Unit 2: Control and Looping Statements

Unit Structure

2.1 LEARNING OBJECTIVE

After studying this unit student should be able to:

- Understand various control Structure statements.
- Understand various Looping statements.
- Understand advance program flow statements.

2.2 INTRODUCTION

One of the main reasons for using scripting languages such as PHP is to build logic and intelligence into the creation and deployment of web-based data.

In order to be able to build logic into PHP based scripts, it is necessary for the script to be able to make decisions and repeat tasks based on specified criteria.

For example, it may be necessary to repeat a section of a script a specified number of times, or perform a task only if one or more conditions are found to be true (i.e. only let the user log in if a valid password has been provided).

The if, if…else and if…elseif…else construct is one of the most important features of many languages. These conditional statements provide us different actions for the different conditions. When we write code, we perform different actions for different decisions. Like any other languages, PHP is built out of a series of control statements. The control statement can be an assignment, a function call, a loop, a conditional statement or even a statement that does nothing or an empty statement.

In programming terms this is known as flow control and looping. In the simplest terms this involves some standard scripting structures provided by languages such as PHP to control the logic and overall behaviour of a script. Each of these structures provides a simple and intuitive way to build intelligence into scripts. These PHP structures can be broken down into a number of categories as follows:

Conditional Statements

- if statements: it is a control statement, we execute some code only if a specified condition is true.
- if ... else ... statements: We use this control statement to execute some code if a condition is true and another code if the condition is false.
- if…elseif…. else statement – We use this control statement to select one of the several blocks of code to be executed.
- switch statement – We use this control statement to select one of many blocks of code to be executed.

Looping Statements

- For loops
- while loops
- do ... while loops

In this unit we will explore each of these conditional statements and create some examples of that, demonstrate how to implement these control statements.

2.3 CONTROL STATEMENT: IF, IF_ELSE

What Is a Control Statement?

In simple terms, a control Statement allows you to control the flow of code execution in your application. Generally, a program is executed sequentially, line by line, and a control structure allows you to alter that flow, usually depending on certain conditions.

Control structures are core features of the PHP language that allow your script to respond differently to different inputs or situations. This could allow your script to give different responses based on user input, file contents, or some other data.

The following flowchart explains how a control structure works in PHP.

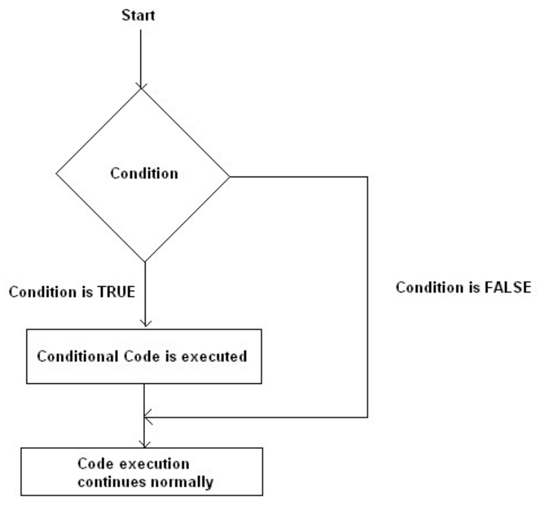

Start
Condition
Condition is TRUE
Condition is FALSE
Conditional Code is executed
Code execution continues normally

Figure 10 Simple Control Structure with single action

As you can see in the above diagram, first a condition is checked. If the condition is true, the conditional code will be executed. The important thing to note here is that code execution continues normally after conditional code execution.

Let's consider the following example.

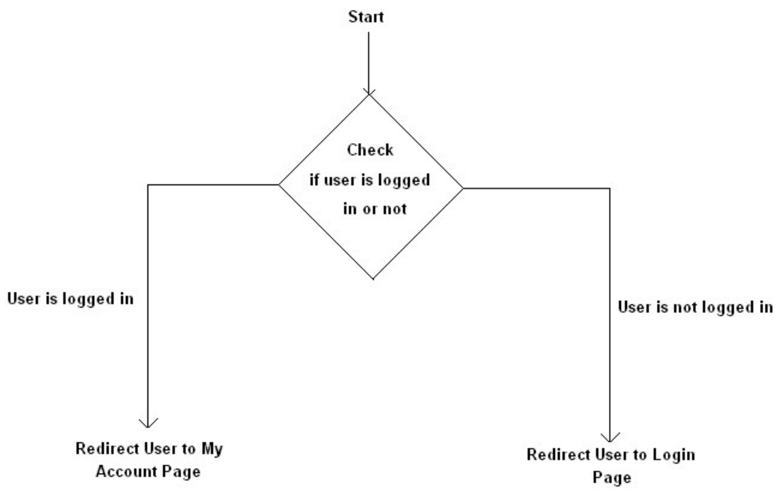

Start
Check if user is logged in or not
User is logged in
User is not logged in
Redirect User to My Account Page
Redirect User to Login Page

Figure 11 Control Structure with two action redirect

In the above example, the program checks whether or not the user is logged in. Based on the user's login status, they will be redirected to either the Login page or the My Account page. In this case, a control structure ends code execution by redirecting users to a different page. This is a crucial ability of the PHP language.

The if Statement (also called as control statement)
Use the if statement to execute some code only if a specified condition is true.

The expression is evaluated to its Boolean value. If expression evaluates to TRUE, PHP will execute the statement, and if it evaluates to FALSE, the statement will not be executed.

Syntax of If:

```
    if (condition) {
    code to be executed if condition is true;
    }
```

The following example would display” A is bigger than B” if $a is bigger than $b:

```
<?php
    $a=10; $b=5;
    if ($a > $b)
    {
```

```
                echo "A is bigger than B";
        }
?>
```

The if…else Statement:

If…else, as its name suggests, is a combination of if and else. It extends an if statement to execute a different statement (conditions in the else statements) in case the original if expression is FALSE.

Syntax of If-else:

```
        if (condition) {
               code to be executed if condition is true;
        }
        else {
               code to be executed if condition is false;
        }
```

For example, the following code would display a is bigger than b or a is smaller than b:

```
<?php
        $a=21;
        $b=12;
        if ($a > $b) {
               echo "a is bigger than b";
        }
        else {
               echo "a is smaller than b ";
        }
?>
```

Here, The output is "a is bigger than b"

2.4 IF…ELSEIF…ELSE, NESTED IF

The if…elseif…. else Statement

Use the if…. elseif…else statement to select one of several blocks of code to be executed. We can create multiple branches using the elseif keyword.
The elseif keyword tests for another condition if and only if the previous condition was not met. Note that we can use multiple elseif keywords in our tests.

Syntax if…elseif….else:

```
if (condition) {
        code to be executed if this condition is true;
} elseif (condition) {
        code to be executed if this condition is true;
} else {
        code to be executed if all conditions are false;
}
```

For example, the following code would display a is bigger than b, a equal to b or a is smaller than b:

```
<?php
        $a=9;
        $b=12;

        if ($a > $b) {
                echo "a is bigger than b";
        } elseif ($a == $b) {
                echo "a is equal to b";
        } else {
                echo "a is smaller than b";
        }
?>
```

Nested if statements in PHP

Nested if statements mean an if block inside another if block. Shortly a control structure inside another control structure.

Syntax of Nested If:

```
if (expression 1)
{
        if (expression 2)
        {
                // statements 1
        }
        else
        {
                // Statements 2
        }
}
else
{
        if (expression 2)
        {
                // Statements 3
        }
        else
        {
                // Statements 4
        }
}
```

Here we can see another if ... else structure inside the if block and else block. Like this we can add any number of nested if else statements.

2.5 CONTROL STATEMENT: SWITCH STATEMENT

The Switch Statement:

The switch statement is a selection control flow statement. It allows the value of a variable or expression to control the flow of program execution via a multiway branch. It creates multiple branches in a simpler way than using the if, elseif statements.

The switch statement works with two other keywords: case and break. The case keyword is used to test a label against a value from the round brackets. If the label equals to the value, the statement following the case is executed. The break keyword is used to jump out of the switch statement. There is an optional default statement. If none of the labels equals the value, the default statement is executed.

For example, the variable $domains_country_extension may have the 'mu' string for Mauritius. We can use the switch statement to test for the value of the variable. There are several options. If the value equals to 'uk' as a string, the 'England' string is printed to the console.

```
<?php
$domain_Name = 'open.ac.mu';
$domain = 'mu';
echo "<b>$domain_Name</b> domain extension is <b>$domain</b><br>Switch
Case output is: ";
switch ($domain) {
case 'mu':
                echo "Mauritius\n";
                break;
case 'uk':
                echo "England\n";
                break;
case 'us':
        echo "United States\n";
                break;
case 'de':
                echo "Germany\n";
                break;
case 'au':
        echo "Australia\n";
                break;
        case 'ca':
                echo "Canada\n";
                break;
default:
        echo "Unknown\n";
                break;
}
?>
```

If we changed the *$domains* variable to 'ca' then we get 'Canada'. If we changed the *$domains* variable to 'my', we would get 'Unknown'.

2.6 LOOPING STATEMENT: WHILE, DO … WHILE, FOR, FOREACH

Loops:

Loop statements are the primary mechanism for telling a computer to perform the same task over and over again until a set of criteria are met.

In PHP, we have the following looping statements:

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

The while Loop

The while loop is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition. The while loop executes a block of code as long as the specified condition is true.

Syntax of while loop:

```
while (condition is true) {
code to be executed;
}
```

The while loop executes the statement when the expression is evaluated to true. The statement is a simple statement terminated by a semicolon or a compound statement enclosed in curly brackets. See example below:

```
<?php

$a = 1;

While ($a <= 4) {

        echo "B.Sc. - Computer Science. Semester: $a <br>";

        $a++;

}

?>
```

The example above first sets a variable $a to 1 ($a = 1). Then, the while loop will continue to run as long as $a is less than, or equal to 4 ($a<= 4). $x will increase by 1 each time the loop runs ($a++).

The do...while Loop

The do...while loop will always execute the block of code once, it will then check the condition, and repeat the loop while the specified condition is true. Do While Loop is very similar to while loop. The difference is, it check the condition at the end of the block and while loop check the condition at the beginning of the block. It means do while will execute the condition at least once, whether the condition is true or false.

Syntax of do...while Loop

```
do {
        code to be executed;
} while (condition is true);
```

Note: The do while loop is a version of the while loop. The difference is that this version is guaranteed to run at least once. See example below:

```
<?php
$a = 1;
do {
        echo "The number is: $a <br>";
        $a++;
} while ($a <= 5);
?>
```

The example above first sets a variable $a to 1 ($a = 1). Then, the do while loop will write some output, and then increment the variable $a with 1. Then the condition is checked (is $a less than, or equal to 5?), and the loop will continue to run as long as $a is less than, or equal to 5:

Note: Notice that in a do while loop the condition is tested AFTER executing the statements within the loop. This means that the do while loop would execute its statements at least once, even if the condition is false the first time, as shown in example below:

```
<?php
$a = 1;
do {
        echo "The number is: $a <br>";
        $a++;
} while ($a >= 5);
?>
```

The example above first sets a variable $a to 1 ($a = 1). Then, the do while loop will write some output, and then increment the variable $a with 1. Then the condition is checked (is $a greater than, or equal to 5?). Since $a is less than 5, the condition is false and therefore, the loop will stop. The output will be 1 in this case since the do while loop is run at least once.

The for Loop

For loop in PHP is the most complicated loop. We use this loop when we know how many times the code should run.

Syntax of for Loop

```
for (Init counter; Test counter; Increment/Decrement counter) {
 code to be executed;
}
```

Parameters:

- Init counter: Initialize the loop counter value (example, $i=1)
- Test counter: Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends. (example, $i<=5)
- Increment/Decrement counter: Increases/Dacres the loop counter value ($i++)

The example below displays the numbers from 1 to 5:

```
<?php
echo "for loop output as below:<br>";
for ($i=1; $i<=5; $i++) {
	echo "$i ";
}
?>
```

In the above code value of $i was set to 1 and printed. Each time it was printed, the value of $i is incremented by 1 and is checked again whether it is <= 5.

The foreach Loop

The foreach loop works on arrays and is used to loop through each key/value pair in an array. It is also used when you don’t know the number of iterations in an array. In this case, the foreach loop helps to print all elements of an array without length calculation. (Majority it Is used in print select query result set operation)

Syntax of foreach loop is as below:

```
foreach (array_expression as $value)
{
        Statement

}
```

For every loop iteration, the value of the current array element is assigned to $value and an array pointer is moved by one, until it reaches the last array element.

The following example demonstrates a foreach loop that will output the values of the given array ($course):

```
<?php
$course = array("B.Sc. Computer Science", "M.Sc. Computer Science", "B.Sc. AI",
"M.Sc. AI");
foreach ($course as $value) {
        echo "Open University Offer $value Course <br>";
}
?>
```

The usage of the foreach statement is straightforward. The $course is the array that we iterate through. The $value is the temporary variable that has the current value from the array. The foreach statement goes through all the course and prints them.

2.7PHP BREAK, CONTINUE STATEMENTS

break statements

The break statement is used to terminate the loop. The continue statement is used to skip a part of the loop and continue with the next iteration of the loop.

```
<?php
while (true) {
        $val = rand(1, 30);
        echo $val." ";
        if ($val == 20) {
                break;
        }
}
echo "\n"; ?>
```

Here, an endless while loop is defined. There is only one way to jump out of a such loop — using the break statement. We choose a random value from 1 to 30 and print it. If the value equals to 20, we break the endless while loop.

PHP continue Statement

Sometimes a situation arises where we want to take the control to the beginning of the loop, skipping the rest statements inside the loop which have not yet been executed.

The keyword continues allow us to do this. When the keyword continue is executed inside a loop, the control automatically passes to the beginning of loop. Continue is usually associated with the if.

Example:

Suppose we want to print the list of odd numbers between 1 to 10. The remainder ($x%2) of every number between 1 to 10 is tested in the while loop. If the remainder is 0 then it becomes an even number. To avoid printing even numbers, the continue statement is immediately applied and the control passes to the beginning of the loop.

```
<?php
$x=1;
echo 'List of odd numbers between 1 to 10 <br />';
while ($x<=10) {
        if (($x % 2)==0) {
                $x++;
                continue;
        }
        else {
                echo $x.'<br />';
                $x++;
        }
}
?>
```

2.8 ADVANCE PROGRAM FLOW STATEMENT

Try Catch Example: Exception & Error Handling

What is an Exception?

Error is an unexpected program result that cannot be handled by the program itself. Errors are resolved by fixing the program. An example of an error would be an infinite loop that never stops executing. Another example of an exception includes trying to open a file that does not exist. This exception can be handled by either creating the file or presenting the user with an option of searching for the file.

In this chapter, you will learn-

- Why handle exception?
- PHP Error handling
- Error handling examples
- Difference between Errors and Exception
- Multiple Exceptions
- Testing the code

Why handle exception?

- Avoid unexpected results on web pages which can be very annoying or irritating to end users.
- Improve the security of web applications by not exposing information which malicious users may use to attack our applications.

PHP Exceptions are used to change the normal flow of a program if any predictable error occurs.

PHP Error handling

When an error occurs, depending on your configuration settings, PHP displays the error message in the web browser with information relating to the error that occurred. PHP offers a number of ways to handle errors. Three commonly used methods are:

- **Die statements**: The die statement combines the echo and exit function in one. It is very useful when we want to output a message and stop the script execution when an error occurs.
- **Custom error handlers**: These are user defined functions that are called whenever an error occurs.
- **Error reporting**: The error message depending on your PHP error reporting settings. This method is very useful in development environment when you have no idea what caused the error. The information displayed can help you debug your application.

Error handling examples

Let’s now look at some simple examples with error handling routines.

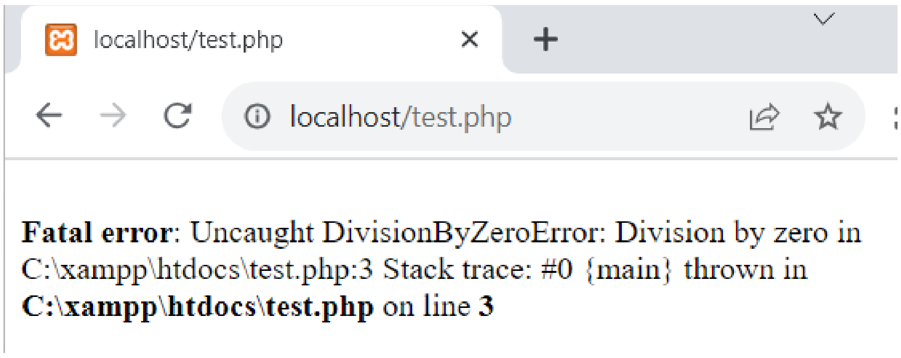
Let’s suppose that we have developed an application that performs division. Now suppose that the number is divided by 0, as shown in the following code:

```
<?php

$denominator = 0;

echo 2/$denominator;

?>
```



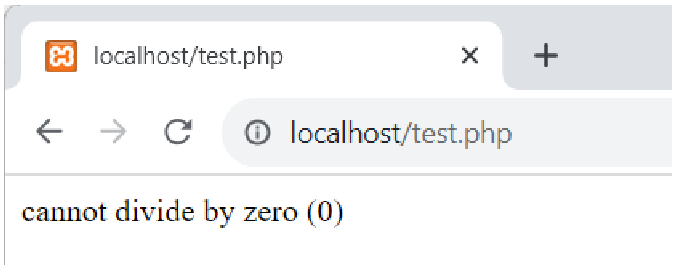
```
Fatal error: Uncaught DivisionByZeroError: Division by zero in C:\xampp\htdocs\test.php:3 Stack trace: #0 {main} thrown in C:\xampp\htdocs\test.php on line 3
```

As you can see from the above results, it makes our application look unprofessional and can be annoying to the user.

Let's modify the above code and write an error handler for the application

```
<?php
$denominator = 0;

if ($denominator != 0) {
	echo 2/$denominator;
} else {
	echo "cannot divide by zero (0)";
}
?>
```



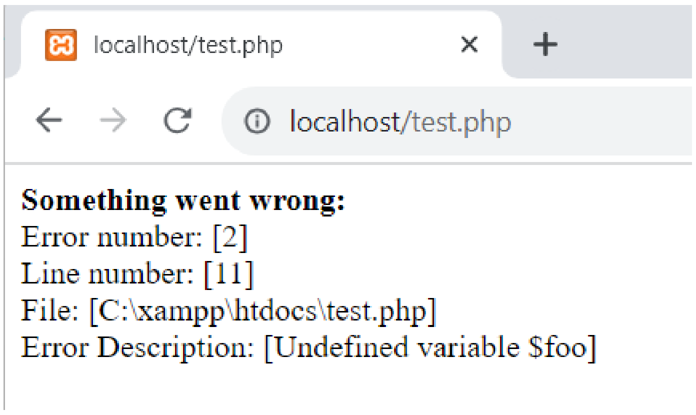
```
cannot divide by zero (0)
```

Custom error handling examples

Let's discuss another example that uses a custom error handler. The custom error handler will be set as the default PHP error handling function and will basically display the error number, the line number, the file name including directories and message. The code below illustrates the implementation:

```
<?php
   function on_error($num, $str, $file, $line) {
        echo "<b>Something went wrong:</b><br>";
        echo "Error number: [$num]<br>";
        echo "Line number: [$line]<br>";
        echo "File: [$file] <br>";
        echo "Error Description: [$str] <br>";
   }

   set_error_handler("on_error");
   print $foo;
?>
```



```
Something went wrong:
Error number: [2]
Line number: [11]
File: [C:\xampp\htdocs\test.php]
Error Description: [Undefined variable $foo]
```

As you can see from the above example, custom error handlers are powerful in the sense that:

- They allow us to customize the error messages.
- The custom error handler can also include error logging in a file/database, emailing the developer etc.

PHP Error reporting

Let’s now look at the third type of error handling. We will be using the PHP built in function error_reporting function. It has the following basic syntax

```
<?php
error_reporting($reporting_level);
?>
```

- "error_reporting" is the PHP error reporting function
- "$reporting_level" is optional, can be used to set the reporting level. If no reporting level has been specified, PHP will use the default error reporting level as specified in the php.ini file.

Reporting Level	Description	Example
E_WARNING	Displays warning messages only. Does not halt the execution of the script	error_reporting(E _WARNING);
E_NOTICE	Displays notices that can occur during normal execution of a program or could bean error.	error_reporting(E _ NOTICE);
E_USER_ERROR	Displays user generated errors i.e. custom error handler	error_reporting(E _ USER_ERROR);
E_USER_WARNI NG	Displays user generated warning messages	error_reporting(E _USER_WARNING);
E_USER_NOTICE	Displays user generated notices	error_reporting(E _USER_NOTICE);
E_RECOVERAB LE_ERROR	Displays error that are not fatal and can behandled using custom error handlers	error_reporting(E _RECOVERABL E_ERROR);
E_ALL	Displays all errors and warnings	error_reporting(E _ ALL);

Difference between Errors and Exception

- Exceptions are thrown and intended to be caught while errors are generally irrecoverable.
- Exceptions are handled in an object-oriented way. This means when an exception is thrown; an exception object is created that contains the exception details.

The table below shows the exception object methods.

Method	Description	Example
getMessage()	Displays the exception's message	<?php echo $e->getMessage(); ?>
getCode()	Displays the numeric code that represents the exception	<?php echo $e->getCode(); ?>
getFile()	Displays the file name and path where the exception occurred	<?php echo $e->getFile(); ?>
getLine()	Displays the line number where the exception occurred	<?php echo $e->getLine(); ?>
getTrace()	Displays an array of the backtrace before the exception	<?php print_r($e->getTrace()); ?>
getPrevious()	Displays the previous exception before the current one	<?php echo $e->getPrevious(); ?>
getTraceAsString()	Displays the backtrace of the exception as a string instead of an array	<?php echo $e- >getTraceAsString(); ?>
__toString()	Displays the entire exception as a string	<?php echo $e-> toString(); ?>

Below is the basic syntax for throwing an exception.

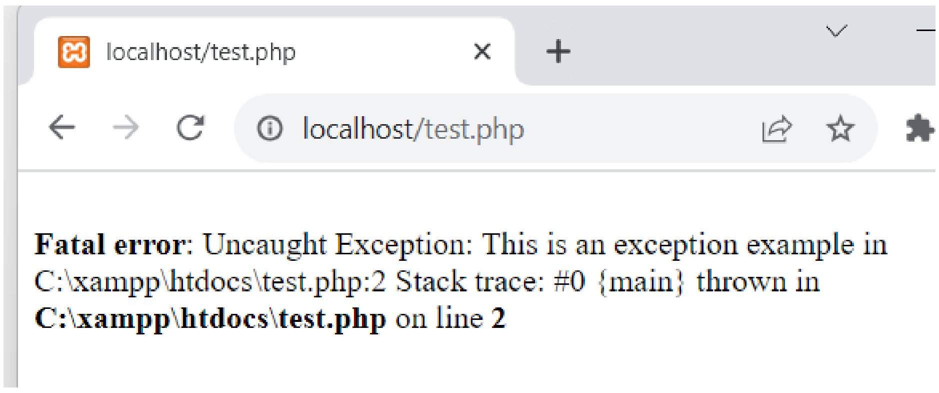
```
<?php   // Script  save  as  PHP_Throw_exception.php  throw
        new Exception("This is an exception example");
?>
```

In the above example:

“throw” is the keyword used to throw the exception

“new Exception(…)” creates an exception object and passes “This is an exception example “ string as the message parameter.

The output for the above code is:



```
Fatal error: Uncaught Exception: This is an exception example in
C:\xampp\htdocs\test.php:2 Stack trace: #0 {main} thrown in
C:\xampp\htdocs\test.php on line 2
```

As you can see, the exception has not been caught. We will modify the above example and include the try, throw and catch exceptions. It has the following basic syntax of **try – catch:**

```
<?php
        try {
                //code goes here that could potentially throw an exception
        }
        catch (Exception $e) {
                //exception handling code goes here
        }
?>
```

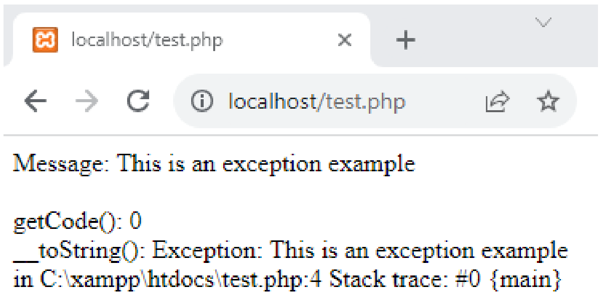
- “**try{…}**” is the block of code to be executed that could potentially raise an exception
- “**catch(Exception $e){…}**” is the block of code that catches the thrown exception and assigns the exception object to the variable $e.

The code below shows the basic exception example with the try, throw and catch exception implemented.

The program deliberately throws an exception which it then catches.

```
<?php
try {
   $var_msg = "This is an exception example <br>";
   throw new Exception($var_msg);
}
catch (Exception $e) {
   echo "Message: " . $e->getMessage();
   echo "<br>";
   echo "getCode(): " . $e->getCode();
   echo "<br>";
   echo "__toString(): " . $e->__toString();
}
?>
```

The output to the above code is as follows:



```
Message: This is an exception example

getCode(): 0
__toString(): Exception: This is an exception example
in C:\xampp\htdocs\test.php:4 Stack trace: #0 {main}
```

It is also possible to create multiple exceptions for one PHP try statement depending on the type of exception thrown.

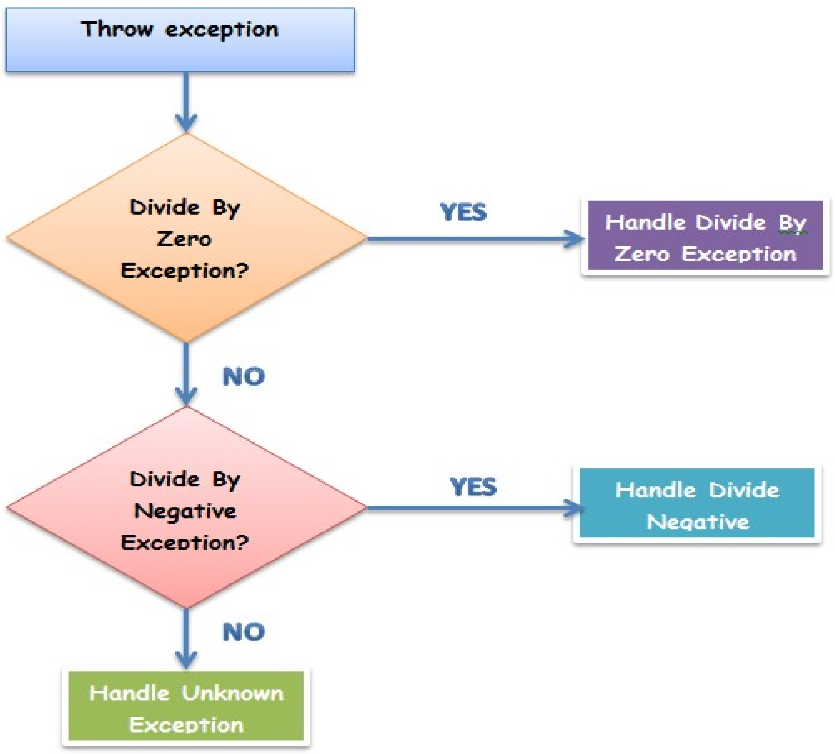
Multiple Exceptions

Multiple exception uses multiple try catch blocks to handle the thrown exceptions. Multiple exceptions are useful when,

- You want to display a customized message depending on the exception thrown.
- You want to perform a unique operation depending on the exception thrown.

The flowchart below illustrates the how multiple exceptions work

PHP Exception Handle in PHP


Throw exception
Divide By Zero Exception?
YES
Handle Divide By Zero Exception
NO
Divide By Negative Exception?
YES
Handle Divide Negative
NO
Handle Unknown Exception

PHP Multiple Exception Handling Flow

Let’s discuss an example that uses multiple exceptions.

We will modify the code that divides a number by the passed in denominator.

We expect two types of exceptions to occur;

- Division by zero
- Division by a negative number

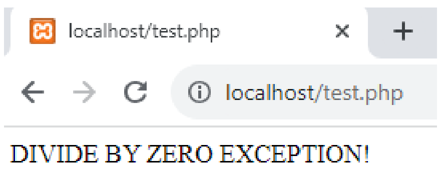
For simplicity’s sake, we will only display the exception type in our catch blocks.

The PHP built in Exception class is used to throw exceptions. We will create two classes that extend the exception class and use them to throw exceptions. The code below shows the implementation of multiple exception.

```
<?php
class DivideByZeroException extends Exception {};
class DivideByNegativeException extends Exception {};

function process($denominator)
{
        try
        {
                if ($denominator == 0)
                {
                        throw new DivideByZeroException();
                }
                else if ($denominator < 0)
                {
                        throw new DivideByNegativeException();
                }
                else
                {
                        echo 25 / $denominator;
                }
        }
        catch (DivideByZeroException $x)
        {
                echo "DIVIDE BY ZERO EXCEPTION!";
        }
        catch (DivideByNegativeException $x)
        {
                echo "DIVIDE BY NEGATIVE NUMBER EXCEPTION!";
        }
        catch (Exception $x)
        {
        echo "UNKNOWN EXCEPTION!";
        }
}
process(0);
?>
```

Since the value passed to the function “process();” is 0, the output will be as follows:



```
DIVIDE BY ZERO EXCEPTION!
```

You can play with the function by inputting a positive and a negative number in the code process() to see the outcome.

Summary on Exception & Error Handling

- Errors are unexpected results produced by PHP code.
- Error handling improves the application performance.
- PHP has built in functions that can be used to customize the way PHP reports errors.
- Exceptions are like errors, but they can be caught using the catch block when thrown.
- Displaying error messages that show error information is considered a bad security practice.