CPS276

PHP and MySQL

By Scott Shaper

Version 2.0.2

Table of Contents

About This Book

Overview

This book covers the material in the CPS276 class taught at Washtenaw Community College. The code examples in this book are colored in a dark blue:

```
This is a code sample
```

Comments are colored as dark green and uppercase

```
//THIS IS A COMMENT
```

Code Examples:

There are also code examples found throughout this book. The examples can be viewed on the GitHub site located at the web address shown below. The names of the code examples listed in this book will coincide with the code examples on the author's GitHub site.

https://GitHub.com/sshaper/cps276_examples

You can find a working version of all the examples found on the GitHub site at the following web address:

http://198.199.80.235/cps276/cps276_examples/

In many examples, I have something written like cps276_examples/… What this means is you can find the code at

https://GitHub.com/sshaper/cps276_examples

and the live examples are at:

http://198.199.80.235/cps276/cps276_examples/

Chapter 1: Setting Up

DigitalOcean

Introduction

DigitalOcean provides you with a Unix server in which you can install and run all your applications and supporting files. Because this is a server you can access from any computer that is connected to the Internet. You can use any SFTP program to upload files and folders to the server you can also SSH (secure shell) into the server and create all your files and folders directly on the server.

DigitalOcean is not free but it is very inexpensive. They charge for each server instance (known as a droplet) by the hour. The cheapest at this time of writing is 5 dollars a month, or about .007 cents an hour. You can pay via PayPal or a credit card (I use a credit card). You are only charged for the servers you have created. You can create as many servers as you want, and you can destroy them as well. You are only charged for the servers you have created. You will need at least one server instance (droplet) for this class.

Other services do similar setups and you are welcome to use those as well, but I will only be using and discussing DigitalOcean.

Setting up your server

Once you have set up an account on DigitalOcean you can create your own server instance. DigitalOcean calls these instances a droplet. To create your droplet follow these steps

- Login to your DigitalOcean account
- Click the green "Create" button in the upper right and select "Droplets"
- Click on the "Marketplace" tab.
- Click the on "LAMP on 18.04" (numbers may be different for you)

- Select your size (I recommend the 5 dollars a month size)
- Select your region (I recommend New York)
- Enter your droplet hostname. I would call it CPS276
- It should have your project listed as the username you created for DigitalOcean just leave that alone.

It will take DigitalOcean about 1 minute to set up your server. Once they have set it up, you will receive an email with your login information, your droplet name, your IP address, username, and password.

Getting set to log into your server

If you followed along with the above processes, you now have a server hosted by DigitalOcean. What you have created is a server instance of Ubuntu. Ubuntu is a distribution of a Linux operating system. An operating system is software that supports a computer's basic functions, such as scheduling tasks, executing applications, and controlling peripherals. For example, Windows 10 is an operating system.

Because this is a server there is no GUI (graphic user interface), meaning it is not visual like Windows or Mac or even a desktop version of Linux. Everything is done via the command line. It will take some getting used too but this is the way things are done in the real world and something you will be expected to understand.

If you are on a Mac or Linux operating system, you will use the terminal window. If you are on a PC (Windows) you will need to download Putty (or use PowerShell). I recommend the Putty.exe listed under "For Windows on Intel x86" on the Putty website. Putty is just an executable application so you can move it to your desktop if you want. Putty is also installed on all the lab and library PCs.

If you don't want to download Putty you can optionally open your PowerShell console (enter PowerShell in your windows search box) and use that (you want to select Windows PowerShell (x86)).

Please watch the video located on Blackboard "Setting Up Your Lamp Stack (Ubuntu Instance) 00:08:35" for instructions on how to do this.

Mac/Linux Login

After you get your email from DigitalOcean containing your login information.

Open the terminal window and enter the following at the command prompt

Note: If you first create an account with DigitalOcean your username will be "root."

ssh root@your ip address

Next, you will enter your password (passwords are case sensitive). If you are logging into your DigitalOcean virtual server for the first time they will make you change your password from the one they sent you.

Windows PowerShell Login

Open the PowerShell console window and enter the following at the command prompt

Note: If you first create an account with DigitalOcean your username will be "root."

ssh root@your ip address

Next, you will enter your password (passwords are case sensitive). If you are logging into your DigitalOcean virtual server for the first time they will make you change your password.

Windows Putty Login

Open putty and follow the steps below

- Under the category window select the session link (should be selected by default)
- Under "Hostname (or IP address) enter your IP address
- Make sure the ssh radio button is selected
- Click open
- A screen will open asking you to login, enter "root" your default username

It will then ask for your password (passwords are case sensitive). If you are logging into your DigitalOcean virtual server for the first time they will have you change your password.

Adding a User with Sudo Privileges

It is not a good practice to login as root. To create a user with "sudo" privileges follow these instructions

Setting up MySQL

MySQL is the database you will be using for this class. DigitalOcean has already created it for you but you need to set it up so you can login using "root" and a password you set. Please watch the video located on Blackboard "Setting Up Your Lamp Stack (Ubuntu Instance) 00:08:35" for instructions, starts around the 4-minute mark.

Using FileZilla with DigitalOcean

Secure File Transfer Protocol (SFTP)

To see any of the web pages you created in PHP you will need to upload the files to the server. In other words, the files must be on the server to work.

When uploading files to a server you commonly use one of two protocols. File Transfer Protocol (FTP) and Secure File Transfer Protocol (SFTP). We will be using SFTP because it is more secure. In the next section, I will introduce you to FileZilla which is an application you can use to put files on the server. FileZilla is a good tool, but I recommend you also set up your editor to automatically upload files to the server when they are saved in your editor. To do it manually each time is a real waste of time. I have videos on Blackboard on how to do this with VSCode (look under VSCode videos).

FileZilla layout

If you are working on your computer, you will need to download a copy of FileZilla. FileZilla is installed on the classroom computers. To get a copy of FileZilla go to the FileZilla website, click the button that says, "Download FileZilla Client", follow the instructions for installing FileZilla.

NOTE: If you are using a PC go to Ninite.com and download FileZilla from there.

When you open FileZilla, you will see several screens. They are as follows:

Top showing four text boxes and a "quick connect" button - This is where you enter your connection information.

The area directly under top - This will display text related to the connection process to the server.

Top left under area - This is your local machine (your computer) main folders.

Bottom left - This is your local files and folders for the main folder you clicked above.

Top right under area - This is your remote site (the server) main folders.

Bottom right - This is your server files and folders for the main folder you clicked above.

Logging into FileZilla

To login to FileZilla enter the following into the boxes at the top

1. **Host** - Server IP Address (this will be your DigitalOcean IP Address)
2. **Username** - Your username (whatever you set up with DigitalOcean)
3. **Password** – Your password (whatever you set up with DigitalOcean for the username you entered)
4. **Port** - Enter "22" for SFTP

Click the quick connect button. On the right-hand side, you should see a directory structure, you want to navigate to the folder in which you need to put all your files. That folder is located at “/var/www/html”.

Login Problems

If you cannot login to the server and it says access denied. Make sure you entered your information correctly and that your host is the IP address supplied from DigitalOcean something like 167.90.345.23

Uploading to FileZilla

The left-hand side of FileZilla is your local files (files on your computer). To upload a file just drag it from the left side into the appropriate folder on the right side. The better way to put files on the server is to use an editor like VSCode and have the files automatically uploaded to the server when you save a file.

Creating folders in FileZilla

Right-click on the folder you want to put the new folder into (the parent folder) and select “New Directory” then enter the folder name.

Renaming folders/files in FileZilla

To rename a file or folder just right-click it and select “Rename”.

Deleting folders/files in FileZilla

Right-click on the folder or file you want to delete and select “Delete”.

Downloading a file/folder from FileZilla

To download a file just drag it from the right side to the appropriate area on the left side.

Ubuntu

Introduction

Ubuntu is a Linux distribution (a distribution is an operating system based on Linux). If you followed the video for setting up DigitalOcean you have a server that is running Ubuntu. Once you SSH into your server you will be in the "root" directory. If you enter the following:

```
cd /
ls
```

cd / puts you at the root level (not to be confused with the root directory) where you can see all the directories. Those directories are the various folders that are created and used by Ubuntu. Below is a table showing the default directories.

IMPORTANT NOTE: A directory is the same thing as a folder on your computer. When a folder is on a server it is sometimes called a directory.

Directory	Content
bin	Essential command binaries
boot	Static files of the boot loader
dev	Device files
etc	Host-specific system configuration
home	User home directories
lib	Essential shared libraries and kernel Chapters

Directory	Content
media	Contains mount points for replaceable media
mnt	Mount point for mounting a file system temporarily
proc	Virtual directory for system information (2.4 and 2.6 kernels)
root	Home directory for the root user
sbin	Essential system binaries
sys	Virtual directory for system information (2.6 kernels)
tmp	Temporary files
usr	Secondary hierarchy
var	Variable data
srv	Data for services provided by the system
opt	Add-on application software packages

Unix Commands

Because you will be working on a Unix based system you will need to learn some basic Unix commands to navigate around the server

Command	Description
/	This separates each subdirectory when attempting to navigate. For example, if your cps276 directory was at the root level and you had a subdirectory inside of cps276 named "projects" you would enter cd /cps276/projects/
../	The code to the left will go up one directory level. cd ../ If you want to go up two directory levels you enter cd ../../
cd	Stands for "change directory" you use it when you want to navigate to another directory. In Ubuntu entering cd~ will take you your directory within the home directory. For example, if I enter cd and then enter pwd, the directory will be /home/sshaper/
cp	Copies a file from one location to another. The following code copies filename1.txt to filename2.txt cp filename1 filename2
ctrl + c	Stops the current command you just entered, if you enter ctrl + c while the command is still executing.

Command	Description
ls	Shows a listing of all subdirectories of the directory you are in. ls -l will give a detailed list. ls -la Will give a detailed list and show all hidden files (those files that start with a period).
mkdir	This will make a new directory. If you wanted to create a directory named "projects" you would write mkdir projects. Note: If you just include the directory name it will create the directory in whatever directory you called it in.
mv	Moves a file from one location to another. mv original-location new-location. You can also copy a file. The following code renames filename1 to filename2 mv filename1 filename2
pwd	Entering this command shows you the directory you are in from the root level.
rm	Will delete a file.
rm -r or rm -rf	Deletes a directory’s sub-folders and files rm -rf directory path/*.
rmdir	Will delete an "empty" directory
sudo	Sudo allows a user (with sudo privileges) to run a command as root. A root user can do anything but is dangerous because you can damage Linux. It is a best practice to create a different user with sudo privileges and then use the keyword sudo.

Command	Description
tab	If you start to write a file or folder name and press tab, Ubuntu will fill in the rest of the name (huge time saver)
touch	Creates or updates a file
up arrow key	Goes up the list of previously entered commands

Updating Ubuntu

Many times, after logging into Ubuntu you will see a list of packages that should be updated. The following commands will update your system.

sudo apt-get update Fetches the list of available updates

sudo apt-get upgrade Strictly upgrades the current packages

sudo apt-get dist-upgrade Installs updates (new ones)

chmod

On Linux and other Unix-like operating systems, there is a set of rules for each file which defines who can access that file, and how they can access it. These rules are called file permissions or file modes. The command name chmod stands for "change mode", and it is used to define the way a file can be accessed.

In general, chmod commands take the form

```
sudo chmod [options] permissions filename
```

One of the more common ways to set file permissions is to use "octal permissions notation." For example, consider the following code

```
sudo chmod 754 myfile
```

Here the digits 7, 5, and 4 each individually represent the permissions for the user, group, and others, in that order. Each digit is a combination of the numbers 4, 2, 1, and 0:

4 Stands for read

2 Stands for write

1 Stands for execute

0 Stands for "no permission"

Examples of permissions

chmod code	meaning
chmod 400 myfile	The user will have read permissions but not any groups or other
chmod 444 myfile	The user, group, and others have read permissions
chmod 664 myfile	The user and group have read and write permissions, other has read-only permissions
chmod 706 myfile	The user has read, write and execute permissions, the group has no permissions, and other has read and write permissions
chmod 777 myfile	User, group, and other have read-write and execute permissions

chmod code	meaning
chmod -R 777 mydirectory	All files in mydirectory have set user, group, and other to read write and execute permissions

The following Computer Hope website has more information on chmod.

Vim

You may want to create/edit your files directly on the server. To do that you need some sort of built-in editor. Vim is an editor that is included with Ubuntu. Vim is an easier version of Vi which is also available. You can use your regular desktop editor if you want to and just SFTP the files to the server.

To edit a file just enter:

```
sudo vim filename
```

Below is a table of some basic vim commands.

Command	Description
i	Switch from command mode to insert mode
ESC	Switch from insert mode to command mode
:q	Quit vim
:wq	Write to and save the file then quit vim

Getting Started with Git

What is Version Control

Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later. Version control is widely used where teams of people are working on the same project. Understanding version control is a must in today's world as I can guarantee you will be using one. Using a VCS also generally means that if you screw things up or lose files, you can recover them. Also, you get all this for very little overhead.

For this class, we will be using Git, which is one of the most popular systems.

How is Git Different

In a Distributed Version Control Systems (such as Git, Mercurial, Bazaar, or Darcs), clients don’t just check out the latest snapshot of the files; rather, they fully mirror the repository, including its full history. Thus, if any server dies, and these systems were collaborating via that server, any of the client repositories can be copied back up to the server to restore it. Every clone is a full backup of all the data.

Furthermore, many of these systems deal well with having several remote repositories they can work with, so you can collaborate with different groups of people in different ways simultaneously within the same project. This allows you to set up several types of workflows that aren’t possible in centralized systems, such as hierarchical models.

More information can be found at https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

In this class, we will be using git to put our files in a private repository so they can be shared with the instructor. Many times, when evaluating student assignments or helping students solve problems it is important for the instructor to see the student's code. Git (actually GitHub) allows the instructor to do that.

Installing Git

Instructions for installing Git can be found at https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

Git on the command line

There are a lot of different ways to use Git. There are the original command-line tools, and there are many graphical user interfaces of varying capabilities. For this lesson, we will be using Git on the command line. For one, the command line is the only place you can run all Git commands — most of the GUIs implement only a partial subset of Git functionality for simplicity. If you know how to run the command-line version, you can probably also figure out how to run the GUI version, while the opposite is not necessarily true. Also, while your choice of graphical client is a matter of personal taste, all users will have the command-line tools installed and available. Most importantly, in the industry, you will be using the command line so you need to get used to it.

I expect you to know how to open Terminal in Mac or Command Prompt or Powershell in Windows and use git via the command line.

Git and GitHub

Git is a revision control system, a tool to manage your source code history. **GitHub** is a hosting service for Git repositories. So, they are not the same thing: Git is the tool, GitHub is the service for projects that use Git.

Creating a GitHub Account

Creating a GitHub account is very straight forward and 100% free as long as your repositories are public (repositories are where you store your code), or private with only three collaborators. To create your GitHub account go to the following page and signup https://github.com/.

Creating a repository (for the first time)

Once you have created a GitHub account and git installed on your local machine, you can login to your account and create a repository.

1. In the upper right-hand corner, you will find a "plus" icon click on that and click "new repository".
2. In repository name write the name of the repo let's call it "pratice_repo".
3. Click the "create repository" button and your repository will be created.
4. You will be taken to a setup page. On that page they will give you some code for creating your repository on your local machine and connecting it to GitHub, using the command line. GitHub recommends a README, LICENSE, and gitignore file. I will get into those later but let's start with the basics.
5. On your local machine create a folder named "practice_repo"
6. Go into the folder and enter git init. This will initialize git for that folder.
7. Within that folder create the file README.md (LNX touch README.md)(PWS New-Item README.md)
8. Add the README.md file to git - git add README.md
9. Commit the README.md file to git - git commit -m "This is my first commit". You want to use the -m flag and follow that with a message in quotes. You want to add a message for every commit you do.
10. Set up the remote path to your git repository - git remote add origin https://GitHub.com/sshaper/practice_repo.git
11. Upload to git - git push -u origin master

Cloning a repository

Optionally you can clone a repository from GitHub by doing the following.

1. Go to the directory you want the cloned directory to be put into.
2. git clone https://GitHub.com/sshaper/practice_repo.git (you can get the address of any GitHub repository by clicking the clone or download button).

Git Basics

Git status, add, commit and push

In the previous lesson, we saw three main git commands (add, commit, and push). Here we are going to describe what each one means and address the "status" command.

git status

Displays the paths that have differences between the index file and the current HEAD commit, paths that have differences between the working tree and the index file, and paths in the working tree that are not tracked by Git (and are not ignored using gitignore). The first is what you would commit by running git commit; the second and third are what you could commit by running git add before running git commit.

More information can be found at https://git-scm.com/docs/git-status.

```
git add . //THE DOT MEANS YOU ARE ADDING EVERYTHING OTHERWISE YOU NEED TO NAME THE FILES/FOLDERS YOU WANT TO ADD

TO ADD MULTIPLE FILES YOU WOULD LIST THEM SEPERATED BY A SPACE.

GIT ADD FILE1.TXT FILE2.TXT FILE3.TXT
```

The above command updates the index using the current content found in the working tree, to prepare the content staged for the next commit. It typically adds the current content of existing paths as a whole, but with some options, it can also be used to add content with only part of the changes made to the working tree files applied or remove paths that do not exist in the working tree anymore.

The "index" holds a snapshot of the content of the working tree, and it is this snapshot that is taken as the contents of the next commit. Thus, after making any changes to the working tree, and before running the commit command, you must use the add command to add any new or modified files to the index.

This command can be performed multiple times before a commit. It only adds the content of the specified file(s) at the time the add command is run; if you want

subsequent changes included in the next commit, then you must run git add again to add the new content to the index.

The git status command can be used to obtain a summary of which files have changes that are staged for the next commit.

The more common add commands and flags are listed below, for a complete listing go to https://git-scm.com/docs/git-add

Command	Description	Example
<pathspec>	Files to add content from. Fileglobs (e.g. *.c) can be given to adding all matching files. Also, a leading directory name (e.g. dir to add dir/file1 and dir/file2) can be given to update the index to match the current state of the directory as a whole (e.g. specifying dir will record not just a file dir/file1 modified in the working tree, a file dir/file2 added to the working tree, but also a file dir/file3 removed from the working tree. Note that older versions of Git used to ignore removed files; use --no-all option if you want to add modified or new files but ignore removed ones.	`git add Documentation/\*.txt`
-A, --all, --no-ignore-removal	Update the index not only where the working tree has a file matching <pathspec> but also where the index already has an entry. This adds, modifies, and removes index entries to match the working tree. If no <pathspec> is given	`git add -A`

Command	Description	Example
	when -A option is used, all files in the entire working tree are updated (old versions of Git used to limit the update to the current directory and its subdirectories).	

git commit

Stores the current contents of the index in a new commit along with a log message from the user describing the changes.

If you make a commit and then find a mistake immediately after that, you can recover from it with a git reset.

The more common commit commands and flags are listed below, for a complete listing go to https://git-scm.com/docs/git-commit

Command	Description	Example
-m "msg", --message="msg"	Use the given as the commit message. If multiple -m options are given, their values are concatenated as separate paragraphs. The -m option is mutually exclusive with -c, -C, and -F.	`git commit -m "my first commit"`
-am "msg"	Shortcut where you can add and commit in one line.	`git commit -am "my first commit"`

git push

The above code updates remote refs using local refs, while sending objects necessary to complete the given refs.

When the command line does not specify where to push with the <repository> argument, branch.*.remote configuration for the current branch is consulted to determine where to push. If the configuration is missing, it defaults to origin.

When the command line does not specify what to push with <refspec>... arguments or --all, --mirror, --tags options, the command finds the default <refspec> by consulting remote.*.push configuration, and if it is not found, honors push.default configuration to decide what to push.

When neither the command-line nor the configuration specifies what to push, the default behavior is used, which corresponds to the simple value for push.default: the current branch is pushed to the corresponding upstream branch, but as a safety measure, the push is aborted if the upstream branch does not have the same name as the local one.

You can view more git push commands at https://git-scm.com/docs/git-push

Updating a repository

Once you have created and done your first push to a repository, updating the repository is very simple. I do the following:

1. git status - Check the files that are to be uploaded, deleted
2. git add -a - Add all the files
3. git commit -m "my next commit" - commit with a message
4. git push or git push origin master

Example:

The following will update our current repository. Start within the repository directory on your local machine.

1. Create a new file (LNX touch test.txt)(PWS New-Item text.txt)
2. git status - Shows the testfile.txt is currently untracked

3. git add -A or git add .
4. git status - Shows the testfile.txt is to be committed. This step is not necessary but is used for this example to show how the status changes
5. git commit -m "added testfile.txt" - Commits the testfile.txt.
6. git status - Now it states, "Your branch is ahead of 'origin/master' by 1 commit.", meaning we must push that file to the master repository. This step is not necessary but used for this example to show how the status changes
7. git push - May ask for a username and password to upload to the git account.

Git Branching and Merging

Git branching

In the previous lessons, we created a repository and then we updated that repository. What we did was update the master repository, which is fine and good if that we are the only developer. However, that is not the case in the real world. Projects have multiple developers that work on parts of the project. If everyone updated the master, then we would have a huge problem with developers overwriting each other.

With git, we can create branches. A branch is a copy of the master under another name that can be updated. To create a new branch and use that branch we can do the following:

1. git branch dev_test - Creates a branch with the name of dev_test
2. git branch - Displays all the branches for that git repository (the branch you are on will have a * next to it)
3. git checkout dev_test - Allows you to use/update the dev_test branch

Now that we have another branch, we can add files and update files on that branch.

1. (LNX touch dev_test.txt)(PWS New-Item dev_test.txt)- Creates a dev_test_file.txt file in the dev_test branch only.
2. git add -A - Adds the new file to the dev_test repo
3. git commit -m "added dev_test_file.txt" - Commits the file to the branch
4. git ls-files - Shows all the files of the dev_test branch. Notice the dev_test_file.txt file.
5. git checkout master - Switching to the master.
6. git ls-files - Shows all the files of the master branch. Notice there is no dev_test_file.txt file.

The above example (when done) clearly shows that the branch of dev_test contains an extra file that the master does not. To push the branch, you can do the following:

1. git checkout dev_test

2. git push --set-upstream origin dev_test - Adds the dev_test branch to practice_repo repository on GitHub.

Git Merging

Eventually, you will need to add the branch back to the master. To do this you need to do the following.

1. git checkout master - Get the master.
2. git merge dev_test - Merges dev_test to the master branch
3. git push - Pushes to the practice_repo repository on GitHub.

Deleting the branch

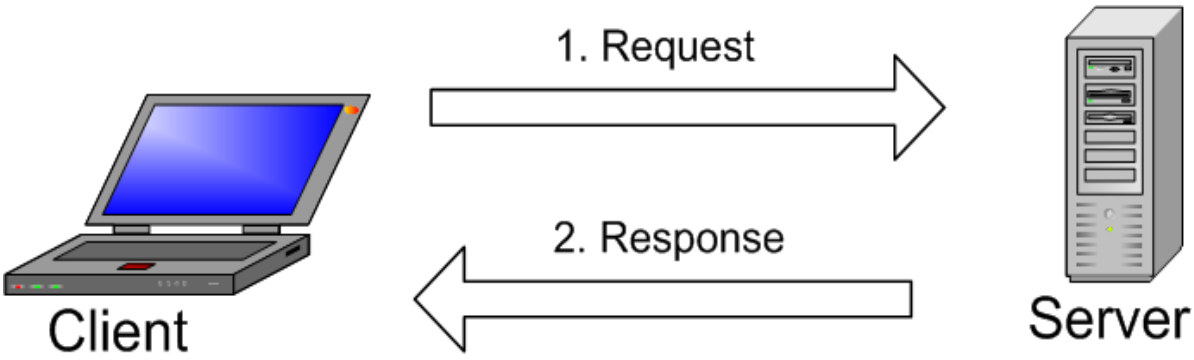
1. git branch -d dev_test - Deletes the dev_test branch (this is optional). This deletes the branch locally.
2. git push origin --delete dev_test (this deletes the branch remotely)

Chapter 2: How the Internet Works

Client and Server Interactions

You have a device (computer, tablet, smartphone, etc) that has access to the Internet, this device is known as the **client**. When you enter a web address (otherwise known as Uniform Resource Locator (URL)) into the client's browser address bar and hit "enter", the client device will send a "request" to a "server". A **server** is a computer that stores the file(s) that can be requested from another computer via the Internet (it can store other files as well but for our purposes, they will be website related files). The server will find the requested file, on the device requested and send a response or "serve" it back to the client device.

The image below shows a typical client-server interaction.

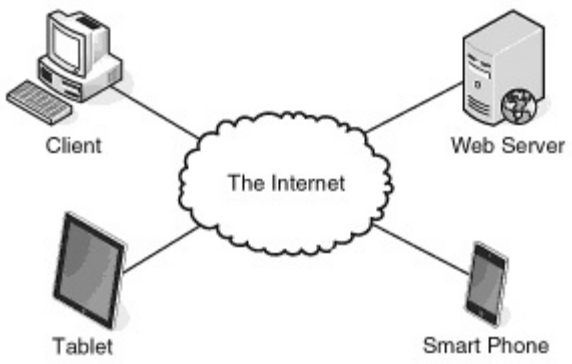


How a web page address works

If you enter the URL "**space.wccnet.edu/~sshaper/courses/web110/index.php**" into the browser address bar it will send a request to the server located at **space.wccnet.edu**. Space.wccnet.edu is the domain name of the server we are sending our request to. Then the server will look for the file "**index.php**", which will be located in the **sshaper/courses/web110** sub-directory. When the server finds the file, it will send a response back which contains that file.

Connected Devices

The following diagram shows various clients and a web server connected via the Internet.



Networks

A network is a system that allows clients and servers to communicate. The Internet, in turn, is a large network that consists of many smaller networks. In the above diagram, the "cloud" represents the network or Internet that connects the clients and servers.

In general, you don't need to know how the cloud works. But you should have a general idea of what's going on.

What are a modem and a router

The modem receives information from your ISP (Internet Service Provider) through the phone lines, optical fiber, or coaxial cable in your home (depending on your service provider) and converts it into a digital signal. The router's job is to push this signal out to connected devices, either through wired Ethernet cables or Wi-Fi, so that all of your devices can hop on board and access the Internet. Your router and ISP can't communicate directly because they speak different languages or rather, they transmit different signal types which is why the modem's role as a translator is so important.

Chapter 3: HTML Elements

HTML Elements and Attributes

HTML Elements

An HTML element is the markup used to create parts of a webpage. For example, the following:

```
<p>This is a paragraph element</p>
```

Would render in a web browser as:

This is a paragraph element

The p element tells the browser what type of element to create in the case of a paragraph element. Notice also how the element is written opens the element (known as the starting tag) and terminates the element (known as the ending tag). What defines the type of element is what character(s) they have between the < and >. Like <h1> (heading 1) and <article> article element. As we progress through this chapter and the next, we will be looking at a lot of different elements.

Attributes

Attributes help to further enhance the element. They are added to the starting tag of an element and are written in a key-value arrangement as shown.

```
<p id="someid" class="someclass" >This is a paragraph element with attributes</p>
```

Not all attributes can be used with all elements. As we progress through this chapter and the next you will see attributes that go with certain elements. What is important is you understand how to write an attribute.

Block-level Elements Defined

Block-level element

Block-level elements are elements that occupy the entire horizontal (left to right) space within their parent element (the parent element is the element that contains them). Many times, the parent element is the body element, which means that the block-level element extends to the edges of the browser window. Once a block element is closed (terminated) anything you write after that element will appear on a new line.

The two characteristics of a block-level element are:

1. Extends the width of its parent element unless width has been applied via CSS
2. Once terminated anything you write after that will be on a new line

block-level elements can contain inline elements. Some block-level elements can contain other block-level elements, while other block-level elements cannot. For example, h1 and p elements cannot contain other block-level elements. However other elements like the divisional, header, address elements, etc can.

NOTE: In the next few lessons I will be categorizing various block-level elements into specific categories to make them clearer. The categories will be **structure**, **text, and list**. These categories are something I did to more clearly present the information and are not an official categorization of block-level elements.

Additional Reading

This area provides an additional reading on the listed topics.

Block-level Elements

Block Level Structure Elements

Introduction

Some block-level elements just contain text (text only) while others can contain other block-level elements (structure) and text, others that are more for lists. In this lesson, we will look at the main structure block-level elements

What we mean by structure is they are the building blocks for our web pages.

Header

A header element can be the main header that is placed at the top of every page on the website. Or as the header for an individual article or section within a page.

Do not confuse the header element with the heading (h1) element.

The header element is written as:

<header>...</header>

Nav

The nav element represents an area for navigation, usually the main nav. There can be multiple nav elements on a page.

The nav element is written as:

<nav>...</nav>

Main

The main element contains the main part of the web page. It is an exact analog of ARIA's role="main" and is designed to show screen readers and assistive technologies exactly where main content begins, so it can be a target for a "skip links" keyboard command, for example.

It should be used once per page.

The main element is written as:

<main>...</main>

Footer

A footer element is the main footer that is placed at the bottom of every page on the website. Or as the footer for an individual article or section within a page.

The footer element is written as:

<footer>...</footer>

Article

Represents a section of a page that consists of a composition that forms an independent part of a document, page, or site. This could be a forum post, a magazine or newspaper article, a Weblog entry, a user-submitted comment, or any other independent item of content.

The article element is written as:

<article>...</article>

Section

The section element groups related content together, and typically each section will have a heading.

For example, on a homepage, there may be several section elements to contain different sections of the page, such as the latest news, top products, and newsletter group.

Because the section element groups related items together, it may contain several distinct article elements that have a common theme or purpose.

Do not use it as a generic styling container that is what we use the div element for.

The section element is written as:

<section>...</section>

Aside

The aside element has two purposes, depending on whether it is inside an article element or not.

When the aside element is used inside an article, it should contain information that is related to the article but not essential to its overall meaning. For example, a pull quote or glossary might be considered as an aside to the article it relates to.

When the aside element is used outside of an article element, it acts as a container for content that is related to the entire page. For example, it might contain links to other sections of the site, a list of recent posts, a search box, or recent tweets by the author.

The aside element is written as:

```
<aside>...</aside>
```

Division Element

A division element denotes areas of a document, they make up a web page. They are generally considered to be high-level elements that contain headings and paragraphs.

Divisions are a generic block-level structured element and can be used when any of the others would not suffice.

The divisional element is written as:

```
<div>...</div>
```

Additional Reading

This area provides additional reading on the listed topics.

Header Element

Nav Element

Main Element

Footer Element

Article Element

Section Element

Aside Element

Div Element

Block Level Text Elements

Introduction

Some block-level elements just contain text (text only elements) while others can contain other block-level elements (structure elements) and/or text, others are more for lists. In this section, we will look at the text-only block level elements

Headings

Heading elements represent the headings and subheadings in your document structure, there are six of them. The heading 1, coded as <h1></h1>, is the highest-level heading and browsers render it the largest. Heading 6, coded as <h6></h6>, is the lowest-level heading and browsers render it the smallest. Using headings appropriately is crucial in web design. Search engine spiders give greater importance to their content. Adaptive technology can notify users that they have encountered a heading, as well as allow users to quickly move between headings in a document

Always start with the heading 1 and progress down level by level: Do not skip levels. For example, do not jump from heading 1 to heading 4. After heading 1 there is a heading 2 and so on.

Skipping heading levels is incorrect from a document structure perspective. It is also potentially an accessibility issue. Users employing screen readers can access a list of headings at each level and progress from one heading to the next. If you skip a level of headings and the user accesses that level, they will hear that none are available. The natural assumption is that lower-level headings do not exist, so the user stops checking. The lower-level headings are effectively "lost".

```
<h1>This is a heading1 example</h1>
<h2>This is a heading2 example</h2>
```

Paragraphs

Paragraphs are coded as . Paragraphs should be used for marking up blocks of text content, although there will be times when you use a paragraph element around a word or a short text segment. Paragraphs always have a line of space above and below them. This space is called a margin. While you might be tempted to nest a paragraph inside a heading or vice versa, that is strongly discouraged. From a document structure perspective, it is always best to have them follow one another and not be nested within each other.

```
<p>This is a paragraph example</p>
```

Address Element

The address element has been around since the HTML3 spec was drafted in 1995, and it continues to survive in the latest drafts of HTML5. But nearly fifteen years after its creation, it's still causing confusion among developers. The most common misconception about the address element is that it should be used to mark up any old address.

Address elements coded as <address></address> represent the contact information for its nearest article or body element ancestor. If that is the body element, then the contact information applies to the document.

The address element must not be used to represent arbitrary addresses (e.g. postal addresses) unless those addresses are the relevant contact information. (The p element is the appropriate element for marking up postal addresses in general).

There are no specialized attributes for this element. Browsers typically render the address on a new line and italicize the content. Content following the address element appears on a new line.

Be careful not to nest this inside block level text elements, as a validation error will result, and rendering issues may arise. Nesting inside block level structure elements is fine.

NOTE: The address element can contain a paragraph element.

```
<div>Contact Scott Shaper at: <address>sshaper@test.com</address></div>
```

Blockquotes

The Blockquote element is used to indicate the quotation of a large section of text from another source. Using the default HTML styling of most web browsers, it will indent the right and left margins both on the display and in printed form, but this may be changed via Cascading Style Sheets (CSS). This structural element is used for block quotations is coded as <blockquote></blockquote>.

The cite attribute has no visual effect in ordinary web browsers but can be used by search engines to get more information about the quotation.

Blockquotes can contain paragraph elements

Blockquote Attributes			
Attribute	Usage and Effect	Values Accepted	Default
Global Attributes	Attributes that can be used for all elements	N/A	N/A
cite	Contains the URL of the cited text.	Text (usually a web address)	

```
<p>A direct quotation from web developers:</p>
<blockquote cite="http://www.examples.com">
   We love HTML5!
</blockquote>
```

Pre Element

Text that is pre-formatted is surrounded by <pre></pre> elements. What makes pre-formatted text unique is that white space inside the elements is preserved, rather than being collapsed down to a single space maximum between characters (as in regular code rendering). Pre-formatted text also does not wrap to new lines when the browser window is reduced in size; a horizontal scrollbar appears (if necessary). Browsers also put a line of margin space above and below pre-formatted content and render pre-formatted text in a monospace font (so that all characters are equal width).

pre-formatted text is used in a variety of situations. Like when writing code examples that follow an indented format. Another example would be a poem where spacing is an important aspect of its message.

```
<pre>
Just a quick example...
...to show how pre-formatted text renders
notice I have multiple spaces        between words and lines
</pre>
```

Additional Reading

This area provides additional reading on the listed topics.

Heading 1-6 elements

Paragraph element

Address element

Pre element

Block Level List Elements

Unordered List

Unordered lists are used for a group of items that do not possess a natural sequencing. They are coded as . Each item in the list is surrounded by .
The elements have a line of margin space above and below them (on the top and bottom of the list); the actual list items are on separate lines but there is no extra vertical space between them.

Example of a simple unordered list

```
<ul>
   <li>list item 1</li>
   <li>list item 2</li>
   <li>list item 3</li>
   <li>list item 4</li>
</ul>
```

Nested Unordered List

You can nest lists within lists

Be careful when nesting lists that the nested is entirely within the holding it, as shown in the following example:

```
<ul>
  <li>list item 1</li>
  <li>list item 2</li>
  <li>list item 3
    <ul>
      <li>sublist item 1</li>
      <li>sublist item 2</li>
    </ul>
  </li>
  <li>list item 4</li>
</ul>
```

Ordered Lists

Ordered lists are used for a group of items that have a natural sequencing. Ordered lists are coded as . Each item in the list is surrounded by .
The elements have a line of margin space above and below them (on the top and bottom of the list); the actual list items are on separate lines but there is no extra vertical space between them.

Example of a simple ordered list

```
<ol>
  <li>list item 1</li>
  <li>list item 2</li>
  <li>list item 3</li>
  <li>list item 4</li>
</ol>
```

You can nest lists within lists

Be careful when nesting lists that the nested is entirely within the holding it, as shown in the following example:

```
<ol>
  <li>list item 1</li>
  <li>list item 2</li>
  <li>list item 3
    <ol>
      <li>sublist item 1</li>
      <li>sublist item 2</li>
    </ol>
  </li>
  <li>list item 4</li>
</ol>
```

Definition Lists

Definition lists are used for defining terms. As an example, these would be appropriate for glossaries. The outermost elements are <dl></dl>. Within the definition list are elements for the defined term (coded as <dt></dt>) and its description (coded as <dd></dd>). The <dl></dl> render with a line of margin space above and below. The <dt></dt> elements typically have their content left-aligned. These elements cannot contain block-level elements. The <dd></dd> elements typically have their content indented and, on the line, immediately below the defined term. These elements can contain block-level elements.

```
<h1>Glossary</h1>
<dl>
  <dt>XHTML</dt>
    <dd>Extensible Hypertext Markup Language, the structural markup of a web page</dd>
  <dt>CSS</dt>
    <dd>Cascading Style Sheets, which determines the presentation of a web page</dd>
</dl>
```

Additional Reading

This area provides additional reading on the listed topics.

Unordered List Element

Ordered List Element

Definition List Element

Linking Web Page Together

The Anchor Element

The anchor element, coded as <a>, serves as the foundation for the Web. It allows you to link pages together and create within page links.

Anchors can be used to:

- Link together pages
- Move to a different location within the same page
- Launch email programs
- Open new browser windows

Anchors are inline elements. They cannot be nested within each other.

Attribute	Usage and Effect	Values Accepted
Global Attributes	Attributes that can be used for all elements	N/A
href	By far the most common use of anchors is to link together documents via these href (hypertext reference) values; they are also used to launch email programs.	Document address (either absolute or relative) or mailto: and email address.
name	This attribute is used to place a within-page anchor.	Name of anchor

Absolute Links

Absolute links give the complete address to the document online. You must include the http:// for these to work properly. A sample absolute link is:

```
<p>Visit the <a href="http://www.google.com">Google search engine</a></p>.
```

This an example of a list of anchor elements used for navigation

```
<nav>
  <ul>
    <li><a href="http://www.sample.com/products.html">Products</a></li>
    <li><a href="http://www.sample.com/services.html">Services</a></li>
    <li><a href="http://www.sample.com/clients.html">Clients</a></li>
    <li><a href="http://www.sample.com/jobs.html">Jobs</a></li>
  </ul>
</nav>
```

The main drawbacks of absolute links are their length (the entire path needs to be specified) and their inflexibility. If you change your domain name all the paths need to be updated. In professional web design, relative links are mostly used. Absolute links are used on pages that link to another domain or for global navigation systems, where all the navigation is in one file.

Relative Links

Relative links use a partial address, based on the relationship between the current document and the document to load. Compared to absolute links, relative links are shorter, and your domain can be changed without breaking links. You can also easily move files from one server to another without breaking links if the directories maintain the same relative locations.

By default, all links are assumed to be relative unless http:// is specified. On Unix and Linux servers all directory and file names are case-sensitive so be careful to match capitalization properly in your code. You should avoid capitalization and use lowercase only. Also, be sure to never use spaces in file and directory names they are replaced with %20 on those servers.

If you just specify the file name, the web server looks in the current directory:

```
<nav>
  <ul>
    <li><a href="products.html">Products</a></li>
    <li><a href="services.html">Services</a></li>
    <li><a href="clients.html">Clients</a></li>
    <li><a href="jobs.html">Jobs</a></li>
  </ul>
</nav>
```

All the files referenced in the previous code would be in the same directory.

If you are moving up a directory level, specify ../ for every level you are moving up.

If you wanted to reach a file named example.html two directory levels up, the code would be:

```
<a href="../../example.html">View the example</a>
```

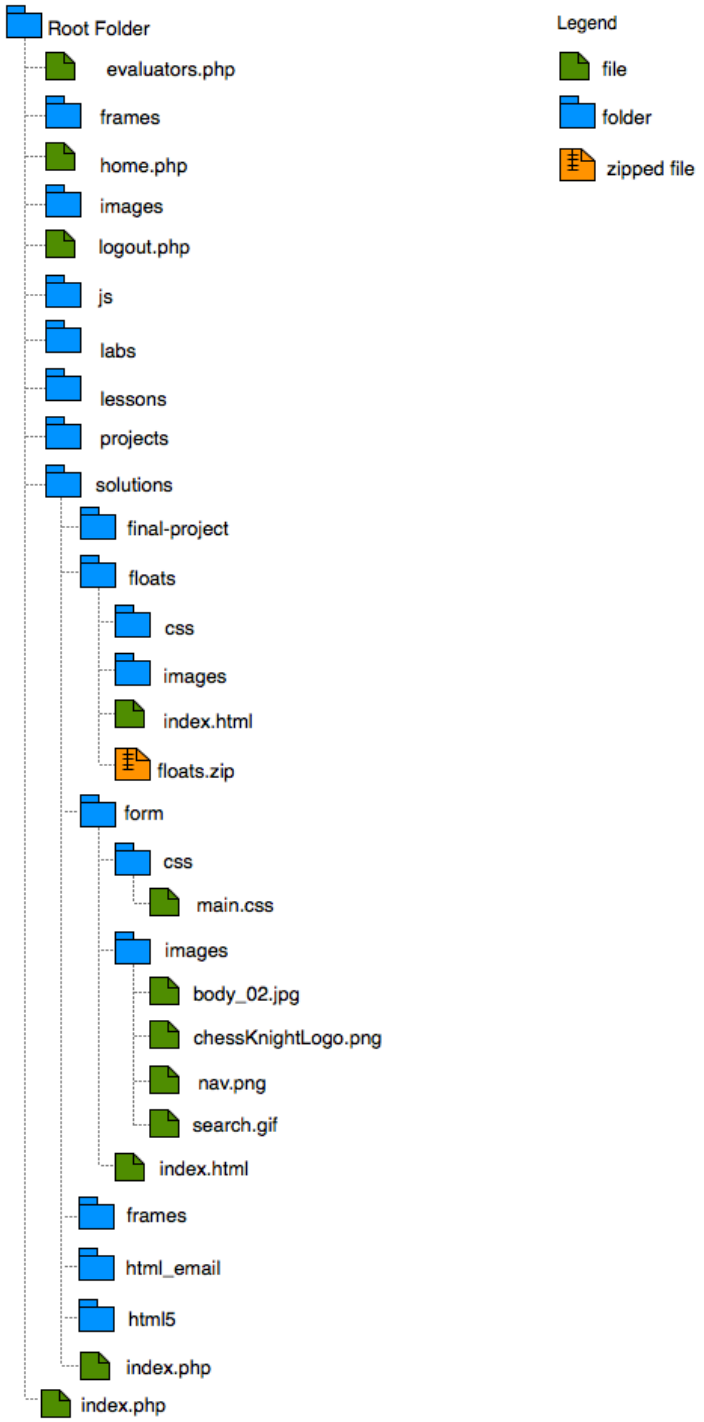
If you are moving down a directory level, you must specify the exact directory name (remember that capitalization matters on some servers).

For example, if you wanted to reach a file named example2.html that is in the 'coding' subdirectory (which is inside the 'examples' directory), the code would be:

```
<a href="examples/coding/example2.html">View the second example</a>
```

Traversing Directory Structures

To provide additional examples of relative linking, the following graphic shows a directory structure:



Using the image above write down the relative path for each link listed below.

1. Link from the evaluators.php page to the logout.php page.
2. Link from the logout.php page to the main.css page.
3. Link from the main.css page to the body_02.jpg image.
4. Link from the body_02.jpg image to the floats.zip file.
5. Link from the floats.zip file to the logout.php page.

Additional Reading

This area provides additional reading on the listed topics.

Anchor Element

Basic Table Elements

Introducing Tables

Tables are intended to be used for holding tabular information. Years ago, they were used for webpage layouts but that is no longer the case now that we have the box model. Tables are still used for HTML emails because of the multiple browser clients.

The table element

The first set of elements are <table></table>, which make up the entire table. All the rest of the table elements go within the table element

Attribute	Usage and Effect	Values Accepted	Default
Global attributes	Attributes that can be used for all elements	N/A	N/A
border	Borders surrounding the table on all outer sides. Sizing is in pixels. This attribute is obsolete and will generate an error in the validator, but it is useful to see our table cells. Until we learn about the CSS alternative, we will use it.	Number	0

The tr element

The tr element written as <tr></tr> creates what is called a table row. Table rows contain one or many th and td elements. Tables can have one or many table rows.

The td element

The td element written as <td></td> creates a single cell. For multi-cell tables, which most are, you need multiple td elements.

Below is a table of attributes for the td element.

Attribute	Usage and Effect	Values Accepted	Default
Global Attributes	Attributes that can be used for all elements	N/A	N/A
colspan	Specifies the number of columns a cell should span	number	1
headers	Specifies one or more header cells a cell is related to. Discussed in Web Development 2	id value	N/A
rowspan	Sets the number of rows a cell should span	number	1

The th element

The th element written as <th></th> specifies a table header cell. This is the most important element in creating an accessible table. It describes the context for the table cells in a column or row. Functionally, this is equivalent to the <td> element (it creates a cell and accepts the same attributes). Current browsers render <th> elements with their content rendered in boldface and centered horizontally and vertically, but there are no guarantees future browsers will do the same.

Below is a table showing the attributes for the th element.

Attribute	Usage and Effect	Values Accepted	Default
Global attributes	Attributes that can be used for all elements	N/A	N/A
abbr	Specifies an abbreviated version of the content in a header cell	number	1
colspan	Specifies the number of columns a cell should span	text	N/A
headers	Specifies one or more header cells a cell is related to. Discussed in Web Development 2	id value	N/A
scope	Specifies whether a header cell is a header for a column, row, or group of columns or rows. Discussed in Web Development 2	col, colgroup, row, rowgroup	N/A
sorted	Defines the sort direction of a column	reversed, number, reversed number, number reversed (HTML5 draft not supported at this time)	N/A
rowspan	Sets the number of rows a cell should span	number	1

The <table>, <tr>, <th>, and <td> cannot be not be inside inline elements, such as , , etc.). It is best to have them inside structure elements like div or main.

A simple table (one row, two cells) is coded as:

```
<table border="1">
  <tr>
    <td>Cell 1</td>
    <td>Cell 2</td>
  </tr>
</table>
```

Increasing the complexity, a bit (two rows, three cells):

```
<table border="1">
  <tr>
    <td>row 1 cell 1</td>
    <td>row 1 cell 2</td>
    <td>row 1 cell 3</td>
  </tr>
  <tr>
    <td>row 2 cell 1</td>
    <td>row 2 cell 2</td>
        <td>row 2 cell 3</td>
  </tr>
</table>
```

Note the symmetry of the table. Cells in the same column share the same width. Cells in the same row share the same height.

NOTE: The border="1" attribute is written so the table cell borders would show up if rendered in a browser. It is a deprecated attribute that is to be replaced with CSS, however, CSS is beyond the scope of this course.

Additional Reading

This area provides additional reading on the listed topics.

Table Element

Table row (tr) Element

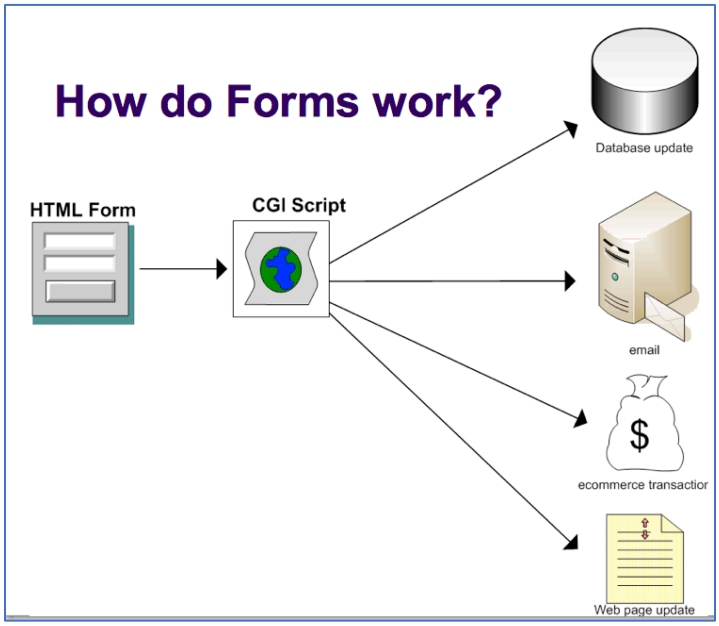
Table head (th) Element

Table data (td) Element

Chapter 4: Form Elements

What forms do

Introduction



Forms are used everywhere on the web. Anytime a website needs to get information from a user a form is used. You have filled out online forms to purchase items, register for classes, login to this website, etc. Below is a diagram of how a form works. It goes as follows from left to right:

- The user enters some information and submits the form.
- The form contents are sent to some back-end script and processed (in the picture it is the "CGI" file) for us it will be a PHP file.
- The processed information may go to one or many different areas
    - It may be stored in a Database
    - It may be sent as an email
    - It may do an e-commerce transaction
    - It may update a webpage

Above are just some of the areas where form data may go but not all.

The Form Element

Main Form Element

All of the form elements should be within the <form></form> element, which is block-level. There is one set of these elements per form. You can have more than one form per page, but forms cannot be nested.

```
<form method="post" action="somepage">
  <!--form elements go here-->
</form>
```

Attribute	Usage and Effect	Values Accepted	Include / Optional	Value to Use / Recommendations
action	Contains the path information to a server-side script that will be sent the form data when the form is submitted.	filename and path	Always include	"#"
method	Determines how form data is sent to the script. Using "get" results in the form data being sent via the URL, which has length limitations. Using "post" sends the data in the body of an HTTP POST request, which avoids the length limitations of "get" but cannot be bookmarked. When	get (the default) | post	Always include	"post"

Attribute	Usage and Effect	Values Accepted	Include / Optional	Value to Use / Recommendations
	creating forms, the post is the most commonly used.			

Additional Reading

This area provides additional reading on the listed topics.

Form Element

Form Label and Text Related Elements

Label Element

The label element, written as <label></label> is used for accessibility. It allows someone with a screen reader to associate the element title (label) with the element. It is written in one of two ways.

Either it surrounds the element

```
<label>Title <input type="text" size="5" name="somename"></label>
```

Or it is connected via the for and id attributes

```
<label for="someid">Title</label> <input type="text" size="5" name="somename" id="someid">
```

Notice how the for and id attributes have the same attribute value "someid". When using this method, you must have a for and id attribute and they must have the same value.

Text Input Fields

Text input fields allow users to enter one line of text only, they cannot do multiple lines of text.

Below are the attributes for a text input field.

Attribute	Usage and Effect	Values Accepted	Include / Optional	Value to Use / Recommendations
type	Specifies the form element to render.	text | password	Always include	"text" (even though this is the default you should still specify it) If you specify "password" t

Attribute	Usage and Effect	Values Accepted	Include / Optional	Value to Use / Recommendations
				hen the data in the field is obscured by dots/asterisks (note: the data is not encrypted so this only protects your data from someone looking over your shoulder)
name	This is the variable name assigned to the data; the script needs this information.	Variable name (no spaces)	Always include	Always assign a unique value for text inputs because you do not want to lose data.
size	Sets the width of the text input field in characters.	Number without a unit (e.g., 12)	Not required HTML 5, but can be used as a fallback if CSS is turned off	Try different values and keep in mind that font size and font family impacts this (and fonts fail to inherit into form elements, so style the elements directly).

Attribute	Usage and Effect	Values Accepted	Include / Optional	Value to Use / Recommendations
				The CSS width property can also be used to size the element. If you set a width property still set a reasonable *size* attribute value as a fallback in case CSS is turned off.
maxlength	Sets the maximum number of characters accepted by the text input field.	Number without a unit (e.g., 12)	Optional	Consider the data being entered, it's formatting, and how many characters that would potentially involve (especially if internationalization is a factor).
value	If you want some information to appear in the text box when	Character data	Optional	Only specify this if you want pre-set text to appear in the text input field when the page loads.

Attribute	Usage and Effect	Values Accepted	Include / Optional	Value to Use / Recommendations
	the page loads, specify that text as the value for this attribute; if you don't want anything in the box leave this attribute out entirely - whatever the user specifies becomes the value.			
placeholder	If you want some text to show up in the field but when the user starts typing in the field the text disappears	Character data	Optional	Use this to indicate to the user what to enter in the box
disabled	Prevents the form element from being used. It cannot	disabled	Optional	Browsers differ in their default styling,

Attribute	Usage and Effect	Values Accepted	Include / Optional	Value to Use / Recommendations
	receive focus, be tabbed to, or have accepted entries by the user. Form values are not sent to the script when the form is submitted. JavaScript can still modify the form element value.			so test cross-browser and style via CSS accordingly to avoid confusing users (field should be grayed out, for example).
readonly	Prevents the form element from having data entered or modified. The element can receive focus, be tabbed to, and its content can be sent when the form	readonly	Optional	Browsers differ in their default styling, so test cross-browser and style via CSS accordingly to avoid confusing users (field should be grayed out, for example).

Attribute	Usage and Effect	Values Accepted	Include / Optional	Value to Use / Recommendations
	is submitted. Typically changes to *readonly* fields are done via JavaScript.			

A sample text input element (with a field name).

```
<label>First Name: <input type="text" name="first_name" size="15"></label>
```

A sample password input element (with a field name).

```
<label>First Name: <input type="password" name="first_name" size="15"></label>
```

Setting the CSS width property is more reliable and consistent than the size attribute. This is especially true when getting consistency cross-browser.

Textarea Boxes

Textarea boxes allow users to enter multiple lines of text. They should be used in situations where a user would enter multiple lines of text like a comment area.

Below is a table of attributes for the text area box.

Attribute	Usage and Effect	Values Accepted	Include / Optional	Value to Use / Recommendations
name	This is the variable name assigned to the data; the script needs this information.	Variable name (no spaces)	Always include	Always assign a unique value for textarea boxes because you do not want to lose data.
rows	Sets the height of the box in rows. In most browsers, the box will never occupy more than this amount of vertical space in the browser window (Safari and Chrome allow resizing).	Number without a unit (e.g., 12)	Not required HTML 5, but can be used as a fallback if CSS is turned off	At your discretion; try different values and keep in mind that font size and font family impacts this (and fonts fail to inherit into form elements, so style the elements directly) The CSS height property can be used instead, but you should still specify the *rows* attribute as a fallback in case CSS is disabled. The CSS property

Attribute	Usage and Effect	Values Accepted	Include / Optional	Value to Use / Recommendations
				will override the HTML5. attribute.
cols	Sets the width of the box in characters. In most browsers, the box will never occupy more than this amount of horizontal space in the browser window (Safari allows resizing).	Number without a unit (e.g., 12)	Not required HTML 5, but can be used as a fallback if CSS is turned off	At your discretion; try different values and keep in mind that font size and font family impacts this (and fonts fail to inherit into form elements, so style the elements directly) The CSS width property can be used instead, but you should still specify the *cols* attribute as a fallback in case CSS is disabled. The CSS property will override the HTML5. attribute.
placeholder	If you want some text to show up in the field but when the user starts typing in the	Character data	Optional	Use this to indicate to the user what to enter in the box

Attribute	Usage and Effect	Values Accepted	Include / Optional	Value to Use / Recommendations
	field the text disappears			
disabled	Prevents the form element from being used. It cannot receive focus, be tabbed to, or have data entered by the user. data is not sent to the script when the form is submitted. JavaScript can still modify the form element value.	disabled	Optional	Browsers differ in their default styling, so test cross-browser and style via CSS accordingly to avoid confusing users (field should be grayed out, for example).
readonly	Prevents the form element from having data entered or modified. The	readonly	Optional	Browsers differ in their default styling, so test cross-browser and style

Attribute	Usage and Effect	Values Accepted	Include / Optional	Value to Use / Recommendations
	element can receive focus, be tabbed to, and its data is sent when the form is submitted. Typically data changes to *readonly* fields are via JavaScript.			via CSS accordingly to avoid confusing users (field should be grayed out, for example).

A sample textarea element (with a field name). Copy and paste the code into the code renderer to see the result

```
<label for="address" >Address:</label>
<textarea id="address" name="address" cols="30" rows="4"></textarea>
```

To have default text appear within the textbox, when the page loads, put that text between the <textarea></textarea> elements. Modifying the vertical-align CSS property for 'textarea' will fix the alignment of the field name's text (which goes by default to the bottom) if they are inside the same container element. If the user enters enough data to fill the available rows a vertical scrollbar will appear and the user can keep typing. As with text input boxes expect some cross-browser variation in sizing and be sure to test for fonts being applied properly.

Additional Reading

This area provides additional reading on the listed topics.

Label Element

Input (type="text") Element

Textarea Element

Form Radio and Checkbox Elements

Radio Buttons

Radio buttons let a user select one and only one selection for each radio button group. The radio button groups are designated by having the same "name" attribute value.

Below is a table of attributes for radio and checkboxes.

Attribute	Usage and Effect	Values Accepted	Include / Optional	Value to Use / Recommendations
type	Specifies the form element to render.	radio	Always include	
name	This is the variable name assigned to the data; the script needs this information.	Variable name (no spaces)	Always include	Sets of radio buttons must share the same name to function properly.
value	This information is associated with the variable from the *name* attribute if that radio button is selected.	Character data	Always include	At your discretion
checked	Selects the indicated radio button as soon as the page loads. Only specify this for one radio button in each grouping.	checked	Optional	

Attribute	Usage and Effect	Values Accepted	Include / Optional	Value to Use / Recommendations
disabled	Prevents the form element from being used. It cannot receive focus, be tabbed to, or have data entered by the user. Data is not sent to the script when the form is submitted. JavaScript can still "check" this radio button.	disabled	Optional	Browsers differ in their default styling, so test cross-browser and style via CSS accordingly to avoid confusing users (field should be grayed out, for example).

Sample radio button elements (with field names). You can select one color and one size only. Copy and past the code into the code renderer application to see how it renders.

```
<p>Color:</p>
<label><input type="radio" name="color" value="blue" > Blue</label>
<label><input type="radio" name="color" value="red" checked > Red</label>
<label><input type="radio" name="color" value="yellow" > Yellow</label>

<p>Size:</p>
<label><input type="radio" name="size" value="small" > Small</label>
<label><input type="radio" name="size" value="medium" checked > Medium</label>
<label><input type="radio" name="size" value="large" > Large</label>
```

Checkboxes

Checkboxes allow the user to check (select) one or more options

Attribute	Usage and Effect	Values Accepted	Include / Optional	Value to Use / Recommendations
type	Specifies the form element to render.	checkbox	Always include	
name	This is the variable name assigned to the data; the script needs this information.	Variable name (no spaces)	Always include	At your discretion; groups of checkboxes can share the same name (their *value* data is joined together so no data is lost) or have different names.
value	This information is associated with the variable from the name attribute if that checkbox is selected.	Character data	Always include	At your discretion
checked	Selects the indicated checkbox(es) when the page loads. Specify for as many checkboxes as desired.	checked	Optional	

Attribute	Usage and Effect	Values Accepted	Include / Optional	Value to Use / Recommendations
disabled	Prevents the form element from being used. It cannot receive focus, be tabbed to, or have data entered by the user. Data is not sent to the script when the form is submitted. JavaScript can still "check" this checkbox.	disabled	Optional	Browsers differ in their default styling, so test cross-browser and style via CSS accordingly to avoid confusing users (field should be grayed out, for example).

Sample checkbox elements (with field names). The user can check one or more of each checkbox. Copy and paste the code into the code renderer application to see how it renders.

```
<p>Sizes:</p>
<label><input type="checkbox" name="size" value="small" > Small</label>
<label><input type="checkbox" name="size" value="medium" checked > Medium</label>
<label><input type="checkbox" name="size" value="large" checked > Large</label>
```

Additional Reading

This area provides additional reading on the listed topics.

Input (type="radio") Element

Input (type="checkbox") Element

Other Form Elements

Drop-Down Menus and Multi-select boxes

Drop-down menus and multi-select boxes allow the user to select one or more selections based upon a list of items to select.

Below is a table of attributes for the select and option elements

Element	Attribute	Usage and Effect	Values Accepted	Include / Optional	Value to Use / Recommendations
<select> </select>	name	This is the variable name assigned to the data; the script needs this information so it can associate the data from the drop-down menu with a variable.	Variable name (no spaces)	Always include	At your discretion
<select> </select>	size	The value assigned to this attribute is the height of the menu in rows.	Number without a unit (e.g., 12)	Optional	At your discretion; if you want a regular drop-down menu leave this attribute out entirely. Values greater than 1 will create a multi-select box.

Element	Attribute	Usage and Effect	Values Accepted	Include / Optional	Value to Use / Recommendations
<select> </select>	multiple	Allows multiple selections in the menu.	multiple	Optional	Only include if the size is more than 1, otherwise, it will have no effect.
<option> </option>	value	This information is associated with the variable from the name attribute if that option is selected.	Character data	Always include	At your discretion; if the item has no meaning, such as 'Please Choose...' then set value="".
<option> </option>	selected	Selects the indicated option(s) when the page loads. Only one option can be selected if the size of the menu is set to 1 (or if the size is omitted because the default is 1).	selected	Optional	

Element	Attribute	Usage and Effect	Values Accepted	Include / Optional	Value to Use / Recommendations
<select> </select> or <option> </option>	disabled	Prevents the form element from being used. It cannot receive focus, be tabbed to, or have data entered by the user. Data is not sent to the script when the form is submitted. JavaScript can still change the selected option.	disabled	Optional	Browsers differ in their default styling, so test cross-browser and style via CSS accordingly to avoid confusing users (field should be grayed out, for example).

Code for a drop-down menu (with a field name) is. NOTE: The options names do not have to be the same as the value attribute value. Copy and paste the code into the code renderer to see how it looks/works.

```
<label for="color">Color:</label><br />
<select id="color" name="color" size="3" multiple>
  <option value="blue">Blue</option>
  <option value="red" selected >Red</option>
  <option value="yellow">Yellow</option>
  <option value="purple">Purple</option>
  <option value="green">Green</option>
  <option value="pink">Pink</option>
</select>
```

Changing the code to remove the size and multiple.

```
<label for="color">Color:</label><br />
<select id="color" name="color">
  <option value="blue">Blue</option>
  <option value="red" selected="selected">Red</option>
  <option value="yellow">Yellow</option>
  <option value="purple">Purple</option>
  <option value="green">Green</option>
  <option value="pink">Pink</option>
</select>
```

Uploading a File

Below is a table of attributes for the file element.

Attribute	Usage and Effect	Values Accepted	Include / Optional	Value(s) to Use / Recommendations
type	Specifies the form element to render.	file	Always include	"file"
name	This is the variable name assigned to the data (file name and path).	Variable name (no spaces)	Always include	At your discretion; always assign a unique value to avoid data loss.
accept	Specifies the MIME (Multipurpose Internet Mail Extensions) types of the files that will be uploaded.	MIME values are separated by commas.	Optional	The choice is based on the expected file types to be uploaded. NOTE: This is not supported by all browsers and it is not secure. Files must be checked on the server

Attribute	Usage and Effect	Values Accepted	Include / Optional	Value(s) to Use / Recommendations
size	Sets the width of the field in characters. The directory path and file name are shown in the field.	Number without a unit (e.g., 40)	Optional	At your discretion; try different values and keep in mind that font size and font family impacts this (and fonts fail to inherit into form elements, so style the elements directly) The CSS width property can also be used but specify a size as a fallback in case CSS is turned off.
disabled	Prevents the form element from being used. It cannot receive focus, be tabbed to, or have data entered by the user. Data is not sent to the script when the form is submitted. JavaScript is never	disabled	Optional	Browsers differ in their default styling, so test cross-browser and style via CSS accordingly to avoid confusing users (field should be grayed out, for example).

Attribute	Usage and Effect	Values Accepted	Include / Optional	Value(s) to Use / Recommendations
	permitted to write to a file upload field, disabled or not.			

In most browsers specifying type="file" creates a text box as well as a 'Browse...' button; Opera labels this button as 'Choose...' Safari and Chrome display 'no file selected' rather than the text box and label the button as 'Choose File'. You do not need a label element for file uploads

```
<input type="file" name="somefile" size="30" >
```

Be careful with file uploads, as files placed on the server could contain viruses or other malware. Users may also attempt to upload extremely large files to break your scripting and expose server vulnerabilities.

Hidden Fields

Below is a table of attributes for the hidden field element.

Attribute	Usage and Effect	Values Accepted	Include / Optional	Value to Use / Recommendations
type	Specifies the form element to render.	hidden	Always include	"hidden"
name	This is the variable name assigned to the data.	Variable name (no spaces)	Always include	At your discretion; always assign a unique

Attribute	Usage and Effect	Values Accepted	Include / Optional	Value to Use / Recommendations
				value to avoid data loss.
value	This value is associated with the *name* when the form is submitted.	Character data	Always include	At your discretion

Hidden fields are used to pass data to a script without the user seeing that data rendered in the browser. For example, a hidden field could pass an email address that the form data will be sent to, or if the form spans multiple pages the data from previous pages can be automatically saved in hidden fields on subsequent pages (the script would dynamically write these hidden fields as it generated the subsequent form pages).

Below is the code. You cannot see it rendered as it is hidden

```
<input type="hidden" name="destination" value="youremail@domain.com" >
```

Note: These fields are not truly hidden; viewing the source code reveals them.

Additional Reading

This area provides additional reading on the listed topics.

Select Element

Option Element

Input (type=file) Element

Input (type=hidden) Element

Form Buttons

Buttons

Buttons allow the user to submit a form, reset a form, or do some other task by clicking a button

Below is a table of elements and attributes for the form button.

Element	Attribute	Usage and Effect	Values Accepted	Include / Optional	Value to Use / Recommendations
<input >	type	Specifies the form element to render.	submit | image | button | reset	Always include	"submit" for a text-based submit button "image" for a graphical submit button (include *src* and *alt* attributes too) "button" for a text-based button that does not submit the form when clicked "reset" to create a reset button

Element	Attribute	Usage and Effect	Values Accepted	Include / Optional	Value to Use / Recommendations
<input >	name	If you have multiple submit buttons this can be used by the script receiving the data to determine what action to take, since any form element with a name has its *value* pass ed to the script when the form is submitted.	Variable name (no spaces)	Optional	At your discretion; always assign a unique value.
<input >	value	This value is shown as the text in the button.	Character data	Always include	At your discretion; default values tend to be ambiguous.
<input >	disabled	Prevents the form element	disabled	Optional	Browsers differ in their default styling,

Element	Attribute	Usage and Effect	Values Accepted	Include / Optional	Value to Use / Recommendations
		from being used. It cannot receive focus, be tabbed to, or have data entered by the user. Data is not sent to the script when the form is submitted. JavaScript can still submit and reset the form and can still modify the value of the button.			so test cross-browser and style via CSS accordingly to avoid confusing users (field should be grayed out, for example).

Standard Submit Button

If you want to use a standard HTML5. button to submit the form, the type attribute is assigned a value of "submit". The value attribute determines the label of the button. If this is not specified, the button will read 'Submit Query' in most browsers; otherwise, the button will be labeled whatever is specified for the value attribute.

Code for this standard (X)HTML submit button is:

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

Buttons can also be written as:

```
<button>Place Order</button>
```

The button element submits a form by default without writing type="submit".

The difference between button and input is that you can put content in between the button open and closing tags. You will see an example of this in the next section.

Graphical Submit Button

You can use an image for the submit button by specifying type="image", such as:

This code displays the search.gif graphic. When that graphic is clicked, the form is processed.

```
<input type="image" src="images/submit.gif" alt="submit" >
```

Using the button element, you can write:

```
<button>
  <img src="images/submit.gif" alt="submit" width="89" height="24">
</button>
```

Reset Button

It is also possible to create a reset button by specifying type="reset"; clicking this button clears all the fields. The danger with using a reset button is that the user can accidentally click the button and lose the data already entered.

```
<input type="reset" value="Reset Form" >
```

Buttons that Do Not Submit the Form

By specifying type="button" you create a button that, when clicked, does not submit the form. It is just a dummy button that needs some JavaScript to be useful.

```
<button type="button">Click Me</button>
```

Or

```
<input type="button" value="Click Me" >
```

Additional Reading

This area provides additional reading on the listed topics.

Button Element

Input (type="button") Element

Chapter 5: PHP Basics

PHP Basics

Introduction

PHP (recursive acronym for PHP: Hypertext Preprocessor, prior it was called Personal Home Page) is a widely used open-source general-purpose scripting language that is especially suited for web development and can be embedded into HTML. What distinguishes PHP from something like client-side JavaScript is that the code is executed on the server, generates HTML, which is then sent to the client. The client would receive the results of running that script but would not know what the underlying code was.

One advantage and draw to PHP is that it is extremely simple for a newcomer but offers many advanced features for a professional programmer.

PHP online manual is very well done with many examples. I recommend you visit it and get used to navigating around it.

There is an online PHP emulator that is useful for evaluating small PHP scripts.

You can also run any PHP script from the server, via the command line, by entering "php scriptname.php"

php.ini file

The php.ini file is the default configuration file for running applications that require PHP. It is used to control variables such as upload sizes, file timeouts, and resource limits. On DigitalOcean the actual file you can edit is found at /etc/php/7.0/apache2/php.ini. (the PHP version number "7.0" may be different on your system.

To view the file in the browser, you have to call the method phpinfo(). To do this you need to create a PHP file In your webroot directory (for DigitalOcean it is /var/www/html). Create a file named info.php and add the following code.

```
<?php phpinfo(); ?>
```

Then in your web browser enter youripaddress/info.php and you will view a webpage showing your PHP settings.

Errors, Warnings, and Notices

PHP has three levels of error notices:

- **Fatal error** - It will terminate the program and print the error in the log. It will also print the error on the page if "show errors" is enabled in the "php.ini" file.
- **Warning** - It will print the warning on the page (if the PHP configurations are set up to do that) and to the error log file but allow the program to continue the best it can.
- **Notice** - It will print the notice on the page (if the PHP configurations are set up to do that) but allow the program to run.

Turning on/off Errors, Warnings, and Notices

If you have an error in your PHP script you may see a blank page when you view it in the browser.

Using DigtialOcean you should be able to turn on errors by going to the php.ini file on the server, it is found at /etc/php/7.0/apache2/ (your version number 7.0 may be different). The following are the steps to follow to turn errors on.

1. Use vim to open the file using sudo vim php.ini.
2. Scroll down until you find display_errors = Off and change it to display_errors = On.
3. Once done restart apache service apache2 restart.

Check your info.php file in your browser and you should see display errors set to on.

To turn on errors or a page when it is turned off normally. You could add the following to the top of the page (wrapped in a PHP block). You will need to have errors either turned on or use this block of code otherwise you will not see any error messages.

```
ini_set('display_errors', 1);
ini_set('display_startup_errors', 1);
error_reporting(E_ALL);
```

This is not as good of a method as turning errors on in your php.ini file

PHP block

For PHP code to be parsed on the server, it must be in PHP code blocks <?php ?>. Below is an example of injecting PHP into HTML.

The echo is how you output (display) something in PHP.

```
<p>This is HTML and <?php echo "this is php";?></p>
```

If you are going to write just PHP and not have any HTML you should write all your PHP code in an open and closing the PHP block as shown.

```
<?php
echo "this is a php block without the ending.";
?>
```

echo and print

There are two ways to output text in PHP, you can use either "echo" or "print". They both do the same thing except print returns a value of 1. Also, print is a function whereas echo is a **language construct** (A language construct is a syntactically allowable part of a program that may be formed from one or more lexical tokens following the rules of a programming language. The term Language Constructs is often used as a synonym for control structure and should not be confused with a function). Echo is more popular and faster; we will be using echo in this course.

Ending Statements with a semi-colon

In PHP you must end every statement (unless there is only one) with a semi-colon. PHP will generate a "fatal error" if a semi-colon is missing.

Using Variables in PHP

Variables are a fundamental part of any programming language. A variable is simply a container that holds a certain value. Variables get their name because that certain value can change throughout the execution of the script. It's this ability to contain changing values that make variables so useful.

For example, consider the following simple PHP script:

```
echo 2 + 2;
```

As you might imagine, this code outputs the number 4 when it's run. This is all well and good; however, if you wanted to print the value of, say, 5 + 6 instead, you would have to write another PHP script, as follows:

```
echo 5 + 6;
```

This is where variables come into play. By using variables instead of numbers in your script, you make the script much more useful and flexible:

```
echo $x + $y;
```

You now have a general-purpose script. You can set the variables $x and $y to any two values you want, either at some other place in your code or as a result of input from the user. Then, when you run the preceding line of code, the script outputs the sum of those two values. Re-run the script with different values for $x and $y, and you get a different result.

Naming Variables

A variable consists of two parts: the variable’s name and the variable’s value. Because you’ll be using variables in your code frequently, it’s best to give your variables names you can understand and remember. Like other programming languages, PHP has certain rules you must follow when naming your variables:

- Variable names begin with a dollar sign ($)
- The first character after the dollar sign must be a letter or an underscore The remaining characters in the name may be letters, numbers, or underscores without a fixed limit
- Variable names are case sensitive ($Variable and $variable are two different variables), so it’s worth sticking to one variable naming method — for example, always using lowercase — to avoid mistakes.

Here are some examples of PHP variable names:

```
$my_first_variable
$anotherVariable
$x
$_123
```

Creating Variables

Creating a variable in PHP is known as declaring it. Declaring a variable is as simple as using its name in your script:

```
$my_first_variable;
```

When PHP first sees a variable's name in a script, it automatically creates the variable at that point.

Many programming languages prevent you from using a variable without first explicitly declaring (creating) it. But PHP lets you use variables at any point just by naming them. This is not always the blessing you might think; if you happen to use a nonexistent variable name by mistake, no error message is generated, and you may end up with a hard-to-find bug. In most cases, though, it works just fine and is a helpful feature.

When you declare a variable in PHP, it’s a good practice to assign a value to it at the same time. This is known as initializing a variable. By doing this, anyone reading your code knows exactly what value the variable holds at the time it's created. (If you don’t initialize a variable in PHP, it’s given the default value of null.)

Here’s an example of declaring and initializing a variable:

```
$my_first_variable = 3;
```

This creates the variable called $my_first_variable and uses the = operator to assign it a value of 3. (You look at = and other operators later in this chapter.) Looking back at the addition example earlier, the following script creates two variables, initializes them with the values 5 and 6, and then outputs their sum (11):

```
$x = 5;
$y = 6;
```

```
echo $x + $y;
```

Understanding Data Types

All data stored in PHP variables fall into one of eight basic categories, known as data types. A variable's data type determines what operations can be carried out on the variable's data, as well as the amount of memory needed to hold the data.

PHP supports four scalar data types. Scalar data means data that contains only a single value. Here's a list of them, including examples:

Scalar Data	Type Description	Example
Integer	A whole number	15
Float	A floating-point number	8.23
String	A series of characters	"hello, world"
Boolean	Represents either true or false	true

As well as the four scalar types, PHP supports two compound types. Compound data is data that can contain more than one value. The following table describes PHP's compound types:

Compound Data Type	Description
Array An ordered map	An ordered map (contains names or numbers mapped to values)

Compound Data Type	Description
Object	A type that may contain properties and methods

Finally, PHP supports two special data types, that don't contain scalar or compound data as such, but have a specific meaning:

Special Data Type	Description
Resource	Contains a reference to an external resource, such as a file or database
Null	May only contain null as a value, meaning the variable explicitly does not contain any value.

About Loose Typing

PHP is known as a loosely - typed language. This means that it's not particularly fussy about the type of data stored in a variable. It converts a variable's data type automatically, depending on the context in which the variable is used. For example, you can initialize a variable with an integer value; add a float value to it, thereby turning it into a float; then join it onto a string value to produce a longer string. In contrast, many other languages, such as Java, are strongly-typed; once you set the type of a variable in Java, it must always contain data of that type.

PHP's loose typing is both good and bad. On the plus side, it makes variables very flexible; the same variable can easily be used in different situations. It also means that you don't need to worry about specifying the type of a variable when you declare it.

However, PHP won't tell you if you accidentally pass around data of the wrong type. For example, PHP will happily let you pass a floating-point value to a piece of code that expects to be working on an integer value. You probably won't see an error message, but you may discover that the output of your script isn't quite what you expected! These types of errors can be hard to track down. (Fortunately, there is a way to test the type of a variable, as you see in a moment.)

Checking Data Types

The following functions allow you to check data types

Function	Description
is_int (value)	Returns true if the value is an integer
is_float (value)	Returns true if the value is a float or double
is_string (value)	Returns true if the value is a string
is_bool (value)	Returns true if the value is a boolean
is_array (value)	Returns true if the value is an array
is_object (value)	Returns true if the value is an object
is_resource (value)	Returns true if the value is a resource
is_null (value)	Returns true if the value is null

You can also check a variable type using the gettype(variable) method.

NOTE: The "\n" forces a new line in the php evaluator.

```
$a="green";
$b=45;
$c=45.23456;
$d=true;
$e=array();
$f=null;
echo gettype($a)."\n"; //OUTPUTS STRING
echo gettype($b)."\n"; //OUTPUTS INTEGER
echo gettype($c)."\n"; //OUTPUTS DOUBLE
echo gettype($d)."\n"; //OUTPUTS BOOLEAN
echo gettype($e)."\n"; //OUTPUTS ARRAY
echo gettype($f)."\n"; //OUTPUTS NULL
```

Comments

Comments are short sentences in your code that are not interpreted. They are useful for providing information about what your code is doing or to comment out sections of your code for debugging.

```
// - comments out a line
/*.....*/ - comments out a block of text
```

Casting

Strings to integers and integers to strings. You can permanently change a data type by just rewriting the value. However, there may be situations where you just want to temporarily change the datatype. You can do that by casting. In most situations, you cast from an integer to a string and visa-versa as shown below.

NOTE: To cast we put the casting type in parenthesis ((int),(string),(float))

```
$string = "51";
$double = 51.50456;
$integer = 51;
echo gettype($string)."\n"; //OUTPUTS STRING
echo gettype((int)$string)."\n"; //OUTPUTS INTEGER
echo gettype($integer)."\n"; //OUTPUTS INTEGER
echo gettype((string)$integer)."\n"; //OUTPUTS STRING
echo gettype($double)."\n"; //OUTPUTS DOUBLE
```

```
echo gettype((float)$string)."\n"; //OUTPUTS DOUBLE
echo gettype((int)$double)."\n"; //OUTPUTS INTEGER
echo gettype((string)$double)."\n"; //OUTPUTS STRING


/*IF YOU ATTEMPT TO CAST A STRING TO AN INT THAT CANNOT BE CAST IT
WILL RETURN A "0." IF YOU ATTEMPT TO ADD TO STRINGS YOU WILL GET A
"0."*/


$str1 = "String1";
$str2 = "String2";
echo $str1 + $str2; //RETURNS 0

echo (int)$str1; //RETURNS 0
echo "45" + "45"; //RETURNS 90 AND CHANGES THE RESULT TO AN INTEGER
```

Arithmetic Operators

Syntax	Description
x + y	Adds
x - y	Subtracts
x * y	Multiplies
x / y	Division always returns a float
x % y	Modulus Remainder
-x	Negates
x.y	Concatenates

Assignment Operators

Assignment	Same as...	Description
x = y	x = y	The left operand is set to the value of the expression on the right
x += y	x = x + y	Addition
x -= y	x = x - y	Subtraction
x *= y	x = x * y	Multiplication
x /= y	x = x / y	Division
x %= y	x = x % y	Modulus
a .= b	a = a . b	Concatenate two strings

Incrementing/Decrementing Operators

Operator	Name	Description
++ x	Pre-increment	Increments x by one then returns x
x ++	Post-increment	Returns x, then increments x by one
-- x	Pre-decrement	Decrements x by one then returns x

Operator	Name	Description
x --	Post-decrement	Returns x, then decrements x by one

Comparison

Syntax	Name	Result
x == y	Equal	TRUE if x is equal to y after type juggling("1"==1 php changes the type thus it is true).
x === y	Identical	TRUE if x is equal to y, and they are of the same type. ("1"===1 this is false. 1===1 this is true)
x != y	Not equal	TRUE if x is not equal to y after type juggling("1"!=1 php changes the type thus it is false)
x !== y	Not identical	TRUE if x is not equal to y, or they are not of the same type ("1" !== 1 is true).
x < y	Less than	TRUE if x is strictly less than y.
x > y	Greater than	TRUE if x is strictly greater than y.
x <= y	Less than or equal to	TRUE if x is less than or equal to y.

Syntax	Name	Result
x >= y	Greater than or equal to	TRUE if x is greater than or equal to y.

Logical Operators

Operator	Name	Description	Example
x and y	And	TRUE if both x and y are true	x=6 y=3 (x < 10 and y > 1) returns true
x or y	Or	TRUE if either or both x and y are true	x=6 y=3 (x==6 or y==5) returns true
x xor y	Xor	TRUE if either x or y is true, but not both	x=6 y=3 (x==6 xor y==3) returns false
x && y	And	TRUE if both x and y are true	x=6 y=3 (x < 10 && y > 1) returns true

Operator	Name	Description	Example
x || y	Or	TRUE if either or both x and y are true	x=6 y=3 (x==5 || y==5) returns false
! x	Not	TRUE if x is not true	x=6 y=3 !(x==y) returns true

PHP Strings

Introduction

There are a lot of string functions in PHP. In this section, we will look at the most common and basic ones but you and find them on the PHP manual website.

Strings

Strings in PHP are a set of characters put in single or double-quotes.

```
$string = 'This is a string';
$string = "This is a string";
```

Strings Constructions

Each character in a string has a position, which is numbered starting with zero.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T	h	i	s		i	s		a		s	t	r	i	n	g

Let's get the following characters for the indicated position

```
$string = "This is a string";
echo $string[2]; //OUTPUTS I
echo $string[6]; //OUTPUTS S
echo $string[12]; //OUTPUTS R
```

PHP strings can be changed dynamically

```
$string = "This is a string";
echo $string;
$string[2] = "a";
echo $string; //OUTPUTS THAS IS A STRING
```

String Length

You can get the length of a string using the strlen() method. Notice it returns 16, length starts the count at one. The length is always one more than the last position.

```
$string = "This is a string";
echo strlen($string); //OUTPUTS 16
```

Multi-line Strings

You can create multi-line strings by enclosing the whole string in single or double-quotes. You can also use something called heredoc, which is useful if your stings contain quotes as well or if you want to add variable data to a string.

```
$string = "this is a
multi-line string";
echo $string;

$string = 'this is a multi-line string with "quotes" the double quotes render because the rest of the string is wrapped in single quotes';
echo $string;

$string = "this is a multi-line string with \"quotes\" the double quotes render because they are escaped with the backslash character.";
echo $string;

$string = <<<STR
This is a heredoc multi-line string with "quotes", the double quotes render because we are using the heredoc, where both 'single' and "double quotes" render;
STR;

echo $string;
```

NOTE When using a heredoc, after this operator <<<, an identifier is provided, then a newline. The string itself follows, and then the same identifier again to close the quotation.

The closing identifier must begin in the first column of the line. Also, the identifier must follow the same naming rules as any other label in PHP: it must contain only alphanumeric characters and underscores and must start with a non-digit character or underscore. It is a good naming convention to make heredoc identifiers in all caps.

DO NOT have any spaces or characters after the closing identifier, start a new line, and then write your statement.

It is very important to note that the line with the closing identifier must contain no other characters, except possibly a semicolon (;). That means especially that the identifier not be indented, and there not be any spaces or tabs before or after the semicolon. It's also important to realize that the first character before the closing identifier must be a newline as defined by the local operating system. This is \n on UNIX systems, including Mac OS X. The closing delimiter (possibly followed by a semicolon) must also be followed by a newline.

If this rule is broken and the closing identifier is not "clean", it will not be considered a closing identifier, and PHP will continue looking for one. If a proper closing identifier is not found before the end of the current file, a parse error will result at the last line.

Inserting Variables into a String

Though you can insert a variable's value in a double-quoted string simply by including the variable's name (preceded by a $ symbol), at times things get a bit more complicated. Consider the following situation:

```
$favoriteAnimal = "cat";
echo "My favorite animals are $favoriteAnimals";
```

This code is ambiguous; should PHP insert the value of the $favoriteAnimal variable followed by an "s" character? Or should it insert the value of the (non-existent) $favoriteAnimals variable? PHP attempts to do the latter, resulting in:

```
 My favorite animals are
```

Fortunately, you can get around this problem using curly brackets, as follows:

```
$favoriteAnimal = "cat";
echo "My favorite animals are {$favoriteAnimal}s";
```

This produces the expected result:

```
My favorite animals are cats
```

You can also place the opening curly bracket after the $ symbol, which has the same effect:

```
echo "My favorite animals are ${favoriteAnimal}s";
```

The important thing is that you can use the curly brackets to distinguish the variable name from the rest of the string. You can use this curly bracket syntax to insert more complex variable values, such as array element values and object properties. (You explore arrays and objects in the next few chapters.) Just make sure the whole expression is surrounded by curly brackets, and you're good to go:

```
$myArray["age"] = 34;
echo "My age is {$myArray["age"]}"; // DISPLAYS "MY AGE IS 34"
```

Below is an example of inserting a string into a double-quoted or heredoc string.

```
$name = "Scott";
$string = "This is a double-quoted string with a variable.  Double quoted
strings the variable value is displayed but not with single-quoted strings.
My name is {$name}.";
echo $string;

$name = "Scott";
$string = <<<STR
This is a heredoc mulit-line string with a variable. The variable value is
displayed. My name is {$name}.
STR;
echo $string;
```

String Concatenation

Strings can be concatenated with the ".". Strings can only concatenate with strings, they cannot concatenate with other data types ("5".1) will generate an error.

```
$string = "This is a string."." This is a concatenated string";
echo $string;
$string = "You can also ";
$string .= "add to strings using the ";
$string .= "dot-equals (.=) operator";
echo $string;
```

```
$string = "You can string numbers like (1 . 1), which will generate 11 as shown below:";
echo $string;
$string = 1 . 1;
echo $string;

$string = "But if you do (1.1) you will get";
echo $string;//1.1

$string = "You can also ";
$string2 = "concatenate variables with string datatypes";


//EITHER IN DOUBLE QUOTES
echo "$string,$string2";


//SINGLE QUOTES
echo $string.' , '.$string2;


//OR WITH NO QUOTES, SEPARATED BY A COMMA
echo $string,$string2;

$string = "You can also";
echo "{$string} add string variables to quotes using curly braces{}, //IF THE WHOLE STRING
IS IN DOUBLE QUOTES";
```

Substrings

Substrings in PHP returns a sub-section of the string but leaves the original string intact. In these examples' substring was put into another variable, but it can also be echoed out directly.

```
$string = "0123456789";
$substring = substr($string,3);//STARTS FROM POSITION 3 ON "3456789"
echo $substring;

$substring = substr($string,0,5); //STARTS FROM POSITION 0 AND DISPLAYS THE NEXT
FIVE CHARACTERS "01234"
echo $substring;
```

```
$substring = substr($string,-4); //STARTS FROM THE END AND GOES TO THE LEFT 4
CHARACTERS "6789"
echo $substring;

$substring = substr($string,3,-3); //STARTS AT POSITION 3 AND COUNTS BACK 3 FROM
END "3456"
echo $substring;

$substring = substr($string,-5,-3); //STARTS FROM THE END AND GOES TO THE LEFT 5
CHARACTERS (INCLUDING THE FIFTH CHARACTER), THEN STARTS FROM THE END
AND GOES TO THE LEFT 3 CHARACTERS "56"
echo $substring;

//using a different string
$string = "456789";

$SUBSTRING = SUBSTR($STRING,3); //STARTS FROM POSITION 3 ON "789"

ECHO $SUBSTRING;

$substring = substr($string,0,5); //STARTS FROM POSITION 0 AND DISPLAYS
THE NEXT FIVE CHARACTERS "45678"
echo $substring;
```

Finding the Number of Occurrences

Occasionally you might need to know how many times some text occurs within a string. For example, if you were writing a simple search engine, you could search a string of text for a keyword to see how relevant the text is for that keyword; the more occurrences of the keyword, the greater the chance that the text is relevant.

You could find the number of occurrences easily enough using strpos() and a loop, but PHP, as in most other things, gives you a function to do the job for you: substr_count(). To use it, simply pass the string to search and the text to search for, and the function returns the number of times the text was found in the string. For example:

```
$myString = "I say, nay, nay, and thrice nay!";
echo substr_count( $myString, "nay" ); // DISPLAYS '3'
```

You can also pass an optional third argument to specify the index position to start searching, and an optional fourth argument to indicate how many characters the function should search before giving up. Here are some examples that use these third and fourth arguments:

```
$myString = "I say, nay, nay, and thrice nay!";
echo substr_count( $myString, "nay", 9 ); // DISPLAYS '2'
echo substr_count( $myString, "nay", 9, 6 ); // DISPLAYS '1'
```

String Replace

str_replace() lets you replace all occurrences of a specified string with a new string. It's the PHP equivalent of using the Replace All option in a word processor.

The function takes three arguments: the search string, the replacement string, and the string to search through. It returns a copy of the original string with all instances of the search string swapped with the replacement string. Here's an example:

```
$myString = "It was the best of times, it was the worst of times,";
echo str_replace( "times", "bananas", $myString );

// DISPLAYS "IT WAS THE BEST OF BANANAS, IT WAS THE WORST OF BANANAS,"
echo str_replace( "times", "bananas", $myString );
```

If you want to know how many times the search string was replaced, pass in a variable as an optional fourth argument. After the function runs, this variable holds the number of replacements:

```
$myString = "It was the best of times, it was the worst of times,";

// DISPLAYS "IT WAS THE BEST OF BANANAS, IT WAS THE WORST OF BANANAS,"
echo str_replace( "times", "bananas", $myString, $num );

// DISPLAYS "THE TEXT WAS REPLACED 2 TIMES."
echo "The text was replaced {$num} times";
```

You can pass arrays of strings for the first and second arguments to search for and replace multiple strings at once. You can also pass an array of strings as the third argument, in which case str_ replace() replaces the text in all the strings in the array and returns an array of altered strings. This is a very powerful way to do a global search and replace.

```
// PROVIDES: YOU SHOULD EAT PIZZA, SODA, AND ICE CREAM EVERY DAY
$phrase  = "You should eat fruits, vegetables, and fiber every day.";
$healthy = array("fruits", "vegetables", "fiber");
$yummy   = array("pizza", "soda", "ice cream");

$newphrase = str_replace($healthy, $yummy, $phrase);

echo $newphrase;
```

Substring Replace

Whereas str_replace() searches for a particular string of text to replace, substr_replace() replaces a specific portion of the target string. To use it, pass three arguments: the string to work on, the replacement text, and the index within the string at which to start the replacement. substr_replace() replaces all the characters from the start point to the end of the string with the replacement text, returning the modified string as a copy (the original string remains untouched). This example shows how substr_replace() works:

```
$myString = "It was the best of times, it was the worst of times,";

// DISPLAYS "IT WAS THE BANANAS"
echo substr_replace( $myString, "bananas", 11 );
```

You can see that the preceding code has replaced all of the original text from the character at index 11 onwards with the replacement text ("bananas").

If you don't want to replace all the text from the start point to the end of the string, you can specify an optional fourth argument containing the number of characters to replace:

```
$myString = "It was the best of times, it was the worst of times,";
```

```
// DISPLAYS "IT WAS THE BEST OF BANANAS, IT WAS THE WORST OF TIMES,"
echo substr_replace( $myString, "bananas", 19, 5 );
```

Pass a negative fourth argument to replace up to that many characters from the end of the string:

```
$myString = "It was the best of times, it was the worst of times,";

// DISPLAYS "IT WAS THE BEST OF BANANAS THE WORST OF TIMES,"
echo substr_replace( $myString, "bananas", 19, -20 );
```

You can also pass a zero value to insert the replacement text into the string rather than replacing characters:

```
$myString = "It was the best of times, it was the worst of times,";

// DISPLAYS "IT REALLY WAS THE BEST OF TIMES, IT WAS THE WORST OF
TIMES,"
echo substr_replace( $myString, "really ", 3, 0 );
```

String Conversions

There are numerous methods for converting strings.

```
$string = "this is a string";
echo strtoupper($string);
echo strtolower($string);
echo ucwords($string);
```

String Comparisons

Returns an integer < 0 if str1 is less than str2; an integer > 0 if str1 is greater than str2, and 0 if they are equal. Strings compare based on their ASCII numbers so capital letters come before lowercase. The integers are the subtraction of the str1 ASCII number from the str2 ASCII number.

ASCII stands for American Standard Code for Information Interchange. It's a 7-bit character code where every single bit represents a unique character. It is best to lowercase both stings before comparing them.

```
echo strcmp("A","Z");
echo strcmp("Z","A");
echo strcmp("z","A");
echo strcmp("a","Z");
echo strcmp("A","A");
```

PHP Logic

Logic Statements

Logic statements (otherwise known as conditionals) allow the program to make decisions based upon the true/false evaluation of expressions. If you imagine the PHP interpreter following a path through your code the conditional statements are places where the code branches into two or more paths and the interpreter must choose which path to follow.

If Statement

An if statement only evaluates if something is true.

```
$string="scott";
if ($string==="scott"){
        echo "The condition is true";
}
```

You can also do the reverse using the !==. It returns true if the conditions are not equal.

```
$string="scott";
if ($string!=="sco"){
        echo "The condition is true";
}
```

If Else Statements

An if-else statement evaluates if something is true or false and does something for either.

PHP evaluates the number zero, an empty string, and of course the boolean false as "false." Everything else is "true."

```
$value = 0;//will output false
if ($value){
  echo "true<br />";}
else{
  echo "false<br />";}

$value = "scott";//will output true
if ($value){
  echo "true<br />";}
else{
  echo "false<br />";}

$value = 0;//will output false
if ($value){
  echo "true<br />";}
else{
  echo "false<br />";}

$value = 1;//will output true
if ($value){
  echo "true<br />";
}
else {
  echo "false<br />";
}
```

If, Else If, Else

An if, else if, else statement evaluates multiple statements until one is true. If all statements are false, then else is the default.

```
$string ="scott";
$output = "";
if ($string ==="john"){
        $output = "john";
}
else if ($string ==="steve"){
  $output = "steve";
}

else if ($string ==="scott"){
        $output = "scott";
}

else{
        $output = "name not found";
}

echo $output; //OUTPUTS SCOTT
```

Nested If, Else If, Else

For more complex decision making you can nest your statements.

```
$fname ="john";
$lname = "smith";
$output = "";
if ($fname ==="john"){
        if ($lname ==="smith"){
                $output = "john smith";
        }
   else{
        $output = "first name found but not last name";
   }
}

else if ($fname ==="scott"){
        if ($lname === "jones"){
      $output = "scott jones";
   }
   else if ($lname === "shaper"){
        $output = "scott shaper";
   }
   else{
      $output = "first name found but not last name";
   }
}

else{
        $output = "first name not found";
}

echo $output;
```

Switch Statement

The switch statement is a shortcut way of doing multiple if, else if, else statements. The break statement stops the iteration if the expression returns true.

NOTE: Switch statements can also call functions if a case evaluates to true.

```
$name = "scott";
$output = "";
switch($name)
```

```
{
   case "john" : $output = "name found"; break;
   case "steve" : $output = "name found"; break;
   case "scott" : $output = "name found"; break;
   default : $output = "name not found"; break;
}
echo $output;
```

Ternary Operator

The ternary operator is another shortcut that replaces a one time if else expression.

```
$name = "scott";
$output = ($name==="scott") ? "the expression is true" : "the expression is false";
echo $output;
```

Boolean Operators

Boolean operators use && for "and" and || for "or." You can also just use the words "and" and "or" as well.

```
$username = "sshaper";
$password = "password";
if ($username === "sshaper" && $password === "password"){
          echo "username and password are correct";
}
else{
           echo "username and password are not correct";
}
if ($username==="sshaper" or $password==="password"){
    echo "username and/or password are correct";
 }
else{
    echo "username and/or password are not correct";
}
```

PHP Loops

Introduction

The basic idea of a loop is to run the same block of code again and again until a certain condition is met. As with decisions, that condition must take the form of an expression. If the expression evaluates to true, the loop continues running. If the expression evaluates to false, the loop exits, and execution continues the first line following the loop's code block.

For Loop

For loops iterate over an arithmetic process of numbers by adding or subtracting on each iteration.

```
for ($i=1; $i<=10; $i++){
  echo "<p>I have counted to $i</p>";
}
```

While Loop

While loops continue until it hits a trigger (when it returns false)

```
$i = 0
while ($i<=10){
          echo "<p>$i</p>";
    $i++;
}
```

Do While Loop

The same as a while loop but will always do the first iteration no matter what the result (true or false). In the example below, it executes and prints $i even though $i is greater than 10.

```
$i=11;
do{
  echo $i;
  $i++;
}
while ($i < 10);
```

Continue and Break Statements

Continue statements stop the current iteration and continue onto the next iteration. If you run the script it will output all the odd numbers. If you comment the continue all the numbers will be listed.

```
for ($i = 0; $i < 10; $i++) {
    if ($i % 2 == 0){
        continue;
    }
    echo "$i\n";
}
```

The break statement stops the looping cycle completely. Breaks are handy when searching for one item and when found stop searching.

```
for ($i = 0; $i < 10; $i++) {
    if ($i % 2 == 0){
        break;
    }
    echo "$i\n";
}
```

Nested Loops

You can nest loops as well.

```
for ($i = 1; $i < 5; $i++){
            echo $i . "\n";
            for ($j = 1; $j < 4; $j++){
                        echo "\t".$j."\n";
            }
}
```

PHP Function

Functions

Functions are snippets of code that are assigned a name. If you want to use a function just call the function name. Functions are extremely powerful as they allow you to have reusable code. Write a function to do a certain task and any time you need that task done call the function.

Function Declarations

Function declarations start with the keyword function, name of the function, followed with parenthesis function functionName(). The code used by the function is in curly braces { }. The naming convention is like variables. Start lower case and camel case after that. Functions must start with a letter and cannot have spaces or special characters.

```
function myFunction(){
   $string = "this is my function";
}
```

Functions may or may not return a value it depends on the circumstance. To return a value use the return keyword followed by the value to return.

```
function myFunction(){
   $string = "This is my function";
   return $string;
}
```

NOTE: Once you use the return keyword no other code is written after the return will run. In other words, the return terminates the function execution.

To call a function just enter the function name.

```
echo myFunction();
```

Function Arguments

Arguments are values that you can pass to a function. Arguments allow the function to be more versatile.

NOTE: Notice the order of operations here. It is not (5 + 7) * 3 but 5 + (7 * 3)

NOTE: Notice the function names will be changing in the examples. If writing multiple functions on the same file at the same time, I cannot redeclare the same function I have to create a new one.

```
function myFunction1($arg0,$arg1,$arg2){
  return $arg0 + $arg1 * $arg2;
}
echo myFunction1(5,7,3); //RETURNS 26

echo myFunction1(5,7); //RETURNS FATAL ERROR
```

PHP 7.0 and above no longer allows you to omit an argument you have to include a parameter for all arguments you have listed in your function.

PHP allows for arguments to be optional. The code to do so is $arg="option". You can include default values to make an argument optional, but they must be in order and they must be the last argument(s). You cannot have an optional argument in the middle of required arguments or PHP will generate a warning and send a null variable.

In the example below "myFunction2" is written correctly and works fine with one to three arguments because of the optional arguments I have included. But "myFunction3" does not because the optional argument is written in the middle.

```
function myFunction2($arg0, $arg1=1, $arg2=2){
        return $arg0 + $arg1 * $arg2;
}

echo myFunction2(5); //OUTPUTS 7
echo myFunction2(5,7); //OUTPUTS 19
echo myFunction2(5,7,3); //OUTPUTS 26
```

```
function myFunction3($arg0, $arg1=1, $arg2){
        return $arg0 + $arg1 * $arg2;
}
echo myFunction3(5,7);
```

Scope

PHP has a lexical scope, which means access is maintained based upon where the variable was created.

In the example below, we have declared a variable outside of the function as a string and set a value to it $string="scott". Yet within the function we also use the variable named "$string." The variable within the function is local only to the function. It is not the same variable name outside of the function.

```
$string = "scott";
function myFunction4(){
        $string = "john";
        return string;
}

echo $string; //OUTPUTS SCOTT BECAUSE THE FUNCTION IS NOT CALLED
myFunction4();

echo $string; //OUTPUTS SCOTT BECAUSE THE VARIABLE WITHIN THE FUNCTION IS
LOCAL ONLY TO THE FUNCTION

echo myFunction4(); //OUTPUTS JOHN BECAUSE THE VARIABLE WAS RETURNED FROM
THE FUNCTION AND THE FUNCTION WAS CALLED WITHIN THE ECHO STATEMENT.
```

global keyword

To make the variable declared outside the function be accessed within the function you use the global keyword.

```
$string = "scott";
function myFunction5(){
	global $string;
	$string = "john";
}

myFunction5();
echo $string; //OUTPUTS JOHN BECAUSE THE VARIABLE WAS GLOBAL SCOPE AND
CHANGED IN THE FUNCTION
```

To make multiple variables global just separate them with a comma.

```
$string1 = "scott";
$string2 = "sam";
function myFunction6(){
	global $string1, $string2;
	$string1 = "john";
	$string2 = "smith";
}

echo $string1." ".$string2; //OUTPUTS SCOTT SAM

myFunction6();

echo $string1." ".$string2; //OUTPUTS JOHN SMITH
```

NOTE: Variables marked as global do not have to be declared outside the function they can be declared within the function as well. For example, if you want a variable created within a function to be used outside of that function use the global keyword.

```
function myFunction7(){
	global $string7;
	$string7 = "Scott";
}
```

```
myFunction7();
echo $string7; //OUTPUTS SCOTT
```

Working with References

You've already learned that you can pass information to a function in the form of arguments, as well as return information from a function to its calling code using the return statement. When you do either of these things, PHP passes copies of the information to and from the function; this is known as passing and returning by value.

Most of the time this isn't a problem, but sometimes you want your function to work on the original information, rather than on a copy. Consider the following example:

```
function resetCounter( $c ) {
        $c = 0;
}
$counter = 0;
$counter++;
$counter++;
$counter++;
echo"$counter\n"; // DISPLAYS"3"
resetCounter( $counter );
echo"$counter\n"; // DISPLAYS"3"
```

This code defines a function, resetCounter() , that resets its argument to zero. A $counter variable is then initialized to zero and incremented three times. As you'd expect, the value of $counter at this point is 3 resetCounter() is then called, passing in $counter, the variable to reset. However, as the second echo statement shows, $counter has not been reset by the function. This is because the parameter $c inside resetCounter() merely holds a copy of the information stored in $counter. So, when the function sets $c to zero, it doesn't affect the value of $counter at all.

Fortunately, PHP provides a mechanism known as references that you can use to work around such issues. A reference is a bit like a shortcut or alias to a file on your hard drive. When you create a reference to a PHP variable, you now have two ways to read or change the variable's contents — you can use the variable name, or you can use the reference. Here's a simple example that creates a reference to a variable:

```
$myVar = 123;
$myRef = &$myVar;
$myRef++;
echo $myRef ."\n"; // DISPLAYS"124"
echo $myVar ."\n"; // DISPLAYS"124"
```

First, a new variable, $myVar, is initialized with the value 123. Next, a reference to $myVar is created, and the reference is stored in the variable $myRef. Note the ampersand (&) symbol after the equals sign; using this symbol creates the reference.

The next line of code adds one to the value of $myRef. Because $myRef points to the same data as $myVar, both $myRef and $myVar now contain the value 124, as shown by the two echo statements. Now that you know what references are, and how to create them, it’s time to look at how you can pass references into and out of your functions.

Recursive Function

A recursive function is a function and calls itself. It is similar to doing a loop.

```
function recursive($i){
  echo "The value of variable i is $i\n";
  $i += 1;
  if ($i <= 10){
    recursive($i);
  }
}
$i = 1;
recursive($i);
```

PHP Arrays

What is an array

Arrays hold multiple data types as compared to a variable which only holds one. Arrays are great for organizing and manipulating a lot of data. PHP arrays can store any of the PHP data types in any order. PHP has many functions for working with arrays. We will cover the more common ones here but you can find others in the PHP manual.

Indexed Array

An indexed array assigns an index (integer) to an element starting with 0. Each additional element gets the next integer in the sequence. To create an indexed array, create a variable, and assign it to an array.

NOTE: Indexed arrays start at 0 as the first index, not 1.

```
$variableName = array();
```

As of PHP5.4, you can also write it in a literal syntax like so:

```
$variableName = [];
```

An array can be thought of as a numbered(indexed) list of data starting with 0. Additionally, arrays can hold multiple data types. In the example below, the first number is the "index" number the second is the element for that index.

NOTE: This is just an example of an array structure your arrays may have different data types.

Index	Value
0	"string"
1	43

Index	Value
2	boolean
3	object
4	array

The above array can be written as follows:

NOTE: The boolean "true" will render as "1." If the boolean was "false" it would render as a space.

```
$arr = array("string",43,true,$object,array(1,2,3));
```

To see the data structure of an array, use the print_r command wrapped in pre tags.

NOTE: The print_r() function is used to print human-readable information about a variable. Print_r() is useful in seeing array and object structures. The pre element is used to make it look prettier. Do not use echo to print arrays as it will not print the array structure will generate an "array to string conversion" error.

```
echo "<pre>";
print_r($arr);
echo "</pre>";
```

Changing Data in Arrays

Arrays are mutable meaning they can be changed. Below is an example of how to change an array element.

```
$arr = array(1,2,3,4);
$arr[2] = "scott";
echo "<pre>";
print_r($arr);
echo "</pre>";
```

Array length

The array length is the count of how many elements the array has.

NOTE: Count starts at 1 not 0

```
$arr = array(1,2,3,4);
$count = count($arr);
echo $count;//will output 4
```

Looping through an array

Using array length, you can loop through an array with a for loop.

```
$arr = array(1,2,3,4,5,6,7,8,9);
$len = count($arr);
$output = "";
for ($i=0;$i<$len;$i++){
    $output .= $arr[$i];
    echo $output."<br />";
}
```

You could also use a foreach loop, which is designed to work with arrays.

```
$arr = array(1,2,3,4,5,6,7,8,9);
$output = "";
foreach($arr as $i){
        $output .= $i;
        echo $output."<br />";
}
```

Adding/Removing elements to/from an Array

For adding elements you can use array methods like push() and unshift().

array_push - adds multiple elements to the end of an array.

array_unshift - add multiple elements to the beginning of an array.

```
$arr = array();
echo "<pre>";
print_r($arr); //OUTPUTS EMPTY ARRAY
array_push($arr,1,2,3,4,5); //PUSH ADDS ELEMENTS TO THE END OF AN ARRAY
```

```
print_r($arr); //OUTPUTS [1, 2, 3, 4, 5, 6]
array_unshift($arr,-2,-1,0); //UNSHIFT ADDS ELEMENTS TO THE BEGINNING OF AN
ARRAY
print_r($arr); //OUTPUTS  [-2, -1, 0, 1, 2, 3, 4, 5, 6]
echo "</pre>";
```

You can remove elements using array methods like pop and shift. However, they only remove one array element.

array_pop - removes and returns the last element of the array.

array_shift - removes and returns the first element of the array

```
$arr = array(-2, -1, 0, 1, 2, 3, 4, 5, 6);
echo "<pre>";
echo array_pop($arr)."<br />"; //OUTPUTS 6
print_r($arr); //OUTPUTS [-2, -1, 0, 1, 2, 3, 4, 5]
echo array_shift($arr)."<br />"; //OUTPUTS -2
print_r($arr); //OUTPUTS  [-1, 0, 1, 2, 3, 4, 5]
echo "</pre>";
```

Slice

Slice copies a part of an array without changing the original array. The start parameter is the array index in which the slice is to begin (remember it counts from zero).

```
echo "<pre>";
$arr = array(1,2,3,4,5,6,7,8,9);
print_r($arr); //SHOW ORIGINAL ARRAY
$newArr = array_slice($arr,4); //COPIES FROM INDEX 4 ON
print_r($newArr);
$newArr = array_slice($arr,4,3); //COPIES FROM INDEX 4 AND GOES FORWARD THREE
INDEXES
print_r($newArr);

$newArr = array_slice($arr,4,3,true); //COPIES FROM INDEX 4 AND GOES FORWARD
THREE INDEXES PRESERVING THE INDEX KEYS
print_r($newArr);
echo "</pre>";
```

You can also use negative integers same as the string.slice() method.

```
echo "<pre>";
$arr = array(1,2,3,4,5,6,7,8,9);
print_r($arr); //SHOW ORIGINAL ARRAY
$newArr = array_slice($arr,-4); //COPIES FROM END BACK 4 INDEXES
print_r($newArr);
$newArr = array_slice($arr,-4,-2); //COPIES FROM END BACK 4 INDEXES AND THEN
STARTS FROM BACK AND GOES TWO INDEXES
print_r($newArr);
$newArr = array_slice($arr,-4,-2,true); //COPIES FROM END BACK 4 INDEXES AND THEN
STARTS FROM BACK AND GOES TWO INDEXES PRESERVING THE INDEX KEYS
print_r($newArr);
echo "</pre>";
```

Splice

Splice deletes a specified number of elements and optionally replaces them.

NOTE: IMPORTANT! splice changes the original array.

```
echo "<pre>";
$arr = array(1,2,3,4,5,6,7,8,9);
print_r($arr); //SHOW ORIGINAL ARRAY
array_splice($arr,4); //GOES FROM FIRST INDEX AND COUNTS 4 IN
print_r($arr);
$arr = array(1,2,3,4,5,6,7,8,9); //BECAUSE IT CHANGES THE ORIGINAL ARRAY I HAVE
TO RESET $ARR
array_splice($arr,4,3); //GOES FROM FIRST INDEX AND COUNTS 4 IN THEN REMOVES
THE NEXT 3 INDEXES
print_r($arr);
$arr = array(1,2,3,4,5,6,7,8,9); //BECAUSE IT CHANGES THE ORIGINAL ARRAY I HAVE
TO RESET $ARR
array_splice($arr,4,3,array(10,11,12)); //GOES FROM FIRST INDEX AND COUNTS 4 IN
THEN REMOVES THE NEXT 3 INDEXES AND REPLACES THEM WITH THE ARRAY, OR A
SINGLE VALUE IF AN ARRAY IS NOT USED.
print_r($arr);
echo "</pre>";
```

You can use negative integers as well.

```
echo "<pre class='no-background'>";
```

```
$arr = array(1,2,3,4,5,6,7,8,9);
print_r($arr); //SHOW ORIGINAL ARRAY

array_splice($arr,-6); //GOES FROM THE END AND REMOVES 6 INDEXES
print_r($arr);
$arr = array(1,2,3,4,5,6,7,8,9); //BECAUSE IT CHANGES THE ORGINAL ARRAY I HAVE TO
RESET $ARR
array_splice($arr,-6,-3); //GOES FROM THE END IN 6 INDEXES, THEN GOES FROM THE
END IN 3 INDEXES REMOVES THE INDEXES IN BETWEEN
print_r($arr);
$arr = array(1,2,3,4,5,6,7,8,9); //BECAUSE IT CHANGES THE ORGINAL ARRAY I HAVE TO
RESET $ARR
array_splice($arr,-6,-3,array(10,11,12)); //GOES FROM THE END IN 6 INDEXES, THEN
GOES FROM THE END IN 3 INDEXES REMOVES THE INDEXES IN BETWEEN AND
REPLACES THEM WITH THE ARRAY.
print_r($arr);
echo "</pre>";
```

Associative Arrays

Array indexes can be names as well as numbers. This type of array is called an "associative array." You can get the value by calling the index name instead of number.

```
$arr = array("fname"=>"scott","lname"=>"shaper");
print_r($arr);
echo $arr['fname']; //OUTPUTS SCOTT
```

Converting Arrays

You can convert a string into an array by using the explode method.

```
$str = "Programming in PHP is cool";
$arr = explode(" ",$str);
echo "<pre>";
print_r($arr);
echo "</pre>";
```

Convert an array to a string we use the implode method.

```
$arr = array("Programming","in","PHP","is","cool");
$str = implode($arr); //BREAK ON EVERY COMMA AND CREATES ONE LONG STRING
echo "$str<br />";
```

```
$str = implode(" ",$arr); //BREAK ON EVERY COMMA AND INSERTS A SPACE
echo "$str<br />";
$str = implode("--",$arr); //BREAK ON EVERY COMMA AND INSERTS A "--"
echo "$str<br />";
```

Sorting Arrays

One powerful feature of arrays in most languages is that you can sort the elements in any order you like. For example, if you've just read 100 book titles from a text file into an array, you can sort the titles alphabetically before you display them. Or you might create a multidimensional array containing customer information, then sort the array by the number of purchases to see who your most loyal customers are. When it comes to sorting arrays, PHP provides a lot of different functions. We will look at the more common ones here.

Sorting Indexed Arrays with sort() and rsort()

The simplest of the array sorting functions are sort() and rsort(). sort() sorts the values of the array in ascending order (alphabetically for letters, numerically for numbers, letters before numbers), and rsort() sorts the values in descending order. To use either function, simply pass it the array to be sorted. The function then sorts the array. As with all the sorting functions covered in this chapter, the function returns true if it managed to sort the array or false if there was a problem. Here's an example that sorts a list of authors alphabetically in ascending order, and then in descending order:

```
$authors = array("Steinbeck", "Kafka", "Tolkien", "Dickens" );

// DISPLAYS "ARRAY ([0] => DICKENS [1] => KAFKA [2] => STEINBECK [3]
=> TOLKIEN)"
sort($authors);
print_r($authors);

// DISPLAYS "ARRAY ([0] => TOLKIEN [1] => STEINBECK [2] => KAFKA [3]
=> DICKENS)"
rsort( $authors);
print_r($authors);
```

Sorting Associative Arrays with asort() and arsort()

Take another look at the previous sort() and rsort() code examples. Notice how the values in the sorted arrays have different keys from the values in the original array. For example, " Steinbeck " has an index of 0 in the original array, 2 in the second array, and 1 in the third array. The sort() and rsort() functions are said to have reindexed the original array.

For indexed arrays, this is usually what you want to happen: you need the elements to appear in the correct order, and at the same time you expect the indices in an indexed array to start at zero. However, for associative arrays, this can cause a problem. Consider the following scenario:

```
$myBook = array( "title" => "Bleak House","author" => "Dickens","year" => 1853 );
sort( $myBook );

// DISPLAYS "ARRAY ( [0] => BLEAK HOUSE [1] => DICKENS [2] => 1853 )"

print_r( $myBook );
```

Notice how sort() has re-indexed the associative array, replacing the original string keys with numeric keys and effectively turning the array into an indexed array. This renders the sorted array practically useless because there’s no longer an easy way to find out which element contains, say, the book title.

This is where asort() and arsort() come in. They work just like sort() and rsort() , but they preserve the association between each element's key and its value:

```
$myBook = array( "title" => "Bleak House","author" => "Dickens","year" => 1853 );

// DISPLAYS "ARRAY ( [TITLE] => BLEAK HOUSE [AUTHOR] => DICKENS [YEAR]
=>1853 )"
asort( $myBook );
print_r( $myBook );

// DISPLAYS "ARRAY ( [YEAR] => 1853 [AUTHOR] => DICKENS [TITLE] =>
BLEAKHOUSE )"


arsort( $myBook );
print_r( $myBook );
```

Note that although you can use asort() and arsort() on indexed arrays, they're commonly used with associative arrays.

Chapter 6: Object-Oriented PHP

PHP Object Oriented Principles

What Is Object-Oriented Programming?

So far, you've written code that passes chunks of data from one function to the next — a technique known as procedural programming. Object-oriented programming takes a different approach. Objects model the real-world things, processes, and ideas that your application is designed to handle. An object-oriented application is a set of collaborating objects that independently handle certain activities.

The concepts of classes and objects, and how you can use them, are the fundamental ideas behind object-oriented programming. As you'll see, an object-oriented approach gives you some big benefits over procedural programming.

Advantages of OOP

Let's look at some of the advantages of an OOP approach to software development.

Because your application is based on the idea of real-world objects, you can often create a direct mapping of people, things, and concepts to classes. These classes have the same properties and behaviors as the real-world concepts they represent, which helps you to quickly identify what code needs to be written and how different parts of the application need to interact.

The second benefit of OOP is code reuse. You frequently need the same types of data in different places in the same application. For example, an application that manages hospital patient records might contain a class called Person. Several different people are involved in patient care — the patient, the doctors, the nurses, hospital administrators, and so on. By defining a class called Person that encompasses the properties and methods common to all these people, you can reuse an enormous amount of code in a way that isn't always possible in a procedural programming approach.

What about other applications? How many applications can you think of that handle information about individuals? Probably quite a few. A well - written Person class can easily be copied from one project to another with little or no change, instantly giving you all the rich functionality for dealing with information about people that you developed previously. This is one of the biggest benefits of an object-oriented approach — the opportunities for code reuse within a given application as well as across different projects.

Another OOP advantage comes from the encapsulation of classes. If you discover a bug in your Person class, or you want to add new features to the class or change the way it functions, you have only one place to go. All the functionality of that class is contained in a single PHP file. Any parts of the application that rely on the Person class are immediately affected by changes to it. This can vastly simplify the search for bugs and makes the addition of features a relatively painless task. Encapsulation is particularly important when working on large, complex applications.

Applications written using OOP are usually relatively easy to understand. Because an object-oriented approach forces you to think about how the code is organized, it’s a lot easier to discover the structure of an existing application when you are new to the development team. What’s more, the object-oriented design of the application gives you a ready-made framework within which you can develop new functionality.

On larger projects, there are often many programmers with varying skill levels. Here, too, an object-oriented approach has significant benefits over procedural code. Objects hide the details of their implementation from the users of those objects. Instead of needing to understand complex data structures and all the quirks of the business logic, junior members of the team can, with just a little documentation, begin using objects created by senior members of the team. The objects themselves are responsible for triggering changes to data or the state of the system. Now you have an idea of the advantages of object-oriented applications. You’re now ready to learn the nitty-gritty of classes and objects, which you do in the next few sections. By the end of this chapter, you’ll probably come to see the benefits of the OOP approach for yourself.

Understanding Basic OOP Concepts

Before you start creating objects in PHP, it helps to understand some basic concepts of object-oriented programming. In the following sections, you explore classes, objects, properties, and methods. These are the basic building blocks that you can use to create object-oriented applications in PHP.

Classes

In the real world, objects have characteristics and behaviors. A car has a color, a weight, a manufacturer, and a gas tank of a certain volume. Those are its characteristics. A car can accelerate, stop, signal for a turn, and sound the horn. Those are its behaviors. Those characteristics and behaviors are common to all cars. Although different cars may have different colors, all cars have a color.

With OOP, you can model the general idea of a car — that is, something with all of those qualities — by using a class. A class is a unit of code that describes the characteristics and behaviors of something, or a group of things. A class called Car, for example, would describe the characteristics and behaviors common to all cars.

Objects

An object is a specific instance of a class. For example, if you create a Car class, you might then go on to create an object called myCar that belongs to the Car class. You could then create a second object, yourCar, also based on the Car class.

Think of a class as a blueprint, or factory, for constructing an object. A class specifies the characteristics that an object will have, but not necessarily the specific values of those characteristics. Meanwhile, an object is constructed using the blueprint provided by a class, and its characteristics have specific values.

For example, the Car class might indicate merely that cars should have a color, whereas a specific myCar object might be colored red.

The distinction between classes and objects is often confusing to those new to OOP. It helps to think of classes as something you create as you design your application, whereas objects are created and used when the application is run.

Properties

In OOP terminology, the characteristics of a class or object are known as its properties. Properties are much like regular variables, in that they have a name and a value (which can be of any type). Some properties allow their value to be changed and others do not. For example, the Car class might have properties such as color and weight. Although the color of the car can be changed by giving it a new paint job, the weight of the car (without cargo or passengers) is a fixed value.

Methods

The behaviors of a class — that is, the actions associated with the class — are known as its methods. Methods are very similar to functions; in fact, you define methods in PHP using the function statement. Like functions, some methods act on external data passed to them as arguments, but an object's method can also access the properties of the object. For example, an accelerate method of the Car class might check the fuel property to make sure it has enough fuel to move the car. The method might then update the object's velocity property to reflect the fact that the car has accelerated. The methods of a class, along with their properties, are collectively known as members of the class.

PHP Classes

Creating Classes

Although the theory behind classes and objects can get quite involved, classes and objects are easy to create in PHP. As you'd imagine, you need to create a class before you create an object belonging to that class. To create a class, you use PHP's class keyword. Here's a simple class:

```
class Car {

    //METHODS AND PROPERTIES GO HERE
}
```

This code simply defines a class called Car that does nothing whatsoever — it merely includes a comment. (You add some functionality to the class shortly.) Notice that a class definition consists of the class keyword, followed by the name of the class, followed by the code that makes up the class, surrounded by curly brackets ({ }).

A common coding standard is to begin a class name with a capital letter, though you don't have to do this. The main thing is to be consistent.

Now that you've defined a class, you can create objects based on the class. To create an object, you use the new keyword, followed by the name of the class that you want to base the object on. You can then assign this object to a variable, much like any other value. Here's an example that shows how to create objects:

```
class Car {

// METHODS AND PROPERTIES GO HERE
}

$beetle = new Car();
$mustang = new Car();
print_r( $beetle ); // DISPLAYS "CAR OBJECT ( )"
print_r( $mustang ); // DISPLAYS "CAR OBJECT ( )"
```

This code first defines the empty Car class as before, then creates two new instances of the Car class — that is, two Car objects. It assigns one object to a variable called $beetle, and another to a variable called $mustang. Note that, although both objects are based on the same class, they are independent of each other, and each is stored in its variable.

Once the objects have been created, their contents are displayed using print_r(). Print_r() can be used to output the contents of arrays. It can also be used to output objects, which is very handy for debugging object-oriented code. In this case, the Car class is empty, so print_r() merely displays the fact that the objects are based on the Car class.

PHP passes variables to and from functions by value and assigns them to other variables by value unless you explicitly tell it to pass them or assign them by reference. The exception to this rule is objects, which are always passed by reference.

Creating and Using Properties

Now that you know how to create a class, you can start adding properties to it. Class properties are very similar to variables; for example, an object's property can store a single value, an array of values, or even another object.

Understanding Property Visibility

Before diving into creating properties in PHP, it's worth taking a look at an important concept of classes known as visibility. Each property of a class in PHP can have one of three visibility levels, known as public, private, and protected:

Declaration	Use
Public	Can be accessed by any code, whether that code is inside or outside the class. If a property is declared public, its value can be read or changed from anywhere in your script.

Declaration	Use
Private	Properties of a class can be accessed only by the code inside the class. So, if you create a private property. Only the methods inside the same class can access its contents.
Protected	These properties are a bit like private properties in that they can't be accessed by code outside the class, but there's one subtle difference any class that inherits from the class can also access the properties.

It’s a good idea to avoid creating public properties wherever possible. Instead, it's safer to create private properties, then to create methods that allow code outside the class to access those properties. This means that you can control exactly how your class's properties are accessed. You learn more about this concept later in this lesson.

Declaring Properties

To add a property to a class, first write the keyword public, private, or protected — depending on the visibility level you want to give to the property — followed by the property's name (preceded by a $ symbol):

```
class MyClass {
   public $property1; // THIS IS A PUBLIC PROPERTY
   private $property2; // THIS IS A PRIVATE PROPERTY
   protected $property3; // THIS IS A PROTECTED PROPERTY
}
```

By the way, you can also initialize properties at the time that you declare them, much like you can with variables:

```
class MyClass {
   public $widgetsSold = 123;
}
```

In this case, whenever a new object is created from MyClass, the object's $widgetsSold property defaults to the value 123.

Accessing Properties

Once you've created a class property, you can access the corresponding object's property value from within your calling code by using the following syntax:

```
$object->property;
```

That is, you write the name of the variable storing the object, followed by an arrow symbol composed of a hyphen (-) and a greater - than symbol (>), followed by the property name. (Note that the property name doesn't have a $ symbol before it.)

Here’s an example that shows how to define properties then set and read the values. The print_r will show the object:

```
class Car {
    public $color;
    public $manufacturer;
}

$beetle = new Car();
$beetle->color = "red";
$beetle->manufacturer = "Volkswagen";
$mustang = new Car();
$mustang->color = "green";
$mustang->manufacturer = "Ford";
echo $beetle->color."\n";
echo $beetle->manufacturer."\n";
echo $mustang->color."\n";
echo $mustang->manufacturer."\n";
print_r($beetle);
print_r($mustang);
```

Static Properties

You can also create a static property, by adding the static keyword just before the property name:

```
class MyClass {
```

```
    public static $myProperty;
}
```

Static members of a class are independent of any object derived from that class. To access a static property, you write the class name, followed by two colons (::), followed by the property name (preceded by a $ symbol):

```
MyClass::$myProperty = 123;
```

Here's an example using the Car class:

```
class Car {
public $color;
public $manufacturer;
static public $numberSold = 123;
}
Car::$numberSold++;
echo Car::$numberSold;
```

Within the Car class, a static property, $numberSold, is declared and initialized to 123. Then, outside the class definition, the static property is incremented and its new value, 124, is displayed. Static properties are useful when you want to record a persistent value that’s relevant to a class, but that isn’t related to specific objects. You can think of them as global variables for classes. The nice thing about static properties is that code outside the class doesn’t have to create an instance of the class — that is, an object — to access the property.

Class Constants

PHP lets you create constants within classes. To define a class constant, use the keyword const, as follows:

```
class MyClass {
   const MYCONST = 123;
}
```

As with normal constants, it's good practice to use all - uppercase letters for class constant names. Like static properties, you access class constants via the class name and the:: operator:

```
echo MyClass::MYCONST;
```

Class constants are useful whenever you need to define a fixed value, or set a configuration option, that's specific to the class in question. For example, for the Car class you could define class constants to represent various types of cars, then use these constants when creating Car objects:

```
class Car {
   const HATCHBACK = 1;
   const STATION_WAGON = 2;
   const SUV = 3;
   public $model;
   public $color;
   public $manufacturer;
   public $type;
}

$myCar = new Car();
$myCar->model = "Dodge Caliber";
$myCar->color = "blue";
$myCar->manufacturer = "Chrysler";
$myCar->type = Car::HATCHBACK;

echo "This $myCar->model is a ";

switch ( $myCar->type ) {
   case Car::HATCHBACK: echo "hatchback"; break;
   case Car::STATION_WAGON: echo "station wagon"; break;
   case Car::SUV: echo "SUV";break;
```

```
}
```

In this example, the Car class contains three class constants — HATCHBACK, STATION_WAGON, and SUV — that are assigned the values 1, 2, and 3, respectively. These constants are then used when setting and reading the $type property of the $myCar object.

Working with Methods

Up to this point, the classes and objects you've created have mostly consisted of properties. As such, they're not much use, except as glorified associative arrays. It's when you start adding methods to classes that they become truly powerful. An object then becomes a nicely encapsulated chunk of functionality, containing both data and the methods to work on that data.

As mentioned earlier, a method is much like a function, except that it's tied to a specific class.

Method Visibility

Earlier you learned that a property can have three visibility levels: public, private, and protected. The same is true of methods. All methods can be called by other methods within the same class. If a method is declared public, any code outside the class definition can also potentially call the method. However, if a method is declared private, only other methods within the same class can call it. Finally, a protected method can be called by other methods in the class, or in a class that inherits from the class.

Creating a Method

To add a method to a class, use the public, private, or protected keyword, then the function keyword, followed by the method name, followed by parentheses. You then include the method's code within curly braces:

```
class MyClass {
  public function aMethod() {

      // WRITE CODE HERE
  }
```

```
}
```

You can optionally leave out the public, private, or protected keyword. If you do this, public is assumed.

Calling Methods

To call an object's method, simply write the object's name, then the same arrow used for accessing properties (->), then the method name followed by parentheses:

```
$object->method();
```

Here's a simple example that creates a class with a method, then creates an object from the class and calls the object's method:

```
class MyClass {
   public function hello() {
      echo "Hello, World!";
   }
}
$obj = new MyClass;
$obj->hello();
```

Adding Parameters and Returning Values

As with functions, you can add parameters to a method so that it can accept arguments to work with. A method can also return a value, just like a function. You add parameters and return values in much the same way as with functions. To add parameters, specify the parameter names between the parentheses after the method's name:

```
public function aMethod( $param1, $param2 ) {

    // WRITE CODE HERE
}
```

To return a value from a method — or to simply exit a method immediately — use the return keyword:

```
public function aMethod( $param1, $param2 ) {

    // WRITE CODE HERE
  return true;
```

```
}
```

Accessing Object Properties from Methods

Although you can happily pass values to and from a method using parameters and return values, much of the power of OOP is realized when objects are as self-contained as possible. This means that an object's methods should ideally work mainly with the properties of the object, rather than relying on outside data to do their job.

To access an object's property from within a method of the same object, you use the special variable name "$this", as follows:

```
$this->property;
```

For example:

```
class MyClass {
   public $greeting = "Hello, World!";
   public function hello() {
      echo $this->greeting;
   }
}
$obj = new MyClass;
$obj->hello(); // DISPLAYS "HELLO, WORLD!"
```

In this example, a class, MyClass, is created, with a single property, $greeting, and a method, hello(). The method uses echo to display the value of the $greeting property accessed via $this ->greeting. After the class definition, the script creates an object, $obj, from the class, and calls the object's hello() method to display the greeting.

Note that the $this inside the hello() method refers to the specific object whose hello() method is being called — in this case, the object stored in $obj. If another object, $obj2, were to be created from the same class and its hello() method called, the $this would then refer to $obj2 instead, and therefore $this ->greeting would refer to the $greeting property of $obj2. By the way, you can also use $this to call an object's method from within another method of the same object:

```
class MyClass {
   public function getGreeting() {
```

```
        return "Hello, World!";
    }
    public function hello() {
        echo $this->getGreeting();
    }
}
$obj = new MyClass;
$obj->hello(); // DISPLAYS "HELLO, WORLD!"
```

Here, the hello() method uses $this->getGreeting() to call the getGreeting() method in the same object, then displays the returned greeting string using echo.

IMPORTANT NOTE: In these examples, we were using echo in the class but in most cases, you will return a value from a method and echo that returned value when the method is called from the object.

```
class MyClass {
    public $greeting = "Hello, World!";
    public function hello() {
        return $this->greeting;
    }
}
$obj = new MyClass;
echo $obj->hello(); // DISPLAYS "HELLO, WORLD!"
```

Static Methods

PHP lets you create static methods in much the same way as static properties. To make a method static, add the static keyword before the function keyword in the method definition:

```
class MyClass {
    public static function staticMethod() {

            // WRITE CODE HERE
    }
}
```

To call a static method, write the class name, followed by two colons, followed by the method name and the arguments (if any) in parentheses:

```
MyClass::staticMethod();
```

As with static properties, static methods are useful when you want to add some functionality that's related to a class, but that doesn't need to work with an actual object created from the class. Here's a simple example:

```
class Car {
   public static function calcMpg( $miles, $gallons ) {
      return ( $miles / $gallons );
   }
}

echo Car::calcMpg( 168, 6 );
```

The calcMpg() method takes two arguments — miles traveled and gallons of fuel used — and returns the calculated miles per gallon. The method is then tested by calling it with some sample figures and displaying the result.

The calcMpg() method doesn't depend on an actual object to do its job, so it makes sense for it to be static. Notice that the calling code doesn't need to create a Car object to use calcMpg(). If you need to access a static method or property, or a class constant, from within a method of the same class, you can use the same syntax as you would outside the class:

```
class MyClass {
   const MYCONST = 123;
   public static $staticVar = 456;
   public function myMethod() {
      echo "MYCONST = " . MyClass::MYCONST . ", ";
      echo "\$staticVar = " . MyClass::$staticVar;
   }
}
$obj = new MyClass;
$obj->myMethod();
```

You can also use the self-keyword (much as you use $this with objects):

```
class Car {
   public static function calcMpg( $miles, $gallons ) {
      return ( $miles / $gallons );
   }
```

```
    public static function displayMpg( $miles, $gallons ) {
        echo "This car's MPG is: " . self::calcMpg( $miles, $gallons );
    }
}
echo Car::displayMpg( 168, 6 );
```

Encapsulation

So far, most of the classes you've created in this chapter have contained public properties, and outside code has been able to reach into a class's innards and manipulate its public properties at will. Usually, this is a bad idea. One of the strengths of OOP is the concept of encapsulation. This means that a class's internal data should be protected from being directly manipulated from outside and that the details of the class's implementation — such as how it stores values or manipulates data — should be hidden from the outside world. By doing this, you gain two advantages:

You can change your class's implementation details at any time without affecting code that uses the class You can trust the state of an object to be valid and to make sense, all internal properties of a class should be declared private. If outside code needs to access those variables, it should be done through a public method. This allows your class to validate the changes requested by the outside code and accept or reject them.

For example, if you're building a banking application that handles details of customer accounts, you might have an Account object with a property called $totalBalance and methods called makeDeposit() and makeWithdrawal(). The only way to affect the balance should be to make a withdrawal or a deposit. If the $totalBalance property is implemented as a public property, you could write outside code that would increase the value of that variable without having to make a deposit. This would be bad for the bank.

Instead, you implement this property as a private property and provide a public method called getTotalBalance(),which returns the value of that private property:

```
class Account {
    private $_totalBalance = 0;

    public function makeDeposit( $amount ) {
        $this->totalBalance += $amount;
    }
```

```
    public function makeWithdrawal( $amount ) {
        if ( $amount < $this->totalBalance ) {
            $this->totalBalance -= $amount;
        } else {
            die( "Insufficient funds" );
        }
    }

    public function getTotalBalance() {
        return $this->totalBalance;
    }
}

$a = new Account;
$a->makeDeposit( 500 );
$a->makeWithdrawal( 100 );
echo $a->getTotalBalance();
$a->makeWithdrawal( 1000 );
```

Because the variable storing the account balance is private, it can't be manipulated directly. Customers have to use makeDeposit() if they want to increase the value of their account. By encapsulating internal data and method implementations, an object-oriented application can protect and control access to its data and hide the details of implementation, making the application more flexible and more stable.

Using Inheritance to Extend the Power of Objects

So far, all the classes you've created have been self-contained. However, objects get interesting when you start using inheritance. Using this technique, you can create classes — known as child classes — that are based on another class: the parent class. A child class inherits all the properties and methods of its parent, and it can also add additional properties and methods.

The wonderful thing about inheritance is that, if you want to create a lot of similar classes, you must write the code that they have in common only once, in the parent class. This saves you from duplicating code. Furthermore, any outside code that can

work with the parent class automatically can work with its child classes, provided the code works only with the properties and methods contained in the parent class.

Imagine that you're creating a program to deal with various regular shapes, such as circles, squares, equilateral triangles, and so on. You want to create a Shape class that can store information such as the number of sides, side length, radius, and color, and that can calculate values such as the shape's area and perimeter. However, not all shapes are the same. Circles don't have a clearly defined number of sides, and you calculate an equilateral triangle's area using a different formula than for a square. So if you wanted to handle all types of regular shapes in a single Shape class, your class's code would get quite complicated.

By using inheritance, however, you can break the problem down into simpler steps. First, you create a parent Shape class that contains just those properties and methods that are common to all shapes. Then, you can create child classes such as Circle, Square, and Triangle that inherit from the Shape class. To create a child class that's based on a parent class, you use the “extends” keyword, as follows:

```
class Shape {

    // GENERAL SHAPE PROPERTIES AND METHODS HERE
}
class Circle extends Shape {

    // CIRCLE-SPECIFIC PROPERTIES AND METHODS HERE
}
```

The following script shows inheritance in action. It creates a parent Shape class, holding properties and methods common to all shapes, then creates two child classes based on Shape — Circle, and Square — that contain properties and methods related to circles and squares, respectively.

```
class Shape {
   private $_color = "black";
   private $_filled = false;
   public function getColor() {
      return $this->color;
   }
   public function setColor( $color ) {
```

```
        $this->color = $color;
    }
    public function isFilled() {
        return $this->filled;
    }
    public function fill() {
        $this->filled = true;
    }
    public function makeHollow() {
        $this->filled = false;
    }
}
class Circle extends Shape {
    private $_radius = 0;
    public function getRadius() {
        return $this->radius;
    }
    public function setRadius( $radius ) {
        $this->radius = $radius;
    }
    public function getArea() {
        return M_PI * pow( $this->radius, 2 );
    }
}

class Square extends Shape {
    private $_sideLength = 0;
    public function getSideLength() {
        return $this->sideLength;
    }
    public function setSideLength( $length ) {
        $this->sideLength = $length;
    }
    public function getArea() {
        return pow( $this->sideLength, 2 );
    }
}

$myCircle = new Circle;
$myCircle->setColor( "red" );
$myCircle->fill();
$myCircle->setRadius(4);

echo "My Circle\n";
```

```
echo "My circle has a radius of " . $myCircle->getRadius()."\n";
echo "It is " . $myCircle->getColor() . " and it is " . ( $myCircle->isFilled() ? "filled" : "hollow"
)."\n";
echo "The area of my circle is: " . $myCircle->getArea()."\n";
$mySquare = new Square;
$mySquare->setColor("green");
$mySquare->makeHollow();
$mySquare->setSideLength(3);
echo "My Square\n";
echo "My square has a side length of " . $mySquare->getSideLength()."\n";
echo "It is " . $mySquare->getColor() . " and it is " . ( $mySquare->isFilled() ? "filled" : "hollow"
)."\n";
echo "The area of my square is: " . $mySquare->getArea()."\n";
```

How It Works

The script first creates the parent Shape class. This class contains just the properties and methods common to all shapes. It contains private properties to store the shape's color and record whether the shape is filled or hollow, then provides public accessor methods to get and set the color, as well as fill the shape or make it hollow and retrieve the shape's fill status.

Next, the script creates a Circle class that inherits from the Shape class. Remember that a child class inherits all the properties and methods of its parent. The Circle class also adds a private property to store the circle's radius and provides public methods to get and set the radius, as well as calculate the area from the radius.

The script then creates Square, another class that inherits from Shape. This time, the class adds a private property to track the length of one side of the square and provides methods to get and set the side length and calculate the square's area.

Finally, the script demonstrates the use of the Circle and Square classes. First, it creates a new Circle object, sets its color, fills it, and sets its radius to 4. It then displays all the properties of the circle and calculates its area using the getArea() method of the Circle class. Notice how the script calls some methods that are in the parent Shape class, such as setColor() and isFilled(), and some methods that are in the child Circle class, such as setRadius() and getArea(). The script then repeats the process with the Square class, creating a hollow green square with a side length of 3, then displaying the square's properties and calculating its area using the Square class's getArea() method.

Overriding Methods in the Parent Class

What if you want to create a child class whose methods are different from the corresponding methods in the parent class? For example, you might create a class called Fruit that contains methods such as peel(), slice(), and eat(). This works for most fruit; however, grapes, for example, don't need to be peeled or sliced. So you might want your Grape object to behave somewhat differently from the generic Fruit object if you try to peel or slice it.

PHP, like most object-oriented languages, lets you solve such problems by overriding a parent class's method in the child class. To do this, simply create a method with the same name in the child class. Then, when that method name is called for an object of the child class, the child class's method is run instead of the parent class's method:

```
class ParentClass {
   public function someMethod() {

          // WRITE CODE HERE
   }
}

class ChildClass extends ParentClass {
   public function someMethod() {

          // THIS METHOD IS CALLED INSTEAD OF THE PARENT CLASS WITH A
METHOD OF THE SAME NAME
   }
}
```

Notice that the parent class's method is called when accessed from an object of the parent class, and the child class's method is called when using an object of the child class.

The following example code shows how you can use inheritance to distinguish grapes from other fruit:

```
class Fruit {
   public function peel() {
      echo "I'm peeling the fruit...\n";
   }
   public function slice() {
      echo "I'm slicing the fruit...\n";
   }
   public function eat() {
      echo "I'm eating the fruit. Yummy!\n";
   }
   public function consume() {
      $this->peel();
      $this->slice();
      $this->eat();
   }
```

```
}
class Grape extends Fruit {
    public function peel() {
        echo "No need to peel a grape!\n";
    }
    public function slice() {
        echo "No need to slice a grape!\n";
    }
}

echo "Consuming an apple...\n";
$apple = new Fruit;
$apple->consume();
echo "Consuming a grape...\n";
$grape = new Grape;
$grape->consume();
```

Taking the previous Fruit and Grape example, say you want to create a Banana class that extends the Fruit class. Generally, you consume a banana like any other fruit, but you also need to break the banana off from a bunch of bananas first. So within your Banana class, you can override the parent's consume() method to include functionality to break off a banana, then call the overridden consume() method from within the Banana class's consume() method to finish the consumption process:

```
class Fruit {
    public function consume(){
        echo "I’m peeling the fruit...\n";
        echo "I’m slicing the fruit...\n";
        echo "I’m eating the fruit. Yummy!\n";
    }
}
class Banana extends Fruit {
    public function consume() {
        echo "I'm breaking off a banana\n";
        parent::consume();
    }
}
$banana = new Banana;
$banana->consume();
```

```
    public function setHeight( $height ) {
        $this->height = $height;
    }

    public function getArea() {
        return $this->width * $this->height;
    }
}
```

Constructor Functions

When creating a new object, often it's useful to set up certain aspects of the object at the same time. For example, you might want to set some properties to initial values, fetch some information from a database to populate the object, or register the object in some way.

Setting Up New Objects with Constructors

Normally, when you create a new object based on a class, all that happens is that the object is brought into existence. (Usually, you then assign the object to a variable or pass it to a function.) By creating a constructor method in your class, however, you can cause other actions to be triggered when the object is created.

NOTE: You cannot overload constructors in PHP.

To create a constructor, simply add a method with the special name _construct() to your class. (That's two underscores, followed by the word "construct, " followed by parentheses.) PHP looks for this special method name when the object is created; if it finds it, it calls the method.

Here's a simple example:

```
class MyClass {
    function __construct() {
        echo "Whoa! I've come into being. < br / > ";
    }
}

$obj = new MyClass; // DISPLAYS "WHOA! I'VE COME INTO BEING."
```

The class, MyClass, contains a very simple constructor that just displays the message. When the code then creates an object from that class, the constructor is called, and the message is displayed.

You can also pass arguments to constructors, just like normal methods. This is great for setting certain properties to initial values at the time the object is created. The following example shows this principle in action:

IMPORTANT NOTE: The underscore used in the variable names ($_firstname) are just used as a naming convention to show that they are private variables. It is not something that is required for the script to work.

```
class Person {
    private $_firstName;
    private $_lastName;
    private $_age;
    public function __construct( $firstName, $lastName, $age ) {
        $this->_firstName = $firstName;
        $this->_lastName = $lastName;
        $this->_age = $age;
    }

    public function showDetails() {
        echo "$this->_firstName $this->_lastName, age $this->_age\n";
    }
}

$p = new Person( "Harry", "Walters", 28 );

$p->showDetails(); // Displays "Harry Walters, age 28"
```

The Person class contains three private properties and a constructor that accepts three values, setting the three properties to those values. It also contains a showDetails() method that displays the property values. The code creates a new Person object, passing in the initial values for the three properties. These arguments get passed directly to the _construct() method, which then sets the property values accordingly. The last line then displays the property values by calling the showDetails() method.

If a class contains a constructor, it is only called if objects are created specifically from that class; if an object is created from a child class, only the child class's constructor is called. However, if necessary you can make a child class call its parent's constructor with parent::__construct().

```
class MyClass {
   function __construct() {
      echo "Hello Class!";
   }
}

//USES THE PARENT CLASS CONSTRUCT
class Test1 extends MyClass {
}

//OVERRIDES THE PARENT CLASS CONSTRUCT
class Test2 extends MyClass {
   function __construct(){
      echo "My name is Scott";
   }
}


//USES THE PARENT CLASS AND ITS OWN CONTRUCTOR
class Test3 extends MyClass {
   function __construct(){
      parent::__construct();
      echo ", Welcome To PHP";
   }
}
$tst1 = new Test1();
echo "\n";
$tst2 = new Test2();
echo "\n";
$tst3 = new Test3();
```

PHP Interfaces

An Interface enables us to make programs, indicating the public methods that a class must execute, without including the complexities and procedure of how the specific methods are implemented. This implies that an interface can define method names and arguments, but not the contents of the methods. Any classes implementing an interface must implement all methods defined by the interface.

Interfaces are characterized similarly as a class, however, only the interface keyword replaces the class phrase in the declaration and without any of the methods having their contents defined.

Classes that use the interface must implement the interface using the "implement" keyword. Also, unlike extending classes where one class can only extend one class, a class have multiple interfaces.

Below is a sample code of how it works

```
<?php

/*THIS IS HOW WE CREATE THE INTERFACES NOTICE WE GIVE THEM METHODS
THAT ARE JUST METHOD NAMES AND HAVE NO CONTENT.  THE METHODS WILL BE
DEFINED IN THE CLASS THAT IMPLEMENTS THE INTERFACE*/


interface Characteristics {
   public function sound();
   public function describe();
}

interface Owner {
   public function setOwnersName($name);
   public function getOwnersName();
}


class Animal {
   private $_legs;
   private $_wings;

   public function __construct($legs, $wings) {
```

```
        $this->_legs = $legs;
        $this->_wings = $wings;
    }

    public function getLegs(){
        return $this->_legs;
    }

    public function getWings(){
        return $this->_wings;
    }
}


/*NOTICE HOW THIS CLASS EXTENDS ANIMAL AND IMPLEMENTS BOTH
CHARACTERISTICS AND OWNER*/


class Dog extends Animal implements Characteristics, Owner{
    private $_name;


/*THE FOLLOWING FOUR FUNCTIONS ARE FROM THE INTERFACES BUT THE METHODS
CONTENT IS WRITTEN IN THE CLASS.  BECAUSE WE IMPLEMENTED THE
INTERFACE, WE HAVE TO USE ALL THE METHODS NAMED IN THE INTERFACE*/


    public function sound(){
        return "woof, woof";
    }

    public function setOwnersName($name){
        $this->_name = $name;
    }

    public function getOwnersName(){
        return $this->_name;
    }

    public function describe(){
        return "<p>This animal has {$this->getLegs()} legs and {$this-
>getWings()} wings and makes a {$this->sound()} sound.  The owners name is {$this-
>getOwnersName()}.</p>";
    }
}
```

```
//HERE WE JUST IMPLEMENT ONE INTERFACE
class Cat extends Animal implements Characteristics{
   public function sound(){
      return "meow";
   }


/*NOTICE THE DESCRIBE HERE IS DIFFERENT THAN THAT OF THE DOG CLASS.
WE CAN DO THIS BECAUSE WE ARE USING INTEFACES AND THE CLASS DESCRIBES
WHAT THE METHOD WILL DO. SO EACH CLASS DOG AND CAT HAVE A DIFFERENT
FUNCTION FOR THE DESCRIBE METHOD.*/


   public function describe(){
      return "<p>This animal has {$this->getLegs()} legs and {$this-
>getWings()} wings and makes a {$this->sound()} sound</p>";
   }
}

$dog = new Dog(4, 0);
$dog->setOwnersName("Scott");
echo $dog->describe();

$dog = new  Cat(4, 0);
echo $dog->describe();

?>
.
```

Chapter 7: PHP and HTML

Integrating PHP with HTML

Introduction

You can add PHP to HTML seamlessly. PHP can be added anywhere within HTML and can be added in multiple places. What needs to be done is all PHP needs to be in code blocks as shown:

```
<?php .... ?>
```

Just because you can add PHP anywhere within your HTML there are some best practices you should follow. Many PHP sites have the behavior and structure mixed which makes maintaining and updating the page difficult.

The recommendation is to write as much of your php above the doctype of the webpage and anything that needs to be displayed will be stored in variables, which will be outputted in the HTML.

Below is an example where I loop through a list of names, put them into an unordered list, and display the result in the webpage.

```
<?php
$names = ["Scott","Karen","Mike","Judy","John","James","Steve"];
$output = "<ul>";
foreach ($names as $name) {
  $output .= "<li>{$name}</li>";
}
$output .= "</ul>";
?>

<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
```

```
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css"
integrity="sha384-
Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm"
crossorigin="anonymous">
    <title>Name List</title>
  </head>
  <body>
    <main class="container">
      <h1>List of names</h1>
      <?php echo $output; ?>
    </main>
  </body>
</html>
```

Using Include and Require Statements

As you can see the functionality is separated from the view. We can separate if further using the include, include_once, require, require_once statements.

include - The include statement includes and evaluates the specified file. If the file is not found or fails, then whatever uses the include will be ignored but the script will render

include_once - The include_once statement includes and evaluates the specified file during the execution of the script. This is a behavior 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

require - Require is identical to include except upon failure it will also produce a fatal E_COMPILE_ERROR level error and halt the script

require_once - The require_once statement is identical to require except PHP will check if the file has already been included, and if so, will not include (require) it again.

NOTE: These are "statements" not functions so they should be written like require 'filename';. They can also be written as functions require('filename') but that is not recommended. Also, single or double quotes can be used it does not matter.

I recommend using the require_once statement in all cases.

I have the same example below but instead of having the PHP written at the top of the page I have abstracted it to another file named nameList.php the PHP file is shown below.

```
<?php
$names = ["Scott","Karen","Mike","Judy","Aaron","Bronson","Brevick"];
$output = "<ul>";
foreach ($names as $name) {
  $output .= "<li>{$name}</li>";
}
$output .= "</ul>";
```

Below is the code for the HTML page. Notice I used the require_once to include the file. I tend to always use require once especially when I use classes.

```
<?php
require_once 'php/nameList.php';
?>
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css"
integrity="sha384-
Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm"
crossorigin="anonymous">
    <title>Name List</title>
  </head>
  <body>
    <main class="container">
      <h1>List of names</h1>
      <?php echo $output; ?>
    </main>
    </body>
```

Calling Functions

Including and requiring files allows you to access all functionality of a file. So, we can call functions as well. I have modified the following PHP file to wrap the code in a function.

```
<?php
function getNames(){
        $names = ["Scott","Karen","Mike","Judy","Aaron","Bronson","Brevick"];
        $output = "<ul>";
        foreach ($names as $name) {
          $output .= "<li>{$name}</li>";
        }
        $output .= "</ul>";

        return $output;
}
```

The HTML was modified as well

```
<?php
require_once 'php/nameListFunction.php';
$output = getNames();
?>
<!doctype html>
<html lang="en">
  <head>
   <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css"
integrity="sha384-
Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm"
crossorigin="anonymous">
    <title>Name List</title>
  </head>
  <body>
    <main class="container">
      <h1>List of names</h1>
      <?php echo $output; ?>
    </main>
  </body>
```

Including and Requiring Classes

You can include classes as well. I have modified the PHP file to be a class.

```
<?php
class NameList{
        public function getNames(){
                $names = ["Scott","Karen","Mike","Judy","John”,“Abby","Bobbie"];
                $output = "<ul>";
                foreach ($names as $name) {
                  $output .= "<li>{$name}</li>";
                }
                $output .= "</ul>";
                return $output;
        }
}
```

The HTML was modified as well

```
<?php
require_once 'php/nameListClass.php';
$names = new NameList();
$output = $names->getNames();
?>
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css"
integrity="sha384-
Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm"
crossorigin="anonymous">

    <title>Name List</title>
  </head>
  <body>
    <main class="container">
      <h1>List of names</h1>
      <?php echo $output; ?>
    </main>
  </body>
```

The examples files found at https://GitHub.com/sshaper/cps276_examples/tree/master/php_web, are the actual example files used for all of the above code. The working examples can be found at http://198.199.80.235/cps276/cps276_examples/php_web

PHP Forms

Introduction

A key part of most PHP applications is the ability to accept input from the person using the application. So far, all the scripts you've created haven't allowed for any user input at all; to run the script, you merely type its URL into your Web browser and watch it do its stuff.

By adding the ability to prompt the user for input and then read that input, you start to make your PHP scripts truly interactive. One of the most common ways to receive input from the user of a Web application is via an HTML form. You've probably filled in many HTML forms yourself. Common examples include contact forms that let you email a site owner; order forms that let you order products from an online store; and Web-based email systems that let you send and receive email messages using your Web browser.

Before looking at the PHP side of things, take a quick look at how an HTML form is constructed. An HTML form, or Web form, is simply a collection of HTML elements embedded within a standard Web page. By adding different types of elements, you can create different form fields, such as text fields, pull-down menus, checkboxes, and so on.

As stated in the "Forms" lectures, all Web forms start with an opening <form> tag, and end with a closing </form> tag:

```
< form action="myscript.php" method="post" >
< !-- Contents of the form go here -- >
< /form >
```

Notice that there are two attributes within the opening form tag:

action - tells the Web browser where to send the form data when the user fills out and submits the form. This should either be an absolute URL (such as http://www.example.com/ myscript.php) or a relative URL (such as myscript.php , /myscript.php , or ../ scripts/myscript.php). The script at the specified URL should be capable of accepting and processing the form data; more on this in a moment.

method - tells the browser how to send the form data. You can use two methods: get is useful for sending small amounts of data and makes it easy for the user to resubmit the form, and post can send much larger amounts of form data.

Once you've created your basic form element, you can fill it with various elements to create the fields and other controls within your form (as well as other HTML elements such as headings, paragraphs, and tables, if you so desire).

Capturing Form Data with PHP

You now know how to create an HTML form, and how data in a form is sent to the server. How do you write a PHP script to handle that data when it arrives at the server? First of all, the form's action attribute needs to contain the URL of the PHP script that will handle the form. For example:

```
< form action="form_handler.php" method="post" >
```

Next, of course, you need to create the form_handler.php script. When users send their forms, their data is sent to the server and the form_handler.php script is run. The script then needs to read the form data and act on it.

To read the data from a form, you use a few superglobal variables.

A superglobal is a built-in PHP variable that is available in any scope: at the top level of your script, within a function, or within a class method. We will look at two superglobal arrays:

Superglobal	Array Description
$_GET	Contains a list of all the field names and values sent by a form using the get method. Get methods send the data as a key-value pair by attaching them as a parameter to the URL. Get methods are not use for large amounts of data.

Superglobal	Array Description
$_POST	Contains a list of all the field names and values sent by a form using the post method. The data is sent when the request is made to the server for a file but is not attached to the URL. Post is normally used for sending data via a form.

Superglobal Array Description

Each of these two superglobal arrays contains the field names from the sent form as array keys, with the field values themselves as array values. For example, say you created a form using the post method, and that form contained the following control:

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

You could then access the value that the user entered into that form field using the $_POST superglobal.

```
$email = $_POST["emailAddress"];
```

You can see that the name attribute (emailAddress) is used as a key for the $_POST super global array. The value would be whatever the user entered into the form field.

Forms that Submit on Themselves

The first type of form we will look at is the form where it submits the data to itself, instead of a separate page. For example, if I had a form on a file named form.php and in the action attribute I entered form.php. The information from that form would be sent to form.php which is the same page the form was on, in other words, submitting the data to itself.

The best way to see this is through an example which can be found at:

http://198.199.80.235/cps276/cps276_examples/php-forms/form_send_data_to_itself.php

For the GitHub page it is:

https://GitHub.com/sshaper/cps276_examples/blob/master/php-forms/form_send_data_to_itself.php

A couple of things to note, with the above example:

1. I have pre-entered some values to save time.
2. I used a method called isset(). Isset() determines if a variable is set and is not null. Returns true if the variable exists and has a value other than null. False otherwise. I use isset() to see if my submit button has been clicked. If the submit button has been clicked then that will be sent as $_POST['submitButton'], where "submitButton" is the name value of the button. I use an if statement to check if the button is clicked. If it is, then the form information is displayed otherwise it skips that process and just displays the form.

Sending Form Data to Another Page

In the previous example, I sent the form data to itself. In this example, I will send the form data to another page.

You can view this example at http://198.199.80.235/cps276/cps276_examples/php-forms/form_send_data_to_another_page.html

For the GitHub page it is:

https://GitHub.com/sshaper/cps276_examples/blob/master/php-forms/form_send_data_to_another_page.html

A couple of things to note with the above example:

1. The page that contains the form has an .html file extension. If no PHP is written on the HTML page then you can just use a basic HTML page.
2. I do not need to use isset() because the only reason the PHP file would be accessed is if the button was clicked.
3. In this example, the data is just outputted but in most cases, the data would be processed and stored somewhere.

Header Redirect

There are times where you want to send information to the server and after the server reads the information it will send the user to another page. We do that using the header:

header() is used to send a raw HTTP header. Remember that header() must be called before any actual output is sent, either by normal HTML elements, blank lines in a file, or from PHP. It is a very common error to have code with include or require functions, or another file access function, and have spaces or empty lines that are outputted before a header() is called. The same problem exists when using a single PHP/HTML file.

In the following example, I will send some data to the server, but on the page, I send it to (based upon what I check) the user will be taken to an acknowledgment page or back to the form. The user will be sent back if any of the post values are blank.

You can see this example at:

http://198.199.80.235/cps276/cps276_examples/php-forms/form_redirect1.html

For the GitHub page it is:

https://GitHub.com/sshaper/cps276_examples/blob/master/php-forms/form_redirect1.html

Passing Data with Header Redirect

In our last example, we saw how to redirect to another page. But what if we wanted to redirect to another page and pass some information to that page. We can do that by adding the information to the web address.

IMPORTANT NOTE: Passing information via the web address is fine if it is short and not any sensitive information.

The following example will pass information (name) to the acknowledgment page:

http://198.199.80.235/cps276/cps276_examples/php-forms/form_redirect2.html

For the GitHub page it is:

https://GitHub.com/sshaper/cps276_examples/blob/master/php-forms/form_redirect2.html

A couple of things to note with the above example:

On the acknowledgment page that I send the header request to, I will be using a $_GET statement to receive the information from the web address. You will see that if you change the values in the web address and reload the page that reads those values will change.

Multiple buttons

What if you want a form to do something different based upon which button the user selected?

You can do that by assigning different name values to the button. In this example if the "Show Name" button is clicked then the names will be sent to the acknowledgment page. If "Dont Show Name" is clicked, then nothing will be sent to the acknowledgment page. Also, the acknowledgment page will have an isset() method added to it to determine if date has been attached to the web address or not. You can view the example at http://198.199.80.235/cps276/cps276_examples/php-forms/form_redirect3.html

The code can be found here

https://github.com/sshaper/cps276_examples/blob/master/php-forms/form_redirect3.html

Dealing with Multi - Value Fields

You learned earlier in the chapter that you can create form fields that send multiple values, rather than a single value. For example, the following form fields can send multiple values to the server:

```
<label for="favoriteWidgets">What are your favorite widgets?</label>
<select name="favoriteWidgets" id="favoriteWidgets" size="3" multiple>
        <option value="superWidget">The SuperWidget</option>
        <option value="megaWidget">The MegaWidget</option>
        <option value="wonderWidget">The WonderWidget</option>
</select>
<label for="newsletterWidgetTimes">Do you want to receive our 'Widget Times'
newsletter?</label>
<input type="checkbox" name="newsletter" id="newsletterWidgetTimes" value="widgetTimes" >
<label for="newsletterFunWithWidgets">Do you want to receive our 'Fun with Widgets'
newsletter?</label>
<input type="checkbox" name="newsletter" id="newsletterFunWithWidgets"
value="funWithWidgets" >
```

The first form field is a multi-select list box, allowing the user to pick one or more (or no) options. The second two form fields are checkboxes with the same name (newsletter) but different values (widgetTimes and funWithWidgets).

If the user checks both checkboxes then both values, widgetTimes, and funWithWidgets, are sent to the server under the newsletter field name.

So how can you handle multi-value fields in your PHP scripts? The trick is to add square brackets ([]) after the field name in your HTML form. Then, when the PHP engine sees a submitted form field name with square brackets at the end, it creates a nested array of values within the $_GET or $_POST superglobal array, rather than a single value. You can then pull the individual values out of that nested array. So, you might create a multi-select list control as follows:

```
<select name="favoriteWidgets[]" id="favoriteWidgets" size="3" multiple="multiple"> ... </select>
```

You'd then retrieve the array containing the submitted field values as follows:

```
$favoriteWidgetValuesArray = $_GET["favoriteWidgets"]; // IF USING GET METHOD
$favoriteWidgetValuesArray = $_POST["favoriteWidgets"]; // IF USING POST METHOD
```

This example found at http://198.199.80.235/cps276/cps276_examples/php-forms/form_multi_value_fields.html demonstrates how this is done.

The GitHub code is here

https://github.com/sshaper/cps276_examples/blob/master/php-forms/form_multi_value_fields.html

Creating File Upload Forms

As well as sending textual data to the server, Web forms can be used to upload files to the server. To upload a file, you click the button created by the file element and select a file. Then, when you submit the form, your browser sends the file to the server along with the other form data. The file element code is shown below.

```
<label for="fileSelectField" >A file select field </label>
<input type="file" name="fileSelectField" id="fileSelectField">
```

In addition, a form containing a file select field must use the post method, and it must also have an enctype="multipart/form - data" attribute in its form tag, as follows:

```
<form action="form_handler.php" method="post" enctype="multipart/form-data">
```

This attribute ensures that the form data is encoded as multipart MIME data — the same format that's used for encoding file attachments in email messages — which is required for uploading binary data such as files.

You can have as many files select fields as you like within your form, allowing your users to upload multiple files at once. Accessing Information on Uploaded Files Once the form data hits the server, the PHP engine recognizes that the form contains an uploaded file or files and creates a superglobal array called $_FILES containing various pieces of information about the file or files.

Each file is described by an element in the $_FILES array keyed on the name of the field that was used to upload the file. For example, say your form contained a file select field called photo:

```
<input type="file" name="photo">
```

If the user uploaded a file using this field, its details would be accessible via the following PHP array element:

```
$_FILES["photo"]
```

This array element is itself an associative array that contains information about the file. For example, you can find out the uploaded file's filename like this:

```
$filename = $_FILES["photo"]["name"];
```

Here's a full list of the elements stored in each nested array within the $_FILES array:

Array Element	Description
name	The filename of the uploaded file.
type	The MIME type of the uploaded file. For example, a JPEG image would probably have a MIME type of image/jpeg, whereas a QuickTime movie file would have a MIME type of video/quicktime.
size	The size of the uploaded file, in bytes.
tmp_name	The full path to the temporary file on the server that contains the uploaded file. (All uploaded files are stored as temporary files until they are needed.)
error	The error or status code associated with the file upload.

The error element contains an integer value that corresponds to a built-in constant that explains the status of the uploaded file. Possible values include:

Constant	Value	Meaning
UPLOAD_ERR_OK	0	The file was uploaded successfully.

Constant	Value	Meaning
UPLOAD_ERR_INI_SIZE	1	The file is bigger than the allowed file size specified in the upload_max_filesize directive in the php.ini file.
UPLOAD_ERR_FORM_SIZE	2	The file is bigger than the allowed file size specified in the MAX_FILE_SIZE directive in the form.
UPLOAD_ERR_NO_FILE	4	No file was uploaded.
UPLOAD_ERR_NO_TMP_DIR	6	PHP doesn’t have access to a temporary folder on the server to store the file.
UPLOAD_ERR_CANT_WRITE	7	The file couldn't be written to the server's hard disk for some reason.
UPLOAD_ERR_EXTENSION	8	The file upload was stopped by one of the currently loaded PHP extensions.

Most of these error codes are self - explanatory. UPLOAD_ERR_INI_SIZE and UPLOAD_ERR_FORM_SIZE are explained in the following section.

Limiting the Size of File Uploads Often it's a good idea to prevent unusually large files from being sent to the server. Apart from consuming bandwidth and hard disk space on the server, a large file can cause your PHP script to overload the server's CPU. For example, if your PHP script is designed to work on an uploaded 10 - kilobyte text file, uploading a 100 - megabyte text file might cause your script some problems.

PHP allows you to limit the size of uploaded files in a few ways. First, if you have access to your php.ini file, you can edit a directive called upload_max_filesize in the file:

```
Maximum allowed size for uploaded files.upload_max_filesize = 32M
```

Then, if a user tries to upload a file larger than this value (32 megabytes in this example), the file upload is canceled, and the corresponding error array element is set to UPLOAD_ERR_INI_SIZE.

You can check the size of an uploaded file manually and reject it if it's too large:

```
if ( $_FILES["photo"]["size"] > 10000 ) die( "File too big!" );
```

Storing and Using an Uploaded File

Once a file has been successfully uploaded, it is automatically stored in a temporary folder on the server. To use the file, or store it on a more permanent basis, you need to move it out of the temporary folder. You do this using PHP's move_uploaded_file() function, which takes two arguments: the path of the file to move, and the path to move it to.

You can determine the existing path of the file using the tmp_name array element of the nested array inside the $_FILES array. move_uploaded_file() returns trueif the file was moved successfully, or false if there was an error (such as the path to the file being incorrect). Below is an example:

```
if(move_uploaded_file( $_FILES["photo"]["tmp_name"], "/home/matt/photos/photo.jpg" )){
        echo "Your file was successfully uploaded.";
}
else{
        echo "There was a problem uploading your file - please try again.";
}
```

An example of a can be seen at http://198.199.80.235/cps276/cps276_examples/php-forms/file_upload.php

For the GitHub page it is:

https://GitHub.com/sshaper/cps276_examples/blob/master/php-forms/file_upload.php

PHP Try Catch

PHP Exception Handling Main Tips

- Using exceptions, you can **modify** the normal **flow** of your script in case a specific error occurs.
- Implemented with **PHP5**.

PHP Exception Handling Method

This is the common way things go when an exception is triggered:

- The state of the current script is saved.
- Script execution switches to the pre-defined function for exception handling.
- The handler then may continue the execution from the script state saved earlier, terminate the code flow entirely, or keep going from a different line of the code.

In this lesson, we will demonstrate the following methods of handling errors in PHP:

- Exceptions basic usage.
- Creation of a custom exception handler function.
- Using multiple exceptions.
- Exception re-throwing.
- Top-level exception handler set up.

Note: Exceptions are specifically meant for error handling, NOT jumping to another point in the script.

PHP Exception Handling Basics

Upon throwing the exception, PHP will stop the normal script flow and look for the first catch() block it encounters, which it will execute.

If there is no such block of code, a fatal error will occur, with the message “Uncaught Error”.

Try throwing an exception without a catch statement:

```
//creating a function, which throws an exception
function check_num($number) {
  if($number > 1) {
    throw new Exception("The value has to be 1 or lower");
  }
  echo "Number is 1 or less";
}

//triggering the exception
try{
  check_num(3);
}
```

When you run the above code, you get an error

Try, throw and catch

If we don't want our web application to crash each time an exception is caught, we have to write a statement, which may handle the exception.

A properly executed script for handling an exception should include:

1. Try – If you want a function that uses an exception, it goes into this block. In case of no exception triggers, the code continues as normal. However, in the case of an exception triggering, it is "thrown".
2. Throw – Using this block you trigger an exception. Every "throw" has to have at least a single "catch" block.
3. Catch – This block retrieves an exception and generates the object containing its data.

In the example below we try triggering an exception while we have the necessary blocks of code:

```
//CREATING A FUNCTION CONTAINING A POTENTIAL EXCEPTION
function check_num($number) {
  if($number > 1) {
    throw new Exception("The value has to be 1 or lower");
  }
  echo "Number is 1 or less";
}
```

```
//TRIGGERING THE EXCEPTION INSIDE A TRY()
try {
  check_num(2);
}

//CATCHING THE EXCEPTION
catch(Exception $e) {
  echo 'Message: ' .$e->getMessage();
}
```

This is the error we get now:

```
Message: Value must be 1 or below
```

Example explained

In the above example we throw an exception and handle it with a catch() block:

Function called check_num() is defined. Its function is to check whether a value is greater than 1. If this condition is met, we throw an exception.

1. This function is called in a try() block.
2. The throw() is in check_num() and gets triggered.
3. catch() receives the thrown exception and creates the $e object, which contains the exception data.
4. We echo the error message from the exception, by calling $e->getMessage() from the object containing exception data.

One of the ways to get around the rule saying that every exception must have a catch(), is to define a top tier exception handler class.

PHP 5 Exception Handling Create Custom Exception Class

To make a custom exception handler, you must create a class containing functions, meant for handling exceptions in PHP once they are triggered. This class has to be an Exception class extension.

Your custom exception class has all of the original exception class properties, but you can also add custom functions to it.

In the code example below, we create a custom exception class:

```
class custom_exception extends Exception {
  public function error_message() {
    //defining the error message
    $error_msg = 'Error caught on line '.$this->getLine()."\n in ".$this->getFile()."\n: ".$this-
>getMessage()."\n is not a valid E-Mail address";
    return $error_msg;
  }

}
$email = "someonesomeone@example.com";
try {
  //checking if
  if(filter_var($email, FILTER_VALIDATE_EMAIL) === FALSE) {
    //throwing an exception in case email is not valid
    throw new custom_exception($email);
  }
  else {
    echo "The email is valid";
  }
}
catch (custom_exception $e) {

    //DISPLAYING A CUSTOM MESSAGE
  echo $e->error_message();
}
```

This new class is an almost identical copy of the original exception class, however, now it has a custom error_message() function. Since it has all the methods and properties of the original class since it has all the methods and properties of the original class, we

can use every function contained in the normal exception class, such as GetLine(), GetFile(), GetMessage().

Example explained

The code example above throws an exception, which is caught using a custom exception class:

1. The class custom_exception() is made as an extension of the original class, inheriting every property and method from the original.
2. Function error_message() is created. Using this function, we make the code return an error message in case the e-mail address is invalid.
3. $email is a string, which is not a valid e-mail address.
4. A try() block executes and an exception is thrown.
5. The catch() block displays an error message.

PHP Exception Handling Multiple Exceptions

By using multiple exceptions, you can check multiple conditions.

This is done by using several if..else code blocks, a switch, or nesting multiple exceptions. Keep in mind, that each of these exceptions may use its own class and display their own error messages:

```
class custom_exception extends Exception {
  public function error_message() {
    //defining the error message
    $error_msg = 'Error caught on line '.$this->getLine()."\n in ".$this->getFile()
    ."\n: ".$this->getMessage()."\n is no valid E-Mail address";
    return $error_msg;
  }
}
$email = "someonesome@one.com";
try {
  //check if
  if(filter_var($email, FILTER_VALIDATE_EMAIL) === FALSE) {
    //throwing an exception in case email is not valid
    throw new custom_exception($email);
  }
  //checking for "example" in mail address
```

```
  if(strpos($email, "example") !== FALSE) {
    throw new Exception("$email contains example'");
  }
  else {
    throw new Exception("$email does not contain the word example");
  }
}
catch (custom_exception $e) {
  echo $e->error_message();
}
catch(Exception $e) {
  echo $e->getMessage();
}
```

Example explained

The above example tests two respective conditions and in case either of them isn't met, an exception is thrown:

1. The class custom_exception() is made as an extension of the original class, inheriting every property and method from the original.
2. The error_message() function is created. This function returns an error message if an e-mail address is invalid
3. $email is a string, which is a valid e-mail address, containing the word "example".
4. A try() block executes, but an exception isn't thrown.
5. On checking the second condition, an exception is thrown, since the $email variable contains the "example"; string value.
6. The catch() block displays an error message.
7. If the email does not contain the word "example" then the catch will demonstrate that.

In case the exception that was thrown belongs to the custom_exception class and there was one for this exception class, only the catch using the base exception class would handle the exception.

PHP Exception Handling Re-throwing Exceptions

At times, upon throwing an exception, you might want to handle it in a different than the standard way. You can throw the exception again within your catch() block.

These scripts are meant to hide system errors from users of the web application. The coder may want to be aware of system errors, but they aren't any good for the users. If you want to make the error message user-friendly, you may re-throw the exception to change the message:

```
class custom_exception extends Exception {
  public function error_message() {
    //error message
    $error_msg = $this->getMessage().' Does not contain “example”.';
    return $error_msg;
  }
}

$email = "someonesomeoneexample.com";
try {
  try {

      //CHECKING FOR "EXAMPLE" IN MAIL ADDRESS
    if(strpos($email, "example") === FALSE) {

        //THROWING AN EXCEPTION IF EMAIL DOES NOT CONTAIN EXAMPLE
      throw new Exception($email);
    }
  }
  catch(Exception $e) {

      //RE-THROWING EXCEPTION
    throw new custom_exception($email);
  }
}

catch (custom_exception $e) {

    //DISPLAY CUSTOM MESSAGE
  echo $e->error_message();
}
```

Example explained

The above code example tests whether the entered email-address has the string “example” as its value, in which case, the exception is re-thrown:

1. The class custom_exception() is made as an extension of the original class, inheriting every property and method from the original.
2. The error_message() function is created. This function returns an error message if an e-mail address does not contain an example
3. $email is a string, which is a valid e-mail address, containing the word “example”.
4. The try() contains another try() block, which is used for re-throwing our exception.
5. An exception is thrown since the $email variable does not contain the “example” string value.
6. The exception is caught a “custom_exception” is then thrown.
7. This exception displays a user-friendly error message.

In case the exception doesn’t run into a catch() block in its current try() block, it may search for a catch() in “higher levels”.

Check the complete set of error handling functions at PHP Error Handling Functions

Chapter 8: Files and Directories

PHP Directories

Create a directory

To create a new directory, call the mkdir() function, passing in the path of the directory you want to create. Note when creating a directory from the web you need to be in a directory with 777 permissions.

```
mkdir( "/home/matt/newfolder" );
```

Note, in this example, the parent directory must exist already (“/home/matt” in the example just shown) for the function to work. mkdir() returns true if the directory was created, or false if there was a problem. You can also set permissions for the directory at the time you create it by passing the mode as the second argument. For example, the following code creates a directory with read, write, and execute permissions granted to all users:

```
mkdir( "/home/matt/newfolder", 0777 );
```

VERY IMPORTANT! When setting file permissions as indicated above you may not get a true 0777, you may end up with 0755 or 0775. The reason is the umask, umask revokes, deletes permissions from system default, resulting in the permissions being less than you had set (usually 0022 or 0002 less depending on the umask default setting).

If you want to set a true 0777 permission, then it is recommended to use chmod (we will discuss chmod later in this section). Other techniques talk about setting umask to zero then changing it back, this is not recommended. So though you can set permissions (sort of) in the mkdir() function, if you want to have the correct permissions assigned to a directory, it should be done as shown:

```
mkdir( "/home/matt/newfolder");
chmod("/home/matt/newfolder", 0777);
```

There is a third parameter that allows the creation of nested directories specified in the pathname. It is a boolean where true is recursive and false is not. Default is false.

NOTE: If you are using the third parameter you must add the second parameter (file permissions), as shown below:

```
mkdir("/home/matt/newfolder", 0777, true);

chmod("/home/matt/newfolder", 0777);
```

IMPORTANT: File and directory modes only work on UNIX systems such as Linux and Mac OS; they have no effect when used on Windows machines.

Below is a list of parameters for the mkdir() function:

pathname - The directory path.

mode - The mode is 0777 by default, which means the widest possible access. Note: mode is ignored on Windows.

recursive - Allows the creation of nested directories specified in the pathname.

context - Added in 5.0.0 see streams

Remove a directory

You can remove a directory using rmdir(). The directory must be empty, and all relevant permissions must allow this. If PHP can't remove the directory — for example, because it's not empty — rmdir() returns false; otherwise, it returns true.

```
rmdir("/home/matt/myfolder");
```

Getting a Directory Path

The dirname() function returns the directory part of a given path. It complements the basename() function, which returns the filename portion of a given path.

For example:

```
$path = "/home/james/docs/index.html";
$directoryPath = dirname($path);
$filename = basename($path);
```

After running this code., $directoryPath contains “/home/james/docs”, and $filename holds “index.html”.

Create a file

To create a file, you use the touch. When you create a file using PHP you must put the file in a folder with 777 permissions. Below is an example.

```
touch("somedirectory/file.txt")
```

Remove a file

You can remove a file using unlink (string $filename [, resource $context]).

Glob

The glob() function searches for all the pathnames matching the pattern according to the rules used by the libc glob() function, which is similar to the rules used by common shells.

Parameters:

- Pattern - The pattern. No tilde expansion or parameter substitution is done.
    - GLOB_MARK - Adds a slash to each directory returned
    - GLOB_NOSORT - Return files as they appear in the directory (no sorting). When this flag is not used, the pathnames are sorted alphabetically
    - GLOB_NOCHECK - Return the search pattern if no files matching it were found
    - GLOB_NOESCAPE - Backslashes do not quote metacharacters

- GLOB_BRACE - Expands {a,b,c} to match 'a', 'b', or 'c'
- GLOB_ONLYDIR - Return only directory entries which match the pattern
- GLOB_ERR - Stop on read errors (like unreadable directories), by default errors are ignored.

In the following example I will get all the file names and sizes with the .txt extension, within the directory they are in. In this example glob is in the same directory as the files.

```
foreach (glob("*.txt") as $filename) {
    echo "$filename size " . filesize($filename) . "\n";
}
```

An example of creating, and removing directories can be found at:

Server: http://198.199.80.235/cps276/cps276_examples/php-directories/

GitHub: https://GitHub.com/sshaper/cps276_examples/tree/master/php-directories

PHP File and Directory Permissions

Changing Permissions

PHP's chmod() function is used to change the mode, or permissions, of a file or directory. It functions much like the UNIX chmod command.

This section applies mainly to UNIX-based Web servers such as Linux and Mac OS X. Windows servers do not have a concept of file and directory modes. Instead, you use Windows Explorer to set access permissions on files and folders by right-clicking the item, choosing Properties, then clicking the Security tab. You need to be an administrator to make these changes. If you're running your PHP scripts on a shared Windows server, and you need to set permissions on a certain file or folder, ask your hosting company for help. Often, they'll do it for you, or point you to a Web-based control panel where you can do it yourself. To change a file's permissions with chmod(), pass it the filename and the new mode to use. For example, to set a file's mode to 644, use:

```
chmod( "myfile.txt", 0644 );
```

The 0 (zero) before the 644 is important because it tells PHP to interpret the digits as an octal number. chmod() returns true if the permission change was successful, and false if it failed (for example, you're not the owner of the file).

So how do file modes work? Here's a quick primer.

File modes are usually expressed as octal numbers containing three digits. The first digit determines what the file's owner (usually the user that created the file) can do with the file. The second digit determines what users in the file's group (again, usually the group of the user that created the file) — can do with it. Finally, the last digit dictates what everyone else can do with the file.

Here the digits 7, 5, and 4 each individually represent the permissions for the user, group, and others, in that order. Each digit is a combination of the numbers 4, 2, 1, and 0:

4 Stands for read

2 Stands for write

1 Stands for execute

0 Stands for "no permission"

Here are some commonly used examples to make the concept of file modes clearer:

```
// OWNER CAN READ AND WRITE THE FILE; EVERYONE ELSE CAN JUST READ IT:
chmod( “myfile.txt”, 0644 );

// EVERYONE CAN READ AND WRITE THE FILE:
chmod( “myfile.txt”, 0666 );

// EVERYONE CAN READ AND EXECUTE THE FILE, BUT ONLY THE OWNER CAN
WRITE TO IT:
chmod( “myfile.txt”, 0755 );

// ONLY THE OWNER CAN ACCESS THE FILE, AND ONLY THEY CAN READ AND
WRITE TO IT:
chmod( “myfile.txt”, 0600 );
```

Note that you can only change the permissions of a file or directory if you own it, or if you’re the super-user (**which is highly unlikely for PHP scripts running on a Web server!**).

So how do modes work with directories? Well, to read the files in a directory, you need to have both read and execute permissions on that directory. Meanwhile, to create and delete files and subdirectories inside the directory, you need to have write and execute permissions on the directory. So, directories that you will be creating files within need to have 0777 permissions.

Checking File Permissions

Before you do something to a file in your script, it can be useful to know what kinds of things your script can do with the file. PHP provides three handy functions to help you out. To check if you're allowed to read a file, use is_readable(), passing in the filename of the file to check. Similarly, you can check that you're allowed to write to a file with is_writable(), and see if you can execute a file with is_executable(). Each function returns true if the operation is allowed, or false if it's disallowed. For example:

```
if (is_readable("myfile.txt") {
        echo "I can read myfile.txt";
}
if (is_writable("myfile.txt") {
        echo "I can write to myfile.txt";
}
if (is_executable("myfile.txt") {
        echo "I can execute myfile.txt";
}
```

PHP Files

Opening a file with fopen()

The fopen() function opens a file and returns a file handle associated with the file. The first argument passed to fopen() specifies the name of the file you want to open, and the second argument specifies the mode, or how the file is to be used. For example:

```
$handle = fopen( "./data.txt", "r" );
```

The first argument can be just a filename ("data.txt"), in which case PHP will look for the file in the current directory, or it can be a relative ("./data.txt") or absolute ("/myfiles/data.txt") path to a file. You can even specify a file on a remote Web or FTP server, as these examples show:

```
$handle = fopen( "http://www.example.com/index.html", "r" );
$handle = fopen( "ftp://ftp.example.com/pub/index.txt", "r" );
```

NOTE: A remote file can only be opened for reading — you can't write to the file.

If you're not familiar with command-line file operations, you might be a little confused by the concept of a current directory and the relative path notation. Usually, the current directory is the same directory as the script, but you can change this by calling chdir(). Within a relative path, a dot (.) refers to the current directory, and two dots (..) refer to the immediate parent directory. For example, ./data.txt points to a file called data.txt in the current directory, and ../data.txt points to a file called data.txt in the directory above the current directory. ../../../data.txt backs up the directory tree three levels before looking for the data.txt file. Meanwhile, an absolute path is distinguished by the fact that it begins with a / (slash), indicating that the path is relative to the root of the file system, not to the current directory. For example, /home/chris/website/index.php is an absolute path.

The second argument to fopen() tells PHP how you're going to use the file. It can take one of the following string values:

Value	Description
r	Open the file for reading only.
r+	Open the file for reading and writing.
w	Open the file for writing only. Any existing content will be lost. If the file does not exist PHP will attempt to create it.
w+	Open the file for reading and writing. Any existing content will be lost. If the file does not exist PHP will attempt to create it.
a	Open the file for appending at the end of the existing file. If the file does not exist PHP will attempt to create it.
a+	Open the file for reading and appending at the end of the existing file. If the file does not exist PHP will attempt to create it.

IMPORTANT NOTE: fopen will only use one of the second arguments. So, if you use "r+" you can "either" read or write from a file as shown.

```
$handle = fopen("../files/data.txt", "r+");
echo fread($handle, 1000);
fclose($handle);
```

Or to write you use:

```
$handle = fopen("../files/data.txt", "r+");
fwrite($handle, "Hello World");
fclose($handle);
```

The file pointer is PHP’s internal pointer that specifies the exact character position in a file where the next operation should be performed.

You can also append the value "b" to the argument to indicate that the opened file should be treated as a binary file (this is the default setting). Alternatively, you can append "t" to treat the file as a text file, in which case PHP attempts to translate end-of-line characters from or to the operating system's standard when the file is read or written. For example, to open a file in binary mode use:

```
$handle = fopen("data.txt","rb");
```

Although this flag is irrelevant for UNIX-like platforms such as Linux and Mac OS X, which treat text and binary files identically, you may find the text mode useful if you're dealing with files created on a Windows computer, which uses a carriage return followed by a line feed character to represent the end of a line (Linux and the Mac just use a line feed).

That said, binary mode is recommended for portability reasons. If you need your application's data files to be readable by other applications on different platforms, you should use binary mode and write your code to use the appropriate end-of-line characters for the platform on which you are running. (The PHP constant PHP_EOL is handy for this; it stores the end-of-line character(s) applicable to the operating system that PHP is running on.)

By default, if you specify a filename that isn't a relative or absolute path (such as "data.txt"), PHP just looks in the current (script) directory for the file. However, you can optionally pass the value true as the third argument to fopen(), in which case PHP will also search the include path for the file.

If there was a problem opening the file, fopen() returns false rather than a file handle resource. Operations on files and directories are prone to errors, so you should always allow for things to go wrong when using them. It's good practice to use some form of error-checking so that if an error occurs your script will handle the error gracefully. For example:

```
if (!($handle = fopen("./data.txt","r"))) die("Cannot open the file");

//RATHER THAN EXITING WITH DIE(), YOU MIGHT PREFER TO RAISE AN ERROR
OR THROW AN EXCEPTION.
```

Closing a File fclose()

Once you've finished working with a file, it needs to be closed. You can do this using fclose(), passing in the open file's handle as a single argument, like this:

```
fclose( $handle );
```

Although PHP should close all open files automatically when your script terminates, it's good practice to close files from within your script as soon as you're finished with them because it frees them up quickly for use by other processes and scripts — or even by other requests to the same script.

Reading and Writing Strings of Characters

The fread() function can be used to read a string of characters from a file. It takes two arguments: a file handle and the number of characters to read. The function reads the specified number of characters (or less if the end of the file is reached) and returns them as a string. For example:

```
$handle = fopen( "data.txt", "r" );
$data = fread( $handle, 10 );
```

This code reads the first ten characters from data.txt and assigns them to $data as a string.

You can use the fwrite() function to write data to a file. It requires two arguments: a file handle and a string to write to the file. The function writes the contents of the string to the file, returning the number of characters written (or false if there's an error). For example:

```
$handle = fopen("data.txt", "w");
fwrite($handle,"ABCxyz");
```

The first line opens the file data.txt for writing, which erases any existing data in the file. (If the file doesn't exist, PHP attempts to create it.) The second line writes the character string "ABCxyz" to the beginning of the file. As with fread(), the file pointer moves to the

position after the written string; if you repeat the second line, fwrite() appends the same six characters again, so that the file contains the characters "ABCxyzABCxyz".

You can limit the number of characters written by specifying an integer as a third argument. The function stops writing after that many characters (or when it reaches the end of the string, whichever occurs first). For example, the following code writes the first four characters of "abcdefghij" (that is, "abcd") to the file:

```
fwrite( $handle, "abcdefghij", 4 );
```

Testing for the End of a File

The feof() function serves a single, simple purpose: it returns true when the file pointer has reached the end of the file (or if an error occurs) and returns false otherwise. It takes just one argument — the file handle to test. This method is useful with fread() in a while loop when you don't know how long the file is:

```
// HELLO_WORLD.TXT CONTAINS THE CHARACTERS "HELLO, WORLD!"
$handle = fopen("hello_world.txt", "r" );
$text = "";
while (!feof( $handle)) {
        $text .= fread( $handle, 3);  // Read 3 chars at a time
}
echo $text; // Displays "Hello, world!"
fclose( $handle );
```

Reading and Writing Entire Files

Writing code to read a file line by line, or string by string, can be tedious. Fortunately, PHP provides you with some functions that can access the complete contents of a file in one go. These include:

- file() — For reading a whole file into an array, without needing to open it
- file_get_contents() and file_put_contents() — For reading and writing the contents of a file without needing to open it
- fpassthru() — For displaying the contents of an open file
- readfile() — For displaying the contents of a file without needing to open it

Because these functions read the entire file into memory in one go, they should be used for relatively small files (a few MB at most). If you're working with a 100MB text file, it's probably best to use fread() to read and process the file in chunks.

file

file() reads the contents of a file into an array, with each element containing a line from the file. It takes just one argument — a string containing the name of the file to read — and returns the array containing the lines of the file:

```
$lines = file("/home/chris/myfile.txt");
```

The newline character remains attached at the end of each line stored in the array. This function, like most of the others described in this section, doesn't require you to specify a file handle. All you need to do is pass in the filename of the file to read. The function automatically opens, reads, and, once it's done, closes the file.

You can optionally specify some useful flags as the second parameter to file():

Flag	Description
FILE_USE_INCLUDE_PATH	Look for the file in the include path
FILE_IGNORE_NEW_LINES	Remove newline characters from the end of each line in the array
FILE_SKIP_EMPTY_LINES	Ignore empty lines in the file

As with other flags in PHP you can combine any of these flags with the bitwise OR operator. For example, the following code looks for a file in the include path and, when found, reads the file, ignoring any empty lines in the file:

```
$lines = file( "myfile.txt", FILE_USE_INCLUDE_PATH | FILE_SKIP_EMPTY_LINES );
```

As with fopen(), you can also use file() to fetch files on a remote host:

```
$lines = file( "http://www.example.com/index.html" );
foreach ( $lines as $line ) echo $line;
```

file_get_contents

A related function is file_get_contents(). This does a similar job to file(), but it returns the file contents as a single string, rather than an array of lines. The end-of-line characters are included in the string:

```
$fileContents = file_get_contents( "myfile.txt" );
```

If there was a problem reading the file, file_get_contents() returns false. You can pass the FILE_USE_INCLUDE_PATH flag (described earlier) as the second argument to file_get_contents().

You can also optionally pass in an offset and/or a length parameter to determine where you want the file reading to start, and how many characters you want to read. For example, the following code reads 23 characters from myfile.txt, starting at character 17:

```
$fileContents = file_get_contents( "myfile.txt", null, null, 17, 23 );
```

The first null argument avoids setting the FILE_USE_INCLUDE_PATH flag, and the second null argument avoids setting a context. Contexts are out of the scope of this book, but you can find out more about them in the online manual at http://www.php.net/manual/en/stream.contexts.php. **file_put_contents()** is the complement to file_get_contents(). As you'd imagine, it takes a string and writes it to a file:

```
$numChars = file_put_contents( "myfile.txt", $myString );
```

The function returns the number of characters written, or false if there was a problem. You can affect the behavior of the function by passing various flags as the third argument. file_put_contents() supports the same flags as file_get_contents(), as well as two additional flags:

Flag	Description
FILE_APPEND	If the file already exists, append the string to the end of the file, rather than overwriting the file.
LOCK_EX	Lock the file before writing to it. This ensures that other processes can't write to the file at the same time.

You can also lock files that are opened using fopen(). To do this, use flock(). See http://www .php.net/manual/en/function.flock.php for more details.

readfile() and fpassthru()

readfile() take a file and output its unmodified contents straight to the Web browser. fpassthru() requires the handle of an open file to work with:

```
$numChars = fpassthru($handle); //READFILE() INSTEAD WORKS ON AN UNOPENED
FILE:
$numChars = readfile("myfile.txt");
```

As you can see, both functions return the number of characters read (or false if there was a problem). fpassthru() reads from the current file pointer position, so if you've already read some of the file only the remaining portion of the file will be sent.

Random Access to File Data

Using the functions, you've met so far, you can only manipulate data sequentially, that is, in the same order that it is arranged in the file. However, sometimes you need to skip around the contents of an open file. For example, you might want to read a file once to search for a particular string, then return to the start of the file to search for another string. Of course, this is easy if you've read the entire file using, for example, file_get_contents(). However, this isn't practical for large files.

Fortunately, it's possible to move the file pointer around within an open file, so that you can start reading or writing at any point in the file. PHP gives you three functions that let you work with the file pointer:

- fseek() — Repositions the file pointer to a specified point in the file
- rewind() — Moves the file pointer to the start of the file
- ftell() — Returns the current position of the file pointer

To use fseek(), pass the handle of the open file, and an integer offset. The file pointer moves to the specified number of characters from the start of the file (use zero to move the pointer to the first character). For example, the following code moves the pointer to the eighth character in the file (that is, seven characters after the first character) and displays the next five characters from that point:

```
// HELLO_WORLD.TXT CONTAINS THE CHARACTERS "HELLO, WORLD!"
$handle = fopen( "hello_world.txt", "r" );
fseek( $handle, 7 );
echo fread( $handle, 5 );  // Displays "world"
fclose( $handle );
```

To specify how the offset is calculated, you can add a third optional argument containing one of the following constants:

- SEEK_SET — Sets the pointer to the beginning of the file plus the specified offset (the default setting)
- SEEK_CUR — Sets the pointer to the current position plus the specified offset
- SEEK_END — Sets the pointer to the end of the file plus the specified offset (use with a negative offset)

fseek() returns 0 if the pointer was successfully positioned, or -1 if there was a problem. You can't use this function with files on remote hosts opened via HTTP or FTP (for example, fopen("http://www.example.com/")).

If you want to move the pointer back to the start of the file (a common occurrence), a handy shortcut is the rewind() function. The following two lines of code both do the same thing:

```
fseek( $handle, 0 );
rewind( $handle );
```

The ftell() function takes a file handle and returns the current offset (in characters) of the corresponding file pointer from the start of the file. For example:

```
$offset = ftell( $handle );
```

As you saw earlier, the fpassthru() function outputs file data from the current file position onward. If you have already read data from an open file but want to output the file's entire contents, call rewind() first.

Copying, Renaming and Deleting Files

PHP also lets you copy, rename, and delete files. The functions to perform these operations are copy(), rename(), and unlink(), respectively. The copy() function takes two string arguments: the first argument is the path to the file to copy, and the second argument is the path to copy it to. It returns true if the file was successfully copied, or false if there was a problem copying the file. The following example copies the source file copyme.txt to the destination file copied.txt in the same folder:

```
copy( "./copyme.txt", "./copied.txt" );
```

The rename() function is used to rename (or move) a file. It works in much the same way as copy() . For example, to rename a file within a folder you could use:

```
rename( "./address.dat", "./address.backup" );
```

To move a file to a different folder, you might use:

```
rename( "/home/joe/myfile.txt", "/home/joe/archives/myfile.txt" );
```

The unlink() function lets you delete files from the server. To use it, pass the filename of the file you want to delete. For example, if you wanted to get rid of the file trash.txt in the current directory, you could write:

```
unlink( "./trash.txt" );
```

copy() , rename() , and unlink() raise warning - level errors if the file or directory in question can ’ t be found. Make sure the file or directory exists first (for example, by using file_exists()) to avoid such errors.

Chapter 9: MySQL

MySQL Definitions

Relational Database

Is a collection of tables (entities) that have a specific relationship to them (we will look at relationships later). These relationships allow you to better construct your data to be more efficient.

Entity (Table)

An entity (also known as a table) is like a noun, it is a person, place thing, or event. The entities of a college database may be "students", "courses", "books", etc. As stated, the above entities have relationships between them.

Attribute (Field)

An attribute is a property of an entity. For example, the attributes of a "students" entity maybe "name","dob","address", etc.

Primary Key

The primary key is a unique identifier for an entity (table) row. In the table below each row has a unique customer number representing that customer. No two customers can be alike thus each primary key is unique.

Customer_num	Customer_name	Balance	Credit_limit	Rep_num
148	Al's Appliance and Sport	$6,550.00	$7,500.00	20
282	Brookings Direct	$331.50	$10,000.00	35
356	Ferguson's	$5,785.00	$7,500.00	65
408	The Everything Shop	$5,285.25	$5,000.00	35
462	Bargains Galore	$3,412.00	$10,000.00	65
524	Kline's	$12,762.00	$15,000.00	20
608	Johnson's Department Store	$2,106.00	$10,000.00	65
687	Lee's Sport and Appliance	$2,851.00	$5,000.00	35
725	Deerfield's Four Seasons	$248.00	$7,5000.00	35

Foreign Key

In the context of relational databases, a foreign key is a field (or collection of fields) in one table that uniquely identifies a row of another table or the same table. In simpler words, the foreign key is defined in a second table, but it refers to the primary key in the first table. For example, a table called Employee has a primary key called employee_id. Another table called Employee Details has a foreign key that references employee_id to uniquely identify the relationship between both the tables. The table containing the foreign key is called the child table, and the table containing the candidate key is called the referenced or parent table.

In the image below we have the first entity which is the customer table, and the second which is the balance table. The customer table has the primary key which is associated with the balance tables customer_num field, which is the foreign key.

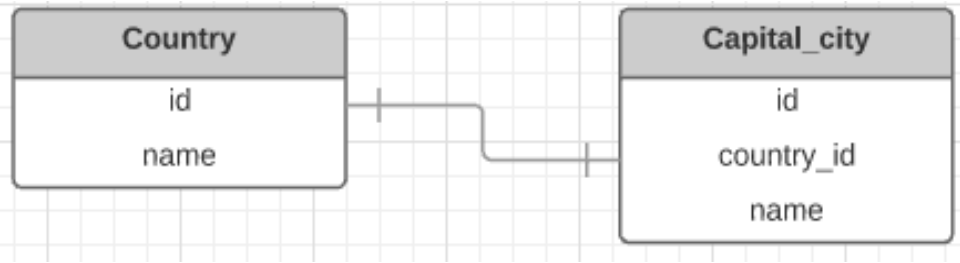
id	Customer_name
148	Al's Appliance and Sport
282	Brookings Direct
356	Ferguson's
408	The Everything Shop
462	Bargains Galore
524	Kline's
608	Johnson's Department Store
687	Lee's Sport and Appliance
725	Deerfield's Four Seasons

id	Customer_num	Balance	Credit_limit	Rep_num
1	148	$6,550.00	$7,500.00	20
2	282	$331.50	$10,000.00	35
3	356	$5,785.00	$7,500.00	65
4	408	$5,285.25	$5,000.00	35
5	462	$3,412.00	$10,000.00	65
6	524	$12,762.00	$15,000.00	20
7	608	$2,106.00	$10,000.00	65
8	687	$2,851.00	$5,000.00	35
9	725	$248.00	$7,5000.00	35

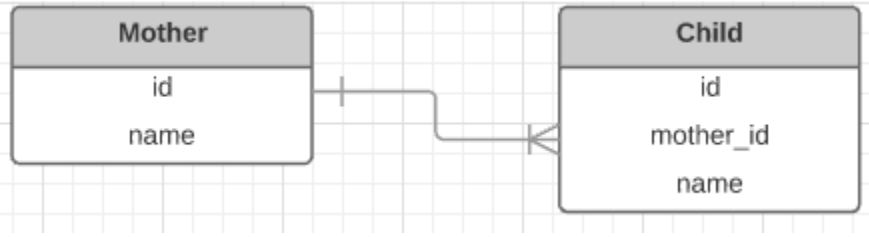
Relationship

The relationship is the association between entities. In a relational database, we have three different relationships

One to one - In a relational database, a one-to-one relationship exists when one row in a table may be linked with only one row in another table and vice versa. It is important to note that a one-to-one relationship is not a property of the data, but rather of the relationship itself. For instance, think of one table as countries, and table as capital cities. A country has only one capital city, and a capital city is the capital of only one country. The image shows an example of this.

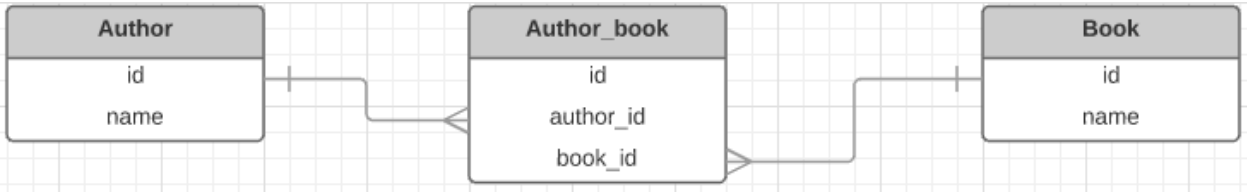


One to many - In a relational database, a one-to-many relationship exists when one row in table A may be linked with many rows in table B, but one row in table B is linked to only one row in table A. It is important to note that a one-to-many relationship is not a property of the data, but rather of the relationship itself. For instance, think of A as mothers, and B as children. A mother can have several children, but a child can have only one biological mother. The image below shows an example of this



Many to many - In a relational database management system, such relationships are usually implemented using a third "associative" table (also known as cross-reference table), say, AB with two one-to-many relationships A -> AB and B -> AB. In this case,

the logical primary key for AB is formed from the two foreign keys (i.e. copies of the primary keys of A and B). The image below demonstrates this.



In the above image, we see two one to many relationships combined into one entity. One author can have one or many books and one book can have one or many authors. The middle entity combines the books with the authors. The author id and book id combination will always be different and thus unique. There is no reason to have a primary key in the author_book table as the combination of author id and book id serves that unique difference. I included a primary key to show that you can have one but not needed in this case.

MySQL Normalization

Normal Forms

Armed with the knowledge of how to join tables and abstract complexity behind views, you should never be intimidated by the number of tables in a normalized database. The term normalization refers to the process of organizing data—by following specific rules—to eliminate redundancy and inconsistencies.

The main benefits of database normalization are:

- **Decreased storage requirements** — duplicate data means there is more data to be stored, which requires more disk space and memory. Minimizing such redundancy makes efficient use of the resources available to the database.
- **Data integrity** — if the same data appears in several places, there's an increased opportunity for inconsistencies to be introduced. For example, if a name appears in two different tables, it's possible to misspell one of the instances or perhaps update one instance and miss the other.
- **Maintainability** — the rules for normalization focus on maintaining relationships and put the R in RDBMS. The result is tables that have a clearly defined and logical purpose. When one table stores actor details and another stores movie data, it's easier to update the data than if it were all mixed in a single table.

A normalized database may be a well-designed database, but you should know that normalization can also hurt performance. It requires time and effort for MySQL to join tables. We can mitigate this by defining and using keys and indexes properly, and by filtering results so that MySQL is doing the hardest work on the smallest possible number of rows.

There are several levels of normalization. A table is said to be in a specific normal form when its tables adhere to that level's rules. I'll keep our discussion here focused on the First Normal Form, Second Normal Form, and Third Normal Form, abbreviated as 1NF, 2NF, and 3NF. Most of the other forms define things that are already a consequence of adhering to these first three forms.

First Normal

FormFirst Normal states that every field should hold only one value. Some RDBMSs support compound data types like arrays, but MySQL doesn't. This fortunately goes a long way in helping us avoid breaking this rule. What's left for us to worry about is to avoid using the SET data type or any cute hacks designed to cram more than one value into a field, like storing JSON-encoded objects or strings of comma-separated values.

Here's a table that stores some actors and the movies they starred in. Some rows contain more than one movie title in the film_title column, violating 1NF.

```
+------------+-----------+-----------------------------+
| first_name | last_name | film_title                  |
+------------+-----------+-----------------------------+
| SCARLETT   | BENING    | ROOF CHAMPION, YOUTH KICK   |
| MICHAEL    | BOLGER    | HOMEWARD CIDER, SANTA PARIS |
| VAL        | BOLGER    | PATIENT SISTER, YOUTH KICK  |
| DARYL      | CRAWFORD  | BROTHERHOOD BLANKET         |
+------------+-----------+-----------------------------+
```

To fix this situation, we need to split up the offending values and place them in their rows. The first_name and last_name values are duplicated so each row is complete.

```
+----------+------------+----------+---------+--------------------+
| actor_id | first_name | last_name | film_id | film_title         |
+----------+------------+----------+---------+--------------------+
|      124 | SCARLETT   | BENING    |     742 | ROOF CHAMPION      |
|      124 | SCARLETT   | BENING    |     997 | YOUTH KICK         |
|      185 | MICHAEL    | BOLGER    |     427 | HOMEWARD CIDER     |
|      185 | MICHAEL    | BOLGER    |     761 | SANTA PARIS        |
|       37 | VAL        | BOLGER    |     663 | PATIENT SISTER     |
|       37 | VAL        | BOLGER    |     997 | YOUTH KICK         |
|      129 | DARYL      | CRAWFORD  |     101 | BROTHERHOOD BLANKET|
+----------+------------+----------+---------+--------------------+
```

The table now meets 1NF because every field holds a single value. No field contains multiple values.

Second Normal Form

A table is in Second Normal Form when it meets the criteria for 1NF (so all 2NFtables are also 1NF) and every non-key field can be accessed using a logically related key. If the key is a compound key, each field that makes up the key must be related to all the columns in the row. Here I've added an actor_id column and film_id column to the example table which gives us a compound key we can use to target an appearance:

actor_id	first_name	last_name	film_id	film_title
124	SCARLETT	BENING	742	ROOF CHAMPION
124	SCARLETT	BENING	997	YOUTH KICK
185	MICHAEL	BOLGER	427	HOMEWARD CIDER
185	MICHAEL	BOLGER	761	SANTA PARIS
37	VAL	BOLGER	663	PATIENT SISTER
37	VAL	BOLGER	997	YOUTH KICK
129	DARYL	CRAWFORD	101	BROTHERHOOD BLANKET

Focusing on the film_title columns, the field in the row with actor_id value 124 and film_id value 997 holds a different "Youth Kick" string from the row with actor_id 37 and film_id 997. We can differentiate between them using the actor_id and film_id values together as a compound key, but the table isn't in 2NF because relying on anything that's not film-related to access a film title is a violation of the rules. That is, there's a logical connection between film_id and film_title because they both pertain to movies, but actor_id has no connection to film_title. 2NF'sfocus is on making sure our keys make logical sense.

The remedy is to move the film_title column to its table.

```
+----------+------------+-----------+---------+
| actor_id | first_name | last_name | film_id |
+----------+------------+-----------+---------+
|      124 | SCARLETT   | BENING    |     742 |
|      124 | SCARLETT   | BENING    |     997 |
|      185 | MICHAEL    | BOLGER    |     427 |
|      185 | MICHAEL    | BOLGER    |     761 |
|       37 | VAL        | BOLGER    |     663 |
|       37 | VAL        | BOLGER    |     997 |
|      129 | DARYL      | CRAWFORD  |     101 |
+----------+------------+-----------+---------+

+---------+---------------------+
| film_id | title               |
+---------+---------------------+
|     742 | ROOF CHAMPION       |
|     997 | YOUTH KICK          |
|     427 | HOMEWARD CIDER      |
|     761 | SANTA PARIS         |
|     663 | PATIENT SISTER      |
|     101 | BROTHERHOOD BLANKET |
+---------+---------------------+
```

Each movie title is now accessible by its key and none of them depends on non-film related information. Using the film_id column as a foreign key between the tables maintains the original connection the records had. Additionally, we've eliminated some redundancy by moving the movie titles to a new table. There's no wasted space from storing a movie title more than once.

The new table certainly follows 2NF, but we haven't entirely fixed the original table. We still have the same problem with actor names as we did with the film titles: not all the columns that make up our key are logically related to the column whose value we want to target. This will be resolved when we apply the rules of 3NF. Normalization is a

process and it's common for one form to serve as a stepping stone to the next, and for a form to satisfy more than just its requirements.

Third Normal Form

A table is in Third Normal Form when the criteria for 2NF are met and all non-key fields are accessible by the table's primary key. We know we still have a problem with actor names because we can only target specific first_name and last_name fields by referencing actor_id and film_id. An actor's name doesn't depend on the films they starred in. actor_id is related logically to the names, but the duplication of values from following 1NF prohibits us from using it that way. We need to migrate the name columns to a separate table just as we did with the movie titles.

```
+----------+---------+
| actor_id | film_id |
+----------+---------+
|      124 |     742 |
|      124 |     997 |
|      185 |     427 |
|      185 |     761 |
|       37 |     663 |
|       37 |     997 |
|      129 |     101 |
+----------+---------+

+----------+------------+-----------+
| actor_id | first_name | last_name |
+----------+------------+-----------+
|      124 | SCARLETT   | BENING    |
|      185 | MICHAEL    | BOLGER    |
|       37 | VAL        | BOLGER    |
|      129 | DARYL      | CRAWFORD  |
+----------+------------+-----------+

+---------+---------------------+
| film_id | title               |
+---------+---------------------+
|     742 | ROOF CHAMPION       |
|     997 | YOUTH KICK          |
|     427 | HOMEWARD CIDER      |
|     761 | SANTA PARIS         |
|     663 | PATIENT SISTER      |
|     101 | BROTHERHOOD BLANKET |
+---------+---------------------+
```

Moving the actor names to their own table permits us to use actor_id as the primary key for accessing them and again eliminates the duplication of data. What's left of our original table are the two ID columns that preserve the appearance connections. The result is not only 2NF but also meets the requirements for 3NF.

The process of normalizing the table resulted in something that looks very similar to how the sakila database's actor, film_actor, and film tables are organized. Every column in the actor table has some logical connection to actors, each column in the film table pertains to movies, and the film_actor table is a junction table of foreign keys that maintain the relationships. A statement can join all three tables to identify the actors and their appearances, but the specific pieces of data are better organized and isolated from one another. Keeping 3NF in mind when planning out your tables goes a long way towards a well-designed database.

MySQL DB and Table Commands

Source

One of the best books I have read on MySQL is "MySQL Crash Course" by Ben Forta. It can be found on Amazon at https://www.amazon.com/MySQL-Crash-Course-Ben-Forta/dp/0672327120/ref=sr_1_2?ie=UTF8&qid=1515674504&sr=8-2&keywords=mysql+crash+course. Many of the examples came from this book I highly recommend it.

Create a Database

The following code will create a database with the name "scott_test"

```
CREATE DATABASE scott_test;
```

Use a Database

You must select a database before you can apply queries to it. To select a database enter the "USE" command

```
USE scott_test;
```

Create a table

To create a table you use the "CREATE TABLE" command followed by the table name. After the table name, you (within parentheses) create the columns their data types, and other properties. Each column is separated by a comma. Below is an example of creating a customer’s table.

The following code creates four tables. One is a customer’s table that stores customer information. The other is an orders table that stores order information for each customer. You will see the cust_id of the customer's table is a primary key, yet in the orders table, it is a foreign key. That relationship allows each customer to have several orders.

The products table just contains the products. We need to find a way to add the products to the orders. Because one order can contain many different products and one

product can belong to several orders (many to many relationship), we have to have another table. In this case, we created one named prod_ords. Prod_ords contains an order_id and product_id, individually the values will not be unique but if they are combined, they will always be a unique number. That is why we can create a primary key of order_id and prod_id. The prod_ords table also contains a quantity for how many of each product the customer ordered.

You should take note of how dividing the information up into separate tables allows us to more easily manage the data. If I change a customer name is will be reflected for all the past and future orders automatically.

NOTE: The tables at the bottom of this lesson describe the datatypes, properties, and engine types.

```
CREATE TABLE customers (
  cust_id int NOT NULL AUTO_INCREMENT,
  cust_name char(50) NULL,
  cust_address char(50) NULL,
  cust_city char(50) NULL,
  cust_state char(50) NULL,
  cust_zip char(50) NULL,
  cust_country char(50) NOT NULL DEFAULT 'US',
  cust_email char(255) NULL,
  PRIMARY KEY (cust_id)
) ENGINE = InnoDB;

CREATE TABLE orders (
    order_id int NOT NULL AUTO_INCREMENT,
    cust_id int NOT NULL,
    order_number int NOT NULL,
    order_data date NOT NULL,
    PRIMARY KEY (order_id),
    FOREIGN KEY (cust_id) REFERENCES customers(cust_id)
) ENGINE = InnoDB;

CREATE TABLE products (
    product_id int NOT NULL AUTO_INCREMENT,
    product_name text NOT NULL,
    product_description text NOT NULL,
    product_price double(5,2) NOT NULL,
    PRIMARY KEY (product_id),
```

```
    FOREIGN KEY (order_id) REFERENCES orders(order_id)
) ENGINE = InnoDB;

CREATE TABLE prod_ords (
    order_id int NOT NULL,
    product_id int NOT NULL,
    quantity int NOT NULL DEFAULT 1,
    PRIMARY KEY (order_id, product_id,)
) ENGINE = InnoDB;
```

The auto-increment property will increment the cust_id number one digit for each record. This is done to keep each customer id unique. The cust_id is a primary key thus it must be unique.

Delete a table

```
DROP TABLE table_name;
```

Truncate a table

Truncating a table removes all the table data but does not delete the table.

```
TRUNCATE TABLE table_name;
```

Show a table structure

```
DESC tablename;
```

Or

```
Explain tablename;
```

Delete Database

```
DROP DATABASE database name;
```

Table Datatypes

Datatypes are used for several reasons:

- Datatypes enable you to restrict the type of data that can be stored in a column.

- Datatypes allow for more efficient storage, internally. Numbers and date-time values can be stored in a more condensed format than text strings
- Datatypes allow for alternative sorting orders. If everything is treated as strings, 1 comes before 10, which comes before 2. Numeric datatypes would be sorted correctly.

Datatype	Description
STRINGS	
CHAR	Fixed-length string from 1 to 255 chars longs. Its size must be specified at create time, or MySQL assumes CHAR(1)
ENUM	Accepts one of a predefined set of up to 64k strings
LONGTEXT	Same as TEXT, but with a maximum size of 4GB
MEDIUMTEXT	Same as TEXT, but with a maximum size of 16K
SET	Accepts zero or more predefined set of up to 64 strings
TEXT	Variable-length text with a maximum size of 64k
TINYTEXT	Same as TEXT, but with a maximum size of 255 bytes
VARCHAR	Same as CHAR, but stores just the text. The size is a maximum, not the minimum
NUMERIC	

Datatype	Description
BIT	A bit-field, from 1 to 64bits wide.
BIGINT	An integer value that supports numbers from -9223372036854775808 to 9223372036854775807
BOOLEAN OR BOOL	Boolean flags, either 0 or 1, used primarily for on/off flags
DECIMAL(M,D) OR DEC(M,D)	An unpacked floating-point number that cannot be unsigned. In the unpacked decimals, each decimal corresponds to one byte. Defining the display length (M) and the number of decimals (D) is required. NUMERIC is a synonym for DECIMAL.
DOUBLE(M, D)	A double-precision floating-point number that cannot be unsigned. You can define the display length (M) and the number of decimals (D). This is not required and will default to 16,4, where 4 is the number of decimals. Decimal precision can go to 53 places for a DOUBLE. REAL is a synonym for DOUBLE.
FLOAT (M, D)	A floating-point number that cannot be unsigned. You can define the display length (M) and the number of decimals (D). This is not required and will default to 10,2, where 2 is the number of decimals and 10 is the total number of digits (including decimals). Decimal precision can go to 24 places for a FLOAT.
INT OR INTEGER	Integer value supports numbers from -2147483648 to 2147483647

Datatype	Description
REAL	4-byte floating-point number
SMALLINT	Integer values support numbers from -32768 to 32767
TINYINT	Integer values support numbers from -128 to 127
DATE AND TIME	
DATE	Date from 1000-01-01 to 999-12-31 in the format YYYY-MM-DD
DATETIME	Combination of DATE and TIME
TIMESTAMP	Functionally equivalent to DATETIME (but with a smaller range).
TIME	Time in the format HH:MM:SS
YEAR	A 2 or 4 digit year, 2 digit years support a range of 70(1970) to 69 (2069), 4 digit years supports a range of 1901 to 2155
BINARY Binary datatypes are used to store all sorts of data such as graphics and multimedia. Though you can store this information in a database it is best to store binary information on the server and the link to it in a database.	
BLOB	Blob with a maximum length of 64k
MEDIUMBLOB	Blob with a maximum length of 16MB
LONGBLOB	Blob with a maximum length of 4GB

Datatype	Description
TINYBLOB	Blob with a maximum length of 255 bytes

Column Properties

Below are some property types and definitions

Property	Definition
AUTO_INCREMENT	Auto-increment allows a unique number to be generated automatically when a new record is inserted into a table. Often this is the primary key field that we would like to be created automatically every time a new record is inserted. By default, the starting value for AUTO_INCREMENT is 1, and it will increment by 1 for each new record. To let the AUTO_INCREMENT sequence start with another value, use the following SQL statement: `ALTER TABLE Persons AUTO_INCREMENT=100;` AUTO_INCREMENT must be a primary key.
DEFAULT	The default value for a field if no value is entered. If the default is a string put it in quotes, if a number no quotes.
PRIMARY KEY	The PRIMARY KEY constraint uniquely identifies each record in a database table. Primary keys must contain UNIQUE values and cannot contain NULL values. A table can have only one or more primary keys. If a single column has a

Property	Definition
	primary key, it must be unique. If multiple columns are used the combination of them must be unique.
NULL	Allows a field to have a null value
NOT NULL	A field must have a value

Engine Types

An engine is used to manage and manipulate data. Unlike other DBMS MySQL does not come with a single-engine but some with several. The reason is so you can pick the right engine for the job. If you do not select an engine one will be given by default, usually MyISAM. Below is a table of engine types:

Engines	Description
InnoDB	This is the default storage engine for MySQL 5.5 and higher. It provides transaction-safe (ACID compliant) tables, supports FOREIGN KEY referential-integrity constraints. It supports commit, rollback, and crash recovery capabilities to protect data. It also supports row-level locking. It's "consistent nonlocking reads" increases performance when used in a multiuser environment. It stores data in clustered indexes which reduces I/O for queries based on primary keys.
MyISAM	This storage engine manages non-transactional tables, provides high-speed storage and retrieval, supports full-text searching.

Engines	Description
MEMORY	Provides in-memory tables, formerly known as HEAP. It stores all data in RAM for faster access than storing data on disks. Useful for quick looks up of reference and other identifying data.
MERGE	Groups more than one similar MyISAM tables to be treated as a single table, can handle non-transactional tables, included by default.
EXAMPLE	You can create tables with this engine, but can not store or fetch data. The purpose of this is to teach developers how to write a new storage engine.
ARCHIVE	Used to store a large amount of data, does not support indexes.
CSV	Stores data in Comma Separated Value format in a text file.
BLACKHOLE	Accepts data to store but always returns empty.
FEDERATED	Stores data in a remote database.

MySQL Select Commands

Setting Up the Test Data

For this lesson and others that follow, we will be using some test data. The test data can be found on my GitHub account at https://GitHub.com/sshaper/cps276_examples/tree/master/database_data. Open the create.sql file, highlight the file contents, copy and paste it into a file on your computer and name it create.sql. Do the same for populate.sql When you are done upload both files to your server. I would recommend putting them into a folder named SQL. Go into the directory where the files are located and do the following.

1. load the create.sql file into mysql mysql -u root -p < create.sql enter your password on next line
2. load the populate.sql file into mysql mysql -u root -p sampledata < populate.sql enter your password on next line
3. Test to make sure it is there mysql -u root -p enter your password on next line
4. At the MySQL prompt enter use sampledata; then enter show tables; you should see a list of tables.
5. Enter SELECT * FROM products you should see a list of products. If so, then you have all the sample data will be using for this lesson.

Commands

The following commands can be run from the MySQL shell command line. Even though I use uppercase for the commands they are not case sensitive.

Retrieving Single Columns

This will retrieve the column (prod_id) from the products table

```
SELECT prod_id FROM products;
```

Retrieving Multiple Columns

This will retrieve the columns (prod_id, prod_name, prod_price) from the products table

```
SELECT prod_id, prod_name, prod_price FROM products;
```

Retrieving All Columns

This will retrieve all the columns from the products table using the wildcard (*). Many times programmers will use the wildcard instead of just entering the fields they need. Depending on what they are doing this can be wasteful of server resources.

```
SELECT * FROM products;
```

Retrieving Distinct rows

The select returns all matched rows but what if you did not want every occurrence of every value? For example, suppose you wanted the vendor id of all vendors in your products table. You could enter.

```
SELECT vend_id FROM products;
```

As you can see you get back 14 records when there are only four vendors. If just want the distinct values you would use the "DISTINCT" keyword that only returns unique vend_id rows. The DISTINCT keyword must be placed after the SELECT keyword

```
SELECT DISTINCT vend_id FROM products;
```

Limiting Results

You may not want all the rows but just a few. You can use the "LIMIT" keyword to limit your results. The limit will just return the first (n) number of rows you specify.

NOTE: Rows start with 0, not 1 so LIMIT 1,1 will return just the second row.

```
SELECT prod_name FROM products LIMIT 5;
```

If you want to limit "n" number of rows starting from a certain row, then enter

```
SELECT prod_name FROM products LIMIT 5, 5;
```

Using Fully Qualified Table Names

In some cases, you may have to use the table name with the field. For example, if you have two tables that share the same column name. The example below shows how to do this.

```
SELECT products.prod_name FROM products;
```

You could technically be specific about the database as well, but I have never had to do this.

```
SELECT products.prod_name FROM sampledata.products;
```

Sorting Data

When selecting data you can specify an order. In the example below I will get the product name data in alphabetical order using the ORDER BY keyword.

```
SELECT prod_name FROM products ORDER BY prod_name;
```

Sorting by Multiple Columns

In some cases, you may want to sort data by multiple columns. In the next example, I will do that first by price then by name

```
SELECT prod_id, prod_price, prod_name FROM products ORDER BY prod_price, prod_name;
```

It is important to understand that when sorting by multiple columns, the sort sequence is exactly as specified. So, the above example will sort by product name only when the product price is the same.

Sort Direction

You can sort by direction as well using the ASC ascending(default) and DESC descending commands.

```
SELECT prod_id, prod_price, prod_name FROM products ORDER BY prod_price DESC;
```

If you want to do a multi-column sort you would enter the ASC or DESC keyword after each column name

```
SELECT prod_id, prod_price, prod_name FROM products ORDER BY prod_price DESC,
prod_name;
```

MySQL Joins

Joins

A join is a mechanism used to associate tables within a SELECT statement (thus the name join). Using a special syntax, multiple tables can be joined so a single set of output is returned, and join associates the correct rows in each table on-the-fly.

Cartesian Product

The results were returned by a table relationship without a join condition. The number of rows retrieved is the number of rows in the first table multiplied by the number of rows in the second table. Below is an example of this.

```
SELECT vend_name, prod_name, prod_price FROM vendors, products ORDER BY
vend_name, prod_name;
```

As you can see in the preceding output, the Cartesian product is seldom what you want. The data returned here has matched every product with every vendor, including products with the incorrect vendor (and vendors with no products at all).

Simple Join

Creating a join is simple. You must specify all the tables to be included and how they are related to each other. Look at the following example

```
SELECT vend_name, prod_name, prod_price FROM vendors, products WHERE
vendors.vend_id = products.vend_id ORDER BY vend_name, prod_name;
```

Look at the FROM clause it has two tables in it instead of one. These are the names of the two tables that are being joined (you can join more than just two). You'll notice that columns are specified as vendors.vend_id and products.vend_id. This fully qualified column name is required here because if you just specified vend_id, MySQL cannot tell which vend_id columns you are referring to.

Inner Joins

The first example was called an equijoin - a join based on the testing of equality between two tables. The kind of join is also called an inner join. You can use a slightly different syntax for these if you want to.

```
SELECT vend_name, prod_name, prod_price FROM vendors INNER JOIN products ON
vendors.vend_id = products.vend_id;
```

Joining Multiple Tables

As stated above you can join multiple tables.

```
SELECT prod_name, vend_name, prod_price, quantity FROM orderitems, products, vendors
WHERE products.vend_id = vendors.vend_id AND orderitems.prod_id = products.prod_id AND
order_num = 20005;
```

Another example

```
SELECT cust_name, cust_contact FROM customers, orders, orderitems WHERE
customers.cust_id = orders.cust_id AND orderitems.order_num = orders.order_num AND
prod_id = 'TNT2';
```

Using Table and Column Aliases

Sometimes it is easier to use a table and/or a column alias when writing a query. All an alias is, is a shorter name.

```
SELECT cust_name AS cs, cust_contact AS cc FROM customers AS c, orders AS o,
orderitems AS oi WHERE c.cust_id = o.cust_id AND oi.order_num = o.order_num AND prod_id
= 'TNT2';
```

The table aliases are nice for getting the column names for each table. The column aliases just display the alias for the column name when the resulting table is created.

Self Joins

One advantage of using table aliases is to be able to refer to the same table more than once in a single select statement

For example, suppose that a problem was found with a product (item id DTNTR) and you, therefore, wanted to know all of the products made by the same vendor to determine if the problem applied to them, too. This query requires that you first find out which vendor creates item DTNTR, and next find which other products are made by the same vendor. The following is a way to do this.

```
SELECT p1.prod_id, p1.prod_name FROM products AS p1, products AS p2 WHERE
p1.vend_id = p2.vend_id AND p2.prod_id = 'DTNTR';
```

In the above query, because of what we need to do, we have to reference the same table twice; the use of table aliases does this.

Outer Join

Outer joins will include table rows that have no associated rows in the related table. Reasons for an outer join might be:

- Count how many orders each customer placed, including customers who have yet to place an order
- List all products with order quantities, including products not ordered by anyone.
- Calculate average sales sizes, taking into account customers who have not yet placed an order.

This first example uses the `LEFT OUTER JOIN` to join all the rows from the table on the left in the FROM clause

```
SELECT customers.cust_name, customers.cust_id, orders.order_num FROM customers LEFT OUTER JOIN orders ON customers.cust_id = orders.cust_id;
```

When you look at the output of the above example you will see that all the customers are listed, regardless of whether they have ordered or not.

The second example does the other, lists all orders and customers for those orders. Because each order must have a customer you don't have any NULL values.

```
SELECT customers.cust_id, orders.order_num FROM customers RIGHT OUTER JOIN orders ON customers.cust_id = orders.cust_id;
```

MySQL Insert Update and Delete

Inserting New Records

You can insert one record or multiple records into a table.

Single insertion

```
INSERT INTO table_name ( field1, field2,...fieldN ) VALUES ( value1, value2,...valueN );
```

Multiple Insertions

```
INSERT INTO table_name ( field1, field2,...fieldN ) VALUES ( value1, value2,...valueN ), ( value1, value2,...valueN )
```

Update a table

If you do not supply a where condition (which indicates the rows you want to update) then all rows will be updated.

```
UPDATE table_name SET column1 = value1, column2 = value2, ... WHERE condition;
```

Delete Rows

If you don't supply a where condition you can delete everything from a table (better to use Truncate)

```
DELETE FROM table_name WHERE condition
```

MySQL Importing and Exporting Data

Introduction

Being able to import and export your database is an important skill to have. You can use data dumps for backup and restoration purposes, so you can recover older copies of your database in case of an emergency, or you can use them to migrate data to a new server or development environment.

Working with database dumps in MySQL is straightforward. This lesson will cover how to export the database as well as import it from a dump file in MySQL.

Prerequisites

To import and/or export a MySQL, you will need:

- Access to the Linux server running MySQL
- The database name and user credentials for it

Exporting the Database

The mysqldump console utility is used to export databases to SQL text files. These files can easily be transferred and moved around. You will need the database name itself as well as the username and password to an account with privileges allowing at least full read-only access to the database.

Export your database using the following command. mysqldump -u root -p database_name > data-dump.sql. When you click the enter key you will be asked for your password.

- username is the username you can log in to the database with.
- database_name is the name of the database that will be exported.
- data-dump.sql is the file in the current directory that the output will be saved too.

Importing the Database

To import an existing dump file into MySQL, you will have to create a new database. This is where the contents of the dump file will be imported.

First, log in to the database as root or another user with sufficient privileges to create new databases mysql -u root -p

This will bring you into the MySQL shell prompt. Next, create a new database for the database that is on your .sql file. Do not create a new database if one with the name already exists. CREATE DATABASE database_name;

You'll see this output confirming it was created.

Output
Query OK, 1 row affected (0.00 sec)
Now exit the MySQL shell by pressing CTRL+D. On the normal command line, you can import the dump file with the following command: mysql -u root -p database_name < data-dump.sql

- root is the username you can log in to the database with
- database_name is the name of the freshly created database
- data-dump.sql is the data dump file to be imported, located in the current directory

The successfully-run command will produce no output. If any errors occur during the process, mysql will print them to the terminal instead. You can check that the database was imported by logging in to the MySQL shell again and inspecting the data. This can be done by selecting the new database with USE database_name and then using SHOW TABLES; or a similar command to look at some of the data.

Chapter 10: PDO

Introduction

PDO is a Database Access Abstraction Layer. The abstraction, however, is two-fold: one is widely known but less significant, while another is obscure but of most importance.

PDO offers a unified interface to access many different databases. Although this feature is magnificent by itself, it doesn't make a big deal for the application, where only one database backend is used anyway. And, despite some rumors, it is impossible to switch database backends by changing a single line in PDO config - due to different SQL flavors (to do so, one needs to use an averaged query language like DQL). Thus, for the average LAMP developer, this point is rather insignificant, and to them, PDO is just a more complicated version of the familiar mysql(i)_query() function, which I disagree with as PDO it is much, much more.

PDO abstracts not only a database API but also basic operations that otherwise must be repeated hundreds of times in every application, making your code extremely inefficiently written. Unlike MySQL and MySQLi, both of which are low-level bare APIs not intended to be used directly (but only as a building material for some higher-level abstraction layer), PDO is such an abstraction already. Still incomplete though, but at least usable.

The real PDO benefits are:

- **security** (*usable* prepared statements)
- **usability** (many helper functions to automate routine operations)
- **reusability** (unified API to access a multitude of databases, from SQLite to Oracle)

Note that although PDO is the best out of native DB drivers, for a modern web-application consider using an ORM with a Query Builder, or any other higher-level

abstraction library, with only occasional fallback to vanilla PDO. Good ORMs are Doctrine, Eloquent, RedBean, and Yii::AR. Aura.SQL is a good example of a PDO wrapper with many additional features. Either way, it's good to know the basic tools first.

Database Support

The extension can support any database that a PDO driver has been written for. At the time of this writing, the following database drivers are available:

PDO_DBLIB (FreeTDS / Microsoft SQL Server / Sybase)

PDO_FIREBIRD (Firebird/Interbase 6)

PDO_IBM (IBM DB2)

PDO_INFORMIX (IBM Informix Dynamic Server)

PDO_MYSQL (MySQL 3.x/4.x/5.x)

PDO_OCI (Oracle Call Interface)

PDO_ODBC (ODBC v3 (IBM DB2, unixODBC and win32 ODBC))

PDO_PGSQL (PostgreSQL)

PDO_SQLITE (SQLite 3 and SQLite 2)

PDO_4D (4D)

Connecting

Different databases may have slightly different connection methods. Below, the method to connect to some of the most popular databases are shown. You'll notice that the first three are identical, other than the database type - and then SQLite has its own syntax.

```
try {
  # MS SQL Server and Sybase with PDO_DBLIB
  $DBH = new PDO("mssql:host=$host;dbname=$dbname, $user, $pass");
  $DBH = new PDO("sybase:host=$host;dbname=$dbname, $user, $pass");

  # MySQL with PDO_MYSQL
  $DBH = new PDO("mysql:host=$host;dbname=$dbname", $user, $pass);

  # SQLite Database
  $DBH = new PDO("sqlite:my/database/path/database.db");
}
catch(PDOException $e) {
    echo $e->getMessage();
}
```

Please take note of the try/catch block - you should always wrap your PDO operations in a try/catch and use the exception mechanism. Typically, you're only going to make a single connection.

You can close any connection by setting the handle to null.

```
//CLOSE THE CONNECTION
$DBH = null;
```

You can get more information on database-specific options and/or connection strings for other databases from PHP.net.

Exceptions and PDO

PDO can use exceptions to handle errors, which means anything you do with PDO should be wrapped in a try/catch block. You can force PDO into one of three error modes by setting the error mode attribute on your newly created database handle. Here's the syntax:

```
$DBH->setAttribute( PDO::ATTR_ERRMODE, PDO::ERRMODE_SILENT );
$DBH->setAttribute( PDO::ATTR_ERRMODE, PDO::ERRMODE_WARNING );
$DBH->setAttribute( PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION );
```

No matter what error mode you set, an error connecting will always produce an exception, and creating a connection should always be contained in a try/catch block.

PDO::ERRMODE_SILENT

This is the default error mode. If you leave it in this mode, you'll have to check for errors in the way you're probably used to if you used the MySQL or MySQLi extensions. The other two methods are more ideal for DRY programming.

PDO::ERRMODE_WARNING

This mode will issue a standard PHP warning and allow the program to continue execution. It's useful for debugging.

PDO::ERRMODE_EXCEPTION

This is the mode you should want in most situations. It fires an exception, allowing you to handle errors gracefully and hide data that might help someone exploit your system. Here's an example of taking advantage of exceptions:

```
# connect to the database
try {
  $DBH = new PDO("mysql:host=$host;dbname=$dbname", $user, $pass);
  $DBH->setAttribute( PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION );

  # UH-OH! Typed DELECT instead of SELECT!
  $DBH->prepare('DELECT name FROM people');
}
catch(PDOException $e) {
    echo "I'm sorry, I'm afraid I can't do that.";
    file_put_contents('PDOErrors.txt', $e->getMessage(), FILE_APPEND);
}
```

There's an intentional error in the select statement; this will cause an exception. The exception sends the details of the error to a log file and displays a friendly (or not so friendly) message to the user.

Insert and Update

Inserting new data, or updating existing data is one of the more common database operations. Using PDO, this is normally a two-step process. Everything covered in this section applies equally to both UPDATE and INSERT operations.

Here's an example of the most basic type of insert:

```
//STH MEANS "STATEMENT HANDLE"
$STH = $DBH->prepare("INSERT INTO folks ( first_name ) values ( 'Cathy' )");
$STH->execute();
```

You could also accomplish the same operation by using the exec() method, with one less call. In most situations, you're going to use the longer method so you can take advantage of prepared statements. Even if you're only going to use it once, using prepared statements will help protect you from SQL injection attacks. An SQL injection attack is where a user will attempt to enter a SQL command into your form in the hopes that it will be executed when the form data goes to the database. This can result in disastrous consequences.

Prepared Statements

Using prepared statements will help protect you from SQL injection.

A prepared statement is a precompiled SQL statement that can be executed multiple times by sending just the data to the server. It has the added advantage of automatically making the data used in the placeholders safe from SQL injection attacks.

You use a prepared statement by including placeholders in your SQL. Here are three examples: one without placeholders, one with unnamed placeholders, and one with named placeholders.

```
//NO PLACEHOLDERS - RIPE FOR SQL INJECTION!
$STH = $DBH->("INSERT INTO folks (name, addr, city) values ($name, $addr,
$city)");
```

```
//UNNAMED PLACEHOLDERS
$STH = $DBH->("INSERT INTO folks (name, addr, city) values (?, ?, ?);


//NAMED PLACEHOLDERS
$STH = $DBH->("INSERT INTO folks (name, addr, city) value (:name, :addr,
:city)");
```

You want to avoid the first method; it's here for comparison. The choice of using named or unnamed placeholders will affect how you set data for those statements.

When you are adding data to a placeholder that is known as "binding" or "data binding"

Unnamed Placeholders

```
// ASSIGN VARIABLES TO EACH PLACE HOLDER, INDEXED 1-3
$STH->bindParam(1, $name);
$STH->bindParam(2, $addr);
$STH->bindParam(3, $city);


// INSERT ONE ROW
$name = "Daniel"
$addr = "1 Wicked Way";
$city = "Arlington Heights";
$STH->execute();


//INSERT ANOTHER ROW WITH DIFFERENT VALUES
$name = "Steve"
$addr = "5 Circle Drive";
$city = "Schaumburg";
$STH->execute();
```

There are two steps here. First, we assign variables to the various placeholders (lines 2-4). Then, we assign values to those placeholders and execute the statement. To send another set of data, just change the values of those variables and execute the statement again.

Does this seem a bit unwieldy for statements with a lot of parameters? It is. However, if your data is stored in an array, there's an easy shortcut:

```
//THE DATA WE WANT TO INSERT
$data = array('Cathy', '9 Dark and Twisty Road', 'Cardiff');

$STH = $DBH->("INSERT INTO folks (name, addr, city) values (?, ?, ?);
$STH->execute($data);
```

The data in the array applies to the placeholders in order. $data[0] goes into the first placeholder, $data[1] the second, etc. However, if your array indexes are not in order, this won't work properly, and you'll need to re-index the array.

Named Placeholders

You could probably guess the syntax, but here's an example:

```
/* THE FIRST ARGUMENT IS THE NAMED PLACEHOLDER NAME - NOTICE NAMED

PLACEHOLDERS ALWAYS START WITH A COLON.*/
$STH->bindParam(':name', $name);
```

You can use a shortcut here as well, but it works with associative arrays. Here's an example:

```
//THE DATA WE WANT TO INSERT
$data = array( 'name' => 'Cathy', 'addr' => '9 Dark and Twisty', 'city' =>
'Cardiff' );


//THE SHORTCUT!
$STH = $DBH->("INSERT INTO folks (name, addr, city) value (:name, :addr,
:city)");
$STH->execute($data);
```

The keys of your array do not need to start with a colon, but otherwise need to match the named placeholders. If you have an array of arrays you can iterate over them, and simply call the execute with each array of data.

Another nice feature of named placeholders is the ability to insert objects directly into your database, assuming the properties match the named fields. Here's an example object, and how you'd perform your insert:

```
//A SIMPLE OBJECT
class person {
    public $name;
    public $addr;
    public $city;

    function __construct($n,$a,$c) {
        $this->name = $n;
        $this->addr = $a;
        $this->city = $c;
    }
    # etc ...
}

$cathy = new person('Cathy','9 Dark and Twisty','Cardiff');

$STH = $DBH->("INSERT INTO folks (name, addr, city) value (:name, :addr,
:city)");
$STH->execute((array)$cathy);
```

By casting the object to an array in the execute, the properties are treated as array keys.

Selecting Data

Data is obtained via the ->fetch(), a method of your statement handle. Before calling fetch, it's best to tell PDO how you'd like the data to be fetched. You have the following options:

PDO::FETCH_ASSOC: returns an array indexed by column name

PDO::FETCH_BOTH (default): returns an array indexed by both column name and number

PDO::FETCH_BOUND: Assigns the values of your columns to the variables set with the ->bindColumn() method

PDO::FETCH_CLASS: Assigns the values of your columns to properties of the named class. It will create the properties if matching properties do not exist

PDO::FETCH_INTO: Updates an existing instance of the named class

PDO::FETCH_LAZY: Combines PDO::FETCH_BOTH/PDO::FETCH_OBJ, creating the object variable names as they are used

PDO::FETCH_NUM: returns an array indexed by column number

PDO::FETCH_OBJ: returns an anonymous object with property names that correspond to the column names

Below is an example of the FETCH_ASSOC which is commonly used:

```
$STH->setFetchMode(PDO::FETCH_ASSOC);
```

This fetch type creates an associative array, indexed by column name. This should be quite familiar to anyone who has used the MySQL/MySQLi extensions. Here's an example of selecting data with this method:

```
$STH = $DBH->query('SELECT name, addr, city from folks');


//SETTING THE FETCH MODE
$STH->setFetchMode(PDO::FETCH_ASSOC);

while($row = $STH->fetch()) {
    echo $row['name'] . "\n";
    echo $row['addr'] . "\n";
    echo $row['city'] . "\n";
```

```
}
```

The while loop will continue to go through the result set one row at a time until complete.

PHP Database and PDO Classes

Introduction

Connecting to a database and using PDO is straight forward, however, it can be tedious to keep writing the same statements over and over. Here, I will show you the classes I wrote for connecting to the database and for executing SQL using PDO.

The sql file for this database is on my GitHub site at:

https://github.com/sshaper/cps276_examples/blob/master/database_data/names_with_shortnames.sql

The database is called names.

A working example can be found at

http://198.199.80.235/cps276/cps276_examples/php-pdo/index.php

The source code for the working example can be found at the address below and will be briefly explained in the next few pages.

https://GitHub.com/sshaper/cps276_examples/tree/master/php-pdo

To understand what I did it is best to visit my GitHub page and look at the class I wrote. I will explain it here, but the code is shown on my GitHub page.

IMPORTANT NOTE: To fully understand how this works you need to look at the code, run the example, and read the rest of this chapter. You will not understand it if you don't.

Database Connection Class

Found at https://GitHub.com/sshaper/cps276_examples/blob/master/php-pdo/classes/Db_conn.php

This class sets up the database connection and sets up some additional attributes as well. All you need to do is enter the database name $dbName and your database password $dbPass the other values can be left at their default settings.

What is nice about this is any updates to my database connection I only have to update this one file. In a past job, I worked at the database was changed about four times during the time I was there. If I wouldn't have written a class to handle all my database connection requests, it would have been a lot of work on me every time IT changed the database.

The PDO class

This is the class that I wrote to make doing queries in PDO easier. The code for this class can be viewed at https://GitHub.com/sshaper/cps276_examples/blob/master/php-pdo/classes/Pdo_methods.php.

This class must be placed in the same folder as the database connection class described above. What is nice about this class is it makes calling the PDO methods easier because all the code needed is written here, you just have to enter the correct information when using the class methods (I will go over the class methods in the next

The following pages will discuss the various methods found in the Pdo_methods.php class. To do this, I have created another class named "Crud.php" that provides an example of using the Pdo_methods class. The crud (short for create, read, update and delete) class it can be found at.
https://GitHub.com/sshaper/cps276_examples/blob/master/php-pdo/classes/Crud.php

Using selectNotBinded($sql) Method

This one will run a query on a SQL statement that does not need to be binded, this would be a statement that is hardcoded into the page and does not require user input. This class will return an error string if there is a problem otherwise it will return a recordset of the records returned.

To see how the selectNotBinded method is used you want to look at the getNames($type) method in the Crud.php file. When looking at the code you can see that I do the following

1. Create an instance of the PdoMethods class

2. Write the SQL statement
3. Call the selectNotBinded method and pass the SQL statement
4. If there was an error I return an error message. Notice I do records = 'error' the string of error is returned from the PDO class if there was an error processing the script.
5. If there was not an error I check to see if there are any records to display, if so I display them (I call the createList or createInput methods which loop through the recordset and create an unordered list or input form). Or if no records exist then I display a message indicating such.

Using selectBinded($sql, $bindings)

This is the same as the above, but it is to be used when a part of the SQL statement will be a value provided by the user. In that case, we will need to bind the parameters of the SQL statement to avoid SQL injections.

To see how the selectBinded method is used you want to look at the addName() method in the Crud.php file. When looking at the code you can see that I do the following

1. Create an instance of the PdoMethods class
2. Write the SQL statement
3. Create the bindings array
4. Call the selectNotBinded method and pass the SQL statement and bindings array
5. If there was an error, I display an error message. Notice I do records = 'error' the string of error is returned from the PDO class if there was an error processing the script.
6. If there was not an error I check to see if there are any records to display, if so I display them (I call the createList or createInput methods which loop through the recordset and create an unordered list or input form). Or if no records exist then I display a message indicating such.

otherBinded($sql, $bindings)

This function will do the insert, update and delete queries, for queries that use user input, so the values need to be binded. Since nothing is returned we just return a message stating if there was an error or not. If you look at the updateNames($post) in the Crud.php file. Below is how we will use the otherBinded method.

1. Create an instance of the PdoMethods class
2. Write the SQL statement
3. Create the bindings array
4. Call the otherBinded method and pass the SQL statement and bindings array.
5. If there was an error, I display an error message. Notice I do records = 'error' the string of error is returned from the PDO class if there was an error processing the script.
6. If there was not an error, then I display a success message.

otherNotBinded($sql)

This function will do the insert, update and delete queries, however, this is used in circumstances where the user does not enter values used in this query, thus it does not have to be binded. This works the same as otherBinded but you don't need to attach the bindings. Since nothing is returned, we just return a message stating if there was an error or not. There is no example in the Crud.php file for this.

Chapter 11: AJAX

Data Sources

Introduction

When communicating from one server or database to another you need to have a common data source so both machines can understand the data. Two common data sources are XML and JSON. In this lesson, we will look at both.

XML

XML (E**x**tensible **M**arkup **L**anguage) serves two fundamental purposes / roles:

Describing data in the most meaningful manner possible. You determine the tags and attributes to use in marking up your data; there are no pre-defined data tags. This is the *extensible* aspect of XML. **Creating other markup languages.** You can create your markup language using the rules of XML; it is a meta-language. Many languages have already been created, such as Scalable Vector Graphics (SVG), Wireless Markup Language (WML), and Math Markup Language (MathML).

XML is free and non-proprietary, using plain text files, which allows it to easily cross boundaries between different platforms and computer systems (referred to as *portability*).

XML files are saved with a .xml file extension. XML goes far beyond the Web; it is used extensively in a wide range of programs from computer games to desktop applications.

XML tags data (the Content layer) and can communicate **data structure** (based on the tag nesting).

Character encoding for XML data requires replacing the following in your actual data (the characters between the open and close tags):

- lesser-than signs with <

- greater-than signs with >
- ampersands (&) with &
- single quotes with '
- double quotes with "
- XML comments:
- Comments are the same as (X)HTML: <!-- -->
- Comments are ignored during parsing.

Setting up an XML Document

The first line in an XML document (referred to as the *XML declaration*) identifies the version of XML: <?xml version="1.0"?>

The next set of tags is the *root element*, which at the broadest level identifies the data being described. There is only one root element per XML document. All other data tags must be within the root element.

```
<?xml version="1.0"?>
<books>
  <book>
    <title>Eric Meyer on CSS</title>
    <author>Eric Meyer</author>
    <publisher>New Riders</publisher>
    <editions>
      <first_edition>2002</first_edition>
    </editions>
  </book>
  <book>
    <title>Web Design in a Nutshell</title>
    <author>Jennifer Niederst</author>
    <publisher>O&apos;Reilly &amp; Associates</publisher>
    <editions>
      <first_edition>1999</first_edition>
      <second_edition>2001</second_edition>
      <third_edition>2006</third_edition>
    </editions>
  </book>
</books>
```

Rules for Naming XML Elements

- Names of elements cannot contain spaces.
- Underscores are recommended to fill in the spaces.
- Hyphens can be interpreted as minuses (by some programming languages) so they should be avoided.
- Periods can have other meanings in programming languages that will read the XML data file so they should also be avoided.
- No other punctuation characters (such as colons or semicolons) are allowed.
- Names of elements can contain letters, numbers, and other characters.
- Names of elements must not start with a number or punctuation character. They can begin with an underscore.
- Names of elements must not start with XML (whether uppercase, lowercase, or mixed case).
- There is no length limit but try to keep these as short as possible while remaining descriptive.

Attributes in XML

Attributes can be created freely, using either single or double quotes to surround their values. There are no established rules for when data must be in a set of child tags or an attribute, but there are some considerations:

- **Attributes can only contain a single value.** Elements accommodate multiple values, often by having the same element be repeated multiple times and/or via child elements.
- **Attributes do not scale as readily as elements if data becomes more complex.** This is because attributes are tied to a given element.
- **Attributes do not describe data structure.** This is because attributes do not convey data hierarchy (elements convey this hierarchy through their nesting).
- **Attributes are more difficult to manipulate by scripts and validate.** This is due to the attribute being buried within an element.

The following are equivalent:

```
<?xml version="1.0"?>
```

```
<book>
  <title author="Eric Meyer">Eric Meyer on CSS</title>
  <publisher year="2002" edition="1st">New Riders</publisher>
</book>
<?xml version="1.0"?>
<book>
  <title>Eric Meyer on CSS</title>
  <author>Eric Meyer</author>
  <publisher>New Riders</publisher>
  <year>2002</year>
  <edition>1st</edition>
</book>
```

JSON

JSON stands for "JavaScript Object Notation" it is a JavaScript object, or it may be an array of JavaScript objects. You can write JSON as a text file and then create an object out of it later. The following is an example of a JSON object

```
[
	{
		"title": "Eric Meyer on CSS",
		"author": "Eric Meyer",
		"publisher": "New Riders",
		"editions" : [
			{"year":"2002","edition":"First Edition"}
		]
	},
	{
		"title": "Web Design in a Nutshell",
		"author": "Jennifer Niederst",
		"publisher": "O'Reilly and Associates",
		"editions" : [
			{"year":"1999", "edition":"First Edition"},
			{"year":"2001", "edition":"Second Edition"},
			{"year":"2006", "edition":"Third Edition"}
		]
	},
	{
		"title": "JavaScript: The Good Parts",
		"author": "Douglas Crockford",
		"publisher": "O'Reilly and Associates",
		"editions" : [
```

```
                {"year":"2008","edition":"First Edition"}
            ]
        }
]
```


Examples

The PHP examples can be found

at https://github.com/sshaper/cps276_examples/tree/master/datasources

AJAX

Introduction

AJAX stands for Asynchronous JavaScript and XML. AJAX relies upon JavaScript, the DOM, and the XMLHttpRequest object (frequently abbreviated as XHR).

AJAX enables you to read data from and write data to the server asynchronously without having to reload the webpage.

This asynchronous setup breaks from the traditional Web approach of synchronous communication, in which the user takes an action and waits for the server response and that response results in the page being reloaded.

One example is a chat application written in AJAX. There could be AJAX calls every couple of seconds that take in whatever you are typing and updates your chat window with what the other people have typed. Rather than typing and then clicking 'Send', AJAX makes the updates automatically. How frequently you update, or poll for new data, is a very important consideration and will depend on the application as well as the available bandwidth.

The chat application example also showcases another important aspect of AJAX: updates are made to just one part of the screen rather than forcing an entire page to reload.

The degree to which AJAX is implemented falls into a range from completely AJAX-driven to using AJAX for some nice enhancements.

The first thing we need to do is set up an XMLHttpRequest. Browsers define their HTTP API on an XMLHttpRequest class. Each instance of this class represents a single request/response pair, and the properties and methods of the object allow you to specify request details and extract response data.

Creating the XMLHttpRequest Object

XMLHttpRequest has been supported by web browsers for many years. To create an XMLHttpRequestObject you write:

```
var xhr;
 if (window.XMLHttpRequest) {
   xhr = new XMLHttpRequest();
 } else {

     // CODE FOR OLDER BROWSERS
   xhr = new ActiveXObject("Microsoft.XMLHTTP");
 }
```

Now that we have a request object we have some properties of that object, as shown in the following table:

Properties of the XMLHttpRequest Object

Property	Description and Considerations
onreadystatechange	This property contains the event handler to be called when the readystatechange event is fired, that is every time the readyState property of the XMLHttpRequest changes.
readyState	This property reflects the state in the request cycle, which ranges from 0-4; • 0 - uninitialized (XMLHttpRequest object has been created but open() has not been called yet.) • 1 - loading (XMLHttpRequest object has been created but send() has not been called yet.) • 2 - loaded (The send() method has been called but the headers and status are not available.) • 3 - interactive (Some of the data has been received from the server.)

Property	Description and Considerations
	• 4 - complete (All the data from the server has been received.)
responseText	This property contains the data from the server and is used for all non-XML formats (the 'Content-Type' would be 'text/html' or 'text/plain'). Data is a String object.
responseXML	This property contains the data from the server and is used for XML data (the 'Content-Type' is generally 'text/xml', although 'application/xml' is also seen). Data is returned as an XML DOM, so use getAttribute() and setAttribute() for attribute read/write.
status	This property stores the HTTP status code as a number. We want either 200 (OK) or 304 (Not Modified). Other numbers and meaning: • 200 - Ok • 201 - Created • 204 - No Content • 205 - Reset Content • 206 - Partial Content • 400 - Bad Request • 401 - Unauthorized • 403 - Forbidden • 404 - Not Found

Property	Description and Considerations
	• 405 - Method not allowed • 406 - Not Acceptable • 407 - Proxy Authentication Required • 408 - Request Timeout • 411 - Length required • 413 - Requested Entity too Large • 414 - Requested URL too long • 415 - Unsupported Media Type • 500 - Internal Server Error • 501 - Not Implemented • 502 - Bad Gateway • 503 - Service Unavailable • 504 - Gateway Timeout • 505 - HTTP Version Not Supported
statusText	This property stores the text description for the HTTP status code, such as "Not Modified", rather than a number.

Methods of the XMLHttpRequest Object

The XMLHttpRequest Object also has the following methods.

Method	Description and Considerations
open()	This method initializes the XMLHttpRequest. It requires three parameters and there are optional fourth and fifth parameters.

Method	Description and Considerations
	The first parameter is the request method, which is either 'GET', 'POST', or 'HEAD': • 'GET' sends and receives data via the URL. • 'POST' sends and receives data via the HTTP request's body. • 'HEAD' just sends and receives the HTTP headers. Typically 'GET' is used for data retrieval from the server and 'POST' is used to send data to the server since 'POST' avoids the character limits of URL-passed data. The second parameter is the file on the server, such as myData.xml. This could also be a server-side script that is passed parameters, such as myData.php?x=1&p=342 The address given can either be absolute or relative; absolute paths fall under the Same Source Policy of JavaScript. The third parameter is a boolean that indicates whether the request is synchronous (false) or asynchronous (true). We always want asynchronous (if we opted for synchronous then the script execution is stopped until the data comes back from the server and we do not want to wait), so stick with true

Method	Description and Considerations
	The optional fourth and fifth parameters are for username and password, respectively. For proper security, these would need to be encrypted values or sent over a secure connection.
send()	After using open() and setting up an (on)readystatechange event handler, send() is used to send the data request. This method sends the request along with a single parameter for the body content of the request: • For 'POST' this is/are the variable(s) holding the information to send. • For 'GET' you must pass null as the parameter value. Once the request arrives back from the server the (on)readystatechange event handler fires and triggers the desired function.
abort()	This method stops the request currently in progress and sets the readyState property to 0 For most browsers this is never needed; giving alternate instructions to an XMLHttpRequest object causes the old instructions to be automatically terminated.

Method	Description and Considerations
	To avoid issues in Internet Explorer we need to assign an empty anonymous function before abort() being called: xhr.onreadystatechange = function() {}; xhr.abort(); Notice that we are also back to DOM0 event handling; the (on)readystatechange event does not generate an event object (except in Safari and Chrome).
getAllResponseHeaders()	This method returns all the HTTP response headers in a string, with each header on a new line.
getResponseHeader()	This method accepts a single parameter, which is the name of an HTTP header. The value for that header is returned. Common requests are (Content-Length, Content-Type, Date, Last-Modified, Server).
setRequestHeader()	This method is used to set HTTP headers. It requires two parameters: 1. The HTTP header. A String object. 2. The value the header is being assigned; also a String object. Note: If you use 'POST' to send data to the server, you must set 'Content-Type' to 'application/x-www-form-urlencoded' using this method.

Writing an AJAX class

You can combine these various ajax command and write a class that will be easier. Below is one that I have written.

```
var Ajax = {}

Ajax.sendRequest = function(url, callback, postData, file) {


  /*SET FILE TO FALSE IF IT IS NOT ALREADY SET.  IF IT IS SET THEN

    IT IS SUPPOSED TO BE TRUE.  IF IT IS SET TO TRUE THAT INDICATES
THAT A FILE IS BEING SENT.*/

 if (file === undefined){
     file = false;
  }


    /*CREATES THE XML OBJECT*/


  var req = Ajax.createXMLHttpObject();


  /*IF RETURNS FALSE CANCEL OPERATION */


 if (!req){return};


  /*CHECK TO SEE IF POSTDATA WAS PASSED IF SO SET METHOD TO POST*/


 var method = (postData) ? "POST" : "GET";


  /*CALL THE OPEN METHOD, SEND THE METHOD "POST" OR "GET" AND PASS
TRUE*/

req.open(method,url,true);
```

```
  /*IF POSTDATA IS SENT AND THE FILE IS ZERO MEANING WE ARE NOT
SENDING A FILE THEN SET REQUEST HEADER FOR FORMS, OTHERWISE JAVASCRIPT
WILL DECIDE THE HEADER TYPE ON THE REQ.SEND METHOD.*/

if (postData &&!file){
   req.setRequestHeader('Content-type','application/x-www-form-urlencoded');
   }


  /*IF EVERYTHING RETURNS OK SEND REQ VALUE TO "CALLBACK"*/

req.onreadystatechange = function () {
   if (req.readyState !== 4) return;
   if (req.status !== 200 && req.status !== 304) {
     return;
   }
   callback(req);
 }


  /*IF WE HAVE ALREADY COMPLETED THE REQUEST, STOP THE FUNCTION SO AS
NOT TO SEND IT AGAIN*/

if (req.readyState === 4){return;}

 /*IF POSTDATA WAS INCLUDED SEND IT TO SERVER SIDE PAGE. INFORMATION
 CAN BE RECEIVED BY USING $_POST['DATA'] (THIS IS VIA PHP)*/


    /*SENDING A FILE AND SOME TEXT*/


  if (postData && file){
  req.send(postData);
 }


    /*SENDING TEXT ONLY*/


  else if (postData && !file){
     req.send("data="+postData);
  }

 else{
  req.send(null);
```

```
  }

}


/*DEPENDING ON THE BROWSER RETURN APPROPRIATE REQUEST*/


Ajax.XMLHttpFactories = [
  function () {return new XMLHttpRequest()},
  function () {return new ActiveXObject("Msxml2.XMLHTTP")},
  function () {return new ActiveXObject("Msxml3.XMLHTTP")},
  function () {return new ActiveXObject("Microsoft.XMLHTTP")}
];


/*THIS METHOD CYCLES THROUGH ALL REQUESTS IN XMLHTTPFACTORIES UNTIL

ONE IS FOUND*/


Ajax.createXMLHttpObject = function() {
  var xmlhttp = false;
  for (var i=0;i<Ajax.XMLHttpFactories.length;i++) {
    try {
      xmlhttp = Ajax.XMLHttpFactories[i]();
    }
    catch (e) {
      continue;
    }
    break;
  }
  return xmlhttp;
}
```

Passing Objects

When sending data to and from the server using AJAX, it is very common to send data as an object. To do so you want to create an object first. The example shown below shows an object named data that contains three properties. One property contains an array just to show that it can be done.

```
let data = {}

data.someproperty1 = "somepropertyValue1";

data.somepropety2 = "somepropertyValue2";

data.somepropertyArr1 = [1,2,3,4];
```

Next, we need to stringify the object so we can send it to the server. To do this we use JSON.stringify()

```
data = JSON.stringify(data);
```

After we stringify the object we can send it via a post request. In this example, I am using my AJAX class as written in my Util object. The only difference from the above code is the object name is Util instead of Ajax. Notice the object named "data" is being included in the Util.sendRequest() method.

```
Util.sendRequest("url to file we are sending the data to", function(res){
  //code to run after callback is here
},data);
```

Receiving/Sending the Object PHP

When we receive the object in PHP we get it using the $_POST[] superglobal array. For example, it may look something like the following:

```
//JSON_DECODE TAKES THE STRING AND PUTS IT INTO A PHP OBJECT

$data = json_decode($_POST['data']);
```

To access a property of the data object you would write something like:

```
$data->someproperty1
```

In PHP we use the arrow -> instead of dot syntax like JavaScript.

To send the data back to the client from the server just do the same thing as sending it to the server stringify it. Take note of how I created the object in PHP. The properties and values are separated by a =>.

```
$response = (object)[
   'masterstatus' => 'error',
   'msg' => 'Could not insert data into database'
 ];


   /*JSON_ENCODE TAKES AN OBJECT AND CONVERTS TO A STRING.  YOU NEED TO USE
"ECHO" TO SEND THE STRINGIFIED OBJECT BACK TO THE CLIENT COMPUTER.*/

 echo json_encode($response);
```

Receiving the Object on the Client-side

To receive the object on the client-side you must use JavaScript and it will be in the AJAX callback, as shown:

```
Util.sendRequest("url to file we are sending the data to", function(res){
  let response = JSON.parse(res.responseText);

   //RESPONSE BECOMES THE NEW JSON OBJECT


}, data);
```

AJAX Examples (PHP)

It is best to see examples of how AJAX works. You can view the code for the following examples at:

https://GitHub.com/sshaper/cps276_examples/tree/master/ajax.

The working examples can be viewed here:

http://198.199.80.235/cps276/cps276_examples/ajax/

The following is a breakdown of how the examples work.

NOTE: Please look at the PHP directory in the examples to see how the

get_example_php.html – When the button is clicked a request is sent to the server. The server sends back a string of text that is displayed on the webpage.

get_json_php.html – When the button is clicked a request is sent to the server for a file written in JSON (files is books.txt located in the data folder). The server returns the file and we use JavaScript to parse the file and display the results.

get_xml_php.html – When the button is clicked a request is sent to the server for a file written in XML (files is books.xml located in the data folder). The server returns the file and we use JavaScript to parse the file and display the results.

get_json_php_obj.html – When the button is clicked a request is sent to the server requesting a list of books to be returned. The PHP script that is called gets the books.txt file, creates a JSON object, parses the object into a list of books. It then creates a PHP object that contains a success message and a string of the book list. That object is stringified using json_encode and sent back to the browser to be put back into a JSON object and parsed, thus displaying the list.

passingObjectsUtilObject.html – When the form is filled out and the button clicked. JavaScript will get all the form field data, put it into an object, serialize the object, and send it to the server. The server creates an object from what was sent, adds some text

to it, serializes it, and returns it to the browser. The browser creates an object, parses it, and displays the result on the browser.

post_example_php.html – When the form is filled out and the button clicked. The form information is sent to the server using a POST request and then the server sends it back to the browser. This demonstrates sending a POST request via AJAX.

Chapter 12: Sessions & Date Methods

PHP Sessions

Introduction

$_SESSION is a super global array used to store information across the page requests a user makes during his visit to your website or web application. The most fundamental way to explain what a session is like is to imagine the following scenario:

You are working with an application. You open it, make some changes, and when you close it. That is a session in its simplest form.

The example scenario is reminiscent of the process that happens when using a login system. The process can be extremely complicated or incredibly simple, if there is a value that persists between requests. Information stored in the session can be called upon at any time during the open session.

While there may be many users accessing the site at the same time, each with their session, it's thanks to unique IDs assigned and managed by PHP for each session that allows each user's session to be available only to themselves. Session information is stored on the server rather than the user's computer (as cookie data is stored), which makes sessions more secure than traditional cookies for passing information between page requests.

In this lesson, we will explore the basics of using sessions in PHP – how to create them, how to destroy them, and how to make sure they remain secure.

Examples

The example files for this lesson can be found at https://GitHub.com/sshaper/cps276_examples/tree/master/php_sessions

The working example can be found at
http://198.199.80.235/cps276/cps276_examples/php_sessions/login_example/

Using Sessions

Before you can store information in a session, you must start PHP's session handling. This is done at the beginning of your PHP code and must be done before any text, HTML, or JavaScript is sent to the browser. To start the session, you call the session_start() function in your first file:

```
// START THE SESSION
session_start();

// STORE SESSION DATA
$_SESSION["username"] = "Scott";
```

session_start() starts the session between the user and the server and allows values to be stored in the $_SESSION superglobal array, which can be accessible in other scripts later on. Notice how I set the "username" to "Scott"

In your second file, you call session_start() again which this time continues the session, and you can then retrieve values from $_SESSION.

```
// CONTINUE THE SESSION
session_start();


// RETRIEVE SESSION DATA
$username = "$_SESSION["username"];
Echo $username
```

This example is a very basic demonstration of storing and retrieving data in a session. In the first script, the value "Scott" was associated with the key "username" in the $_SESSION array. In the second script, the information was requested back from the $_SESSION array using the key "username". $_SESSION allows you to store and retrieve information across the page requests of a user's active browsing session.

Ending a Session

As important as it is to begin a session, it is equally important to end one. Even though a session is only a temporary way to store data, it is very important to clean up after yourself to ensure maximum security when dealing with potentially sensitive information. It is also good practice and will avoid having a huge amount of stale session data sitting on the server.

To delete a single session key and value, you use the unset() function:

```
session_start();

// DELETE THE USERNAME KEY AND VALUE
unset($_SESSION["username"]);
```

To unset all of the session's values, you can use the session_unset() function:

```
//MUST CALL SESSION_START() FIRST
session_start();

// DELETE ALL SESSION VALUES
session_unset();
```

Both examples only affect data stored in the session, not the session itself. You can still store other values to $_SESSION after calling them if you so choose. If you wish to completely stop using the session, for example, a user logs out, you use the session_destroy() function.

```
//MUST CALL SESSION_START() FIRST
session_start();


// TERMINATE THE SESSION
session_destroy();
```

I highly recommended that when you are sure you no longer need the session that you destroy it using session_destroy(), rather than just unsetting all of its values with session_unset(). If you just unset all the values, the session itself is still active and malicious code could give those sessions harmful values.

Client-Side Cookies

When a session is created in PHP a cookie is put on the user's computer with a unique id number that ties directly to the session on the server. Once this cookie is destroyed the session is technically no longer accessible on the server, the session on the server will still be there. The following code will also delete the cookie. It does so by setting the time back one hour.

```
setcookie("PHPSESSID", "", time() - 3600, "/");
```

NOTE: If you want to learn more about setting cookies you can refer to the PHP manual

Securing Passwords

Hashed login

Storing passwords in a database it is vitally important to hash the password so if the database is hacked the passwords will not be shown. For this example, I use the following php methods to hash and read the password.

Function	Description
password_hash	password_hash() creates a new password hash using a strong one-way hashing algorithm. password_hash() is compatible with crypt(). Therefore, password hashes created by crypt() can be used with password_hash(). The following algorithms are currently supported: **PASSWORD_DEFAULT** - Uses the bcrypt algorithm (default as of PHP 5.5.0). Note that this constant is designed to change over time as new and stronger algorithms are added to PHP. For that reason, the length of the result from using this identifier can change over time. Therefore, it is recommended to store the result in a database column that can expand beyond 60 characters (255 characters would be a good choice). **PASSWORD_BCRYPT** - Use the CRYPT_BLOWFISH algorithm to create the hash. This will produce a standard crypt() compatible hash using the "$2y$" identifier. The result will always be a 60 character string or FALSE on failure.

Function	Description
password_verify	password_verify — Verifies that a password matches a hash. Returns a boolean.

The example I created allows you to login to a site using a username and password. The password is hashed in the database, but the user will type the password normally. Once you have successfully logged in you will see a listing of all usernames and hashed passwords for each administrator. You can also add new administrators. This application uses a table named "admin" which has an id (primary key), the username (string), and password (string) fields.

The example can be found at

https://GitHub.com/sshaper/cps276_examples/tree/master/php-hashed-login

The code is at

http://198.199.80.235/cps276/cps276_examples/php-hashed-login/

PHP Date and Time Formats

Working with Dates and Times

Web applications frequently need to deal with dates and times. For example, an application might need to track page access times, handle a user-entered date (such as a date-of-birth) from a Web form, or format a date field returned from a MySQL database in a human - friendly way so that it can be displayed in a Web page.

Though dates and times may, on the surface, seem like simple concepts to work with, in fact, they can be quite tricky for computers to handle. Issues such as leap years, time zones, and the fact that the number of days in a month is variable can cause all sorts of problems when it comes to storing, retrieving, comparing, adding, and subtracting dates.

To this end, PHP gives you a few date and time related functions to make your life easier.

Understanding Timestamps

Most computers store dates and times as UNIX timestamps, or simply timestamps. A timestamp is an integer that holds the number of seconds between midnight UTC on January 1, 1970, and the date and time to be stored (also in UTC). For example, the date and time" February 14, 2007, 16:48:12" in the GMT zone is represented by the UNIX timestamp 1171471692, because February 14, 2007, 16:48:12 is exactly 1,171,471,692 seconds after midnight on January 1, 1970. UTC stands for Universal Time Coordinated. For most purposes, you can consider it to be equivalent to Greenwich Mean Time (GMT).

You're probably wondering why computers store dates and times in such a strange format. Timestamps are a very useful way of representing dates and times. First of all, because a timestamp is simply an integer, it's easy for a computer to store it. Secondly, it's easy to manipulate dates and times when they're just integers. For example, to add one day to a timestamp, you just add the number of seconds in a day (which happens to be 86,400 seconds) to the value. It doesn't matter if the timestamp represents a date at

the end of a month or a year; you can still add one day just by adding 86,400 seconds to the timestamp value.

The majority of PHP date and time functions work with timestamps — if not explicitly, then certainly internally.

Getting the Current Date and Time

Computers — including Web servers, as well as your PC — keep track of the current date and time using a built-in clock. You can access this clock's value (on the server) with the PHP time() function, which simply returns the current date and time as a timestamp:

```
echo time(); // DISPLAYS E.G."1229509316"
```

Although not particularly useful in its own right, you can use time() in combination with other PHP functions to display the current time and compare dates and times against the current date and time, among other things.

If, when using any of PHP's date-related functions, you get an error warning message telling you that it is not safe to rely on the system's time zone settings, you need to configure PHP's time zone. To do this you use the date_default_timezone_set();

Here is an example of setting the timezone to Detroit time:

```
date_default_timezone_set('America/Detroit')
```

A list of PHP supported time zones

Creating Your Own Timestamps

Although time() is useful for getting the current time, often you want to work with other dates and times. You can use various PHP functions to create timestamps for storing dates and times. The three that you're likely to use most often are mktime() , gmmktime() , and strtotime().

Creating Timestamps from Date and Time Values

The mktime() function returns a timestamp based on up to six-time/date arguments, as follows:

- Hour (0 – 23)
- Minute (0 – 59)
- Second (0 – 59)
- Month (1 – 12)
- Day of the month (1 – 31)
- Year (1901 – 2038)

For example, the following code displays the timestamp corresponding to 2:32:12 pm on January 6, 1972:

```
echo mktime( 14, 32, 12, 1, 6, 1972 );
```

You can leave out as many arguments as you like, and the value corresponding to the current time will be used instead. For example, if the current date is December 22, 2008, the following code displays the timestamp representing 10 am on December 22, 2008:

```
echo mktime( 10, 0, 0 );
```

If you omit all the arguments, mktime() returns the current date and time, just like time().

Incidentally, you can pass in arguments that are outside the allowed ranges, and mktime() adjusts the values accordingly. So passing in a value of 3 for the month and 32 for the day causes mktime() to return a timestamp representing April 1.

Creating Timestamps from GMT Date and Time Values

mktime() assumes that the arguments you pass are in your computer's time zone — it converts the supplied time to UTC so that it can be returned as a timestamp. However, sometimes it's useful to be able to store a date and time that's already in the GMT zone. For example, many HTTP headers and other TCP/IP protocols work with dates and times that are always in GMT.

To create a timestamp from a GMT date and time, use gmmktime(). This works in the same way as mktime(), except that it expects its arguments to be in GMT. For example, let's say the computer running your PHP script is in the Indianapolis time zone, which is 5 hours behind GMT, and that it is running the following code:

```
$localTime = mktime( 14, 32, 12, 1, 6, 1972 );

$gmTime = gmmktime( 14, 32, 12, 1, 6, 1972 );
```

After this code has run, $localTime holds the timestamp representing Jan 6, 1972, at 7:32:12 pm GMT/ UTC (which is 2:32 pm on the same day Indianapolis time). Meanwhile, $gmtime holds the timestamp for 2:32:12 pm GMT/UTC; in other words, no time zone conversion has taken place.

Creating Timestamps from Date and Time Strings

mktime() is great if you already have the individual numeric values for the date and time you want to store. However, often your PHP script will receive a date or time as a string. For example, if your script works with emails, it may need to handle message dates, which are normally represented in the following format:

```
Date: Mon, 22 Dec 2008 02:30:17 +0000
```

Web server logs tend to use a format such as the following:

```
15/Dec/2008:20:33:30 +1100
```

Alternatively, your script might receive a user - input date along the lines of:

```
15th September 2006 3:12pm
```

Although you can use PHP's powerful string manipulation functions and regular expressions to split such strings into their parts for feeding to mktime(), PHP provides a useful function called strtotime() to do the hard work for you. strtotime() expects a string representing a date, and attempts to convert the string into a timestamp:

```
$timestamp = strtotime("15th September 2006 3:12pm");
```

You can pass in dates and times in practically any human-readable format you like. Here are some examples of valid date/time strings that you can pass to strtotime():

Date/Time String	Meaning
6/18/99 3:12:28pm	3:12:28 pm on June 18 th, 1999
15th Feb 04 9:30am	9:30 am on February 15 th, 2004
February 15th 2004, 9:30 am	9:30 am on February 15th, 2004
tomorrow 1:30pm	The day after the current date at 1:30 pm
Today	Midnight on the current date
Yesterday	Midnight on the day before the current date
last Thursday	Midnight on Thursday before the current date
+2 days	The day after tomorrow at the current time of day
-1 year	One year ago at the current time of day
+3 weeks 4 days 2 hours	3 weeks, 4 days, and 2 hours from now
3 days	3 days after the current date at the current time
4 days ago	4 days before the current date at the current time
3 hours 15 minutes	The current time plus 3 hours 15 minutes

As with mktime(), strtotime() assumes by default that the string you pass represents a date and time in the computer's time zone, and converts to UTC accordingly. However, you can specify a time in a different time zone by adding an offset from UTC, using a plus or minus sign followed by a four-digit number at the end of the string. The first two digits represent the hours component of the offset, and the second two digits represent the minutes.

For example:

```
$t = strtotime("February 15th 2004, 9:30am +0000"); // GMT
$t = strtotime("February 15th 2004, 9:30am +0100"); // 1 HOUR AHEAD OF GMT
$t = strtotime("February 15th 2004, 9:30am -0500"); // INDIANAPOLIS TIME
$t = strtotime("February 15th 2004, 9:30am +1000"); // SYDNEY TIME (NOT DST)
$t = strtotime("February 15th 2004, 9:30am +1100"); // SYDNEY TIME (WITH DST)
```

strtotime() calculates relative dates (such as" tomorrow 1:30pm") based on the current date. If you want to calculate a relative date based on a different date, pass that date as a second argument to strtotime() , in timestamp format:

```
$localTime = strtotime("tomorrow 1:30pm", 0 ); // JANUARY 2ND 1970, 1:30:00 PM
```

Extracting Date and Time Values from a Timestamp

Now you know how to create timestamps from time/date values and strings. You can also go the other way and convert a timestamp to its corresponding date and time components.

getdate()

getdate() accepts a **timestamp** and returns an associative array of date/time values corresponding to the supplied timestamp. The array contains the following keys:

Array Key	Description	Possible Values
seconds	The seconds component	0 to 59
minutes	The minutes component	0 to 59
hours	The hours component, in 24 - hour format	0 to 23
mday	The day of the month	1 to 31
wday	The day of the week as a number	0 (Sunday) to 6 (Saturday)
mday	The day of the month	1 to 31
mon	The month component as a number	1 to 12
year	The year component as a four-digit number	Typically 1970 to 2038

Array Key	Description	Possible Values
yday	The day of the year	0 to 365
weekday	The day of the week as a string	Sunday to Saturday
month	The month of the component as a string	January to December
0 (zero)	The Timestamp	Typically – 2147483648 to 2147483647

You can also call getdate() without a timestamp to return the components of the current date and time. Here are a couple of getdate() examples:

```
// DISPLAYS"JOHN LENNON WAS BORN ON 9 OCTOBER, 1940"
$johnLennonsBirthday = strtotime("October 9, 1940");
$d = getdate( $johnLennonsBirthday );
echo "John Lennon was born on" . $d["mday"] ."" . $d["month"] ."," .$d["year"];


//DISPLAYS THE CURRENT TIME IN HOURS AND MINUTES
$t = getDate();
echo "The current time is" . $t["hours"] .":". $t["minutes"];
```

If you just want to extract a single date or time component from a timestamp, you can use idate(). This function accepts two parameters: a format string and an optional timestamp. (If you omit the timestamp, the current date and time are used.) The single - character format string dictates the component to return, and the format in which to return it, as follows:

Format String	Description
B	Swatch Internet Time — a time - zone - free, decimal time measure. See http://en.wikipedia.org/wiki/Swatch_Internet_Time for details
d	Day of the month
h	Hours (in 12 - hour format)
H	Hours (in 24 - hour format)
i	Minutes
I	1 if DST (Daylight Saving Time) is in effect, 0 otherwise
L	1 if the date is in a leap year, 0 otherwise
M	Month number (1 - 12)
s	Seconds
t	The number of days in the month (28, 29, 30, or 31)
U	The timestamp
w	The day of the week, as a number (0 is Sunday)
W	The week number within the year (counting from 1)
y	The year as a two-digit number

Format String	Description
Y	The year as a four-digit number
z	The day number within the year (0 is January 1)
Z	The offset of the computer's time zone from UTC (in seconds)

As you can see, you can use idate() to retrieve all sorts of useful information from a date. Here's an example:

```
$d = strtotime("February 18, 2000 7:49am");

// DISPLAYS"THE YEAR 2000 IS A LEAP YEAR."
echo "The year" . idate("Y", $d );
echo " is" . ( idate("L", $d ) ?"":"not") ." a leap year\n";

// DISPLAYS"THE MONTH IN QUESTION HAS 29 DAYS."
echo " The month in question has" . idate("t", $d ) ." days\n";
```

Formatting Date Strings

Although computers like to work in timestamps, in many situations you need to convert a timestamp to a string representation of a date. Common scenarios include displaying a date on a Web page or passing a date to another application that expects to receive a date string in a specified format.

PHP's date() function lets you convert a timestamp to a date string. It works similarly to idate(), in that you pass it a format string and a timestamp to work with (omit the timestamp to convert the current date and time). The main difference is that the format string can contain multiple characters, allowing you to generate a date string containing as many components of the date and time as you like. You can also use additional formatting characters that are not available when using idate(). Here is a list of the date-related formatting characters allowed in date() format string:

Character	Description
j	The day of the month without leading zeros
d	The 2 - digit day of the month, with a leading zero of appropriate
D	The day of the week as a three-letter string (such as "Mon")
l	(lowercase 'L') The day of the week as a full word (such as "Monday")
w	The day of the week as a number (0=Sunday, 6=Saturday)
N	The day of the week as an ISO - 8601 number (1=Monday, 7=Sunday)
S	An English ordinal suffix to append to the day of the month (" st","nd," "rd," or" th"). Often used with the j formatting character
z	The day of the year (zero represents January 1)
W	The 2 - digit ISO-8601 week number in the year, with a leading zero if appropriate. Weeks start on Monday. The first week is week number 01
M	The month as a three-letter string (such as"Jan")
F	The month as a full word such as ("January")
n	The month as a number (1 - 12)
m	The month as a two-digit number, with a leading zero if appropriate (01 - 12)

Character	Description
t	The number of days in the month (28, 29, 30, 31)
y	The year as a two-digit number
Y	The year as a four-digit number
o (lowercase "o")	The ISO-8601 year number. This is usually the same value as Y; however if the ISO-8601 week number belongs to the previous or the next year, that year is used instead. For example, the ISO-8601 year number for January 1, 2000 is 1999
L	1 if the date is in a leap year, 0 otherwise

date() also allows the following time - formatting characters:

Character	Description
g	The hour in 12 - hour format, without leading zeros (1 – 12)
h	The hour in 12 - hour format, with leading zeros (01 – 12)
G	The hour in 24 - hour format, without leading zeros (0 – 23)
H	The hour in 24 - hour format, with leading zeros (00 – 23)
i	Minutes, with leading zeros (00 – 59)

Character	Description
s	Seconds, with leading zeros (00 – 59)
u	Microseconds (will always be zero because, at the time of writing, date() can only accept an integer timestamp)
B	Swatch Internet Time — a time - zone - free, decimal time measure. See http://en.wikipedia.org/wiki/Swatch_Internet_Time for details
a	"am" or "pm", depending on the value of the hour
A	"AM" or "PM", depending on the value of the hour
e	The full-time zone identifier of the currently set time zone (such as" UTC" or" America/Indiana/Indianapolis")
T	The time zone abbreviation for the currently set time zone (such as" UTC" or" EST"). Abbreviations are best avoided because the same abbreviation is often used for multiple time zones throughout the world
O	(capital "O") The time zone offset from GMT, in the format hhmm. For example, the "America/Indiana/Indianapolis" time zone is 5 hours behind GMT, so its offset is – 0500
P	Same as O, but with a colon between the hours and minutes (for example, -5:00)

Character	Description
Z	The time zone is offset from GMT, in seconds. For example, the offset in seconds for the" America/Indiana/Indianapolis" time zone is – 18000, because – 5 hours x 60 minutes x 60 seconds = – 18,000 seconds
I (capital "I")	1 if the currently set time zone is in daylight saving time; 0 otherwise

Note that the time zone formatting characters deal with the script's time zone because the timestamp is always in UTC. Usually, the script's time zone is set by the date. timezone directive in the php.ini file, but you can use PHP's date_default_timezone_set() function to change the time zone within your script, if necessary.

As well as the separate date and time formatting characters just mentioned, date() gives you three more formatting characters that return the date and time in one go:

Character	Description
c	The date and time as an ISO 8601 date. For example, 2006 - 03 - 28T19:42:00+11:00 represents March 28, 2006, at 7:42 in the evening, in a time zone that is 11 hours ahead of GMT
r	The date and time as an RFC 2822 date. For example, Tue, 28 Mar 2006 19:42:00 +1100 represents March 28, 2006, at 7:42 in the evening, in a time zone that is 11 hours ahead of GMT. RFC 2822 dates are commonly used in Internet protocols such as Web and email

Character	Description
U	The timestamp that was passed to date(), or the timestamp representing the current time if no timestamp was passed

For example, you could format a date and time in a nice, easy - to - read fashion like this:

```
$d = strtotime("March 28, 2006 9:42am");

// DISPLAYS"THE 28TH OF MARCH, 2006, AT 9:42 AM"
echo date("\T\h\e jS \o\\f F, Y, \a\\t g:i A", $d );
```

Notice that non - formatting characters in the format string need to be escaped with a backslash, and some special characters — like \f for the form feed character and \t for the tab character — need an additional backslash to escape them. date() converts the UTC timestamp supplied to your server ' s time zone. If you'd rather keep the date in UTC, use gmdate() instead:

```
// SET THE CURRENT TIME ZONE TO 5 HOURS BEHIND GMT
date_default_timezone_set("America/Indiana/Indianapolis");

// SET $D TO THE TIMESTAMP REPRESENTING MARCH 28, 2006 2:42 PM UTC
$d = strtotime("March 28, 2006 9:42am");

// DISPLAYS"MARCH 28, 2006 9:42 AM"
echo date("F j, Y g:i A", $d )\n";

// DISPLAYS"MARCH 28, 2006 2:42 PM"
echo gmdate("F j, Y g:i A", $d )\n";
```

Checking Date Values

Often a script needs to work with dates that have been entered by visitors to the site. For example, a Web form might contain three select menus allowing visitors to enter the

month, day, and year of their date of birth. However, in this scenario, there’s nothing to stop the visitors from entering a date that doesn’t exist, such as February 31, 2009. It would be a good idea to validate the date fields entered by the users to make sure they have supplied a legitimate date.

PHP's checkdate() function takes the month number (1 – 12), day number (1 – 31), and year components of a date, and returns true if the date is valid, or false if it's invalid:

```
echo checkdate( 2, 31, 2009 )."\n"; // DISPLAYS""(FALSE)
echo checkdate( 2, 28, 2009 )."\n"; // DISPLAYS"1"(TRUE)
```

It's a good idea to call checkdate() on any user-entered date before passing it to, say, mktime() for conversion to a timestamp.

Working with Microseconds

The date and time functions you've seen so far in this chapter work with integer timestamps — that is, timestamps representing whole numbers of seconds. Most of the time this is all you need. If you do need extra precision, use PHP's microtime() function. As with time() , microtime() returns a timestamp representing the current time. However, microtime() returns an additional microseconds component, allowing you to determine the current time more precisely:

```
// DISPLAYS, FOR EXAMPLE,"0.45968200 1230613358"
echo microtime();
```

As you can see, microtime() returns a string consisting of two numbers separated by a space. The first number is the microseconds component, represented as a fraction of a second, and the second number is the whole number of seconds — that is, the standard integer timestamp. So, the example output shown in the preceding code snippet represents 1,230,613,358.459682 seconds after midnight, Jan 1, 1970 (UTC). If you prefer, you can get microtime() to return a floating-point number of seconds, rather than a two - number string, by passing in an argument of true:

```
// DISPLAYS, FOR EXAMPLE,"1230613358.46"
echo microtime( true );
```

Note that using echo() only displays the number of seconds to two decimal places. To see the floating - point number more precisely, you can use printf():

```
// DISPLAYS, FOR EXAMPLE,"1230613358.459682"
printf("%0.6f", microtime( true ) );
```

One common scenario where microseconds are useful is when benchmarking your code to find speed bottlenecks. By reading the microtime() value before and after performing an operation, and then subtracting one value from the other, you can find out how long the operation took to execute.

Here’s an example:

```
// START TIMING
```

```
$startTime = microtime( true );

// PERFORM THE OPERATION
for ( $i=0; $i < 10; $i++ ) {
echo"Hello, world!\n";
}

// STOP TIMING
$endTime = microtime( true );
$elapsedTime = $endTime - $startTime;
printf("The operation took %0.6f seconds to execute.", $elapsedTime );
```

Chapter 13: Regular Expressions and Form Validation

PHP Regular Expressions

Introduction

Regular expressions provide you with a special syntax for searching for patterns of text within strings. At its simplest, a regular expression is simply a string of search text such as you would pass to strstr(). Regular expressions are enclosed in delimiters (usually forward slashes).

For example, this simple regular expression searches for the word " world " anywhere within the target string:

```
/world/
```

This, however, is a trivial example (in fact you would be better off using the faster strstr() in this case). Regular expressions start to become useful once you insert additional characters that have special meanings.

For example, the caret (^) character at the start of the expression means " the following characters must appear at the start of the target string":

```
/^world/
```

This example will match the string " world ", but not the string " hello, world " (because "world" isn't right at the start of the string).

Here are some simple examples of the kind of searches you can perform with regular expressions:

- The word " train " but not the word " training "
- At least one digit, followed by a single letter A, B, or C

- The word " hello " followed by between five and ten other characters, followed by " world "
- One or two digits, followed by the letters " st ", " nd ", " rd ", or " th ", followed by a space, followed by three letters (good for identifying dates embedded within strings)

You can see that, whereas strstr() can match only the exact string passed to it, regular expressions can contain a series of " rules " for creating quite complex matching patterns.

Pattern Matching in PHP

PHP's main pattern-matching function is preg_match(). This function takes the following arguments:

- The regular expression to search for (as a string)
- The string to search through
- An optional array to store any matched text in. (The matched text is stored in the first element.)
- An optional integer specifying any flags for the match operation. Currently, only one flag is supported: PREG_OFFSET_CAPTURE . Pass this constant to get preg_match() to return the position of any match in the array as well as the text matched. (If you need to pass a fifth argument to preg_match() and you want to turn off this feature, specify a value of zero instead.)
- An optional integer offset from the start of the string (the first character has an offset of zero, the second character has an offset of 1, and so on). If specified, preg_match() starts the search from this position in the string, rather than from the first character

preg_match() returns zero if the pattern didn't match, or 1 if it did match. (preg_match() only finds the first match within the string. If you need to find all the matches in a string, use preg_match_all())

For example, to match the word " world " anywhere in the string " Hello, world! " you can write:

```
echo preg_match( "/world/", "Hello, world!" ); // DISPLAYS "1"
```

To match " world " only at the start of the string, you'd write:

```
echo preg_match( "/^world/", "Hello, world!" );// DISPLAYS "0"
```

To access the text that was matched, pass an array variable as the third argument:

```
echo preg_match( "/world/", "Hello, world!", $match ); // DISPLAYS "1"
echo $match[0]; // DISPLAYS "WORLD"
```

To find out the position of the match, pass PREG_OFFSET_CAPTURE as the fourth argument. The array then contains a nested array whose first element is the matched text and whose second element is the position:

```
echo preg_match( "/world/", "Hello, world!",$match, PREG_OFFSET_CAPTURE ); //
DISPLAYS "1"
echo $match[0][0]; // DISPLAYS "WORLD"
echo $match[0][1]; // DISPLAYS "7"
```

Finally, to start the search from a particular position in the target string, pass the position as the fifth argument:

```
echo preg_match( "/world/", "Hello, world!", $match, 0, 8 ); // DISPLAYS "0"
```

(This example displays zero because the " world " text starts at position 7 in the target string.)

Now that you know how PHP's regular expression matching function works, it's time to learn how to write regular expressions.

Matching Literal Characters

The simplest form of a regular expression pattern is a literal string. In this situation, the string stored in the pattern matches the same string of characters in the target string, with no additional rules applied. As you've already seen, alphabetical words such as " hello " are treated as literal strings in regular expressions. The string " hello " in a regular expression matches the text " hello " in the target string. Similarly, many other characters — such as digits, spaces, single and double quotes, and the %, &, @, and # symbols — are treated literally by the regular expression engine. However, as you see later, some characters have special meanings within regular expressions. These nineteen special characters are:

\ + * ? [^] $ ( ) { } = ! < > | :

If you want to include any character from this list literally within your expression, you need to escape it by placing a backslash (\ ) in front of it, like so:

```
echo preg_match( "/love\?/", "What time is love?" ); // DISPLAYS "1"
```

Because the backslash is itself a special character, you need to escape it with another backslash (\\ ) if you want to include it literally in an expression. What's more, because a backslash followed by another character within a string is itself seen as an escaped character in PHP, you usually need to add a third backslash (\\\ ). Also, if you use your delimiter character within your expression, you need to escape it:

```
echo preg_match( "/http\:\/\//", "http://www.example.com" ); // DISPLAYS "1"
```

Slashes are commonly used as regular expression delimiters, but you can use any symbol you like (provided you use the same symbol at both the start and end of the expression). This is useful if your expression contains a lot of slashes. By using a different delimiter, such as the | (vertical bar) character, you avoid having to escape the slashes within the expression:

```
echo preg_match( "|http\://|", "http://www.example.com" ); // DISPLAYS "1"
```

Although some of these special characters are sometimes treated literally in certain contexts, it's a good idea always to escape them. (However, don't try to escape letters and digits by placing a backslash in front of them, because this conveys a different special meaning, as you see later.)

You can write various characters literally within regular expressions by using escape sequences, as follows:

Escape Sequence	Meaning
\n	A line feed character (ASCII 10)
\r	A carriage return character (ASCII 13)
\t	A horizontal tab character (ASCII 9)
\e	An escape character (ASCII 27)
\f	A form feed character (ASCII 12)
\a	A bell (alarm) character (ASCII 7)
\xdd	A character with the hex code dd (for example, \x61 is the ASCII letter " a ")
\ddd	A character with the octal code ddd (for example, \141 is the ASCII letter " a ")
\cx	A control character (for example, \cH denotes ^H, or backspace)

Matching Types of Characters using Character Classes Rather than searching for a literal character, often it's useful to search for a certain class or type of character. For example, you might care only that the character is a digit, or that it is one of the letters A, B, or C. By placing a set of characters in square brackets, you can search for a single character that matches any one of the characters in the set. For example, the following expression matches " a ", " b ", " c ", " 1 ", " 2 ", or " 3 " :

```
echo preg_match( "/[abc123]/", "b" ); // DISPLAYS "1"
```

You can specify ranges of characters using the hyphen (-) symbol. The following example matches the same set of characters as the previous example:

```
echo preg_match( "/[a-c1-3]/", "b" ); // DISPLAYS "1"
```

So, you can match any letter or digit using:

```
echo preg_match( "/[a-zA-Z0-9]/", "H" ); // DISPLAYS "1"
```

To negate the sense of a character class — that is, to match a character that is not one of the characters in the set — place a caret (^) symbol at the start of the list:

```
echo preg_match( "/[abc]/", "e" ); // DISPLAYS "0"
echo preg_match( "/[^abc]/", "e" ); // DISPLAYS "1"
```

You don't need to escape most of the previously mentioned special characters when they're inside a character class. The exceptions are the caret, which still needs to be escaped (unless you're using it to negate the class as just shown), and the backslash, which is used for specifying shorthand character classes, as you see in a moment. You can also use various shorthand character classes comprising a backslash followed by one of several letters, as follows:

Character Class	Meaning
\d	A digit \D Any character that isn't a digit
\w	A word character (letter, digit, or underscore)
\W	Any character that isn't a word character
\s	A whitespace character (space, tab, line feed, carriage return, or form feed)
\S	Any character that isn't a whitespace character

So, to match a digit character anywhere in the target string you could use either of the following two expressions:

```
/[0-9]/
/\d/
```

Incidentally, you can also use a shorthand character class within a longhand class. The following expression matches the letter " e" or " p ", or any digit, in the target string:

```
/[ep\d]/
```

Here are some examples:

```
echo preg_match( "/\d[A-Z]/", "3D" ); // DISPLAYS "1"
echo preg_match( "/\d[A-Z]/", "CD" ); // DISPLAYS "0"
echo preg_match( "/\S\S\S/", "6&c" ); // DISPLAYS "1"
echo preg_match( "/\S\S\S/", "6c" ); // DISPLAYS "0"
```

To match any character at all, use a dot (.):

```
echo preg_match( "/He.../", "Hello" ); // DISPLAYS "1"
```

Matching Multiple Characters

If you want to match the same character (or character class) multiple times in a row, you can use quantifiers. A quantifier is placed after the character or character class and indicates how many times that character or class should repeat in the target string. The quantifiers are:

Quantifier	Meaning
*	Can occur zero or more times
+	Can occur one or more times
?	Can occur exactly once, or not at all
{n}	Must occur exactly n times
{n,}	Must occur at least n times
{n,m}	Must occur at least n times but no more than m times.

For example, you can match a string of at least one digit with:

```
/\d+/
```

Say you wanted to search a string for a date in the format mmm/dd/yy or mmm/dd/yyyy (for example, jul/15/06 or jul/15/2006). That's three lowercase letters, followed by a slash, followed by one or two digits, followed by a slash, followed by between two and four digits. This regular expression will do the job:

```
echo preg_match( "/[a-z]{3}\/\d{1,2}\/\d{2,4}/", "jul/15/2006" ); // DISPLAYS "1"
```

(This expression isn't perfect — for example, it will also match three-digit " years, " but you get the idea.)

Greedy and Non - Greedy Matching

When you use quantifiers to match multiple characters, the quantifiers are greedy by default. This means that they will try to match the largest number of characters possible. Consider the following code:

```
preg_match( "/P.*r/", "Peter Piper", $matches );
echo $matches[0]; // DISPLAYS "PETER PIPER"
```

The regular expression reads, " Match the letter 'P' followed by zero or more characters of any type, followed by the letter “r”. " Because quantifiers are, by nature, greedy, the regular expression engine matches as many characters as it can between the first " P" and the last " r" — in other words, it matches the entire string. You can change a quantifier to be non - greedy. This causes it to match the smallest number of characters possible. To make a quantifier non - greedy, place a question mark (?) after the quantifier.

For example, to match the smallest possible number of digits use :/\d+?/ Rewriting the Peter Piper example using a non - greedy quantifier gives the following result:

```
preg_match( "/P.*?r/", "Peter Piper", $matches );echo $matches[0]; //
Displays "Peter"
```

Here, the expression matches the first letter " P" followed by the smallest number of characters possible (" ete "), followed by the first letter "r ".

Using Subpatterns to Group Patterns

By placing a portion of your regular expression's rules in parentheses, you can group those rules into a subpattern. A major benefit of doing this is that you can use quantifiers (such as * and ?) to match the whole subpattern a certain number of times. For example:

```
// DISPLAYS "1"
echo preg_match( "/(row,? )+your boat/", "row, row, row your boat" );
```

The subpattern in this regular expression is " (row,?) ". It means: " The letters 'r', 'o', and 'w', followed by either zero or one comma, followed by a space character. " This subpattern is then matched at least one-time thanks to the following + quantifier, resulting in the " row, row, row " portion of the target string being matched. Finally, the remaining characters in the pattern match the " your boat " part of the string. The result is that the entire string is matched.

A side - effect of using subpatterns is that you can retrieve the individual subpattern matches in the matches array passed to preg_match() . The first element of the array contains the entire matched text as usual, and each subsequent element contains any matched subpatterns:

```
preg_match( "/(\d+\/\d+\/\d+) (\d+\:\d+.+)/", "7/18/2004 9:34AM", $matches );
echo $matches[0]; // DISPLAYS "7/18/2004 9:34AM"
echo $matches[1]; // DISPLAYS "7/18/2004"
echo $matches[2]; // DISPLAYS "9:34AM"
```

Referring to Previous Subpattern Matches You can take the text that matched a subpattern and use it elsewhere in the expression. This is known as a backreference. Backreferences allow you to create quite powerful, adaptable regular expressions. To include a subpattern's matched text later in the expression, write a backslash followed by the subpattern number. For example, you'd include the first subpattern's matched text by writing \1, and the next subpattern's matched text by writing \2. Consider the following example:

```
$myPets = "favoritePet=Lucky, Rover=dog, Lucky=cat";
preg_match( '/favoritePet\=(\w+).*\1\=(\w+)/', $myPets, $matches );

// DISPLAYS "MY FAVORITE PET IS A CAT CALLED LUCKY."
echo "My favorite pet is a " . $matches[2] . " called " . $matches[1] . ".";
```

This code contains a string describing someone's pets. From the string, you know that their favorite pet is Lucky and that they have two pets: a dog called Rover and a cat called Lucky. By using a regular expression with a backreference, the code can deduce that their favorite pet is a cat.

Here's how the expression works. It first looks for the string " favoritePet= " followed by one or more word characters (" Lucky " in this case):/favoritePet\=(\w+). Next, the expression looks for zero or more characters of any type, followed by the string that the first subpattern matched (" Lucky "), followed by an equals sign, followed by one or more word characters (" cat " in this example):.*\1\=(\w+). Finally, the code displays the results of both subpattern matches (" Lucky " and " cat ") in a message to the user.

By the way, notice that the expression string was surrounded by single quotes in this example, rather than the usual double-quotes. If the code had used double quotes, an extra backslash would have been needed before the \1 (because PHP assumes \1 to be the ASCII character with character code 1 when inside a double-quoted string):

```
preg_match( "/favoritePet\=(\w+).*\\1\=(\w+)/", $myPets, $matches );
```

Character escaping issues like this have been known to trip up many a seasoned programmer, so this is something to watch out for. Matching Alternative Patterns Regular expressions let you combine patterns (and subpatterns) with the | (vertical bar) character to create alternatives. This is a bit like using the || (or) operator; if any of the patterns combined with the |character match, the overall pattern matches. The following pattern matches if the target string contains any one of the abbreviated days of the week (mon-sun):

```
$day = "wed";
echo preg_match("/mon|tue|wed|thu|fri|sat|sun/", $day); // DISPLAYS "1"
```

You can also use alternatives within subpatterns, which is very handy. Here's the earlier " date detection " example, rewritten to be more precise:

```
echo preg_match( "/(jan|feb|mar|apr|may|jun|jul|aug|sep|oct|nov|dec)" ."\/\d{1,2}\/\d{2,4}/",
"jul/15/2006" ); // DISPLAYS "1"
```

Using Anchors to Match at Specified Positions Often you're interested in the position of a pattern within a target string, as much as the pattern itself. For example, say you wanted to make sure that a string started with one or more digits followed by a colon. You might try this:

```
echo preg_match( "/\d+\:/", "12: The Sting" ); // DISPLAYS "1"
```

However, this expression would also match a string where the digits and colon are somewhere in the middle:

```
echo preg_match( "/\d+\:/", "Die Hard 2: Die Harder" ); // DISPLAYS "1"
```

How can you make sure that the string only matches if the digits and colon are at the start? The answer is that you can use an anchor (also known as an assertion), as follows:

```
echo preg_match( "/^\d+\:/", "12: The Sting" ); // DISPLAYS "1"
echo preg_match( "/^\d+\:/", "Die Hard 2: Die Harder" ); // DISPLAYS "0"
```

The caret (^) symbol specifies that the rest of the pattern will only match at the start of the string. Similarly, you can use the dollar ($) symbol to anchor a pattern to the end of the string:

```
echo preg_match( "/\[(G|PG|PG-13|R|NC-17)\]$/", "The Sting [PG]" );

// DISPLAYS "1"


echo preg_match( "/\[(G|PG|PG-13|R|NC-17)\]$/", "[PG] Amadeus" );

// DISPLAYS "0"
```

By combining the two anchors, you can ensure that a string contains only the desired pattern, nothing more:

```
echo preg_match( "/^Hello, \w+$/", "Hello, world" ); // DISPLAYS "1"
echo preg_match( "/^Hello, \w+$/", "Hello, world!" ); // DISPLAYS "0"
```

The second match fails because the target string contains a non-word character (!) between the searched-for pattern and the end of the string. You can use other anchors for more control over your matching. Here's a full list of the anchors you can use within a regular expression:

Anchor	Meaning
^	Matches at the start of the string
$	Matches at the end of the string
\b	Matches at a word boundary (between a \w character and a \W character)
\B	Matches except at a word boundary
\A	Matches at the start of the string
\z	Matches at the end of the string
\Z	Matches at the end of the string or just before a newline at the end of the string
\G	Matches at the starting offset character position, as passed to the preg_match()function

It's important to note that an anchor doesn't itself match any characters; it merely ensures that the pattern appears at a specified point in the target string.

\A and \z are similar to ^ and $. The difference is that ^ and $ will also match at the beginning and end of a line, respectively, if matching a multi-line string in multi-line mode (explained in the " Altering Matching Behavior with Pattern Modifiers " section later in this lesson). \A and \z only match at the beginning and end of the target string, respectively. \Z is useful when reading lines from a file that may or may not have a newline character at the end. \b and \B are handy when searching the text for complete words:

```
echo preg_match( "/over/", "My hovercraft is full of eels" );

// DISPLAYS "1"


echo preg_match( "/\bover\b/", "My hovercraft is full of eels" );

// DISPLAYS "0"


echo preg_match( "/\bover\b/", "One flew over the cuckoo's nest" );

// DISPLAYS "1"
```

When using \b , the beginning or end of the string is also considered a word boundary:

```
echo preg_match( "/\bover\b/", "over and under" ); // DISPLAYS "1"
```

By using the \b anchor, along with alternatives within a subexpression, it's possible to enhance the earlier " date detection " example further, so that it matches only two - or four - digit years (and not three -digit years):

```
echo preg_match( "/\b(jan|feb|mar|apr|may|jun|jul|aug|sep|oct|nov|dec)"
."\/\d{1,2}\/(\d{2}|\d{4})\b/", "jul/15/2006" ); // DISPLAYS "1"

echo preg_match( "/\b(jan|feb|mar|apr|may|jun|jul|aug|sep|oct|nov|dec)"
."\/\d{1,2}\/(\d{2}|\d{4})\b/", "jul/15/206" ); // DISPLAYS "0"
```

The last part of the expression reads, " Match either two digits or four digits, followed by a word boundary (or the end of the string) ":

```
(\d{2}|\d{4})\b
```

You can also create your own types of anchor; for example, you can match text only when it comes before an ampersand, or only when it follows a capital letter (without actually including the ampersand or capital letter in the match). These kinds of custom anchors are known as lookahead and lookbehind assertions you can read about them in the PHP manual at http://www.php.net/manual/en/regexp.reference.assertions.php.

Altering Matching Behavior with Pattern

Modifiers By placing a single letter, known as a pattern modifier, directly after the closing delimiter of a regular expression, you can change the way that the expression behaves. Here's a list of the more useful modifiers:

Modifier	Description
i	Causes the matching to be case insensitive: letters in the pattern match both upper - and lowercase characters in the string
m	Causes the target string to be treated as separate lines of text if it contains newlines. This means that ^ and $ characters in the expression match not only the beginning and end of the string but also the beginning and end of each line in the string
s	Normally, the dot (.) character in an expression matches any character except newline characters. By adding this modifier, you can make the dot character match newlines too

Modifier	Description
x	This modifier causes whitespace characters in the pattern to be ignored, rather than treated as characters to match. (However, whitespace inside a character class is never ignored.) This allows you to split your regular expression over lines and indent it, much like regular PHP code, to aid readability. You can also include comments in the expression by preceding them with a # symbol. If you explicitly want to match whitespace characters when using this modifier, use " \ " (for a space), " \t " (for a tab), or " \s " (for any whitespace character)
e	Only used by preg_replace(). This modifier allows you to use PHP code in your replacement string. Any backreferences ($1, $2, and so on) in the replacement string are first replaced by their matched text. Then the string is evaluated as PHP code, and the resulting expression used for the replacement
U	Inverts the " greediness " of quantifiers within the expression: any non - greedy quantifiers become greedy, and any greedy quantifiers become non - greedy

For example, you can make an expression case insensitive by adding i after the closing delimiter of the expression:

```
$text = "Hello, world!";echo preg_match( "/hello/", $text ); // DISPLAYS "0"
echo preg_match( "/hello/i", $text ); // DISPLAYS "1"
```

The following example shows how the m modifier works. The first expression attempts to match the characters " world! " followed by the end of the string. Because " world! " is

not at the end of the target string, the match fails. However, the second expression uses the m modifier. This causes the $ character to match the newline after "world!":

```
$text = "Hello, world!\nHow are you today?\n";
echo preg_match( "/world!$/", $text ); // DISPLAYS "0"
echo preg_match( "/world!$/m", $text ); // DISPLAYS "1"
```

The m modifier is useful if you're working with a multiline string (such as that read from a file or database query) that you want to treat as multiple lines of text rather than as one long string. By adding the x modifier to your expression, you can split the expression over multiple lines and add comments — very handy for complex returns.

PHP had other functions using regular expressions like preg_match_all() and preg_split(). You can find them in the PHP manual.

Sanitizing Data

PHP sticky form and sanitizing data

When a user sends data to the server through a form that data needs to be sanitized before it can be used. What we mean by sanitized is the data is checked to make sure it is what we are intending and not some malicious code. Data sanitation must be done on the server using PHP.

Each time we submit a form the page goes to whatever script is going to process the form data. Once a user submits the form, they have lost the data, so if there is an error, they have to re-enter the data they just entered. This is a bad user experience and not one that we want to create. The solution is creating what is called a "sticky form" that will remember the data the user sent and if there is an error in the data (the wrong format, no data entered, etc) the application will display the form data back, and any custom error message for incorrectly submitted data.

The following example http://198.199.80.235/cps276/cps276_examples/php-form-validation/index.php demonstrates this technique.

The code can be found on my GitHub page at https://GitHub.com/sshaper/cps276_examples/tree/master/php-form-validation

Data sanitation using AJAX

Many times, JavaScript is used for data sanitation. This should be done on both the client-side using JavaScript and the backend using PHP. The JavaScript only validation creates a good user experience but is insecure. Validating the backend is a must because it is secure. Using AJAX, you can validate a form on the backend with PHP and use JavaScript to display the messages. It is a little less verbose than just using PHP.

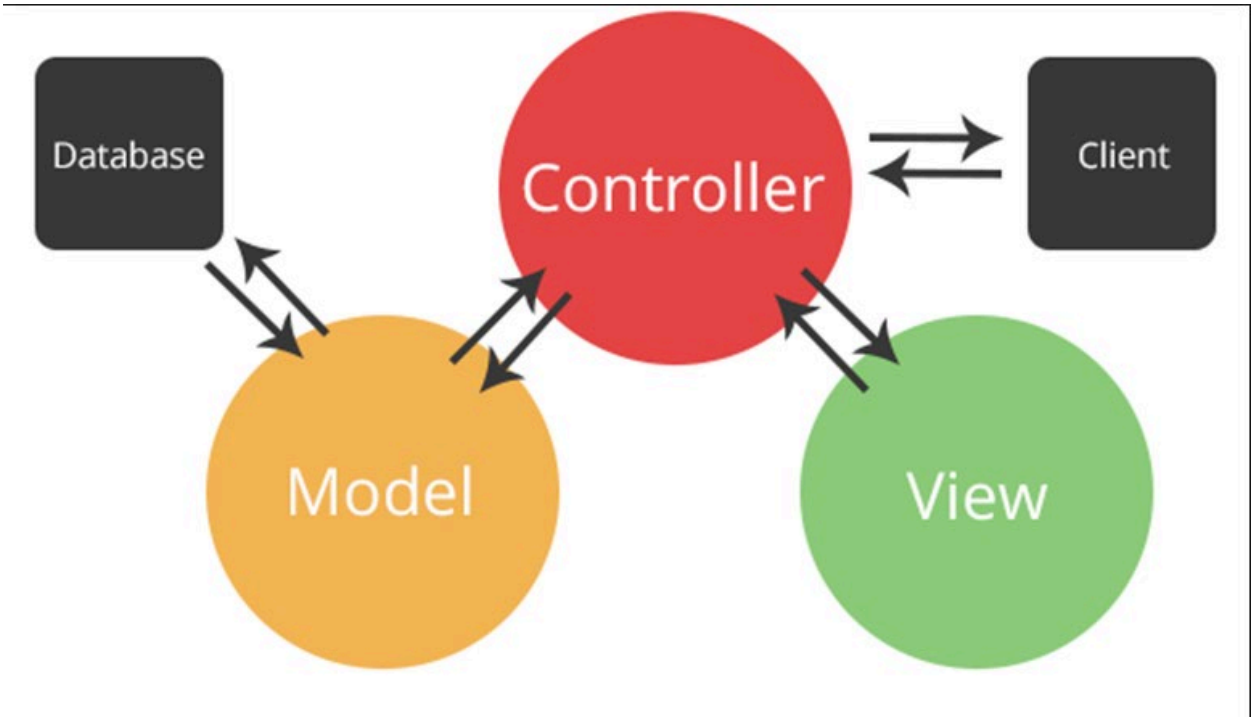
The following example `http://198.199.80.235/cps276/cps276_examples/php-form-validation-ajax/` demonstrates this technique. The source code is on my GitHub page at https://GitHub.com/sshaper/cps276_examples/tree/master/php-form-validation-ajax.

Chapter 14: MVC

MVC (Model View Controller)

Introduction

Model View Controller or MVC as it is popularly called, is an Architectural pattern for developing web applications. A Model View Controller pattern is made up of the following three main parts (model, view, and controller).



MVC is popular as it isolates the application logic from the user interface layer and supports separation of concerns. Here the Controller receives all requests for the application and then works with the Model to prepare any data needed by the View. The View then uses the data prepared by the Controller to generate a final presentable response.

The model

The model is responsible for managing the data of the application. It responds to the request from the controller and it also responds to instructions from the controller to update itself.

The view

A presentation of data in a format, triggered by a controller's decision to present the data. They are usually in script-based templating.

The controller

The controller is responsible for responding to user input and perform interactions on the data model objects. The controller receives the input, validates the input, and then performs the business operation that modifies the state of the data model.

There are some other subparts like routes, public folders, xhr, etc. We will look at these in the example.

Putting Files in Their Place

As stated, the advantage of an MVC architecture is that everything has a place, which makes it much easier to manage your site. At first, it seems confusing, and on small sites, it seems to be verbose, but it makes things much easier to manage and is used in most larger applications.

MVC in PHP

PHP is not natively written to utilize MVC. Many of the PHP frameworks use an MVC architecture but they must initially be built from scratch.

The best way to understand MVC architecture is to see an application built that way. I have created an MVC example at https://GitHub.com/sshaper/cps276_examples/tree/master/php_mvc. The rest of this section will discuss the parts of that example.

MVC Example Files

While reading this section make sure to have the cps276_examples/php_mvc folder open located at https://GitHub.com/sshaper/cps276_examples/tree/master/php_mvc

index.php

The first file you should see is the index.php file. This one file is designed to display all the views of this application. In a sense, it is a container for views that will be displayed. If you notice it requires on file server/pageroutes.php, that file takes the web address entered into the browser window and based upon the $_GET global array value will call the necessary classes and files to create the page view then that view will be displayed in the index.php page.

server/pageroutes.php

You can see from the code sample below that we take a value from $_GET[‘file’] and use send that to a switch statement. Based upon what the page is the switch statement requires a file and calls a function.

For example, if "homepage" is the value then the script requires the controller/pages.php page and we call the homepage function.

```php
<?php
if (isset($_GET['file'])){
        $page = $_GET['file'];
}

else {
        $page = "homePage";
}

switch($page){
        case "homePage" : require_once 'controller/pages.php'; $pageData = homePage(); break;
        case "addNamePage" : require_once 'controller/pages.php'; $pageData = addNamePage(); break;
        case "updateDeleteNamePage" : require_once 'controller/pages.php'; $pageData = updateDeleteNamePage(); break;
        default : require_once 'controller/pages.php'; $pageData = homePage(); break;
}
```

controller/pages.php

If we look at the homepage function, we can see that it creates an associative array named $pageData that contains the information needed to be displayed on the index.php page.

We call the function getNames(list) and store that into the $namelist variable. The getName(list) function calls the database and returns a list of names from the database in list format. That function can be found in the controller/crud.php file.

We are instantiating the Page class and bringing in the nav method. The nav method returns a navigation menu that is used for the whole site.

```
function homePage(){
        $nameList = getNames('list');
        $page = new Page();

        $pageData['title'] = "Home Page";
        $pageData['heading'] = "Home Page";
        $pageData['nav'] = $page->nav();
        $pageData['content'] = require_once 'views/home.php';
        $pageData['js'] = "";
        return $pageData;
}
```

We are requiring the home.php page from the views folder. That page will be the HTML content for the index.php. All the content for the index.php page is stored in separate files in the views folder.

When looking at the views/home.php code you can see that we inject $namelist into it. That is where the list is stored that was returned from the getNames(list) function.

On the next page is a screenshot of the views/home.php code (the paragraph text is cut off in the screenshot, but you should be the idea.

```
<?php

$page = <<<HTML

<main>
  <p>This example (in addition to the pdo stuff) is a great example for showing how a MVC type

  <div>{$nameList}</div>
</main>

HTML;

return $page;

?>
```

server/xhrRoutes.php

We saw how a typical page was displayed using the routes and the controller. But what about the AJAX calls that we do in this application. Well, they are handled similarly except that any AJAX call will go to the server/xhrRoutes.php file. This file will do the same thing as the other routes file but for AJAX request only.

See screenshot below.

```
<?php

$data = $_POST['data'];
$data = json_decode($data);

switch($data->flag){
        case "addName" : require_once '../controller/crud.php'; echo addName($data); break;
        case "update" : require_once '../controller/crud.php'; echo updateName($data); break;
        case "delete" : require_once '../controller/crud.php'; echo deleteName($data); break;
}

?>
```

You can see from the above screenshot that every AJAX request requires the controller/crud.php file. The crud.php file contains various functions (as shown in the screenshot) that do all the database calls needed to get/put the required data. Notice

that the file is in the controller folder. That is because it is a controller file that gets the data from the model and returns it to the view. I would recommend you look at both the pages.php and crud.php files to see what those files are doing.

Other classes

In the classes folder, I have other classes that do certain things like, connect to the database, do the PDO methods, and a Page.php class for global related page items (in this case a navigation bar).

.htaccess file

The .htaccess file is used to create what is called “pretty URLs”. Google does not like URLs that are in the form of

index.php?file=somefilename

What this .htaccess file does is take what is entered into the browser window (a friendly URL) and convert it into a URL that the server needs. For example, in the application the home page URL is:

http://198.199.80.235/cps276/cps276_examples/php_mvc/homePage

But the actual file the server needs is

index.php?file=homepage

The .htaccess file converts the URL from a search-friendly URL to a URL the server needs to use.