European IT Certification Curriculum
Self-Learning Preparatory Materials

EITC/WD/PMSF

PHP and MySQL Fundamentals

This document constitutes European IT Certification curriculum self-learning preparatory material for the EITC/WD/PMSF PHP and MySQL Fundamentals programme.

This self-learning preparatory material covers requirements of the corresponding EITC certification programme examination. It is intended to facilitate certification programme's participant learning and preparation towards the EITC/WD/PMSF PHP and MySQL Fundamentals programme examination. The knowledge contained within the material is sufficient to pass the corresponding EITC certification examination in regard to relevant curriculum parts. The document specifies the knowledge and skills that participants of the EITC/WD/PMSF PHP and MySQL Fundamentals certification programme should have in order to attain the corresponding EITC certificate.

Disclaimer

This document has been automatically generated and published based on the most recent updates of the EITC/WD/PMSF PHP and MySQL Fundamentals certification programme curriculum as published on its relevant webpage, accessible at:

https://eitca.org/certification/eitc-wd-pmsf-php-and-mysql-fundamentals/

As such, despite every effort to make it complete and corresponding with the current EITC curriculum it may contain inaccuracies and incomplete sections, subject to ongoing updates and corrections directly on the EITC webpage. No warranty is given by EITCI as a publisher in regard to completeness of the information contained within the document and neither shall EITCI be responsible or liable for any errors, omissions, inaccuracies, losses or damages whatsoever arising by virtue of such information or any instructions or advice contained within this publication. Changes in the document may be made by EITCI at its own discretion and at any time without notice, to maintain relevance of the self-learning material with the most current EITC curriculum. The self-learning preparatory material is provided by EITCI free of charge and does not constitute the paid certification service, the costs of which cover examination, certification and verification procedures, as well as related infrastructures.

TABLE OF CONTENTS

EITC/WD/PMSF PHP AND MYSQL FUNDAMENTALS DIDACTIC MATERIALS
LESSON: INTRODUCTION
TOPIC: REASONS TO LEARN PHP

PHP and MySQL are fundamental technologies in web development. Despite some negative opinions, PHP continues to be widely used in popular content management systems, e-commerce platforms, and websites like WordPress, Magento, Drupal, Facebook, Tumblr, and Slack. Learning PHP is beneficial for several reasons.

Firstly, it is easy for beginner web developers to pick up PHP as it is tightly coupled with HTML. Many web servers are pre-configured to run PHP, making it easy to upload and run PHP sites on the internet.

Secondly, PHP has a large and active community of support, which is helpful for beginners and intermediate programmers. Despite the preferences of more experienced developers, PHP remains a significant player in web development, making it a valuable addition to one's skillset.

Lastly, PHP offers numerous job opportunities in web development. It is one of the most requested tutorial series on various platforms, including this channel.

Now, let's discuss what PHP is and what you will learn in this course. PHP stands for PHP Hypertext Preprocessor, which is a server-side scripting language used to create dynamic websites. Unlike HTML, CSS, and JavaScript, which run in the browser, PHP runs on the server.

When you visit a website, the browser sends a request to the server hosting that website. On the server, the PHP code executes, allowing us to generate dynamic HTML templates. For example, we can compile a dynamic HTML template and send it back to the browser. Additionally, PHP enables us to process user input, store it in a database, and retrieve it later.

Throughout this course, we will cover various topics, starting with setting up PHP on your computer and exploring essential tools. We will then dive into the basics of PHP, creating PHP files, and rendering dynamic content in HTML templates using PHP. We will also learn how to communicate with a MySQL database, work with sessions and cookies, and explore objects and classes in PHP. Additionally, we may touch upon some newer features introduced in PHP 7.

By the end of this course, you will have a solid understanding of PHP and MySQL fundamentals, enabling you to create dynamic websites and interact with databases.

In web development, PHP and MySQL are fundamental technologies that allow us to create dynamic and interactive websites. In this course, we will explore the basics of PHP and how it can be used in conjunction with MySQL to build a simple website.

The website we will be creating is called "Ninja Pizza." It is a basic website where users can add new pizzas, read about existing pizzas, and delete pizzas. The pizzas are stored in a MySQL database, and we can view their names, ingredients, and creation details. We can also delete pizzas from the database.

To demonstrate the functionality of the website, we have three pizzas listed on the page. These pizzas are retrieved from the MySQL database and displayed on the website. By clicking on "More Information," we can view additional details about the pizza, such as who created it and when it was created. We also have the option to delete pizzas directly from the website.

To add a new pizza, we need to provide an email address and a pizza title. In the example given, the email address is "Mario@netNinjaCode.co.uk," and the pizza title is "The Mario Supreme." We can then add ingredients, such as tomato, cheese, and mushrooms, and submit the form. The new pizza will be added to the list on the website.

Throughout this course, we will be using Sublime Text as our text editor. While this choice is based on personal preference and nostalgia, you are free to use any text editor of your choice, such as Atom, VS Code, or Sublime Text. If you decide to use Sublime Text, you can download it from the official website at sublimetext.com.

Additionally, all the course files for this playlist are available on a GitHub repository. You can find the link to the repository below this material. It is important to note that each lesson has its own separate branch in the repository. If you want to access the code for a specific lesson, you need to select the corresponding branch from the branch dropdown menu. This ensures that you can view the code for each lesson.

This course will provide you with an introduction to PHP and MySQL in the context of web development. By following along and completing the exercises, you will gain a solid foundation in these technologies and be able to create dynamic websites. We hope you enjoy this playlist, and if you find the videos helpful, please consider sharing, subscribing, and liking them. We look forward to seeing you in the next material.

EITC/WD/PMSF PHP AND MYSQL FUNDAMENTALS DIDACTIC MATERIALS
LESSON: GETTING STARTED WITH PHP
TOPIC: INSTALLING PHP (XAMPP)

To get started with PHP, the first step is to install PHP onto your computer. In addition to PHP, you will also need to install a MySQL database, as you will be storing data in this database later on in the course. PHP will communicate with the database to retrieve and display data in HTML templates.

To serve up your PHP files, you will need a local development server. In this case, we recommend using XAMPP (or ZAMPP), which stands for cross-platform Apache, MariaDB, PHP, and Perl. XAMPP is a tool that installs all of these components for you, making the setup process easier.

To install XAMPP, you can download it from the provided link. Once downloaded, run the installer and follow the instructions. During the installation, you can leave the default settings as they are. After the installation is complete, you will see the XAMPP control panel.

In the control panel, you can start the necessary services. Start the Apache module, which will act as your development server. You can also start the MySQL module, but it is not required at this point. The control panel will indicate whether the services are running or not.

To access the XAMPP dashboard, open a web browser and go to "localhost/dashboard". This means that your computer is acting as a server, serving up the web page from the local development server. This is similar to how websites are served from remote servers, but in this case, you are using your own computer as the server.

If you encounter a port error, you can change the port number that Apache listens to. In the XAMPP control panel, go to the "Config" section and open the Apache config file. Look for the "listen" directive and change the port number to a different value, such as 8090. Save the file and restart the Apache module.

Using "localhost" to preview your website locally is a common practice in web development, not just for PHP but for other programming languages as well.

To get started with PHP, you need to install PHP, a MySQL database, and a local development server. XAMPP is a recommended tool that installs all these components for you. After installation, you can start the necessary services in the XAMPP control panel and access the dashboard through "localhost/dashboard".

To get started with PHP development, you will need to install a local development server on your computer. One popular option is XAMPP, which includes the Apache web server, PHP, and MySQL. In this didactic material, we will guide you through the process of installing XAMPP and explain how web pages are served.

First, let's install XAMPP. Visit the XAMPP website and download the appropriate version for your operating system. Once the download is complete, run the installer and follow the on-screen instructions. During the installation, you will be prompted to select the components you want to install. Make sure to select Apache, PHP, and MySQL.

After the installation is complete, open XAMPP Control Panel. Start the Apache server by clicking the "Start" button next to it. You will notice that the server is listening on port 80 by default. If you encounter any issues with this port, you can change it in the configuration file.

To find out where your web pages are being served from, navigate to the XAMPP installation directory. In the "htdocs" folder, you will find all the files that are served by the Apache server. For example, if you have a folder named "dashboard" inside "htdocs" and it contains an "index.html" file, when you access "localhost/dashboard" in your browser, the server will automatically serve the "index.html" file.

By default, when you navigate to a directory in the URL, the server looks for an "index.html" or "index.php" file and serves it if it exists. You can create your own files and directories inside the "htdocs" folder to serve your web pages. For example, if you create a folder named "test" and inside it, a file named "test.html", you can access it by visiting "localhost/test" in your browser.

To create a PHP file, open a text editor and save the file with a ".php" extension. You can then write PHP code inside this file. When you access the PHP file through the server, it will execute the PHP code and display the result in the browser.

In the next material, we will guide you through creating your first PHP file and running it on your computer.

EITC/WD/PMSF PHP AND MYSQL FUNDAMENTALS DIDACTIC MATERIALS
LESSON: GETTING STARTED WITH PHP
TOPIC: YOUR FIRST PHP FILE

To get started with PHP, it is important to understand how PHP files are run on the server and how they are served to the browser using a local development server. In this tutorial, we will create a folder inside the "htdocs" directory to contain all the course files for this project.

To begin, navigate to the "xampp" folder, then go to "htdocs" and create a new folder. Let's name this folder "torts tutorials". This folder will serve as the location for all our code files. When previewing the code in a browser, we will access it using the URL path "/torts".

Next, open the folder in your preferred code editor, such as Sublime Text. Create a new file inside the "torts tutorials" folder by right-clicking and selecting "New File". Save the file with the name "index.php". It is important to note that PHP files have the extension ".php".

To write PHP code inside the file, we need to use PHP tags. These tags are similar to HTML tags and are used to encapsulate PHP code. To open a PHP tag, use "<?php" and to close it, use "?>". Alternatively, you can use a shortcut in Sublime Text by typing "php" and then pressing the "Tab" key.

Now, let's write a simple PHP statement inside the PHP tags. We will use the "echo" statement, which is used to output text. Inside the "echo" statement, we can include a string, which is a collection of letters, numbers, and symbols enclosed in quotation marks. For example, we can use the statement "echo 'Hello ninjas';" to output the string "Hello ninjas".

Remember to end each PHP statement with a semicolon. This is important to avoid errors. Save the file and run it in the browser by accessing the URL path "/torts". You should see the output "Hello ninjas" displayed in the browser.

Behind the scenes, when we request the "index.php" file, the server runs the PHP code within the file. The PHP code outputs the string "Hello ninjas", which is then sent to the browser. The browser interprets the string as HTML and displays it within an HTML document.

It is important to note that if we forget to add a semicolon at the end of a PHP statement, it will result in a syntax error. Always remember to include the semicolon to indicate the end of the statement.

To embed PHP code within HTML code, simply include the PHP tags inside the HTML tags. For example, you can create an HTML template and use PHP tags to insert dynamic content into the HTML code.

We have learned how to create a folder to store our project files, create a new PHP file, write PHP code using PHP tags, and output text using the "echo" statement. We have also seen how PHP code is executed on the server and how the output is displayed in the browser.

In this didactic material, we will discuss the fundamentals of PHP and MySQL, specifically focusing on getting started with PHP and creating your first PHP file.

To begin, let's give our PHP file a title. In the body of the file, we will use an h1 tag to display the text "Hello ninjas". However, instead of hard-coding this text, we will use PHP to echo it. To do this, we need to enclose our PHP code within PHP tags by using the PHP keyword followed by a space. Then, we use the echo keyword to output the string we want to display, which in this case is "Hello ninjas". Don't forget to end the statement with a semicolon.

When we request this PHP file in the browser, it will be sent to the server, which will process the PHP code. Any PHP code within the file will be executed, and the result will be embedded in the HTML response sent back to the browser. In this case, the resulting HTML will contain the "Hello ninjas" text enclosed within the h1 tags. The browser will interpret the HTML and render it on the screen.

By combining PHP and HTML in this way, we can easily create dynamic HTML templates. Although it may not

seem necessary at first, the power of PHP lies in its ability to generate dynamic content. Instead of hard-coding static text, we can use PHP to output dynamic data such as user information or product details from a database. This allows us to create interactive webpages that can be customized based on user input or database information.

Now that we know how to create PHP files and embed PHP code within HTML templates, let's move on to the next topic: variables and constants.

EITC/WD/PMSF PHP AND MYSQL FUNDAMENTALS DIDACTIC MATERIALS
LESSON: PHP DATA STRUCTURES
TOPIC: VARIABLES AND CONSTANTS

In PHP, variables are used to store information or data that can be recalled and used later in a program. For example, a variable can be used to store a user's email address and then be called upon later to output it to the screen.

To create a variable in PHP, the syntax is to start with a dollar sign ($) followed by the variable name. The variable name must start with a letter or an underscore and cannot start with a number or special characters. After the first letter, a combination of letters (uppercase or lowercase), underscores, and numbers can be used.

There are different conventions for naming variables, such as camel case where the first letter of each word is capitalized, or using underscores to separate words. However, the naming convention is up to the developer's preference.

In PHP, strings are used to store characters, numbers, or special characters like symbols. Strings are enclosed in quotes (either single or double). For example, a string variable can be created by assigning a value to it using the equal sign (=).

To access the value of a variable, it can be echoed using the echo statement. The variable name is preceded by the dollar sign ($) within the echo statement. The value of the variable will be outputted to the screen.

Variables can also be used within HTML templates. By enclosing the PHP code within opening and closing PHP tags, the variable can be echoed within the HTML template.

In addition to strings, PHP supports other data types such as integers, floats, booleans, arrays, and objects. Each data type has its own syntax and rules for assignment.

Variables can be overridden by assigning a new value to them later in the program. The new value will replace the previous value assigned to the variable.

Comments in PHP can be added using two forward slashes (//) or a pound sign (#). Comments are used to add explanatory notes or disable certain lines of code.

In PHP, variables are used to store and manipulate data. They can be assigned values and these values can be changed throughout the execution of the program. However, there may be cases where we want to define a value that remains constant and cannot be changed. In such situations, we can use constants.

Constants in PHP are similar to variables, but their values cannot be modified once they are defined. To create a constant, we use the `define` function. The syntax for defining a constant is as follows: `define('CONSTANT_NAME', value);`

It is common practice to write the constant name in all capital letters to differentiate it from variables. This helps us identify and recognize constants when we use them in our code.

For example, let's define a constant named `Yoshi` with the value "Yoshi". We can define it using the following code: `define('YOSHI', 'Yoshi');`

To access the value of a constant, we can simply use its name without the dollar sign (`$`). For example, to output the value of the `Yoshi` constant, we can use the following code: `echo YOSHI;`

When we run the code, it will display "Yoshi" on the screen.

Unlike variables, constants cannot be overridden or changed once they are defined. If we try to redefine a constant, PHP will throw an error. For example, if we try to set the value of the `Yoshi` constant to "Mario" using the code `define('YOSHI', 'Mario');`, PHP will generate a syntax error.

Variables in PHP can store and manipulate data, while constants are used to define values that remain constant and cannot be changed. To create a constant, we use the `define` function, and to access its value, we simply use its name without the dollar sign. Constants are helpful when we want to define values that should not be modified throughout our program.

EITC/WD/PMSF PHP AND MYSQL FUNDAMENTALS DIDACTIC MATERIALS
LESSON: PHP DATA STRUCTURES
TOPIC: STRINGS

Strings are one of the fundamental data types in PHP. They are used to store sequences of characters, such as words, sentences, email addresses, and more. In PHP, strings can be enclosed in either single quotes ('') or double quotes ("").

To create a string, we simply assign a sequence of characters to a variable. For example, we can create a string variable called "string1" and set it equal to "my email is". Similarly, we can create another string variable called "string2" and set it equal to "Mario123@thenetninjacode.co.uk".

To output a string to the browser, we can use the echo statement followed by the variable name or the string itself. For example, if we echo "string1", it will display "my email is" on the screen.

Strings can contain any kind of characters, including special characters and numbers. They are simply a sequence of characters.

If we want to join two strings together, we can use a process called string concatenation. In PHP, we can concatenate two strings by using the dot operator (.) between them. For example, if we concatenate "string1" with "string2", it will result in "my email is Mario123@thenetninjacode.co.uk".

It is important to note that there can be a space between the two strings if desired, as it does not affect the concatenation process.

In addition to concatenating strings, we can also concatenate strings with variables. This is known as variable interpolation. For example, if we have a variable called "name" set to "Mario", we can concatenate it with a string using the dot operator.

There is a difference between using single quotes and double quotes for strings. When using double quotes, we can directly output variables inside the string without concatenation. This is known as string interpolation. However, this is not possible with single quotes.

To include quotes or special characters within a string, we can escape them using the backslash (\) character. For example, if we want to include the word "Wow" within quotes, we can escape the quotes by using the backslash.

Strings are used to store sequences of characters in PHP. They can be created using single or double quotes and can contain any kind of characters. String concatenation is used to join strings together, and variable interpolation allows us to include variables directly inside a string. Special characters within a string can be escaped using the backslash character.

In PHP, when working with strings, there are several important concepts to understand. One of these concepts is escaping characters. To escape a character, we use a backslash (\) before the character we want to escape. This allows the character to appear in the string when we output it. For example, if we want to include a quotation mark within a string, we would escape it by using a backslash before the quotation mark.

Another way to handle characters in strings is by using different types of quotes. For example, we can use single quotes ('') to open the string and double quotes ("") to put what we want to output in quotes. This allows us to avoid closing the string prematurely.

To access specific characters within a string, we can use square bracket notation. In PHP, strings are zero-based, meaning the first character is referenced by 0, the second by 1, and so on. To find a specific character, we can use the variable name followed by square brackets containing the index of the character we want to access. For example, if we want to find the second character in a string, we would use the index 1.

PHP also provides built-in functions for manipulating strings. One such function is `strlen()`, which returns the length of a string. To use this function, we pass the string as an argument. For example, `strlen($string)` would

return the length of `$string`.

Another useful function is `strtoupper()`, which converts a string to uppercase. Similarly, `strtolower()` converts a string to lowercase. These functions can be helpful when we need to change the case of a string for formatting or comparison purposes.

Additionally, PHP provides the `str_replace()` function, which allows us to replace certain characters or sequences of characters within a string. This function takes three arguments: the string to search for, the replacement string, and the original string. For example, `str_replace('M', 'aw', $string)` would replace all occurrences of the letter 'M' with 'aw' in the `$string`.

These are just a few examples of the many string functions available in PHP. By understanding and utilizing these functions, we can effectively manipulate and work with strings in our PHP programs.

In PHP, strings are an important data structure that allows us to work with text and manipulate it in various ways. Strings can be enclosed in either single or double quotes.

One common operation we can perform on strings is concatenation, which allows us to combine multiple strings together. We can use the concatenation operator (.) to achieve this. For example, if we have two strings, $str1 = "Hello" and $str2 = "World", we can concatenate them using $str1 . $str2, resulting in "HelloWorld".

Another useful feature of strings in PHP is variable interpolation. When using double quotes to enclose a string, we can include variables directly within the string by using the $ symbol followed by the variable name. For example, if we have a variable $name = "John", we can include it in a string like this: "Hello, $name!". This will output "Hello, John!".

In addition to concatenation and variable interpolation, we can also perform various operations and manipulations on strings using built-in PHP functions. For example, we can replace specific characters or substrings within a string using the str_replace() function. This function takes three arguments: the substring or character we want to replace, the replacement string, and the original string.

For instance, if we want to replace all occurrences of the letter 'M' with 'W' in a string, we can use the str_replace() function like this: str_replace('M', 'W', $string). This will search for all instances of 'M' in the string and replace them with 'W'.

Lastly, it's worth mentioning that we can escape characters within a string using the backslash (\) character. This allows us to include special characters or symbols that would otherwise have a different meaning in PHP. For example, if we want to include a double quote within a string enclosed in double quotes, we can escape it like this: "She said, \"Hello!\"".

To summarize, strings in PHP are a versatile data structure that allows us to work with text. We can enclose strings in single or double quotes, concatenate them, use variable interpolation, escape characters, and perform various operations on them using built-in functions.

EITC/WD/PMSF PHP AND MYSQL FUNDAMENTALS DIDACTIC MATERIALS
LESSON: PHP DATA STRUCTURES
TOPIC: NUMBERS

In this tutorial, we will discuss the different data types for numbers in PHP, namely integers and floats. Integers are whole numbers, while floats contain decimals. For example, a variable called "radius" set to 25 would be an integer, whereas a variable called "pi" set to 3.14 would be a float.

To perform mathematical operations with numbers in PHP, we have basic operators such as addition, subtraction, multiplication, and division. The symbols used for these operators are +, -, *, and /. Additionally, to find the power of a number, we use the double asterisk symbol (**), not the caret symbol (^) as mentioned earlier.

Let's consider an example where we calculate the area of a circle using the formula echo pi * radius ** 2. If we echo this out, we should see the result in the browser.

Next, we will discuss the order of operations in mathematical calculations, commonly known as BIDMAS (Brackets, Indices, Division, Multiplication, Addition, Subtraction). This order dictates the sequence in which calculations are performed. First, any calculations inside brackets are executed. Then, calculations involving indices are performed, followed by division, multiplication, addition, and subtraction.

For example, if we have the equation echo 2 * (4 + 9) / 3, the calculations would be executed as follows: first, the expression inside the brackets (4 + 9) is evaluated to 13. Then, the division operation 13 / 3 is performed, resulting in a value of approximately 4.33. Finally, the multiplication operation 2 * 4.33 is executed, yielding the final result.

Now, let's move on to the increment and decrement operators. These operators allow us to add or subtract values from numbers. For instance, if we want to add 1 to a variable called "radius," we can use either the expression "radius = radius + 1" or the shorthand operator "radius++". The double plus symbol (++) is the increment operator. Similarly, the decrement operator is represented by the double minus symbol (--).

To demonstrate this, we can echo the value of "radius" before and after using the increment operator. Initially, the value of "radius" is 25. After executing the increment operator, the value becomes 26.

Lastly, we will explore shorthand operators, which provide a concise way to perform mathematical operations. For example, instead of writing "age = age + 10" to add 10 to a variable called "age," we can use the shorthand operator "+=". This shorthand operator allows us to write "age += 10" to achieve the same result.

These shorthand operators can be used for addition, subtraction, multiplication, division, and modulus operations. They provide a more efficient and readable way to update variables.

This tutorial covered the different data types for numbers in PHP, including integers and floats. We also discussed mathematical operations, the order of operations, and various operators such as increment, decrement, and shorthand operators.

In PHP, we can perform various operations on numbers using different operators and functions. Let's start with the basic arithmetic operators. To add a number to a variable, we can use the shorthand operator '+='. For example, if we have a variable called 'age' and we want to add 10 to it, we can write 'age += 10'. This will update the value of 'age' by adding 10 to it. Similarly, if we want to subtract a number from a variable, we can use the shorthand operator '-='. For example, 'age -= 10' will subtract 10 from the current value of 'age'.

We can also perform multiplication on variables using the shorthand operator '*='. For instance, 'age *= 2' will multiply the current value of 'age' by 2. These shorthand operators allow us to perform arithmetic operations and update the value of a variable in a more concise way.

In addition to these operators, PHP provides several functions that can be used with numbers. One such function is 'floor'. The 'floor' function takes a number or a float as input and rounds it down to the nearest integer. For example, if we pass the value of 'pi' to the 'floor' function, it will return the value 3. This is because 'pi' is

approximately 3.14, and 'floor' rounds it down to 3.

On the other hand, the 'ceil' function does the opposite of 'floor'. It rounds a number up to the nearest integer. If we pass 'pi' to the 'ceil' function, it will return the value 4, as 'pi' is closer to 4 than to 3.

Finally, there is a built-in function called 'echo', which returns the value of the mathematical constant 'pi'. This function does not take any parameters and simply outputs the value of 'pi'. If we use 'echo' to display the value of 'pi', we will see a longer version of the constant.

These are just some of the basic operations and functions that can be performed on numbers in PHP. Throughout this course, we will explore more functions and ways to manipulate numbers and strings. It is important to understand these fundamentals as they form the basis for further learning.

EITC/WD/PMSF PHP AND MYSQL FUNDAMENTALS DIDACTIC MATERIALS
LESSON: PHP DATA STRUCTURES
TOPIC: ARRAYS

Arrays in PHP are a fundamental data structure that allows us to store multiple values inside a single variable. This is useful when we want to organize related data together. In PHP, there are three types of arrays: indexed arrays, associative arrays, and multi-dimensional arrays. In this didactic material, we will focus on indexed and associative arrays.

Indexed arrays are the simplest and most common type of array. To create an indexed array, we use square brackets to indicate an array and assign it to a variable. We then place the values inside the square brackets, separated by commas. For example:

```
1. $people1 = ["Shaun", "Crystal", "Riley"];
```

In this example, we have created an indexed array called `$people1` that stores three strings.

To access a specific value in an indexed array, we use the index of that element. The index starts at 0 for the first element, 1 for the second element, and so on. We use square brackets to access the index. For example:

```
1. echo $people1[1]; // Output: Crystal
```

In this example, we are accessing the second element of the `$people1` array, which is "Crystal".

Another way to create an indexed array is by using the `array` keyword followed by parentheses. For example:

```
1. $people2 = array("Ken", "Chun-Li");
```

This creates an indexed array called `$people2` with two elements.

We can also add other data types, such as numbers, to an indexed array. For example:

```
1. $ages = [20, 30, 40, 50];
```

This creates an array called `$ages` that stores four numbers.

When we want to print out the values of an array, we can use the `print_r` function. This function prints a readable version of the array. For example:

```
1. print_r($ages);
```

This will display the contents of the `$ages` array in a readable format.

Indexed arrays in PHP allow us to store multiple values inside a single variable. We can access specific values using their index, and we can use the `print_r` function to display the contents of an array.

Arrays are an essential data structure in PHP for storing multiple values in a single variable. In this lesson, we will focus on two types of arrays: indexed arrays and associative arrays.

Indexed arrays are arrays where each element is assigned a numeric index starting from 0. To access elements in an indexed array, we use square bracket notation. For example, to access the element at position 1 in the array "ages", we use the syntax "ages[1]". We can also modify elements in an indexed array by assigning a new value to a specific index.

To add a new element to the end of an indexed array, we can use either square bracket notation or the "array_push" function. In square bracket notation, we leave the brackets empty, like this: "ages[] = 60". This will add the value 60 to the end of the array. Alternatively, we can use the "array_push" function, which takes two arguments: the array we want to add a value to, and the value itself. For example, "array_push(ages, 70)" will

add the value 70 to the end of the "ages" array.

To count the number of elements in an indexed array, we can use the "count" function. For example, "count(ages)" will return the number of elements in the "ages" array.

Associative arrays are arrays where each element is assigned a specific key. The key-value pairs in associative arrays allow us to access elements using the key instead of a numeric index. To create an associative array, we use the syntax "array(key => value)". For example, "$ninjas1 = array('name' => 'Ninja')". In this example, the key is "name" and the value is "Ninja". We can access elements in an associative array using the syntax "$ninjas1['name']".

To merge two arrays together, we can use the "array_merge" function. This function takes two arrays as arguments and returns a new array that contains all the elements from both arrays.

Indexed arrays are accessed using numeric indexes, while associative arrays are accessed using keys. Indexed arrays can be modified by assigning new values to specific indexes, and new elements can be added using square bracket notation or the "array_push" function. The "count" function can be used to determine the number of elements in an array. Associative arrays use key-value pairs, and elements can be accessed using the key. The "array_merge" function allows us to combine two arrays into one.

An associative array is a data structure in PHP that allows us to associate keys with values. In this context, a key is like an arrow that points to the value it is associated with. For example, we can have an associative array called "ninjas" with keys such as "Shan", "Mario", and "Luigi", and their respective values can be "black", "orange", and "brown".

To access the value associated with a key in an associative array, we use the key within square brackets after the variable name. For instance, to access the value associated with the key "Mario" in the "ninjas" array, we would use the expression "ninjas['Mario']".

If we want to print out the value associated with a key, we can use the "echo" statement followed by the variable name and the key inside square brackets. For example, "echo $ninjas['Mario'];" would output "orange".

To print the entire associative array, we can use the "print_r" function followed by the variable name. This will display all the keys and their corresponding values in the array.

Associative arrays can also be created using the "array" function. For example, the array "ninjas2" can be created with the keys "Bowser" and "Peach" and their respective values "green" and "yellow".

To add a new key-value pair to an associative array, we can simply assign a value to a new key using square brackets. For instance, "$ninjas['Toad'] = 'pink';" would add the key-value pair "Toad" and "pink" to the "ninjas" array.

If we want to override the value associated with a key in an associative array, we can simply assign a new value to that key. For example, "$ninjas['Peach'] = 'pink';" would change the value associated with the key "Peach" to "pink".

To count the number of elements or values in an associative array, we can use the "count" function. By passing the array as an argument to the "count" function, we can obtain the number of elements in the array.

Associative arrays can also be merged using the "array_merge" function. This function takes two or more arrays as arguments and returns a new array that is the combination of all the arrays. For example, "$ninjas3 = array_merge($ninjas1, $ninjas2);" would merge the "ninjas1" and "ninjas2" arrays into a new array called "ninjas3".

To print the merged array, we can use the "print_r" function followed by the variable name. This will display all the keys and their corresponding values in the merged array.

Associative arrays in PHP allow us to associate keys with values. We can access values using keys, print the values or the entire array, add new key-value pairs, override values, count the number of elements, and merge

arrays. These concepts are fundamental to PHP programming and will be used frequently in web development projects.

EITC/WD/PMSF PHP AND MYSQL FUNDAMENTALS DIDACTIC MATERIALS
LESSON: PHP DATA STRUCTURES
TOPIC: MULTIDIMENSIONAL ARRAYS

Multi-dimensional arrays are an important concept in PHP data structures. They are essentially arrays within arrays. While we have been working with one-dimensional arrays so far, multi-dimensional arrays allow us to store more complex data structures.

To understand multi-dimensional arrays, let's create an example. Imagine we are storing an array of blog posts. Each blog post has a title, author, content, and number of likes. Instead of using a single-dimensional array, we can use a multi-dimensional array to store all this information.

To create a multi-dimensional array, we use square brackets for the outer array. Each element inside the outer array is an array itself. In our example, each blog post will be represented by an array with multiple values.

Let's create our first blog post array. We'll give it a title of "Mario Party", an author of "Mario", some content (using lorem ipsum for now), and 30 likes. This array represents the first blog post.

Next, let's create another array for a second blog post. This time, it could be called "Mario Kart Cheats", with an author of "Toad", some content, and 25 likes.

Finally, let's create a third array for another blog post. This one could be titled "Zelda Hidden Chests", with an author of "Link", some content, and 50 likes.

Now we have these different arrays as elements inside the outer array. This is what makes it a multi-dimensional array. Both of these inner arrays are indexed arrays because they don't have any key-value pairs. The outer array also doesn't have any key-value pairs.

To print the entire multi-dimensional array, we can use the print_r function. If we want to print a specific element of the array, we can access it using the index. For example, to print the second blog post, we would use the index 1.

In the simplest form, we have been using indexed arrays. However, in this case, it might make more sense to use associative arrays. Instead of relying on indices, we can use keys to represent each value. For example, we can use "title" as the key for the title of the blog post, "author" as the key for the author, and so on.

By using associative arrays, we can improve the readability and maintainability of our code. Other developers can easily understand what each value represents without having to rely on indices.

Multi-dimensional arrays are a powerful tool in PHP for storing complex data structures. By using arrays within arrays, we can represent more meaningful and organized data. Associative arrays can further enhance the readability and maintainability of our code.

In PHP, we can use multidimensional arrays to store complex data structures. A multidimensional array is an array that contains other arrays as its elements. This allows us to organize and access data in a hierarchical manner.

To create a multidimensional array, we can use associative arrays as the elements of an indexed array. Each associative array represents a block of data and contains key-value pairs. The keys are used to access specific values within the associative array.

For example, let's say we have an indexed array called "blogs" that contains three associative arrays. Each associative array represents a blog and contains keys such as "title", "author", "content", and "likes". We can access specific values within these associative arrays using square brackets and the corresponding key.

To echo the author of the second blog, we can use the following code:

```
1. echo $blogs[1]['author'];
```

This will output the author's name for the second blog.

We can also count the number of blogs in the array using the `count()` function. For example:

```
1. echo count($blogs);
```

This will output the total number of blogs in the array.

To add a new blog to the array, we can use the `[]` notation to specify that we want to add an element to the end of the array. We can then define a new associative array with the desired keys and values. For example:

```
1. $blogs[] = [
2.     'title' => 'Castle party',
3.     'author' => 'Peach',
4.     'content' => 'Lorem',
5.     'likes' => 100
6. ];
```

This will add a new blog with the title "Castle party", the author "Peach", the content "Lorem", and 100 likes to the end of the "blogs" array.

To remove an element from an array, we can use the `array_pop()` function. This function removes the last element from the array and returns it. We can assign the returned value to a variable for further use. For example:

```
1. $popped = array_pop($blogs);
2. print_r($popped);
```

This will remove the last blog from the "blogs" array and store it in the variable "$popped". We can then use the `print_r()` function to display the removed blog.

Multidimensional arrays in PHP allow us to store and access complex data structures. We can use associative arrays as elements of an indexed array to represent blocks of data. By using keys, we can access specific values within these associative arrays. We can also add and remove elements from the multidimensional array using appropriate functions.

EITC/WD/PMSF PHP AND MYSQL FUNDAMENTALS DIDACTIC MATERIALS
LESSON: PHP PROCEDURES AND FUNCTIONS
TOPIC: LOOPS

Loops are a fundamental concept in programming languages, including PHP. They allow us to execute a block of code repeatedly for a specified number of times or for each element in a collection. Loops are useful when we want to avoid writing the same code multiple times and instead, execute it efficiently.

One commonly used loop in PHP is the for loop. It consists of three statements inside parentheses: initialization, condition, and increment. The initialization statement sets a counter variable to a starting value. The condition statement defines the condition for executing the loop. As long as the condition is true, the loop will continue to execute. The increment statement updates the counter variable after each iteration. Here's an example:

```
1. for ($i = 0; $i < 5; $i++) {
2.     // Code to be executed
3.     echo "Iteration: " . $i . "<br>";
4. }
```

In this example, the loop will execute the code block five times. The counter variable `$i` starts at 0, and as long as it is less than 5, the loop will continue. After each iteration, `$i` is incremented by 1. Once `$i` becomes equal to 5, the loop terminates.

Sometimes, we may not know the exact number of times we need to execute a code block. In such cases, we can use a for loop with the count of an array or collection as the condition. This allows us to iterate through all the elements dynamically. Here's an example:

```
1. $blogs = ["Blog 1", "Blog 2", "Blog 3"];
2.
3. for ($i = 0; $i < count($blogs); $i++) {
4.     // Code to be executed
5.     echo "Blog: " . $blogs[$i] . "<br>";
6. }
```

In this example, the loop will iterate through each element in the `$blogs` array and execute the code block. The condition `$i < count($blogs)` ensures that the loop continues until `$i` is no longer less than the number of elements in the array.

Another type of loop is the foreach loop, which is useful when iterating through arrays or collections without the need for a counter variable. The foreach loop automatically handles the length of the array. Here's an example:

```
1. $blogs = ["Blog 1", "Blog 2", "Blog 3"];
2.
3. foreach ($blogs as $blog) {
4.     // Code to be executed
5.     echo "Blog: " . $blog . "<br>";
6. }
```

In this example, the loop will iterate through each element in the `$blogs` array and execute the code block. The variable `$blog` represents the current element in each iteration.

By using loops, we can efficiently execute code blocks multiple times, either for a known number of iterations or for each element in an array or collection. Loops are a powerful tool in PHP and are essential for automating repetitive tasks and processing data effectively.

Loops are an essential concept in programming that allow us to repeat a block of code multiple times. In PHP, there are different types of loops, including the for loop and the foreach loop. In this didactic material, we will explore how these loops work and provide examples to illustrate their usage.

Let's start with the for loop. The for loop is useful when we want to iterate over a specific range of values or

elements in an array. It consists of three parts: the initialization, the condition, and the increment.

Here is the syntax of a for loop:

```
for (initialization; condition; increment) {
    // code to be executed
}
```

In the initialization part, we declare and initialize a counter variable. This variable will keep track of the current iteration. The condition part determines whether the loop should continue or stop based on a specific condition. If the condition evaluates to true, the loop will continue; otherwise, it will terminate. The increment part updates the counter variable after each iteration.

Now let's see an example of a for loop in action. Suppose we have an array called "ninjas" with three elements: "Shaun", "Ryu", and "Yoshi". We want to output each element of the array on a separate line.

```
for ($i = 0; $i < count($ninjas); $i++) {
    echo $ninjas[$i] . "<br>";
}
```

In this example, we initialize the counter variable `$i` to 0. The condition is `$i < count($ninjas)`, which means the loop will continue as long as `$i` is less than the number of elements in the array. After each iteration, we increment `$i` by 1. Inside the loop, we use `$ninjas[$i]` to access the current element of the array and echo it out with a line break.

Another type of loop in PHP is the foreach loop. The foreach loop is specifically designed to iterate over the elements of an array. It simplifies the process of accessing each element without the need for a counter variable.

Here is the syntax of a foreach loop:

```
foreach ($array as $value) {
    // code to be executed
}
```

In the foreach loop, we specify the array we want to iterate over, followed by the keyword "as" and a variable name that will represent each individual element of the array during each iteration. Inside the loop, we can use this variable to perform operations on the current element.

Let's rewrite the previous example using a foreach loop:

```
foreach ($ninjas as $ninja) {
    echo $ninja . "<br>";
}
```

In this example, we iterate over the elements of the array `$ninjas`, and during each iteration, the current element is assigned to the variable `$ninja`. We then echo out `$ninja` followed by a line break.

Both the for loop and the foreach loop can be used to achieve the same result. However, the foreach loop is often preferred when iterating over arrays, as it simplifies the code and makes it more readable.

To summarize, loops are an essential part of programming that allow us to repeat code blocks multiple times. In PHP, we have different types of loops, including the for loop and the foreach loop. The for loop is useful when we need to iterate over a specific range of values, while the foreach loop is specifically designed for iterating over elements in an array.

In PHP, we can use procedures and functions to organize our code and make it more efficient and reusable. In this material, we will focus on loops, specifically the for each loop and the while loop.

The for each loop is used to iterate through arrays and perform a certain action for each element. It is

particularly useful when working with multi-dimensional arrays. To demonstrate this, let's consider an example where we have an array of products. Each product is represented by an associative array with a name and a price. We can use the for each loop to cycle through each product and echo its name and price.

First, we create a multi-dimensional array where each element represents a product. Inside this array, we have several associative arrays, each containing the name and price variables for a single product. The surrounding array is an indexed array.

To use the for each loop, we write the keyword "foreach" followed by the array we want to iterate through. In our case, it is the products array. We also define a variable that will represent each individual product as we cycle through the array. We can name this variable whatever we want, but it is common to use the singular form of the array name. Inside the loop, we can access the individual product using the variable we defined. We use square bracket notation to access the name and price of each product.

Finally, we concatenate the product name and price using the echo statement. We separate the name and price with a hyphen and a space. After each iteration, we echo a line break tag to move to a new line.

Now, let's move on to the while loop. The while loop is similar to other types of loops but is used when we want to cycle through a block of code as long as a certain condition is true. In our example, we will use the while loop to cycle through the products array and echo the name of each product.

To use the while loop, we write the keyword "while" followed by a condition inside parentheses. In our case, the condition is that a local variable "I" is less than the count of elements in the products array. This ensures that we cycle through the code as long as "I" is less than the number of products.

Inside the loop, we echo the name of the product using the products array and the "I" variable. We also echo a line break tag to move to a new line after each iteration.

However, before running the code, we need to initialize the "I" variable outside the loop. In our case, we set it to 0. Additionally, after each iteration, we increment the "I" variable by 1 using the "I++" notation. This ensures that the loop eventually stops when "I" is no longer less than the count of products.

It is important to note that if we forget to initialize the "I" variable or increment it inside the loop, we may end up with an infinite loop that can crash our browser.

The for each loop is used to cycle through arrays, while the while loop is used to cycle through a block of code as long as a certain condition is true. Both loops are valuable tools in PHP for performing repetitive tasks efficiently.

In PHP, loops are essential for iterating through arrays or performing repetitive tasks. In this lesson, we will focus on using loops to output HTML templates for each item in an array.

To begin, let's initialize an array called "products" and populate it with some data. We will use this array to demonstrate how to loop through the products and generate HTML templates for each item.

```
$products = [
    ['name' => 'Product 1', 'price' => 10],
    ['name' => 'Product 2', 'price' => 20],
    ['name' => 'Product 3', 'price' => 30],
];
```

Now, let's create an HTML template that will display the product names and prices. We will use the "foreach" loop to cycle through each product in the array and generate the necessary HTML code.

```
<h1>Products</h1>
<ul>
    <?php foreach ($products as $product): ?>
        <li>
            <h3><?php echo $product['name']; ?></h3>
            <p><?php echo '£' . $product['price']; ?></p>
```

```
        </li>
    <?php endforeach; ?>
</ul>
```

In the code above, we start by outputting an `<h1>` tag with the title "Products". Then, we open a `<ul>` tag to create an unordered list. Inside the list, we use the "foreach" loop to iterate through each product in the "products" array.

For each product, we output an `<li>` tag, which contains an `<h3>` tag with the product name and a `<p>` tag with the product price. The product name and price are retrieved from the current item in the loop using the syntax `$product['name']` and `$product['price']`.

Finally, we close the loop using the "endforeach" keyword and close the `<ul>` tag.

When we run this code, it will generate an HTML template for each product in the array, displaying the product name and price. This technique allows us to dynamically generate HTML content based on the data in the array.

EITC/WD/PMSF PHP AND MYSQL FUNDAMENTALS DIDACTIC MATERIALS
LESSON: PHP PROCEDURES AND FUNCTIONS
TOPIC: BOOLEANS AND COMPARISONS

Boolean data type is a new data type in PHP that we haven't discussed yet. In the previous material, we looked at boolean comparisons without even realizing it when we discussed loops. In PHP, a condition that we evaluate in a loop can be either true or false. If the condition is true, the code inside the loop will be executed. If the condition is false, the code will not be executed. True and false are two special values in PHP and they are their own type, called booleans. We use booleans to execute conditional code. When we perform a comparison, it will return a boolean value. If the comparison is true, the code will run. If the comparison is false, the code will not run.

When we echo a boolean to the browser, we don't see the words "true" or "false". Instead, the boolean values are converted into strings. When a boolean is true, it is converted into a string of "1". When a boolean is false, it is converted into an empty string. This is because everything that is output to the browser needs to be a string. The reason it uses "1" and an empty string for true and false is because in loose comparison, "1" is a truth value that evaluates to true, and an empty string evaluates to false. Therefore, when we echo a boolean, it will be converted into either "1" or an empty string.

Let's now move on to doing some comparisons using booleans. We will start with comparisons between numbers. For example, let's compare if 5 is less than 10. This comparison should evaluate to true, so when we echo it to the screen, we should see "1". And indeed, when we preview it in the browser, we see "1". Next, let's compare if 5 is greater than 10. This comparison is false, so we shouldn't see anything on the screen. And indeed, when we comment it out and preview, we don't see anything. Now, let's compare if 5 is equal to 10. Notice that we are using double equal signs here for a loose comparison. This comparison is false, so we shouldn't see anything on the screen. And indeed, when we comment it out and preview, we don't see anything.

We can also compare if two values are equal using double equal signs. For example, let's compare if 10 is equal to 10. This comparison is true, so when we echo it to the screen, we should see "1". And indeed, when we preview it in the browser, we see "1". Lastly, we can use the negation operator to check if a value is not equal to another. For example, let's check if 5 is not equal to 10. This comparison is true, so when we echo it to the screen, we should see "1". And indeed, when we preview it in the browser, we see "1".

Booleans are a special type in PHP used for executing conditional code. They can have two values: true or false. When we echo a boolean to the browser, it gets converted into a string, either "1" for true or an empty string for false. We can perform comparisons using booleans to check if a condition is true or false. These comparisons can be done between numbers or any other values. We can use double equal signs for loose comparisons to check if two values are equal. We can also use the negation operator to check if a value is not equal to another.

In PHP, we can use comparison operators to compare values and determine if they are equal or not. To check for equality, we use the double equals sign (==). For example, if we want to see if 5 is equal to 10, we would write "5 == 10". In this case, the statement is false, so the result would be 0.

To check if two values are not equal, we use the exclamation mark followed by the equal sign (!=). For example, if we want to see if 5 is not equal to 10, we would write "5 != 10". Since this statement is true, the result would be 1.

We can also compare numbers using other comparison operators, such as less than (<), less than or equal to (<=), greater than (>), and greater than or equal to (>=). For example, if we want to check if 5 is less than or equal to 5, we would write "5 <= 5". Since this statement is true, the result would be 1.

Similarly, we can compare strings using comparison operators. In PHP, strings are compared based on their alphabetical order. For example, if we want to check if the string "Sean" is less than the string "Yoshi", we would write "'Sean' < 'Yoshi'". Since "S" comes before "Y" in the alphabet, this statement is true, and the result would be 1.

We can also use the greater than (>) operator to compare strings. For example, if we want to check if "Sean" is greater than "Yoshi", we would write "'Sean' > 'Yoshi'". Since "S" comes after "Y" in the alphabet, this statement

is false, and the result would be 0.

It's important to note that when comparing strings, uppercase letters are considered to be less than lowercase letters. For example, "Shan" is considered greater than "shan" because "S" is less than "s" in terms of alphabetical order.

In addition to the double equals (==) and exclamation mark followed by the equal sign (!=) for loose comparison, PHP also provides strict comparison using the triple equals sign (===). Strict comparison takes into account the type of data being compared. For example, if we compare the string "v" with the number 5 using loose comparison ("'v' == 5"), the result would be true because loose comparison doesn't consider data types. However, if we use strict comparison ("'v' === 5"), the result would be false because the data types are different (string vs. number).

When outputting booleans to the browser, PHP converts them into strings. True is represented as "1" and false as an empty string. However, in loose comparison, the string "1" is considered equal to true. For example, if we compare true with the string "1" using loose comparison ("true == '1'"), the result would be true.

In general, it is recommended to use strict comparison when comparing values to ensure accurate and consistent results.

In PHP, we can perform various comparisons using booleans. Booleans are a fundamental data type that represents true or false values. In this lesson, we will explore different comparison operators and their usage in PHP.

One common comparison operator is the equal to operator (==). This operator checks if two values are equal. For example, if we compare the boolean value true with the string "true", PHP will consider them equal. This is because PHP performs a loose comparison, where it tries to convert the values to a common type before comparing them.

Another comparison operator is the strict equal to operator (===). This operator not only checks if the values are equal but also ensures that their types are the same. For example, if we compare the boolean value true with the string "true" using the strict equal to operator, PHP will consider them not equal because their types are different.

We can also use comparison operators like less than (<) and greater than (>). These operators are commonly used when comparing numerical values. For example, if we compare the number 5 with the number 10 using the less than operator, PHP will return true because 5 is indeed less than 10.

In addition to these comparison operators, we can also perform comparisons using the empty string. In PHP, an empty string is considered false. So, if we compare the boolean value false with an empty string using the equal to operator, PHP will consider them equal.

It is important to note that comparisons in PHP can be a bit tricky at times. It is always recommended to use the appropriate comparison operator based on your specific requirements. Understanding how booleans and comparisons work is crucial for writing effective PHP code.

In the next lesson, we will delve into conditional statements, where we will use booleans and comparisons to make decisions in our code.

EITC/WD/PMSF PHP AND MYSQL FUNDAMENTALS DIDACTIC MATERIALS
LESSON: PHP PROCEDURES AND FUNCTIONS
TOPIC: CONDITIONAL STATEMENTS

Conditional statements are an essential part of programming languages. They allow us to check certain conditions and perform different actions based on the result. A common example of a conditional statement is checking if a user is logged in. If the user is logged in, we may want to display one navigation, and if they are not logged in, we may want to display a different navigation.

Another example is when we have a list of products and we want to cycle through them. For each product, we can check the price and decide whether to display it or not based on a certain condition. These conditional statements act like forks in the code, determining which path the code will take.

To work with conditional statements in PHP, we can use boolean expressions and comparisons that we learned in the previous lesson. Let's start by creating a simple variable called "price" and setting it to the value 20. We can then perform a check on this variable using an "if" statement. For example, we can check if the price is less than 30. If this condition is true, we can execute a block of code, such as displaying a message.

We can also add an "else" clause to handle the case when the condition is not met. In this case, we can execute a different block of code. For example, we can display a different message. This allows us to handle both scenarios based on the condition.

Additionally, we can include multiple "else if" clauses to evaluate additional conditions. These clauses are checked in order, and the first one that evaluates to true will execute its corresponding block of code. If none of the conditions are true, we can have a final "else" clause that acts as a catch-all and executes its block of code.

It's important to note that conditional statements are not limited to checking fixed values. They can be used to evaluate variables that may have different values, such as when working with data from a database. This allows us to dynamically handle different scenarios based on the values we receive.

Conditional statements are fundamental in programming languages. They allow us to check conditions and perform different actions based on the results. By using boolean expressions and comparisons, we can create if statements, else clauses, and else if clauses to handle various scenarios. This flexibility enables us to build dynamic and responsive applications.

A multi-dimensional array is used, where the arrays inside are all associative arrays. Each array contains a "name" key and a "price" key, representing the product name and price respectively. To cycle through each element of the array, a foreach loop is used. Inside the loop, a check is performed to see if the product price is less than a certain value. If it is, the product name is outputted.

To implement this, the foreach loop is written as follows: "foreach ($products as $product)". Within the loop, an if statement is used to check if the current product's price is less than 15. If it is, the product name is echoed along with the price and a line break.

To display the correct output, the code is saved and previewed. The result shows the product names "green shell", "gold coin", and "banana skin", which have prices less than 15.

The concept of multiple conditions within an if statement is then introduced. In this case, the price should be less than 15 and greater than 2. This is achieved by using the "&&" operator to check both conditions simultaneously. As a result, the product "banana skin" is no longer displayed.

To demonstrate another example, the conditions are modified. This time, the price needs to be either over 20 or less than 10. The "||" operator, representing logical OR, is used to check for either condition. The products "gold coin", "lightning bolt", and "banana skin" meet these criteria and are displayed.

The usage of if statements within the template itself is then explained. In this case, different content is displayed based on whether a condition is true or false. An example is provided where a div tag is used, and within it, a foreach loop is used to cycle through the products. Inside the loop, an if statement is used to check a

condition. If the condition is true, a certain content is displayed, and if it is false, a different content is displayed.

To summarize, this didactic material covers the usage of if statements and conditional statements in PHP. It explains how to check multiple conditions using logical operators and how to output different content based on the condition's result. It also demonstrates the usage of if statements within HTML templates to dynamically display content based on conditions.

In PHP, we can use conditional statements to check certain conditions and execute different code blocks based on the result of the condition. One of the conditional statements we can use is the "if" statement.

The syntax of the "if" statement in PHP is as follows:

```
if (condition) {
    // code block to be executed if the condition is true
}
```

In the given code snippet, the "if" statement is used to check if the price of a product is greater than 15. If the condition is true, the code block inside the "if" statement will be executed.

To output the price of the product, we use the PHP echo statement. The echo statement is used to output text or variables. In this case, we are outputting the price of the product.

Additionally, the code snippet also demonstrates the usage of the PHP echo statement to output the name of the product. This is done inside a loop that iterates through all the products.

Here is the modified code snippet:

```
<?php
// Iterate through all the products
foreach ($products as $product) {
    // Check if the price of the product is greater than 15
    if ($product['price'] > 15) {
        // Output an li tag for the product
        echo '<li>' . $product['name'] . '</li>';
    }
}
?>
```

In this code snippet, we are using the foreach loop to iterate through each product in the $products array. Inside the loop, we check if the price of the product is greater than 15 using the "if" statement. If the condition is true, we output an li tag with the name of the product.

By using conditional statements like the "if" statement, we can dynamically control the content that is displayed on a webpage based on certain conditions.

EITC/WD/PMSF PHP AND MYSQL FUNDAMENTALS DIDACTIC MATERIALS
LESSON: PHP PROCEDURES AND FUNCTIONS
TOPIC: CONTINUE AND BREAK

In PHP, there are two keywords that are commonly used in loops: break and continue. These keywords allow us to control the flow of the loop by either breaking out of the loop completely or skipping to the next iteration.

The break keyword is used to break out of a loop at any point, regardless of where we are inside the loop. For example, if we are cycling through a list of products and we come across the keyword break, PHP will immediately exit the loop and continue with the code outside of the loop. This means that any remaining iterations of the loop will be skipped.

To illustrate this, let's consider a simple example. We have a loop that iterates through a list of products, and we want to break out of the loop when we find a specific product called "lightning bolts". Inside the loop, we use an if statement to check if the current product's name is equal to "lightning bolts". If it is, we use the break keyword to exit the loop. If it is not, we continue with the loop and echo out the product name.

Another keyword, continue, is used to skip the rest of the code in the current iteration and move on to the next iteration of the loop. It is similar to break in that it controls the flow of the loop, but it does not exit the loop completely. Instead, it goes back to the beginning of the loop and starts the next iteration.

In our example, let's say we want to skip any products with a price greater than 15. We use another if statement inside the loop to check if the product's price is greater than 15. If it is, we use the continue keyword to skip the rest of the code in that iteration and move on to the next product. If the price is not greater than 15, we continue with the code and echo out the product name.

By using these keywords, we can have more control over the behavior of our loops and make our code more efficient. The break keyword allows us to exit a loop prematurely, while the continue keyword allows us to skip certain iterations based on specific conditions.

Understanding how to use break and continue in PHP loops is essential for effective programming and can help improve the efficiency of your code.

EITC/WD/PMSF PHP AND MYSQL FUNDAMENTALS DIDACTIC MATERIALS
LESSON: PHP PROCEDURES AND FUNCTIONS
TOPIC: FUNCTIONS

Functions in PHP are blocks of code that can be executed to perform specific tasks. They can be thought of as black boxes, where input goes in and output comes out. Inside the black box, there is a block of code that runs to produce the desired output. Functions can be called or invoked multiple times, allowing us to reuse the code without having to write it again.

In PHP, there are built-in functions that we can use, such as the "stringToUpper" function, which takes a lowercase string as input and returns an uppercase string. These built-in functions save us time and effort by providing pre-defined functionality. However, we can also create our own functions to perform custom tasks.

To create a function, we use the "function" keyword followed by the function name. The function name should be chosen appropriately and can consist of multiple words, either using camel case or underscores. After the function name, we use parentheses to define any arguments that the function expects. Arguments are values that we pass to the function when we call it.

Inside the function, we write the code that we want to execute when the function is called. This code is enclosed within curly braces. For example, we can use the "echo" statement to output a greeting message like "Good morning, Yoshi".

After declaring the function, we need to call or invoke it to execute the code inside. We do this by simply stating the function name followed by parentheses. This tells PHP to find the function and run it.

We can also pass arguments to our own functions, just like we do with built-in functions. To do this, we specify the expected arguments in the function declaration, and then use those arguments within the function code. For example, if we want to pass a name to the function, we can declare a parameter in the function declaration, such as "$name". When we call the function, we provide the value for the parameter.

By using arguments and parameters, we can make our functions more flexible and reusable. For example, if we pass the name "Mario" as an argument, the function will use that value and output "Good morning, Mario".

Functions in PHP allow us to encapsulate blocks of code, give them a name, and call them whenever we need to perform a specific task. They enhance code organization, reusability, and help make our programs more efficient.

In PHP, functions are reusable blocks of code that perform specific tasks. They can accept arguments, which are values passed into the function, and return a value.

When defining a function, we can specify parameters that represent the expected arguments. For example, we can create a function called "sayHello" that takes in a parameter called "name":

```
1. function sayHello($name) {
2.     echo "Good morning, $name!";
3. }
```

To call this function and pass in a value for the "name" parameter, we simply use the function name followed by parentheses, with the value inside:

```
1. sayHello("Mario");
```

This will output "Good morning, Mario!".

If we don't pass in a value for the "name" parameter, we can set a default value. In the example below, we set the default value to "Sean":

```
1. function sayHello($name = "Sean") {
```

```
    echo "Good morning, $name!";
}
```

Now, if we call the function without passing in a value:

```
sayHello();
```

It will output "Good morning, Sean!".

We can also create functions that accept more complex parameters. For example, let's create a function called "formatProduct" that takes in a parameter called "product". This parameter can be an associative array with a "name" and a "price" value:

```
function formatProduct($product) {
    echo "The product {$product['name']} costs {$product['price']} pounds to buy.<br>";
}
```

To call this function and pass in a product, we create an associative array with the "name" and "price" values:

```
$product = [
    'name' => 'Gold Star',
    'price' => 20
];

formatProduct($product);
```

This will output "The product Gold Star costs 20 pounds to buy."

In some cases, we may want a function to return a value instead of echoing it directly. To do this, we use the "return" keyword. For example:

```
function formatProduct($product) {
    return "The product {$product['name']} costs {$product['price']} pounds to buy.";
}
```

To store the returned value in a variable, we can use the assignment operator:

```
$formatted = formatProduct($product);
```

Now, the value returned by the function will be stored in the variable "$formatted".

Functions in PHP allow us to encapsulate blocks of code that perform specific tasks. They can accept arguments, which are values passed into the function, and return a value. We can also set default values for parameters and create functions that accept more complex parameters, such as associative arrays.

In PHP, functions are a fundamental concept that allows us to encapsulate reusable blocks of code. They are defined using the keyword "function" followed by the function name and a pair of parentheses. Inside the parentheses, we can specify any parameters that the function should accept. These parameters act as placeholders for values that we can pass into the function when we call it.

To demonstrate the concept of functions, let's consider an example. Suppose we have a function called "sayHello" that takes in a parameter called "name". Inside the function, we can use the "echo" statement to output a greeting message that includes the provided name. Here's how the function would look like:

```
function sayHello($name) {
    echo "Hello, $name!";
}
```

To call this function and pass in a value for the "name" parameter, we simply write the function name followed

by parentheses, and inside the parentheses, we provide the desired value. For example:

```
1. sayHello("John");
```

When we run this code, it will output the message "Hello, John!".

Now, let's explore the concept of returning values from functions. Sometimes, instead of directly echoing something out, we might want to perform some calculations or operations and store the result in a variable for later use. To achieve this, we can use the "return" statement inside the function. The "return" statement allows us to specify the value that should be returned when the function is called.

Consider the following example of a function called "addNumbers" that takes in two parameters, "num1" and "num2", and returns their sum:

```
1. function addNumbers($num1, $num2) {
2.     $sum = $num1 + $num2;
3.     return $sum;
4. }
```

To call this function and retrieve the result, we can assign the function call to a variable. For example:

```
1. $result = addNumbers(5, 3);
2. echo $result; // Output: 8
```

In this case, the function call "addNumbers(5, 3)" will return the sum of 5 and 3, which is 8. We then assign this result to the variable "$result" and echo it out, resulting in the output "8".

Additionally, functions can accept multiple arguments by specifying them inside the parentheses, separated by commas. We can also provide default values for these arguments, which will be used if no value is explicitly passed when calling the function. This allows for greater flexibility and customization.

Let's look at an example of a function called "sayGoodbye" that accepts two parameters: "name" and "time". The "time" parameter has a default value of "morning". Inside the function, we output a farewell message that includes the provided name and time. Here's how the function would be defined:

```
1. function sayGoodbye($name, $time = "morning") {
2.     echo "Good $time, $name!";
3. }
```

When calling this function, we can pass in values for both parameters, overriding the default value for "time". For instance:

```
1. sayGoodbye("Yoshi", "night");
```

This will output the message "Good night, Yoshi!".

However, if we call the function without providing a value for "time", it will use the default value of "morning":

```
1. sayGoodbye("Sean");
```

In this case, the output will be "Good morning, Sean!".

To summarize, functions in PHP allow us to encapsulate reusable blocks of code. They can accept parameters, which act as placeholders for values that can be passed into the function when calling it. Functions can also return values using the "return" statement. Multiple arguments can be accepted, and default values can be specified for greater flexibility.

EITC/WD/PMSF PHP AND MYSQL FUNDAMENTALS DIDACTIC MATERIALS
LESSON: ADVANCING IN PHP
TOPIC: VARIABLE SCOPE

Variable scope is an important concept in PHP that determines where a variable can be accessed within a program. Variables can have either local scope or global scope.

Local variables have scope limited to the block of code in which they are declared. For example, if we create a function called "myfunc" and declare a variable called "price" inside it, we can only use that variable within the function. If we try to access it outside the function, we will encounter an error.

```
function myfunc() {
    $price = 10;
    echo $price;
}

myfunc(); // Output: 10

echo $price; // Error: Undefined variable
```

Global variables, on the other hand, can be accessed from anywhere within the program, including inside functions. To declare a global variable, it needs to be defined outside of any function. Inside the function, we can use the "global" keyword to access the global variable.

```
$name = "Mario";

function sayHello() {
    global $name;
    echo "Hello " . $name;
}

sayHello(); // Output: Hello Mario

echo $name; // Output: Mario
```

If we modify the value of the global variable inside a function, it will also change outside of the function.

```
$name = "Mario";

function sayHello() {
    global $name;
    $name = "Yoshi";
    echo "Hello " . $name;
}

sayHello(); // Output: Hello Yoshi

echo $name; // Output: Yoshi
```

It's important to be aware of variable scope when working with PHP to avoid unexpected behavior and errors. Local variables are limited to the block of code in which they are declared, while global variables can be accessed from anywhere within the program.

In PHP, variable scope refers to the visibility and accessibility of variables within different parts of a program. Understanding variable scope is important for writing efficient and error-free code. In this lesson, we will explore the concept of variable scope and how it affects the behavior of variables in PHP.

When we declare a variable outside of any functions or classes, it is considered a global variable. Global variables can be accessed and modified from anywhere in the program. However, when we declare a variable inside a function, it becomes a local variable and can only be accessed within that function.

In the provided example, we have a function that takes a parameter called "name". When we pass a value to this function, it creates a local variable with the same name and assigns the passed value to it. This local variable only exists within the function and does not affect any global variables with the same name.

If we try to update the value of the "name" variable inside the function, it does not affect the global variable. This is because the local variable takes precedence over the global variable within the function's scope. So, even though we update the local variable to "Wario", the global variable remains unchanged.

To update the global variable from within the function, we can use the "global" keyword. By declaring "global $name" inside the function, we can access and modify the global variable directly. This allows us to override the global variable with a new value.

Alternatively, we can pass the variable by reference by adding an ampersand (&) before the parameter name. This means that any changes made to the parameter inside the function will also affect the variable passed in from outside. In the example, when we pass the variable by reference, updating its value inside the function also updates the global variable.

Understanding variable scope and how to properly use global variables and function parameters is essential for writing clean and maintainable code. It helps prevent naming conflicts and ensures that variables are used correctly within different parts of a program.

Variable scope in PHP determines the visibility and accessibility of variables within different parts of a program. Local variables are limited to the scope of the function they are declared in and do not affect global variables with the same name. To update global variables from within a function, we can use the "global" keyword or pass the variable by reference. By understanding variable scope, we can write more efficient and reliable PHP code.

EITC/WD/PMSF PHP AND MYSQL FUNDAMENTALS DIDACTIC MATERIALS
LESSON: ADVANCING IN PHP
TOPIC: INCLUDE AND REQUIRE

In PHP, there are two built-in functions called include and require that allow us to include the contents of one file into another file. These functions are useful for modularizing our code and reducing code repetition.

The include function works by including the specified file into the current file. It allows the code to continue running even if there is an error while including the file. On the other hand, the require function also includes the specified file, but if there is an error, it will result in a fatal error and stop the code execution.

To use these functions, we need to provide the path to the file we want to include as a string. If the file is in the same directory as the current file, we can simply provide the file name. For example, if the file we want to include is called "ninjas.php" and it is in the same directory, we can use the following code:

```
1. include 'ninjas.php';
```

Alternatively, we can also use the functions without parentheses, like this:

```
1. include 'ninjas.php';
```

Both forms of the functions work the same way.

When including a file, any code within that file will be executed in the current file. This allows us to reuse code and avoid duplicating it across multiple files. For example, if we have a block of code that appears on multiple pages of a website, we can create a separate file for that code and include it in each page where we want it to appear.

Here's an example of how we can include a file called "content.php" that contains a div with some content:

```
1. include 'content.php';
```

By including this file, the div with the content will be added to the current file.

The power of include and require functions becomes evident when we need to update the included code. Instead of updating it in multiple places, we only need to update it once in the external file, and the changes will be reflected in all the files where it is included.

The include and require functions in PHP allow us to include the contents of one file into another file. The include function continues running even if there is an error, while the require function results in a fatal error if there is an error. These functions are useful for modularizing code and reducing code repetition.

In this didactic material, we will explore the process of creating a navbar and footer templates for the Ninja Pizza project in the realm of web development, specifically focusing on PHP and MySQL fundamentals, with an emphasis on advancing in PHP through the use of include and require.

To begin, let's delve into the concept of templates. Templates serve as reusable components that allow for consistent design and structure across multiple web pages. By separating the layout from the content, templates enhance code organization and simplify maintenance. In our case, we will be creating a navbar and footer template.

The navbar template is a fundamental element of any website, as it provides navigation options for users. It typically contains links to various pages or sections within the website. By creating a navbar template, we can easily include it in multiple pages, ensuring a consistent and user-friendly experience throughout the Ninja Pizza project.

Similarly, the footer template is located at the bottom of each webpage and often includes copyright information, contact details, or additional navigation links. By creating a footer template, we can avoid

duplicating code and ensure that any changes or updates made to the footer are automatically reflected across all web pages.

To accomplish this, we will utilize the include and require statements in PHP. These statements allow us to import code from other files into our current PHP file, effectively merging the content of the included file with the current file. The key difference between include and require is that require will produce a fatal error and halt the script execution if the specified file is not found, whereas include will only generate a warning and continue execution.

To create our navbar and footer templates, we will follow these steps:

1. Create a new PHP file for the navbar template, for example, "navbar.php". Inside this file, write the HTML and CSS code for the navbar, including any necessary PHP code for dynamic elements such as active page highlighting.
2. Save the navbar.php file in a designated directory, ensuring it is easily accessible for inclusion in other PHP files.
3. In the PHP files where you want to include the navbar, use the include or require statement to import the navbar.php file. For example: `include 'path/to/navbar.php';`
4. Repeat the above steps for the footer template, creating a new PHP file named "footer.php" and including it in the desired PHP files using the include or require statement.

By utilizing the include and require statements, we can easily incorporate the navbar and footer templates into our Ninja Pizza project, promoting code reusability and maintaining a consistent design across web pages.

This didactic material has covered the creation of navbar and footer templates for the Ninja Pizza project, focusing on PHP and MySQL fundamentals, with an emphasis on advancing in PHP through the use of include and require statements. By implementing these templates, we can enhance code organization, promote consistency, and simplify maintenance across the project.

EITC/WD/PMSF PHP AND MYSQL FUNDAMENTALS DIDACTIC MATERIALS
LESSON: ADVANCING IN PHP
TOPIC: PROJECT HEADER AND FOOTER

In this lesson, we will learn how to create templates for a web development project using PHP and MySQL. Specifically, we will focus on creating a header and footer that will be included in multiple pages of the project.

To start, we will use the concepts of "require" and "include" that we learned in the previous lesson. Instead of writing the header and footer code repeatedly for each page, we will create separate template files for them. These template files will be included in the different pages when needed.

First, we need to create the header and footer files in a new folder called "templates". Inside this folder, we will create two files: "header.php" and "footer.php".

It's important to note that this series primarily focuses on PHP and not CSS styling. Therefore, we will be using a third-party library called Materialized CSS for styling purposes. Materialized CSS is similar to Bootstrap but follows Google's design philosophy.

In the header.php file, we will include the HTML code that will be common to all pages. This includes the opening "head" tag and the opening "body" tag. The closing "body" tag will not be included in the header file because it will be placed in the footer file. Additionally, we will set the title of the page to "Ninja Pizza" and include a link to the Materialized CSS library.

In the body of the header.php file, we will add the necessary classes from Materialized CSS to style our templates. For example, we can set the background color of the body to gray and lighten it by four shades using the "lighten" class.

To include the header.php file in our HTML pages, we will use the "include" function in PHP. By adding the appropriate PHP tags and using the "include" function, we can include the header.php file in our HTML pages.

Next, we will create a navigation bar inside the body of the HTML page. We will give the "nav" tag a class of "white" to set the background color to white and remove the default drop shadow using the "z-depth-0" class.

Inside the navigation bar, we will create a container using a "div" tag with a class of "container". This class ensures that the content remains within a central column rather than spanning the entire width of the page.

Within the container, we will add a title for the navigation bar using an anchor tag. For now, the "href" attribute will be set to "#" so that it doesn't navigate anywhere. We will give this anchor tag a class of "brand" to style it accordingly.

By following these steps, we can create a header and footer template for our web development project using PHP and MySQL. These templates can be included in multiple pages, saving us from writing repetitive code.

To create a consistent header and footer for our web pages, we can use PHP's include function. This allows us to reuse code and make updates in one place, which will reflect across all pages.

Let's start with the header. We want to create a navbar with a logo and a title. We'll use Materialize classes for styling. First, we'll create a div with the class "brand-logo" and "brand-text". Inside this div, we'll add an anchor tag with the title "Ninja Pizza". We'll position the navbar on the left using Materialize styles.

Next, we'll create a ul element with the id "nav-mobile" and the class "right". This ul will contain our navigation links. Each link will have its own li tag. For now, we'll only have one link, which is to add a new pizza. We'll use an anchor tag with a class of "btn" to make it look like a button. We'll also add our custom classes "brand-text" and "brand".

To style the navbar, we'll create two custom classes, "brand" and "brand-text". The "brand" class will have a background color, and the "brand-text" class will have a text color.

Now, let's move on to the footer. We'll include it at the bottom of every page using the include function. In the footer, we'll have a closing body tag and a footer tag. We'll give the footer tag a class of "section" to space things out. Inside the footer, we'll add a div with the class "center" and "gray-text". Inside this div, we'll put the copyright information, such as "Copyright 2019 Ninja Pizzas".

By including the header and footer using the include function, we can easily update them in one place and have them appear on every page. This saves us from having to rewrite or copy and paste the code on multiple pages.

We have created a reusable header and footer for our web pages. The header includes a navbar with a logo and title, and the footer includes copyright information. We have used PHP's include function to include the header and footer on each page, allowing for easy updates and consistency across the site.

In the previous lessons, we have made significant progress in creating our project. In this lesson, we will delve into forms in PHP and explore the GET and POST methods.

Forms play a crucial role in web development as they allow users to input data and interact with websites. PHP provides us with powerful tools to handle form submissions and process the data received.

The GET and POST methods are two common ways to send data from a form to the server. The main difference between them lies in how the data is transmitted.

When using the GET method, the form data is appended to the URL as query parameters. This means that the data is visible in the URL itself. GET requests are commonly used for retrieving data from the server, such as searching for specific information.

On the other hand, the POST method sends the form data in the body of the HTTP request. This data is not visible in the URL, making it more secure for sensitive information. POST requests are typically used when submitting data that should not be publicly accessible, such as login credentials or credit card details.

To handle form submissions in PHP, we can access the data sent through the GET or POST methods using the `$_GET` and `$_POST` superglobal arrays, respectively. These arrays contain key-value pairs, where the keys correspond to the names of the form fields.

For example, if we have a form with an input field named "username", we can access the value entered by the user using `$_POST['username']` or `$_GET['username']`, depending on the method used.

Once we have obtained the form data, we can perform various operations on it, such as validation, sanitization, and database manipulation. This allows us to create dynamic and interactive web applications that respond to user input.

In the next lesson, we will explore how to handle form submissions in PHP and demonstrate practical examples of working with the GET and POST methods.

EITC/WD/PMSF PHP AND MYSQL FUNDAMENTALS DIDACTIC MATERIALS
LESSON: FORMS IN PHP
TOPIC: WORKING WITHS FORMS IN PHP

In web development, forms are an essential component for collecting user input. In PHP, forms can be created to gather data from users and send it to the server for further processing. There are two main methods for sending data from the client to the server: GET and POST.

GET requests send data in the URL, which means that the data is visible in the address bar of the browser. This method is commonly used for retrieving data from the server, such as searching for information. On the other hand, POST requests send data in the request header, which is hidden from view. This method is considered more secure and is commonly used for sending sensitive information, like passwords or credit card details.

To create a form in PHP, the first step is to create a page where the form will be displayed. This page should have an HTML template that includes the necessary header and footer. In this template, the HTML form can be added.

In the provided example, the form is created using Materialize CSS classes. The form is contained within a section with a class of "container" and a class of "grey-text" for styling purposes. The form itself has a class of "white" to give it a white background.

Inside the form, input fields are added for the user to enter their email, pizza title, and ingredients. Each input field is accompanied by a label, which helps with accessibility and usability. The input fields are given names (email, title, and ingredients) so that the data can be accessed on the server-side.

Finally, a submit button is added to the form using a div with a class of [class name]. This button allows the user to submit the form and send the data to the server.

It's important to note that the action attribute of the form is left blank in this example. This attribute specifies the URL where the form data should be sent. The method attribute determines whether the form data should be sent using GET or POST.

Forms in PHP are used to collect user input and send it to the server for further processing. GET and POST are the two main methods for sending data from the client to the server. GET sends data in the URL, while POST sends data in the request header. When creating a form, an HTML template is needed, and input fields with corresponding labels are added to gather user data. A submit button is included to allow users to send the form data to the server.

In this didactic material, we will discuss the fundamentals of working with forms in PHP. Forms are an essential part of web development as they allow users to input data that can be sent to the server for processing. We will focus on the use of the GET method to send data from the form to the server.

To begin, let's take a look at the HTML code for our form template. We have an input field of type "submit" with the name and value set to "submit". This will create a button for users to submit the form. We have also applied some classes from the Materialize library to style the button.

Next, we want to make some adjustments to the form's appearance. We add some custom CSS inside the header section of the HTML. We set the form's max width to 460 pixels and add a margin of 20 pixels on the top, bottom, left, and right. Additionally, we set the padding to 20 pixels to create some space inside the form.

Now that we have our form template ready, we can discuss how to send the data entered by the user to the server. We have two methods to choose from: POST and GET. In this case, we will focus on the GET method.

To use the GET method, we need to specify the method attribute in the form tag. Additionally, we need to specify an action attribute to indicate which file on the server will handle the data. In our case, we specify that the "add.php" file will handle the request. It is important to note that this is the same file serving the HTML template.

When the form is submitted, the data will be sent to the server and displayed in the URL because of the GET request. The server will then look for the "add.php" file and run it. If we have PHP code in this file, we can process the data sent by the user.

To handle the data sent to the server, we need to check if any data is present. We can do this using the "isset" function in PHP. In this case, we check if any data has been sent via the GET method by accessing the global array "$_GET". This array stores the data sent through the URL parameters. If data is present, we can retrieve the values for the email, title, ingredients, and submit button.

We have learned about working with forms in PHP. We have covered the HTML code for the form template, made adjustments to its appearance using CSS, and discussed how to send data to the server using the GET method. We have also explored how to handle the data on the server-side using PHP.

In PHP, we can work with forms to collect data from users and process it on the server. There are two methods commonly used to send data from a form to the server: GET and POST.

The GET method sends data as part of the URL, which means the data is visible in the address bar of the browser. This method is commonly used when we want to retrieve data from the server. To access the data sent via the GET method, we can use the $_GET superglobal in PHP.

On the other hand, the POST method sends data in the body of the HTTP request, making it more secure as the data is not visible in the URL. This method is commonly used when we want to submit data to the server. To access the data sent via the POST method, we can use the $_POST superglobal in PHP.

To check if a form has been submitted, we can use the isset() function in PHP. If the form has been submitted, we can then process the data sent by accessing the appropriate superglobal array ($_GET or $_POST) and extracting the values based on their keys.

For example, if we have a form with input fields for email, title, and ingredients, and we want to retrieve the values entered by the user, we can use the following code:

```
1. if (isset($_POST['submit'])) {
2.     $email = $_POST['email'];
3.     $title = $_POST['title'];
4.     $ingredients = $_POST['ingredients'];
5.
6.     // Process the data here
7.     // ...
8. }
```

In this code, we first check if the submit button has been pressed (assuming the name attribute of the submit button is "submit"). If it has been pressed, we retrieve the values entered by the user by accessing the $_POST superglobal array with the appropriate keys.

We can then perform any necessary processing on the data, such as storing it in a database or performing calculations. The specific processing will depend on the requirements of the application.

It's important to note that when working with user input, we need to be mindful of security. In the next lesson, we will cover how to address security vulnerabilities associated with handling user input.

Forms in PHP allow us to collect data from users and send it to the server for processing. We can use the GET method to retrieve data from the server and the POST method to submit data to the server. By accessing the appropriate superglobal array ($_GET or $_POST), we can extract the values entered by the user and process them as needed.

EITC/WD/PMSF PHP AND MYSQL FUNDAMENTALS DIDACTIC MATERIALS
LESSON: FORMS IN PHP
TOPIC: XSS ATTACKS

Cross-site scripting (XSS) attacks are a common security issue that can make websites vulnerable to malicious code injection. In an XSS attack, an attacker can inject malicious JavaScript code into a website, which is then executed by the user's browser. This can lead to various harmful consequences, such as redirecting the user to a malicious website or downloading viruses.

One way that XSS attacks can occur is through user input fields, such as forms on a website. An attacker can enter JavaScript code into these fields, which is then sent to the server and returned to the browser. When the browser receives the code, it executes it, potentially causing harm.

For example, an attacker could enter JavaScript code that redirects the user to a different website. This can be done by injecting a script tag with the code "window.location = 'https://example.com'". When the user submits the form, the code is sent to the server and then returned to the browser. The browser executes the code, redirecting the user to the specified website.

To prevent XSS attacks, it is important to sanitize user input before rendering it on the website. In PHP, a function called "htmlspecialchars" can be used for this purpose. This function converts special HTML characters, such as angle brackets and quotes, into HTML entities. HTML entities are safe representations of special characters that do not execute as code.

To protect against XSS attacks, the "htmlspecialchars" function should be applied to any user input before it is outputted on the website. This can be done by surrounding the outputted data with the function. For example, if we have user input stored in variables called "email", "title", and "ingredients", we can use the "htmlspecialchars" function as follows:

- $sanitized_email = htmlspecialchars($email);
- $sanitized_title = htmlspecialchars($title);
- $sanitized_ingredients = htmlspecialchars($ingredients);

By applying the "htmlspecialchars" function to the user input, any special HTML characters will be converted into HTML entities. This ensures that the input is rendered as intended, without executing any malicious code.

By implementing this sanitization process, we can protect our website from XSS attacks and ensure the safety of our users' data and browsing experience.

EITC/WD/PMSF PHP AND MYSQL FUNDAMENTALS DIDACTIC MATERIALS
LESSON: FORMS IN PHP
TOPIC: BASIC FORM VALIDATION

Form validation is an essential part of web development, as it ensures that the data entered by users is accurate and meets the required criteria. In this lesson, we will focus on server-side validation using PHP.

When a form is submitted, the data is sent to the server and validated using PHP code. This step is important to ensure that the data used in the application or saved to a database is correct and meets the expected format.

One way to validate form data is to check if the required fields have been filled out. In PHP, we can use the empty() function to determine if a field is empty. For example, to check if the email field is empty, we can use the following code:

```
1.  if (empty($_POST['email'])) {
2.      echo "An email is required.<br>";
3.  } else {
4.      echo "Email: " . $_POST['email'] . "<br>";
5.  }
```

If the email field is empty, we display an error message. Otherwise, we display the value entered by the user.

Similarly, we can check other fields, such as the title and ingredients. For example:

```
1.  if (empty($_POST['title'])) {
2.      echo "A title is required.<br>";
3.  } else {
4.      echo "Title: " . $_POST['title'] . "<br>";
5.  }
6.
7.  if (empty($_POST['ingredients'])) {
8.      echo "At least one ingredient is required.<br>";
9.  } else {
10.     echo "Ingredients: " . $_POST['ingredients'] . "<br>";
11. }
```

By performing these checks for each field, we can ensure that all required fields are filled out. If any field is empty, an error message is displayed. Otherwise, the entered values are echoed for confirmation.

It's important to note that this basic validation only checks if the fields are empty. To perform more advanced validation, such as checking if the email is in the correct format, additional code would be required.

Form validation is crucial to ensure the accuracy and integrity of user-entered data. By using PHP, we can validate form data on the server-side, checking if required fields are filled out and displaying appropriate error messages if necessary.

In web development, form validation is an essential aspect to ensure the accuracy and reliability of user-submitted data. In this tutorial, we will explore the basics of form validation using PHP and MySQL.

One common scenario in form validation is checking if a user-submitted field contains a comma-separated list of ingredients. This can be a challenging task, but we can simplify it by using filters and additional validation techniques.

To achieve this, we will utilize regular expressions (regex). While you don't need to have prior knowledge of regex, I will provide the necessary regex codes for verifying different aspects of the input. If you are interested in learning more about regex, I have a comprehensive tutorial series on this topic available on this channel. You can find the link in the description below this material and in the next material as well.

By implementing form validation with PHP and MySQL, we can ensure that the submitted data meets the

required criteria, such as the format of a comma-separated list of ingredients. This validation process will help maintain data integrity and prevent potential issues that may arise from incorrect or invalid user inputs.

In the next tutorial, we will delve into the details of using filters and regex for form validation in PHP. Stay tuned to learn how to implement these techniques effectively.

EITC/WD/PMSF PHP AND MYSQL FUNDAMENTALS DIDACTIC MATERIALS
LESSON: FORMS IN PHP
TOPIC: FILTERS AND ADVANCED VALIDATION

In web development, it is crucial to validate user input to ensure data integrity and prevent security vulnerabilities. In this tutorial, we will focus on validating form inputs using PHP and MySQL. Specifically, we will discuss filters and advanced validation techniques.

When validating form inputs, it is not enough to check if a field has a value. We also need to verify that the value is of the correct type. For example, if we have an input field for an email address, we want to ensure that the value entered is a valid email address. Similarly, if we have a field for a title or a comma-separated list of ingredients, we need to validate those as well.

In PHP, we can use filters to validate certain types of data. PHP provides built-in filters for emails, but not for other types such as titles or comma-separated lists. For those, we will need to use regular expressions. Regular expressions are a powerful tool for pattern matching and can be used in various programming languages.

To validate an email address in PHP, we can use the filter_var function with the FILTER_VALIDATE_EMAIL filter. This function takes two parameters: the value to be checked (in this case, the email address) and the filter type. If the email address passes the validation, the function will return true. If it fails, it will return false.

If the email address is not valid, we can display an error message to the user. To do this, we can use an if statement with the negation operator (!) to check if the validation fails. If it does, we can echo an error message indicating that the email must be a valid email address.

It is important to note that this tutorial assumes some familiarity with regular expressions. If you are not familiar with regular expressions or want to learn more about them, there is a separate series on regex available for further study.

To summarize, when validating form inputs in PHP, we can use filters for certain types of data, such as emails. For other types, like titles or comma-separated lists, we need to use regular expressions. By combining these techniques, we can ensure that user input meets the required criteria.

In this lesson, we will focus on forms in PHP, specifically on filters and advanced validation. We will start by discussing the negation operator and its purpose in PHP. The negation operator returns the opposite of a given value.

Next, we will apply the negation operator to validate the email field in a form. We will use the filter_var function to check if the email is a valid email address. If the email is not valid, we will display an error message. On the other hand, if the email is valid, we will proceed with the form submission.

Moving on, we will address the validation of the title and ingredients fields. Unlike the email field, PHP does not provide built-in filters for these fields. Therefore, we will need to validate them ourselves using regular expressions.

Regular expressions, also known as regex, are patterns used to match strings of characters. We can create a regex pattern to define the required format for the title and ingredients. For example, the title may require uppercase and lowercase letters, spaces, but not special characters like '@'.

To validate the title field, we will retrieve the title from the form and use the preg_match function. The first parameter of preg_match is the regular expression pattern, and the second parameter is the title itself. If the title matches the pattern, the function will return true. If not, it will return false, and we will display an error message.

Similarly, we will validate the ingredients field using the same approach. We will retrieve the ingredients from the form and use preg_match to match it against a different regular expression pattern.

It is important to note that regular expressions can be complex, and this course does not cover them in-depth. If

you are interested in learning more about regular expressions, you can find additional resources in the provided link.

In this lesson, we learned how to use filters and advanced validation techniques in PHP to validate form fields. We used the negation operator, filter_var, and preg_match functions to validate the email, title, and ingredients fields. By understanding these concepts, we can ensure that the data submitted through forms meets our desired criteria.

In this didactic material, we will discuss the topic of Forms in PHP, specifically focusing on Filters and advanced validation.

One important aspect of form validation is ensuring that the input data meets certain criteria. This can include checking for the correct format, length, or type of data. In PHP, we can use filters to achieve this.

To begin, let's consider a scenario where we want to validate a comma-separated list of ingredients. We can use regular expressions (regex) to check if the input string matches the desired format. In this case, we want to ensure that the ingredients are words separated by commas.

To achieve this, we can use the negation operator to check if the input string does not match the desired format. If it does not match, we will display an error message. For example, if the input string is not a valid comma-separated list, we will display the error message "Ingredients must be a comma-separated list."

To demonstrate this, let's consider an example where we have a form with a field for the list of ingredients. If the input is not a valid comma-separated list, we will display the error message. If the input is valid, no error message will be displayed.

In the next step, we will discuss how to persist the data in the form. This means that if the user submits the form and there are errors, the previously entered values should remain in the form fields. This avoids the inconvenience of having to retype the values.

To achieve this, we will show the errors underneath the relevant form fields. This will allow the user to easily identify and correct any mistakes.

We have learned about filters and advanced validation in PHP forms. We have seen how to use regular expressions to validate a comma-separated list of ingredients. We have also discussed the importance of persisting data in the form fields to enhance user experience.

EITC/WD/PMSF PHP AND MYSQL FUNDAMENTALS DIDACTIC MATERIALS
LESSON: ERRORS HANDLING IN PHP
TOPIC: SHOWING ERRORS

In PHP, it is important to handle errors effectively to provide a better user experience. In this didactic material, we will learn how to show errors in a form using PHP.

To begin, we need to store the errors in a variable so that we can display them in the form. We will create an associative array called "errors" at the top of our code. This array will have positions for each type of error we want to handle, such as email, title, and ingredients. Initially, all positions will have empty strings as values.

Next, we will update each position of the "errors" array with the corresponding error message. For example, if there is an error with the email field, we will update the "email" position of the "errors" array with the error message. We will do the same for the title and ingredients.

Now that we have stored the errors in the "errors" array, we can display them in the form. Underneath each input field, we will add a separate div with a class of "error-text" to make it stand out. Inside the PHP tags, we will use the "echo" statement to output the error message for each field. We will retrieve the error message from the "errors" array using the corresponding position.

After updating the code, we can refresh the page and test it. If there are any errors, they will be displayed below the corresponding form field. For example, if the email field is empty, we will see the error message "Email is required" below the email input field.

It is important to note that in the provided transcript, there is a mistake in updating the "errors" array. The error messages should be stored in the respective positions of the array, not directly echoed. This mistake has been corrected in the explanation above.

By implementing this error handling technique, we can provide more informative and user-friendly feedback to users when they submit a form.

In PHP, when working with web forms, it is important to handle errors and display them to the user in a user-friendly manner. One common error is the "undefined variable" error, which occurs when trying to access a variable that has not been defined or initialized.

To handle this error, we can initialize our variables at the beginning of the code, before they are used. By setting them to empty strings, we ensure that they have a value even if the user has not yet submitted the form. This prevents the "undefined variable" error from occurring.

To initialize the variables, we can use a simple trick. Since we want to set all the variables to the same value (an empty string), we can assign the value of one variable to the others. For example, we can set the title, email, and ingredients variables to empty strings by writing:

```
1. $title = $email = $ingredients = "";
```

By doing this, we ensure that these variables have a value even before the user submits the form. This way, when the template is sent back to the user, these variables will be displayed as empty strings in the input fields.

After initializing the variables, we can proceed with our code and handle form submissions. When the user submits the form, we update the variables with the values entered by the user. This way, when the template is sent back to the user, the input fields will be populated with the user's entered values.

It is worth noting that when outputting user-entered data to the browser, it is important to sanitize the data to prevent any potential security risks. One way to do this is by using the `htmlspecialchars()` function. This function converts special characters to their HTML entities, ensuring that the data is displayed correctly and safely.

To use the `htmlspecialchars()` function, we can wrap it around the variables that we are outputting to the

browser. For example, when echoing out the variables in the value attribute of an input field, we can write:

```
1. <input type="text" name="title" value="<?php echo htmlspecialchars($title); ?>">
```

By doing this, we ensure that any special characters entered by the user are properly escaped and displayed as intended.

To handle errors and show errors in PHP when working with web forms, we can initialize our variables to empty strings before they are used, ensuring that they have a value even if the user has not yet submitted the form. This prevents the "undefined variable" error from occurring. Additionally, when outputting user-entered data to the browser, it is important to sanitize the data using the `htmlspecialchars()` function to prevent potential security risks.

EITC/WD/PMSF PHP AND MYSQL FUNDAMENTALS DIDACTIC MATERIALS
LESSON: ERRORS HANDLING IN PHP
TOPIC: CHECKING FOR ERRORS AND REDIRECTING

When developing a web application, it is crucial to handle errors effectively. In this didactic material, we will focus on checking for errors and redirecting users in PHP.

To begin, after a user submits a form, we need to check for any errors in the PHP code. This check is essential to ensure the form's validity. If there are no errors, we can redirect the user to the home page or any other desired page. If there are errors, we won't perform any redirect, allowing the errors to be displayed to the user.

To check for errors, we can use the array_filter method. This method cycles through an array and applies a callback function to each element. In our case, we pass in the errors array, which contains potential errors. If all positions in the array are empty or evaluate to false, the array_filter function will return false. This indicates that there are no errors. Conversely, if there are non-empty errors, the function will return true.

Once we have determined whether there are errors or not, we can take appropriate action. If there are errors (the array_filter function returns true), we can use the echo statement to display a message indicating the presence of errors. On the other hand, if there are no errors (the array_filter function returns false), we can use the echo statement to indicate that the form is valid.

Now, let's consider the redirection process. We want to redirect the user only when there are no errors. To achieve this, we can use the header function. By passing the string "location" as a parameter to the header function, we can specify the desired destination page. In our case, we redirect the user to the index.php file. This means that the server will process the index.php file and send it back to the browser instead of the current page.

To summarize the steps involved in error handling and redirection:
1. Check for errors using the array_filter method on the errors array.
2. If there are errors, display a message indicating their presence.
3. If there are no errors, indicate that the form is valid.
4. Redirect the user to a desired page using the header function and specifying the location.

It is important to note that in the future, instead of redirecting directly to the index.php file, we can first save the data to the database and then redirect the user. This ensures that only valid data is stored in the database.

By following these steps, you can effectively handle errors and redirect users in your PHP web application.

EITC/WD/PMSF PHP AND MYSQL FUNDAMENTALS DIDACTIC MATERIALS
LESSON: GETTING STARTED WITH MYSQL
TOPIC: INTRODUCTION TO MYSQL

In this material, we will introduce the basics of MySQL, a relational database management system commonly used in web development. MySQL allows us to store and manipulate data efficiently. We will learn how to create a database, save data, retrieve data, and delete data using MySQL.

MySQL is a server-based system, meaning it runs on a server and can be accessed by multiple clients. To interact with the database from our PHP files, we use a query language called SQL (Structured Query Language). SQL allows us to perform various operations on the database, such as creating new data, deleting data, updating data, and retrieving data.

A typical MySQL database consists of multiple tables. Each table is used to store a specific type of record or data. For example, we could have tables for users, blogs, reviews, and pizzas. Each table is made up of rows and columns. Each row represents an individual record in the table, while each column represents a property or attribute of that record.

For example, in a blogs table, we might have columns for ID, Content, and User ID. The ID column uniquely identifies each blog record, the Content column stores the content of the blog, and the User ID column references the user who wrote the blog. This User ID is known as a foreign key, which allows us to link tables together based on related data.

To illustrate the structure of a MySQL database, let's consider a simplified example. We have a form on a website to capture an email, a pizza title, and some ingredients. Our table for pizzas would have columns for ID, title, ingredients, email, and created. The ID column uniquely identifies each pizza record, the title column stores the title of the pizza, the ingredients column stores the ingredients, the email column stores the email captured from the form, and the created column stores the timestamp indicating when the pizza was created.

In this example, the email field could be replaced with a user ID if we had an authentication system and actual users on the website. However, since we don't have that in our project, we will store the email directly in the pizzas table.

To summarize, MySQL is a powerful relational database management system used in web development. It allows us to store and manipulate data efficiently. A MySQL database consists of multiple tables, each storing a specific type of record or data. Each table is made up of rows and columns, where each row represents an individual record, and each column represents a property or attribute of that record. We use SQL to communicate with the database from our PHP code, performing operations such as creating, deleting, updating, and retrieving data.

In the next lesson, we will start by creating our first MySQL database.

EITC/WD/PMSF PHP AND MYSQL FUNDAMENTALS DIDACTIC MATERIALS
LESSON: GETTING STARTED WITH MYSQL
TOPIC: SETTING UP A MYSQL DATABASE

To create our first MySQL database, we need to ensure that XAMPP is running and click on the "Start" button next to MySQL. This will start the MySQL service. Then, we can click on the "Admin" button, which will open up the PHPMyAdmin interface in a browser.

PHPMyAdmin is a visual interface that allows us to create databases and tables. It is commonly found on web servers with PHP installed. In PHPMyAdmin, we can see a list of current databases on the left side of the interface. Clicking on the "Databases" tab will also display the same list.

Within a database, there can be multiple tables. By clicking on a database, we can see the tables listed within it. Each table consists of columns that represent different properties, and rows that represent records.

To create a new database, we can go to the "Databases" section and click on "Create". We need to provide a name for the database, such as "ninja pizza". After creating the database, we can proceed to create a new table within it.

In our case, the table will store pizza information. We will name the table "pizzas" and define five columns: ID, title, ingredients, email, and created_at. The ID column will be set as an auto-incrementing primary key. The title, ingredients, and email columns will be of type varchar with a maximum length of 255 characters. The created_at column will be of type timestamp.

Once we have specified the columns, we can click on "Go" to create the table. Now we have a database with a table ready to store pizza information.

Please note that the PHPMyAdmin interface also provides additional features, such as managing user accounts, exporting/importing data, and various settings. However, these topics are beyond the scope of this beginner-level series.

We have learned how to create a MySQL database using the PHPMyAdmin interface. We have also created a table within the database to store pizza information, defining the necessary columns and their data types.

In this material, we will learn how to set up a MySQL database for web development using PHP. MySQL is a popular open-source database management system that is widely used in web development. Setting up a MySQL database involves creating a table, defining its structure, and adding data to it.

To begin, open the PHPMyAdmin interface and select the database where you want to create the table. Click on the "SQL" tab to execute SQL queries. In the SQL editor, enter the following query to create a new table:

```
CREATE TABLE pizza (
id INT AUTO_INCREMENT PRIMARY KEY,
title VARCHAR(255),
ingredients VARCHAR(255),
email VARCHAR(255),
timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

This query creates a table named "pizza" with five columns: "id", "title", "ingredients", "email", and "timestamp". The "id" column is an auto-incrementing integer and serves as the primary key. The "title", "ingredients", and "email" columns are of type VARCHAR and can store up to 255 characters. The "timestamp" column is of type TIMESTAMP and has a default value of the current timestamp.

After entering the query, click on the "Go" button to execute it. The table will be created, and you can see its structure in the "Structure" tab. The "id" column will have the AUTO_INCREMENT attribute, which means it will automatically generate a unique value for each new record.

To add data to the table, click on the "Insert" tab. In the form, enter the values for each column. For example, you can add a record with the following values:

- id: 1
- title: "The Ninja Supreme"
- ingredients: "tomato, cheese, tofu"
- email: "shawn@thenetninja.co.uk"
- timestamp: current timestamp

Click on the "Go" button to insert the record into the table. You can see the SQL query that was executed to perform the insertion. In the "Browse" tab, you will now see the added record in the table.

You can add more records by repeating the process. The "id" column will automatically increment for each new record, so you don't need to specify it. For example, you can add another record with the following values:

- title: "The Mario Supreme"
- ingredients: "tomato, cheese, mushroom"
- email: "mario@thenetninja.co.uk"
- timestamp: current timestamp

After inserting the record, you can see it in the "Browse" tab. The "id" column will have incremented to 2.

Now that we have set up our database and added data to it, we can start using PHP to interact with the database. In the next material, we will learn how to connect to the database and perform CRUD (Create, Read, Update, Delete) operations using PHP.

EITC/WD/PMSF PHP AND MYSQL FUNDAMENTALS DIDACTIC MATERIALS
LESSON: GETTING STARTED WITH MYSQL
TOPIC: CONNECTING TO A DATABASE

To connect to a database from PHP, we need to provide a username and password. In this tutorial, we will be using the default root username and password. However, if you prefer, you can create a new user account specifically for this database.

To create a new user account, go to the user accounts section and click on "Add User Account". Enter the desired username and hostname, which in this case will be "Shaun" and "localhost" respectively. Set a password for the account, such as "test1234". Check the box to allow global privileges, which will grant the user account full access to the database. Finally, click "Go" to create the new user account.

Now that we have the user account set up, let's move on to connecting to the database from the PHP code. Open the "index.php" file, as this is where we will establish the initial connection. We have two options for communicating with the database: MySQLi (improved) and PDO (PHP Data Objects). For this tutorial, we will be using MySQLi, as it allows for a more procedural approach.

To connect to the database, we will use the "mysqli_connect" function. Create a variable called "con" to store the connection reference. Set it equal to the "mysqli_connect" function, which takes four parameters: the host (localhost), the username (Shaun), the password (test1234), and the database name (ninja_pizza).

After establishing the connection, it is important to check if it was successful. Use an if statement to check if the connection variable is false, indicating an error. If there is an error, echo a message stating "Connection error" and use the "mysqli_connect_error" function to retrieve the specific error message.

Save the file and run it in the browser. If there are no errors, nothing will be echoed to the screen. However, if there is a mistake in the connection details, an error message will be displayed.

To connect to a MySQL database in PHP, we need to ensure that the connection is established successfully. If there are any issues with the connection, an error message will be displayed, allowing us to debug the problem.

To begin, let's change the code back to the original "ninja pizza" and refresh the page to ensure that the connection has been established successfully. If everything is working correctly, we should see the page load without any errors.

Once we have successfully connected to the database, we can move forward and learn how to retrieve data from it and display it on the webpage. Additionally, we will explore how to save data to the database when we navigate to the "add.php" page.

In the upcoming material, we will cover the process of displaying data on the index page and saving data to the database on the add.php page.

EITC/WD/PMSF PHP AND MYSQL FUNDAMENTALS DIDACTIC MATERIALS
LESSON: GETTING STARTED WITH MYSQL
TOPIC: GETTING DATA FROM A DATABASE

To retrieve data from a MySQL database, we need to perform three steps: constructing the query, making the query, and fetching the results.

First, let's construct the query to get all the pizzas. We will use the SELECT command followed by the keyword "star" to indicate that we want all the columns from the table. For example, if each record has columns for ID, title, ingredients, email, and created, we can write the query as follows: SELECT star FROM pizzas.

Next, we need to make the query and get the results. To do this, we create a variable called "results" and set it equal to "mysqli_query(connection, query)". The "connection" variable represents our connection to the database, and the "query" variable represents the query we constructed earlier.

Finally, we need to fetch the resulting rows. Each row represents a record in the table and is stored as an array. To retrieve the rows, we create a variable called "pizzas" and set it equal to "mysqli_fetch_all(results, MYSQLI_ASSOC)". This will return the rows as an associative array.

To test if everything is working correctly, we can print out the "pizzas" array using the "print_r" function. This will display the retrieved data in the browser.

To retrieve data from a MySQL database, we first construct the query using the SELECT command. Then, we make the query and store the results in a variable. Finally, we fetch the resulting rows and store them in an array.

After executing the SQL query to retrieve data from a database, there are a few steps we should follow to properly handle the results and close the connection.

First, it is good practice to free the results from memory. Although this step is not mandatory, it helps optimize memory usage. To achieve this, we can use the `mysqli_free_result()` function. This function takes the result object as a parameter and releases the associated memory.

Next, we need to close the connection to the database. This is done using the `mysqli_close()` function, which takes the connection object as a parameter. Closing the connection is important to free up system resources and ensure proper termination of the database connection.

To summarize the steps:
1. Free the results from memory using `mysqli_free_result($result)`.
2. Close the connection to the database using `mysqli_close($connection)`.

By following these steps, we ensure that the resources used for the query are properly released and the connection to the database is closed.

In the example provided, the results are stored in an array called "pizzas". Each element of the array represents a pizza and contains an associative array with details such as the title and ingredients. However, in this particular video, the data is not yet output to the browser. This will be covered in a subsequent video.

EITC/WD/PMSF PHP AND MYSQL FUNDAMENTALS DIDACTIC MATERIALS
LESSON: FURTHER ADVANCING IN PHP
TOPIC: RENDERING DATA TO THE BROWSER

When working with databases, it is often necessary to order the retrieved data in a specific way. In PHP, we can achieve this by using the "ORDER BY" clause in our SQL query. By specifying the column or property we want to order by, we can arrange the data in ascending or descending order.

For example, if we have a table of pizzas and we want to order them by their creation timestamp, we can use the "ORDER BY" clause with the "created_at" column. This will display the pizzas in chronological order.

To output the ordered data to the browser, we can use an array. Each pizza in the array is represented as an associative array, allowing us to access and display its properties.

To begin, we create a header element with the class "sensor" and "gray-text", and set its content to "Pizza". Below that, we create a div element with the class "container", which centralizes the content within a central column. We then utilize the materialized grid system to create a row for the grid using the class "row".

Inside the row, we use a foreach loop to iterate through the "pizzas" array, which contains the ordered data. Each individual pizza is referred to as "$pizza" within the loop. Within the loop, we create a div element with the class "col" to define the width of the pizza's space on the grid system. We specify the width for small screens as 6 columns and for medium screens as 3 columns.

Inside the space given to each pizza, we create a div element with the class "card" to enhance its appearance. We also add a class of "z-depth-0" to remove the box shadow. Inside this card, we create another div element with the class "card-content" to ensure proper display on the screen. We also add a class of "center" to centralize any text within the div.

To output the pizza's title, we create an h6 element and use PHP tags to echo the value of the "title" property of the pizza. To ensure security, we use the "htmlspecialchars" function to escape any malicious code.

Similarly, we can output the pizza's ingredients by creating a div element and using PHP tags to echo the value of the "ingredients" property.

By following this approach, we can render the ordered data to the browser in a visually appealing and organized manner.

In this material, we will discuss how to render data to the browser in PHP. Rendering data to the browser is an essential part of web development, as it allows us to display dynamic content to the user.

To begin with, let's assume that we have a database containing information about pizzas. Our goal is to retrieve this data and display it on a webpage. We will start by outputting the title and ingredients of each pizza.

First, we need to retrieve the data from the database. We can do this by using PHP's database functions, such as mysqli_query() or PDO. Once we have fetched the data, we can store it in a variable.

Next, we can use a loop, such as a foreach loop, to iterate through the array of pizzas. For each pizza, we can access its title and ingredients and output them to the browser.

To create a visually appealing layout, we can use HTML and CSS. We can structure the output using div elements and apply classes to style them accordingly. For example, we can create a div with a class of "card" to represent each pizza. Inside this div, we can have a div with a class of "card-content" to hold the title and ingredients.

Additionally, we can add a button, such as a "More Info" button, for each pizza. When clicked, this button will take the user to a dedicated page with more details about the pizza, such as who created it and when it was created.

To align the button to the right, we can create another div with a class of "card-action" and apply a CSS rule to align the text inside it. We can also add a class of "brand-text" to the anchor tag within this div to give it a gold color.

Finally, it is important to ensure the security of our application. When outputting user-generated data, such as the title and ingredients of a pizza, we should use the htmlspecialchars() function to prevent any potential malicious code from being executed.

Rendering data to the browser in PHP involves retrieving data from a database, iterating through it, and outputting it to the browser using HTML and CSS. By following these steps, we can create dynamic webpages that display relevant information to the user.

EITC/WD/PMSF PHP AND MYSQL FUNDAMENTALS DIDACTIC MATERIALS
LESSON: FURTHER ADVANCING IN PHP
TOPIC: THE EXPLODE FUNCTION

In this material, we will discuss how to use the explode function in PHP to convert a string into an array. This function is useful when we want to split a string into multiple parts based on a specific character or delimiter.

To begin, let's consider a scenario where we have retrieved data from a database and need to output the individual ingredients for a recipe. Currently, the ingredients are stored as a single string within a div tag. However, we would like to display each ingredient separately using a ul tag and a separate li tag for each ingredient.

To achieve this, we can use the explode function in PHP. The explode function takes two arguments: the delimiter character and the string we want to split. In our case, the delimiter will be a comma, as we want to split the string whenever a comma is encountered.

For example, if we have a string "tomato, cheese, mushroom", the explode function will split it into an array with three elements: "tomato", "cheese", and "mushroom".

To demonstrate this, we can use the explode function on the ingredients string. We retrieve the ingredients from the first pizza in the array and pass it as the second argument to the explode function. The first argument is the comma delimiter. The result of the explode function will be an array of ingredients.

We can then print this array using the print_r function, which is used to display arrays in a readable format. When we run the code, we will see the array of ingredients displayed on the screen.

Now that we have the array of ingredients, we can iterate over it using a foreach loop. Within the loop, we will output an li tag for each ingredient. This will ensure that each ingredient is displayed as a separate item in the list.

To summarize the steps:
1. Use the explode function to split the ingredients string into an array, using a comma as the delimiter.
2. Print the resulting array to verify that the split was successful.
3. Iterate over the array using a foreach loop.
4. Output an li tag for each ingredient.

By following these steps, we can convert a string of ingredients into an array and display them as separate items using the explode function in PHP.

To further advance in PHP, we will now explore the explode function. The explode function allows us to take a string and convert it into an array. This can be particularly useful when dealing with user input or when we need to manipulate strings.

To use the explode function, we start by enclosing our code block within PHP tags. Within the code block, we use the explode function to split a string into an array. In this example, we are splitting the string "cheese tomato mushroom" into individual ingredients.

To output each ingredient as a list item (li tag), we use a loop to cycle through the array. For each element in the array, we echo out an li tag containing the ingredient. However, it's important to remember that user input should not be trusted, so we use the HTML special chars function to ensure that any potentially harmful characters are properly encoded.

After saving and refreshing our page, we can see that each ingredient now has its own li tag, and they are displayed as a list. This provides a more organized and visually appealing representation of the ingredients.

The explode function in PHP allows us to convert a string into an array. By using this function, we can easily manipulate and work with individual elements of the string. Additionally, when dealing with user input, it's important to properly sanitize and encode the data to ensure security.

EITC/WD/PMSF PHP AND MYSQL FUNDAMENTALS DIDACTIC MATERIALS
LESSON: FURTHER ADVANCING IN PHP
TOPIC: CONTROL FLOW ALT SYNTAX

In PHP, when we start to nest loops and conditional statements like if statements, it can become challenging to keep track of the closing curly braces and what they are closing for. To address this, there is an alternative syntax that can be used, which is more explicit and helps in keeping track of the code structure.

Instead of using curly braces, we can use a colon at the beginning of the block and use the "end" keyword followed by the name of the loop or condition to indicate the end of the block. For example, instead of using curly braces for a foreach loop, we would use a colon at the beginning and "end foreach" at the end.

This alternative syntax makes it clearer which block is being closed, as curly braces can close anything and it can be challenging to match them up correctly, especially when dealing with nested code.

To illustrate this, let's convert the code example provided into the alternative syntax. We will replace the opening curly braces with colons and the closing curly braces with the "end" statement for the respective loops or conditions.

After making these changes, we can still check if the code works correctly by running it in a browser.

Additionally, let's consider another example using an if statement. In this case, we will use curly braces initially, but we can convert them to the alternative syntax as well.

Inside the if statement, we will check if the count of pizzas is greater than or equal to two. If it is, we will output a message indicating that there are two or more pizzas. Otherwise, we will output a message indicating that there are less than two pizzas.

By using the alternative syntax, we can make the code structure cleaner and easier to read. We will replace the opening and closing curly braces with colons and use the "end if" statement at the end.

By implementing these changes, we can still achieve the same results when running the code in a browser.

The alternative syntax in PHP allows for a more explicit representation of code blocks, making it easier to keep track of their structure and avoid confusion when dealing with nested code.

EITC/WD/PMSF PHP AND MYSQL FUNDAMENTALS DIDACTIC MATERIALS
LESSON: ADVANCING WITH MYSQL
TOPIC: SAVING DATA TO THE DATABASE

In order to add functionality to the navbar, we need to add links to the form and home page. This can be done by modifying the header template. The brand logo link should be directed to the home page (index.php), and the button link should be directed to the add page (add.php). Once these changes are made, the links will work as intended.

Now, let's focus on saving data to the database. In the add page, after entering the details and clicking Submit, we need to run a file that will handle the data submission. Currently, if there are no errors in the form, the file redirects to the index page. However, instead of redirecting, we want to save the data to the database.

To accomplish this, we first need to establish a connection to the database. We have already done this in the index.php file, but we also need to do it in the add.php file since this is where we will be saving the data. Instead of duplicating the code, we can modularize it by creating a separate file. Let's create a new folder called "config" and inside it, create a file named "DB_connect.php". This file will handle the database connection.

To include the connection code in the add.php file, we can use the "include" statement. By including the "DB_connect.php" file, we can reuse the connection code whenever we need it. In the add.php file, add the following line at the top to include the "DB_connect.php" file:

```
include 'config/DB_connect.php';
```

Now that we have included the connection code, we can proceed with saving the data to the database. In the add.php file, within the else clause of the form validation, we can add the data to the database. We have the email, title, and ingredients stored in variables. Before directly saving them to the database, it is recommended to use the "mysqli_real_escape_string" function. This function escapes any potentially harmful SQL characters and protects against SQL injection.

To use this function, we need to override the existing values of the email, title, and ingredients variables. Replace the current values with the following lines of code:

```
$email = mysqli_real_escape_string($connection, $email);
$title = mysqli_real_escape_string($connection, $title);
$ingredients = mysqli_real_escape_string($connection, $ingredients);
```

By passing the connection reference and the respective values to the "mysqli_real_escape_string" function, we ensure that the data is safe to be stored in the database.

Please note that there are alternative approaches, such as using PDO (PHP Data Objects) and prepared statements, which provide additional security. However, since we have not covered PDO in this material, we are using this method for now. In the future, you may want to explore PDO and prepared statements as they are highly recommended for secure database interactions.

To save data to a MySQL database in web development using PHP, we need to follow a few steps. Let's go through them in detail.

First, we need to gather the data that we want to save. In this example, we want to save the email, title, and ingredients of a pizza. We retrieve this data from the user and store it in variables.

Next, we create an SQL string that will insert the data into the database. We use the "INSERT INTO" command to specify the table we want to insert data into, which in this case is the "pizzas" table. We then list the columns we want to insert data into, which are the title, email, and ingredients columns. Finally, we specify the values we want to insert, which are the variables containing the data.

Once we have the SQL string ready, we can execute it by using the "mysqli_query" function. We pass in the database connection and the SQL string as parameters. We wrap this function call in an if statement to check if

the query was successful. If it was, we know that the data has been saved to the database. If it wasn't, there might be an error, and we can display the error message.

If the query was successful, we can redirect the user to another page, such as the index page, to indicate that the data has been saved.

To save data to a MySQL database in web development using PHP, we gather the data from the user, create an SQL string to insert the data into the database, execute the SQL query, check if it was successful, and then redirect the user if it was.

EITC/WD/PMSF PHP AND MYSQL FUNDAMENTALS DIDACTIC MATERIALS
LESSON: ADVANCING WITH MYSQL
TOPIC: GETTING A SINGLE RECORD

In this didactic material, we will learn how to create a details page for a pizza website using PHP and MySQL. The details page will display information about a specific pizza, including the title, ingredients, creator, and creation date. Additionally, we will add a delete button to the page to allow users to delete the pizza.

To begin, we need to create a new file called "details.php" in our project directory. Inside this file, we will use PHP tags to write our code. We will also include HTML tags to structure the page.

Next, we will include our header and footer templates in the details page. Instead of manually writing out the templates, we can copy and paste them from our existing code.

For now, let's add a simple heading to the details page using the <h2> tag. We will set the heading to "Details" but we can change it later.

In our index file, where we list the different pizzas, we need to link each pizza to its corresponding details page. To do this, we will modify the href attribute of the link. We will use a query string to pass additional information about the pizza to the details page.

We will append a question mark to the URL, followed by the property name "ID" and the ID of the individual pizza. We can retrieve the ID by echoing it out from the PHP code. Since we are iterating through a loop of pizzas, we have access to the ID property of each pizza.

To retrieve the information on the details page, we will use the GET array. We will check if the ID parameter is present in the URL using the isset() function. If it is set, we will proceed with retrieving the data.

We also need to include the DB connect file to interact with the database. We will use the include statement to include the config file and the DB connect file.

Before making any queries with user-entered data, we need to sanitize the input to prevent SQL injection attacks. We will use the mysqli_real_escape_string() function to escape any special characters in the ID parameter.

Once we have the sanitized ID, we can construct the SQL query to retrieve the specific pizza record. We will use the SELECT statement to select all fields from the table.

Finally, we will execute the query and display the retrieved information on the details page.

To retrieve a single record from a MySQL database using PHP, we can follow these steps:

1. Construct the SQL query: In this case, we want to retrieve a single record from the "pizzas" table based on a specific ID. To do this, we use the "WHERE" clause in our query to specify that we only want the record where the ID field is equal to the ID we have obtained from the user.

2. Execute the query: We use the "mysqli_query" function to execute the query. We pass in the database connection and the SQL query as parameters.

3. Fetch the result: To retrieve the result of the query, we use the "mysqli_fetch_assoc" function. Since we only want a single record, we fetch the result as an associative array. This allows us to access the data using the column names as keys.

4. Store the result: We store the fetched result in a variable called "pizza" for further use.

5. Free the result and close the connection: It is good practice to free the result and close the connection once we are done with them. We use the "mysqli_free_result" function to free the result and the "mysqli_close" function to close the connection.

6. Output the result: We can now output the retrieved data to the browser. We use HTML tags and PHP echo statements to display the pizza details, such as the title, creator, and creation date. We use the "htmlspecialchars" function to ensure that any special characters in the data are properly encoded.

Here is an example of the PHP code that accomplishes these steps:

```
1.  <?php
2.  // Construct the SQL query
3.  $sql = "SELECT * FROM pizzas WHERE ID = $id";
4.
5.  // Execute the query
6.  $result = mysqli_query($connection, $sql);
7.
8.  // Fetch the result as an associative array
9.  $pizza = mysqli_fetch_assoc($result);
10.
11. // Free the result and close the connection
12. mysqli_free_result($result);
13. mysqli_close($connection);
14.
15. // Output the result
16. if ($pizza) {
17.     echo "<div class='container'>";
18.     echo "<div class='center'>";
19.     echo "<h4>" . htmlspecialchars($pizza['title']) . "</h4>";
20.     echo "<p>Created by: " . htmlspecialchars($pizza['email']) . "</p>";
21.     echo "<p>Created on: " . date("F j, Y", strtotime($pizza['created_at'])) . "</p>
    ";
22.     echo "<h5>Ingredients:</h5>";
23.     echo "<p>" . htmlspecialchars($pizza['ingredients']) . "</p>";
24.     echo "</div>";
25.     echo "</div>";
26. } else {
27.     echo "No pizza found.";
28. }
29. ?>
```

This code retrieves a single pizza record from the database based on the provided ID and displays the relevant details if the pizza exists. If no pizza is found, it displays a message indicating that no pizza was found.

To retrieve a single record from a database table and display it in a web browser, we will follow a few steps.

First, we need to ensure that the PHP code is correctly written. In the given code snippet, there is an error regarding an undefined variable on line 38. To fix this, we should change the variable name to "double_zet" and save the file. After refreshing the page, the error should be resolved.

Once the code is error-free, we can proceed to retrieve the desired record from the database. In this case, we are retrieving information about a pizza. The name, creation date, and ingredients of the pizza will be displayed on the webpage.

To remove unnecessary content from the page, we can remove the top section that is no longer needed.

Now, let's consider the scenario where the requested pizza ID does not exist in the database. In this case, we want to display an error message on the page. To achieve this, we can add an "else" statement after the database query. Inside the "else" block, we can use HTML to output an appropriate error message, such as "There's no such pizza."

After making these changes, we can save the file and refresh the page. If we provide a valid pizza ID, such as 6, we will see the pizza details displayed. However, if the ID is not valid, we will see the error message we defined.

It is important to note that this code assumes that the database connection and query execution have been

properly implemented. These aspects are not covered in this specific material, but should be included in a complete web development course.

To summarize, this material demonstrates how to retrieve a single record from a database table and display it in a web browser using PHP. We covered fixing an error, removing unnecessary content, and handling the scenario where the requested record does not exist.

EITC/WD/PMSF PHP AND MYSQL FUNDAMENTALS DIDACTIC MATERIALS
LESSON: ADVANCING WITH MYSQL
TOPIC: DELETING A RECORD

To delete a record from a database table, we typically use a form with a hidden input field containing the ID of the record we want to delete. When the form is submitted, a POST request is made to the server, which then detects if the submit button was pressed. If it was, the ID value is extracted from the hidden input field and used to construct a SQL query to delete the record from the database.

In the context of our web development project, we want to add a delete button to the details page of a pizza. When the delete button is clicked, the corresponding pizza record should be deleted from the database and the user should be redirected to the home page.

To implement this functionality, we need to modify the details page template. We add a form with a hidden input field to hold the ID of the pizza we want to delete. The value of the hidden input field is set to the ID of the pizza. We can access this ID from the URL when loading the details page. We then output the pizza ID as the value of the hidden input field.

Next, we add a submit button inside the form. The button has a name of "delete" and a value of "delete". We also apply some CSS classes to style the button.

Upon refreshing the page, the delete button will be visible. However, clicking the button currently does nothing because the form lacks an action and method. We set the action to the details.php file, which contains the logic for deleting the record. The method is set to POST.

To handle the POST request in the details.php file, we use the isset() function to check if the "delete" value is set in the $_POST array. If it is, we proceed with the deletion process.

Inside the deletion code block, we first retrieve the ID value from the hidden input field. We then use the mysqli_real_escape_string() function to sanitize the ID value and prevent any potential SQL injection attacks.

After sanitizing the ID, we construct a SQL query to delete the record with the corresponding ID. We execute the query and check if it was successful.

That's the process of deleting a record from the database using PHP and MySQL. By implementing this functionality, we can now delete pizzas from our web application.

To delete a record from a MySQL database using PHP, we need to follow a few steps. First, we need to store the value of the ID we want to delete in a variable. To ensure the security of our application, we use the mysqli_real_escape_string function to escape any harmful characters that may be included in the ID value. This function takes two parameters: the database connection and the value to be escaped.

Next, we create a query to delete the record from the database. We use the DELETE keyword and specify the table name, in this case, "pizzas". We also use a WHERE clause to specify that only the record with the ID equal to the one we want to delete should be deleted.

After creating the query, we execute it using the mysqli_query function. We pass the database connection and the query as parameters. If the query is successful, we perform the desired action, in this case, changing the user's location to a specified page using the header function. If the query fails, we output an error message using the echo function and concatenate it with the actual MySQL error message.

Here is the code in a step-by-step format:

1. Store the ID value to be deleted in a variable using mysqli_real_escape_string:

```
1. $delete = mysqli_real_escape_string($connection, $_POST['ID_to_delete']);
```

2. Create the DELETE query:

```
$query = "DELETE FROM pizzas WHERE ID = $delete";
```

3. Execute the query and check if it is successful:

```
if(mysqli_query($connection, $query)){
    // Success: Perform desired action
    header('Location: index.php');
} else{
    // Failure: Output error message
    echo "Query error: " . mysqli_error($connection);
}
```

By following these steps, we can safely delete a record from a MySQL database using PHP.

EITC/WD/PMSF PHP AND MYSQL FUNDAMENTALS DIDACTIC MATERIALS
LESSON: EXPERTISE IN PHP
TOPIC: DESIGN ELEMENTS

In this lesson, we will focus on enhancing the functionality of our web development project by adding images and making CSS adjustments. Although this is not directly related to PHP, it will help us achieve a more polished and complete design for our template.

To begin, we need to obtain the image files for the pizzas. These images are available on my GitHub repository under the branch "less than 33" in the image folder. Specifically, we will be using the SVG file named "pizza.svg". Make sure to download this file and add it to your project folder, placing it inside the "image" directory.

Next, we will incorporate these pizza images into our index page. Within the code where we cycle through the pizzas and display them, we will add an image tag for each pizza. The source attribute of the image tag should point to the "pizza.svg" file in the image folder. Additionally, we will assign a class to this image tag, which we will use for CSS styling purposes. Let's name the class "pizza".

After saving the changes and viewing the page in a browser, you will notice that the images appear in a somewhat awkward manner. To improve their appearance, we will apply some CSS wizardry. In the header section of our template, we will define a CSS style for the "pizza" class. The style will include the following properties:
- Width: 100 pixels
- Margin: 40 pixels at the top, and auto for left and right margins, with -30 pixels at the bottom to close the gap between the image and the content underneath.
- Display: block
- Position: relative, with a vertical adjustment of -30 pixels to compensate for the image's position.

Upon saving these CSS changes and refreshing the page, you will notice that the images now align properly within the designated area for each pizza. Additionally, whenever a new pizza is added, the corresponding image will be displayed alongside it.

To further enhance the design, we can modify the text color of the details page. By applying a gray color to the text, we can achieve a more visually pleasing appearance. Simply add the CSS style "color: gray" to the appropriate section of the template.

With these adjustments, our project is now complete in terms of functionality and design. However, please note that there are still more PHP-related topics to cover in the upcoming lessons. In the next material, we will explore ternary operators, which will take us away from this specific project.

EITC/WD/PMSF PHP AND MYSQL FUNDAMENTALS DIDACTIC MATERIALS
LESSON: EXPERTISE IN PHP
TOPIC: TERNARY OPERATORS

Ternary operators are a concise alternative to using if statements in PHP. They allow us to have two different outcomes based on a certain condition, but in just one line of code.

To understand how ternary operators work, let's look at an example. Suppose we have a variable called "score" and we set it equal to 50. If we wanted to check if the score is over a certain value using an if statement, it would look something like this:

```
1. if ($score > 40) {
2.     echo "High score";
3. } else {
4.     echo "Low score";
5. }
```

In this example, we are doing two different things based on the evaluation of the condition. If the score is greater than 40, we echo "High score". Otherwise, we echo "Low score".

Now, let's see how we can achieve the same outcome using a ternary operator. The syntax of a ternary operator is as follows:

```
1. $variable = (condition) ? value_if_true : value_if_false;
```

In our example, we can write the ternary operator like this:

```
1. $val = ($score > 40) ? "High score" : "Low score";
```

Here, we are evaluating the condition `$score > 40`. If this condition is true, the value assigned to `$val` will be "High score". If the condition is false, the value assigned to `$val` will be "Low score".

It's important to note that a ternary operator returns a value, so we can assign it to a variable like we did with `$val` in the example. Alternatively, we can directly echo the result without storing it in a variable.

For example:

```
1. echo ($score > 40) ? "High score" : "Low score";
```

This will directly output "High score" or "Low score" based on the condition.

Ternary operators can be particularly useful when working with HTML templates, as they allow for more concise and readable code. Instead of using multiple lines with open and close braces, we can achieve the same result in just one line.

For instance, if we wanted to output a result based on the condition, we could write:

```
1. echo "<p>" . (($score > 40) ? "High score" : "Low score") . "</p>";
```

In this example, we use the ternary operator to determine the value to be echoed within the `<p>` tags. This approach simplifies the code and improves readability.

Ternary operators provide a concise and efficient way to handle conditional statements in PHP. They allow us to have two different outcomes based on a condition, all in just one line of code.

EITC/WD/PMSF PHP AND MYSQL FUNDAMENTALS DIDACTIC MATERIALS
LESSON: EXPERTISE IN PHP
TOPIC: SUPERGLOBALS

Super globals are special array variables in PHP that are automatically populated with values when the code runs. They can be accessed from anywhere in the PHP code. Two examples of super globals are `$_GET` and `$_POST`, which are arrays that store values from query strings and form submissions respectively.

Another super global in PHP is `$_SERVER`, which contains information about the server. For example, `$_SERVER['SERVER_NAME']` returns the name of the server. The names of super globals are always in capital letters.

To access the value of `$_SERVER['SERVER_NAME']`, we can use the `echo` statement. We can also concatenate a `<br>` tag to create a new line after each echo. When we refresh the web page, we see the server name, which in this case is "localhost".

We can access other information from `$_SERVER` as well. For example, `$_SERVER['REQUEST_METHOD']` returns the request method used to access the PHP page. When we refresh the page, we see "GET" because we used the GET method.

To access additional information, we can duplicate the `echo` statement and change the index of `$_SERVER`. For example, `$_SERVER['SCRIPT_FILENAME']` returns the path of the current file. We can also use `$_SERVER['PHP_SELF']` to get the path of the current file relative to the server.

These super globals can be useful when creating forms. For example, instead of hardcoding the action attribute of a form, we can use `$_SERVER['PHP_SELF']` to dynamically set the action to the current page. This way, even if the file name changes, we don't need to update the form.

In the upcoming videos, we will discuss two more super globals: `$_SESSION` and `$_COOKIE`. These super globals are used for managing user sessions and cookies respectively.

EITC/WD/PMSF PHP AND MYSQL FUNDAMENTALS DIDACTIC MATERIALS
LESSON: EXPERTISE IN PHP
TOPIC: SESSIONS

Sessions in web development are a way to carry over variables between different pages. Unlike cookies, sessions store data on the server rather than on the user's computer. By starting a session on a website, special session variables can be accessed between different pages.

To demonstrate how sessions work, let's create a simple form where a user can enter their name. We will store this name in a session variable and then access it on different pages.

First, we create a form tag with an input field for the name and a submit button. The action of the form will be set to the current page.

Next, we handle the post request on the page. We check if the form has been submitted using the `isset` function and the `$_POST` superglobal. If the form has been submitted, we start the session using `session_start()`. Then, we can access the session superglobal and add the name variable to it.

To test if the session variable is working, we can echo out the name on the same page.

Instead of echoing the name, we can redirect the user to another page using the `header` function.

To access the session variable on any page in the project, we include the session start function at the top of the file. Then, we can retrieve the session variable and store it in a local variable.

Finally, we can output the name in the template using an HTML list item tag.

By using sessions and session variables, we can keep track of data as a user navigates around a website on different pages until they close the webpage.

In PHP, sessions are a way to store and retrieve data across multiple pages or requests. Sessions are stored on the server and are associated with a unique session ID, which is usually stored in a cookie on the client's browser.

To start a session, we use the `session_start()` function. This function must be called before any output is sent to the browser. Once a session is started, we can store data in session variables using the `$_SESSION` superglobal array.

To store a value in a session variable, we simply assign a value to it. For example, if we want to store the user's name, we can do it like this:

```
1. $_SESSION['name'] = 'Mario';
```

To access the value stored in a session variable, we can simply use the session variable name. For example, to display the user's name, we can do it like this:

```
1. echo $_SESSION['name'];
```

To ensure security and prevent any potential issues, it is recommended to use the `htmlspecialchars()` function to sanitize any user input before storing it in a session variable. This function converts special characters to their HTML entities, preventing any potential cross-site scripting (XSS) attacks.

```
1. $_SESSION['name'] = htmlspecialchars($_POST['name']);
```

Session variables can be accessed from any page as long as the session is active. This means that even if we navigate to a different page, the session variables will still be available.

To override the value of a session variable, we can simply assign a new value to it. For example, if we want to

change the user's name to "Yoshi", we can do it like this:

```
1. $_SESSION['name'] = 'Yoshi';
```

To delete or unset a session variable, we can use the `unset()` function and pass the session variable as an argument. For example, if we want to delete the "name" session variable, we can do it like this:

```
1. unset($_SESSION['name']);
```

Alternatively, if we want to unset all session variables, we can use the `session_unset()` function. This function will remove all the session variables, effectively clearing the session.

```
1. session_unset();
```

It is important to note that once a session variable is unset, it cannot be accessed anymore. Trying to access an unset session variable will result in an error.

In addition to sessions, PHP also provides another mechanism for storing data across requests called cookies. There are some differences between sessions and cookies, which will be explained in future tutorials.

EITC/WD/PMSF PHP AND MYSQL FUNDAMENTALS DIDACTIC MATERIALS
LESSON: EXPERTISE IN PHP
TOPIC: NULL COALESCING

In the previous material, we learned about setting up sessions in PHP and passing session variables between pages. We also saw how to unset a session variable when a specific condition is met. However, we encountered an issue when trying to set the value of a variable to a session variable that has been unset. This resulted in an error message being displayed on the page. In this material, we will explore a solution to this problem using the null coalescing operator.

The null coalescing operator provides a concise and elegant way to set a variable to a default value when the original value is null or does not exist. The operator is represented by two question marks (??). Here's how it works: if the value on the left side of the operator is not null, it will be assigned to the variable; otherwise, the value on the right side of the operator will be assigned.

Let's take a closer look at an example. Suppose we have a session variable called "name" that we want to assign to a variable called "nameVariable". We can use the null coalescing operator to set a default value for "nameVariable" in case the session variable is null or has been unset. The code would look like this:

```
1. $nameVariable = $_SESSION['name'] ?? 'guest';
```

In this example, if the session variable "name" exists and is not null, its value will be assigned to "nameVariable". However, if the session variable is null or does not exist, the string 'guest' will be assigned to "nameVariable" as a fallback option.

By using the null coalescing operator, we can prevent any error messages from being displayed when trying to assign a value to a variable that may not exist anymore.

To illustrate this, let's consider the previous scenario where we unset the session variable "name" if the query string in the URL equals "no name". If we try to access the value of "name" after unsetting it, we would encounter an error. However, by using the null coalescing operator, we can set a default value for "name" that will be displayed instead of an error message.

For example, if we have the following code:

```
1. $name = $_SESSION['name'] ?? 'guest';
```

Even if the session variable "name" has been unset, the variable "name" will be assigned the value 'guest'. This ensures that when we output the value of "name" in our template, we will see "hello guest" instead of an error message.

The null coalescing operator is a useful tool in PHP that allows us to set default values for variables when the original value may not exist. By using this operator, we can avoid errors and provide fallback options for variables in our code.

EITC/WD/PMSF PHP AND MYSQL FUNDAMENTALS DIDACTIC MATERIALS
LESSON: EXPERTISE IN PHP
TOPIC: COOKIES

Cookies are a way to persist data and keep track of it between different pages on a website. Unlike sessions, which store data on the server, cookies are stored on the user's computer. When you visit a website, you may have noticed a cookie notification asking for your permission to store data on your computer in the form of cookies. This data is used to enhance your experience on the website, such as remembering if you've visited a particular page before or if you've added an item to your basket.

While sessions are often preferred for sensitive data as they keep the data hidden on the server, cookies have their own uses. They can be used for content marketing, where the website can tailor its content to users based on their past behavior. For example, if a user visits an e-commerce website for clothing and looks at men's clothes, a cookie can be dropped on their computer. The next time they visit, the website can show them similar items on the homepage, based on their past behavior.

To create a cookie, you need to use the function `setcookie()`. In the provided code example, a form with a select field for gender is used. When the form is submitted, a cookie is set for the selected gender. The `setcookie()` function takes three arguments: the name of the cookie, the value of the cookie (in this case, the selected gender), and the expiration time of the cookie. The expiration time is set to one day from the current time using the `time()` function and adding the number of seconds in a day (86400).

To retrieve the cookie on another page, you can use the `$_COOKIE` superglobal variable. In the provided code example, the gender cookie is retrieved using `$_COOKIE['gender']` and stored in a variable called `gender`. If the cookie is not set, a fallback option of "unknown" is used. The gender value is then outputted in the website header using HTML brackets.

Cookies are a useful tool in web development for persisting and tracking data between different pages on a website. They can enhance user experience and enable personalized content based on past behavior.

Cookies are an essential part of web development, allowing websites to store and retrieve information from a user's computer. In this material, we will explore the fundamentals of creating and accessing cookies using PHP.

When a user interacts with a website, it can be useful to remember certain information about them, such as their preferences or settings. Cookies enable us to achieve this functionality. A cookie is a small piece of data that is stored on the user's computer by the web browser. It can contain various types of information, such as user preferences or session identifiers.

To create a cookie in PHP, we use the setcookie() function. This function takes several parameters, including the name of the cookie, its value, and optional parameters such as expiration time and path. In the provided example, the cookie is created with the name "gender" and the value "female". This means that the website will remember the user's gender preference as "female".

When we submit the form again, the cookie is reset with the new value. In this case, the value is overridden with "female" when the form is submitted. This means that the previous value of the cookie is replaced with the new value.

To access the value of a cookie, we can use the $_COOKIE superglobal variable in PHP. In the given example, we check the value of the "gender" cookie. If a cookie with that name exists, we can retrieve its value using $_COOKIE['gender']. This allows us to access and utilize the stored information as needed.

It is important to note that cookies are stored on the user's computer and can be accessed by the server on subsequent requests. This means that cookies can be used to personalize the user experience and remember user preferences across multiple visits to a website.

Cookies are a valuable tool in web development for storing and retrieving information from a user's computer. By creating and accessing cookies using PHP, we can enhance the functionality and personalization of our websites.

EITC/WD/PMSF PHP AND MYSQL FUNDAMENTALS DIDACTIC MATERIALS
LESSON: WORKING WITH FILES IN PHP
TOPIC: FILE SYSTEM - PART 1

In the previous materials, we discussed superglobals, sessions, and cookies in PHP. Now, we will shift our focus to communicating with the file system using PHP. PHP has the capability to interact with files on our computer or on a server. To demonstrate this, we have removed the project files and left only the sandbox PHP page and a readme text file.

Interacting with the file system in PHP is straightforward. In this first part, we will explore different operations we can perform on a file, such as finding out the file size. In the next part, we will learn a more efficient way to read and write to files.

A simple way to read a file is by using the `readfile` function. We can store the contents of the file in a variable, for example, `$quotes = readfile("readme.txt");`. When we echo this variable, the content of the file will be displayed in the browser. Additionally, the number displayed at the end represents the number of bytes in the file.

Although `readfile` provides a quick and easy way to read a file, there is a better method that allows us to open a file and keep a reference to it. This approach offers more options than the `readfile` function. We will cover this in the next part.

Sometimes, we may encounter a situation where we attempt to read a file that does not exist. To handle this, it is good practice to check if the file exists before performing any operations on it. We can accomplish this using an `if` statement and the `file_exists` function. If the file exists, we can read its contents. If it does not exist, we can display an appropriate message.

In addition to reading files, we can also copy them. The `copy` function allows us to copy a file by specifying the source file and the destination file name. PHP will create the destination file for us.

Another useful operation is finding the absolute or real path of a file. We can achieve this using the `realpath` function. By passing the file as an argument to `realpath`, we can obtain the absolute path of the file.

Additionally, we can determine the file size using the `filesize` function. This function returns the size of the file in bytes.

Lastly, we can rename a file using the `rename` function. By providing the current file name and the desired new name, we can rename the file.

In the next part, we will explore more advanced file operations in PHP.

In PHP, there are various functions that allow us to interact with the file system. These functions enable us to perform tasks such as reading, copying, finding the path and size of a file, renaming a file, and creating new directories.

To read the content of a file, we can use the file_get_contents() function. This function takes the file name as a parameter and returns the contents of the file as a string. For example, to read the contents of a file named "test.txt", we can use the following code:

```
1. $fileContents = file_get_contents("test.txt");
```

To copy a file, we can use the copy() function. This function takes two parameters: the source file and the destination file. It creates a copy of the source file at the specified destination. Here's an example:

```
1. copy("source.txt", "destination.txt");
```

To find out the path of a file, we can use the realpath() function. This function takes the file name as a parameter and returns the absolute path of the file. Here's an example:

```
1. $path = realpath("test.txt");
```

To get the size of a file, we can use the filesize() function. This function takes the file name as a parameter and returns the size of the file in bytes. Here's an example:

```
1. $size = filesize("test.txt");
```

To rename a file, we can use the rename() function. This function takes two parameters: the current file name and the new file name. It renames the file accordingly. Here's an example:

```
1. rename("old.txt", "new.txt");
```

In addition to working with files, we can also create new directories using the mkdir() function. This function takes the name of the directory as a parameter and creates a new directory with the specified name. Here's an example:

```
1. mkdir("new_directory");
```

By using these simple and basic functions, we can easily interact with the file system in PHP. In the next material, we will explore a more advanced way of opening and reading files, as well as writing to files using the F open method.

EITC/WD/PMSF PHP AND MYSQL FUNDAMENTALS DIDACTIC MATERIALS
LESSON: WORKING WITH FILES IN PHP
TOPIC: FILE SYSTEM - PART 2

To work with files in PHP, we can open a file and keep a reference to it in a variable. This allows us to have more options than just using the read file function. To open a file, we use the fopen function and pass in the file name as the first argument and the mode as the second argument. In this case, we use the 'r' mode, which means read-only.

To read from the file, we have a few options. One way is to use the fread function. We pass in the file handle as the first argument and the number of bytes we want to read as the second argument. If we want to read the entire file, we can use the filesize function to get the size of the file and pass that as the second argument to fread.

Another way to read from the file is to use the fgets function, which reads a single line from the file. We pass in the file handle as the argument. Each time we call fgets, it reads the next line from the file.

We can also read individual characters from the file using the fgetc function. Again, we pass in the file handle as the argument. Each time we call fgetc, it reads the next character from the file.

If we want to write to the file, we need to open it in write mode. Currently, we have opened the file in read-only mode, so we cannot write to it. To open the file in write mode, we would use the 'w' mode. This will overwrite the existing contents of the file. If we want to append to the file instead of overwriting it, we can use the 'a' mode.

To work with files in PHP, we can open a file using the fopen function and keep a reference to it in a variable. We can then use functions like fread, fgets, and fgetc to read from the file. If we want to write to the file, we need to open it in write mode using the 'w' mode.

In PHP, there are different ways to read and write to files. One option is to use the `fopen` function to open a file and then use the `fwrite` function to write to it. The first argument of `fwrite` is the file handle, and the second argument is the content you want to write. For example, you can use the `"\n"` escape character to create a new line and then write the desired content.

To demonstrate this, let's consider an example where we want to write the phrase "Everything popular is wrong" to a file. We can start by opening the file using the `fopen` function and specifying the file name and the mode as "w" (write). Then, we can use `fwrite` to write the content to the file. After that, we can close the file using the `fclose` function.

```
1. $file = fopen("quotes.txt", "w");
2. fwrite($file, "Everything popular is wrong\n");
3. fclose($file);
```

If we run this code and refresh the browser, the file "quotes.txt" will be created, and the phrase "Everything popular is wrong" will be written to it.

Now, what if we want to append the content to the end of the file instead of overwriting it? In that case, we can use the mode "a" (append) instead of "w" when opening the file. This will place the file pointer at the end of the file, allowing us to write new content without overwriting the existing content.

```
1. $file = fopen("quotes.txt", "a");
2. fwrite($file, "Everything popular is wrong\n");
3. fclose($file);
```

If we run this code and refresh the browser, the phrase "Everything popular is wrong" will be added to the end of the file.

It's important to note that when working with files, it's good practice to close the file after you're done using it.

This can be done using the `fclose` function, passing in the file handle as the argument.

```
1. fclose($file);
```

Lastly, if you need to delete a file, you can use the `unlink` function and provide the file name as the argument.

```
1. unlink("quotes.txt");
```

This will delete the file named "quotes.txt" from the file system.

PHP provides functions like `fopen`, `fwrite`, `fclose`, and `unlink` to work with files. These functions allow you to open files, write content to them, close files, and even delete files if needed.

EITC/WD/PMSF PHP AND MYSQL FUNDAMENTALS DIDACTIC MATERIALS
LESSON: CLASSES AND OBJECTES IN PHP
TOPIC: CLASSES AND OBJECTS - PART 1

In PHP, we have the ability to create our own custom data types or objects with specific properties and functions. This allows us to create more complex and specialized data structures for our applications.

To create a custom data type, we use what is called a class. A class is like a blueprint for an object, describing what properties and functions an object of that type will have. We can then create an object based on that class, known as instantiating a class.

To create a class, we use the "class" keyword followed by the name we want to give to the class. Class names typically start with a capital letter. Inside the class, we define the properties and functions that our objects will have.

Properties are variables that hold data specific to an object. We can define properties as public or private. Public properties can be accessed and changed from anywhere outside the class, while private properties can only be accessed from inside the class itself. It is good practice to set properties to private to protect them from being accessed or modified directly.

Functions, also known as methods, are actions that an object can perform. We can define functions as public or private as well. Public functions can be called from outside the class, while private functions can only be called from inside the class.

Let's take an example of a user data type. We want our user objects to have an email property and a name property, as well as a login function. We will set these properties to public for now.

To access the properties and functions of an object, we use the object variable followed by an arrow "->" and then the property or function we want to access. For example, to access the login function of a user object named "user1", we would use "user1->login()".

Now, let's create a new object based on our user class. We use the "new" keyword followed by the class name to instantiate the class and create a new object. We can store this object in a variable, such as "user1".

We can then access the properties and functions of the user object using the arrow syntax mentioned earlier. For example, "user1->email" would give us the value of the email property.

In our example, we have created a user object and called the login function, which simply echoes "User logged in". If we were to echo the value of the email property, it would not display anything as we have not assigned a value to it yet.

To summarize, in PHP, we can create custom data types or objects by defining a class. Classes serve as blueprints for objects, describing their properties and functions. Objects are created by instantiating a class using the "new" keyword. Properties and functions of objects can be accessed using the arrow syntax.

A constructor function is a special type of function inside classes that runs whenever we instantiate a class. It is used to set initial values for the properties of the class. To create a constructor function, we use the keyword "public" followed by the function name "__construct". The constructor function can access the properties of the class using the keyword "$this".

For example, let's say we have a class called "User" with two properties: "name" and "email". We can set initial values for these properties in the constructor function. To do this, we use "$this->propertyName = value".

In the given example, the constructor function sets the initial values of the "name" and "email" properties to "Mario" and "thenetninjacody@uk" respectively. When we create a new object of the class, these initial values are assigned to the properties.

However, setting the initial values in the constructor function may not be ideal if we want to have different

initial values for different objects. In such cases, we can pass the values as parameters to the constructor function when creating a new object.

In the example, a new object "user2" is created with the values "Yoshi" and "thenetninjacody@uk" for the "name" and "email" properties respectively. These values are passed as parameters to the constructor function. Inside the constructor function, the received values are assigned to the properties using "$this->propertyName = parameterName".

We can access the values of the properties using "$this->propertyName". In the example, the values of the "name" and "email" properties of "user2" are echoed using the statement "echo $user2->name" and "echo $user2->email".

It is worth noting that it is common practice to make properties private instead of public. This is to prevent them from being updated directly from outside the class.

By using constructor functions and setting initial values for properties, we can create objects with different initial values and access those values within the class.

In the previous material, we discussed the fundamentals of classes and objects in PHP. Now, we will continue our exploration of this topic.

In object-oriented programming (OOP), a class is a blueprint for creating objects. It defines the properties and behaviors that an object of that class will possess. Objects, on the other hand, are instances of a class. They are the actual entities that can be manipulated and interacted with.

To create a class in PHP, we use the `class` keyword followed by the class name. The class name should be meaningful and descriptive, following proper naming conventions. Inside the class, we define the properties and methods that the objects of that class will have.

Properties are variables that hold data specific to each object. They can be public, protected, or private, depending on their visibility. Public properties can be accessed and modified from outside the class, while protected and private properties have restricted access.

Methods, also known as functions, are actions or behaviors that an object can perform. They are defined within the class and can access the class's properties. Like properties, methods can also have different visibility levels.

To create an object from a class, we use the `new` keyword followed by the class name and parentheses. This will allocate memory for the object and initialize its properties. We can then access the object's properties and methods using the object's name followed by the arrow operator (`->`).

In PHP, we can also define constructors and destructors for classes. A constructor is a special method that is automatically called when an object is created. It is used to initialize the object's properties or perform any necessary setup. On the other hand, a destructor is called when an object is no longer needed and is about to be destroyed. It is used to clean up any resources or perform final actions.

Classes and objects are essential concepts in object-oriented programming. Classes define the structure and behavior of objects, while objects are the actual instances that can be manipulated. Understanding how to create and use classes and objects is crucial for building robust and maintainable PHP applications.

EITC/WD/PMSF PHP AND MYSQL FUNDAMENTALS DIDACTIC MATERIALS
LESSON: CLASSES AND OBJECTES IN PHP
TOPIC: CLASSES AND OBJECTS - PART 2

In PHP, classes and objects are fundamental concepts in object-oriented programming. In the previous lesson, we created a user class with two properties: email and name. However, these properties were declared as public, which means they can be accessed and changed from outside the class.

To demonstrate this, let's create a new user object and set the initial name to "Yoshi". If we echo out the name property, we will see "Yoshi" as expected. Now, if we want to update the name property, we can do so anywhere in our code. For example, we can set the name of the user object to "Mario". When we echo out the name, we will see "Mario" instead of "Yoshi".

While there is nothing inherently wrong with directly updating properties like this, it is not considered good practice. It allows for potential misuse or invalid values. For example, we could update the name property to a number like 50, which is not a valid name. To prevent such issues, a better approach is to make the properties private. This means they can only be accessed and modified through special class functions known as getters and setters.

To implement this, we need to change the visibility of the properties from public to private. Now, if we try to access the properties directly, we will get an error stating that we cannot access private properties.

Instead of accessing the properties directly, we can create a public function called getName that will return the value of the name property. Inside this function, we use the keyword "this" to refer to the current object and access its name property. By calling this function on the user object, we can retrieve the name value.

To update the name property, we need to create a public function called setName that takes a parameter representing the new name value. Inside this function, we can perform validation before updating the name property. In our example, we check if the passed value is a string and if its length is greater than 1 character. If the validation passes, we update the name property with the new value.

By using getters and setters, we can ensure that the properties are accessed and modified in a controlled manner, allowing for validation and other custom logic if needed.

In PHP, we can output variables directly inside strings by using the concatenation operator (.) or by enclosing the variable in curly braces ({}) and prefixing it with a dollar sign ($). We can also pass in the variable as an argument when calling a function.

In the given example, we have a function called "set_name" that takes in a name as an argument. This function updates the value of the private property "name" if the passed name is valid. If the name is not valid, the function returns the string "not a valid name".

To demonstrate this, we create an instance of a class called "User" and call the "set_name" function, passing in a name. If the name is not valid, we echo the returned value. In this case, the name "50" is not a valid name, so the function returns "not a valid name" and we echo this out.

If we want to access the updated name, we use the function "get_name" instead of directly accessing the private property "name". This is because the property is set to private, meaning it can only be accessed from within the class itself. We echo the returned value of "get_name" to see the updated name.

By setting the property to private and providing public methods to access and update it, we are protecting the object and providing controlled entry points for interacting with the data.

It is also mentioned that similar validation and manipulation can be done for other properties, such as email, if desired. However, this is not demonstrated in the example.

This concludes the series on objects and classes in PHP. The focus of this series was to introduce the concept of objects to beginners. Advanced topics such as object-oriented PHP inheritance were not covered extensively.

If there is enough interest, the speaker mentions the possibility of creating a mini-series on object-oriented PHP in the future, which would go into more detail.