
$pens = ['Parker', 'Waterman', 'Sheaffer'];  
  
$pens = array('Parker', 'Waterman', 'Sheaffer');  
  
$people = array(  
    'Alice' => array('Leeds', 'London'),  
    'Lucy'  => array('Paris', 'Lyon')  
);  
  
// list() can be used to destruct an array  
// within a foreach  
  
foreach($people as $name => list($first, $second)) {  
    echo "$name may be at either $first or $second";  
}
```

As PHP has transitioned through its major versions it has retained backwards compatibility. This commitment is two fold: the core developers not only wish to allow the old syntax but also to allow the new approach. This is particularly clear with its array syntax. The older syntax uses the array keyword and a parenthetical list of elements; the new syntax uses square brackets. The newer syntax is more common across other languages and many developers have now switched to this form. However, you will still see a considerable amount of historical code written using the array keyword and you will see some new code by developers who retain this preference.

The foreach loop above contains the list() variable destructuring construct. The list() construct is used to assign values to variables. The value of the foreach loop is an array of place names, so list() assigns each sequential element to the variables \$first and \$second. This is a very new feature and is more of a syntactical curiosity than anything critical you need to understand.

## ARRAY 'DEREFERENCING'

- Functions which return arrays may be used as arrays directly

```
$parts = explode('/', 'example.com/a/b/c');  
echo $parts[2];
```

```
//compare with dereferencing  
echo explode('/', 'example.com/a/b/c')[2];
```

- Using [ ] (subscript) notation directly is called 'dereferencing'
- Convenient if you need only a part of the result since the return value will be otherwise discarded

Recall that you can apply the subscript operator (i.e. key-lookup) to an array to access one of its elements:

```
$array['someKey']
```

The subscript operator [ ] however, can also be applied to expressions which evaluate and return arrays.

In the example above the function explode() splits the URL string into parts based on the '/' separator. This array can be assigned to a variable or, as in the second example, accessed directly as if it were the array.

This feature was introduced in PHP 5.4.

## SORTING

- **sort()** sort arrays in ascending order
- **rsort()** sort arrays in reverse (descending) order
- **asort()** sort associative arrays in ascending order, according to the value
- **ksort()** sort associative arrays in ascending order, according to the key
- **arsort()** sort associative arrays in descending order, according to the value
- **krsort()** sort associative arrays in descending order, according to the key
- **usort()** sort array using a supplied callback function

PHP has a variety of sorting functions that you can use to change the order of the data that is stored. You may want to change the order of data that a user has inputted. The sorting functions allow you to choose whether to sort your array in ascending or descending order or by an associate array's key or value.



## SORTING



PHP sort functions sort arrays *in-place* (i.e. by reference). So you must supply a variable, not an array directly

```
$numbers = [8, 6, 3, 0];  
  
sort($numbers);  
var_dump($numbers);  
  
$ages = ['Mr. S. Holmes' => 27,  
         'Mr. M. Holmes' => 34,  
         'Ms. Hudson' => 70];  
  
arsort($ages);  
var_dump($ages);  
  
//compare var_dump with print_r
```

```
array(4) {  
    [0]=> int(0)  
    [1]=> int(3)  
    [2]=> int(6)  
    [3]=> int(8)  
}  
array(3) {  
    ["Ms. Hudson"]=>  
    int(70)  
    ["Mr. M. Holmes"]=>  
    int(34)  
    ["Mr. S. Holmes"]=>  
    int(27)  
}
```

In the example, the variable \$numbers contains the array you wish to sort. Note that this array is passed to the sort function and the variable itself changes: if you inspect its values before and after the function call you will see that \$numbers has changed.

This is unusual behaviour for a function which usually returns new data rather than modifying one of its input arguments, however, PHP has decided to make sort()'s argument a by-reference parameter.

In practice most of the data which drives a web application comes from a database which provides powerful sorting tools of its own. Alternatively, you may wish to sort an array that the user is viewing in their browser. In each case a different language is responsible for the data sorting: SQL or JavaScript.

NOTE: print\_r prints human-readable information about a variable and will therefore display slightly less data in a more readable format as it excludes some of the information that var\_dump produces, such as the datatype, some brackets and the array length.

## SORTING WITH A CALLBACK

- **usort()** and some other array functions accept callbacks
  - we can use an anonymous function directly

```
$basket = [];  
$basket[] = ['name' => 'oranges'];  
$basket[] = ['name' => 'lemons'];  
$basket[] = ['name' => 'apples'];  
  
usort($basket, function ($l, $r) {  
    return strcmp($l['name'],  
                  $r['name']);  
});  
  
print_r($basket);
```

Array ( [0] => Array ( [name] => apples ) [1] => Array ( [name] => lemons ) [2] => Array ( [name] => oranges ) )

By way of illustration, the usort() function behaves differently to the other set provided by PHP: it requires that you supply a function which helps determine the order in which the elements will be sorted.

Note: The basket array contains associative array elements. Each defines a key, 'name' and a value 'some fruit'. In order to sort basket by the string value of each item we could not use the basket array itself, since its keys are integers and do not relate to the string value of the items.

The callback we pass to PHP's usort() function supplies two parameters: elements in the array we wish to sort. Your function should return -1 if the first parameter "sorts lower" than the second, 0 if they sort equally and 1 if it "sorts higher".

Helpfully, the strcmp() function behaves in exactly this way. However, you cannot use it directly because the elements of basket are associative arrays and not strings. You use usort() to pass the correct values to strcmp() as required.

## MERGING AND REDUCING

- **array\_merge()**
  - concatenates two arrays, *joining* them together
- **array\_slice()**
  - from a sequential input array, extract elements at a specific key range
- **array\_shift()**
  - remove an element from the beginning of an array
- **array\_pop()**
  - remove an element from the end
- **array\_push()**
  - append an element – compare with `\\$array[]` which is more idiomatic
- **array\_unshift()**
  - prepend an element (i.e. at the beginning)
- **array\_unique()**
  - return only unique values

Other than sorting, arrays can be "reshaped": sliced, concatenated, resized and merged.

When manipulating an array you are mostly concerned with adding or removing elements either at the beginning or the end; `array_shift`, `array_unshift`, `array_pop` and `array_push` work well as defined above.

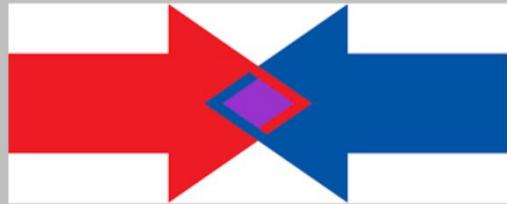
`array_unique()` removes any duplicate elements from an array.

For example, suppose you had an array of all the links on a webpage. Some of those links may be repeated; there are often many links to a site's home page. To calculate the number of independent links you would write:

```
count(array_unique($links));
```

## SLICE AND MERGE

```
$path = array_slice(  
    explode('/', 'example.com/blog/post/1'), 1);  
  
print_r(array_merge( ['yourdomain.tld'], $path ));  
  
Array (  
    [0] => yourdomain.tld  
    [1] => blog  
    [2] => post  
    [3] => 1  
)
```



- **array\_slice** takes a slice of the array starting at position 1
- **array\_merge** concatenates (or joins) the two arrays together
- **array\_merge** behaves differently to array addition ('\$a + \$b')

Here two array manipulation functions are illustrated: `array_merge` and `array_slice`.

In the first case recall that `explode()` returns an array of elements, each a segment of the supplied string divided by its separator. Here `explode()`'s zeroth element is `example.com` which this application does not need. `array_slice()` here "slices from the 1st element", i.e. it omits the "`example.com`" segment.

Thus `$path` is now an array of `['blog', 'post', '1']`.

`array_merge()` concatenates two arrays; it glues them together. It takes the first array as the "head" of the result and glues on the "tail". `array_merge()` is somewhat like the string concatenation operator `( . )` for strings.

If you perform array addition `(+)` instead of `array_merge` it would result in an array of `['yourdomain.tld', 'post', '1']` because an addition adds elements from keys that are not already in use. Index position 0 in the first array is in use by '`yourdomain.tld`' therefore '`blog`' which is also at index position 0 in the second array is not added.

## KEYS AND VALUES

- **array\_keys()**
  - from an input array, returns only the keys
- **array\_values()**
  - from an input array, returns only the values
- **array\_combine()**
  - from two sequential arrays of keys & values, make an associative array
- **range()**
  - produces an array of elements each within a range
- **array\_key\_exists()**
  - determines if a key is in an array
  - `isset($array[$key])` is more idiomatic
- **in\_array()**
  - determines if a value is in an array
- **array\_column()**
  - from an input array whose elements are arrays, extract values at a key



There are a variety of functions PHP provides for accessing parts of associative arrays as well as determining what exists in each of them.

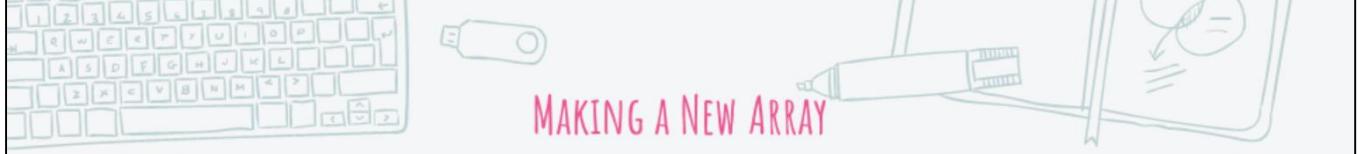
The `array_keys()` function takes an associative array, suppose a database result set, and returns a sequential array of its keys which in this case would be the database fields.

`array_values()` basically "throws away" all the keys of an associative array and makes it sequential which can be useful when passing arrays to functions that require sequential array arguments, for example, `str_replace`.

```
str_replace(array_keys($dictionary), array_values($dictionary), $article);
```

`in_array()` takes two arguments: a value and an array to search. It returns true if the value is found in the array.

```
if(in_array('Student', ['Student', 'Sherlock', 'Mycroft']) { echo "It's in!"; }
```



## MAKING A NEW ARRAY

```
$users = [  
    ['id' => 1, 'usern' => 'shomes', 'pwd' => 'irene'],  
    ['id' => 2, 'usern' => 'mhomes', 'pwd' => 'sher'],  
    ['id' => 3, 'usern' => 'watson', 'pwd' => 'lock'],  
];  
  
print_r(  
    array_combine(  
        array_column($users, 'id'), //array of ids  
        array_column($users, 'pwd') //array of pwds  
    )  
);  
  
Array (  
    [1] => irene  
    [2] => sher  
    [3] => lock  
)
```

The example above illustrates the power of PHP's array manipulation functions.

The goal of this piece of code, and so often with array manipulation code, is to take an array of one "shape" and turn it into another.

Here the initial array, \$users, is a sequential array of 3 elements. Each element is an associative array connecting keys (id, usern, pwd) to values.

The goal of the code is to take a list of users and get their id-password pairs. You could loop over this array, however there are some specialized functions for this case.

The `array_column()` function collects the values of a given key from a list of arrays with that key. `array_column($users, 'id')` is therefore a sequential array of IDs.

Using `array_column` you can get an array of ids and an array of passwords, however they are not associated. Finally, the `array_combine()` PHP function takes a sequence of keys and a sequence of values and pairs them up into an associative array.

## ITERATING

- **array\_map()**
  - apply a callback to every element of an input array, keep its return value
- **array\_filter()**
  - apply a predicate function to every element of an array
  - if it returns false, discard that element
- **array\_walk()**
  - apply a callback to every element of an input array, discard its return value
- **array\_walk\_recursive()**
  - as above, but continue into nested arrays



The next set of functions relating to PHP arrays concern how you can access their values.

In each case these functions take a callback and run that callback on each element of the array.

In the case of `array_map()`, it stores the result and returns it. In the case of `array_walk()` and `array_walk_recursive()` it throws it away. The `array_walk` functions can be used to manipulate the data within the array e.g. to do something to the data.

`array_filter()` expects an array and a callback function which returns true or false. If the callback returns true, `array_filter()` will keep the value in the original array, otherwise it will remove it.

A typical example usage follows.



## TRANSFORMING AN ARRAY

```
$transformer = function ($el) { return $el ** 2; };  
$input = range(0, 8, 2); //produces 0, 2, 4, 6, 8  
  
$output = array_map($transformer, $input);  
print_r($output);  
  
Array (  
    [0] => 0  
    [1] => 4  
    [2] => 16  
    [3] => 36  
    [4] => 64  
)
```

- This style of programming is known as *functional programming*:
  - some transformation from input data to output data

The original array, \$input, is the values [0, 2, 4, 6, 8] – obtained by calling the range() function which makes linear counting arrays of this kind. Its first parameter is the starting value of the range, the second is the end of the range, the third parameter is the step. Therefore, range(0, 8, 2) produces an array of values 0,2,4,6,8.

A function \$transformer is then defined; it is an anonymous function assigned to a variable. It could be defined in any way (anonymous functions are increasingly the convention).

The function transformer takes one argument and returns that argument squared. It is very simply, the squaring function:  $5^{**} 2$  is 25.

The aim of this code is to transform the input values using this squaring function; to apply the squaring function to each element of our input so that we get an array whose values are the input values squared.

This is the job of array\_map: it applies a callback function to each element and collects the result into an array which it returns.

The functions discussed on the previous slide follow the same pattern: an input array, a callback and an output.

## FUNCTION 'DEREFERENCING'

Functions which return functions may be used as functions directly

```
function prep($prefix) {  
    return function ($msg) use ($prefix) {  
        return $prefix . $msg;  
    };  
}  
//two lines to call  
$logger = prep('Logging: ');  
echo $logger('message');  
  
//compare with single line to call  
  
echo prep('Logging: ')('message');
```

This section focuses specifically on the functions PHP provides for analyzing and supporting functions themselves. This is where the idea of using a function as a value becomes even more pronounced: you can return a function. In the example, the prep() function returns a closure which takes one argument. That is, if you call prep() you will get a function that you can call and it will take one argument.

NOTE: A **closure** is a **function** that has an environment of its own. Inside this environment, there is at least one bound variable. In the example above this is \$prefix. When a **closure** is used more than once, the bound variables stay the same in between.

Let's be extra clear: if you call prep the value returned is a function. This function can itself be called.

In the example, \$logger is a function, since it is returned by the call to prep(). You can use the \$logger function as you would any other.

The prep() function returns a function you can call directly by chaining the parameters together:

```
prep('Logging: ')('message');
```

This is an extremely subtle snippet of code which layers several concepts: functions-as-values, closures, returning functions, calling return values (dereferencing).

There is no expectation that a developer new to PHP will grasp this immediately, however code of this style is appearing more often in contemporary PHP frameworks so it is worthwhile attempting to follow it through.

## CALLING FUNCTIONS

- **call\_user\_func()**
  - call a function
  - compare with `\\$f()` which is also used
- **call\_user\_func\_array()**
  - call a function and pass arguments as array
  - compare with `\\$f(...\$arguments)` which is newer and may become idiomatic
- **forward\_static\_call** and **forward\_static\_call\_array**
  - call a static method – discussed later
- **function\_exists()**
  - true if the given function has been defined
- **get\_defined\_functions()**
  - returns array of all defined functions

You have seen that you can assign a variable to a function or a string and then call it as though it were a function:

```
$fn = 'ucwords';  
$fn("this really works!");
```

This is equivalent to writing:

```
call_user_func('ucwords', 'this really works');
```

The choice is purely stylistic and there are no well established conventions. `call_user_func` may be clearer to a person unfamiliar with PHP.

`call_user_func_array` will treat a single array passed as a parameter as individual sequential arguments to the function you are calling. This performs the same job as the unpacking operator `...$array` discussed earlier.

`function_exists()` takes a function name and returns true if it exists – functions may not exist, for example, if an extension hasn't been enabled in the `php.ini` file. This is rarely used. Equally so with `get_defined_functions()` which returns an array of the names of all defined functions.



## DYNAMIC DISPATCH

- Calling a function by using a string version of its name is sometimes called *dynamic dispatch*
- *Dynamic* since which function gets called depends on a variable which might change at runtime

```
function webpageA($domain, $page) { echo 'A'; } //Define  
function webpageB($domain, $page) { echo 'B'; } //functions  
  
$url = 'example.com/A';  
$parts = explode('/', $url);  
  
call_user_func_array('webpage' . $parts[1], $parts);  
  
//or  
$fn = "webpage{$parts[1]}"; //assign function to variable  
$fn(...$parts); //call the function with params
```

- Can be used to connect URLs to functions (i.e. urlA calls functionA)

Users input text strings into web applications. You may often want to associate some element of user input (a form field, a url value, etc.) with a particular function your application can perform. The goal then, is to associate a string with a function call.

In the example above two functions are defined webpageA and webpageB which represent two areas of your website. For example, webpageA is a function which provides all the content of one web page.

The \$url variable simulates an incoming URL – in reality you could use an element of \$\_SERVER.

\$parts is an array of strings each representing a part of this URL divided by a slash, so that \$parts[1] is 'A'.

The first argument to the function call\_user\_func\_array() is therefore the string 'webpageA'. Notice then that this code begins with a URL and ends with a corresponding function call.

The second argument to call\_user\_func\_array() are the url parts however this addition serves no purpose in this small example.

The second set of code using "..." achieves the same result.



## ARGUMENTS



- **func\_get\_args()**
  - from within a function, return its arguments as an array
- **func\_get\_arg()**
  - from within a function, get only one argument
- **func\_num\_args()**
  - from within a function, get the number of arguments supplied

```
function show() {  
    print_r(func_get_args());  
}  
  
show(1, 2, ['a', 'b']);
```

```
Array (  
    [0] => 1  
    [1] => 2  
    [2] => Array (  
        [0] => a  
        [1] => b  
    )  
)
```

Finally there are two additional functions you may come across which pertain to PHP's function definitions.

Within a function definition, func\_get\_args() will return a sequential array of the arguments that function was passed at call time – and func\_num\_args() the count() of this array.

Notice that show() does not define any arguments yet func\_get\_args() returns all the ones which were passed – this is not an error in PHP. The argument list is the minimum amount required, not the maximum. So func\_get\_args() can capture any "extras" that may have been passed.

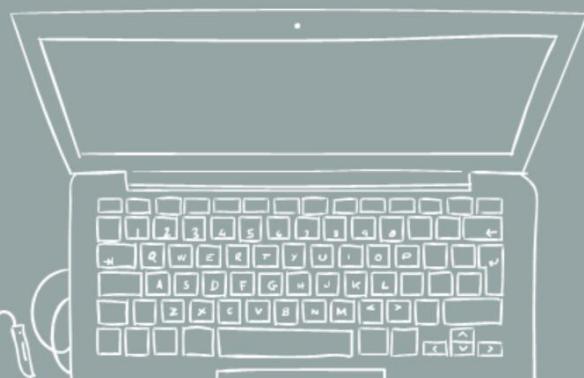
You should review the earlier chapter on function definitions for additional examples on the use of these functions.



# Any questions?



## EXERCISE 14



## RESOURCES

Array Sorting

<http://php.net/manual/en/array.sorting.php>

Function Reference

<http://php.net/manual/en/language.functions.php>

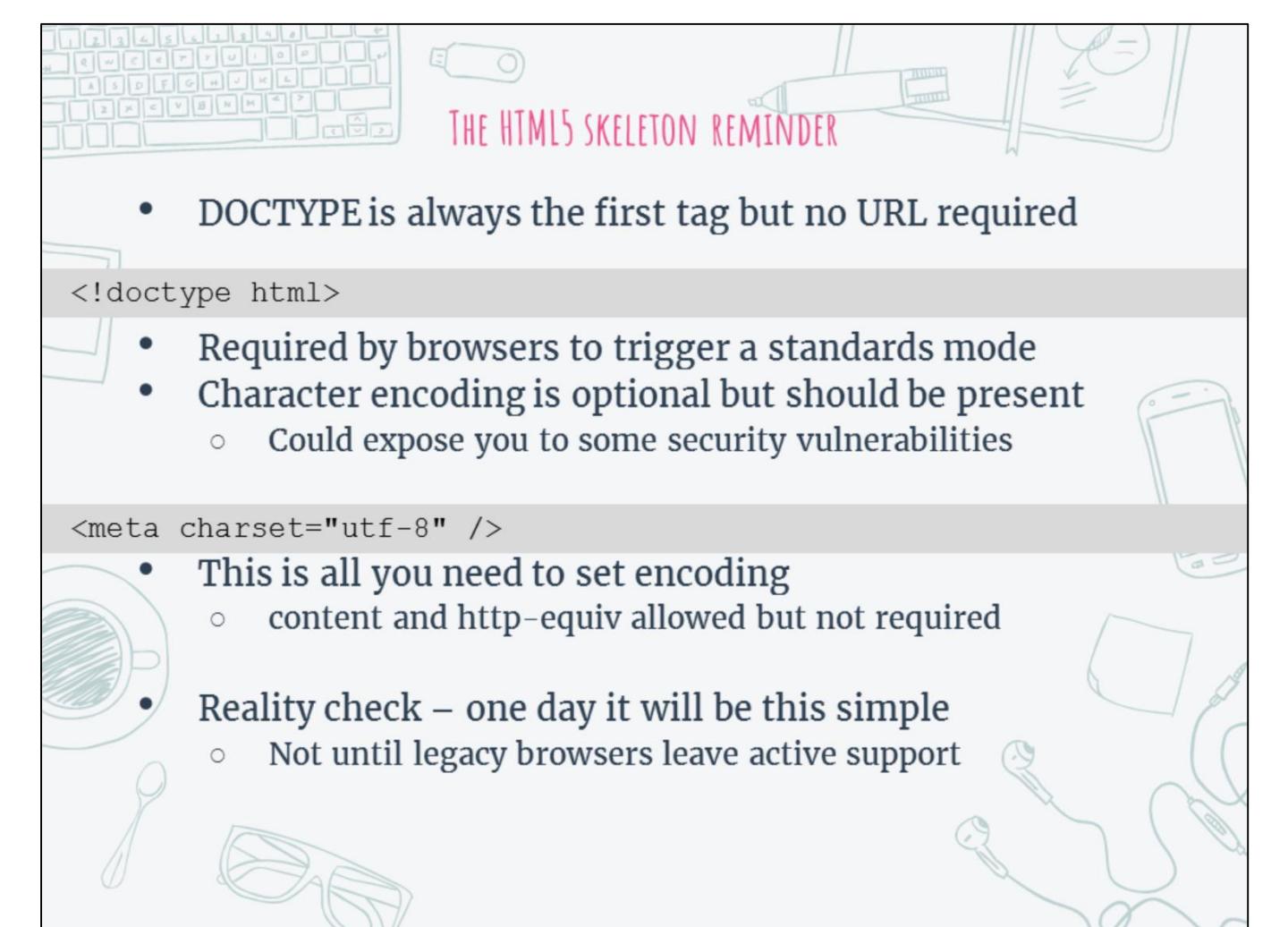
# HTTP & HTML FORMS

## CHAPTER OVERVIEW

- HTML Forms
- The HTTP Protocol
- GET, POST, PUT, DELETE
- The Superglobals
- Streams

### Exercise

HTML Forms & Superglobals



## THE HTML5 SKELETON REMINDER

- DOCTYPE is always the first tag but no URL required

```
<!doctype html>
```

- Required by browsers to trigger a standards mode
- Character encoding is optional but should be present
  - Could expose you to some security vulnerabilities

```
<meta charset="utf-8" />
```

- This is all you need to set encoding
  - content and http-equiv allowed but not required
- Reality check – one day it will be this simple
  - Not until legacy browsers leave active support

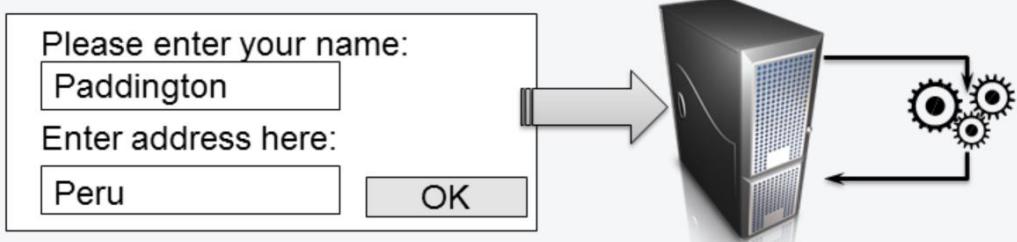
First of all note the simplicity of the doctype code, no need to add in the URL that most HTML authors praise their code editor for supplying. It is an essential tag and most browsers needs it to switch into the right document processing mode.

The encoding is also required. The page will process without it but there are security vulnerabilities that could exploit character encoding vulnerabilities without it including the UTF-7 vulnerability  
<http://code.google.com/p/doctype/wiki/ArticleUtf7>

You may be very used to adding the attributes content="text/html" and http-equiv="content-type". These are entirely optional and QA have opted to take the low attribute usage approach to HTML5.

## UNDERSTANDING FORMS – WHAT ARE FORMS

- Forms allow us to send data to the server for processing



```
<form action="/enrol.php" method="POST" >  
  <input type="text" name="username" />  
  <input type="text" name="address" />  
  <input type="submit" value="OK" />  
</form>
```

### Define:

- Form – action & method
- Inputs – name, type & value

Users can interact with programs running on the server by using HTML forms; in this case the Web server acts as nothing more than a gateway to forward the form's data to the back-end application and to return any generated response. Forms allow the Web to be used for a variety of purposes including user inquiries, ordering and booking systems, and front ends to databases.

A form is described using the `<form>` tag; this will contain other elements and text, including the elements describing the form's input fields. The `FORM` tag takes two attributes: the `method` states how the data is to be sent to the Web server, and the `action` gives the URL of a resource used to process the form. This is normally a server-side program or script but can be a `mailto:` URL.

The values for the `method` attribute determine how the data is sent to the server and takes the value `POST` or `GET`. Depending which you use, the server application needs to be programmed slightly differently.

# UNDERSTANDING FORMS – HTML FORM INPUTS

- Text boxes / areas

```
<input type="text" name="username" value="" />  
<input type="password" name="password" value="" />  
<textarea name="comment" rows="10" cols="40"></textarea>
```

- Checkboxes and radio buttons

```
<input type="checkbox" name="milk" value="CHECKED"/>Milk?<br />  
  
<input type="radio" name="drink" value="tea" />Tea<br />  
<input type="radio" name="drink" value="coffee" />Coffee<br />  
<input type="radio" name="drink" value="choc" />Chocolate<br />
```

- Selections

```
<select name="title">  
  <option value="Dr">Dr</option>  
  <option value="Ms">Ms</option>  
  <option value="Mr">Mr</option>  
</select>
```

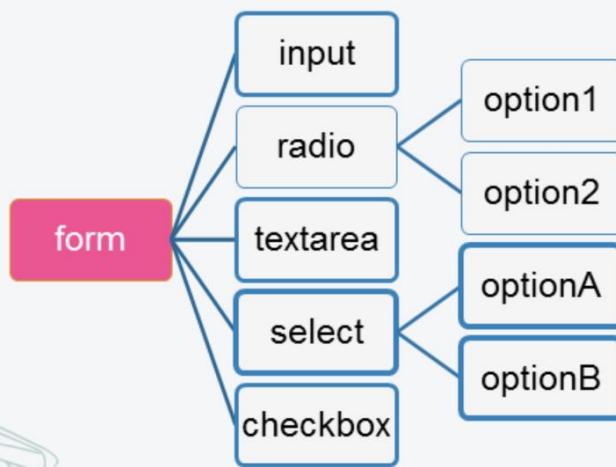
```
<select name="prod" multiple>  
  <option value="a">Apples</option>  
  <option value="p">Pears</option>  
  <option value="g">grapes</option>  
</select>
```

Most form fields are defined using an `<input>` tag; the `type` attribute specifies whether this is a checkbox, radio button, etc. A few field types have their own specific tags, for example `textarea` and `select`. These are shown above.

Each input element has name and a value. The name is specified in the `name` attribute, the value may be specified in the `value` attribute (except for text areas when it is specified by placing text within the `<textarea>...</textarea>` tags) or left for the user to enter (e.g. with text box input). The name-value pairs are how back-end code accesses the data from the form.

## UNDERSTANDING FORMS – HTML HIERARCHY

- The Form is a DOM object and container of other elements
  - When a form is submitted these details are sent to the server
- The majority of the form fields hold a single value
- Radio and Select controls are slightly more complex



When you submit data to a server you send all form DOM (Document Object Model)\* objects contained within it. Most of these controls are simple to work with; they have a value attribute that holds the current user entry. Certain multi option controls like select and radio are slightly more complex to work with.

\*DOM – The Document Object Model is a W3C standard for accessing documents. *"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."*

## FORM METHODS AND EVENTS

- Properties

- action
- encoding (ENCTYPE)
- method
- name
- target
- length



Submit

- Methods

- submit()
- reset()



Reset

The `elements` array and its objects are the most useful properties of the form object, but it does have some others:

The `action`, `encoding`, `method`, `name`, and `target` properties are all reflections of the HTML attributes of the form. The `encoding` property reflects the `ENCTYPE` attribute, and the others all reflect the named attributes.

The `length` attribute specifies the number of elements in the form, and as such has the same value as the `length` property of the `elements` array.

The form object has two methods:

`submit()`, which submits the form immediately, without waiting for the user to press the submit button;  
`reset()` , which clears the form contents and sets all fields back to their default values.

## HTML5 FORM MARKUP

- Forms need to be validated
  - Client side JavaScript
  - Server side checks and balances
- HTML5 provides validation tags that need no scripting
  - Plus much more semantic intent to data
- Supported across all modern browsers
  - Issues for those still using Internet Explorer 9 or less

## TAG HIERARCHY AND OWNERSHIP

- Most form tags had to be within a <form> container
- Now can be placed anywhere and add a form attribute
  - This can be useful when form tags are dynamically added
  - Or the logical form needs to be broken up in the UI

```
<form id="myForm" action="forms.htm" method="get">
    <input type="submit" value="click!" />
</form>
    <!-- not supported in all browsers -->
<input name="myVal" type="text" form="myForm" />
```

A number of elements which were required to be in a form element to be valid and function correctly (<button>, <input>, <label> for instance) can now occur outside the form element. Previously the form would submit all elements that occurred within its container. For these new external items to submit you must add a form attribute to the external element. This denotes a form owner for the element.

The main purpose of this is flexibility in layout allowing you to place elements where you desire on the form structure.

## NEW INPUT ELEMENTS

- The HTML5 spec started with forms
  - Opera and Safari are the driving force and most complete
- 13 new type options to date
- Mostly extend the <input> tags with new type values
  - If a browser does not understand the extension it is rendered as:

```
<input type="text" />
```

- No requirement in the spec for how browsers present
  - Different browsers show different UI and error messages
  - Browsers that do not understand the new types treat them as text
  - JavaScript defence is needed for legacy browsers

Consistent rendering is important and if any browser fails to understand the date type it will drop back to treating it as a default input="text" (it is actually optional to use the input attribute just typing <input> will give you a text box).

This gives us some consistency in handling browsers which do not yet implement the data types we are about to explore. If a browser fails to support the new HTML5 types support we can defend against this and revert to javascript:

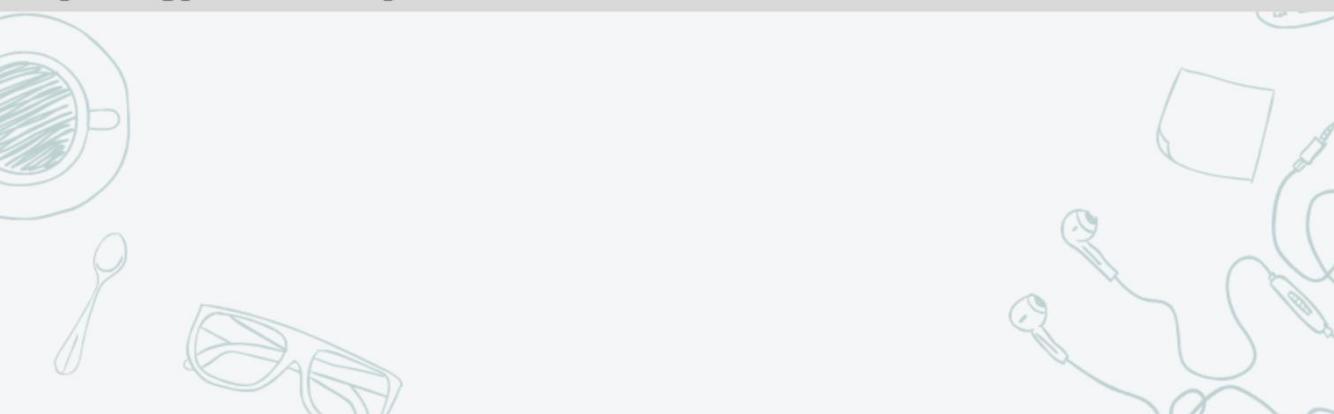
```
<script>
    var i = document.createElement("input");
    i.setAttribute("type", "date");
    if (i.type == "text")
    {
        // No native date picker support
    }
</script>
```

## PLACEHOLDER ATTRIBUTE

The placeholder attribute offers default text

- Gives the user example or instruction for the field
- Sometimes called a watermark
- Can only be used for text values
- Is not a default value

```
<input type="text" placeholder="I am a watermark" />
```



The first improvement HTML5 brings to web forms is the ability to set placeholder text in an input field. Placeholder text is displayed inside the input field as long as the field is empty and not focused. As soon as you click on (or tab to) the input field, the placeholder text disappears.

## AUTOFOCUS ATTRIBUTE

- It is common to set the focus to the first form field
  - To have the cursor flashing ready to type
- Previously achieved with JavaScript
  - The markup representation is faster
  - Part of the page rendering rather than code execution
- Supported in all browsers other than IE9 and less:
  - Use JavaScript to support legacy browsers

```
<form>
    <input name="q" autofocus="true">
    <input type="submit" value="Search">
</form>
```

HTML5 introduces an autofocus attribute on all web form controls. The autofocus attribute does exactly what it says on the tin: as soon as the page loads, it moves the input focus to a particular input field. But because it's just mark up instead of script, the behaviour will be consistent across all web sites.

## REQUIRED FIELDS

- You can force a field to be mandatory on the client

```
<input type="text" autofocus="true" required />
```

- On a submit action an error message will appear in the following browsers:
  - Safari 6+
  - Firefox 4.0
  - Opera 9+
  - Chrome 9+
  - IE10+
- Message will appear differently in each browser



! Please fill in this field.

## EMAIL INPUT TYPE

```
<input type="email" />
```

Browsers behave differently

- Opera and safari provides submit validation
- Firefox provides client validation on blur
- Safari mobile changes the input keyboard
- IE 9< does nothing

Form will not submit until the error is solved

Provides a simple input mask to check input

e.g. boffin@qa.com



This is where the different browsers become key and an interesting part of the standard, at this point, demanding more of an interface approach than a solid functionality requirement. Different browsers behave in different ways. If we are a Firefox user right now we have form validation without the developer doing a jot of code. If we are an IE user we see nothing different. If you are a mobile user the touch screen of an iPhone for instance loads up a different keyboard suited to emails.

It is an easy win for a small change!

## WEB ADDRESS INPUT TYPE

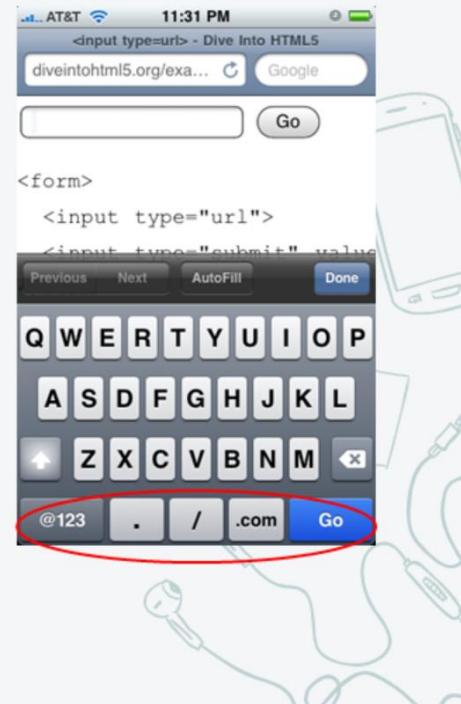
```
<input type= "url" />
```

Denotes the input must have schema  
prefixing an address e.g. <http://www.qa.com>  
or <ftp://ftp.example.com>

Browsers behave differently

- Chrome and Firefox 4 force user to add schema
- Opera prefixes an address with http://
- Safari mobile provides a different keyboard

To use effectively combine with a placeholder  
to ensure the user understands



The input of this in the major browsers is a bit more of a scrap at the moment with IE, even IE9 bringing up the rear. In this environment Safari and Firefox 4 force the user to enter a http:// address to pass the validation. While Opera instead adds the http prefix upon submission of the form.

Microsoft have not implemented any input support at this time.

While the iPhone / iPad brings a different virtual keyboard up if you have set the field accordingly

## DATE INPUT TYPE

- A popup calendar is standard for date selection
  - Normally requires a JavaScript framework
  - Around 10% of web users do not support javascript
- HTML5 defines six date time types
  - Use UTC in the same way as time element
- Limited browser support currently
  - Opera leads the way
  - Chrome has some support
- Will not require javascript enabled
  - Native support is the ultimate aim
- ECMAScript5 allows you to create dates from UTC

Within forms, constraint of users input and providing a user with effective JavaScript selection, is a UI developers bread and butter. In recent years this has often involved a library like jQuery to quickly create rich UI functionality such as calendar widgets. The forms section of the HTML5 spec expects, for full support, there will be widget functionality provided with no code required, just set the input type.

Currently only Opera fully implements this in the browser with Chrome offering more limited support. These functionalities will work with or without JavaScript enabled in the client browser.

Input Type	Definition
date	Reflects a UTC date
month	Choice for only month and year
week	Select only a specific week
time	Time only in UTC format
datetime	Reflects a UTC date and time



## PATTERN



- The pattern attribute allows use of regular expressions

```
<input type="text" pattern="[0-9]{13,16}" name="CreditCardNo"/>
```

- Pattern works with the following input types:

- text
- search
- url
- Tel
- email
- password

- Ensure the user understands the regular expression

- Support with a placeholder



The pattern attribute specifies a regular expression that the <input> element's value is checked against.

Note: The pattern attribute works with the following input types: text, search, url, tel, email, and password.

Useful pattern generation website:

<http://html5pattern.com/>

## SUBMIT/RESET INPUT TYPES

- The type=“**submit**” attribute setting is the easiest way to allow a user to submit a form

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

- On clicking, form action is automatically executed
- The type=“**reset**” attribute setting clears the form to its default state when clicked

```
<input type="reset">
```

- These inputs can be customised by adding other attributes
  - E.g. the *value* attribute can be set to control the text that appears

HTML5 introduced better controls for forms. The submit and reset input types are a quick and easy way to add functionality to the form without the need for lots of additional JavaScript programming. Other attributes can be set for these input types, as with all input types. For more information see:  
[http://www.w3schools.com/tags/tag\\_input.asp](http://www.w3schools.com/tags/tag_input.asp)

## BUTTON ELEMENT

- The button element defines a clickable button

```
<button type="button">Click Me!</button>
```

- <button> allows content like text and images to be put inside it
  - <input> is a self-closing tag and therefore no content can be put inside it apart from text in the *value* attribute
- Recommended to specify type for the <button> element
  - Different browsers have different default types
  - They can be set and used as **submit** and **reset** too

The <button> element is intended to provide additional functionality to buttons on forms. The button element allows the addition of icons and images within buttons which may help the look, feel and usability of a page. The onclick attribute can be set to execute some other code before the form's action is performed.

e.g. To present the user with a popup 'Thanks' box before performing the form action would require the following code:

```
<button type="submit" id="submit" onclick="alert('Thanks')">Submit</button>
```

# HTTP REQUESTS

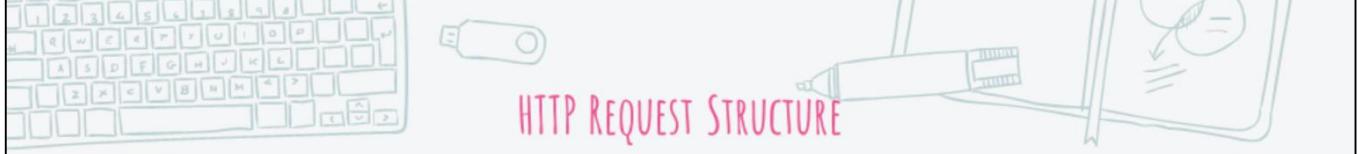
What happens when the user clicks the submit button?

How does the browser and server know what to do with the information inputted on the form?

HTTP requests and responses are a major part of this.

## HTTP REQUESTS AND RESPONSES

- When a client connects to a server it sends an HTTP request which contains:
  - Metadata representing information about the request
  - The data of the request itself, e.g. login information
- The server responds with an HTTP response which has a similar structure:
  - Metadata (e.g. the content length)
  - The content (e.g. a webpage)
- Browsers (Chrome) and servers (PHP –S, Apache,...) automatically send and receive these requests



## HTTP REQUEST STRUCTURE

HTTP Requests have

1. Request Line
2. Zero or more header fields followed by CRLF (\r\n)
3. An empty line (CRLF) indicating the end of the header fields
4. An optional message body (i.e. data / content)

```
POST /php/myapplication.php HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5; Windows NT)
Host: www.example.com
Content-Type: application/x-www-form-urlencoded
Content-Length: length
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive

key=value&username=example&email=me@example.com
```

The first line of the request is the REQUEST LINE. It contains the request method (in this case POST), the URI that made the request (in this case /php/myapplication.php) and the HTTP version used to send the request (in this case HTTP/1.1). A CLRF is at the end of the request line.

The rest of the request is made up of the headers, which are in "Name : Value" pairs. Each pair contains information about the HTTP request and the browser that sent it. More information about the "Name : Value" pairs can be found at:

<https://www.w3.org/Protocols/rfc2616/rfc2616-sec5.html#sec5.3>

The final line of the request is the message body, that is an optional part of the request. If it is available, it is used to carry the entity-body associated with the request. If included, the Content-Type and Content-Length part of the header specify the nature of the message included. It carries the actual HTTP request data including form data and uploaded file data.

## HTTP VERBS

- GET -- Retrieve information for a given URI
- POST -- Send data to the given URI
- PUT -- Replace data at the given URI
- DELETE -- Remove data at the given URI

```
GET /index.php HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5; Windows NT)
Host: www.example.com
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
```

```
DELETE /users/id/1012 HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5; Windows NT)
Host: www.example.com
```

The METHODS used in HTTP requests are most often the four listed above.

The GET request method is used to 'get' information from the specified server using a given URI. GET requests should only retrieve data and should have no other effect on it.

The POST request method is used to send data to the server, for example, form data, file uploads, etc.

The PUT request method is used to replace data at the target with the data included in the request.

The DELETE request method removes all current representations of the target resources given by URI

Others, often not implemented or of limited use are:

HEAD – same as GET but only transfers status line and header sections

CONNECT – Establishes a tunnel to the server identified by a given URI

OPTIONS – Describes communication options for the target resource

TRACE – Performs a message loop back test along with the path to the target resource.

## HTTP RESPONSE STRUCTURE

HTTP Responses have

1. a status line
2. Zero or more header fields followed by CRLF (\r\n)
3. An empty line (\r\n)
4. An optional message body (data, content)

```
HTTP/1.1 200 OK
```

```
Date: Mon, 21 Mar 2016 09:15:56 GMT
```

```
Server: Apache/2.2.14 (Win32)
```

```
Last-Modified: Mon, 21 Mar 2016 09:14:01 GMT
```

```
Content-Length: 88
```

```
Content-Type: text/html
```

```
Connection: Closed
```

```
<html>
<body>
<h1>Hello, World!</h1>
</body>
</html>
```

These are similar to the HTTP requests seen earlier. They are used to allow the server to respond to a request.

The first line of a response is the STATUS LINE that contains the protocol version being used to transmit the response followed by a numeric status code and a textual interpretation of this. The status codes follow these patterns:

- 1xx: Informational – Request received and is being processed
- 2xx: Success – Action successfully received, understood and accepted
- 3xx: Redirection - Further action required to complete the request
- 4xx: Client Error – Request contains incorrect syntax or cannot be fulfilled
- 5xx: Server Error – Server has failed to fulfill apparently valid request.

For a valid request, in most cases, a “200 OK” status will be issued. You may have seen others like a “404 Not Found” when you have tried to access a web page on a site but typed in the address incorrectly. A full list of status codes can be seen at: <https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html>

The response headers are similar to those used in a request, containing information about the server software, when the file was last modified, the Content-Type, etc. Again, most are optional.

The final part of the response is the response message that is to be served back to the request. This can be code or data, as shown above, an HTML page has been sent.

## HTTP STATUS CODES

- **1\*\*** e.g. 102 Processing
  - Informational responses – e.g. processing still going on
- **2\*\*** e.g. 200 OK
  - Successful responses
- **3\*\*** e.g. 301 Moved Permanently
  - Redirection – resource has moved to a different location
- **4\*\*** e.g. 404 Not Found
  - Client Error – a problem with the requesting client
- **5\*\*** e.g. 500 Internal Server Error
  - Server Error – a problem with the responding server



## EXAMPLE HTTP RESPONSE 404

HTTP/1.1 404 Not Found

Date: Mon, 21 Mar 2016 10:36:20 GMT

Server: Apache/2.2.14 (Win32)

Content-Length: 230

Connection: Closed

Content-Type: text/html; charset=iso-8859-1

```
<html>
<head>
    <title>404 NOT FOUND</title>
</head>
<body>
    <h1>Oops!</h1>
    <p>The requested URL example.com/missing.php was not
found!</p>
</body>
</html>
```



GET

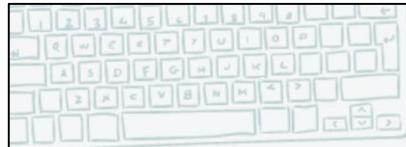


- GET requests have no request body
- The URI specifies most of the data the server requires
  - The protocol e.g. HTTP
  - The host e.g. example.com
  - The resource location e.g. /index.php
  - The query string e.g. [?username=eg&email=foo@example.com](http://example.com/index.php?username=eg&email=foo@example.com)

`http://example.com/index.php?username=eg&email=foo@eg.com`

- The query string contains associative key/value pairs
- Technically limited to 255 characters however in practice they can be over 1000 characters long
- Other information can be supplied with headers, either standard ones or custom headers typically preceded by "X-"





POST



- POST requests include a message body
- They can use the query string system and the message body to send data
- Larger or more sensitive information is usually sent via POST
  - The message body can have a practically unlimited size (usually limited by what the server will accept e.g. 20MB)
  - The browser sends the POST body "behind the scenes" whereas the URL is visible in the address bar
- POST messages are not more secure than GET messages as both are sent in plain text over the network
  - Use HTTPS for encryption



# PHP SUPERGLOBALS

## PHP'S HTTP WORLD AND THE SUPERGLOBALS

- PHP processes the (string) request into associative arrays which are "superglobal"
  - When an HTTP request is sent to the webserver (e.g. Apache) it is typically forwarded to PHP
  - Accessible everywhere without the global keyword
- 
- **`$_GET`** – key-value pairs from the URL's Query String
  - **`$_POST`** – key-value pairs from the POST message body
  - **`$_SERVER`** – some headers and server information
  - **`$_FILES`** – uploaded file data (sent as a type of POST)
  - **`$_COOKIE`** – special data headers sent with requests
  - **`$_ENV`** – server and system information (environment variables)

There is also `$_REQUEST`, a combination of `$_GET` and `$_POST`.

It's not typically a good idea to use this, its always clearer to know where your data is coming from... use `$_GET` or `$_POST`.

Imagine reading code which said `$_REQUEST['id']` – has that been sent via the query string or not?

However there may be services where you want to accept both – consider first `$_GET['id'] ?? $_POST['id']` since this shows that `$_GET` has the priority.

There is an additional superglobal which we will look at later: `$_SESSION`.

Finally, `$GLOBALS` (no leading underscore!) contains an associative array of all global variables. This bypasses the global keyword, since the array is a superglobal.



## SUPERGLOBALS

Suppose at <http://example.com/index.php> there is a file which outputs the superglobals

- we send a request to this location...

```
//curl code
$ch = curl_init();
curl_setopt($ch,
CURLOPT_URL,"http://example.com/?name=Sherlock");
curl_setopt($ch, CURLOPT_POST, true);
curl_setopt($ch, CURLOPT_POSTFIELDS,
"username=example&email=me@example.com&password=foo");
```

 which sends...

```
//HTTP request generated
```

```
POST /?name=Sherlock HTTP/1.1
Host: example.com
Content-Type: application/x-www-form-urlencoded
username=example&email=me@example.com&password=foo
```

curl (ClientURL) is an open source command line tool and library for transferring data with URL syntax. It is used in command lines or scripts to transfer data. It is free and open source software that compiles and runs under a wide variety of operating systems.

In the code above, `curl_init()` and `curl_setopt()` are a predefined function of the library that sets an option for a curl transfer. The `curl_setopt()` function has 3 arguments:

- \$ch – a curl handle returned by the `curl_init()` function
- \$option – the `CURLOPT_XXX` to set
- \$value – the value to be set on option

`CURLOPT_URL`, `CURLOPT_POST` and `CURLOPT_POSTFIELDS` are predefined constant integer values.



## SUPERGLOBALS AVAILABLE IN PHP

```
Array (   //$_GET
    [name] => Sherlock
)
Array (   //$_POST
    [username] => example
    [email] => me@example.com
    [password] => foo
)
Array (   //$_COOKIE
    [cookieKey] => cookieValue
)
Array (   //$_SERVER
    [SERVER_SOFTWARE] => PHP 7.0.3RC1 Development Server
    [SERVER_NAME] => localhost
    [SERVER_PORT] => 80
    [REQUEST_URI] => /OutputSupers.php?name=Sherlock
    [REQUEST_METHOD] => POST
    [CONTENT_LENGTH] => 50
    [CONTENT_TYPE] => application/x-www-form-urlencoded
    [REQUEST_TIME] => 1458566343 //...abridged!
```

This shows the arrays that would be generated through the use of the code on the previous page.

## THE SERVER ENVIRONMENT

The web server and operating system define variables related to the broader execution environment in `$_SERVER` and `$_ENV`

Some items of interest include

- `REQUEST_METHOD`
  - the HTTP request verb
- `HTTP_REFERER`
  - the URL the user came from
- `REQUEST_URI`
  - the URL without the host
- `HTTPS`
  - non-empty if using https
- `HTTP_USER_AGENT`
  - browser name-version string

# PHP ALTERNATIVE SYNTAX

## PHP ALTERNATIVE SYNTAX

- PHP offers an alternative syntax that more easily allows you to jump between PHP and other languages, such as HTML
- This is useful when creating template files (.phtml)
- Alternative syntax exists for:

if

switch

while

for

foreach

Opening braces become a colon :

The closing brace is replaced by an end statement e.g.  
endif; endswitch; endwhile; endfor; endforeach;

When working with pages that switch back and forth between php and html (such as phtml template pages) it can be tricky keeping track of code blocks and can result in the over use of the `echo` function. Luckily php supports an alternative syntax for control structures allowing you to jump in and out of php and html. Any code (html, php, anything) nested within a conditional will only parse if the defined conditions are matched. The end result is clean, organized, efficient code.

PHP's alternative control structure syntax makes it easy to identify code blocks and it's widely used in many popular open source projects, such as WordPress.

PHP provides this alternative syntax for if, while, for, foreach, and switch statements. Any opening braces (`{`) you would normally use become a colon (`:`), and the closing brace is replaced by an `endif;`, `endwhile;`, `endfor;`, `endforeach;`, or `endswitch;`, respectively. PHP provides this alternative syntax for if, while, for, foreach, and switch statements. Any opening braces (`{`) you would normally use become a colon (`:`), and the closing brace is replaced by an `endif;`, `endwhile;`, `endfor;`, `endforeach;`, or `endswitch;`, respectively. I included an example of each below.

## PHP ALTERNATIVE SYNTAX: EXAMPLES IF & WHILE

```
<?php $a=1; $b=4;?>
```

```
<p>----Example IF Statement<br>
<?php if ($a == $b): ?>
A equals B<br>
<?php else:>
A does not equal B<br>
<?php endif; ?></p>
```

```
<p>----Example WHILE Statement<br>
<?php while($a < $b): ?>
Here is some code inside a while<br>
<?php
$a++;
endwhile; ?>
</p>
```

Above are examples of an if and a while using alternative syntax.

## PHP ALTERNATIVE SYNTAX: EXAMPLES FOR & SWITCH

```
<p>----Example FOR Statement<br>
<?php for($i = 1; $i <= 10; $i++):?>
Some code in a for<br>
<?php endfor; ?>
</p>
```

```
<p>----Example SWITCH Statement<br>
<?php switch($a): case 1:>
```

```
    Code block for CASE 1<br>
    <?php break;?>
    <?php case 2: ?>
    Code block for CASE 2<br>
    <?php break;?>
    <?php case 3: ?>
    Code block for CASE 3<br>
    <?php break;?>
    <?php case 4: ?>
    Code block for CASE 4<br>
    <?php break;?>
<?php endswitch; ?>
</p>
```

Above are examples of a for and a switch using alternative syntax. Note that the first case label in the switch statement is included within the switch block. This is because the php parser does not allow any whitespace between the switch block and the first case statement.

## PHP ALTERNATIVE SYNTAX: EXAMPLE LOGIN FORM

```
<?php  
$user = true;
```

PHP

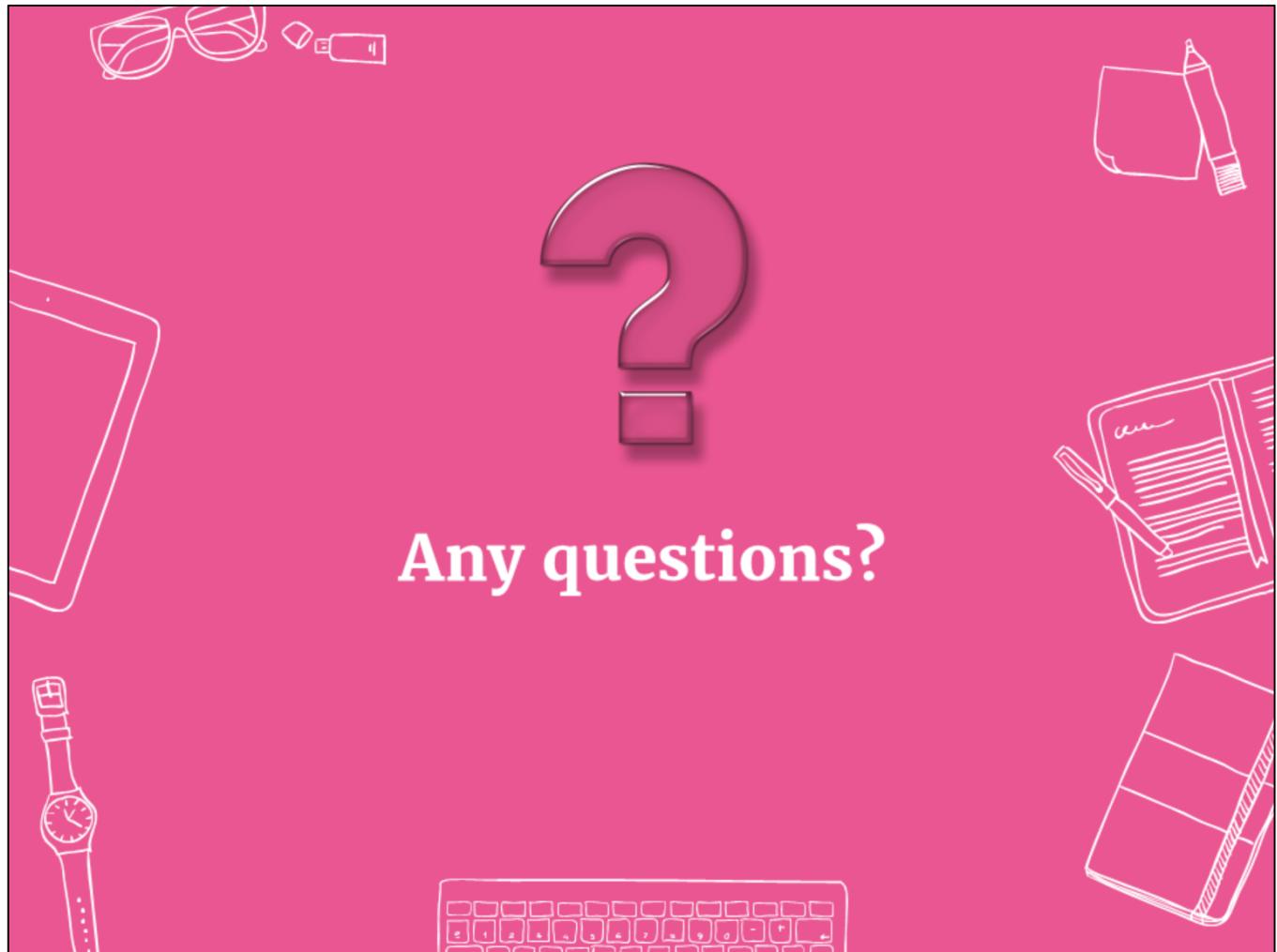
```
if($user == true):?  
    <div class="user_message">  
        <h2>Welcome! You have logged in.</h2>  
    </div>
```

```
<?php else:?  
    <div class="sign_in">  
        Please sign in.  
        <form>  
            username: <input type="text">  
            password: <input type="password">  
            <input type="submit">  
        </form>  
    </div>
```

HTML

```
<?php endif; ?>
```

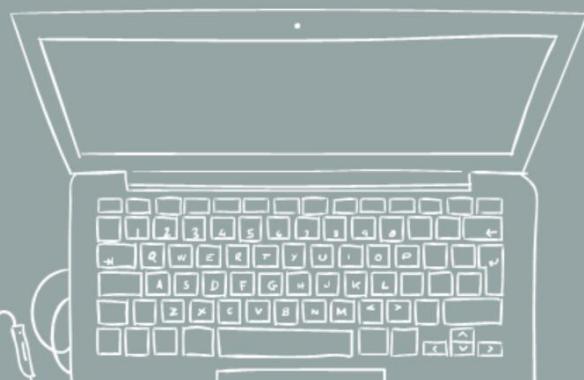
This example displays two different blocks of html code depending on the value of \$user. You could retrieve the user value from your \$\_GET or \$\_POST superglobals.



**Any questions?**



## EXERCISE 15



## RESOURCES

Superglobals

<http://php.net/manual/en/language.variables.superglobals.php>

Alternative Syntax

<http://php.net/manual/en/control-structures.alternative-syntax.php>

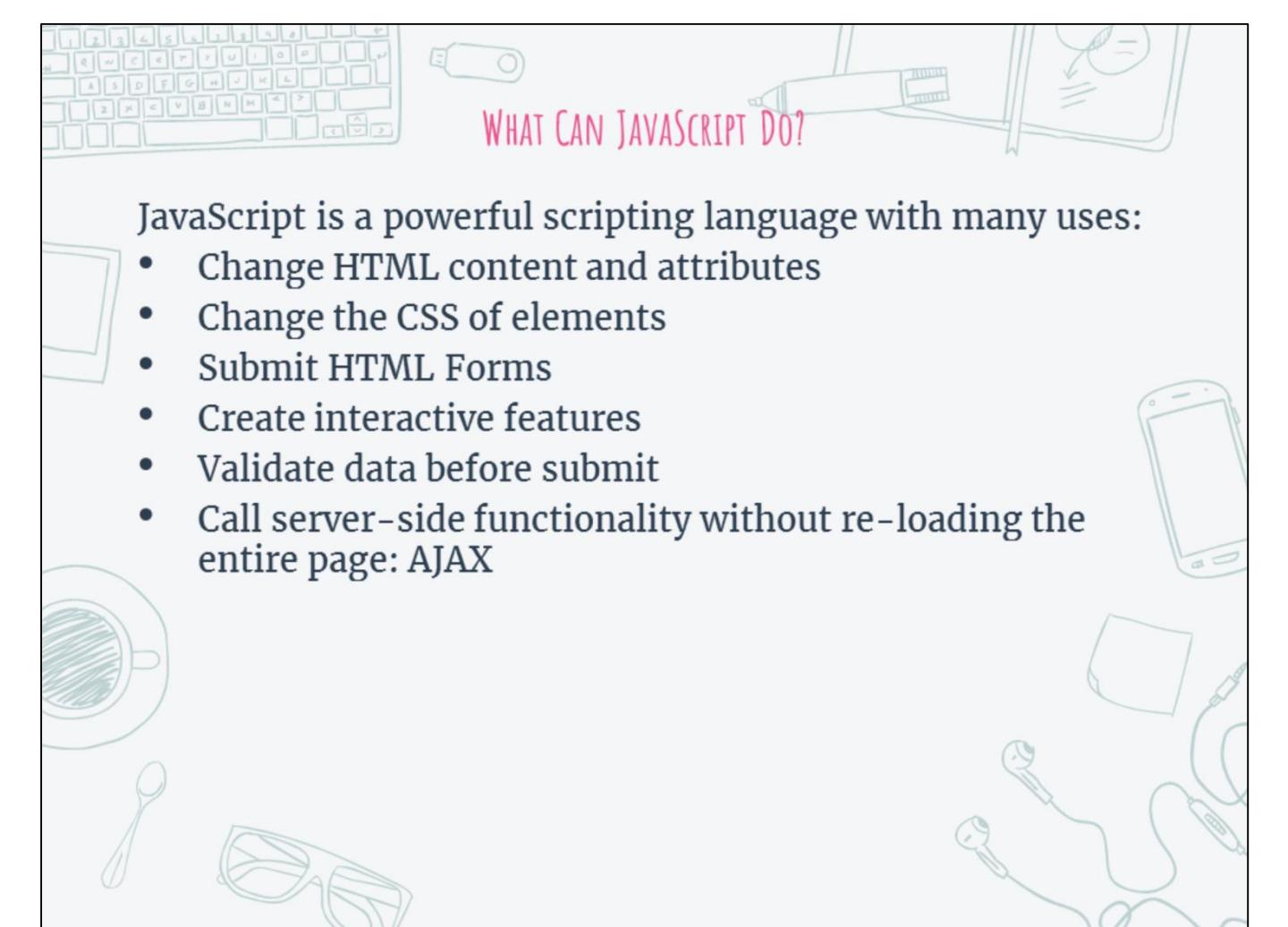
# JAVASCRIPT & JQUERY

## CHAPTER OVERVIEW

- What can JavaScript do?
- Where to put JavaScript
- JavaScript syntax
- An introduction to jQuery

### Exercise

#### JavaScript & jQuery

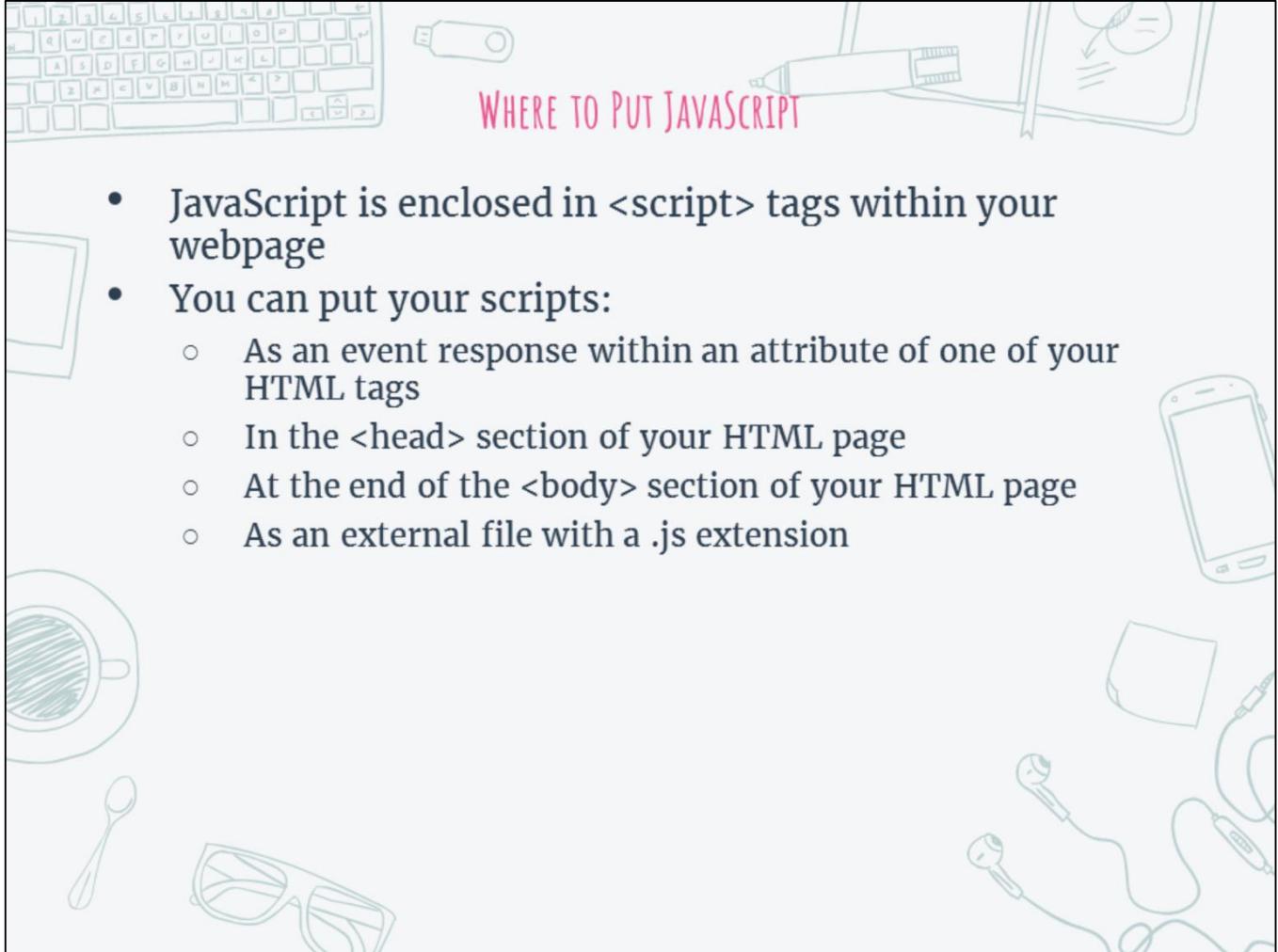


## WHAT CAN JAVASCRIPT DO?

JavaScript is a powerful scripting language with many uses:

- Change HTML content and attributes
- Change the CSS of elements
- Submit HTML Forms
- Create interactive features
- Validate data before submit
- Call server-side functionality without re-loading the entire page: AJAX

NOTE: JavaScript is case-sensitive.



## WHERE TO PUT JAVASCRIPT

- JavaScript is enclosed in <script> tags within your webpage
- You can put your scripts:
  - As an event response within an attribute of one of your HTML tags
  - In the <head> section of your HTML page
  - At the end of the <body> section of your HTML page
  - As an external file with a .js extension

## JAVASCRIPT IN AN ATTRIBUTE

```
<!DOCTYPE html>
<html>
<body>

<h2>What Can JavaScript Do?</h2>

<p id="demo">JavaScript can change HTML content.</p>

<button type="button"
onclick="document.getElementById('demo').innerHTML =
'Hello JavaScript!'">Click Me!</button>

</body>
</html>
```

The JavaScript is contained within the **onclick** attribute of the button. When the button is clicked the script will run and change the content of the item with the ID of “demo” to “Hello JavaScript!”.

## JAVASCRIPT IN THE <HEAD>

```
<!DOCTYPE html>
<html><head>
<script>
function myFunction() {
    document.getElementById("demo").innerHTML =
"Hello from JavaScript in the head section. ";
}
</script>
</head>
<body>
<h1>A Web Page</h1>
<p id="demo">A Paragraph</p>
<button type="button" onclick="myFunction()">Try
it</button>
</body>
</html>
```

The JavaScript is contained within the `<head>` section of the HTML document. The script is called when an event happens i.e. when a user clicks on a button the **myFunction** JavaScript function is called.

JavaScript accesses elements and attributes of the HTML page by using the DOM (Document Object Model) methods. The above method is **getElementById** and is used to return the element that has the ID attribute with the specified value of "demo".

For more information on DOM methods see: [https://www.w3schools.com/jsref/dom\\_obj\\_document.asp](https://www.w3schools.com/jsref/dom_obj_document.asp)

## JAVASCRIPT IN THE <BODY>

```
<!DOCTYPE html>
<html>
<body>

<h1>A Web Page</h1>
<p id="demo">A Paragraph</p>
<button type="button" onclick="myFunction()">Try
it</button>

<script>
function myFunction() {
    document.getElementById("demo").innerHTML = "Hello
from JavaScript in the body section.";
}
</script>

</body>
</html>
```

The JavaScript is contained within the `<body>` section of the HTML document. This can yield better performance than placing your scripts within the `<head>` section because a large script in the head section will delay the rest of the HTML page as it tries to load.

## EXTERNAL JAVASCRIPT FILES

```
function myFunction() {  
    document.getElementById("demo").innerHTML =  
    "Hello from an external JavaScript file."  
}
```

myScript.js

```
<!DOCTYPE html>  
<html>  
<body>  
<p id="demo">A Paragraph.</p>  
  
<button type="button" onclick="myFunction()">Try  
it</button>  
  
<script src="myScript.js"></script>  
  
</body>  
</html>
```

The JavaScript is contained within an external file saved as myScript.js

You link to the JavaScript file using the src attribute of the script tag.

Placing scripts in external files enables you to separate HTML and code and makes the HTML and JavaScript easier to read and maintain. Page load time can be increased because JavaScript files can be cached by the browser. If you want to include several script files to one page, use a script tag for each file.

## THE JAVASCRIPT LANGUAGE

JavaScript is a case-sensitive programming language made up of:

- Variables defined using the var keyword: **var x = 5;**
- Fixed literal values: **42**
- Comments: **// var y = 6;**
- Expressions: **2 + 2**
- Statements that end with semicolons: **var z = x + y;**
- Keywords: **if...else, return, switch, var, for** etc
- Functions

```
function myFunction() {  
    document.getElementById("demo").innerHTML =  
    "Hello from a function.";  
}
```

JavaScript is a case-sensitive programming language made up of typical language constructs such as variables, literals, comments, expressions, statements, keywords and functions. You should see a lot of familiarity between the PHP language and JavaScript.



# ACTIVITY



Your instructor will walk you through some JavaScript activities.



jQuery is a JavaScript Library that greatly simplifies JavaScript programming. It is a lightweight, "write less, do more", JavaScript library. jQuery takes a lot of common tasks that require many lines of JavaScript code to accomplish, and wraps them into methods that you can call with a single line of code. jQuery also simplifies a lot of complicated tasks from JavaScript, such as AJAX calls and DOM manipulation.

The jQuery library contains the following features:

HTML/DOM manipulation

CSS manipulation

HTML event methods

Effects and animations

AJAX

Utilities

Plugins for almost any task out there.

## USING JQUERY

Two options for using jQuery:

1. Download jQuery using [jQuery.com](http://jquery.com)

```
<head>
<script src="jquery-3.2.1.min.js"></script>
</head>
```

2. Refer to a hosted version using a Content Delivery Network (CDN)

```
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery
/3.2.1/jquery.min.js"></script>
</head>
```

To use jQuery you can either download it from [jQuery.com](http://jquery.com) or refer to a hosted version using a CDN (Content Delivery Network) such as Google, Microsoft or MaxCDN. The example above links to Google's jQuery files.

## JQUERY SYNTAX

```
$(this).hide() - hides the current element  
$("p").hide() - hides all <p> elements  
$(".demo").hide() - hides all elements with class="demo"  
$("#demo").hide() - hides the element with id="demo"
```



With jQuery you select (query) HTML elements and perform "actions" on them. The jQuery syntax is tailor-made for **selecting** HTML elements and performing some **action** on the element(s).

The basic syntax is: **`$(selector).action()`**

jQuery is aliased with a \$ (dollar) sign as a shortcut.

A *(selector)* to "query (or find)" HTML elements

A jQuery *action()* to be performed on the element(s)

Examples:

`$(this).hide()` - hides the current element.

`$("p").hide()` - hides all <p> elements.

`$(".demo").hide()` - hides all elements with class="demo".

`$("#demo").hide()` - hides the element with id="demo".



# ACTIVITY



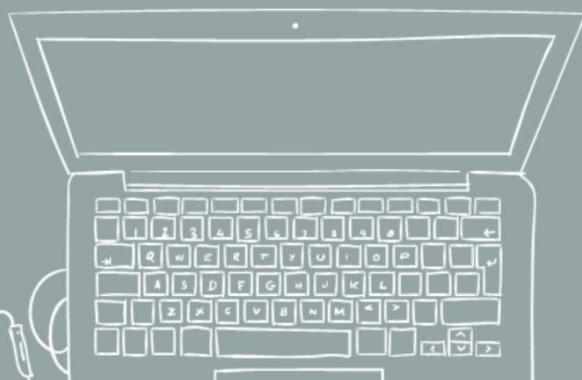
Your instructor will walk you through some jQuery activities.



# Any questions?



## EXERCISE 16



## RESOURCES

JavaScript Tutorial

<https://www.w3schools.com/js/default.asp>

The HTML DOM Document Object

[https://developer.mozilla.org/en-US/docs/Web/API/Document\\_Object\\_Model](https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model)

[https://www.w3schools.com/jsref/dom\\_obj\\_document.asp](https://www.w3schools.com/jsref/dom_obj_document.asp)

jQuery Tutorial

<https://www.w3schools.com/jquery/default.asp>

jQuery

<https://jquery.com/>

jQuery UI

<http://jqueryui.com/>

# UPLOAD FORMS & STATE

## CHAPTER OVERVIEW

- HTTP Requests
- \$\_POST
- Forms
- Uploads
- Filter Functions
- File Functions
- HTTP is Stateless
- Cookies
- Sessions
- Authentication

### Exercise State

This chapter is vitally important for your understanding of PHP. The bread and butter of PHP development is taking input from the URL or forms and "doing something with it". Thankfully PHP makes the whole process very simple.

By the end of this chapter you should understand the workhorse of PHP, superglobals, are and how they are used.

By the end of this chapter you should understand what a cookie is and how they relate to PHP's sessions.

The material in this chapter forms the basis of nearly every PHP authentication system there is – if you login to a web site written in PHP the techniques discussed in this chapter will have been used.

However these concepts often trip up programmers new to web development or new to PHP. Throughout this course the distinction between HTTP request data and PHP data has been emphasised, in this chapter in particular, understanding the distinction will be important.

It is an over simplification to say PHP is the language in which HTML forms are processed. However less dramatic than it first appears.

Most of the user's interaction with a site is either via GET, that is, browsing to it or via POST typically submitting a form. Web designers have evolved ever more complex ways of laying out forms but this is what it boils down to.

In this chapter we will look at forms and the data handling abilities of PHP which go hand-in-hand with them.

## BROWSER REQUESTS: FORMS AND AJAX

- The easiest way of sending HTTP requests is with the browser
  - the address bar implies a GET request
  - forms can send POST or GET requests
  - and a special type of POST request for uploading files
  - Forms cannot send PUT, DELETE and other HTTP verbs
- An alternative is to use AJAX
  - a JavaScript mechanism to generate and send HTTP requests as the page runs
  - AJAX is able to send other HTTP verbs
- The browser will generate a specific HTTP request given
  - the structure of the form (what kinds of inputs there are)
  - the name of each input (, textarea, button, ...)
  - the value of each input

The two key HTTP verbs for the web are GET and POST.

An HTTP GET request is sent whenever your browser connects to a site. It determines which IP address of the machine to connect to and then sends the GET request. The URL may optionally contain a "query string" or set of key-value pairs which follow a question-mark.

<http://example.com?username=sholmes&id=10>

The second most important HTTP verb is POST. An HTTP POST request is sent to a site whenever a user submits a form. Similarly it contains a set of key-value pairs, however in the request body, rather than the URL.

Therefore it is possible to sent a POST request to a URL with query parameters. To submit, for example, a form to the url [example.com/?a=b](http://example.com/?a=b)

## STRUCTURE OF \$\_POST, \$\_GET

```
<input type="text" name="firstname">      $_POST == Array (  
<input type="password" name="pwd">        [firstname] => test  
<input type="checkbox" name="cbox">        [pwd] => test  
   [cbox] => on  
  
<select name="car">                      [car] => 0  
  <option value="0">Saab</option>        [gender] => female  
  <option value="1">Fiat</option>         [message] => A sample Message  
  <option value="2">Audi</option>        [submit] =>  
</select></label></div>                  )  
  
<input type="radio" name="gender" value="male">  
<input type="radio" name="gender" value="female" checked>  
<input type="radio" name="gender" value="other">  
<input type="radio" name="gender" value="nospec">  
  
<textarea name="message" rows="10" cols="30">  
A sample Message  
</textarea>  
  
<button type="submit" name="submit">Send</button>
```

Whenever a form is submitted the browser builds up an HTTP POST request and fills its body with key-value pairs corresponding to the name-value pairs of form fields.

PHP automatically populates an associative array called `$_POST` whose keys are the name of the form fields and whose values are their values.

This array is known as a superglobal and is available anywhere in code (inside functions, etc.).

Checkboxes are a little strange, they typically contain the value "on" if they are checked. However they are not present in the `$_POST` array if they are unchecked, so if you wish to determine if a checkbox has been checked you only need to check for the existence of the corresponding element,

```
if(isset($_POST['checkboxfield']))
```

## MULTIDIMENSIONAL \$\_POST

- Given array-like notation for form field name inputs, PHP will parse the generated HTTP request into sequential and associative arrays

```
<input type="text" name="User[firstname]">
<input type="password" name="User[pwd]">

<input type="text" name="User[opt] []">
<input type="text" name="User[opt] []">

$_POST == Array (
    [User] => Array (
        [firstname] => test
        [pwd] => test
        [opt] => Array ( [0] => abc [1] => def )
    )
)
```

- It can be useful to put every field of a particular form element inside a main key e.g. User so you can extract exactly this data easily

```
$userData = $_POST['User'];
```

A feature unique to PHP a particularly useful way of parsing HTTP POST request bodies.

You can name input fields like PHP arrays and PHP will parse them into \$\_POST as they were an array. So that a field name User[name] is at \$\_POST['User']['name'] .

This feature is special to PHP, so if you send similar forms to other languages or applications you may find different results.

One of the most helpful features of this syntax is that PHP will also parse sequential additions (ie, []) so that if you name several fields the same: x[], x[], x[] – you will get a sequential array called x with elements indexed 0, 1, 2.

Be particularly wary with this syntax since this entails that input fields have the same name. Some other language might take this to mean the latest field overwrites the earlier ones, rather than PHP's concatenative approach.

The screenshot shows the Chrome Developer Tools Network tab. The timeline at the top has markers at 500 ms, 1000 ms, 1500 ms, 2000 ms, 2500 ms, 3000 ms, 3500 ms, 4000 ms, 4500 ms, 5000 ms, 5500 ms, and 6000 ms. A single request is visible, starting around 500 ms and ending around 4500 ms. The request is for "SuperStructure.php".

Name	Headers	Preview	Response	Timing
SuperStructure.php	<b>General</b> Request URL: http://localhost/16-HTTP.ii/SuperStructure.php Request Method: POST Status Code: 200 OK Remote Address: [::1]:80			
	<b>Response Headers</b> view source Connection: close Content-type: text/html; charset=UTF-8 Host: localhost X-Powered-By: PHP/7.0.3RC1			
	<b>Request Headers</b> view source Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8 Accept-Encoding: gzip, deflate Accept-Language: en-US,en;q=0.8,en-GB;q=0.6 Cache-Control: max-age=0 Connection: keep-alive Content-Length: 94 Content-Type: application/x-www-form-urlencoded Host: localhost Origin: http://localhost Referer: http://localhost/16-HTTP.ii/multidim.html Upgrade-Insecure-Requests: 1 User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/49.0.2623.87 Safari/537.36			
	<b>Form Data</b> view source view URL encoded User[name]: test User[pwd]: test User[opt][]: abc User[opt][]: def			

Here is chrome developer tools (F12) showing the request/response for the form on the previous slide.

Show source gives you a more accurate view of what was sent...

Screenshot of the Chrome Network tab showing a POST request to SuperStructure.php. The request URL is http://localhost/16-HTTP.ii/SuperStructure.php, Method is POST, Status Code is 200 OK, and Remote Address is [::1]:80.

- General**
  - Request URL: http://localhost/16-HTTP.ii/SuperStructure.php
  - Request Method: POST
  - Status Code: 200 OK
  - Remote Address: [::1]:80
- Response Headers** (view parsed)
  - HTTP/1.1 200 OK
  - Host: localhost
  - Connection: close
  - X-Powered-By: PHP/7.0.3RC1
  - Content-type: text/html; charset=UTF-8
- Request Headers** (view parsed)
  - POST /16-HTTP.ii/SuperStructure.php HTTP/1.1
  - Host: localhost
  - Connection: keep-alive
  - Content-Length: 94
  - Cache-Control: max-age=0
  - Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,\*/\*;q=0.8
  - Origin: http://localhost
  - Upgrade-Insecure-Requests: 1
  - User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/49.0.2623.87 Safari/537.36
  - Content-Type: application/x-www-form-urlencoded
  - Referer: http://localhost/16-HTTP.ii/multidim.html
  - Accept-Encoding: gzip, deflate
  - Accept-Language: en-US,en;q=0.8,en-GB;q=0.6
- Form Data** (view parsed)
 

```
User%5Bname%5D=test&User%5Bpwd%5D=test&User%5Bopt%5D%5B%5D=abc&User%5Bopt%5D%5B%5D=def&submit=
```

Here is the actual source of the request. The "form data" element shows the request body, it has been urlencoded for transmission by the browser.

This is the HTTP request that PHP processes into the superglobals. Form Data becomes \$\_POST. Many headers go in \$\_SERVER.

## USING FILTERING FUNCTIONS

Prefer `filter_input` over raw `$_GET`, or `$_POST`

- `filter_input_array` to get the entire array
- a policy of never using the superglobals is reasonable

```
$search_html = filter_input(INPUT_GET, 'search',
                           FILTER_SANITIZE_SPECIAL_CHARS);
$search_url = filter_input(INPUT_GET, 'search',
                           FILTER_SANITIZE_ENCODED);
echo "You have searched for $search_html.\n";
echo "<a href='?search=$search_url'>Search again.</a>";
myinputs = filter_input_array(INPUT_POST, $args);
```

PHP includes a `filter_*` function library which offers basic input validation and sanitization.

`filter_input(INPUT_GET ...)` -- means retrieve a value from the `$_GET` array.

`filter_input(INPUT_GET, 'search')` -- specifically `$_GET['search']`

`filter_input(INPUT_GET, 'search', FILTER_SANITIZE_SPECIAL_CHARS);` -- encoding any special characters for output

Each `FILTER_SANITIZE_*` option and other `FILTER_*` options provides different functionality. See the resource links at the end of this chapter.

`filter_input()` functions give you access to `$_POST` and `$_GET` either filtered, validated, sanitized or in raw form.

Therefore there is a strong case never to use `$_GET` or `$_POST` directly.

## FILTERING FUNCTIONS

### Filter Functions

- **filter\_has\_var** — Checks if variable of specified type exists
- **filter\_id** — Returns the filter ID belonging to a named filter
- **filter\_input\_array** — Gets external variables and optionally filters them
- **filter\_input** — Gets a specific external variable by name and optionally filters it
- **filter\_list** — Returns a list of all supported filters
- **filter\_var\_array** — Gets multiple variables and optionally filters them
- **filter\_var** — Filters a variable with a specified filter

A reference slide for the filter functions.

filter\_input\* concern \$\_GET, \$\_POST only.

filter\_var\* allow you to apply the same kinds of filters available for filter\_input to your own data.

## UPLOADING FILES

- Forms also support file uploading
  - Specify a form encoding type
  - And a specialized input type and the browser will take care of the rest
- File uploading with PHP is particularly sensitive to configuration issues
  - The .ini file configuration settings should be reviewed to ensure they are appropriate
- Files are first uploaded to a temporary directory
  - This directory must exist and have the right permissions
  - Files will be moved from this directory by your script and may be placed wherever (e.g. in a database)

HTML forms may also be used to upload files and PHP supports this straightforwardly, however there are some configuration options which can interfere with the process (the important ones follow).

The approach to uploading is always the same: the user selects a file using the browser. The browser then chunks that file up for transmission and sends it to the web server which stores it in a temporary location on disk.

It is then up to your PHP application to decide what to do with it, or even perhaps, delete it.

Since uploads are to a temporary folder (typically /tmp) the drive of the host machine needs to be big enough to support it and you should ensure the temporary folder is routinely cleaned by the operating system (it usually is).

## PHP.INI

- **file\_uploads**
  - Enable/disable PHP support for file uploads
- **max\_input\_time**
  - in seconds, how long a PHP script is allowed to receive input
  - `set_time_limit()` -- change allowed execution time of script
  - set to 0 for no limit
- **post\_max\_size**
  - size of the total allowed POST data
- **upload\_tmp\_dir**
  - directory where uploaded files will be temporarily stored
- **upload\_max\_filesize**
  - size of the largest possible file upload allowed

These are the most important ini configuration variables which affect file uploading, often these being misconfigured causes development headaches.

Many developers set `max_input_time` to 0 to allow the script to continue to run for however long it needs to. This affects more than just file uploading. Sometimes your application will request remote services which can take a long time to respond. For example, if your application is hosted on a cloud provider (e.g. Amazon Web Services) a typical uploading step is to move the temporary file to the cloud provider's storage mechanism e.g. S3 (Simple Storage Service) which can take a long time.

`upload_tmp_dir` could be set to a temporary storage volume or partition so as to isolate anything uploaded and to make it easier to free space if necessary.

If you are expecting to upload very large files you may want to reconsider using HTTP/POST which is not a very reliable uploading mechanism. For example, you could provide your client with an ftp server.



## THE UPLOAD FORM

- PHP requires a MAX\_FILE\_SIZE hidden field,
- This field can be at most `upload\_max\_filesize`

```
<form action="upload.php"
      method="post"
      enctype="multipart/form-data" >

<input type="hidden"
       name="MAX_FILE_SIZE"
       value="10000000" />

<input type="file" name="myfile" />
<input type="submit" value="send" />
</form>
```

• Above it is 10 000 000 or ~10MB



Here is a typical file-upload form. The enctype must be multipart/form-data and the input type should be "file" so that the browser will display the correct "Browse" option.

PHP additionally requires a hidden field, MAX\_FILE\_SIZE which must be no greater than your ini settings. The ini settings provide the hard limit, so that a user who changes the hidden field cannot accomplish anything destructive.



## \$\_FILES



- **\$\_FILES** – each key from name of form input
- If names use sequential array notation, **\$\_FILES** will be sequential
- Each element of files e.g. **\$\_FILES[0]**, contains an associative array

Key	Value
name	original file name, as uploaded
type	MIME type of file, inferred by browser
size	Size, in bytes, of the file
tmp_name	The temporary location of the file on the server
error	any (HTTP, PHP) errors associated with upload

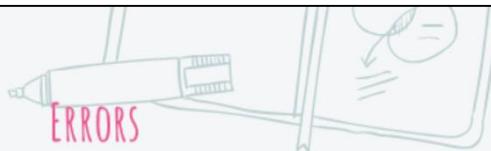
When a file form has been POSTed PHP parses the resulting data into a **\$\_FILES** superglobal. The key of this associative array is the name of the "file"-type input you sent.

Note that this array merely contains the location of the temporary file, it does not contain the file data itself.

It is possible to upload multiple files, that is to have multiple "file"-type inputs. In which case the **\$\_FILES** array has an independent file for each element.



## UPLOADING STEPS:



```
const InputKey = 'myfile';      //<input> name

const AllowedTypes = ['image/jpeg', 'image/jpg'];

if (empty($_FILES[InputKey])) {      //handle error
    die("File Missing!");
}

if ($_FILES[InputKey]['error'] > 0) { //handle error
    die("Handle the error!");
}

if (!in_array($_FILES[InputKey]['type'], AllowedTypes)) {
    die("Handle File Type Not Allowed");
}
```



Let's walk through the steps required to process this upload.

First of all the const InputKey has been defined, this is the name of the "file"-type input field. It did not need to be a constant, but it is here for clarity.

Next an AllowedTypes const array is defined. This is a list of MIME types the the site will accept, any kind of file type not in this list will be rejected.

The first check tests if the \$\_FILES array has an entry for InputKey, that is, there is actually an upload. If it's empty, the script dies.

Then the error key is checked, a value of 0 indicates success, so any number greater than 0 will be an error.

Finally the MIME type of the file upload is compared to the AllowedTypes. Recall that in\_array() is a function of two arguments, it checks to see if the first argument is contained in the second, which must be an array.

In each case die() is called if there is an error. This is for illustrative purposes, in a real site you would for example, display an HTML error page.



## UPLOADING STEPS:



### Options:

- Move the temporary file to a suitable location
- Read it and puts its contents in a database
- Read it and parse it for e.g. configuration
- Generate thumbnails, ... + domain-specific uses

```
$tmpFile = $_FILES[InputKey]['tmp_name'];

//DOMAIN SPECIFIC: e.g. move the file
$dstFile = 'uploads/'.$_FILES[InputKey]['name'];

if (!move_uploaded_file($tmpFile, $dstFile)) {
    die("Handle Error");
}

//Clean up the temp file
if (file_exists($tmpFile)) {
    unlink($tmpFile);
}
```



If we reach this code the file has been uploaded correctly, since none of the error checks were triggered.

The tmp\_name key of our file array provides the location of the temporary file (e.g. /tmp/afileIHaveJustUploaded.jp).

In this example the code just moves the file from the temporary location (where it will likely be automatically deleted) to some local uploads folder. This is only one kind of approach – you could, for example, read the file into a database.

The function move\_uploaded\_file() is the idiomatic PHP function for this, but can only move the file from one area of disk to another – there's nothing fancy. It will ensure that the temporary file you're trying to move really was the one uploaded (you, could for example, accidentally move a different file).

Finally we unlink() (i.e. delete) the file if it still exists.



## UPLOADING MULTIPLE

- Use array notation and multiple file-inputs

```
<form action="Upload.php"
      method="post"
      enctype="multipart/form-data">

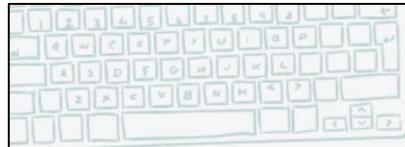
<input name="myfile[]" type="file" />
<input name="myfile[]" type="file" />
<input type="submit" value="Upload" />

</form>
```

- **`$_FILES`** is a sequential array
  - each element corresponds to one file-input

You can combine the HTML-array syntax with multiple file inputs. In this case `$_FILES` will be a sequential array (since `[]` is sequential concatenation).

This example here is for illustrative purposes and because it is the approach PHP developers tend to use.



## FILE FUNCTIONS



Here are some useful file functions often used when managing uploads

- **mkdir()** – Create a directory
- **filesize()** – Get the size of a file
- **copy()** – Copy a file
- **rename()** – Rename a file or folder (i.e. move)
- **filemtime()** – Get file modification time



Often when uploading files you will need to manipulate them in some way, and indeed, perhaps create and remove directories.

Here are a few useful PHP functions that you might typically see in a file uploading script.



# STATE: COOKIES & SESSIONS

The HTTP protocol has its limitations. One of the most important is its inability to maintain state: there is nothing about HTTP which requires a browser or a server to remember anything about the client/server sending the request.

According to the protocol alone, each HTTP request sent to a server is completely ahistorical. Everything a webserver needs to send a response must be contained in the HTTP request the client sends: the client cannot expect the web server to remember who it is.

However that is not our experience of websites. They do remember who we are: you can login and they remember that you have logged in. You can go to a shopping web site and add items to your cart and the cart retains the items you've added to it. And they are *your* items, for your browser session. If you add an item only you see it, not everyone else in the world.

This chapter will discuss how this is achieved and present PHP's tools for managing cookies, sessions and HTTP state in general.

## HTTP AND STATE

- As we have seen HTTP requests and responses are the mechanism by which browsers and servers communicate
- HTTP is a stateless protocol
- The requests / responses are the only form of data transmission
- Each contains all the data relevant to that request/response
- No data, as part of the protocol, is "remembered" by either the server or client
- Cookies are a way around this problem
- Cookies are "remembered" pieces of data the browser stores and sends as headers on each request

HTTP requests are "complete", they must contain all the information a webserver needs. This limits web development: often we would like to refer to a lot of data in a requests, for example, you may want to say "this request relates to my shopping cart that I created several requests ago". However HTTP has no way of referring back to "several requests ago".

We may want to recognize a user from an old session, say, perhaps remember what their username was. Again with HTTP there is no way to say, "remember that POST request I sent 3 days ago".

However there is a way to put enough data into an HTTP request to get around this problem . A HTTP request has two customizable "data" aspects: the headers and the request body. POST data is usually sent in the request body, so that's taken up. What remains are the headers.

The technique then is to get the *browser* to remember the information we would like it to remember (e.g. username) and then send that piece of information as a header with every request. The HTTP protocol is still stateless: the *browser* remembers what it needs to add.

This little header addition is called a cookie.

## COOKIES

- Cookies are small pieces of data, given they are sent on every request
- They are key-value pairs sent in every GET or POST request to the cookie's domain
- They are used for:
  - authentication – remembering a username, email or that a person is logged in
  - identifying and tracking individual users – identifying that a browser session belongs to a particular user
  - maintaining site specific preferences e.g. remembering what colour you like a page background

Cookies are just small bits of data, they don't do anything

They aren't themselves "tracking" anything, or viruses

Usually do not contain sensitive information



Cookies are nothing to be afraid of, they are an essential part of the web. A website has no other way of remembering who you are: no way to log you in, or personalize your experience.

They are also used for invasive tracking and advertising systems that some people have political or ethical objections to, however this is only one use of cookies and not essential to their primary function.

At the end of the day a cookie is just a small piece of data the browser remembers on behalf of a website – that's it.

## TYPES OF COOKIE

- **Session cookies**
  - default kind
  - stored in browser's memory therefore when the browser is closed, this is lost
  - does not track long term individual use, but is session-specific
  - not on the file system so cannot be accessed by anything else (e.g. viruses)
- **Persistent cookies**
  - stored on file system
  - retains information long-term, therefore *can* be used to track users
  - historically less secure and vulnerable to viruses

Quite a lot of browser protection today (e.g. encryption)

There are two types of cookie divided by how the browser remembers: session cookies live in memory and only while the browser window is open. Persistent cookies are stored in files.

Since sessions cookies only live in memory and only for a brief amount of time they cannot be used for user tracking, nor can they be stolen (by viruses etc.) and indeed are the most harmless kind.

By default PHP creates session cookies.



## PHP'S COOKIE FUNCTIONS

**setcookie()**- PHP sends headers to set a cookie

- `setcookie()` and all header functions must be called before any text output (i.e. request body is sent)
- Can be used ~50 times per user
- Stores ~4kB of information
- By default stores a session cookie
- Third parameter sets a persistent cookie

```
setcookie('key', 'value');  
setcookie('user', 'shomes');
```

```
//time() is time since unix epoch in seconds  
$oneWeekFromNow = time() + 60 * 60 * 24 * 7;
```

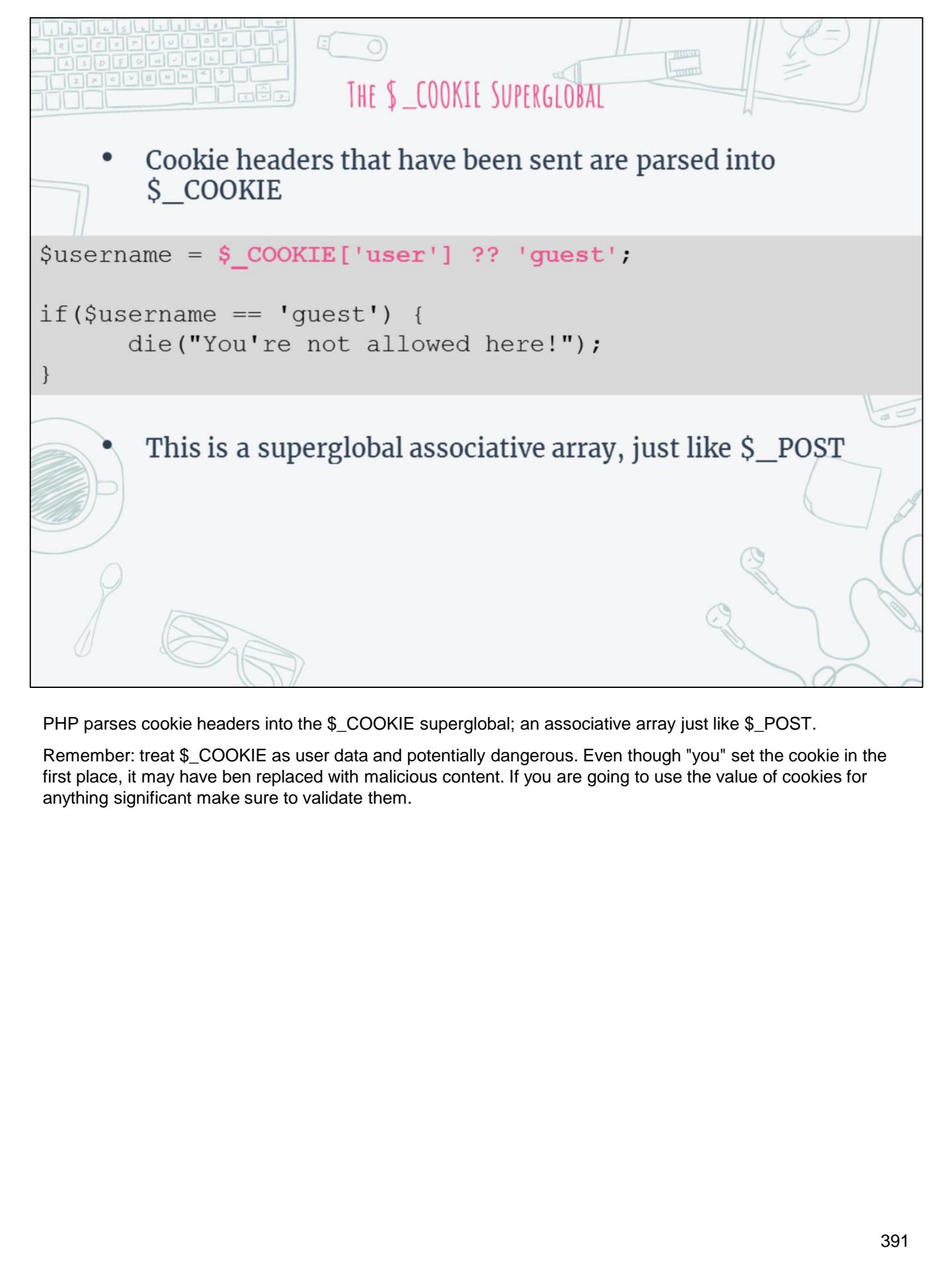
```
setcookie('sales_offer', 'freebeer', $oneWeekFromNow);
```

```
//setcookie() in the past to delete!  
setcookie('sales_offer', '', time() - 1);
```

PHP's `setcookie` functions create cookies. They send an HTTP response header which the browser understands and uses to create a cookie file.

Since `setcookie()` sends a header it must occur before any other echo or output. When your script outputs ordinary text PHP takes that as a sign the headers are finished, and files the response body with whatever output you supply.

Recall that HTTP response bodies have to come after headers and are typically HTML pages.



## THE \$\_COOKIE SUPERGLOBAL

- Cookie headers that have been sent are parsed into `$_COOKIE`

```
$username = $_COOKIE['user'] ?? 'guest';  
  
if($username == 'guest') {  
    die("You're not allowed here!");  
}
```

- This is a superglobal associative array, just like `$_POST`

PHP parses cookie headers into the `$_COOKIE` superglobal; an associative array just like `$_POST`.

Remember: treat `$_COOKIE` as user data and potentially dangerous. Even though "you" set the cookie in the first place, it may have been replaced with malicious content. If you are going to use the value of cookies for anything significant make sure to validate them.

## SESSIONS

- A browser "session" is the opening of a window, browsing to a URL and staying at that URL for some time
  - more abstractly it is a series of HTTP requests and responses
  - that occur between a browser and webserver
  - which represent the user being "continuously active" on that domain
  - HTTP has no notion of a "session". This is something we say of some set of continuous HTTP Requests and Responses
- PHP supports real sessions
  - PHP can identify a particular browser and a particular set of Request/Responses as belonging to a particular user active on the site
  - Sessions typically use cookies as an identifier
  - This cookie connects a browser session to a temporary file which stores data for that particular user

S·E·S·S·I·O·N

Thus far you have seen that the browser can add cookie headers to include extra information that the browser remembers. The most important use of this feature for PHP for its Sessions.

When you logon to a web site you expect it to remember that the browser you are connecting with "belongs to" your session: your profile, your shopping cart, etc.

PHP provides a simple way to achieve this complex result: it can send a cookie, on your behalf, with a unique identifier. The browser then sends this unique id back to PHP on every request. Therefore every browser is given its own unique id – so PHP can identify particular users. And, for example, associate a shopping cart with the right person.

## ESTABLISHING SESSIONS

1. Client connects to the website: an initial HTTP request is sent to the webserver
2. Webserver (via PHP) notes `$_SERVER` data (i.e. IP address, browser, etc.)
3. Server sends back a session id cookie header
4. Client sends this cookie back on every further request
5. Server receives cookie containing session id, uses it to connect user to their data e.g. login information

Here is a high-level view of the steps taken.

You browse to a PHP site. It records all your header/ip/etc. data.

In the HTTP response PHP inserts a cookie header containing a unique id.

The browser stores this unique id.

On every subsequent request the browser inserts that id into a cookie header which PHP reads on your behalf.

## SESSION FUNCTIONS

- **session\_start();**
  - sends the initial cookie, or if cookie has already been sent:
  - retrieves the session data for this user
  - must be called before any output (i.e. request body sent)
  - populates the `$_SESSION` superglobal with data *for that user*
- **session\_destroy();**
  - remove all session data *for this user*

```
if(empty($_SESSION)) {  
    die("please log in");  
}  
$username = $_SESSION['user'] ?? 'guest';  
  
if($username == 'guest') {  
    die("please log in properly!");  
}
```

PHP maintains a special superglobal `$_SESSION`; an associative array which is particular to every browser session.

`session_start()` is the function which organizes the operation: if a cookie needs sent, it will send one. If a cookie has been received it will read it and handle it: i.e. will populate `$_SESSION` with the data corresponding to that cookie's ID,

## CUSTOM SESSION HANDLER

- The session identifier is stored in a cookie called `PHPSESSID`
- On the server the session data is stored in a temporary file
- To change the location of the file use `session_save_path()`
- To change how PHP stores session data, set a callback using `session_set_save_handler()`

The data contained in `$_SESSION` is stored on the server, by default in a temporary file – one per user.

The session identifier is stored in a cookie called `PHPSESSID`. On the server the session data is stored in a temporary file e.g. `/tmp/sess_aksdj21123/`

This location can be found using `session_save_path()`. You can use the same function to change the location.

To change how PHP stores session data, set a callback using `session_set_save_handler()`. Larger applications will likely store session data in a database and cloud applications can rarely use standard file-based sessions since browsers are not necessarily always hitting the same server with the session files stored on it.

File-based sessions are not much of a performance issue however they are regularly cleaned up, and typically not many users are using the site at the same time (even 100s is OK)

## SESSIONS WITHOUT COOKIES

- Sessions do not require cookies
- If PHP cannot set a cookie or they have been disabled it will append a PHPSESSID to the URL
- Potentially insecure so re-authenticate for major actions

Sessions do not require cookies. If your browser does not support cookies, or they have been turned off, PHP will insert a query parameter in the URL of the request (?PHPSESSID) which it will use to store the id of that user's session. All requests will have this appended. Be aware that this is insecure because a user might share the link and this will give another user the same session. To protect against this you could re-authenticate for major actions such as buying products by asking for known information, for example, the last 4 digits of their credit card number.

However since PHP doesn't control all the URLs in your application (you output them in <a> tags for example) you will manually need to add the query parameter sometimes.

This approach is incredibly insecure and very cumbersome. It is generally just better to reject users whose browsers do not support cookies (at least from your login process).



```
session_start();  
  
function login($username, $password) {  
    if($username == 'notguest'  
    && $password == 'testpwd') {  
        $_SESSION['User'] = [  
            'id' => 1,  
            'access_level' => 'admin'  
        ];  
    }  
}  
function logout() {  
    session_destroy();  
}  
function check_access($level) {  
    return empty($_SESSION) ? false :  
        $_SESSION['User']['access_level'] == $level;  
}
```



Here is a schematic for a simple authentication library.

`session_start()` must always be called to ensure `$_SESSION` is populated properly, and if not, to issue a cookie to a browser so that it can be.

The `login` function checks a username and password and if they are correct adds some information to the user's `$_SESSION`, i.e., their id and access level.

`logout()` merely calls `session_destroy()` which deletes all their data.

`check_access()` checks what access level has been stored in their session. Recall that all this data is on the server machine, so there is no realistic chance of tampering: we have set the access level in `login()`.

## HTTP FUNCTIONS

- **header()**
  - send a particular header
- **exit(), die() – end session**
  - always exit() when you redirect to prevent anything else in the script running

```
function auth_failure() {  
    header("Location: /login.php");  
    exit();  
}
```

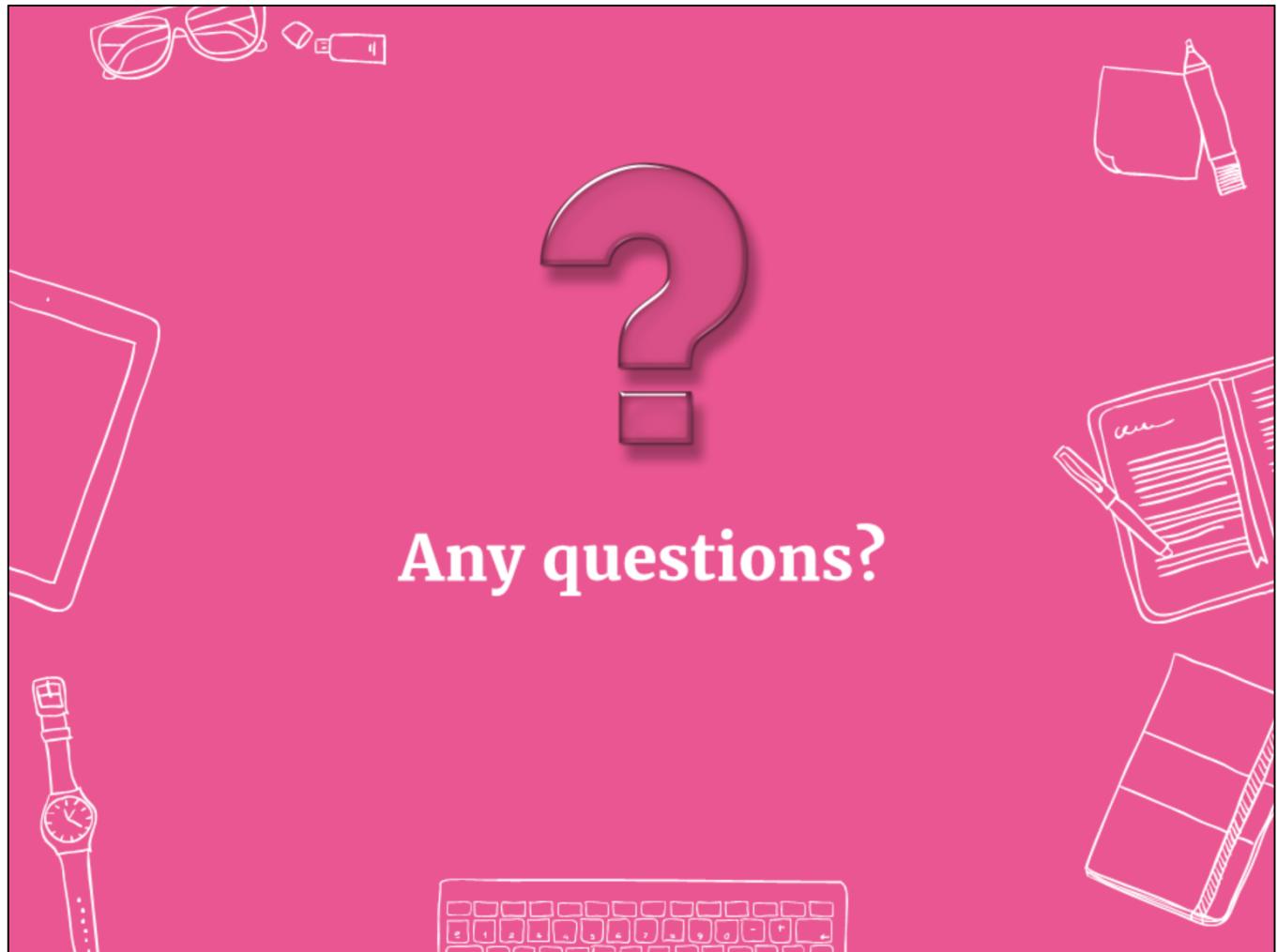
- exit() is more often used with an intentional exit condition, die() when debugging

Some functions you may see used with authentication systems:

header() sends custom headers, "Location" being the most common which is a redirection header.

However header() does not terminate the script, so it may continue running even if you think you have redirected the user (a potential security issue) – therefore always exit() after issuing a redirection header.

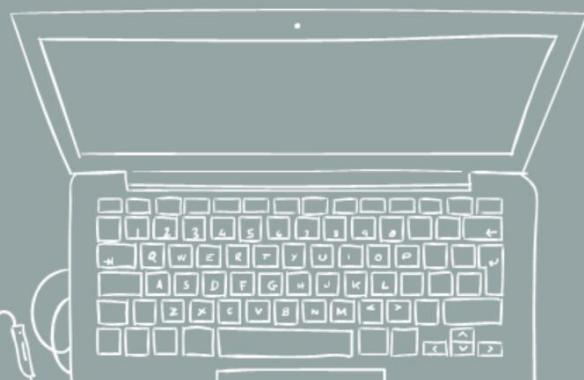
And indeed always exit() if you expect the script to stop at that point.



**Any questions?**



## EXERCISE 17



## RESOURCES

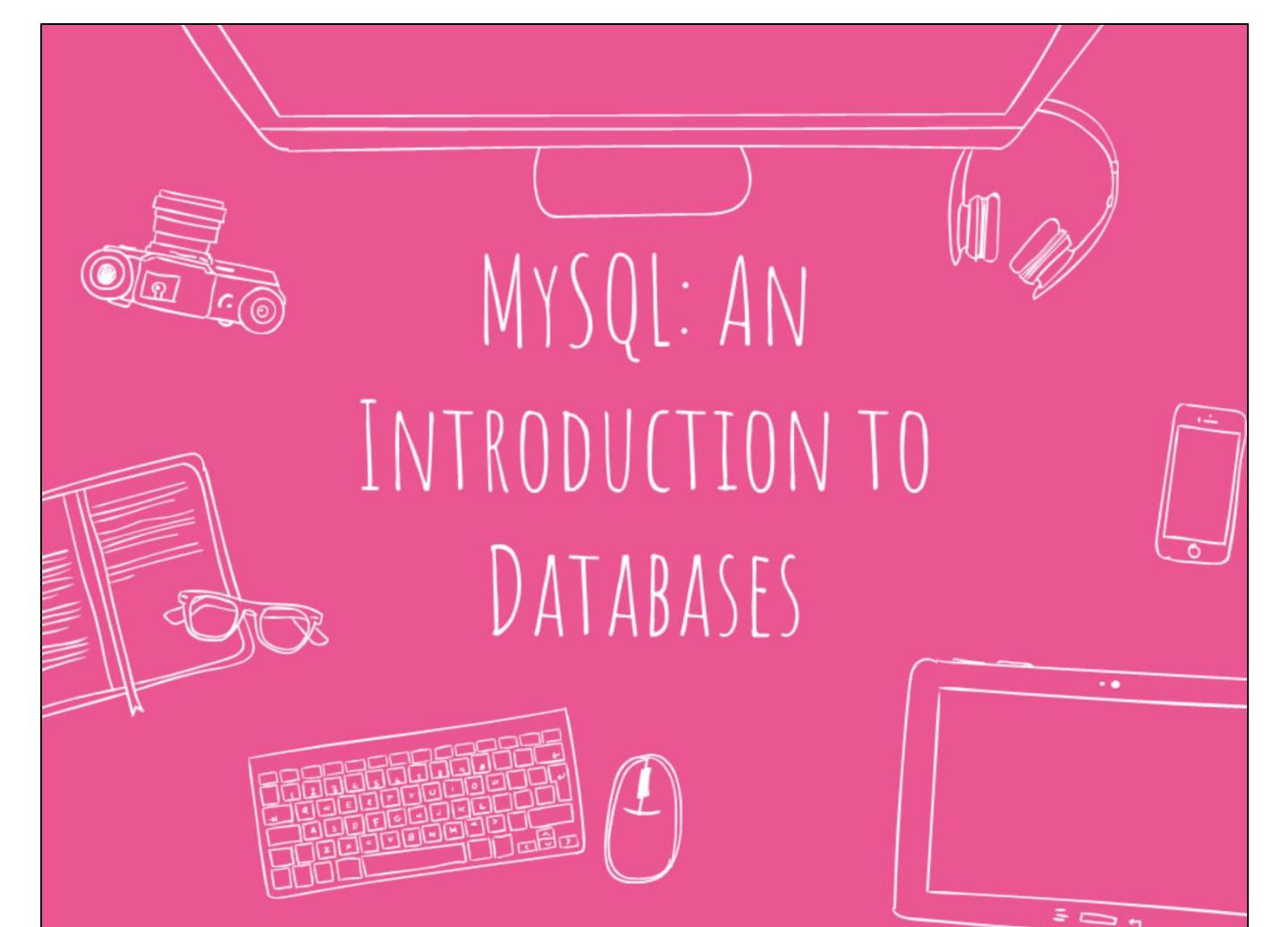
Session Configuration

<http://php.net/manual/en/session.configuration.php>

Custom Session Handling

<http://php.net/manual/en/memcached.sessions.php>

<http://php.net/manual/en/function.session-set-save-handler.php>



# MySQL: AN INTRODUCTION TO DATABASES

In previous chapters you have seen how to store data in arrays and files. Dedicated databases are a much more efficient and useable form of data storage for large quantities of data.. Databases store their data in tables. These tables can have defined relationships between them: for example the users table stores a reference to the profile table. The users table contains the basic data for a user and the profiles table contains the data for a user's profile.

Relational databases have a language of their own called SQL (structured query language) which can be used to specify how data should be retrieved and managed.

This chapter will introduce key database concepts.

## CHAPTER OVERVIEW

- What are databases?
- Types of database
- Relational Databases
- NoSQL Databases
- DBMS
- MySQL
- Introduction to Relational Databases

### Exercise Database Design

By the end of this chapter you should know what a database is and, in broad terms, how it works.

On this course you will focus on the database most frequently used with PHP: MySQL.

## WHAT ARE DATABASES?

"A database is a collection of data, typically describing the activities of one or more related organizations"

— Ramakrishnan & Gehrike, Database Management Systems, 3<sup>rd</sup> Ed, 2003

- A collection of objects and the relationships that link them together
  - They can store a wide range of information
  - Mostly text / numbers
  - Images
  - Sound
  - Video
- Databases come in different forms
  - Flat-files store all the data in a single file
  - Networks of linked objects and their relationships
  - Hierarchical databases are a tree structure

We use databases comfortably every day, from our telephone directories (either the paper versions or the electronic versions on the phone), choosing music from our playlists, looking up flights, searching through Wikipedia or even searching for a website on Google. Everything which stores data has a database.

There are two main types of databases popular today: "relational" and so-called NoSQL databases, which are mostly "document-oriented". You could contrast the two approaches by comparing single dimensional and multidimensional arrays in PHP. Relational databases take a single-dimensional approach and if two sets of data are related then a key in one array matches up to a key in another array (e.g. an articles table has an author field which is an id found in the users table). Document-oriented databases nest data so that retrieving one record gets you all the data in one go.

There are trade-offs for each choice, but relational databases tend to be more popular because they give you a greater degree of control over which data you wish to include in your results.

## A BRIEF HISTORY OF DATABASES

- As more information was being stored there was a need to organise it somehow
  - Need some way of searching for information
  - Need to answer questions
- They have existed in some form since the early 1960s
  - Started with a network model
  - Relational model proposed in the 1970s representing both objects and the relationships between them
- SQL – Structured Query Language
  - Used to communicate with the database
  - Formalised in the late 1980s
  - Various revisions (1986, 1989, 1992, 1999, 2003, 2006, 2008, 2011)

Database technology is very old and the standard language used to query databases – Structured Query Language (SQL), was invented in the 1970s.

## WHAT ARE DATABASES USED FOR?

- Storing information in a way that we can use
  - Query information to answer questions
- Databases are used in most programs, webpages and applications
  - Track current stock levels
  - Login details
  - High score table in a game
  - Music collection
  - Financial reports
- A good database is able to
  - Store information efficiently
  - Provide fast and easy access
  - Keep data correct and safe (integrity and security)

In web development there are recurring sets of data you are likely to store. Very often there will be a users table which provides user accounts for the users of your site. This will typically store at least a username, password, and email address.

Various kinds of "groups" or "categories" tables recur which provide tags for your data. A blog post might be tagged with the Politics category, or a user might be tagged with the Admin group.

## RELATIONAL DATABASES

- Relational databases store information in **tables**
  - The links between the tables are known as relations
  - Tables are formed of columns and rows, sometimes known as fields and records
  - Each field or column has a name and a data type
  - The types of data you can use depend upon the database server you are using
- Within each table one or more composite columns are designated as the **primary key**
  - The data in this column(s) must be unique within the table
- A table may contain **foreign keys** which cross-reference the record to a primary key in another table
- The **schema** of a database describes its tables including their column names and data types

To organise the relationships between data, relational databases require you to designate one key as the *Primary Key*. This field of your table (viz. key of your array) is unique for every entry in your database. This key can then be used to associate one particular entry in a table with a particular entry in another.

There are other kinds of keys too. An article table might store a `user_id` for its author, and a `category_id` for its category. These are known as foreign keys. They typically relate to keys in "foreign" tables so that, in this case, a particular article is associated with a particular user and a particular category.

## NOSQL DATABASES ("NOT ONLY SQL")

- Does not use table based data
  - Used for storing (persisting) objects
  - Objects can be anything
- Different methods of storing and accessing information
  - Column – Uses timestamps to differentiate data (Cassandra)
  - Document – Uses XML / JSON to store information (MongoDB)
  - Key/Value – Associative arrays (CouchDB)
  - Graph – Based on graph theory, many links between objects
  - Multi-modal – Mixtures of the above and relational model
- Support query languages
  - Often similar to SQL



NoSQL databases come in a variety of forms and are artificially grouped by the choice **not** to use SQL, rather than a single design theme which runs between them.

This course does not go further into NoSQL databases.

## DATABASE MANAGEMENT SYSTEMS

- The DBMS is the engine that controls the database
  - You do not interact directly with data, but instead pass queries through the DBMS
- Many different DBMS available
  - Oracle
  - Sybase (Part of SAP)
  - DB2
  - Microsoft SQL Server
  - PostgreSQL
  - MySQL
  - MariaDB



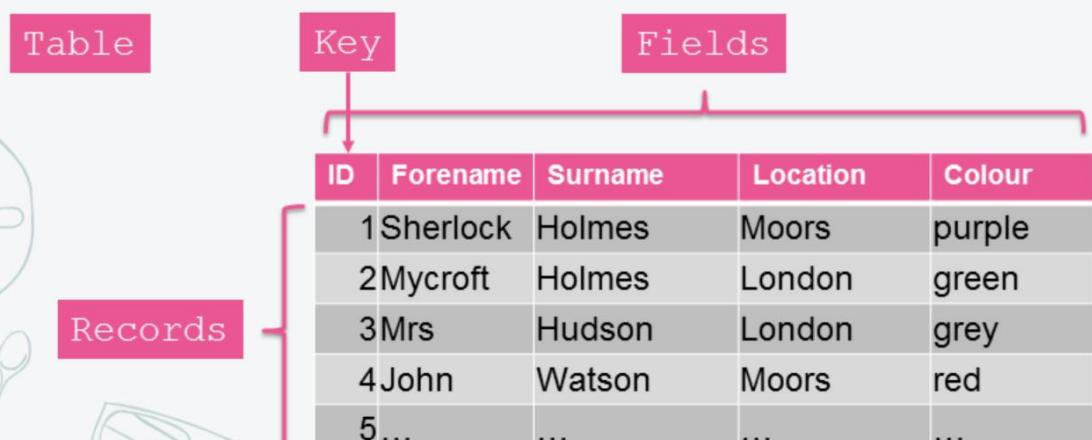
If the interpreter is the program which deals with our PHP code, then the DBMS engine or database management system engine is the program(s) which deal with our SQL code (and data).

MySQL is one of the most popular DBMSs in the world and developed alongside PHP as a first-class citizen in the PHP world. PHP's MySQL libraries have always been well supported and the easiest to use of any PHP provides.

## INTRO TO RELATIONAL DATABASES

Information is stored in databases in tables

- Each table should be about one 'thing'
- Each row in a table is known as a record
- Each column has a heading known as the field
- Each row must have a unique identifier known as the key



Conceptually a table is only an association of keys and values, or in database terms, fields and rows.

The salient question is however, what are tables used for? And that's not straightforward. They have lots of uses and some of them quite intricate.

In the simplest case you use a table to store information about "one kind of thing". You do not have a table about the car and the driver which stores car\_speed and person\_height, but rather two independent tables one about drivers and one about cars.

What counts as "one thing" is an open problem in philosophy and not going to be solved by a web developer on a Friday afternoon, the question of how related the data has to be is one which develops from experience (and essentially, the experience of getting it wrong often enough to know what works well).

# RELATIONSHIPS BETWEEN DATA

- **One to One**
    - For each ID in the first table, there is exactly one record in the second table
    - Often used when a table has many different fields and we want to segment data
  - **One to Many**
    - A one way relationship, there is one record in the first table connecting to many in the second
    - For example:
      - *One trainer will have one home location but...*
      - *One home location will have many trainers*
  - **Many to Many**
    - For each record in the first table there are many records in the second table and vice versa
    - For example:
      - *A delegate can attend many events*
      - *An event can have many delegates*
- 
- The diagram illustrates three types of relationships between data tables:
- one-to-one:** Represented by two squares connected by a single vertical line.
  - one-to-many:** Represented by a single square at the top connected by multiple lines to four squares below it.
  - many-to-many:** Represented by four squares on the left connected by multiple lines to four squares on the right.

Given that each table is restricted to being about one thing, how do we relate these one things? A car, after all, has many passengers. A single row in the car table (a car) will correspond to several rows in the person table (passengers).

This relationship is achieved by relying on primary keys to associate the data. In the case above there would be an intermediate table "passengers" with four entries, the id of the car paired with the id of each passenger.

# TERMINOLOGY

- **DBMS – Database Management System**
  - The engine that we interact with that stores the tables, queries and procedures. Processes SQL statements and responds with the requested information
- **Relation / Table**
  - Information stored about a specific ‘thing’ or entity
- **Record / Row**
  - A row in the table, a single instance of that ‘thing’
- **Field / Column**
  - The column headings, specific attributes about each record
- **Key**
  - Primary, Foreign and Compound
  - Unique identifiers for tables and ways to relate data together
- **SQL**
  - Structured Query Language

This slide is mainly for reference, make notes on it and add your own terms as we go through the course

## INTRODUCING MYSQL

- MySQL a popular open source relational database management system
  - Currently owned by Oracle
  - Available for Linux, AIX, BSD, Mac OS X, Windows, Solaris, others ...
    - Free version released under GPL2 licence
- Enterprise versions offer additional features
  - Backup
  - Replication and High Availability
  - Scalability
  - Auditing
  - Technical Support



## THE MYSQL COMMAND LINE (MONITOR)

- Type “mysql” on the command line
  - You may need to link the file to your path
- The MySQL monitor provides a command-line interface to MySQL
  - SQL statements can be evaluated

```
mysql> show databases;
+-----+
| Database      |
+-----+
| information_schema |
| mysql          |
| test           |
+-----+
2 rows in set (0.00 sec)

mysql>
```

When MySQL has been installed the MySQL program (mysql.exe on windows) becomes available.

If you run this program you are greeted with a prompt at which you can type any valid SQL and any SQL commands that are specific to MySQL.

In this case the "show database;" command lists the available databases.

You will need to navigate to the folder where MySQL is installed. On Windows this is under the XAMPP install folder e.g. C:\XAMPP\mysql\bin

NOTE: Don't forget the semicolon at the end of the line of code. You can type 'quit' to exit the MySQL monitor and return to the command prompt.

## SWITCHING DATABASE

```
mysql> use mysql;  
Reading table information for completion of table and column  
names  
You can turn off this feature to get a quicker startup with -A  
Database changed  
mysql>
```



To select a database from this list write "use dbname;" where dbname is the database you wish to run queries on.

Newer version of MySQL run using an engine called MariaDB. Your command line might look like the below:

```
MariaDB [(none)]> use mysql;
```

```
Database changed
```

```
MariaDB [mysql]>
```

NOTE: The mysql database is the system database. It contains tables that store information required by the MySQL server as it runs.



## SHOWING TABLES



```
mysql> show tables;
+-----+
| Tables_in_mysql |
+-----+
| columns_priv    |
| db              |
| event           |
| func            |
|
...
| user            |
+-----+
34 rows in set (0.00 sec)
```

```
mysql>
```

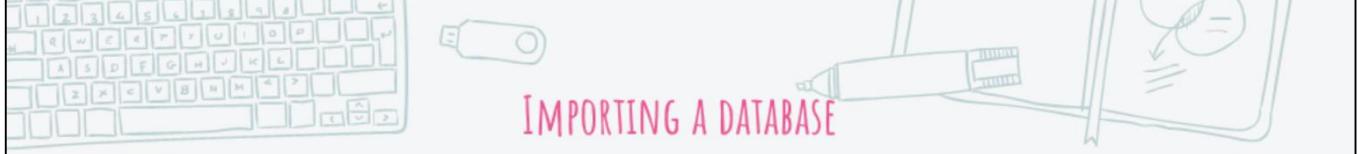


Now that we have selected a particular database we can ask to see which tables have been defined in it. There are 34 tables in the mysql database.

## WHAT IS SQL?

- Structured Query Language
  - It allows us to communicate with a database
  - Statements are interpreted by the DBMS and changes made to the underlying tables and structure
- SQL can be used for many tasks
  - Creating a database
  - Updating and inserting new records
  - Querying a database
- Database backup is done by translating the tables and contents into a set of SQL statements that can be executed to recreate the system

The language we have just been using to speak to the database is called SQL. It allows us to create, update, read and delete data and data structures (tables). It can also administrate a database, creating user accounts which may access only particular sets of data.



## IMPORTING A DATABASE

- Step 1: Create an SQL file

```
-- create a database
CREATE DATABASE QATRAINING;
use QATraining;

-- TABLES
CREATE TABLE clients(
    ClientID int not null primary key AUTO_INCREMENT,
    CompanyName varchar(50) not null
);
...etc
```

- Step 2: import the database via the command line
  - Different GUI based programs will have different methods for importing the file

```
# mysql -uroot < file.sql
```

Typing SQL at the MySQL prompt is fine during development, and for small queries. However for larger amounts of SQL you will want to keep your queries in an independent file.

The MySQL program will accept SQL files, along with any required usernames and passwords and execute the steps one by one in the file.

The following instructions are for the Windows OS:

1. Create a Notepad file and add the SQL statements from the slide.
2. Save the file with a .sql file extension
3. Run the file in MySQL with the following command: C:\xampp\mysql\bin>**mysql -u root < C:\MySQL\testsq1.sql**
4. To verify the database and table exist connect to MySQL and use **show databases** and **show tables** respectively.

## EXPORTING DATABASES FOR BACKUP

- To export a database use the mysqldump program
  - Installed with MySQL as standard

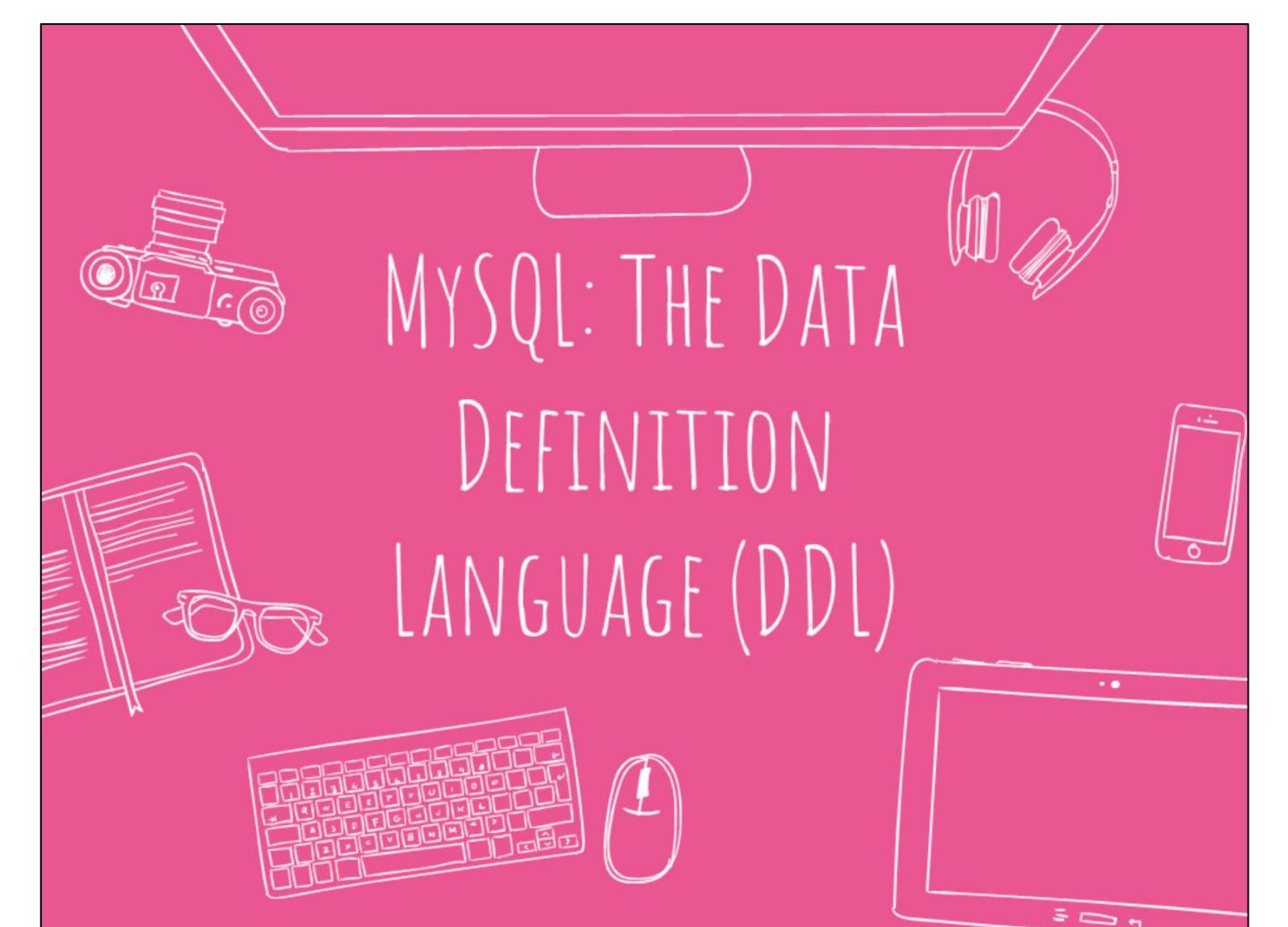
```
# mysqldump -uroot qatraining > file.sql
```

The mysqldump program is a handy extra tool included in the MySQL installation, it can output all the SQL that would be required to recreate a database. This then serves as a handy backup for MySQL data. You can take this file and give it to the MySQL program and it will recreate the database for you.

Export a database as follows:

```
C:\xampp\mysql\bin>mysqldump -u root qatraining > C:\MySQL\ExportedDB.sql
```

NOTE: The < imports and executes a SQL script and the > exports a database.



# MySQL: THE DATA DEFINITION LANGUAGE (DDL)

This chapter introduces more SQL commands, and SQL is the most important element when learning about databases.

CREATE statements are the first set of SQL that will be covered. These provide the means to create new databases and new tables within databases. You will then look at INSERT statements which provide the mechanism of getting data into a database.

Finally you will look at administering databases, in particular, user accounts and privileges.

## CHAPTER OVERVIEW

- Creating Databases
- CREATE TABLE syntax
  - Types
  - Creating keys
  - Auto incrementing
  - Not null
- Users and Privileges

The goal of this chapter is to get you familiar with CREATE and INSERT statements in particular, and to understand how SQL can be used to manage and administrate a database.

# CREATING AND DELETING DATABASES USING MYSQL

- Create a database

Without this, attempting to create a database that already exists would cause an error

```
create database if not exists store;
```

- Delete a database

- Be careful!

Without this, attempting to delete a database that does not exist would cause an error

```
drop database if exists store;
```

All SQL statements end with a semi-colon. SQL keywords are optionally in upper or lower case and both styles are common today.

The CREATE DATABASE command is followed by one argument, the name of the database you wish to create. Optionally an IF NOT EXISTS clause can be added to the command to prevent you recreating an existing database.

The DROP DATABASE command removes databases and is followed by the name of the database you're dropping. Here you should add an IF EXISTS clause to prevent dropping databases that do not exist in the system (which will cause an error).



YOUR DESIGN



- Table Notation

```
tablename ( Primary Key,  
           Foreign Key,  
           field1, field2, field3  
);
```

- Database Design

```
user( id, username, password );
```

```
item(id, name, price);
```

```
cart(user_id, item_id, amount);
```

In SQL, tables are described by a name followed by parentheses (brackets).

Within these parentheses there will be a comma separated list of fields, each having their own name.

Typically the first field in this list will be the primary key, some conventions follow this with foreign keys (keys which refer to data in other tables), other conventions place these last.

## CREATING AND DELETING TABLES

### Creating tables

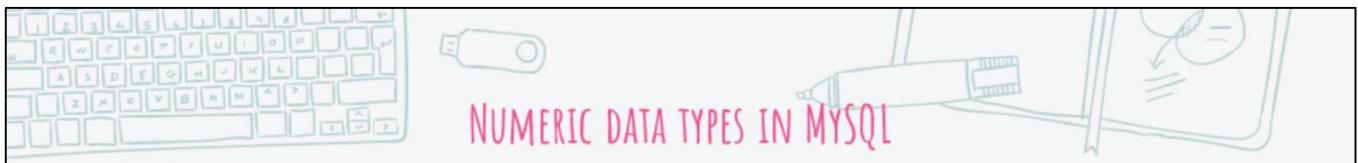
```
use cart;           ← Create the table in this database  
  
create table user  
( id int not null primary key auto_increment,  
  
    username varchar(100) not null,  
    password varchar(100) not null,  
) ;  
  
drop table user;
```

Column name      Column type

Data types will follow column names and whether that column is allowed to be empty (or NULL). NOT NULL indicates that it required.

MySQL supports AUTO\_INCREMENT column values. This feature is most commonly used for the primary key of the table where the requirement is simply in having a set of unique values.

If a null value is inserted into an AUTO\_INCREMENT column, MySQL will automatically generate the next available sequence number (an integer) and insert that into the column.



## NUMERIC DATA TYPES IN MYSQL

Data type	Description
TINYINT(size)	-128 to 127 normal. 0 to 255 UNSIGNED*. The maximum number of digits may be specified in parenthesis
SMALLINT(size)	-32768 to 32767 normal. 0 to 65535 UNSIGNED*. The maximum number of digits may be specified in parenthesis
MEDIUMINT(size)	-8388608 to 8388607 normal. 0 to 16777215 UNSIGNED*. The maximum number of digits may be specified in parenthesis
INT(size)	-2147483648 to 2147483647 normal. 0 to 4294967295 UNSIGNED*. The maximum number of digits may be specified in parenthesis
BIGINT(size)	-9223372036854775808 to 9223372036854775807 normal. 0 to 18446744073709551615 UNSIGNED*. The maximum number of digits may be specified in parenthesis
FLOAT(size,d)	A small number with a floating decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter
DOUBLE(size,d)	A large number with a floating decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter
DECIMAL(size,d)	A DOUBLE stored as a string , allowing for a fixed decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter

Here are the typical numeric datatypes used with MySQL.

## TEXT DATA TYPES IN MYSQL

Data type	Description
CHAR(size)	Holds a fixed length string (can contain letters, numbers, and special characters). The fixed size is specified in parenthesis. Can store up to 255 characters
VARCHAR(size)	Holds a variable length string (can contain letters, numbers, and special characters). The maximum size is specified in parenthesis. Can store up to 255 characters. <b>Note:</b> If you put a greater value than 255 it will be converted to a TEXT type
TINYTEXT	Holds a string with a maximum length of 255 characters
TEXT	Holds a string with a maximum length of 65,535 characters
BLOB	For BLOBS (Binary Large OBjects). Holds up to 65,535 bytes of data
MEDIUMTEXT	Holds a string with a maximum length of 16,777,215 characters
MEDIUMBLOB	For BLOBS (Binary Large OBjects). Holds up to 16,777,215 bytes of data
LONGTEXT	Holds a string with a maximum length of 4,294,967,295 characters
LONGBLOB	For BLOBS (Binary Large OBjects). Holds up to 4,294,967,295 bytes of data
ENUM(x,y,z,etc.)	Let you enter a list of possible values. You can list up to 65535 values in an ENUM list. If a value is inserted that is not in the list, a blank value will be inserted.  <b>Note:</b> The values are sorted in the order you enter them.  You enter the possible values in this format: ENUM('X','Y','Z')
SET	Similar to ENUM except that SET may contain up to 64 list items and can store more than one choice

Here are the typical textual datatypes.

For web development you will mostly be concerned with VARCHAR() for smaller text, TEXT for large pages of text and BLOB for storing files.

# TIME AND DATE DATA TYPES IN MYSQL

Data type	Description
DATE()	A date. Format: YYYY-MM-DD <b>Note:</b> The supported range is from '1000-01-01' to '9999-12-31'
DATETIME()	*A date and time combination. Format: YYYY-MM-DD HH:MM:SS <b>Note:</b> The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'
TIMESTAMP()	*A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD HH:MM:SS <b>Note:</b> The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC
TIME()	A time. Format: HH:MM:SS <b>Note:</b> The supported range is from '-838:59:59' to '838:59:59'
YEAR()	A year in two-digit or four-digit format. <b>Note:</b> Values allowed in four-digit format: 1901 to 2155. Values allowed in two-digit format: 70 to 69, representing years from 1970 to 2069

MySQL is fairly relaxed about the formats it will accept for a literal date or time. For example all of the following represent the last day of the year 2012:

'2012-12-31'  
'2012/12/31'  
20121231 (note no quotes in this one)

A "nonsensical" date such as '2012-31-12' will be entered as 0000-00-00

Sometimes programmers store unix timestamps in integer fields to make calculations in their programs easier, however the database is capable of making datetime adjustments itself.

## DEFINING A PRIMARY KEY

- To identify a field as a primary key add the **primary key** attribute after the field type
  - `id int primary key`
- This will enforce the fact that each entry must be unique
- To create a composite (multi-column key) define a key after the column list

```
mysql>
```

```
create table MyTable (
    id1 int,
    id2 int,
    primary key (id1, id2)
);
```

A primary key is a column which for every row is unique. So, for example, in a users table the username could be a primary key which would require there to be no duplicate usernames.

Typically primary keys are integer ids.

## DEFINING A FOREIGN KEY

Foreign key constraints can be defined when the table is created

```
create table cart
(
    user_id int,
    item_id int,
    foreign key (item_id) references items(id),
foreign key (user_id) references users(id)
) engine = innodb;
```

Must specify innodb. The default storage back-end ("myisam") does not understand foreign keys

Parent table

Primary key in parent table

Foreign key constraints tell MySQL that the data in one row in one table ought to refer to data in another row in another table. They check the integrity of your references. They ensure the ids you've entered are correct.

Foreign key constraints are not required, they only check that the data you've inserted is correct. Without a foreign key constraint you may still insert ids, but they are not checked to exist.

## STORAGE BACK-ENDS

- MySQL supports several "back-ends" for storage, including

Backend	Description
MyISAM	The default storage engine. Fast and lightweight but does not support transactions or foreign keys
InnoDB	A modern storage engine that supports transactions and foreign keys
IBMDB2I	Intended for interoperability with native IBM DB2 databases
MEMORY	In-memory storage – very fast but will not survive a restart of the server
CSV	Uses simple comma separated value files
BLACKHOLE	The mysql equivalent of /dev/null

- The back-end type is specified when the table is created:

```
create table foobar
(
  ...
) engine = innodb;
```

In versions of MySQL prior to 5.5, MyISAM was the default table type. From MySQL 5.5 onwards, InnoDB is the default table type.

MySQL has lots of ways it can store data which are known as engines. The standard engine today InnoDB supports transactions, which the previous standard (MyISAM) did not.

## AUTO\_INCREMENT, NOT NULL AND OTHER CONSTRAINTS

- Fields can be constrained so only certain items can be stored
- auto\_increment
  - Allows for the field to be automatically calculated based on the previous record. This is frequently used for primary keys
- not null
  - When adding a record to the table this field must be provided
- default
  - The default value to be stored in the table if no value is provided
- check [salary < 50000]
  - Allows us to place a constraint directly on the data being stored

Here are some constraints you can use on your fields for reference.

In field definitions within table create statements, you can add in additional constraints. You have seen "NOT NULL" to mean required. You can also specify a default value, for example: username NOT NULL DEFAULT "GUEST".

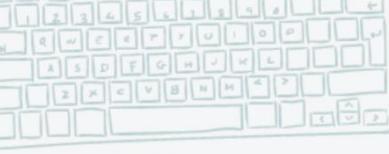
## REFERENTIAL INTEGRITY

- Referential integrity means that every foreign key in a table has a matching primary key in the table it refers to

```
mysql> delete from cart where item_id = 102;
```

- The InnoDB storage back-end is able to enforce referential integrity:

```
mysql> delete from cart where item_id = 102;  
ERROR 1451 (23000): Cannot delete or update a parent row: a  
foreign key constraint fails
```



## INSERTING DATA

- The Insert SQL statement adds new information into the database

```
insert into [table] values ([value1], [value2], ... );  
insert into item values(1, "Camera", 1700);
```

- You can specify which fields to insert
  - Any column not named will be filled with nulls

```
insert into [table](column1, column2)  
values ([value1], [value2], ... );
```

```
insert into user(username, email) values ('sholmes',  
'sholmes@example.com');
```

Now that you know how to create tables, here is how to insert data into them.

The command begins **INSERT INTO** and is followed by a table name, a comma separated list of field **VALUES** and then the particular values you wish to insert.

Strings must be quoted, floats have their decimal points and so on.

## INSERTING MORE THAN ONE ROW AT ONCE

- You can add more than one row at a time
  - Keep adding the values as comma separated tuples

```
insert into items(name, price)
values ('Sony A6000', 500),
       ('Sony A7S', 1500),
       ('Sony A7RII', 2000);
```

- This is often the format generated by the sqldump program
- This format typically does not work with PHP libraries
- In your PHP code you will need to issue multiple independent INSERTs

After the VALUES clause you can append as many parenthetical comma separated lists as you wish.

Here the INSERT statement inserts the data for three camera types into the database in one go.

## COMMON ERRORS

- **ERROR 1062 (23000): Duplicate entry '1' for key 'PRIMARY'**
  - You have tried to add a record with the same key as another record in the table already. Primary keys need to be unique
- **ERROR 1364 (HY000): Field 'id' doesn't have a default value**
  - You have tried to add a record but not specified all the columns and the record can not be created as there is no default value for the non-null column stated.
  - You can also get this if you use default for a field that does not have a default value available
- **ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint fails**
  - The foreign key used in the record does not exist in the other table. The database is trying to preserve referential integrity
  - This will only work in InnoDB tables!

MySQL's error statements are often quite descriptive and accurate. Here are some of the most common.

In the first case you have, for example, inserted two users each having the username – assuming username is the primary key.

The second error implies you have sent no data or a NULL where data should have been. Perhaps the user entered nothing into a form and you failed to check it, and have sent nothing to the database. If that field is NOT NULL, then you need to supply a default value in your application or with a DEFAULT clause.

The final error indicates a foreign key constraint hasn't been met. You have for example, tried to insert an article for a blog with an author's id that doesn't exist in the author's table – presumably in this case, some programming error.

## WALKTHROUGH: CREATING A DATABASE AND TABLES

```
CREATE DATABASE dbname;  
  
USE dbname;  
  
CREATE TABLE tblname (  
    id int primary key auto_increment,  
    fieldTwo text  
);  
  
INSERT INTO tblname(fieldTwo) VALUES ("sometext");
```

- If you omit the id field one will be automatically generated
- (if auto\_increment is used)
  - SQL commands may be upper or lower case

In one go, here are the sequence of commands to move from creating a fresh database to inserting a single piece of data in it.

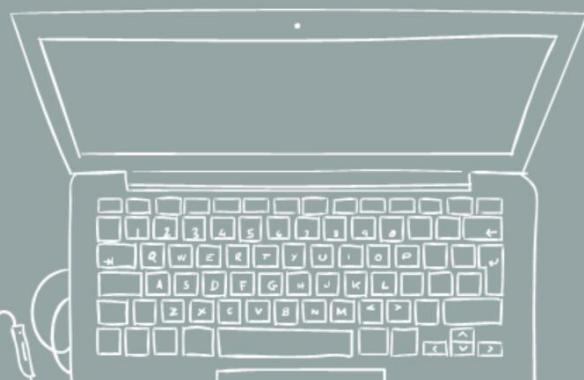
Note that the id field has not been specified in the insert statement. It is omitted from the field list and the values list. This will insert null, triggering the auto-increment behaviour where the database will assign an id to your row.



# Any questions?



## EXERCISE 18



# MySQL PART 2: DCL & DML

# DCL: DATA CONTROL LANGUAGE

## CHAPTER OVERVIEW

### Data Control Language (DCL)

- SQL Privileges
- Creating users
- Granting privileges
- Revoking privileges

## MySQL Privileges

- So far we have logged in to MySQL as `root`
  - This account has far more privilege than our web application really needs
  - No damage limitation in the event of a SQL injection attack, for example
- Best practice is to create additional accounts that have only the privileges required for the task at hand
  - Some pages of our application require read-only access
  - Some pages require read-write access
- MySQL allows assignment of user privileges at four levels
  - Global
  - Database
  - Table
  - Column

We could get into some very fine-grained access control here. Think about the web pages we have created. What tables do each of these pages need to access? Do they need read-only access, or do they need to modify the tables; and if so, which tables?

For example, in a shopping cart application the `product.php` page only needs access to the `items` table and the `login.php` page to the `users` table. You could, if you wanted to, create users with different permissions for each page – though this isn't very common.

We will keep things fairly simple by defining two users – one with read-only access to our database, and one with read-write access.

## USER PRIVILEGES

- User privileges determine whether a user is allowed to run specific SQL commands. Privileges include:

Privilege	Applies to	Allows users to:
SELECT	Tables, columns	Select rows from a table
INSERT	Tables, columns	Insert new rows into tables
UPDATE	Tables, columns	Update values in existing rows
DELETE	Tables	Delete existing table rows
CREATE	Databases, tables	Create new databases or tables
DROP	Databases, tables	Delete databases or tables
USAGE	Global	Do nothing
ALL	Global	Do everything. Usually applied only to the "root" account

The owner / creator of a table automatically has all the privileges

What we have not shown here are the "administrator" privileges. These give you finer-grained control over whether (for example) an account can create temporary tables or shut down the server. We are assuming that there's a root account that can do all of these things; our concern here is with the regular accounts that only have sufficient privilege to carry out the database operations required by our web application. Column-based privileges are useful if you want to hide a specific part of the table.

In large organizations some users may be restricted from seeing all the real, live data. For example a salary field might be hidden in an employees table. Some developers would typically have administrative privileges since they are required for making revisions to the database.

## CREATING USERS

The `create user` command creates user accounts

- User names and passwords need not be related to Linux accounts
- Passwords are hashed before storage in the database

User name

Machine name

Use % as a wildcard

Use `localhost` for loopback connection

```
create user fred@example.com identified by pa$$
```

Password

The username is given *per host* so your account is tied to the machine's address you are using. Use an asterisk (\*) to allow a connection from that user on any host.

Often the host will be `localhost`, meaning non-local connections will be rejected.

## GRANTING PRIVILEGES

The `grant` command assigns MySQL privileges

The list of privileges to be assigned  
Example: select, insert, update

The database and table to apply  
the privilege to. '\*' means all tables  
in the database

```
grant select on store.* to someUser  
identified by 'pa$$'
```

The MySQL account to  
assign the privilege to

The password to assign to the  
account (may be omitted if the  
account already exists)

GRANT associates access privileges with particular users.

## REVOKEING PRIVILEGES AND DELETING ACCOUNTS

- The REVOKE command takes away privileges
- General form:

```
revoke privileges on item from user
```

- Example:

```
revoke update, delete on cart.users from badUser
```

- The drop user command deletes an account

```
drop user badUser
```

REVOKE removes privileges on tables from user accounts.

## FINE GRAINED CONTROL

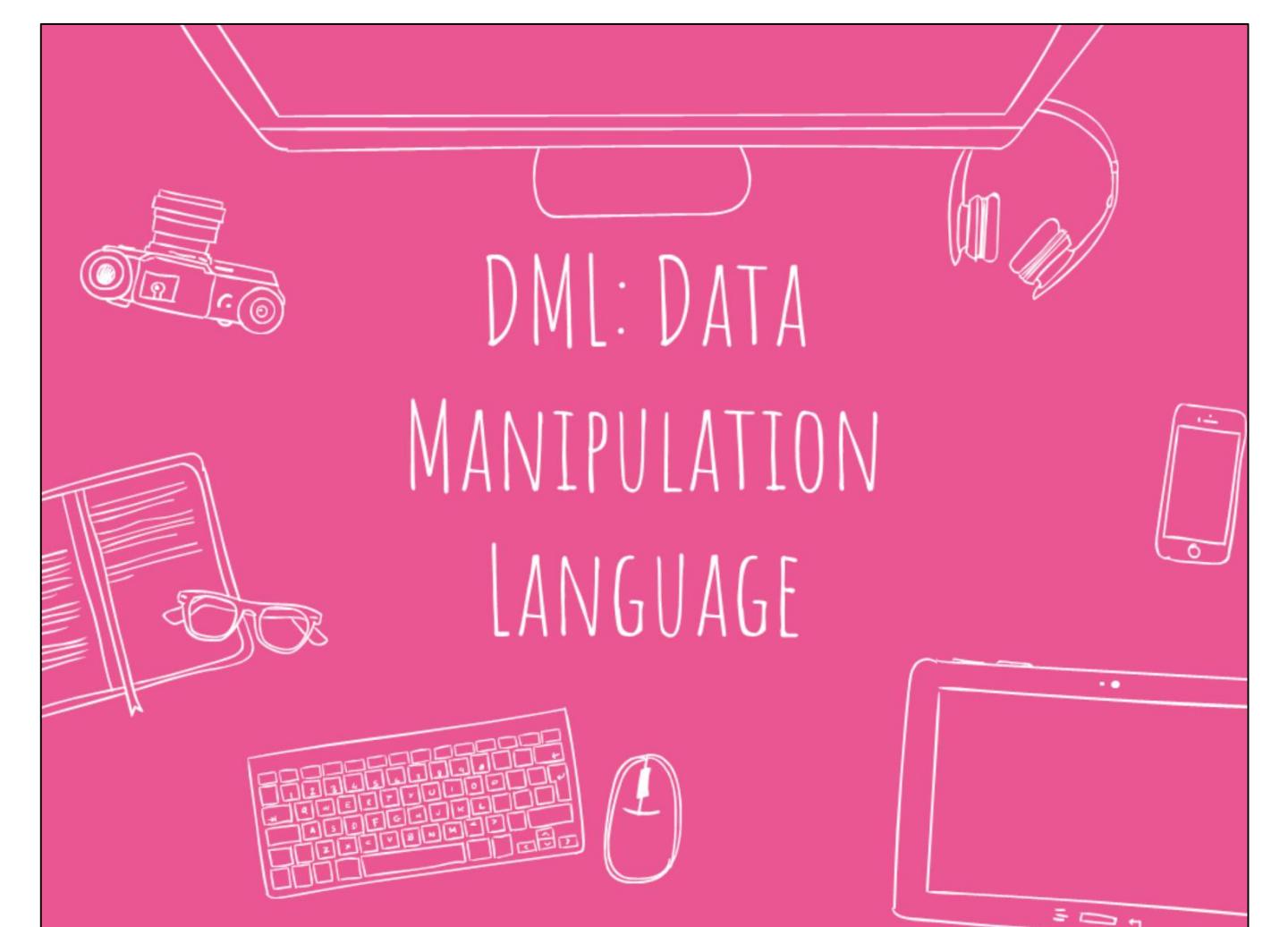
A Library application has five pages that access the database. For each one, identify what types of access is required to each table

- select, insert, update, delete (or some combination)

Script	Access to books table	Access to borrowers table
addbook.php	?	?
addborrower.php	?	?
booksearch.php	?	?
checkin.php	?	?
checkout.php	?	?

The purpose of this exercise is to relate what goes on in your applications to the kind of activities in a database, in other words, to relate PHP to SQL.

A PHP product page which creates new items will need to INSERT into an items table, for example.



# DML: DATA MANIPULATION LANGUAGE

Often the most complex aspect of querying a database is not INSERTing data or CREATEing tables, but SELECTing the data you want.

In web development projects you may want "cheap items" ( $\text{price} < 100$ ) or sale items or items which suit a combination of conditions. You may find the raw data the database stores not quite suitable for your application, you may want an item's price plus VAT ( $\text{price} * 1.2$ ) either in place of price or as an additional field in your result set.

The SELECT statement supports all of this and more, coupled with the many functions SQL provides. Your querying can be very powerful.

Before we look at select we'll cover the UPDATE statement, which can also have similarly complex semantics. You can increase the prices of all items under £100 by 20% or capitalize the first names of all of your users.

## CHAPTER OVERVIEW

### Data Manipulation Language (DML)

- Referential Integrity
- Drop table / database
- Selecting data
- Select... from
- Selecting columns
- Column aliases
- Distinct entries
- Filtering using where statements
- And / Or logic



## UPDATING RECORDS



The **update** statement will change values stored in the database.

- You can update everything in a single table
- Or target it more specifically based on a where clause

```
update [table] set [column] = [value]
    -- will update all the records in the table
    -- with this value!
```

```
update [table] set [column] = [value]
where [some condition is true];
    -- will update just the records that are
    -- referenced by this condition
```

```
UPDATE users SET username = 'sherlock'
    WHERE username = 'sholmes';
```

The UPDATE command is a little unusually structured, unlike INSERT which separates field names, UPDATE uses a SET clause to list them.

UPDATE users SET username = 'guest', 'password' = 'guest' WHERE id = 0;

This comma separated list uses “=” (equals) to associate fields and values.

## DELETING RECORDS

- The delete command will delete all rows that match the criteria you specify
- If no criteria is specified all rows are deleted

```
delete from [table] where [condition is true];
```

```
delete from items;
```

-- will delete everything in the table!

```
delete from items where id = 2;
```

-- will only delete the item with id 2



Mirroring INSERT INTO, the DELETE FROM statement is another command which compounds two keywords together.

DELETE FROM is given a table and a condition. Whichever rows match that condition will be removed from the table.

There is no "Are you sure?" prompt here. If something matches it will be deleted. It is always a good idea to run queries on test databases while you're developing an application and keep the production database separate.

## DROP TABLE / DROP DATABASE

The **drop** keyword deletes an entire table or database

- The **delete** command removes records

```
drop table [table name];  
drop table users;
```

```
drop database [databasename];  
drop database store;
```

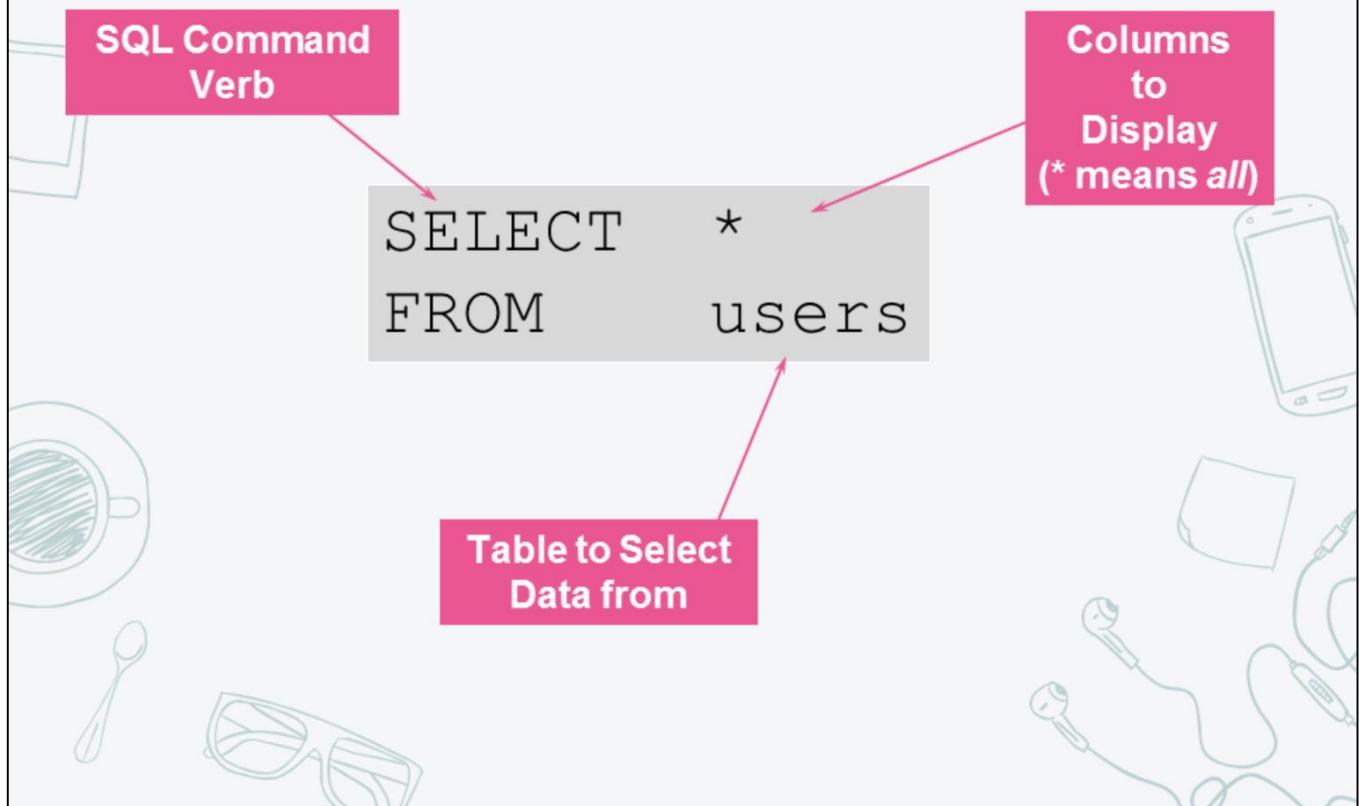
- THERE IS NO PROMPT
  - THERE IS NO WARNING
  - THERE IS NO UNDO FUNCTION
- Be careful with these commands!



Mirroring CREATE, the DROP statement can be used with TABLE and DATABASE.

This command simply takes the name of the table or database you wish to remove and removes it.

## SIMPLE SELECT

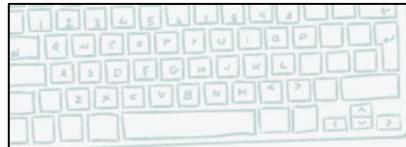


The simplest form of the SELECT statement is:

```
SELECT      *
FROM        table
```

The '\*' signifies that you want to display data from all of the columns in the table. A simple SELECT like this will display data from every column in the table mentioned in the FROM clause. Notice how powerful this is compared with a procedural programming language, where you would have to use a looping construct to achieve the same result.

Indeed SQL is sometimes seen as a functional-declarative language a paradigm that is neither procedural nor object oriented. In the object oriented world data is far more "encapsulated" thus making queries across objects impossible. SQL is a very well thought out language in this sense, all the data is at your fingertips and you can write very powerful queries.



## STATEMENT FORMAT



- SQL is a free format language
  - Use new lines, tab keys and indentation to make it readable
  - White space is ignored by the parser
- Make use of comments, ignored by runtime engine

```
SELECT      *          -- all columns  
FROM        items
```

Comment

SQL is described as a free format language. This means that you are free to input statements over as many lines as you wish with as much blank space as you want. Use this feature to make SQL statements more readable. Sometimes the phrase "whitespace is ignored" is used – but not always, for example, the strings 'hello world' and 'helloworld' are different.

SQL also supports comments. A comment starts with a pair of hyphens (--) and continues until the physical end of the line (the next newline character). Some systems also allow you to use /\* and \*/ to mark comments that run down several lines. In MySQL /\* \*/ style comments are perhaps more common than the double-dash-style.

## SPECIFYING COLUMNS

Columns  
to  
Display

```
SELECT      username, email  
FROM        users
```

You have two choices

- Use \* or list the columns separated by commas
- Columns may be listed in any order
- Columns are displayed in the order you specify

The syntax of the SELECT statement can be extended by replacing the \* with a comma-separated list of column names. Only these columns are then displayed.

One problem with this is that you need to know the names of the columns in the table. The \* syntax avoids this, but for large tables it can result in vast amounts of data being displayed.

In web development scenarios you typically know the names of the fields you require. Moreover, your queries will be more efficient if you restrict the amount of data you're querying to exactly that which you need. This will ultimately make your webpages load faster.

## CALCULATED (VIRTUAL) COLUMNS AND ALIASES

```
SELECT name,  
       price * 0.7 AS 'USD'  
FROM   items
```

Real  
Column

Calculated (virtual)  
Column

Column  
Alias

- 'AS' keyword can be omitted

- Quotes can also be omitted if no spaces in alias name

A 'new' column can be created by combining existing columns into expressions.

In the above case we can create a column in the result by multiplying the current price by 0.7. In the result, this is displayed like a column from the table.

However, this column only exists during the life of the query: it's in the result set but not the table. As soon as the query is complete the column disappears. This means that you can display virtual columns but not insert or update values within them.

The fields in a SELECT clause of an SQL statement may be renamed using the AS syntax shown above. This alias is used as the heading for the column if the results of the statement are being displayed on the screen.

Most database manufacturers support column aliasing, but nearly all omit the use of the AS keyword (which is optional in SQL2). This can lead to subtle bugs occurring in statements if a comma is accidentally omitted.

When updating old websites with a new database you may find that the PHP code uses particular column names that you want to change in the database. You could use column aliasing to give PHP what it wants while switching to new a new database.

## COMBINING STRINGS - CONCATENATION

```
SELECT concat(firstname, ' ', lastname)  
FROM users;
```

Concat is a function in SQL

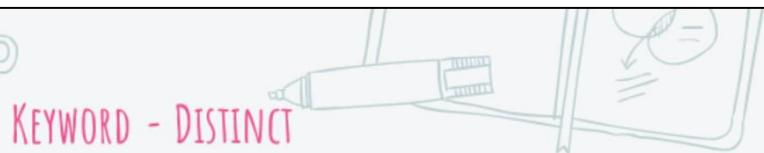
- It takes in a number of strings (identified using the quotation marks '')
- The output is a single string with all the parts in order
- We can run these as a standalone statements

```
mysql> select concat('hello', ' ', 'world');  
+-----+  
| concat('hello', ' ', 'world') |  
+-----+  
| hello world |  
+-----+  
1 row in set (0.00 sec)
```

concat() is a function which does the same job as the concatenation operator in PHP.



## KEYWORD - DISTINCT



- Distinct returns only unique values

```
mysql> select distinct price from items;  
+-----+  
| price |  
+-----+  
|      500 |  
|     1700 |  
|    2000 |  
+-----+
```

- The distinct keywords will look for unique combinations of the fields given

While the normal SELECT operation defaults to returning all the rows, there is a modifying clause, DISTINCT. DISTINCT causes the engine to eliminate all the duplicate result set rows and to return one copy of each.

Typically to eliminate duplicates the engine must do a sort, hence the result set is sorted. However, don't rely on this. Database engines can use other techniques of removing duplicates that do not require a sort.

Using DISTINCT with an SQL query can be a sign of a design mistake with your table structures. If you have repeated data perhaps that should be a single entry in another table with a repeated id.

## SORTING THE RESULTS

```
SELECT      *
FROM        users
ORDER BY    username, firstname
```

By Specific Column(s)

- ORDER BY username ASC
- ORDER BY username DESC

The SQL Standard does not specify the order in which results are displayed. This is implementation dependent.

You can specify an order by using the ORDER BY clause. The column can be specified by name or by its ordinal position in the column list.

You may specify several columns and have the data sorted in ascending or descending order e.g.:

```
SELECT * FROM users
ORDER BY username DESC, lastname;
```

Consider foreach()ing over a MySQL result set in PHP. With the arrays you have seen so far the foreach tends to follow the order the array was defined. The ORDER BY clause will determine which key comes first for a result set.

## BOOLEAN OPERATORS

- We can filter the results of any select statement by using the **where** clause
  - This requires a boolean statement (predicate)
- Less than, greater than, equal to, in a range
  - $1 < 2 \rightarrow \text{true}$
  - $1 > 2 \rightarrow \text{false}$
  - $\text{'hello'} = \text{'world'} \rightarrow \text{false}$
  - $\text{'harry'} \text{ in } (\text{'tom'}, \text{'harry'}) \rightarrow \text{true}$
  - 55 between 10 and 100  $\rightarrow \text{true}$
- In SQL queries, use fields in your predicates
  - `where username = 'sholmes'`

Some examples for now, and more follow:

```
SELECT * FROM items WHERE price < 100
```

```
SELECT * FROM articles WHERE hits BETWEEN 1000 AND 10000
```

## LIMITING ROWS WITH BASIC OPERATORS

```
SELECT *  
FROM   items  
WHERE  price > 500
```

A Basic 'Relational' Operator

Limits rows to those matching the condition

Numeric column

Value

Think of a WHERE clause as an 'IF' statement

- Include this row in result set 'if' the test returns true
- Basic operators include  $>$ ,  $\geq$ ,  $<$ ,  $\leq$ ,  $=$ ,  $\neq$

Think of a WHERE clause as an 'IF' statement i.e. "Select this row IF boolean\_expression is true". The boolean\_expression or "predicate" may contain function calls i.e. WHERE SUBSTRING(lname,1,1) = 'P'

The WHERE clause can be used to limit the rows to those that meet some specific criteria.

There are six basic operators that may be used in a WHERE clause to compare values and to select rows that meet a test:

=	Equal
$\neq$	Not Equal
!=	
<	Less Than
>	Greater Than
$\leq$	Less Than or Equal To
$\geq$	Greater Than or Equal To



BETWEEN



```
SELECT *  
FROM items  
WHERE price BETWEEN 500 AND 1000
```

Starting Value

Stopping Value

- A lot easier than typing

```
WHERE price >= 500 AND  
      price <= 900
```

- Values are inclusive

BETWEEN is used to specify a range of values between (and including) two limits.

The first value sets the start of the range; the second value sets the end. The AND keyword between the two values is mandatory.

Note: The ANSI Standard expects the lower end of the range to be specified first, otherwise no rows will be returned.



IN



**Peter  
George  
Tom  
Mike  
Sandy  
Eleanor  
Bill  
Gary  
Grace  
Harry  
Samantha  
Dick**

```
SELECT    firstname
FROM      users
WHERE     firstname IN ('Tom', 'Dick',
'Harry')

-- easier than coding
WHERE     forename = 'Tom' OR
          forename = 'Dick' OR
          forename = 'Harry'
```

**Tom  
Dick  
Harry**

The IN predicate allows us to specify a list, or set of values to test against. This can simplify tests where the result may take one of several values. The version using IN is easier to read and maintain.

Note: WHERE fname = 'Fred' OR 'Tom' OR 'Dick' OR 'Harry' is invalid syntax.

It is most unusual for the database engine to be ‘case sensitive’, Oracle being the notable exception.

If it is, use this syntax to find any ‘FrEd’

WHERE UPPER(fname) = ‘FRED’

or

WHERE LOWER(fname) = ‘fred’

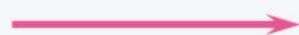


NOT



**Peter  
George  
Tom  
Mike  
Sandy  
Eleanor  
Bill  
Gary  
Grace  
Harry  
Samantha  
Dick**

```
SELECT    firstname  
FROM      users  
WHERE     firstname NOT IN  
          ('Tom', 'Dick', 'Harry')
```



**Peter  
George  
Mike  
Sandy  
Eleanor  
Bill  
Gary  
Grace  
Samantha**

The NOT clause is a modifier that may be used to reverse the boolean outcome of a test. It places into the result set any rows that do NOT meet the test criteria.



LIKE



## When used with 'LIKE'

- ‘\_’ Matches any single character
- ‘%’ Matches any number of characters (including none)

WHERE surname LIKE ‘a%’

← starts with ‘A’

WHERE surname LIKE ‘%a’

← ends with ‘A’

WHERE surname LIKE ‘%a%’

← contains an ‘A’

WHERE surname LIKE ‘\_t%n%r\_’

← has a ‘T’ in pos 2  
has an ‘R’ 1 char from end  
and an ‘N’ between them

- **LIKE** is only used with character columns

e.g ‘stationary’

Normal comparison tests (=, <, >, <>, etc.) on strings must match the tested string exactly. LIKE allows us to use wildcard searches on the strings.

There are two command characters, underscore ( \_) and percent ( % ). Underscore is used to match any single character. Percent is used to match any number of characters, including no characters at all.

Some systems, for example MS SQL Server, allow further comparisons with the LIKE keyword e.g.

- [ ] Any single character within the specified range [a-f] or set [abcdef]
- [^] Any single character not within the specified range [^a-f] or set [^abcdef]

## LOGICAL OPERATORS AND / OR

```
SELECT *  
FROM users  
WHERE lastname = 'Holmes'  
      AND password = 'sherlock';
```

Both conditions must be true

Every time you say AND you are likely to get less rows

```
SELECT *  
FROM users  
WHERE lastname = 'Holmes'  
      OR password = 'sherlock';
```

Either (or both) conditions can be true

Every time you say OR you are likely to get more rows

Putting parentheses into either of the queries above will make no difference.

Potentially and very likely, every time you say AND less rows will qualify for selection. ‘This’ must be true and also ‘that’ must be true.

Every time you say OR you are saying “I want these rows AND I want these rows” – i.e. where either (or both) conditions are true the row is selected.

You can have as many AND’s as you like and parentheses might make things clearer but will not change the outcome.

Likewise you can have as many OR’s as you like and parentheses might make things clearer but will not change the outcome.

But when you mix & match AND & OR in the same query you have to think harder about it.

## NULLS

- Basic premise of an RDBMS is the concept of optional columns
  - NULL means 'not applicable' or 'unknown', different from zero or blank
- On INSERT of a row, must supply values for mandatory columns
  - Other columns may be left as NULL (assuming no 'DEFAULT' value)
- NULL propagates through expressions:  $(5 + \text{null})$  is null, not 5
  - Nothing is equal to null, not even  $\text{null} = \text{null}$
- WHERE clause expressions will evaluate to TRUE, FALSE or NULL
  - Need to think 3 way logic
  - Only rows whose expressions evaluate to TRUE are output
- Can use IS NULL to retrieve rows with NULL entries:

```
select *
from users
where email is null
```

NULL Handling: SQL2 supplies the COALESCE function for trapping and replacing NULLs e.g.

```
SELECT emp_no, fname, lname,
COALESCE (bus_phone, home_phone, mobile_phone, 'Not on phone')
FROM employees
```

COALESCE returns the first non-NUL value from the arguments supplied. In this example, as a last resort, the string 'Not on phone' is returned if all 3 phone number columns contain NULL.

SQL Server supports COALESCE and also ISNULL, a cut-down version that only takes 2 arguments.

PHP's NULL coalesce operator derives its name and inspiration from here.



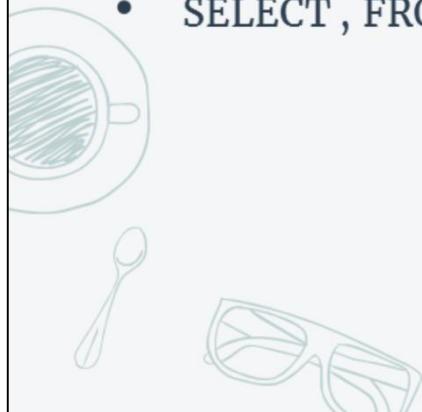
## CURRENT ORDERING

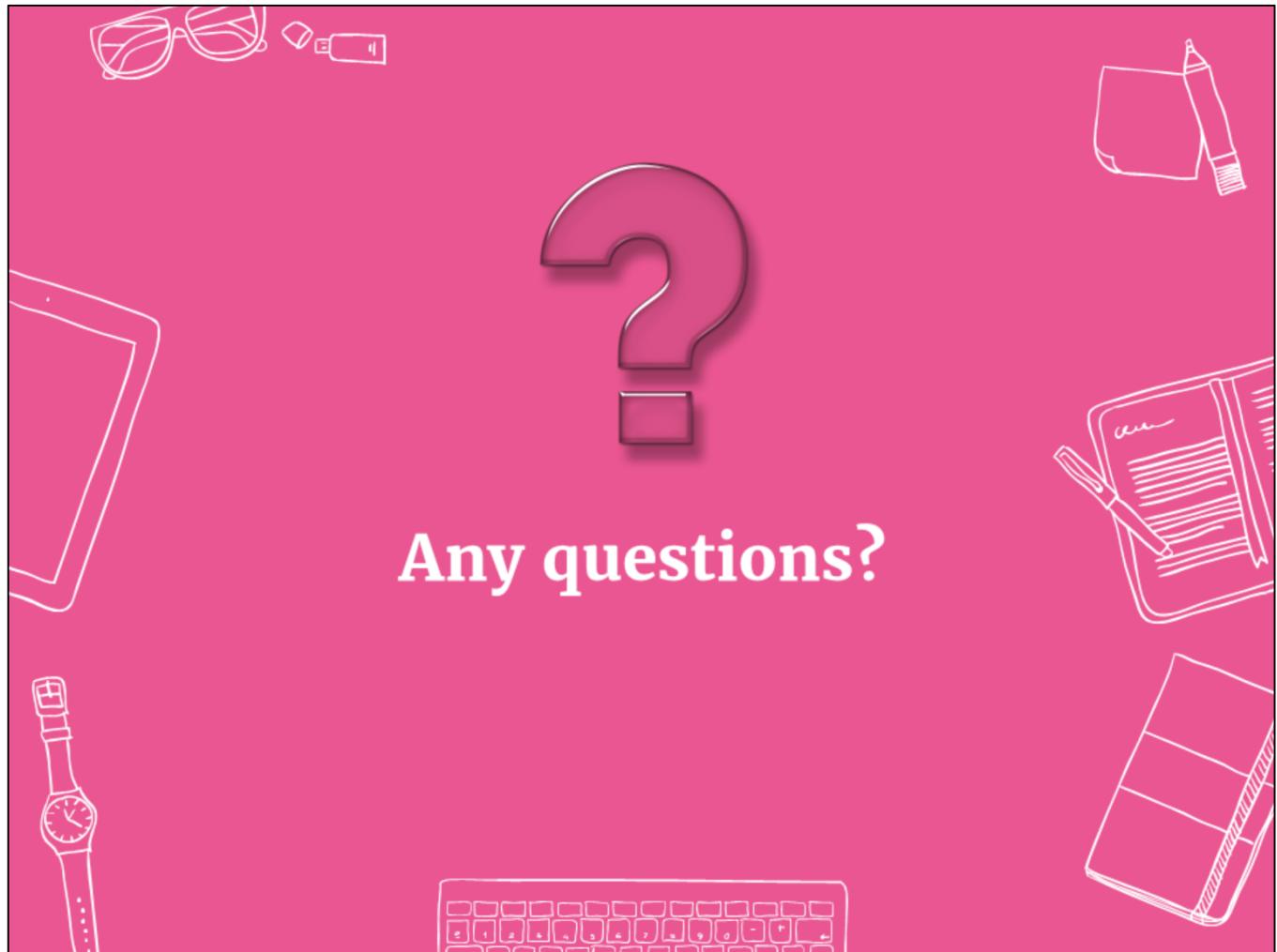


- The order of the statements is as follows:

```
select [columns, calculated columns] [as alias]  
from [table name]  
where [some condition is true]  
order by [a field] [asc/desc]
```

- SELECT , FROM, WHERE, ORDER BY

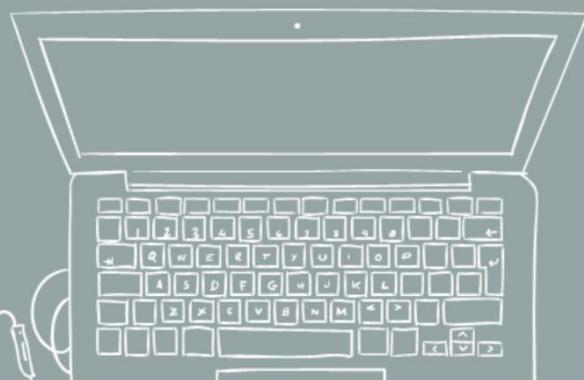




**Any questions?**



## EXERCISE 19



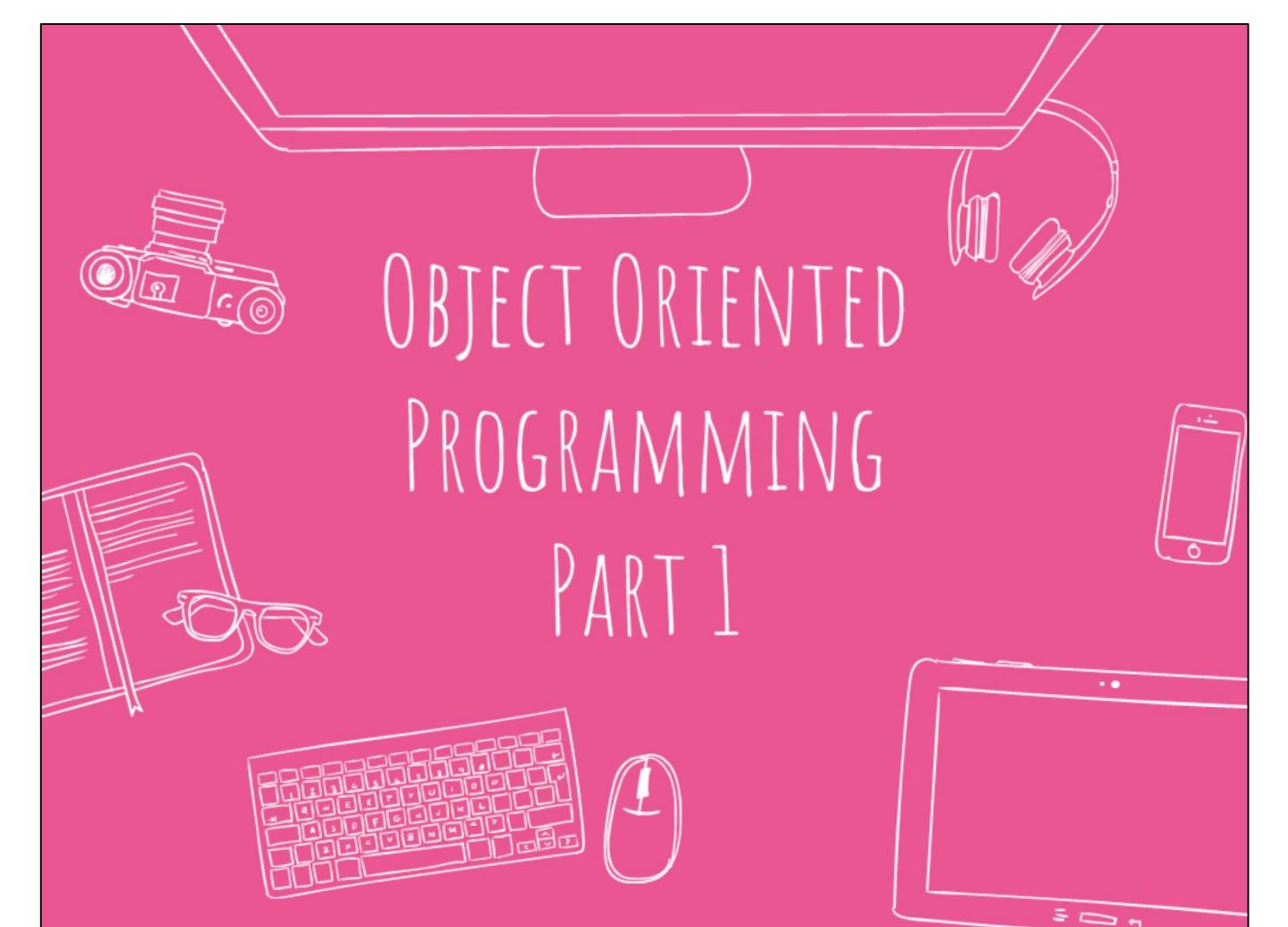
## RESOURCES

MySQL Documentation

<https://dev.mysql.com/doc/>

Select Syntax

<https://dev.mysql.com/doc/refman/5.7/en/select.html>



# OBJECT ORIENTED PROGRAMMING

## PART 1

In most of your previous PHP code, data and functionality have been separated. Data is passed into functions which perform their operation and return new data that you can continue to use.

This approach entails a way of thinking about programming problems. If your task is to build a shopping cart you begin to think in terms of arrays and operations on arrays – you think that to delete an item is really to take some data (a database or array) and perform some operation (a delete) and get new data back out. This is a perfectly reasonable way of thinking about programming, however it is not the only one.

Object Orientation as the name suggests involves orienting your applications around objects. Objects are fusions of data and code, and to some people, more intuitively fit certain problems. The original idea comes from trying to model the physical world: there is a pen which is an object that has certain characteristics (properties, attributes) and which has certain behaviours, writing for example.

A shopping cart system under this "paradigm" is then rephrased: a cart holds items, and a cart itself removes them or adds them. The objects are: cart, and item; each of which has certain behaviors such as "adding to".

This chapter will present the PHP syntax which supports this view of programming and expand on some of the concepts involved.

## CHAPTER OVERVIEW

- Classes
- Constructors
- Methods and Attributes
- Visibility
- Statics
- Type Declarations

There is a lot of new syntax introduced in this chapter and quite a few new concepts. Nevertheless nearly everything is very important to your education in PHP. Object orientation is ubiquitous in contemporary web development and this syntax will be quite common.

However, do not expect to have mastered it from these slides alone. As with all programming skills, practice is highly important. Object orientation, especially, is a matter of practice – not only in syntax, but in understanding when and why to use it.

The goal of this chapter is to help you on your journey to learning object orientation, to present the syntax, concepts and code you need to begin.

## OBJECT ORIENTATION

- Thus far PHP programs have focused on functions and values

```
$output = fn($input);
```

- PHP is part of a family of languages which can combine data and behaviour into objects

```
$output = $input->fn();
```

- \$input is an object
- Objects contains both data and behaviour
- To access the internals of \$input, use ->
- fn() is a function which belongs to \$input and is "called on" \$input
- In this style of programming 'objects' which contain both functions and data are the main tool for designing the program

Compare the order of \$input, \$output and fn in the above examples. \$output is always first. Then we have either a function name or another variable.

If the function "comes first", the design is procedural (function-oriented). If it's the data followed by the function, it's object-oriented: this variable is an "object". It is a combination of function and data.

In the first case we're asking a function to transform our input into our output, in the second case we're asking the data itself to manage its own change.

The code here is only to illustrate this conceptual point about a shift in the way of thinking about designing a program and you will see more illustrative ones soon.

In the first section of this chapter the focus will be on these "objects": how they are created, how they have behave, what properties they have. Then their internals will be exposed and you will see how you can design objects of your own.

## OBJECTS AND CLASSES

- Objects are created from classes
- In English, 'class' denotes a group of common objects: the class of all the objects in a penpot is Pen
  - they all have a colour
  - they can all write
- In programming a 'class' is a blueprint
- A class defines the members that objects of that class have:
  - what attributes those objects have (data)
  - what methods those objects have (behaviour)
  - methods and attributes are known as members

```
$myPen = new Pen();           // `Pen` is the class  
                           // $myPen is the object
```

```
$myPen->write("Hello");  
// object->member
```

Hello

To get hold of an object you first need a class. A class defines what an object looks like, it is something like a specification. If you want an instance of a house (that is, a particular house) you have to give the blueprint to the builder and he'll create it for you.

Here, the builder is the `new` operator which is given the name of a class, a blueprint.

The blueprint metaphor has its limits however. It's also possible to think of a class as defining what characteristics something needs to have in order to be in a group. So that the class "pen" says that everything which is able to write and everything that has a colour is a pen. This is closer to the English meaning of class as group.

Objects have behaviour (functions defined on them) and state (variables defined on them). A class then defines the behaviour and characteristics (attributes) of an object: it tells you what the pen does and how it does it, and what attributes it has (e.g. colour) so that you can create particular pens of that kind.

In this code example all you can tell is that \$myPen is a kind of Pen; it is part of that class or group of all things Pen.

## FEATURES OF OBJECTS: CONSTRUCTORS

- When creating instances we can specialize the kind of object we get by passing attributes or other data straight to the class

```
$redPen = new Pen('red');
```

```
$bluePen = new Pen('blue');
```

```
$myPen = new Pen; //optional if nothing passed
```



- The syntax looks like a function call and, in part, it is
  - there is a function called a 'constructor' which is called and passed this data
  - in general, a constructor is a function which makes objects – in PHP `new` makes the object and the constructor just handles passing the data

For now, let's assume there is a class available, Pen. What can we do with this class?

To create an object from a class we use the `new` operator, but a class is free to ask us for more information about the kind of object we want to create: what particular colour should the pen be?

If there is information to pass then the new statement will look a little like a function call prefixed by `new` -- and it is. There is a function (called a constructor) the class defines which tells `new` how to create objects, this function takes parameters that are passed when new is used.

## FEATURES OF OBJECTS: METHODS AND ATTRIBUTES

- Objects store their data in attributes

```
$myPen = new Pen('red');           //construct a new pen object  
  
echo $myPen->colour;             //get the attribute colour  
  
$myPen->colour = 'blue';          //set the attribute colour  
  
$myPen->name = 'Parker';          //create a new attribute  
  
$myPen->write('Hello');          //call the write method
```

Hello

- Objects have methods

- the name for a function which belongs to an object

```
$redPen = new Pen('red');  
$redPen->write('Hello');
```

Hello

Objects store their data in attributes which are little more than variables which live inside objects. You can access these with the thin-arrow operator as shown. Functions that live on objects are called methods, they are accessed the same way as attributes, using a thin-arrow.

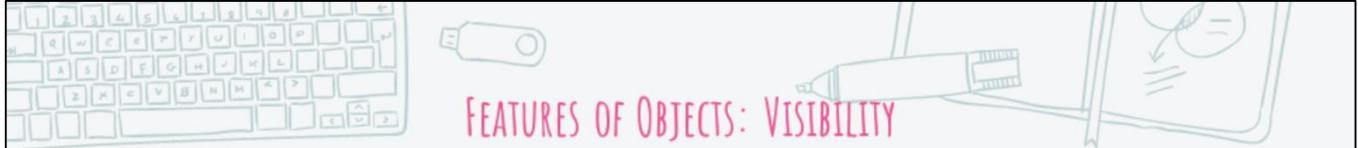
Data that is held in one place, and particularly data which affects behaviour, is known as "state", as in the state of a light switch might be "on". Objects therefore have state and behaviour: attributes and methods.

In the example above a new pen object is created. This object is defined to have a colour attribute whose value is provided when new is called. The attribute on this object is then overwritten. Therefore, the pen is initially in the red state and then is changed to the blue state.

A new attribute can be created at any time, by just assigning to a name on the object.

The write method on the pen object uses the colour attribute to determine what colour to write the message, since we have changed the colour attribute it writes in blue. Thus the state of an object (its attributes) are mostly to specialize its behaviour.

Note that the write method wasn't passed a colour option, which would be the procedural style, but is aware of \$myPen's colour by belonging to the same object. Different pens will write in different colours.



## FEATURES OF OBJECTS: VISIBILITY

- Objects may have data that is entirely internal to itself
  - if you access it from "outside" you get an error
- Methods and attributes visible from the outside are *public* and from inside are *private*

```
$myPen = new Pen('red');

echo $myPen->colour;           //ERROR: private!

$myPen->name = 'Parker';       //create a new attribute
echo $myPen->name;
```

The state of an object is mostly hidden, that is, its attributes are not accessible from outside the object as in the previous example. Here colour has been set to private, so attempting to access it directly produces an error.

However, any new attributes you create from outside the object are always public and thus accessible.

Objects get most of their state from their constructors, that is, you should pass most of what an object needs when constructing it. Otherwise later modifications will change the behaviour of the object and make it unpredictable.

In the previous slide the pen's colour was changed after it had been created, so if you were to look at the new statement you'd be confused about what colour the pen would write in. Code should behave, wherever possible, as expected.

## CLASSES

- A class statement defines a class
  - a blueprint for the objects of that class
- The typical layout when defining PHP classes is:
  - first all the attributes
  - then the constructor (a method called `__construct`)
  - then the methods
  - methods leading with `\_\_` (double underscore) provide special functionality

```
class Pen {  
    private $colour;  
    public function __construct($colour) {  
        $this->colour = $colour;  
    }  
    public function write($message) {  
        echo $message, "[ in $this->colour ]";  
    }  
}
```

Thus far objects may seem quite strange or mysterious. They appear to have many interesting kinds of properties, from here-on you will learn how they acquire them.

Firstly is the class statement. A class defines the state (attributes) and behaviour (methods, functions) that all objects of this class will have.

Class definitions begin with the class keyword, the name of the class and a code block ( {} ). Within this block are defined items which look a little like variables and functions. These can appear in any order however there are well established conventions.

First should come the definition of the attributes, with what visibility each attribute will have -- 'private \$colour' means that all objects of this class will each be able to store a colour (particular to each of them) and that this will be private: not accessible from outside but only from within its methods.

Then follow the methods. Each method has a visibility modifier also: whether those methods can be accessed from within or outside the object.

Visibility will be discussed in more depth soon.



\$THIS



- We call a method on an object, it is the same method for all objects of that class

```
$redPen = new Pen('red');  
$bluePen = new Pen('blue');
```

```
$redPen->write('hello'); //same method  
$bluePen->write('hello'); //same method
```

hello [in red]  
hello [in blue]

- Notice the different output
- How does a generic method access the particular data of a particular object?
  - inside methods '\$this' refers to the particular object a method is called on

```
public function write($message) {  
    echo "$message [in $this->colour]";  
}
```

There are some conceptual problems dividing up objects into particular values and classes into their blueprints. Since classes, in some sense, have a life of their own.

It's a little more accurate to think of an object as being only the particular state it has been assigned, like a "super variable". When you call a method on an object it finds the method on the class – each object is not actually given its own copy.

So what happens when you have two objects, of the same class, each initialized with different data? They do in fact still call the same method. In the example we have two pens: each is an object of the pen class; however one has been given a red attribute and the other a blue. And calling write() produces different results for each!

The method defined on the class, write(), has implicit access to a variable called \$this – this variable is the particular object calling write. In the case of \$redPen, \$this is \$redPen so write uses itscolour when writing the message.

Note: You could think of object-oriented methods as having a hidden parameter, \$this, so that function calls are actually Pen::write(\$redPen, 'hello')

## RETURNING FROM METHODS (AND \$THIS)

- Methods are just like ordinary functions and follow the same syntax and behaviour, they may
  - Return data
  - Return objects
  - Execute other functions
  - Do nothing
- Inside classes however you may also return '\$this'
  - I.e. the object itself – known as "fluid chaining"

```
//class Pen { ...
public function write($message) {
    echo $message;
    return $this;
}
//...}

$myPen->write('hello')->write('world');
```

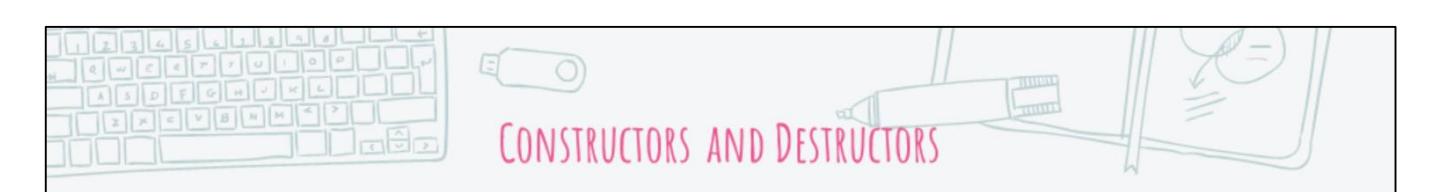
Methods defined on classes behave just as ordinary functions, except, of course, having their hidden \$this parameter.

In the example above \$this is returning, thus the object which called the method is returned.

Therefore if you were to call \$redPen->write() the object returned would be \$redPen, so you could again call write() on it, \$redPen->write()->write()... and so on.

Sometimes there isn't a sensible return value for an object's method, because a lot of the code will be occupied modifying the objects state (e.g. changing its colour) which is a purely internal operation. So you might find returning \$this useful. It allows you to keep calling methods in an easy way.

This technique however is a quite advanced, and if you are understanding how it works it is a good sign that you're understanding the finer details of object oriented syntax.



## CONSTRUCTORS AND DESTRUCTORS

- Constructors are responsible for making sure the object has sensible values for its attributes
  - And is in a usable state after being created

```
private $colour;  
public function __construct($colour) {  
    $this->colour = $colour;  
}  
public function write($message) {  
    if(empty($this->colour)) { die('NEED A COLOUR!'); }  
    echo $message;  
}
```

- PHP also provides the rarely used **\_\_destruct** which is called when there are no more references to that object
  - I.e. it goes out of scope

The constructor, always named `__construct`, is the method called by the `new` operator. Any arguments you pass while instantiating are passed to this constructor method.

The constructor is responsible for giving the object everything it needs to perform its job.

Suppose you were to write, `$myPen = new Pen(null);`

That's is, pass `null` as the argument for the colour – then the write method wouldn't work at all: there would be no sensible colour to write a message with.

Likewise a constructor is defined so that when the object is created it is able to work straight away. This is not required by PHP's syntax – it is quite possible to write an empty constructor and expect developers to call various methods later on in order to get the object working properly. However unless absolutely required, this is a bad habit.

For example, `$pen = new Pen(); $pen->setColour('red');`

would also work, however if we do not call `setColour()` then pen has no colour and is therefore essentially "broken".



## FEATURES OF CLASSES: CONSTANTS

- Classes are not entirely dataless
  - remember objects have their own attributes
- Classes may also define constants

```
class Pen {  
    const RED = '#A00';  
    const BLUE = '#00A';  
    const GREEN = '#0A0';  
  
    public function __construct($colour = Pen::RED) {}  
}  
  
$myPen = new Pen();  
echo $myPen::RED;
```

- Note `::` - known as the scope resolution operator
  - used for accessing static items e.g. fixed data on classes

As mentioned earlier, classes are not merely empty schema – they hold methods. And, alas, they also hold data. However, data held on the class itself is almost always static, literal, data which cannot be modified. Within class definitions you can define constants which live on the class itself. Thus in the above code example, the full name of the constant RED, is Pen::Red.

Let's be very clear here, since there is now two kinds of state in play. The attributes defined by a class is purely schematic, they hold no data themselves; the data is held within each object instance. They say only that all objects of this class will have an attribute of this name, though what particular value it will have will depend on the object in question.

Thus if there is a class Pen which defines an attribute colour, all pens have a colour. However each of the dozen pen instances you may create each have a different colour.

In the case of class constant, the class itself holds the data. You can either access the constant on the class directly (Pen::RED) or via the object (\$myPen::RED). Either way the object itself stores nothing.

Note: The technical name for `::` in PHP is the Paamayim Nekudotayim – which is Hebrew for double colon. "Scope Resolution Operator" however is an official alternative, and far more common. Unfortunately however, the PHP engine itself will report errors which concern `::` as T\_Paamayim\_Nekudotayim



## FEATURES OF CLASSES: SELF

- The **self** keyword refers to *the class, from inside the class*

```
class Pen {  
    const RED = '#A00';  
    public function __construct($colour = self::RED) {}  
  
    //returns new object of class pen  
    public function MakeSamePen() {  
        return new self($this->colour);  
    }  
//...  
}
```

- Using `self` to refer the class is idiomatic
- It allows the class to be written the same and its name changed

If you are within the class, the keyword `self` will refer to the class itself. For example, if you wanted to refer a constant defined on a class (e.g. Pen::RED) from within the class you could write (self::RED).

The main advantage to using self is that the code of the class remains the same even if the class name changes. Often class names will change as the purpose of the class undergoes revision, and throughout your application design process.

Equally you can write `new self` which will create an object of the class, this is idiomatic for creating instances of a class from within a class.

It is often useful to return new instances of the class from one of the methods, this is a like having a secondary constructor i.e. a secondary way of creating new instances.



## FEATURES OF CLASSES: STATIC

- Classes are far from dataless and behaviourless
- The **static** keyword can be used to give classes functions and variables
  - known as *static variables* and *static methods*
  - if they are static there is no `\\$this`, as we use the class directly not objects of the class

```
class Pen {  
    public static $colour;  
    public static function write($message) {  
        echo "$message in self::$colour";  
    }  
  
    Pen::$colour = "red";  
    Pen::write("hello");  
  
$pen = new Pen(); //all instances get the same statics  
$pen->write("hello"); //so this is red!
```

The `static` keyword can be prepended to methods and attributes and forces those things to live on the class itself *only* and have no access to any object states.

In the case of static methods this means there is no "\$this" available, i.e. you call the method Pen::write() without instancing any object and therefore without having any object-specific state.

Static variables are particularly confusing and unsightly kludges, they essentially turn your class into a real repository for variable data rather than the somewhat neater "schematic-only" intention behind them.

As a general rule, avoid statics. A public static on a class *is* a global variable (a bit of mutable state available everywhere). Static functions may be reasonable, however they are functionally identical to ordinary procedural functions.

Note: Typically if you add one static function to a class you have to add others, and so on to each related class, because you lose \$this and instance data. This causes your whole application to become mostly static, which is to say, procedural. If you want to write procedural code, in a deliberate and well-designed fashion, it's better to do so without the pageantry of classes/statics.

## CLASSES ARE (NOT) TYPES

- In many OO languages a class denotes a type of object
  - However PHP's types, are fixed – and defined by `gettype()`
  - variables containing instances of classes are type 'object'
- It is entirely reasonable to think of a class as a new type however:
  - The class Person is a new type of data
  - A complex type, similar to an array, which is defined in part by the data it contains
  - `get_class()` will give you the actual class name

```
class Person {  
    public $name;  
    public $age;  
}
```

```
$person = new Person();  
echo get_class($person);
```

Person

- `instanceof` will tell you if an object belongs to a class

```
if($person instanceof Person) {  
    echo "It's human!";  
}
```

It's Human

The word "type" means very different things to different programmers: computer scientists, mathematicians, PHP programmers, Java Programmers, "and more" each have something slightly different in mind.

The intuition behind types is quite straightforward however: "raw data" the kinds of 0s and 1s computers process doesn't *mean* anything to us. A certain number of 0s and 1s might represent "Sherlock" but what is "Sherlock"? At one level its 0s and 1s; at another it's text; at another it's a first name of a person; at another it's a detective. We could infer several types for this string then: Text, FirstName, Person, Detective. Each of these is a perfectly reasonable "type".

In object oriented programming types are nominal i.e. given by their name. You *name*, for example, a class and the name of that class is the type of the object and that's that. A pen object is a pen because it belongs to the Pen class, it was instantiated as a Pen.

PHP however (and some other languages too) confuse the issue a little, they weren't designed with object orientation in mind at the beginning so they have their official types (int, bool, float, object, resource, ...) and classes. However it is entirely appropriate to think of a class as defining a type, a "kind of thing".

## FEATURES OF CLASSES: INHERITANCE

- Classes may be related so that objects of one class also belong to another class
  - Everything which is a Dog is also an Animal
  - Every Child can spend their Parents' money

```
$child = new Child();
```

```
$child->spendMoney();           //from Parent  
$child->getHomeAddress();      //from Parent
```

- The parent class provides everything of *public* and *protected* visibility to the child
  - Unless the child already has it (i.e. something of the same name)
  - This includes attributes, methods, but not *private*
- Children may have only one *parent*

An object may belong to more than one class. This should be quite intuitive considering ordinary objects: a group of all animals would also include a group of dogs. A dog belongs both to the class Dog and to Animal.

In PHP what classes an object can belong to is restricted. You should have one principle class – the one you 'new'. However this class can define one, and only one, parent class from which it inherits methods and attributes.

In the example above a child instance is created from the Child class. However, the child class itself does not define `spendMoney()` nor `getHomeAddress()`. They are defined on a Parent class which provides the child object with access to these methods.



EXTENDS



```
class Animal {  
    public function walk() {  
        echo "walks ";  
    }  
    public function speak() {  
        echo "speaks! ";  
    }  
}  
  
class Bird extends Animal {  
    public function speak() {  
        echo "tweet! ";  
    }  
}  
  
class Dog extends Animal {  
    public function speak() {  
        echo "woof! ";  
    }  
}
```

```
$dog = new Dog();  
$bird = new Bird();  
  
$dog->walk();  
$bird->walk();  
$dog->speak();  
$bird->speak();
```



```
var_dump(  
    $dog instanceof Dog,  
    $dog instanceof Animal,  
    $dog instanceof Bird  
) ;
```

```
walks walks woof! tweet!
```

```
bool(true)  
bool(true)  
bool(false)
```



Above is the syntax which sets up this special relationship between classes.

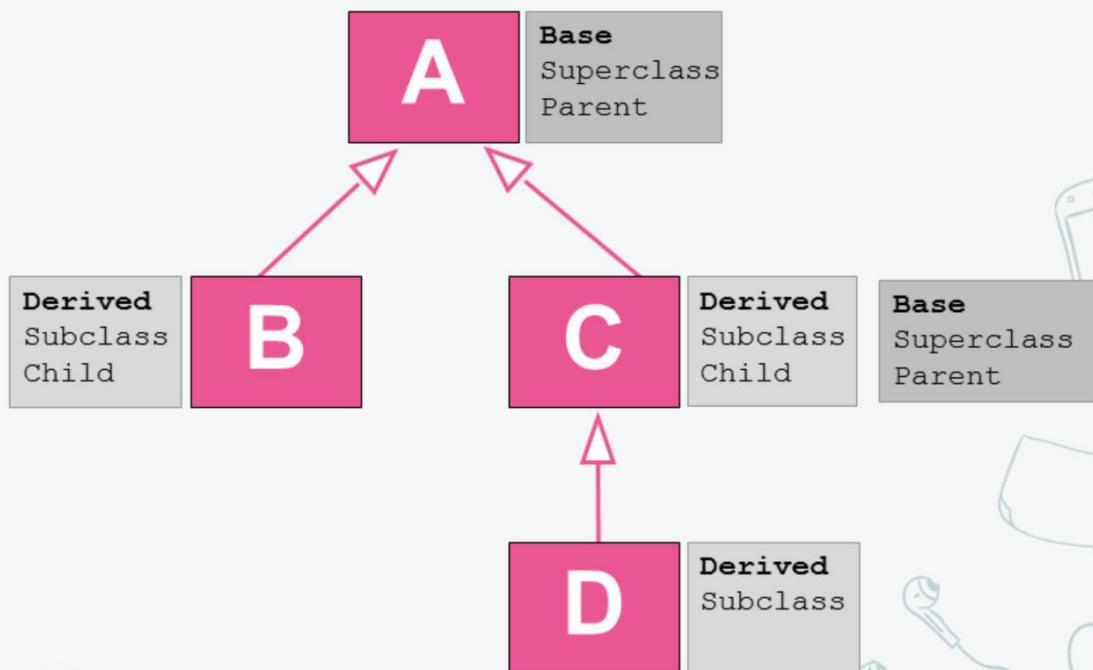
Animal is the parent (or "superclass") which Bird and Dog extend. Thus all dog objects are also animal objects. A dog object has both its particular speak() method and the walk() method the parent Animal defines. Recall that it is appropriate to think of objects as merely a bundle of state, their particular attribute values (a little like associative arrays) and if you call a method on them then they access the class which defines this method.

If you access speak() on a dog object it will first look on its principle class, Dog. If it cannot find it on that class it will look for Dog's parent class, Animal.

PHP like many languages with OO features has single inheritance, which makes looking up methods and attributes simple: for a \$dog object, first check the Dog class, then its parent Animal, then its parent --- if it has one. If PHP finds what it's looking for on \$dog then it stops.

Note: (Python, C++, and others provide forms of multiple inheritance – this is tricky to design applications with however).

# INHERITANCE



This diagram shows this single inheritance structure.

Classes B and C both inherit from A. D, further, inherits from C. Thus all objects of class D are also instances of C and A.

Inheritance is a means of distributing functionality across objects of a *similar* kind. For example, a Pen and a Pencil can both write() so perhaps a parent class WritingImplement provides the write() method, so that Pen and Pencil can specialize further.

Note: Inheritance is a very difficult design principle to master and contemporary design practice recommends avoiding it if possible. There is a tendency for programmers to create "God Objects", classes such as AnythingThatCanWrite which provides methods such as write(), type(), print(), ... and every other conceivable iteration. They extend from these classes in strange ways. This leads to classes 1000s of lines long.

Class should always be cohesive, that is, all their methods and attributes should be -- quite narrowly – about the same thing.



## TYPE HINTS AND TYPE DECLARATIONS

In the definition of functions we can use class names or types to restrict the kind of value a function will accept

```
function length(array $values) { //PHP5.1
    return count($values);
}
function call(callable $fn) { //PHP5.4
    return $fn();
}
function allowed(bool $condition) { //PHP7
    return $condition ?? "NO!";
}
function say(string $message) { //PHP7
    echo $message;
}
function exp(int $base, float $power) { //PHP7
    return $base ** $power;
}
```

One of the significant advantages of types is the ability to place them in function signatures. If you place types before variables in a function parameter list PHP will require the values you pass when calling that function to conform to that type.

Since PHP is a very forgiving, dynamic, language it is sometimes difficult to track errors down. Using type declarations in function signatures gets PHP to check your values are what you expect them to be. You do not want to find you've made a mistake when the code is running in production. Always be glad of an error during development: at least it isn't hidden and waiting to explode the production site!

You can use the ordinary "scalar" (or simple) PHP types, as of version 7, and since 5 you have been able to use class names.

Note: PHP calls these types *hints* prior to PHP7. It's known as type checking in other languages. From PHP7 onwards this feature is known as "Type Declarations" (a more accurate name, since it was never about *hinting* but declaring). Type hints used to be a purely OO feature (i.e. concerning mostly class names and `array`). However this has been expanded in PHP7 to include simple types (aka. scalar types).



## TYPE DECLARATIONS FOR OO

- **self** is a valid type declaration, here meaning exactly and only 'Pen'

```
class Pen {  
    public function writeWith(self $otherPen) {}  
}
```

- Type declarations are not just strictly nominal

```
class Animal {}  
class Dog extends Animal {}  
class Bird extends Animal {}
```

```
function withAnimal(Animal $a) {}  
function withDog(Dog $d) {}
```

```
withAnimal(new Dog()); //works!  
withDog(new Animal()); //PHP Fatal error  
withDog(new Bird()); //PHP Fatal error
```

By way of illustration, here is a type declaration within a class. The type being required here, for \$otherPen, is 'self'. As you may recall within a class 'self' refers to the class.

You could rewrite this, writeWith(Pen \$otherPen).

The second code example requires more attention. Notice how the withAnimal() function requires an Animal object however we pass a Dog object. This is because a Dog object *is* an Animal object: it belongs to both classes in virtue of the class Dog inheriting from class Animal.

Notice the withDog() function requires a Dog object. Here it will not accept an Animal object, it wants something more specific. This is key to understanding the parent-child relationship with inheritance.

The most general kind is the parent and more specific kinds are children. Animal is more general than Dog.

Suppose you ask for a Dog for Christmas, will any old animal do? No. But if you ask for any animal, will a dog do? Yes. Similarly with type declarations and inheritance.

# POLYMORPHISM

- That an object may belong to multiple classes (Animal, Dog, Poodle, ..) is called *polymorphism*
  - Or, having many forms
- Type declarations support (simple) polymorphism
  - Anywhere an Animal is required a Dog is valid
  - Anywhere a Dog is valid, a Poodle is valid
- This kind of polymorphism is called covariance
  - Any object of a child type *is also* a parent type
  - **Not** the reverse: an Animal isn't a Dog!

The technical term for "belonging to multiple classes" is polymorphism.

Consider an inheritance hierarchy, (pseudocode),

Circle extends Ellipse extends Shape

Square extends Rectangle extends Shape

We can supply a circle object OR a square object to any function which just requires a Shape. There is a sense then in which a Square is more than just one form, it is also a Rectangle and it is also a Shape – it has many forms (i.e. polymorphism).

For a more practical example, imagine you required an object with a "show()" method – you could then require, as part of your function definition, a Showy. If HtmlShow and JsonShow extended from Showy both HtmlShow and JsonShow objects would work with that methods. They are both the right shape to fit.

## VISIBILITY, GETTERS AND SETTERS

- The visibility of a method/attribute is where it can be accessed
  - **public** – anywhere
  - **private** – only inside the class
  - **protected** – inside the class or available to derived classes
- The default visibility is *public* if not specified
  - Best practice is to encapsulate as much as possible by using *private*
  - Provide public methods for external access where necessary e.g. getName, setName
- The ideal class is immutable
  - everything set in the constructor
  - none of its values may be changed (i.e. no setters, only getters)
  - immutable values are easier to track, debug and pass around

Visibility modifiers (also known as access modifiers) determine "where" an attribute or method can be accessed. "where" is more or less defined *lexically* meaning you can point to it in the file – it has nothing to do with running code, but written code. You could literally print the code out and draw a circle around where something was visible or not.

Visibility modifiers both apply to methods and attributes. You can make everything public (the default) so that it can be accessed on instanced objects by whatever scope they are instanced in. Or private, fully internal to the object, accessible only in its methods.

The final kind of visibility is "protected". Since visibility is *lexical*, its about bits of code, labelling a method "private" means that it can only be used within the body ( {}) of that class statement. However, inheritance complicates issues because there are two (at least) class definitions which pertain to an object (a Dog is also an Animal). The protected visibility says that anything defined on the parent class can also be accessed by children classes.

Setting a method private in a parent class restricts that method to that parent class which may be convenient. If Teacher extends Employee, then the Employee class may keep its internal IDs to itself while allowing Teacher to inherit only a getEmployeeID() method.



## MIXING VISIBILITIES

```
class Account {  
    private $balance;  
  
    public function __construct($amt) {  
        $this->balance = $amt;  
    }  
    public function getBalance() {  
        return $this->balance;  
    }  
    private function setBalance($amt) {  
        $this->balance = $amt;  
    }  
}  
  
$myAccount = new Account(100);  
echo $myAccount->getBalance();  
echo $myAccount->balance; //error  
$myAccount->setBalance(100); //error
```

There is nothing which prevents you having some methods private and others public.

In general attributes should be private. They represent the state of your object, "it's dials" and you tend not to want anything changing their values except for *your* object's methods.

However, you may also want private methods. In the example above the balance can only be changed within the class and it cannot be accessed directly.



## AN IMMUTABLE OBJECT

```
class Account {  
    private $balance;  
  
    public function __construct($amt) {  
        $this->balance = $amt;  
    }  
    public function getBalance() {  
        return $this->balance;  
    }  
    public function withdraw($amt) {  
        $this->balance() -= $amt;  
        return new self($this->getBalance());  
    }  
}  
$myAccount = new Account(100);  
echo $myAccount->getBalance();  
//to change the balance, create a new account  
$myAccount = $myAccount->withdraw(10);
```

Object state is a very difficult design problem. What state should objects have (colour, height, etc.) and how should this state change? (Can a pen just randomly change what colour it is?)

There are two principles which help when designing object state: 1. only let the object's methods change its state. If a pen needs to change its colour then preferably have a `changeColour()` method which, at least, checks the colour you've chosen is sensible. In web development terms, suppose you have a `User` object and you wish to change the email address, preferably have a method `updateEmailAddress($newAddress)` which validates `$newAddress`.

Principle 2, is to have wherever possible, "immutable" objects. That is to say, objects which do not change their state. These are then very predictable and do not cause debugging problems later on.

Immutable objects set their data in their constructors (they are initialized to a particular state) but then always maintain this state. They can, and often do however, provide `get()` method to let outsiders see their state.

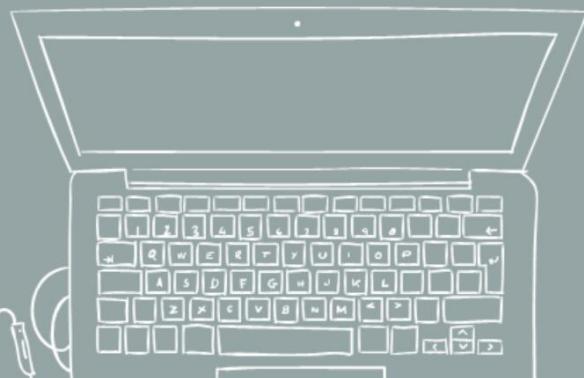
Note: This slide is here to illustrate some design considerations surrounding OO programming. These considerations may not be very intuitive at the moment, but are here to illustrate the kinds of thought which may go into program design.



# Any questions?



## EXERCISE 20



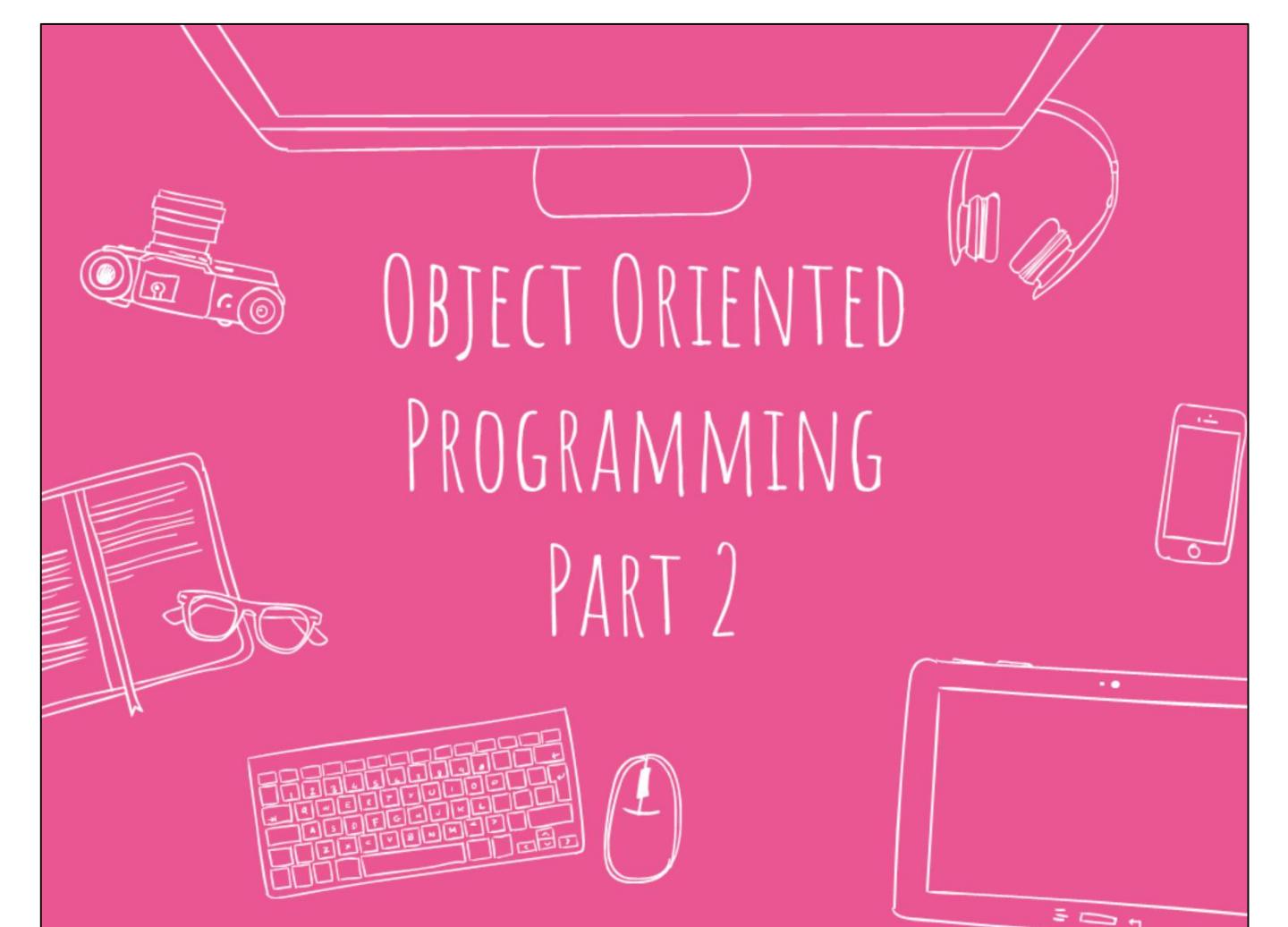
## RESOURCES

Classes and Objects

<http://php.net/manual/en/language.oop5.php>

Visibility

<http://php.net/manual/en/language.oop5.visibility.php>



# OBJECT ORIENTED PROGRAMMING

## PART 2

You are able to completely design an object oriented program with everything you have covered so far. However there are some problems which you would encounter: how do you share methods across objects? How do you tie together namespaces and classes?

Perhaps, indeed: is there are a more elegant way of dealing with errors?

There are good answers to all of these questions and this chapter will present them.

This chapter focusses on interfaces, abstracts and traits.

## CHAPTER OVERVIEW

- Interfaces
- Abstract Classes
- Traits
- Namespaces
- Autoloading



## GETTING TO THE PARENT

- When the same method occurs in the child and the parent, the child may wish to call the parent's version
  - however `$this->method()` would be a recursive call to itself!
- PHP provides the **parent** keyword which always refers to the parent's *instance* members

```
class Super {  
    private $balance;  
    public function __construct($balance) {  
        $this->balance;  
    }  
}  
//subclasses often need to construct their superclass  
class Sub extends Super {  
    public function __construct($pocketMoney) {  
        parent::__construct($pocketMoney);  
    }  
}
```

One of the first problems the budding-OO developer encounters is how to "correctly" substitute children for parents.

Recall that you can pass a Dog object whenever an Animal is required. Suppose an Animal defines a walk() method and Dog defines a walk() method – the object will call the Dog's method, because that is the first class in the inheritance chain. However what if Dog's walk() method wants to call Animal's ?

It thus far has no means of accessing the parent's walk(), since \$this refers to the current object and the current object is a Dog. Here PHP introduces a new keyword `parent` , which refers to the current object (just like \$this) however only the "parental" parts of \$this: \$this->walk() is Dog's walk, parent::walk() is Animal's walk.

This is most often used in constructors. A child constructor will very often need to construct a parent -- when you construct Dog you do not automatically construct Animal ... Dog itself has to do this with parent::\_\_construct().

Note: The `parent::` syntax suggests that parent is static, or there's something static going on (since that's when PHP uses :: ) however this is just a quirk. `parent` really is the object-instance.



## ASIDE: GETTING TO THE PARENT (STATIC)

- **self** refers to the *current class* statically
- **static** refers to the *parent class* statically

```
class A {  
    public static function who() {  
        echo 'A';  
    }  
    public static function test() {  
        static::who();  
    }  
}  
class B extends A {  
    public static function who() {  
        echo 'B';  
    }  
}  
B::test();
```

B

It is possible for classes to contain only static methods, and indeed for other classes to extend from these and only contain static methods too. (This is a red-flag as far as application design goes).

The `static` keyword within methods refers to the parent *class*.

Therefore, in keyword terms, there is:

**self** – the current **class**

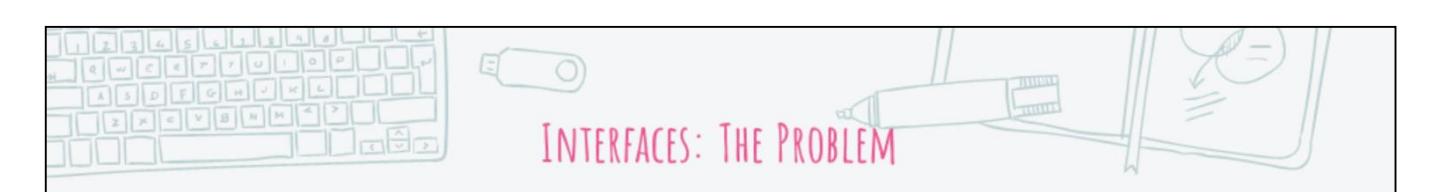
**\$this** – the current **object** instance

**parent** – the current **object** instance's parental methods/attributes

**static** – the parent **class**

Note: To be able to work out what class is a parent of another *statically* PHP performs "late static binding" a feature added in PHP5.3. This feature is widely regarded ... "poorly" to say the least. It is better to ignore this use of the static keyword.

Inheritance is one of the most problematic aspects of object orientation and combining it with statics (that imply global state) is extremely discouraged. Therefore this feature isn't often seen.



## INTERFACES: THE PROBLEM

- Without a type declaration you could not be sure that \$message provides certain methods or properties

```
function write($message) {  
    echo $message->asHtml();  
}
```

- Does \$message have an asHtml method?
- An option would be to require a specific kind of base class for \$message to derive from e.g. Htmlable

```
function write(Htmlable $message) {  
    echo $message->asHtml();  
}
```

- This works, however recall that derived classes (subclasses / children) may have only one base class (superclass / parent) and therefore this is probably too restrictive

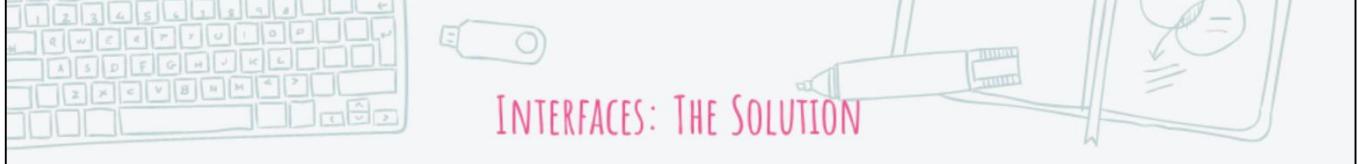
Recall that an object may belong to multiple classes, however there is a restriction. Each class it belongs to must inherit from the other so that a Poodle is a Dog is an Animal.

However this seems a peculiar restriction. A person may be a sister and an employee, but it would seem odd to have "sister extends employee" – there is no relationship between these types.

The problem is that inheritance isn't only a type relationship, it doesn't merely say "this object is both X and Y" it also says "whatever Y can do, X can do". If animal can walk() then so can dog(). Which makes inheritance a little over-powered, it spreads methods around too liberally. If you could inherit from lots of classes then your class would be full of disparate functionality: class Person extends Sister extends Employee extends PhotographyEnthusiast extends NovelReader!

Nevertheless to design programs you will require the ability to say "A person can take a photograph": the Person object has the photograph() ability.

Indeed, in function signatures you will wish to say, takePhoto(SomethingThatCanTakeAPhoto \$x)



## INTERFACES: THE SOLUTION

- Interfaces define a set of methods a class must provide
- Interfaces cannot be instantiated

```
interface Htmlable {  
    public function asHtml();  
}  
class Message implements Htmlable {  
    public function asHtml() {  
        return '<p>Hello!</p>';  
    }  
}  
//some other class file  
function write(Htmlable $message) {  
    echo $message->asHtml();  
}  
write(new Message());
```

<p>Hello!</p>

Interfaces to the rescue! They define what an object can do without telling it how to do it.

Thus if you have a function above, such as `write()` it can require, via a type declaration, that its argument have an `asHtml()` method.

It requires an interface `Htmlable`, which defines the signature of this method. And then specific classes can "implement" this interface – and classes may implement as many interfaces as they wish.

So the `Message` object belongs to the class `Message` *and* `Htmlable`, so it conforms to the type requirement the `write()` function specifies.

Interfaces are purely "contractual" constructs. They cannot specify any functionality.

When defining the class you specify which interfaces it implements.



## MULTIPLE INTERFACES

A class may implement as many interfaces as it wants

```
interface Htmlable {
    public function asHtml();
}

interface Jsonable {
    public function asJson();
}

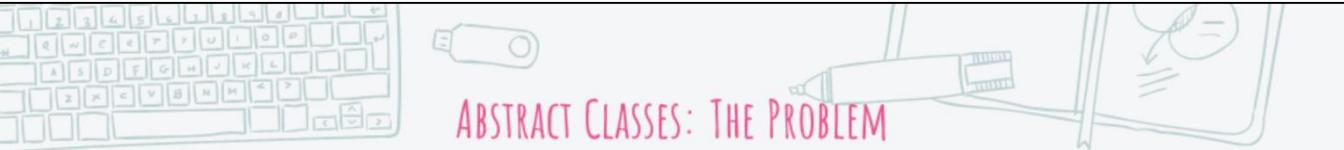
interface Pdfable {
    public function asPdf();
}

class Message implements Htmlable, Jsonable, Pdfable {
    public function asHtml() {
        return '<p>Hello!</p>';
    }
    //ERROR! - PHP requires Message to also provide
    //asJson, asPdf
}
```

Here a Message object can be used wherever an Htmlable or Jsonable or Pdfable is required.

Though it fails to define asJson() and asPdf() which is a Fatal error. Implementing an interface is a commitment to fulfilling a contract, the implementing class must conform exactly to the interface.

Note: Interfaces can also extend from one another, though this is rarely seen.



## ABSTRACT CLASSES: THE PROBLEM

Sometimes superclasses require subclasses to implement specific methods without themselves being able to do so

```
class Animal { //how does the animal move?  
    public function move($location) {}  
    public function getFood() {  
        $this->move("kitchen");  
    }  
}  
  
class Dog extends Animal {  
    public function move($location) {  
        echo "dog runs to $location";  
    }  
}  
  
class Bird extends Animal {  
    public function move($location) {  
        echo "bird flies to $location";  
    }  
}
```

Often parent classes will be defined only to be extended from. Consider for example an Animal class. Will anyone ever want something as abstract as an "Animal"? Probably not: more likely they will want particular kinds of animal.

In web development terms, will you ever wish to record a Person? Or will you want to record an Employee, Staff, Admin, Guest, etc. "Person" is quite an abstract type of object, its unlikely to contain all the information you would like.

In this case you can label a class "abstract", that is "abstract class Person" which prevents it from being instanced (`new'd) – expressing the intention to other developers that only child classes should be used.

There is another problem `abstract` is designed to solve also. In cases where you do not wish to instance the parent, say Animal, perhaps you will expect all children to nevertheless have the same method (e.g. move()) – this requirement can be imposed with `abstract`....



## ABSTRACT CLASSES: THE SOLUTION

- The **abstract** keyword on functions requires the derived class to implement the functionality just like interface implementation

```
abstract class Animal {  
    abstract public function move($location);  
    public function getFood() {  
        $this->move("kitchen");  
    } }  
class Dog extends Animal {  
    public function move($location) {  
        echo "dog runs to $location";  
    } }  
$dog = new Dog();  
$dog->getFood();
```

- A class with abstract members must be declared abstract
- Abstract classes cannot be instantiated
- All classes including abstract classes can only have one base class

If you declare a method abstract then a child must implement it when extending from the class. You must also label the class itself abstract (and of course, how could you instance a class with one of its methods missing?).

Abstract classes can be used to provide common behaviour to all children, but sometimes they need to refer to the particular way child classes do things. As in the example above, `getFood()` is common to all animals, but each moves differently.

Here `move()` is used in `getFood()` but declared abstract, then `Dog` provides the actual implementation.

When calling `getFood()` on a `Dog` object, `getFood()`'s `$this` refers to the `Dog` object which has a `move!`

Note, abstract classes are "full classes" in the sense they occupy the single inheritance slot.

Note: Abstract classes can also be used like "super interfaces" to provide requirements: abstract functions \*and\* some default implementations.

## TRAITS: THE PROBLEM

How do you provide independent functionality to classes without "using up" their parent slot?

- You cannot use interfaces to provide *functionality*
- You cannot use multiple abstract classes
- You can put instances of other classes as private attributes e.g. \$this->htmlable

```
class Htmlable {  
    public function asHtml($msg) { echo "<p>$msg</p>";  
}  
class Plainable {  
    public function asPlain($msg) { echo $msg, PHP_EOL; }  
}  
class Message {  
    public function say($type) {  
        $type == 'html' ?  
            $this->htmlable->asHtml('hello') :  
            $this->plainable->asPlain('hello');//...
```

A recurring problem in many object-oriented languages is sharing functionality across objects without inheritance. Inheritance hierarchies often end up complex, messy and inflexible (what happens if you make a mistake, Rectangle -> Square -> Shape – this isn't easy to fix 1000s of lines later...). (Design consideration such as these may not be obvious to you until many years of practiced development.)

One approach, known as composition, is to store objects as private attributes and use their methods, as in the example above. htmlable and plainable are private variables set in the constructor (omitted). In this way you can access the functionality of several classes quite easily without extending from them. Indeed there is a design principle "prefer composition over inheritance" which recommends this and it's quite a good approach.

PHP however has an excellent additional solution to this problem: traits.



## TRAITS: THE SOLUTION

- Traits define a "snippet" of code that is pasted into a class

```
trait Htmlable {  
    public function asHtml($msg) { echo "<p>$msg</p>";  
}  
  
trait Plainable {  
    public function asPlain($msg) { echo $msg, PHP_EOL; }  
}  
  
class Message {  
    use Htmlable, Plainable;  
}
```

- Traits cannot be instantiated
- You can use multiple traits
- Traits can have no state of their own and can be read as literal code similar to a cut and paste

PHP has traits. Traits are essentially named code snippets that can be slotted into classes without much trouble: and as many as you like. As snippets, they're not classes... they cannot be new`d.

The `use` statement in the example above "pastes" the code in Htmlable and Plainable into Message, so Message objects have asHtml() and asPlain() methods.

There is a style of programming which places as much functionality in small, modular trait-units and has very thin classes which `use` multiple of these traits. As in the example above, you could have class message be one line with all of its methods coming from traits.

The motivation for this style is two fold: traits are very easy to test. You do not need to instance an entire class that may require lots of \_\_constructor parameters, but `use` one trait at a time and test only that trait. And secondly it makes sharing functionality across classes very easy.

Note: Consider the meaning of the word "trait", as in, a characteristic. `use`ing a trait means having a characteristic.

Other languages have "mixins" (e.g. python, ruby) which are similar.

## NAMESPACES

- (Abstract) Classes, Interfaces, Traits all namespace

```
namespace QA\People {  
    class Employee {  
        function __construct() { echo 'QA'; }  
    }  
}  
  
namespace {  
    new QA\People\Employee;  
}
```

QA

- You can use their full path in **use** statements directly

```
namespace {  
    use QA\People\Employee;  
    new Employee;  
}
```

QA

Classes, interfaces and traits all fit in namespaces in the same manner as did the functions. Their name forms the last part of the namespace path.

However, unlike functions or consts you do not have to use a `function` or `const` keyword when you `use` them.

Recall that if you specify the full path of an identifier when you `use` it, you remove its entire namespace prefix.

In the above example, by specifying the full namespace path in the use statement the class Example is named only Example.

## AUToloading AND NAMESPACES

- The real power of namespaces comes with autoloading
  - when **newing** a class, if PHP cannot find it, PHP will look for an autoloader
- An autoloader is a function registered with PHP
  - it is passed the full namespaced name of a class
  - if namespacing matches up to your directory structure the autoloader has an easy job

```
spl_autoload_register(function ($classname) {  
    include "$classname.php";  
});  
//...  
new Some\Long\Path\SpecificClass();  
  
//includes Some\Long\Path\SpecificClass.php !
```

- **No need for include statements**
- **new + autoloading will locate the class**

The real power of namespacing comes when it is conjoined with an autoloading mechanism.

Previously, if you try to refer to a class (either by extend'ing from it, or new'ing it) you must include the file it is defined in otherwise you will get a Fatal error. If however you register an autoloading function PHP will call this function if it detects a classname is used without being defined. It will pass the fullname of the class to the autoloader (i.e. including the namespace).

Thus if you register an autoloader, and if your namespaces match up to your directory structure, you need include no class files. PHP will call the autoloader whenever it finds a missing class. If your autoloader fails, you will still get a Fatal error.

This technique is extremely common in object-oriented PHP development, and allows you to write code freely without worrying which files have or have not been included. The only constraint is that your project directory structure relates to your namespace structure. However that is a sensible requirement anyway.

Note: Your directory structure need not be an exact match, the autoloader registers a function which can include whatever it likes.



## ‘USE` STATEMENTS

- 
- For namespaces **use** can be coupled with an **as** clause to create aliases for classes, traits and interfaces
  - Useful when two class names would conflict with each if un-prefixed

```
namespace Blog\Html {  
    class Page {}  
}
```

```
namespace Blog\Pdf {  
    class Page {}  
}
```

```
namespace {  
    use Blog\Html\Page as HtmlPage;  
    use Blog\Pdf\Page as PdfPage;  
}
```

Recall that if you `use` the fullpath of a class you will drop its entire namespace prefix. This however may cause clashes.

The `as` keyword may be added to a `use` statement to alias the identifier you are `use`ing.

NB.

This is also possible when `use`ing traits within classes.



## BACK TO THE GLOBAL



When inside a namespace you need to either **use** an identifier or write the full path from the global namespace e.g. \Exception.

This includes predefined PHP classes, such as Exception

```
namespace Mine;
```

```
use Exception;
```

```
throw new Exception; //full path not needed
```

```
namespace Mine;
```

```
throw new \Exception; //full path needed as no 'use'
```

The full path of a namespaced identifier begins with a leading back slash (\) however, somewhat confusingly, this is often omitted.

This can only be omitted if you are in the global namespace (i.e. you haven't declared one of your own) or in a 'use' statement.

Therefore if you have declared a namespace, the predefined Exception class is \Exception. Without the leading slash PHP will assume there is a class defined in your namespace called Exception.

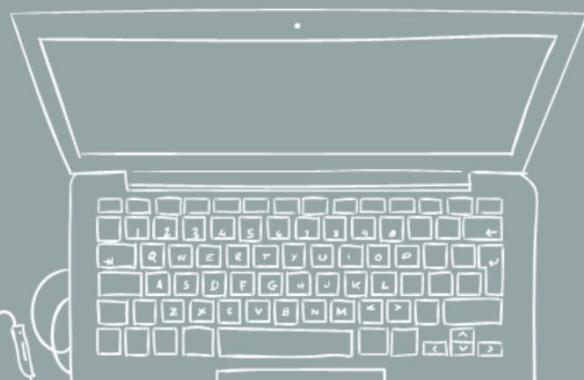
It is always a good idea to 'use' classes at the top of your PHP files, especially predefined classes. This alerts the reader of your code to which classes the file uses, and makes writing the code from that point onwards simpler.



# Any questions?



## EXERCISE 21



# ERRORS AND EXCEPTIONS

## CHAPTER OVERVIEW

- Errors generated
- Error handling
- Exceptions

## COMMON MISTAKES AND ERRORS

- When first starting out in PHP it is easy to make rudimentary errors
  - Knowing how to deal with them can save a lot of time
- The browser is showing the PHP code and not the expected page
  - This is because the page has not been served
    - Check your address bar, if it does not start with `localhost` you have most likely double clicked the file from file explorer
    - To remedy, open your page using the local server (if you are in XAMPP it is the path from `htdocs` including the file name)

This is not an error on the part of the coding at this stage, more the way that the user has asked the browser to display the page. As PHP is a *server-side scripting language* any pages that contain PHP code have to be processed and returned by a server to display correctly. For this reason, it is necessary to have a local server running when writing and testing code out of a real development environment (such as XAMPP). In a real development environment, there is usually some form of development server that allows teams of developers to work on sites as though they were live, without risking the integrity of the actual live site.

## PHP ERROR MESSAGES

### 5 types of error message

- Each displays name of the file and the line number of where the error was encountered

#### 1. NOTICE

- PHP sees a condition that may or may not be an error

#### 2. WARNING

- PHP sees a problem but it is not serious enough to prevent the code from running

#### 3. PARSE ERROR

- A syntax error that PHP has found before scanning the code

#### 4. FATAL ERROR

- A serious error that stops the execution of the rest of the code

#### 5. STRICT

- Warnings about coding standards presented when poor coding practice is found or better code can replace it

These are the errors that PHP code can produce. They are more fully explained in the following pages. The errors generally stop a PHP page from being displayed in the intended way.



## NOTICES

- Notices are generated when something is wrong with the execution of the code but it does not cause a fatal program error
- Consider the following code:

```
1 | <?php
2 |     $numbers = [1, x, 5];
3 |     foreach($numbers as $number) {
4 |         echo $number * 2;
5 |         echo "<br />";
6 |     }
7 | ?>
```

- It generates the following page:

**Notice:** Use of undefined constant x - assumed 'x'  
in /Applications/XAMPP/xamppfiles/htdocs/test.php on line 2

2  
0  
10



- The code can still be executed but the result is likely to be incorrect

In this example, it assumes x as a constant that has a value of zero. It then uses this to calculate the value in the foreach loop, allowing the page to be displayed.

Using the `error_reporting(0);` function call at the start of PHP code will suppress these **Notices** from appearing but it can hinder debugging and error reporting. Different ways to deal with this will be covered later in this chapter.



## WARNINGS

- We have seen that undefined variables do not necessarily cause errors
- Warning errors are generated when code cannot execute

```
1 | <?php  
2 |     $numbers = [1, x, 5];  
3 |     foreach($somethingElse as $number) {  
4 |         echo $number * 2;  
5 |         echo "<br />";  
6 |     }  
7 | ?>
```

- The above code generates the following messages:

**Notice:** Undefined variable: somethingElse  
in /Applications/XAMPP/xamppfiles/htdocs/test.php on line 3



**Warning:** Invalid argument supplied for foreach()  
in /Applications/XAMPP/xamppfiles/htdocs/test.php on line 3



The fact that we have an undefined variable is not an error in itself and generates a notice. The issue here is that the foreach loop cannot execute because it does not have valid data to loop over. Inspecting line 3 shows that the variable \$somethingElse has been used in the foreach loop but it is not able to be used in this context. Replacing the \$somethingElse with the valid \$numbers argument clears this warning.

Again, these errors can be hidden from the browser using an `error_reporting(0);` call at the start of the code.

## PARSE ERROR

- This means that there is an error in your PHP code
  - Read what the rest of the message says
  - It will give you an idea of what the error is and where it is in the code
- Consider the following code:

```
1 | <?php  
2 |     echo "Hello world!"  
3 |     echo "<br />";  
4 | ?>
```

- Serving this page from localhost produces the following error message:

**Parse error:** syntax error, unexpected 'echo' (T\_ECHO), expecting ',' or ';' in **/Applications/XAMPP/xamppfiles/htdocs/test.php** on line **3**

**( ! ) Parse error:**

The error message is stating that it has been unable to parse the page into HTML and suggests that there is a missing ; near line 3 – re-examining the code above reveals that there is indeed a missing ; at the end of line 2. Correcting this, resaving and reloading clears the error. These errors are detected during the processing of the PHP code on the server and returned to the user when found.

In the error message, T\_ECHO is a parser token. These represent parts of the PHP language and usually relate to native PHP functions (e.g. T\_IF). A list of parser tokens can be found at:  
<http://www.php.net/manual/en/tokens.php>

Other errors can be cleared in a similar way using the information supplied.

## FATAL ERROR

- A fatal error means that there is a serious error in the PHP code and it cannot continue to execute
  - Common causes are undefined functions
- Consider the following code:

```
1 | <?php  
2 |     echo "Start of code <br />";  
3 |     $x = multiply(2, 5);  
4 |     echo "Mulitplying 2 and 5 gives $x";  
5 | ?>
```

- Serving this page from localhost produces the following error message:

( ! ) Fatal error:

**Fatal error:** Uncaught Error: Call to undefined function multiply() in /Applications/XAMPP/xamppfiles/htdocs/test.php:3 Stack trace: #0 {main} thrown in **/Applications/XAMPP/xamppfiles/htdocs/test.php** on line 3

The error message is stating there is a call to an undefined function multiply(). This means that line 3 of the code uses a function that PHP has not seen before. To use the multiply function it must exist in the PHP code somewhere.

Fatal errors occur when the code cannot continue to execute. The error message generated gives as much information as possible to help the developer identify what has caused the error.

The term *fatal error* does not appear in the PHP manual.

## STRICT

- Warn about coding standards
  - Highlight poor coding practice or language that has been replaced by better code
- Does not stop the execution of the code
  - Changing code to eliminate these warnings helps make code more reliable
- Some STRICT messages are about language features that have been deprecated
  - Old functions that have been replaced during the development of the different versions of the language
  - Most are still supported in later versions but could be removed in future

## PHP'S ERRORS

- PHP has three main error levels
  - **E\_NOTICE** – An error
  - **E\_WARNING** – A critical but non-fatal error
  - **E\_ERROR** – A fatal error
- Depending on which level **error\_reporting** (php.ini) is set to, the engine will raise or suppress errors
  - E\_ALL is the level for all errors
  - the `error_reporting()` function sets the configuration value

```
error_reporting(E_ALL);
```

- On older PHP applications E\_NOTICES may be suppressed by lowering the `error_reporting` level

```
error_reporting(E_ALL & ~E_NOTICE);
```

## BASIC ERROR HANDLING – DIE()

- Whilst understanding error messages is important, handling them is more so
- Error handling tells the program what to do if it encounters an error
- In PHP the most basic way is to tell the code to stop executing
  - Achieved by using the **die()** function
- Consider:

```
1 | <?php  
2 |     $file=fopen("someText.txt", "r");  
3 | ?>
```

- If the file someText.txt cannot be found this would generate the following error:  
Warning: fopen(someText.txt): failed to open stream: No such file or directory in/Applications/XAMPP/xamppfiles/htdocs/test.php on line 2





## BASIC ERROR HANDLING – DIE()

- Change the code so that it reads:

```
1 | <?php  
2 |     if(file_exists("someText.txt")) {  
3 |         $file=fopen("someText.txt", "r");  
4 |     } else {  
5 |         die("File not found");  
6 |     }  
7 | ?>
```

- In this instance, the error is handled and the message displayed now is:

File not found

- Any message could have been displayed here

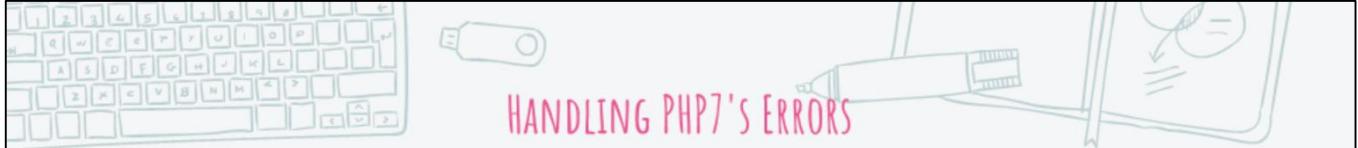
The die() function stops the further execution of any code, including any HTML that may follow. It basically tells the server to stop what it is processing and return whatever it needs to at that point to the browser.

The die() function without any arguments can be used to stop the code executing after it has done some other lines of code:

```
<?php  
    if(file_exists("someText.txt")) {  
        $file=fopen("someText.txt", "r");  
    } else {  
        echo "<script>alert('File not found');</script>";  
        die();  
        echo "Do not display this!"; // Will not be displayed  
    }  
?>
```

## PHP7'S ERRORS

- Most errors in PHP7 are now **Exceptions**
- Errors
  - ArithmeticError
  - DivisionByZeroError
  - AssertionError
  - ParseError
  - TypeError
- Exceptions are error which are available at run time as objects
  - They contain the line causing the error
  - The error message
  - The execution history of the application to that point
- Uncaught (unhandled) exceptions are all E\_ERROR (i.e. fatal)



## HANDLING PHP7'S ERRORS

Wrap any code which may cause an error in a **try** block

- Include a **catch** block for `Error \$e`

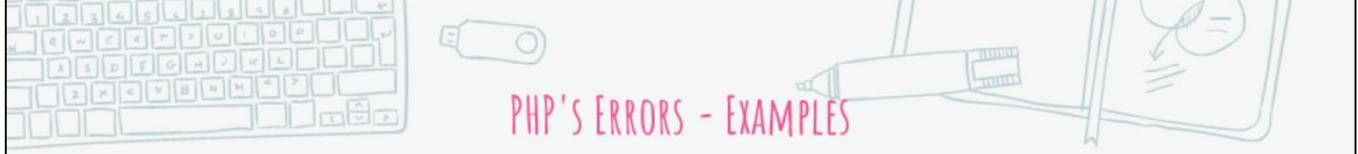
```
try {  
    undefined();  
} catch (Error $e) {  
    echo "Handling the error";  
}
```

Handling the error

- Inside the catch block you can also get the error's line and message

```
catch (Error $e) {  
    echo $e->getMessage(), ' @ ', $e->getLine();  
}
```

The try and catch blocks are explained in more detail later in this chapter.



## PHP'S ERRORS - EXAMPLES

The `trigger_error()` function will raise PHP Errors

- First argument is the error message
- Second argument is a predefined constant representing the error level

```
echo $undefined; //E_NOTICE
```

```
echo undefined(); //E_WARNING or Error Exception
```

```
7 / 0; //E_WARNING or DivisionByZeroError
```

```
trigger_error("Hello"); //E_NOTICE  
trigger_error("Hello", E_USER_WARNING); //E_USER_WARNING  
trigger_error("Hello", E_USER_NOTICE); //E_USER_NOTICE  
trigger_error("Hello", E_USER_ERROR); //E_USER_ERROR
```

It is useful to be able to trigger errors when an illegal input occurs. Errors can be triggered anywhere in the code and with the use of a second parameter you can specify what error level is triggered.

Possible error types:

- `E_USER_WARNING` – Non-fatal user-generated runtime warning where the execution of the code does not stop.
- `E_USER_NOTICE` – The default and a user-generated runtime notice. There might or might not be an error.
- `E_USER_ERROR` – Fatal user-generated runtime error that cannot be recovered from and the execution of the code stops.

## ERROR LOGS

- Use the **ini\_set()** function to modify php.ini configuration variables
  - display\_errors – whether to output errors to the console or browser
  - **ini\_set('display\_errors', 'off')** – send them to the log only
  - **error\_log()** – location of PHP error log
- Using the **error\_log()** function allows errors to be sent to a specified file or other destination (such as an email address)

By default, errors are sent to the server's logging system or file and this is set in the php.ini file.

The **error\_log()** function can be used to change the destination of the errors to log. This can be a file that the developer has specified or a remote destination, such as an email address. The **error\_log()** function itself has the following description (from [php.net/manual/en/function.error-log.php](http://php.net/manual/en/function.error-log.php)):

```
bool error_log ( string $message [, int $message_type = 0 [,string $destination [, string $extra_headers ]]] )
```

The **message** argument is the error message that should be logged.

The **message\_type** argument specifies the type of error and where the error should go (the possible types can be seen at [php.net/manual/en/function.error-log.php](http://php.net/manual/en/function.error-log.php)).

Within the **message\_type** is the destination. This is determined by the message type.

The **extra\_headers** is used when the **message\_type** parameter is set to 1 (email).

This function returns TRUE on success and FALSE on failure.

## CUSTOM ERROR HANDLER

- Many errors can be handled globally by a custom error handler
  - **set\_exception\_handler()** – PHP7-style & object oriented exceptions
  - **set\_error\_handler()** – notices, fatals, pre-7 errors
- Both functions take a callback
  - for set\_exception\_handler the call-back should accept one argument: the exception value
- The @ operator can be applied to expressions which produce errors
  - it will silence those errors and is therefore widely regarded as bad practice

```
set_exception_handler(function ($exception) {  
    echo '<h1> ERROR! </h1>'; //custom HTML page  
    echo '<h2>', $exception->getMessage(), '</h2>';  
});  
  
undefined();
```

The error suppression operator:

```
$value = @$cache[$key];
```

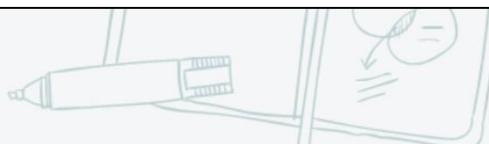
Would suppress a NOTICE error if \$key didn't exist in the array.

Better to write explicitly,

```
if(exists($cache[$key])) {  
}
```

Always handle an error, ignoring it will cause greater problems in the long run!

# PHP ERRORS



Constant	Description
E_ERROR	Fatal run-time errors. These indicate errors that can not be recovered from, such as a memory allocation problem. Execution of the script is halted.
E_WARNING	Run-time warnings (non-fatal errors). Execution of the script is not halted.
E_PARSE	Compile-time parse errors. Parse errors should only be generated by the parser.
E_NOTICE	Run-time notices. Indicate that the script encountered something that could indicate an error, but could also happen in the normal course of running a script.
E_CORE_ERROR	Fatal errors that occur during PHP's initial startup. This is like an E_ERROR, except it is generated by the core of PHP.
E_CORE_WARNING	Warnings (non-fatal errors) that occur during PHP's initial startup. This is like an E_WARNING, except it is generated by the core of PHP.
E_COMPILE_ERROR	Fatal compile-time errors. This is like an E_ERROR, except it is generated by the Zend Scripting Engine.
E_COMPILE_WARNING	Compile-time warnings (non-fatal errors). This is like an E_WARNING, except it is generated by the Zend Scripting Engine.
E_USER_ERROR	User-generated error message. This is like an E_ERROR, except it is generated in PHP code by using the PHP function <a href="#">trigger_error()</a> .
E_USER_WARNING	User-generated warning message. This is like an E_WARNING, except it is generated in PHP code by using the PHP function <a href="#">trigger_error()</a> .
E_USER_NOTICE	User-generated notice message. This is like an E_NOTICE, except it is generated in PHP code by using the PHP function <a href="#">trigger_error()</a> .
E_STRICT	Enable to have PHP suggest changes to your code which will ensure the best interoperability and forward compatibility of your code.
E_RECOVERABLE_ERROR	Catchable fatal error. It indicates that a probably dangerous error occurred, but did not leave the Engine in an unstable state. If the error is not caught by a user defined handle (see also <a href="#">set_error_handler()</a> ), the application aborts as it was an E_ERROR.
E_DEPRECATED	Run-time notices. Enable this to receive warnings about code that will not work in future versions.
E_USER_DEPRECATED	User-generated warning message. This is like an E_DEPRECATED, except it is generated in PHP code by using the PHP function <a href="#">trigger_error()</a> .
E_ALL	All errors and warnings, as supported, except of level E_STRICT prior to PHP 5.4.0.

## EXCEPTIONS: THE PROBLEM

- Procedural-style error handling is very invasive i.e. **Trigger\_error()**
  - It is difficult to stop errors from causing the whole application to stop
  - It is difficult to handle errors appropriately and specifically
- The types of errors are very limited
  - Information about errors is very limited
- Errors are *events* not values
  - Errors "happen" to the application
  - You cannot inspect them, like values, as the application is running

Many programmers think of errors as events. They are something that happens *to* the program but are not part of the program itself. However this view has some problems: in the end, the program itself needs some way to handle this event.

Procedural and evented programs have "recovery" steps, some code after the error-event which ensures, for example, the file has been saved.

Object oriented programs managed to make the events themselves available to the programmer.



## EXCEPTIONS: THE SOLUTION

PHP provides a family of classes called **Exceptions**

- These classes can be instantiated
- Can also *throw* objects of these classes

```
function causeError() {  
    throw new Exception("Message");  
}  
causeError();  
  
Fatal error: Uncaught Exception: Message  
    in C:\...\23-00.ii\CauseError.php:4  
  
Stack trace:  
#0 C:\...\23-00.ii\CauseError.php(7): causeError()  
#1 {main}  
    thrown in C:\...\23-00.ii\CauseError.php on line 4
```

PHP has a defined class called `Exception` which can be used in conjunction with the `'throw'` operator to create an error event.

`'throw'` is passed an *object* which represents the kind of error in question. When the program encounters the `'throw'` statement an error is raised.



## CATCHING EXCEPTIONS

- Exception may be "caught" and prevented from causing an Error

```
function causeError() {  
    throw new Exception("Message");  
}  
  
try {  
    causeError();  
} catch(Exception $e) {  
    echo $e->getMessage();  
}
```

Message

- An uncaught exception that has been thrown becomes a Fatal Error

To manage this error event behaviour PHP has a `try {}` block. Any code within this block which `throws` an exception object will be monitored, and if an error occurs the exception object will be captured.

PHP then looks for a corresponding catch {} block, if it finds one, it will pass the exception object to this catch block which can handle the error in the normal flow of the program.

The call stack, the sequence of function calls which led to the present location, will unwind until a catch {} block is found.

Look at the causeError() function: it is called in the global namespace where there's a catch block{}. Thus, throw causes the program to jump from the `throw` to the catch block.

In this sense `throw` is even more powerful than `return` as it will jump out of a function until it hits a catch. And if no catch is found there will be a Fatal error.



## THROWING YOUR OWN



- It is possible to extend from PHP's base Exception class

```
class MyException extends Exception {  
  
}  
  
function broken() { throw new MyException(); }  
  
try {  
    broken();  
} catch(MyException $e) {  
    //do nothing  
}
```

This is particularly useful with multiple catch clauses

You can throw your own exception objects, just extend from the Exception class.

Typically custom exceptions will not have any particular code in them, but are defined so that you can throw a more specialized kind of object with a more specialized type.

For example: `throw new CannotFindPetStoreException()` is clearer than `throw new Exception()`.



## MULTIPLE CATCH CLAUSES

With multiple catch clauses the first matching one is executed

```
function causeError() {  
    throw new MyException("Message");  
}  
try {  
    causeError();  
} catch(SomeOtherException $e) { //derived  
    die("NOPE!");  
} catch(MyException $me) { //derived  
    echo $e->getMessage();  
} catch(Exception $e) { //base  
    die("NOPE!");  
}
```

Message

Combined with a hierarchy of Exception types this is a powerful way to handle specific errors in specific ways or if you cannot, fall through to the generic catch Exception \$e)

Specialized kinds of exceptions (that is, ones which inherit from Exception) are useful with multiple catch clauses.

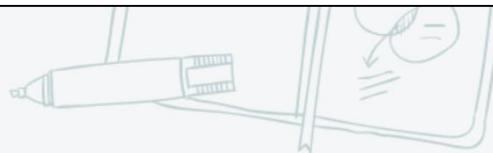
PHP will enter the first catch clause whose type argument matches the type of exception thrown.

In the above example, a MyException object is thrown. The first catch clause specifies SomeOtherException, so it doesn't match. The MyException clause does however, and is executed. The final catch is not.

Note also that in the body of the catch clauses the exception object is itself made available and has methods such as getMessage() which maybe useful. The \$e argument can be named anything, \$e however is quite conventional.



FINALLY



```
function causeError() {  
    throw new MyException("Message");  
}  
  
try {  
    causeError();  
} catch(SomeOtherException $e) { //derived  
    die("NOPE!");  
} finally { //PHP5.5 onwards  
    die("ALWAYS!");  
}
```

## ALWAYS

- A finally block will always be executed – even if no matching catch() was found
- Useful for closing resources (e.g. files, database connections)

Optionally a finally clause can be included. The body of a finally clause will always be executed (if there is an error).

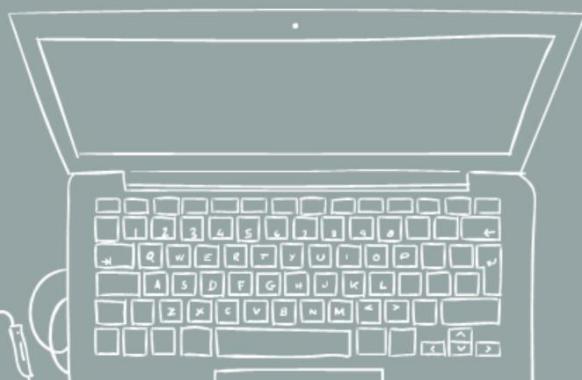
This is especially useful for closing file or database connections or memory intensive resources.



# Any questions?



## EXERCISE 22



## RESOURCES

Trigger Error

<http://php.net/manual/en/function.trigger-error.php>

Error Control Operators

<http://php.net/manual/en/language.operators.errorcontrol.php>

<http://php.net/manual/en/function.set-error-handler.php>

Error Logging

<http://php.net/manual/en/function.error-log.php>

# PHP DATA OBJECTS (PDO)

You have seen how arrays in PHP can be used to represent collections of data. And you have seen how databases and SQL can be used to store and manage data. The task of a PHP database library is to connect the two: bringing together PHP's representation of data collections with SQL's.

The standard contemporary library for managing databases in PHP is known as PDO or PHP Data Objects.

PDO is an object oriented library which abstracts over the particulars of managing a database. It allows you to use the same classes and methods each time you interact with it, regardless of which database you are connecting to: whether that's MySQL or MSSQL.

Using PDO is very routine, you only have to learn a few methods of a couple of classes and you can reapply this knowledge to each new querying problem.

## CHAPTER OVERVIEW

- PDO Classes
- Connecting to a DB
- PDO Iterators and Arrays
- Prepared Statements
- Error Handling

This chapter is designed to make you familiar with the PDO library both conceptually and practically. The new fundamental concepts around PDO are prepared queries and iterators and these will be emphasised.

By the end of this chapter you will have seen a full set of code which can be used to connect to and query a database.

## PHP DATA OBJECTS

- Set of core extensions to PHP that provide
  - A base PDO class that interfaces with a variety of databases
  - and database specific drivers
- The same PDO classes are used regardless of the database vendor
  - promotes code reuse, simplicity and easy abstraction
  - class **PDO** for connection
  - class **PDOStatement** for using queries
- Primary features are:
  - prepared queries and bound parameters
  - support of error and exception handling
- Supported databases
  - MySQL, PostgreSQL, SQLite
  - Oracle, DB2, OCI, MSSQL
  - and others

The PHP language is comprised of many interdependent extensions besides "PHP Core" (its core features). These extensions are usually in the /ext folder of the PHP installation and contain shared object (.so) or dynamic link (.dll) files depending on your operating system.

One of the most routinely used extensions is PDO and particularly PDO MySQL. However PDO supports a variety of databases.

Activating the PDO MySQL extension allows you to use two PDO classes, `PDO` and `PDOStatement`. The former represents the database connection itself and the latter represents particular queries you can issue against this connection.

## CONNECTING TO A DATABASE

- Database connections are initialized as soon as you instantiate the PDO class

```
try {  
    $pdo = new PDO($dsn, $user, $password, $options);  
} catch (PDOException $e) {  
    die($e->getMessage()); //die() for illustration  
                                //always handle errors  
}
```

- The DSN is the connection string:

- mysql:host=name;dbname=dbname
- pgsql:host=name dbname=dbname
- sqlite:/path/to/file

You connect to a database by creating a new instance of the PDO class. The constructor itself creates the connection and will throw an exception if it fails.

The constructor takes four arguments. The first two are the username and password of your database user account. The third parameter is a connection string known as a DSN (datasource name) which includes the type of database, where it is and what it is called.

The DSN always begins with the database type but can vary considerably and it is perfectly acceptable to rely on references to tell you what it should be. The PHP PDO documentation uses mysql as an example connection so it is a valuable reference for typical MySQL DSNs.

## PDO ITERATORS

- Use PDO::query() to SELECT

- Executes the statement and returns a result set which you can iterate through

```
$pdo = new PDO($dsn);  
$stmt = $pdo->query(  
    "SELECT * FROM table",           //query  
    PDO::FETCH_COLUMN,              //data structure  
    0);                            //0th field  
  
foreach ($stmt as $row) {  
    echo $row;  
}  
  
unset($stmt);                  //frees result memory  
                                //not necessary but efficient
```

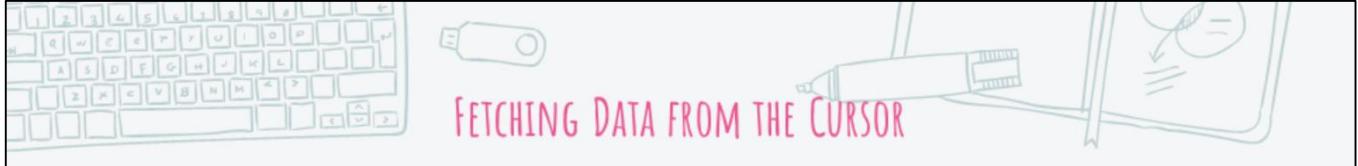
Having created a PDO instance, here called \$pdo, we can issue queries.

The query() method accepts a SQL string, a return type and any further options. The object returned from query() is an iterable PDOStatement, meaning that it can be used in a foreach statement as though it were an associative array.

In the example above a SQL query is run with the PDO::FETCH\_COLUMN "fetch mode" which specifies that the associated array returned should contain only a single database field or column, in this case, column 0.

The important element of this example however is the foreach statement. Note that the object returned from query() is used directly and that on each iteration a row of the single column selected is echo'd, showing the relationship between the "fetch mode" and the data structure returned.

PHP is a garbage collected language and PHP scripts typically run for short amounts of time however sometimes it is worth unsetting the query statement object directly to free any data in memory associated with it. This will be most useful in long running applications.



## FETCHING DATA FROM THE CURSOR

- Database results are not fetched all at once
  - The database points a cursor at the result set and you move the cursor forwards to get results
- You may also use PDO::prepare() to create a prepared statement that can be parameterized
  - Use PDOStatement::execute() to execute

```
$pdo = new PDO($dsn);
$stmt = $pdo->prepare("SELECT * FROM table");
$stmt->execute();

//moving cursor forward, getting a row at a time
while ($row = $stmt->fetch()) {
    print_r($row); //array of columns
}
```

To save on memory the query result is not entirely stored in a single variable at once, but rather the database creates a cursor to a place in its memory store which can be advanced by PDO.

This cursor advances when you loop over a statement or call the fetch() method which provides one row of data at a time. The cursor is forward-only unless otherwise specified meaning that once you read the end of the data (through looping) you will not be able to go back over it.

There is an additional method on the PDO class which produces a PDOStatement object; prepare(). This method takes a SQL statement and "prepares" it for later execution. This could theoretically mean sending it to the database so it can make it more efficient. In practice PDO does not yet take this step.

The PDOStatement object may then be executed at whatever time is appropriate. Note however that the return value of execute() is a Boolean indicating success. You still need to loop over the statement object to retrieve the data.

## CURSORS

- Forward only
  - Known as unbuffered queries
  - Default kind
  - When reading is finished, result set is exhausted so you cannot go over the results again
- PDO::CURSOR\_SCROLL
  - *Scrollable (forwards and backwards)*
- PDO::FETCH\_ORI\_\*
  - *A variety of other specialised cursor "orientations"*

The two main cursor types are represented by the PDO class constants PDO::CURSOR\_FWDONLY and PDO::CURSOR\_SCROLL . The forward only cursor, as discussed, allows you to iterate, in sequence, once over a query result.

PDO::CURSOR\_SCROLL allows you to iterate in sequence in reverse or forward directions and revisit previous records.

The infrequently used PDO::FETCH\_ORI\_\* constants give you fetch modes that work with scrollable cursors, for example, PDO::FETCH\_ORI\_PRIOR will fetch the previous row in a result set if the cursor is scrollable.

## PRE-BUFFERING DATA

- The easiest way to loop over data several times, or to store it for later use, is to use **PDOStatement::fetchAll()**

```
$pdo = new PDO($dsn);  
$stmt = $pdo->query("SELECT * FROM table");  
  
$rows = $stmt->fetchAll();  
  
$numRows = count($rows);  
  
foreach ($rows as $row) {  
    print_r($row);  
}
```

- This loads the entire result set into memory

If you need to iterate over a small amount of data multiple times the easiest solution is to use the `fetchAll()` method on the `PDOStatement`. This runs through the cursor and stores each result in PHP's memory. That is, it assigns the entire result set to an array which `fetchAll()` returns.

Fetch all can then be used as many times as you like in whichever kind of loop suits you. Note that iterables -- objects that work in `foreach` loops -- only work in `foreach` loops).

## FETCHING MODES

When fetching data (`PDOStatement::fetch*()`) you can specify what kind of data structure you would like returned

- `stmt->fetch(PDO::FETCH_BOTH)`
  - Array with numeric and string keys – default
- `PDO::FETCH_NUM`
  - numeric keys only i.e. sequential array
- `PDO::FETCH_ASSOC`
  - string keys only i.e. associative array
- `PDO::FETCH_OBJ`
  - `$obj->name == 'name' column`
- `PDO::FETCH_CLASS`
  - useful with MVC Models (see later)
  - provided class
- `PDO::FETCH_INTO`
  - provided object

PDO can return data in whatever standard form you like. By default it returns both a sequential and an associative array: an array with integer *and* string keys that represent the various fields of your table or query result set.

The most frequent fetch mode used is `PDO::FETCH_ASSOC` which instructs PDO not to insert the integer sequential indexes since most people know the names of the fields they're using and find it clear to write them out.

However there are couple of other interesting modes. `PDO::FETCH_CLASS` creates an instance of a class you supply and populates its fields with data – if you name those fields according to the fields in your result set.

`PDO::FETCH_OBJ` does much the same thing, however you supply a class already-instanced rather than letting it instance a class for you.

The OO-style fetch modes are a little tricky: they may not call the constructor properly.

# MODIFYING DATA

## STATEMENTS WITH DATA RETURNED

- PDO::exec() executes a query without any returned data

```
$pdo->exec("INSERT INTO table VALUES(1)");
```

- It will return the number of modified rows

```
$deleted = $pdo->exec("DELETE FROM table WHERE 1");
```

```
$changes = $pdo->exec(  
"UPDATE table SET field=0 WHERE NAME LIKE '%lock%'");
```

The exec() method runs a query which has no result set and so requires no PDOStatement object. INSERT statements for example, do not return results.

exec() itself returns the number of rows in the database which have been modified.

If you wish to get the last inserted id, the lastInsertId() method on the PDO object will tell you.

## PREPARED STATEMENTS

SQL statements often need populating with user data (e.g. `$_POST` data)

- Inserting this data raw into SQL is dangerous
- Prepared statements create "safe slots" for data to go in

```
$stmt = $pdo->prepare("
    INSERT INTO user(username, email)
        VALUES (:username, :email)");
$stmt->execute([
    'username' => $_GET['username'], //filter_input?
    'email' => 'sh@example.com',
]);
```

These are prefixed with `:` and given as an associative array

Preparing statements before you execute them allows you to separate out data from SQL. You supply `prepare()` with a SQL string which contains placeholders for data. Placeholders are either named with a colon prefix, or they are question marks (?).

If you used named parameters you should call `execute()` with an associative array which provides the value for each parameter with the appropriate key. The keys in the array do not need to be prefixed with colons.

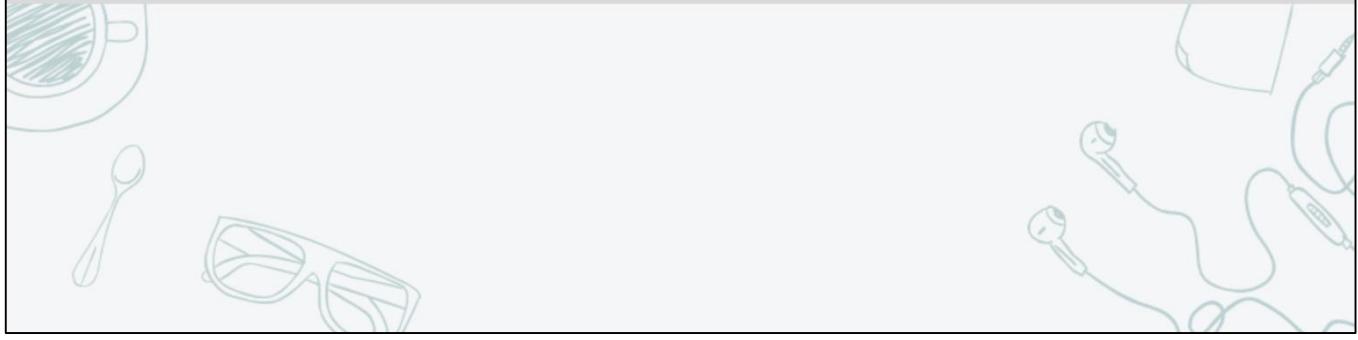
Using this method PDO will never confuse data with SQL and will send any data to the database using the correct escaping rules. If you insert user data (e.g. from `$_GET`) into SQL directly this data may itself contain malicious SQL which interferes with the query you were trying to send (for example., imagine a username called "`-- DROP users;`").



## PREPARED STATEMENT PARAMETERS

Use a named variable prefixed with a colon or use ? as a parameter placeholder and provide a sequential array.

```
$stmt = $pdo->prepare("INSERT INTO user(username, email) VALUES (?, ?)");  
$stmt->execute(['shomes', 'sh@example.com']);
```



If you use question marks (?) rather than named parameters execute() will expect a sequential array and use each element in turn for each question mark in turn.

# ERROR HANDLING

## ERROR HANDLING

Three error modes for PDO:

- silent (default)
- warning
- exception

```
// default  
//pdo functions return statements OR false if error  
if (!$pdo->query($sql)) {  
    $error = $pdo->errorInfo();  
}
```

- `$error[0] == $pdo->errorCode()`
- `$error[1]` is driver specific error code
- `$error[2]` is driver specific error message
- `$error[3]` is error message

By default PDO does not throw exceptions, this is a very strange choice for an object-oriented library and may have been chosen because procedural PHP developers also wanted to use it. The core PHP development team has historically been committed to keeping procedural PHP on a feature-based level playing field with OO.

Under the default regime you have to test whether or not `query()` actually returns a `PDOStatement` – it will be Boolean false if the query has failed. You should then call `errorInfo()` on the PDO object to find out what went wrong.

## EXCEPTION ERROR MODE

```
$pdo->setAttribute(  
    PDO::ATTR_ERRMODE,  
    PDO::ERRMODE_EXCEPTION);  
  
try {  
    $pdo->exec($sql);  
} catch (PDOException $e) {  
    echo $e->getMessage();  
    $error = $e->errorInfo();  
}
```

To configure PDO statements to throw exceptions rather than returning false set the attribute error mode to PDO::ERRMODE\_EXCEPTION

The exception error mode is a little more user friendly. All query functions throw an exception if they fail, so you do not have to check if the query() returns a PDOStatement or false. Instead supply a try/catch block in case it throws an exception.

Exceptions are also much more difficult to avoid by mistake. You do not want to find that a query fails when it comes to deploying it live in a production site.

# PDO CLASSES AND METHODS

## PDO

- PDO::beginTransaction — Initiates a transaction
- PDO::commit — Commits a transaction
- PDO::\_\_construct — Creates a PDO instance representing a connection to a database
- PDO::errorCode — Fetch the SQLSTATE associated with the last operation
- PDO::errorInfo — Fetch extended error information associated with the last operation
- PDO::exec — Execute an SQL statement and return the number of affected rows
- PDO::getAttribute — Retrieve a database connection attribute
- PDO::getAvailableDrivers — Return an array of available PDO drivers
- PDO::inTransaction — Checks if inside a transaction
- PDO::lastInsertId — Returns the ID of the last inserted row or sequence value
- PDO::prepare — Prepares a statement for execution and returns a statement object
- PDO::query — Executes an SQL statement, returning a result set as a PDOStatement object
- PDO::quote — Quotes a string for use in a query.
- PDO::rollBack — Rolls back a transaction
- PDO::setAttribute — Set an attribute

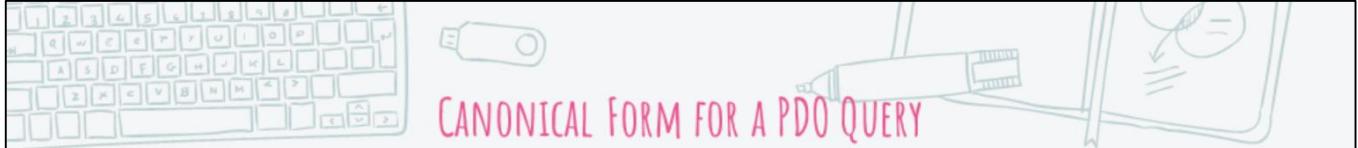
Here is the full API of the PDO class methods. There are additional methods above for transactions (beginTransaction(), commit(), rollback()).

We have covered the other major elements of this API.

## PDOStatement

- PDOStatement::bindColumn — Bind a column to a PHP variable
- PDOStatement::bindParam — Binds a parameter to the specified variable name
- PDOStatement::bindValue — Binds a value to a parameter
- PDOStatement::closeCursor — Closes the cursor, enabling the statement to be executed again.
- PDOStatement::columnCount — Returns the number of columns in the result set
- PDOStatement::debugDumpParams — Dump an SQL prepared command
- PDOStatement::errorCode — Fetch the SQLSTATE associated with the last operation
- PDOStatement::errorInfo — Fetch extended error information associated with the last operation
- PDOStatement::execute — Executes a prepared statement
- PDOStatement::fetch — Fetches the next row from a result set
- PDOStatement::fetchAll — Returns an array containing all of the result set rows
- PDOStatement::fetchColumn — Returns a single column from the next row of a result set
- PDOStatement::fetchObject — Fetches the next row and returns it as an object.
- PDOStatement::getAttribute — Retrieve a statement attribute
- PDOStatement::getColumnMeta — Returns metadata for a column in a result set
- PDOStatement::nextRowset — Advances to the next rowset in a multi-rowset statement handle
- PDOStatement::rowCount — Returns the number of rows affected by the last SQL statement
- PDOStatement::setAttribute — Set a statement attribute
- PDOStatement::setFetchMode — Set the default fetch mode for this statement

The PDOStatement class has a much larger API however, there is a lot of repeated functionality. For example `fetchObject()` is equivalent to `fetch(PDO::FETCH_OBJ)`, which is a query which puts its result data into an object.



## CANONICAL FORM FOR A PDO QUERY

```
const DB_DSN = 'mysql:host=localhost;dbname=dbname';
const DB_USER = 'root';
const DB_PASS = 'pwd';
try {   //1.connect
    $pdo = new PDO(DB_DSN, DB_USER, DB_PASS);
} catch (PDOException $e) {
    die($e->getMessage());
}
$pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

$stmt = $pdo->prepare("SELECT * FROM users WHERE username = :username");           //2.prepare query
try {
    $stmt->execute($_GET);   //3. execute
} catch (PDOException $e) {
    echo $e->getMessage();
    $error = $e->errorInfo();  //4. check error
    die();
}
$user = $stmt->fetch() ?   //5. use data
    echo "found $user['email']" :
    echo "could not find user's email!";
```

Finally here is how all of these methods and features should be put together. This code example forms a reliable template that is likely to be a good starting point for most of the queries you will want to use.

At the top, constants which denote the literal values of the DNS, username and password are included. It is fine to have password data in PHP files, though these should be outside your public web root so a web server cannot accidentally display their contents.

Next, a connection to the database is made by 'new'ing the PDO object and catching an error if it fails. The die() statement is a placeholder for an error handling solution that is relevant to your project, such as a website error page.

Next, the error mode is set to use exceptions. This is to ensure we do not miss any errors that might arise.

The database SQL is prepared and placeholders for data are put in. When executing the \$\_GET array is passed directly, since in this example it contains the 'username' parameter as a key.

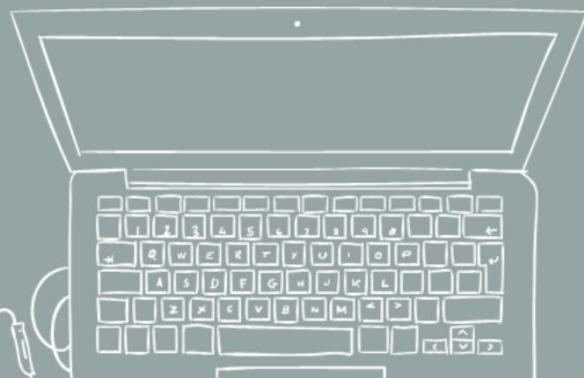
After a little error handling, finally fetch() is called which returns one row from the result (which should only have one row if usernames are unique).



# Any questions?



## EXERCISE 23





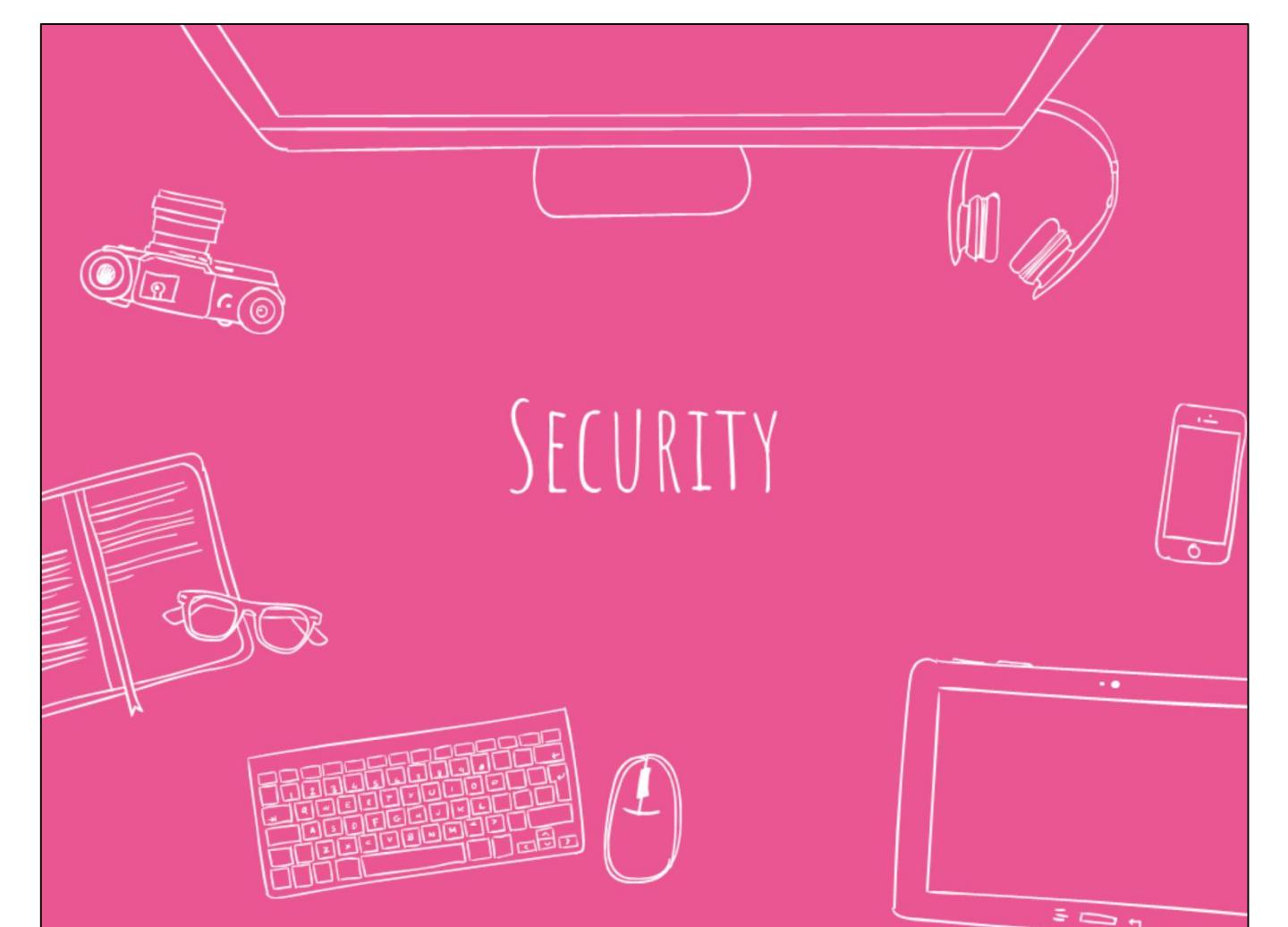
## RESOURCES

Introduction to PDO

<http://php.net/manual/en/intro pdo.php>

PDO Errors and Exceptions

<http://php.net/manual/en/pdo.error-handling.php>



# SECURITY

The issue of security should be foremost in every web developers mind, perhaps even more so than many other kinds of programming. Web developers manage user data, user input and user interaction.

Web developers and web companies are increasingly being held liable for negligent security practices and in most instances "hackers" are not wizards, but educated developers themselves who are aware of the common oversights common in the industry.

PHP has historically been associated with poor security, this however, has nothing to do with the language itself. Loved early by hobbyists, quite a lot of PHP software was written without a sophisticated appreciation of the dangers of development.

In the early web user data was not commodified and remained largely anonymous. Increasingly today however ecommerce, social networks and even personal sites display and store highly personal information. With laws now specifying who owns this data and what can be done with it, an insecure app is not only poor for users but a potential disaster for a company.

## CHAPTER OVERVIEW

- Basics of Web security
- Vulnerabilities
- SQL and query string injection
- Cross-site scripting
- Cookie vulnerability
- Hidden field vulnerability
- Sniffing and replay attacks
- PHP authentication aids

Web security is an extended course in itself, however applications can be very secure without a great deal of expertise. The difficult elements of web security have mostly been solved (SSL, password hashing, etc.) – the challenge for developers is to remember to use them.

This chapter introduces the most common kinds of security flaws present in web applications.

## THREE KINDS OF ATTACK

- Invalid Input
  - SQL
  - Javascript
- Invalid Output
  - Javascript
  - Hidden/Secret
- Invalid State
  - Bad Configuration
  - Bad Session Data
  - Bad Database Data

Insecurity can be divided into three categories: dangerous or invalid input, dangerous or invalid output and dangerous or invalid machine state.

In the case of input, for web developers, this is mostly the kinds of text users enter in forms: is it exactly what you expect? If it isn't, what could happen?

For output, it's the particular web pages you display. Do you have complete control over everything on your web page? Is it exactly as you expect?

Monitoring the state of the machine processing the input and providing the output is vital. What data is stored on the machine? Is it secure? How is the machine configured?

If you are unsure of some aspect of your system you should try to investigate further. The most significant barrier to good security is complacency.

## BASICS OF WEB SECURITY

### Website vulnerabilities:

- Allow an attacker to:
  - Pose as a trusted user
  - Execute malicious commands
  - Gather confidential data
  - Block access to your website by conducting a denial of service attack
- Websites can be secured by:
  - Identifying and fixing security holes in application code
  - Implementing security mechanisms on the server
  - Configuring security on the network
- Recommended reading:
  - "Essential PHP Security" by Chris Shiflett (O'Reilly)
  - PHP the Right Way

Consider a standard website, an ecommerce store for example. What *could* happen?

Let's take an example from the slide: a person poses as a trusted user.

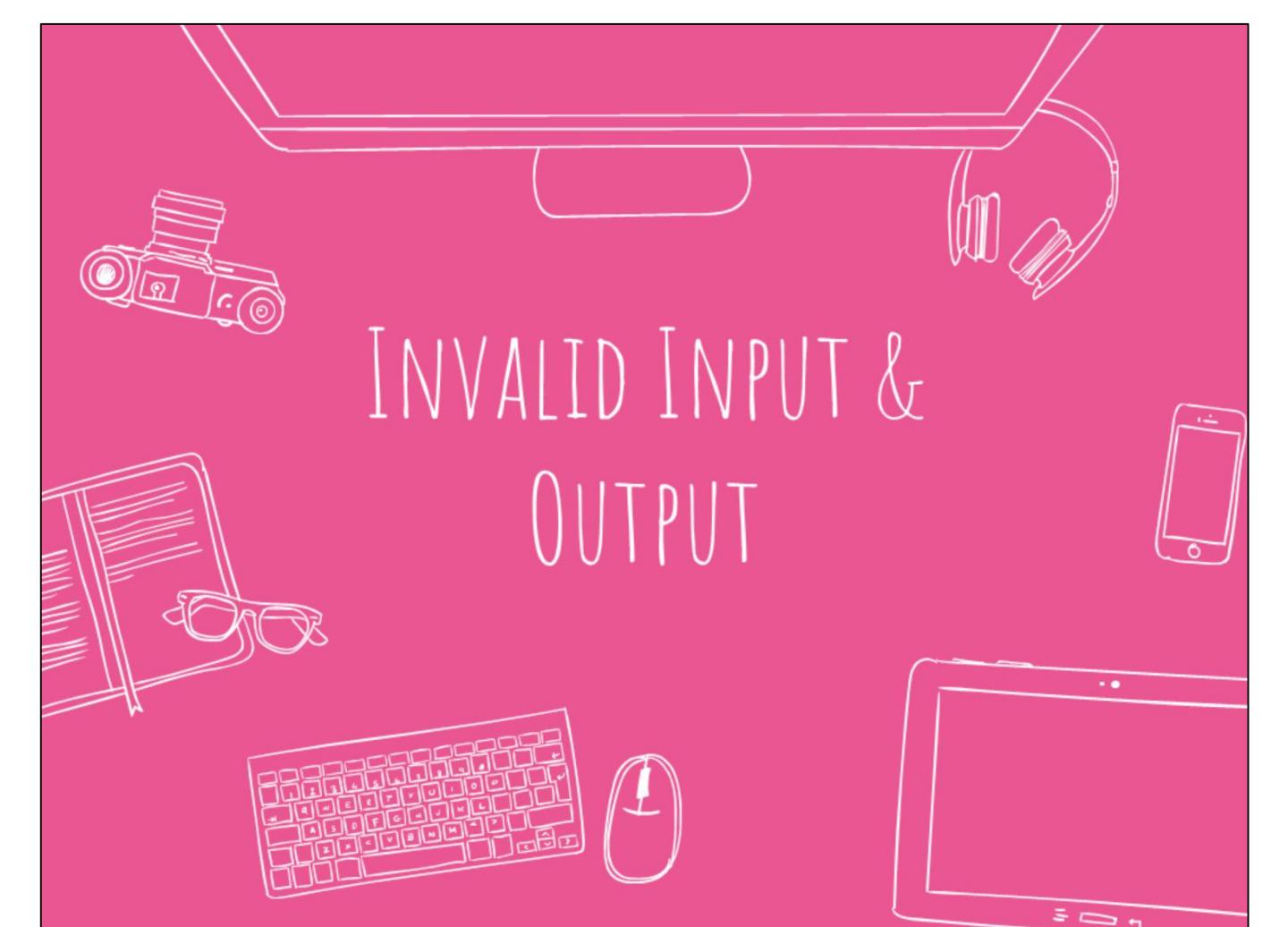
Suppose our password reminder system is weak (systems which merely ask questions are typically very weak). Is the number of attempts limited? No. Then the attacker has access.

So an attacker gains access to an account which has a stored credit card. Do you ask for a 4-digit confirmation of the card's number before sale? No.

The attacker purchases a variety of things to an address. Do you check the address is the same as the one on the card? No. Then he sends it to an untraceable PO BOX.

Are you monitoring transactions for fraudulent characteristics? Sudden address changes? Purchases much larger than typical?

Your customer has now been defrauded. You might be liable. And certainly a system that can be exploited once will be done 100s or 1000s of times before you realize there is an issue.



# INVALID INPUT & OUTPUT

The issue of security should be foremost in every web developers mind, perhaps even more so than many other kinds of programming. Web developers manage user data, user input and user interaction.

Web developers and web companies are increasingly being held liable for negligent security practices and in most instances "hackers" are not wizards, but educated developers themselves who are aware of the common oversights common in the industry.

PHP has historically been associated with poor security, this however, has nothing to do with the language itself. Loved early by hobbyists, quite a lot of PHP software was written without a sophisticated appreciation of the dangers of development.

In the early web user data was not commodified and remained largely anonymous. Increasingly today however ecommerce, social networks and even personal sites display and store highly personal information. With laws now specifying who owns this data and what can be done with it, an insecure app is not only poor for users but a potential disaster for a company.

## TYPES OF ATTACK

- SQL injection:
  - Modifying and executing SQL present in application code
- Query injection:
  - Manipulating URLs and query strings via GET requests
- Cross-site scripting:
  - Injecting malicious code in web pages viewed by other users
- HTTP Redirects
  - Redirecting based on input
- Cookie theft/poisoning:
  - Accessing cookies and modifying their value before sending them back to the server
- Hidden Field dependencies

User input, anywhere exists, ought to be treated as hostile.

Typically commands sent to the database will contain data entered by the user in a HTML form (e.g. a username for a login).

Query (\$\_GET) parameters are often used unthinkingly.

Templates are often filled unknowingly with user data. PHP code does not flag which variables contain "untrusted" data, code can often read safely but actually operate on input a user has sent in.

# VULNERABILITIES OF AUTHENTICATION

## Password Guessing

- Process of breaking a security scheme, such as authentication, by trying different combinations
- Brute force
  - Programmatically generating successive passwords
  - Long passwords and locking accounts reduces the effectiveness
- Dictionary
  - Using a dictionary to generate likely passwords
  - Test the password strength on creation against a dictionary

## Reverse directory transversal:

- Bypassing an authentication scheme to access any protected resource

Password guessing can never be totally avoided over the long term when we consider that, given any password length and any character set, there is a finite number of password combinations that exist. Of course the longer the password and the larger the character set the more combinations. So the password should be case-sensitive and allow (and encourage) digits.

Unfortunately some social engineering reveals that uppercase is most commonly used for the first character, or after digits. A common use of digits is to substitute the number 1 (one) for the letter I ('ell') and the number 0 (zero) for the letter o ('oh') - just to conform to the site's password rules. A dictionary attack can easily be extended to include these cases, and so should your testing.

Given that over time any password can be guessed, what we can do is detect a number of failed attempts and disable the account. The down side is that the attacker has now partially achieved a Denial of Service (DOS). If all accounts are known then trying to unsuccessfully logon to every account will make the system unusable. Therefore usernames should also be difficult to guess.

All this is meaningless if the attacker can bypass the authentication scheme and go straight to the meaty screens by knowing their URL. An additional check, usually using a session and some other salt, is also required for every subsequent transaction.

## SQL INJECTION

- Process of manipulating an existing SQL command present in application code
- Occurs through:
  - Input fields of forms
  - Query strings
  - Custom HTTP requests
- SQL injection is vulnerable code that directly executes user-supplied data in SQL statements



Attackers use SQL injection to pass SQL commands through a Web application for execution by a database.

In Web applications, programmers often use SQL commands that execute directly with user-provided parameters. Attackers use this feature to embed malicious SQL commands in these parameters. Attackers often pass parts of SQL commands through form fields, which are embedded in the SQL command of the application code at run time, and finally executed at the database.

## EXAMPLE OF SQL INJECTION IN A LOGON FORM

```
$login_id=$_GET['login_id'];
$password=$_GET['password'];
$sql="select * from login_master where
      login_id='$login_id' and password='$password'";
$run=mysql_query($sql);
```

In both fields, type:

'attack' OR ''=''

- The resultant SQL statement that will execute on server:

```
select * from logon_master where
logon_id='attack' OR ''=''' and password='attack' OR ''=''
```

- Consequence:

- A result (the first row) will always be returned indicating that the logon is valid

The SQL statement resulting from the SQL injection shown on the slide contains a WHERE clause in which all the qualifying conditions are met. This is because the WHERE clause compares a quotation mark (nothing) to another quotation mark (nothing), which will always return True. As a result, the SQL command will always return the first row of the table. As the application code receives a row indicating that a matching user name and password has been found, it authorizes the attacker sending the request.

## AVOIDING SQL INJECTION

- Always validate user input
- Validate user input for special characters:
  - Double dash (--), single quote('), semicolon (;) and so on
  - Use a regular expression, PHP ctype\_ or filter\_ functions
- Use prepared statement with PDO
- Use the mysqli\_real\_escape\_string() function with mysqli
  - Escapes special characters by prefixing backslashes
  - Applies to the \x00, \n, \r, \, ', " and \x1a characters
  - If using a different database, use the escape function for it
  - Databases have different special character tolerances

Prepared queries (provided by PDO and MySQLi) separate SQL syntax from user data. Always use them.

However do not assume because you have used a prepared query that there is no longer any threat.

There are data values specific to your application which might cause security issues. For example, if you are checking a person is a certain years old and they enter the year 1800, what would happen? Or 0? Or a million zeros?

Validate all user input, even if the conditions are trivial. There are an infinite number of inputs and you cannot predict all of them; however you can specify limits.

## QUERY STRING INJECTION

- Process of manipulating the query string of a URL
- Can be used to perform SQL injection
- Can be used to access resources requiring authentication but not protected from direct access
  - Example: directly accessing the administrator page by bypassing the logon page
- Query string injection examples:

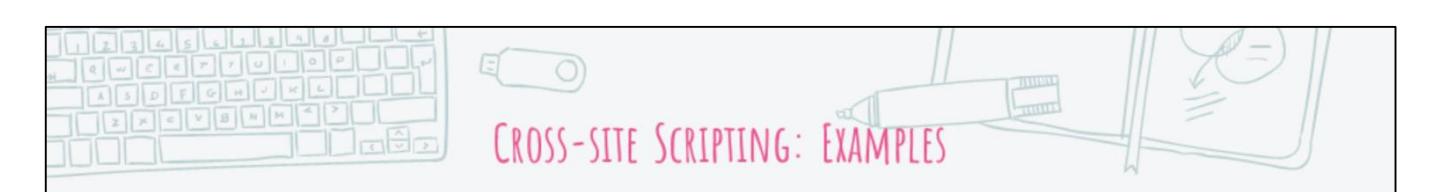
```
http://example.com/web/enquire.php?user_name=Mark+Smith  
http://example.com/web/enquire.php?user_name=Hack;DROP TABLE  
users
```

## CROSS-SITE SCRIPTING (XSS)

- Allows an attacker to:
  - Direct malicious code from a trusted site to unsuspected users
  - Steal cookies and redirect user to other websites
- The attacker:
  - Locates cross-site scripting holes in websites
  - Injects client-side scripts in hyperlinks targeting the cross-site scripting holes
  - Sends the hyperlinks to unsuspecting users
- Users click the hyperlink
- Consequence
  - Send back the malicious script to be executed on the client browser

In a cross-site scripting attack, an attacker usually sends a hyperlink to a trusted website with malicious scripts to unsuspecting users. The attacker can send the hyperlink through an e-mail message, message board, or an instant message. When a user clicks the hyperlink, the trusted website sends back the malicious script along with the response to the user. The script executes on the browser of the user.

Be wary of what you are echoing. Any user input could be a malicious trick.



## CROSS-SITE SCRIPTING: EXAMPLES

- PHP code vulnerable to cross-site scripting:

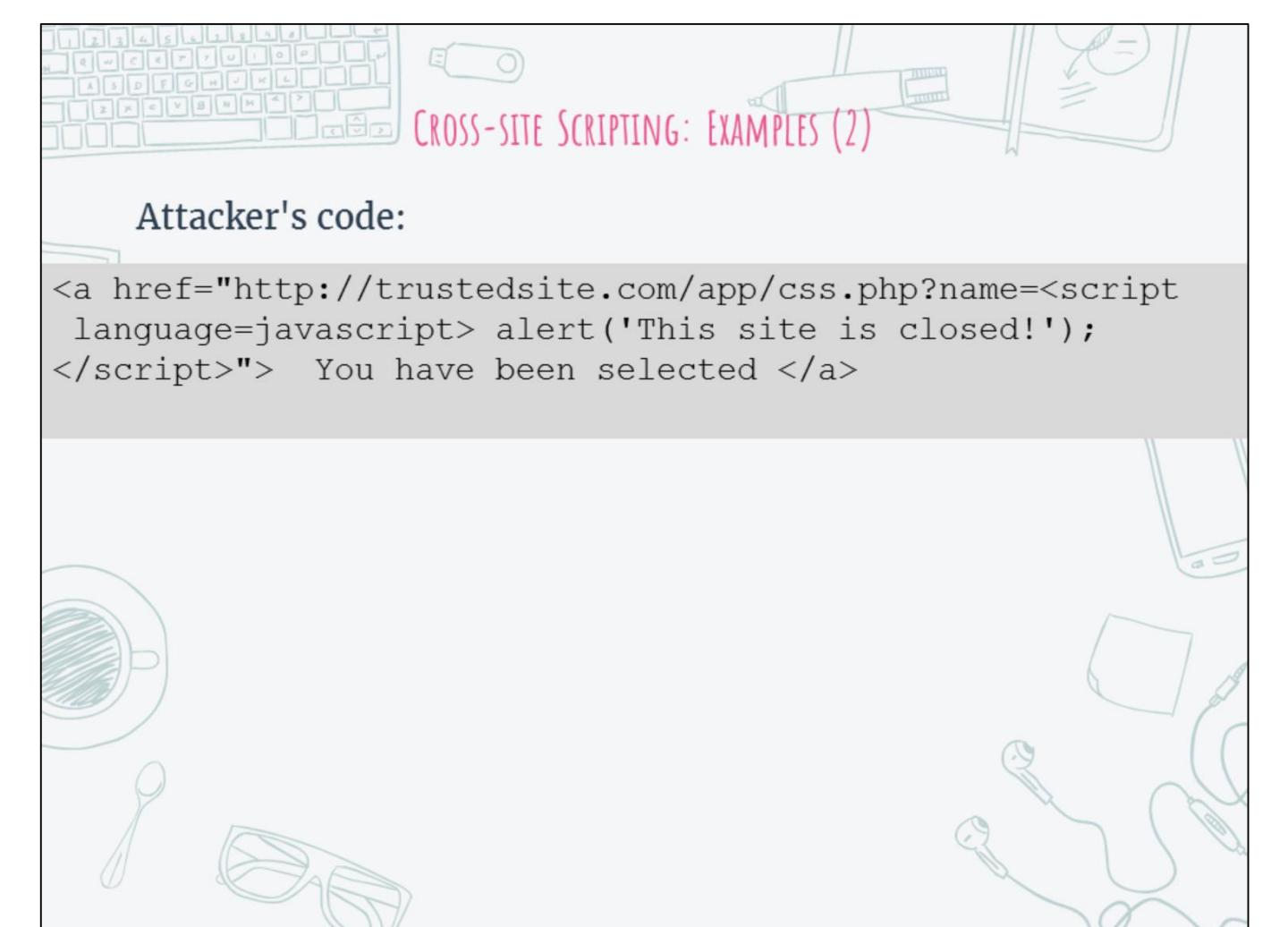
```
<?php  
    echo 'Welcome to our site '. $_GET['name'];  
?>
```

In fact, any code that uses user supplied data without:

- Filtering
  - PHP ctype\_ functions are useful for validation
  - See also the PHP filter\_ functions
- Escaping
  - htmlentites() on any transmitted text
  - striptags() to remove html tags

For example, ctype\_alnum is useful for checking fields such as passwords. Regular expressions are an alternative, but generally slower.

The filter\_ functions are newer (PHP 5.2) and require a little study. They are worth the effort though, for example one of the validations supported is FILTER\_VALIDATE\_EMAIL - almost certainly a better validation for an email address than you can dream-up yourself.



## CROSS-SITE SCRIPTING: EXAMPLES (2)

### Attacker's code:

```
<a href="http://trustedsite.com/app/css.php?name=<script  
language=javascript> alert('This site is closed!');  
</script>"> You have been selected </a>
```

The Attacker's code on the slide shows how to create a hyperlink for cross-site scripting. The hyperlink displays an innocuous or enticing text message, such as 'You have been selected', and points to a trusted URL with malicious JavaScript embedded in the query string.

The following script shown in the example on the slide opens an alert window with a message when a user clicks on the script:

```
script language=javascript> alert('This site is closed!');</script>">
```

## CROSS-SITE SCRIPTING: EXAMPLES (3)

### Attacker's code

```
<a href="http://trustedsite.com/app/css.php?name=
<script language=javascript>
location.href='http://evilhacker.com';
</script>"> Click for your gift </a>
```



The script shown in the example redirects a user to a different URL.

While creating a hyperlink, the attacker usually replaces the JavaScript code with the HEX value of the ASCII character prefixed with the "%" to make it look unsuspecting. For example, the `<script>` part of the URL can be encoded as `3C%73%63%72%69%70%74%3E`.

## AVOIDING XSS

- Validate and sanitize all user input
- Use `htmlentities()` function on any user-supplied text being displayed
  - Converts special characters to HTML text
  - < to &lt;, > to &gt; and so on.
  - Reverse with `html_entity_decode()`
- Remove HTML tags with `strip_tags()` function

User supplied text

```
$cleaned = htmlentities($text, ENT_QUOTES, 'UTF-8');
```

Did we already say you should validate, filter, and escape text? This is the single most important thing you can do as a programmer to help secure your applications. Of course it is not the *only* thing you have to do, just the one that is most often forgotten.

Unfortunately `htmlentities()` is not 100% perfect, for example it returns NULL if it finds a character it does not recognise, and, for simple characters, can be slow.

These are PHP's native tools however. You should almost certainly consult one of the major PHP frameworks and use its validation and sanitation tools.

Do not try to solve complex problems yourself, wherever possible, use well-regarded open source code. Of course, however, learn what it does and how it works.

## COOKIE VULNERABILITY

- Cookies are vulnerable to theft and poisoning
- Cookie theft:
  - A process that allows an unauthorized party to receive a cookie or information about a cookie
  - Can be performed through packet sniffing, query injection, and cross-site scripting
- Code vulnerable to cookie theft:

```
<?php  
    setcookie('password', 'pass123');  
    setcookie('usertype', 'guest');  
    echo 'Welcome to our site '. $_GET['name'];  
?>
```



## COOKIE VULNERABILITY (2)

- Cookie theft through cross-site scripting:

```
http://vulnerable.com/web/
```

```
cookie.php?name=<script>alert(document.cookie)</script>
```

- Cookie poisoning:
  - Process of modifying the value of a cookie before sending it back to the server
- To minimize damage, do not store sensitive information in cookies

Cookie poisoning is a process in which an attacker modifies a cookie before sending it back to the server. For example, suppose a shopping store stores the total cost of items that a user buys in cookies. In this application, an attacker can minimize the total shopping cost stored in a cookie before sending it back to the server. Because cookies store information in plain text, you should never store sensitive information in cookies. Instead, assign a session ID to a cookie and map it to the sensitive information stored on the server to maintain the session.

Store all information in `$_SESSION` and preferably `set_session_handler()` to use a secure database.

Use cookies only for very basic information, even "non-sensitive" information may turn out to lead to a security hole.

## HIDDEN FIELD VULNERABILITY

- Hidden fields are vulnerable from:
  - Being modified in the client-side, and in the browser address bar
- Scenario:
  - Step 1: Website stores cost of items in hidden fields
  - Step 2: Attacker saves the HTML page to local computer
  - Step 3: Attacker modifies the hidden fields to reflect lower cost
  - Step 4: Attacker submits the modified Web page
  - Step 5: Website sells the items at lower cost
- Code with hidden field vulnerability

```
<input type='hidden' name='price_product1' value='80.50'>
```

Hidden input fields are ordinary form fields. The word "hidden" is perhaps a little unfortunate, contemporary browsers (e.g. Chrome) have developer tools that make modifying live html trivial.

Hidden user input is user input, even if you wrote it.

INVALID STATE

## CONFIGURATION: APACHE & PHP

- If mod\_php isn't installed properly the raw source of PHP files can be exposed
  - Move highly sensitive passwords to .Ppk files or
  - Put passwords in known secure locations **outside of the document root**
  - So code contains no passwords!
- VirtualHosts, environmental settings can affect what code is executed
  - Always understand and review configuration files in their entirety
  - Understand .htaccess and whatever folder-specific settings are used
- Use highest level of error reporting (E\_ALL)
  - Send all errors to logs
  - Display\_errors, off

Web development typically involves four classes of configuration:

database,  
php  
operating system  
web server.

Unless you have dedicated team members, you have to understand the configuration for each, though this is a difficult task it is necessary.

Even in larger teams with dedicated engineers for each element of the system, you will be handed design constraints for your program that derive from configuration details in each of these systems. Without understanding how each of these may be configured you may find writing applications difficult.

## PASSWORD SECURITY

- Start with the way passwords are stored
  - Never store passwords as plain text
  - Even encrypted passwords should be seeded
- One-way encryption is more secure
  - When a password is forgotten, generate a random one
- Reversible encryption allows password reminders
  - But if you can reverse the encryption, so can a hacker
- Once signed-on, an encrypted expiry date-time can be embedded as a hidden field or session
  - Avoids replay attacks
  - Server-side checks should detect interference with the field

Some sites have been known to store passwords either as plain text or using a reversible encryption. This is so that a forgetful user can be reminded by email what their password is. This is very user-friendly, but also attacker-friendly. Better to change the password randomly and send the user a new password which they should be able to change once they have logged on. Here is some PHP code to generate a random password. It uses `mt_rand()` which is fast and does not require a seed:

```
$password = '';
$length    = mt_rand(8,12);
while ( strlen($password) < $length ) {
    $char = chr(mt_rand(48,122));    # ASCII characters
    if (preg_match('/[A-Za-z0-9]/', $char))
        $password .= $char;
}
echo "Password of $length chars is: $password";
```

By the way, when changing a password always check the old password first, in case a hacker manages to get to the change-password screen. Defence in depth.

Even when a password is encrypted, if someone can see the encrypted version then it is possible to infer things from it. For example, if two users have the same password then the encrypted versions will be the same.

Crack one and you have cracked the other as well. One way around this is to include the username with the password, which should give a different encryption or hash for each user.

## SESSION VULNERABILITY

- If cookies are disabled in the browser, PHP uses GET
  - Easy to see and to alter

```
http://www.someplace.com?PHPSESSID=4567890123
```

- Force the user to have cookies turned on
  - Site will fail, not display a GET request

```
ini_set('session.use_only_cookies',true);
```

- Can force a regeneration of the session id
- Prevents "session fixation"
  - Regeneration of the session id at regular intervals (every 30 seconds, say) makes life difficult for the hacker

```
session_regenerate_id();
```

Always call `session_regenerate_id()` after a change in status, for example after login.

Refusing to serve users who reject cookies is a better approach. However given contemporary EU directives which require users accept cookies, you may find you need to rely on the `PHPSESSID` query parameter.

## SNIFFING ATTACKS

- Intercept and log traffic passing over a network
- Involve intercepting, analyzing, and decoding the content of data packets
- Are performed using packet sniffers plugged into computer networks
- Consequences:
  - Attacker can get access to sensitive information, such as user name/password, credit card numbers, and other messages
- Prevention:
  - Use secure communication, such as HTTPS over SSL
  - Implement encryption/decryption techniques



Packet sniffers are tools for network administrators to analyze network problems, monitor network usage, debug client and server communication, and detect malicious activities. However attackers are increasingly misusing packet sniffers for malicious activities.

Many services, protocols, and applications on the Web exchange data in plain text making communication vulnerable to sniffing attacks.

Sniffing attacks work passively by logging data without altering. As a result, it is nearly impossible to detect such attacks.

## REPLAY ATTACKS

- Intercept data and retransmit it repeatedly, immediately or at a later time
- Scenario:
  - Step 1: User sends encrypted user name and password
  - Step 2: Attacker intercepts the user name and password
  - Step 3: Attacker retransmits the user name and password at a later time
- Prevention:
  - Use digital signatures



Replay attacks, also known as the man-in-the-middle attacks, are employed by attackers to perform unauthorized operations, such as illegally accessing protected resources through false identification, performing duplicate transactions, and even launching a denial of service attack.

Using digital signatures on the server can prevent a replay attack. The digital signature confirms that the information received is indeed from the person sending it.

## MORE INFORMATION

- **WebGoat**
  - An insecure J2EE Web Application
  - Allows you to learn security testing skills
  - Because you should never attack a real site without permission
  - Part of the OWASP community
  - [http://www.owasp.org/index.php/Category:OWASP\\_WebGoat\\_Project](http://www.owasp.org/index.php/Category:OWASP_WebGoat_Project)
- **OWASP Guide**
  - A guide on how to write secure applications
  - [http://www.owasp.org/index.php/OWASP\\_Guide\\_Project](http://www.owasp.org/index.php/OWASP_Guide_Project)
- **The PHP Security Consortium**
  - <http://phpsec.org>
  - <http://phpsec.org/projects/guide>
  - <http://www.php.net/manual/en/security.php>

 WEBGOAT

PHPSeclInfo, supplied by the PHP Security Consortium at <http://phpsec.org/projects/phpseclinfo/index.html> checks your PHP settings for security holes.

# PHP SECURITY FUNCTIONS & TECHNIQUES

## HASHING PASSWORDS

- Password hashing is one of the most basic security considerations that must be made when designing any application that accepts passwords from users.
    - Without hashing, any passwords that are stored in your application's database can be stolen if the database is compromised,
  - Hashing algorithms such as MD5, SHA1 and SHA256 are designed to be very fast and efficient.
    - With modern techniques and computer equipment, it has become trivial to "brute force" the output of these algorithms, in order to determine the original input.
- PHP 5.5 provides a native password hashing API that safely handles both hashing and verifying passwords in a secure manner.
- There is also a pure PHP compatibility library available for PHP 5.3.7 and later.

Passwords should not be stored in plain text. They should not even be stored encrypted.

Encryption is a reversible processes, hashing is not. By analogy, hashing is a little like mixing a lot of paint together – the final colour tells you nothing about which exact colours you started with. Or adding numbers: given the number 20 which numbers did you add together to get it? Was it  $10 + 10$  or  $19 + 1$  or ...

You must always store hashes, never the original or even encryptions of the original. When the user logs in you hash the password they submitted and compare it to the hash in the database to see if the hashes match.

Hashing isn't perfect however there is a PHP library which simplifies the process.

## HASHING API

### Password Hashing Functions

- `password_get_info` — Returns information about the given hash
- `password_hash` — Creates a password hash
- `password_needs_rehash` — Checks if the given hash matches the given options
- `password_verify` — Verifies that a password matches a hash

```
$hash = '$1$toHVx1uW$KIvW9yGZZSU/1YOidHeqJ/';
```

```
if (password_verify('rasmuslerdorf', $hash)) {  
    echo 'Password is valid!';  
} else {  
    echo 'Invalid password.';  
}
```

Shown is an example \$hash that might be stored in a database.

`password_verify` compares some user input password (e.g. rasmusledorf) with the known hash.

`password_verify()` will hash the user input using the same algorithm that the stored hash used, and will otherwise manage all the message details of comparing passwords.

NOTE: Rasmus Lerdorf is a Danish-Canadian programmer. He created the PHP scripting language, authoring the first two versions of the language and participated in the development of later versions.

## REHASHING AND HASHING COST

- Standard approach for using password\_\* API is to also ask if the password needs rehashing

```
f (password_verify($password, $hash)) {  
    if (password_needs_rehash($hash, PASSWORD_DEFAULT)) {  
        $hash = password_hash($password, $algorithm);  
        /* Store new hash in db */  
    }  
}
```

- This will occur when PHP updates its hashing API
  - You can increase the time it takes to hash a password with a cost parameter

```
$hash = password_hash($password, PASSWORD_BCRYPT, ['cost'  
=> 10]);
```

If PHP were to update its hashing system you may wish to take advantage of it. In the code above the function `password_needs_rehash()` checks to see if there is an update and will give you a more secure hash if there is.

This check isn't required but it is a recommended best-practice.

## TINTED DATA

- All user data should be considered "tainted"
  - `$_GET`, `$_POST`, `$_REQUEST`
  - Also `$_SERVER` because REFERERs, USER AGENTS, and other data comes from the HTTP Request
  - `$_COOKIE`
  - Anything coming from the database tables
- The entire HTTP Request is user data
  - every element may be designed to be hostile
  - a recent flaw in PHP (and many other languages) meant a specific number of query parameters designed in a particular way caused the VM to run out of memory and crash

One of the most frequent oversights of security-minded developers is the `$_SERVER` array. Many elements of that array come from the HTTP request or are otherwise inferred from user data.

It is quite easy, for example, to fake a User-Agent and put in there some malicious SQL.

## PHPINFO(), GIT AND FILE EXPOSURE

Developers have been known to:

- Put git repositories (.git directories) under the document root
  - .git is a hidden folder so easily missed
  - This then exposes the repository and all of its contents (including passwords!)
- Leave phpinfo()-style files accessible at the root
  - This isn't inherently insecure, but it reveals the configuration state of your machine revealing potential vulnerabilities
- Leave .ppk, other primary key, password files and text files exposed
- Your document root should be a "vault"
  - Totally separate from any development code
  - Ensure file permissions are appropriate
  - Back up your document root for easy revert from live



The only code which should be accessible on your domain is exactly and only the code which needs to run. This may, even, just be an index.php.

All the PHP code which is included, can be put outside your public document root.

You must audit every file placed on a publically accessible machine, and understand exactly what files can be accessed from the webserver's document root.

Developers are often inclined to put all their code on every machine for the sake of convenience. For example, all the PHP code on the database machine. However this is an invitation for a hacker to compromise your entire system.

Chown is a command on Unix to change the owner of a file-based resource.

The **chgrp** (from **change group**) command may be used by unprivileged users on Unix-type systems to change the group associated with a file system object (such as a file, directory, or link) to one of which they are a member. A file system object has 3 sets of access permissions, one set for the owner, one set for the group and one set for others. Changing the group of an object could be used to change which users can write to a file.

## PHP CRYPTOGRAPHY

### A number of encryption and hash systems exist

- PHP extensions
  - Crack Tests password strength
  - Hash Generate hash values
  - Mcrypt Encrypt text using a variety of algorithms
  - OpenSSL Generation and verification of signatures
- PHP builtins
  - crc32 Generate a cyclical redundancy checksum
  - for text
  - crypt Encrypt text using the DES, MD5, or Blowfish algorithm
  - md5 Encrypt text using the MD5 algorithm
  - sha1 Generate a hash using the Secure Hash Algorithm 1
- Others: Several pure PHP implementations of AES. Careful use of Rijndael encryption in Mcrypt gives AES compliance

**Hash** replaces the older MHash extension.

One of the best encryption standards is **AES** (Advanced Encryption Standard) and AES-256 is one of the block ciphers. The Rijndael cipher is general adopted as conforming to this standard ("Rijndael" is the concatenation of the first four characters of the author's names). The **Mcrypt** extension supports AES by the name RIJNDAEL, AES is the standard, not the implementation, so don't search for AES in the documentation!

**crypt** produces a one-way encryption, and by default will attempt to use the best algorithm available on your system. Although based on the C function **crypt (3)**, from PHP 5.3 it uses its own versions of algorithms if not available in the C library.

**SHA1** generates a condensed representation of text - a hash, so strictly speaking it is not an encryption. However we can store this hash instead of the password, and check that the supplied password has the same hash.

**md5\_file** and **sha1\_file** are also available to create a hash for a file using their respective algorithms.

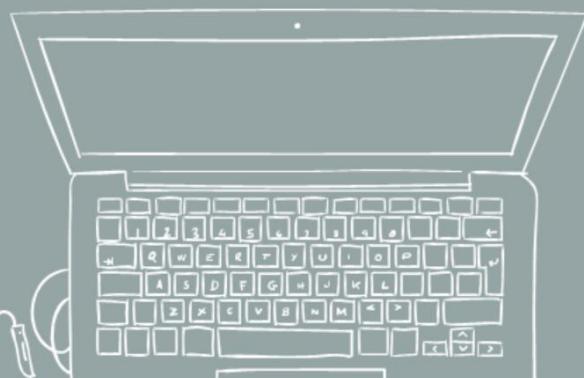
**str\_rot13()** function also exists, but should not be considered secure in any way.



# Any questions?



## EXERCISE 24



## RESOURCES

OWASP Attacks

<https://www.owasp.org/index.php/Category:Attack>

OWASP Vulnerabilities

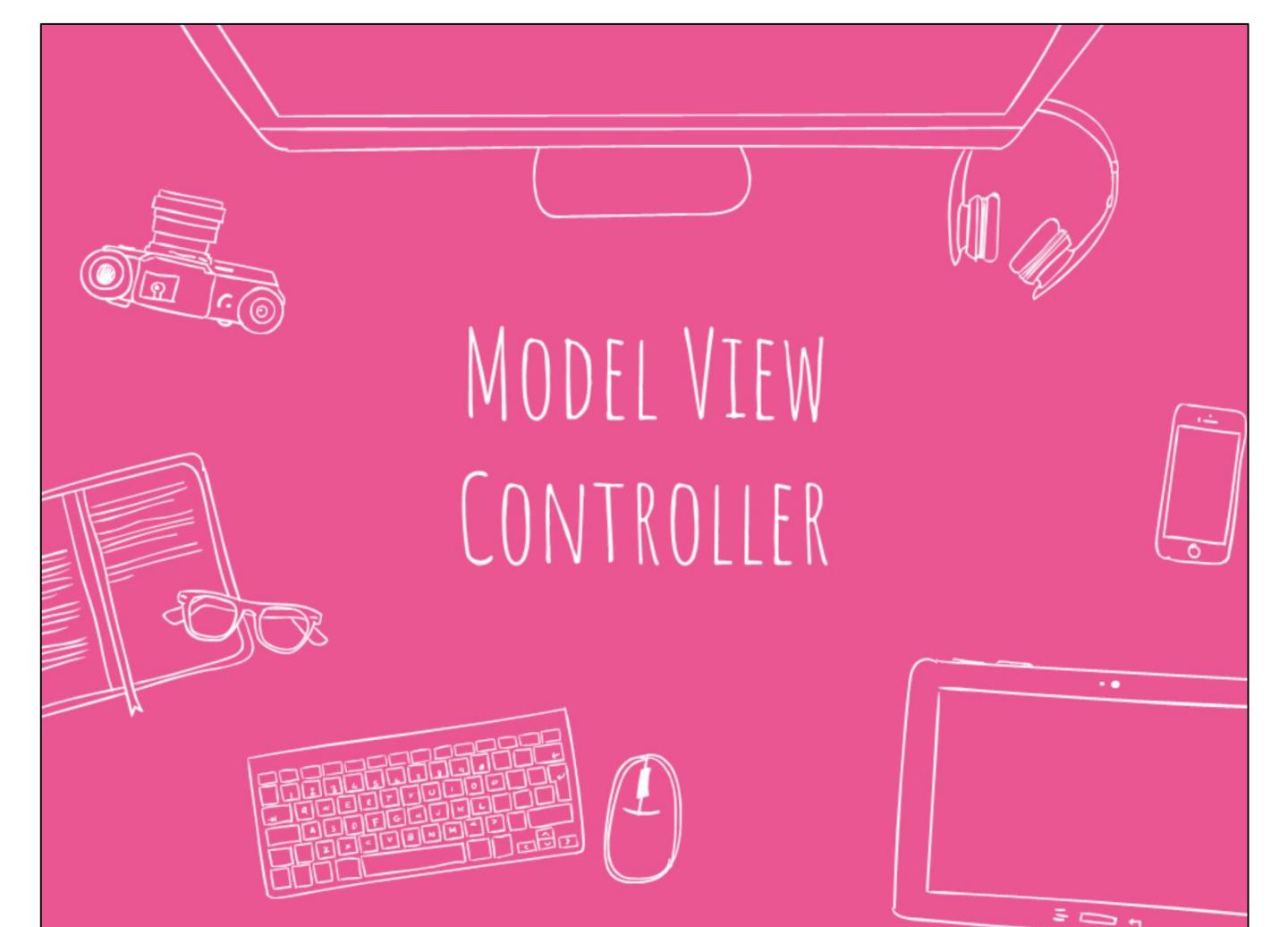
<https://www.owasp.org/index.php/Category:Vulnerability>

WebGoat User Guide

[https://www.owasp.org/index.php/WebGoat\\_User\\_Guide\\_Introduction](https://www.owasp.org/index.php/WebGoat_User_Guide_Introduction)

WebGoat PHP

<https://www.owasp.org/index.php/WebGoatPHP>



# MODEL VIEW CONTROLLER

In contemporary web development programmers usually begin their projects by selecting a framework. This will provide a library of helper utilities that assist with common tasks, but it also will come with a vision for how web applications should be structured: what code should go where, why it should and how it should.

And frameworks tend to be quite opinionated. They say code regarding data should have a very particular kind of structure and so for code regarding templates, and decision logic and domain logic and the rest. However the way a framework requires you to operate isn't usually arbitrary but comes from a very considered view on what works best.

Nevertheless different frameworks will work differently. The overarching view which joins many PHP frameworks together is known as "MVC" or the model-view-controller pattern.

This chapter will look at frameworks, design patterns and MVC in particular and provide some practical code which is characteristic of PHP MVC frameworks.

## CHAPTER OVERVIEW

- Design Patterns
- MVC Concepts
- MVC Code
- MVC Applications
- MVC Frameworks

By the end of this chapter you should understand basic MVC terminology: what a controller is, what an action is; what the words model and view mean.

The code is at times new and a little challenging and you are not expected to be able to write it without any assistance yourself, however it is important to understand what the code does, and particularly, why it does it that way.

## DESIGN PATTERNS

- In the 1970s Christopher Alexander noted that in architecture there were certain recurring patterns of design that might be reused across projects
  - The idea has been incorporated into application design theory
  - Object oriented programs are often taken to follow certain forms and patterns which recur over projects
  - These are known as "Design Patterns"
- Learning design patterns is important to progressing as an application designer:
  - First learn the pattern – the code and how it works
  - Then learn the motivation for the pattern – why was it invented?
  - Then learn to critique the pattern – what does it do wrong?

All house builders know, that despite the details of a particular project, there are sets of recurring patterns to their work. The houses they build will have a roof, and there will be patterns for roofs they can follow: arched, tiled, etc.

In the same way there are standard patterns of design that have been worked out by experienced developers over the last several decades of development. The phrase "design pattern" itself specifically refers to patterns which reoccur in object oriented development which derives from the 1994 book "Design patterns : elements of reusable object-oriented software".

Patterns are given names (e.g. Singleton, Factory, Registry) and generic forms that can be applied to particular problems. The decision to apply a design pattern however isn't automatic: just as a builder might choose the wrong roof for the climate, a developer can easily choose the wrong design pattern for the problem.

With PHP's increasing focus on object-oriented development there is a growing expectation that new developers will at least know some design patterns. You should however, if you want to be an expert, go beyond rote learning of patterns and try to understand why they exist and particularly what their limitations are.

Note: Christopher Alexander had a more far reaching and ethically-oriented view of design patterns as they applied to architecture. His basic idea and terms have been copied across to program design theory but in practice they bare little resemblance to what Christopher originally had in mind.

## FRAMEWORKS

- Frameworks are "scaffolds" on which you can hang domain-specific code
  - they use a lot of design patterns to abstract away the same code which is repeated across web projects
  - they often use powerful and advanced features of the language to achieve this level of abstraction
- In 2005 – 2006 PHP web development leapt forward
  - the first object-oriented web frameworks for PHP were introduced
  - these frameworks were large unmodular behemoths
  - With PHP5.3 and Git in 2009, PHP leapt forward again
  - Auto-loading and namespacing lead to a new framework revolution
  - With PHP5.4 – 7 a third generation of smaller, functional frameworks emerged

Thankfully in the PHP world there are leading examples of object-oriented design and the application of design patterns. These are frameworks.

Frameworks do not solve any "domain" or business problem: they do not come with code specifically for shopping carts. However, they solve the problems of developers: reusable code that helps with common tasks; how to layout your application well so that it is modular and maintainable; and building skills that can be reapplied over-and-over.

PHP frameworks mostly require you to follow the MVC pattern for the layout of your code.

## THE MVC PATTERN

- The Model View Controller pattern is not quite a design pattern
  - A design pattern is a *repeatable form of code*
  - The MVC pattern is sometimes called an "architectural pattern"
  - it implies a separation of code into particular categories which might be realised by more specific kinds of MVC-design-pattern
- MVC says code should be separated into three layers:
  - Models: the data-representation layer
  - Views: the data-presentation layer
  - Controllers: the data-control layer
- How each layer is connected together is not specified by MVC itself!
  - Some people connect models and views together directly
  - Some connect views to models via controllers. This is the most typical "reading" of MVC for web frameworks

There are many ways of reading the term MVC and understanding what it requires. In the most general sense it requires you separate your code into three kinds of logic: that which concerns data, that which concerns templates and that which concerns business or domain decisions.

MVC doesn't end there: since this is an object-oriented pattern, MVC further states that every kind of logic should have its own class. Data should be represented by Model classes, Views by view classes and control flow code (business logic) by Controllers.

This is about as specific as MVC is if it is taken to apply to all the frameworks which call themselves "MVC". However, some web developers read "MVC" to mean a very specific implementation of a 3-layer design (e.g. with an Object Relational Mapping (ORM) layer, a Front Controller, etc.) due to that having a kind of de facto status amongst web frameworks from 2006, to perhaps, 2013. This particular style of MVC is still common today.

This chapter will look at a basic framework and implementation of MVC that is very close to its most common interpretation by MVC PHP frameworks.

## MODEL VIEW CONTROLLER

- Models are typically classes that "know something about the data"
  - What exactly they know depends on the specific implementation
  - In the simple case models are just classes that "look like" database tables
  - In the complex case models have database access and can perform save/update/delete operations
  - This is known as an ORM (object-relational mapping)
- Views are typically just PHP/HTML mixtures (".

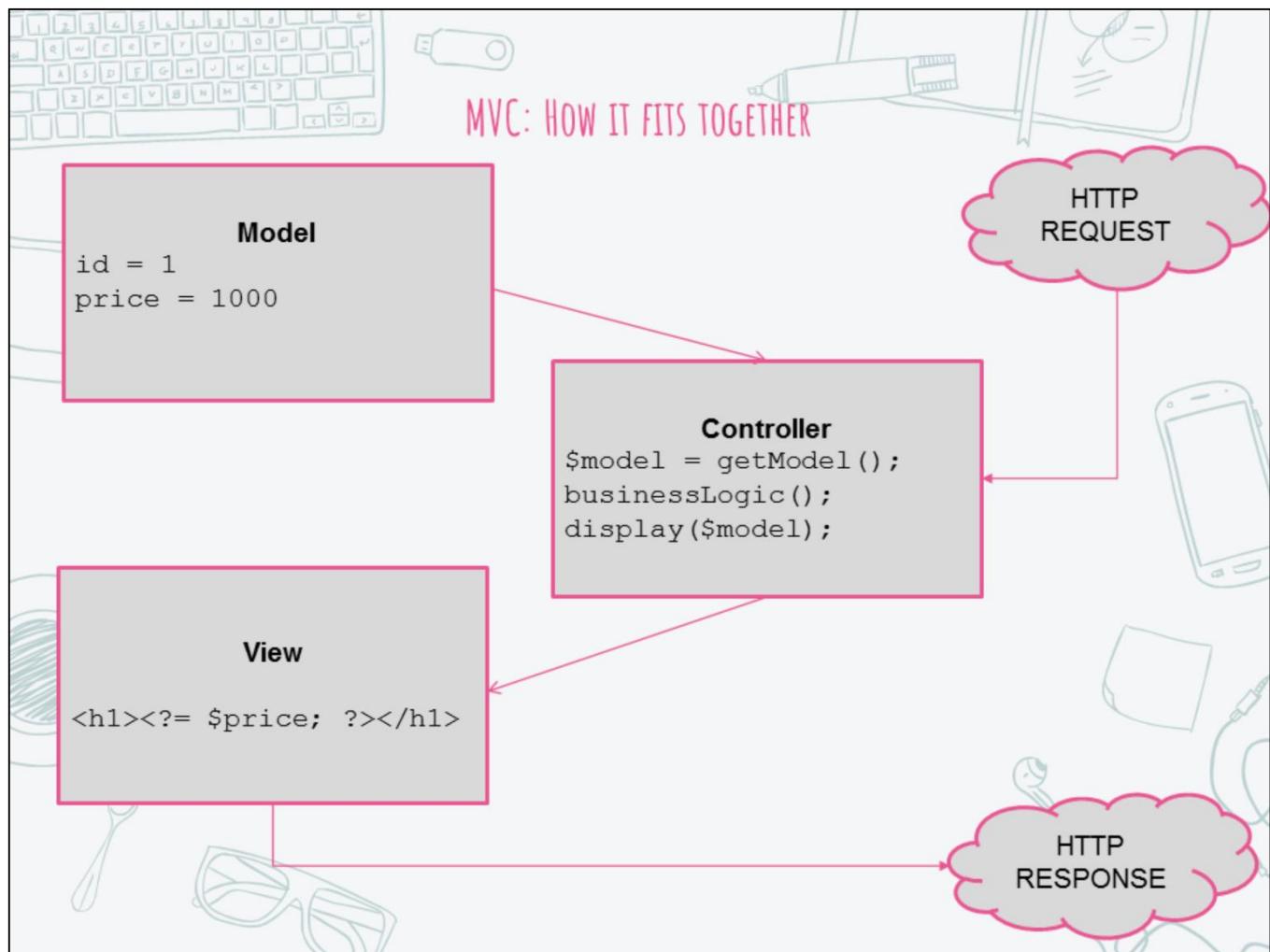
html"s)
  - They might, in addition, be classes related to these files
  - They might be json\_encode()s or XML
- Controllers are typically classes of several methods
  - Each method corresponds to a meaningful operation within your domain
  - E.g. a BlogController has a submitArticle() method

Let's now look more closely at the meaning of each term. The ordinary sense of "Model" means a representation or likeness, as in a model town or a model bridge which has the same structure, and perhaps functionality, of the thing that it is representing. That is something like the meaning here, a model class (e.g. class ItemModel) is taken to have the same structure as, for example, the database table it corresponds to. Model classes may also run SQL queries and be responsible for saving data to the database.

PHP frameworks tend not to have a new class for every template, but a principle 'View' class which can render any template you ask for. The term "View" then refers both to templates (often called "views") and to the logic and functionality which display them.

Finally, controllers are where all the control flow happens (ifs, foreaches, etc). They are the place where your application makes its decisions (e.g. if a person purchases an item a week before Christmas, then they get 10% off). Controller classes are partitioned into several methods (called "actions") which handle specific jobs. A class UserController will for example, have the methods newUser(), editUser().

## MVC: HOW IT FITS TOGETHER

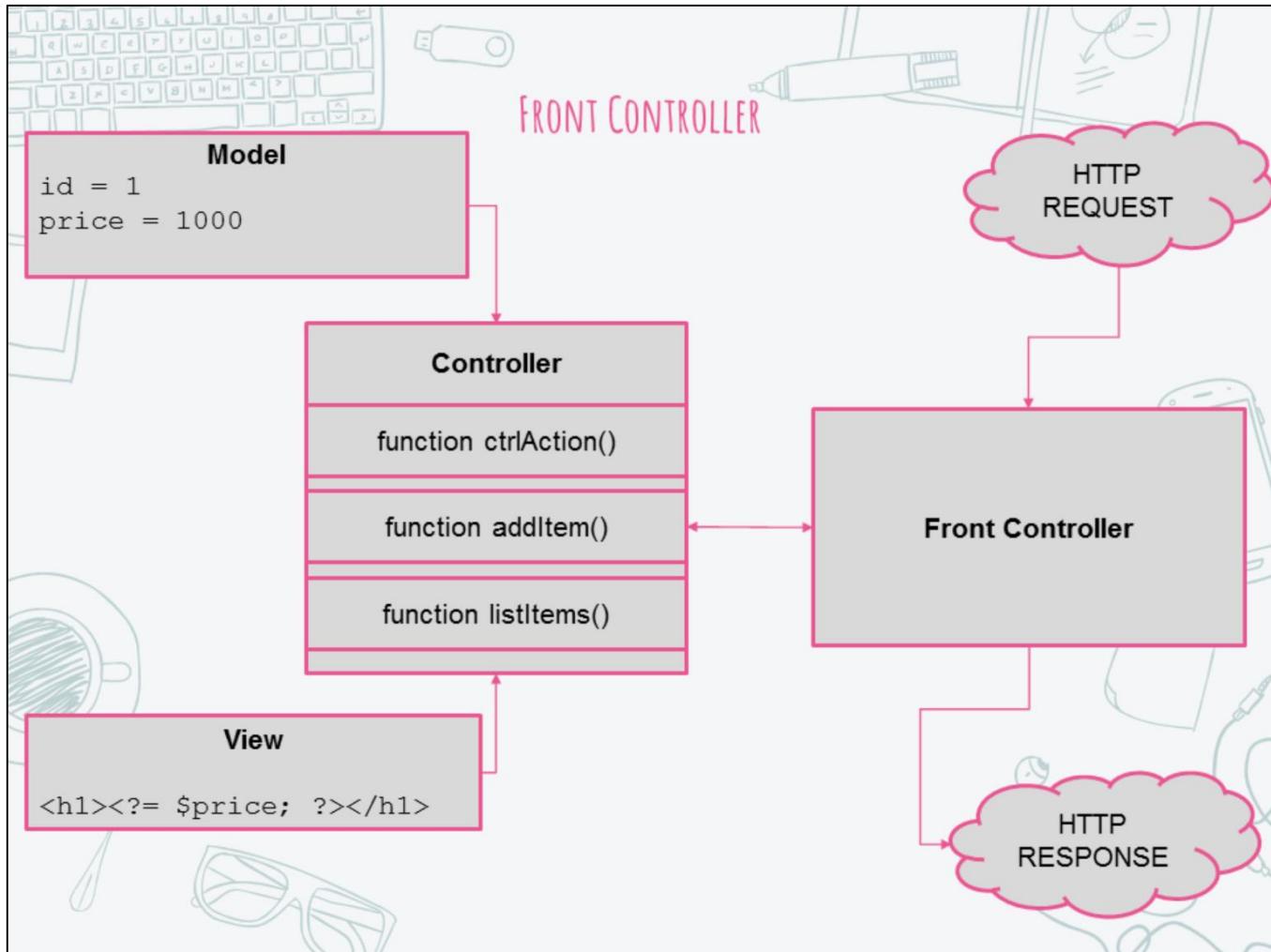


How MVC patterned code relates to the outside world, that is, to HTTP Requests is not defined. MVC is not just a pattern for web frameworks; it can be used for almost any kind of application. Here then frameworks differ more noticeably.

It is usually the case that a particular URL will correspond to a particular action (controller method). In this diagram the incoming request is handed to a controller directly which pulls in a model, makes some decisions and passes the data to a view.

Views and Models are usually highly related in this way. A view's variables – the data it displays on the page – will very often just be a Model. The controller's job is often to connect views and models together, making a few decisions along the way.

In this diagram an HTTP request is passed to a controller which hands off to a view which is sent as an HTTP response.



In this diagram an HTTP Request is sent to a special kind of controller known as a Front Controller, the responsibility of this controller is to organize the request/response process. It does this by using the HTTP request to determine which more specific controller class should be instanced, and which method on the class should be selected.

Read the diagram beginning with the HTTP request which comes in to the font controller class. This class decides to create a new controller object and call a method on it. This method, a controller action, creates a model and joins it together with a view. The controller method then returns this view to the front controller. The front controller then echo's it as the HTTP response.

Code for this entire process will be discussed shortly and the point of this diagram is conceptual at this moment: to layout the key terms (Front Controller, Controller, Action, Model, View) and approximate how they relate to one another. You may wish to return to this diagram as a review for this chapter.

## MVC FOR A SHOPPING CART

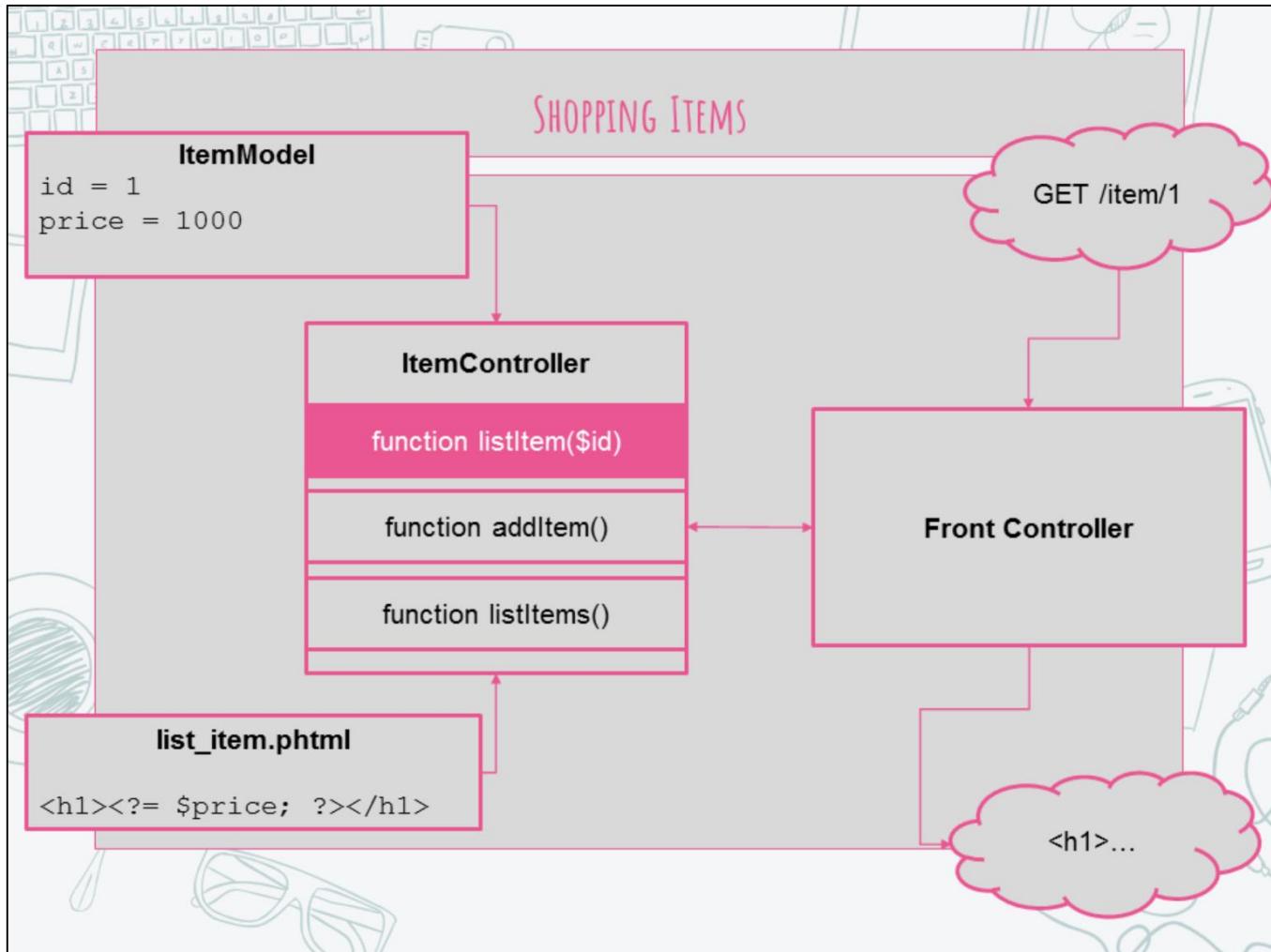
- **ItemController**
  - getItem()
  - getItems()
  - newItem()
  - updateItem()
  - deleteItem()
- **Model**
  - ItemModel: name, price, id
  - UserModel: username, ...
- **View**
  - add\_item
  - list\_items
  - edit\_user...

Let's now translate the implications of an MVC design to a shopping cart website.

What actions will a user need to perform using your site? There will be products or "items" generally, and each will need a page to let an administrator create it/update it/delete it. There will need to be a page displaying the details of a particular item as well as a page which lists all items.

Therefore, at least, the views are clear: add\_item, list\_items, new\_item, etc. If you were to write these views first (called wire-framing) you would see what data you needed to make them work. This would be a strong guide to what the models will look like. In this case an ItemModel will have a \$name and a \$price – and an \$id since you are using a relational database.

The controller is clear given the models and the views. Each view will have its own controller method which will pass the specific models it needs. For example, the updateItem() method will issue a database query and perhaps check the updated prices are sensible using some rules (e.g. price change should never be more than 50%).

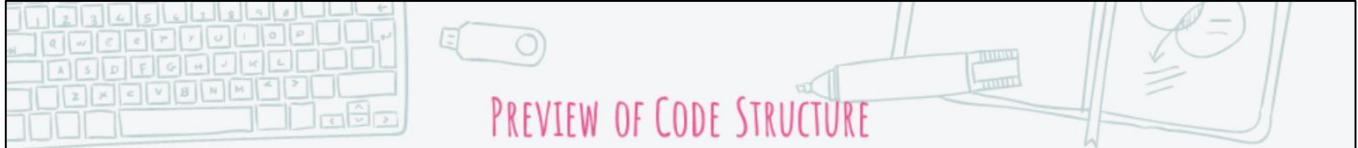


An example of what your MVC pattern might look like for your shopping items is shown in the above diagram.

An HTTP request comes in for the `/item/1` URL endpoint, the front controller identifies that the `/item/` part of the URL means it's the `ItemsController` and the `/1` part tells it the user is looking for a particular item (rather than a list, which might just be `/item`).

The front controller creates the `ItemController` object and calls the `listItem()` method. This method creates an `ItemModel`, querying the database for a row of `id 1` in the process. This model is passed to the `list_item.phtml` view for display. The view output is returned to the front controller which finally echoes it out.

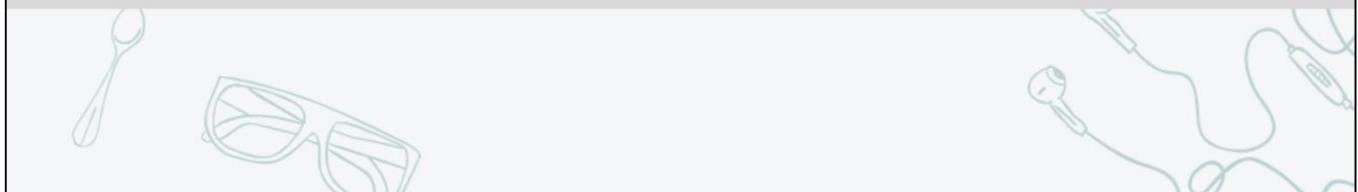
The user receives this echo as html output.



## PREVIEW OF CODE STRUCTURE

### A simplified outline of what MVC looks like

```
class ItemController {  
    public function listItems() {}  
}  
  
class ItemModel {  
    public function __construct($id, $name, $price);  
    public static function getItems();  
}  
  
class HtmlView {  
    public function displayTemplate() {}  
}
```



Here is a sketch of MVC classes implied by the previous diagram. There is an `ItemController` with a `listItems()` method (or "action"). A `Model` with `$id`, `$name` and `$price` fields.

Then there are two additions which might not seem obvious. On the `ItemModel` class there is a static method, this returns instances of `ItemModel` (methods which return new instances of their classes are sometimes called factories).

This method is static because it's not clear we can create a new instance of `ItemModel` without it. How do we pass the `$name` of an item that exists in the database without first querying the database? So we can't use the constructor directly. Rather, we call `getItems()` which performs a database query for us then returns a new instance of the model.

Finally, the `HtmlView` class which is the reusable bit of code that displays arbitrary templates we pass it. If we say `displayTemplate('add_item')` it will return the output for that page.



## DYNAMIC NEW



- The `new` operator can take a string argument, and will `new` the class the string specifies

```
$object = new $classname($parameter);
```

- This is useful for tying URLs to classes

```
class Blog {  
    public function run() {}  
}  
class Forum {  
    public function run() {}  
}
```

```
$url = 'http://example.com/blog';  
$class = end(explode('/', $url));  
  
$area = new $class;  
$area->run();
```

Before we can get into the real code for an MVC framework we need to understand a few more of PHP's features.

Firstly that the `new` operator can take a string argument, rather than just a class name. If its given a string it will assume it's a class name and create an instance of it.

So we can pull 'item' from the url and use it to build a class name we can instantiate.

## DYNAMIC METHOD CALLS

- Recall, for functions you can use a string value of their name to call them

```
$words = 'ucwords';  
$words('hello world');
```

- However since methods "live on" objects, you need to specify both the method name and the object it lives on when you call it

```
$method = [$object, $method]  
$method();
```

```
//or directly  
[$object, $method]();
```

- This can be useful for objects to dynamically call their methods (i.e. when given a string input)

```
$input = 'someMethod';  
[$this, $input](); //calls $this->someMethod()
```

We can also call methods given a string of their name, for example, from the URL.

However, you need to specify both the object and the method name. Build an array with the first element as the object and the second element as the string name of the method. You can then call this array itself and it will call the appropriate function.



## MAGIC METHODS: \_\_TOSTRING

- Adding the method `__toString` to a class will allow objects of its kind to be cast to strings
  - the return value of this method is used as the value of the string cast

```
class Stringable {  
    public function __toString() {  
        return 'some text';  
    }  
}  
  
$str = (string) new Stringable();  
  
//or  
  
echo new Stringable();
```

If you add a `__toString()` method to a class then objects of that class can be cast to a string, and importantly then, can be used with the `echo` statement.

When you try to echo an object with a `__toString()` method, `__toString()` is called and its return value is used.



## MAGIC METHODS: \_\_INVOKE



- Call an object, what happens?

```
$object = new cls();  
$object();
```

- This becomes....

```
$object = new cls();  
$object->__invoke();
```

- Objects with an \_\_invoke method are callable!
  - used to be useful for simulating closures – but we have those now!
  - might be a valid stylistic choice for designing a program

If you add an \_\_invoke() method to a class, objects of that class can be called. That is you can treat objects like they were functions.

If you call an object (i.e. follow it with parentheses) then the \_\_invoke() method will be called instead. You can think of PHP rewriting the parentheses as ->\_\_invoke



## MAGIC METHODS: COMBINING METHODS

```
class LikeAString {  
    public function __toString() {  
        return 'Hello World!';  
    }  
  
}  
  
class LikeAFunction {  
    public function __invoke() {  
        return new LikeAString();  
    }  
}  
  
$fn = new LikeAFunction();  
echo $fn(); //calls __invoke which returns LikeAString  
            //echo itself then calls __toString
```



Combining `__toString()` and `__invoke()` can be quite powerful, since this allows you to call an object and then echo it. This makes objects look a lot like functions and in fact, makes it easy to use functions in place of objects if you want to.

In the above example a class `LikeAString` has a `__toString()` method, meaning it can be cast to a string and echoed. If it is echoed PHP will echo 'Hello World'.

`LikeAFunction` is an invokable, it has `__invoke()`. When objects of class `LikeAFunction` are called they return `LikeAString` objects.

Therefore, if we echo the result of calling a `LikeAFunction` object, we get the string 'Hello World!'. We have simulated a simple procedural lookup and using objects to do so.



## MODEL CLASS

```
class ItemModel {
    private $id;
    private $name;
    private $price;

    public function __construct($id, $name, $price) {
        $this->id = $id;
        $this->name = $name;
        $this->price = $price;
    }

    public static function read($pdo, $id) {
        $sql = "SELECT * FROM items WHERE id = ?";
        $stmt = $pdo->prepare($sql);
        $stmt->execute([$id]);

        $item = $stmt->fetch(PDO::FETCH_ASSOC);

        return new self($item['id'], $item['name'], $item['price']);
    }
}
```

This is more realistic model class code. The constructor allows us to fill in the fields of the class, and we have a public static method that uses this constructor to create new objects and return them. It uses a database call to actually get the data.

The `self` keyword inside classes refers to the class itself. The code "`new ItemModel`" and "`new self`" is interchangeable here however, "`new self`" will continue to work if you were to change the name of the model class – which sometimes happens.

You do not then write "`new ItemModel`" yourself, but rather `ItemModel::read()` which returns an object for you.

Note: This is not the only pattern models can follow. Object-relational mapping or ORMs are also common. An ORM system maps between the relational world of databases and the object oriented world of code, such as PHP.



## VIEW CLASS



```
class HtmlView {  
    const DIR = 'views' . DIRECTORY_SEPARATOR;  
    const EXT = '.phtml';  
  
    private $vars;  
    private $template;  
  
    public function __construct($template, $vars = []) {  
        $this->vars = $vars;  
        $template = self::DIR . $template . self::EXT;  
    }  
    public function __toString() {  
        extract($this->vars);  
        ob_start();  
        include $this->template;  
        return ob_get_clean();  
    }  
}
```

<h1> <?= \$heading; ?></h1>

This view class takes a template name and some variables to display (typically from the model).

It's `__toString()` method includes the template file. However, it's in a region with output buffering meaning any output sent by `include` will be captured in memory. It returns this output using `ob_get_clean()`.

If we echo a view object, we get the output template.



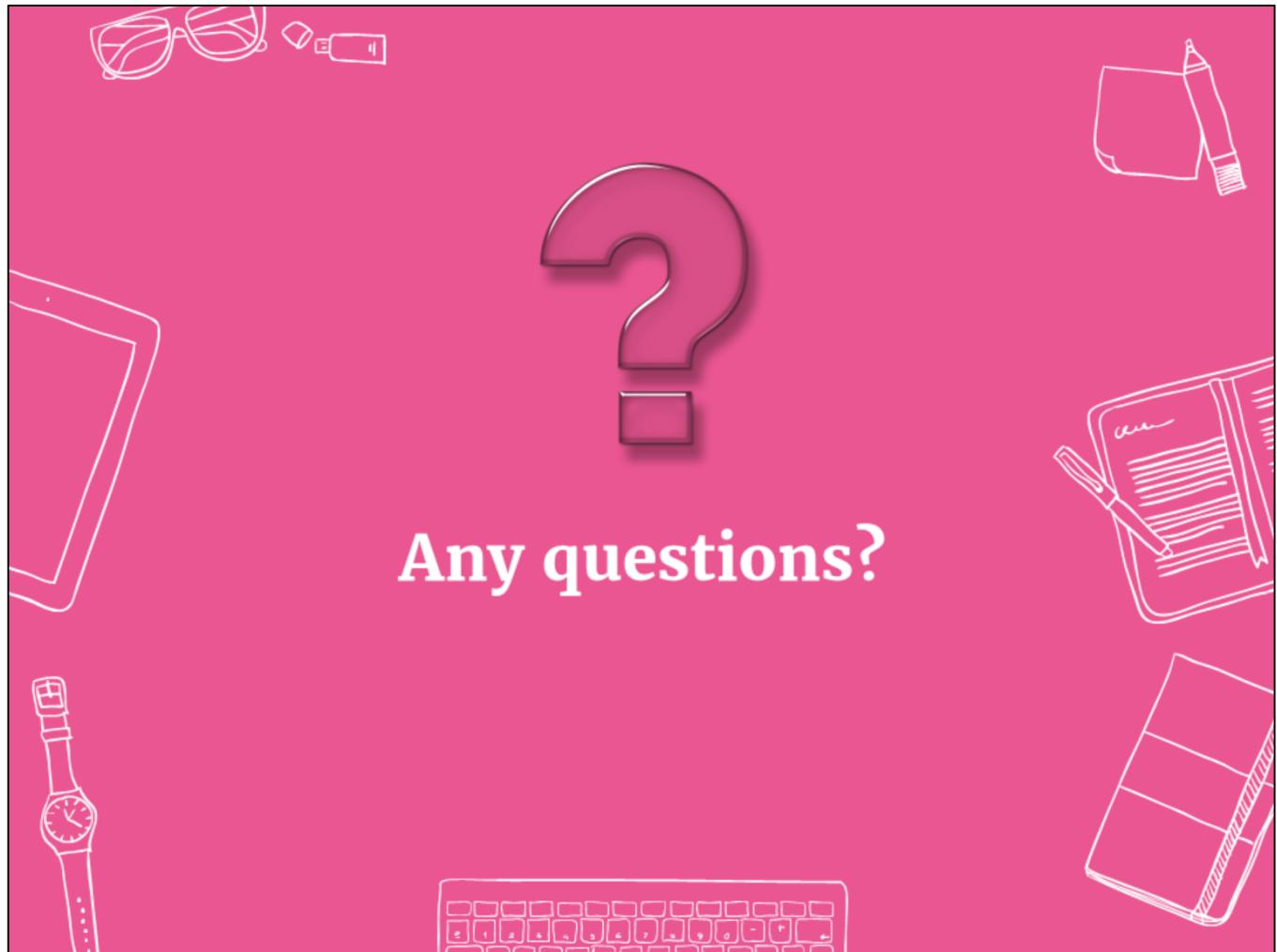
## CONTROLLER CLASS

```
abstract class Controller {  
    private $method;  
  
    public function __construct($method) {  
        $this->method = $method;  
    }  
    public function __invoke() {  
        return [$this, $this->method]();  
    }  
}  
  
class UserController extends Controller {  
    public function index() {  
        return new HtmlView('layout',  
            ['heading' => 'User Index!']);  
    }  
}
```

Finally, controllers are invokable.

When you create a controller you tell it which action (method) to call. When you call the controller object it will call the action you initially specified.

If `index` were called then the controller's `__invoke()` would return a view object, which we know can be echoed.



**Any questions?**



## RESOURCES



### MVC in PHP Articles

<http://www.sitepoint.com/the-mvc-pattern-and-php-1/>

<http://www.htmlgoodies.com/beyond/php/article.php/3912211>



To continue your learning journey, visit:

**QA.COM/TECHIT**



[company/qa-ltd](#)



[QALtd](#)



[QALtd](#)



[qatraining](#)