KUMASI TECHNICAL UNIVERSITY

COMPUTER SCIENCE DEPARTMENT

BACK – END WEB DEVELOPMENT TECHNOLOGY

COURSE CODE: BCT 205

MERCY VICENTIA ADU GYAMFI (MRS)

Comparisons, Conditions, and Compound Conditions

Comparison operators

- Comparison operators are generally used inside a construct such as an if statement in which you need to compare two items. For example, you may wish to know whether a variable you have been incrementing has reached a specific value, or whether another variable is less than a set. value, and so on.
- **Note** the difference between **= and ==.** The first is an assignment operator, and the second is a comparison operator.

Comparison operators

Operator	Description	Example
==	Is **equal** to	$j **==** 4
!=	Is **not equal** to	$j **!=** 21
>	Is **greater than**	$j **>** 3
<	Is **less than**	$j **<** 100
>=	Is **greater than or equal** to	$j **>=** 15
<=	Is **less than or equal** to	$j **<=** 8

Verifying Variables

- PHP is most often used for developing web applications. As such, you need to get data from your form entries. For security reasons and standard practice, you want to verify the content of this input. Maybe you want to check if a user entered anything at all into a field.
- You could use **==** to check for empty strings, however it is better to use the built-in PHP functions: isset() and empty().
- **isset( ) checks that a variable exists and is set. empty() checks that a variable has any contents**.

[1] isset(): https://php.net/isset [2] empty(): https://php.net/empty

Verifying and Checking Variable Types

- Since PHP is a dynamically typed language, you might also want to use some of the built-in functions to verify the types of PHP variables:

Function	Type Checked	Explanation
is_bool()	boolean	returns true if parameter is a boolean
is_int()	integer	returns true if parameter is an integer
is_float()	float	returns true if parameter is a float
is_string()	string	returns true if parameter is a string
is_scalar()	scalar data types	returns true if parameter is a scalar type, any of the above

Looping

- Looping in PHP is used to run a block of code either a predetermined number of times or until a specific condition is reached. When looping a predetermined number of times, this is known as a counting loop.
- We don’t know how many times a code block will repeat, but we know the condition for ending the repetition. This state is known as a sentinel loop.
- There are four loop structures in PHP.
  - **1. for**
  - **2. while**
  - **3. do/while**
  - **4. foreach**
- for and while loops are the most common looping structures in all programming languages.

Counting Loops

- for and foreach are considered counting loops. They repeat a block of code based on a counter variable or the number of items in a collection.

for Loops

- A for loop has three clauses separated by two semi-colons (;). We call the first clause the **“initializer.”** It is executed only the first time upon encountering the loop. The second clause is called the **“conditional”** and is executed every time the compiler encounters the loop. The last clause is called the “**incrementer**” and is used to either increment or decrement a variable upon completing a loop iteration.

Counting Loops

- It is important to note when incrementing a variable in a loop, the variable to increment (in this case, $looping_value) must be initialized. Otherwise, you end up with a NULL value as your final result. This bug is common in looping. Also, note the incrementer (in this case, $counter) will be set to 10 because the last successful iteration of the loop increments the variable to the next value. This final value causes the loop to exit—the incrementer at this point no longer passes the conditional expression with a value of true.

Sentinel Loops

- while and do/while are sentinel loops. They execute until some condition is false.

while Loops

- When new programmers learn about looping, they are first taught how to use the while loop because it only has one clause and is the simplest loop to create.
- *It's essential to ensure your loops terminate. If not, you create an infinite loop that only ends if PHP's max execution time is reached.*

- The proper use of the while loop is to continue executing a block of code until a particular event occurs.

Sentinel Loops

- Let us assume we have a MySQL database that contains product information stored in a table. Let us say we want to output the product name and unit price for every row in the table, but we don’t know how many rows it contains. We just want to output all of them.
- PHP offers several functions for storing and retrieving data in MySQL databases. When querying a database, we get an associative array returned that contains the whole result set. This result set needs to be passed into a function **(mysqli_fetch_array())** that returns the next row in the record set as an associative array that uses the field names as element references for the values contained for this row. Each subsequent call to **mysqli_fetch_array()**
- returns the next row until there are no more rows left in the record set. When there are no more rows in the record set, NULL

Functions

- It is very easy to write code and continue to add functionality to it. The problem is that our program code becomes too large and unwieldy.
- It is common to write code modularized into functions and limit that functionality to a specific task.
- Doing so allows us to reuse these modules throughout our program. Functions also give us the ability to pass in arguments and return results.

Simple Functions

- The simplest function takes no arguments and does not return anything; it just performs some functionality. Let’s say you want a function that will output the greeting, “Welcome tomy Website!” you display for every page of your website. Here is what it might look like:

```
function outputGreetingToWebsite()
{
echo "<h1>Welcome to my Website!</h1>";
}
```

Simple Functions

- When creating a function in PHP, you always start using the function keyword.
- Next, you want to give your function a good name. Since your function is invoked to do something, it is usually a good idea to start the name of your function with a verb.
- Naming your function as specifically as possible helps with the readability and maintainability of our code. Therefore outputGreetingToWebsite is a good name.

Simple Functions cont.

- As you can see by the above example, we start the first word of the function using a lowercase letter, then with each subsequent word joined together in the function, we capitalize the first character of the joined word.
- Next,the function defines parameters that are passed into the function in between a set of parenthesis. In this case, we do not have any parameters to pass, so we will just use an empty set of parenthesis.
- Next, the code within the function is contained within two curly braces ({}).

Simple Functions cont.

- It is also a good idea to organize our functions. Later, when I talk about object-oriented programming, we will put functions (called methods) into something called a “class.” For now, let’s assume we have several functions used to output information to our websitecollected into a single PHP script called outputToWebsite.php.
- When we want to invoke or call our function somewhere else in our code (like in another PHP script), we need to do a couple of things. First, we need to make sure we include the code where our function is defined. We do that using the require_once() statement and include the name of the script containing the function we want to use inside a set of quotes:

```
<?php

require_once('outputToWebsite.php')
```

Function Parameters/Arguments

- Functions can also take parameters. Let's say we want to create a function that adds two numbers as arguments to the function. Let's call this function addTwoNumbers.
- We need two parameters in between the parenthesis separated by a comma. These variables are parameters to the function and are considered local variables scoped to the function.
- It is a good idea to be descriptive if you can with the name of these parameters as well:

```
function addTwoNumbers($num1, $num2)
{
$sum = $num1 + $num2;
echo "$num1 + $num2 = $sum<br/>";
}
```

Returning Values from a Function

- Finally, we can write functions so that they return results to us.

```
function sum($num1, $num2)
{
$sum = $num1 + $num2;
return $sum;
 }
```

- Now when we call this function we can create a variable that will hold the result of the call and set it equal to the call of the function:

```
$first_num = 5;
$second_num = 7;
$sum_of_two_numbers = sum($first_num, $second_num);
echo "The result of adding $first_num and $second_num is $sum_of_two_numbers<br/>";
```

Further Advice On Writing Good Functions

- Writing functions is tricky, and it's more of an art than science, but it's also 90% common sense. Let's look at some advice for writing functions with high readability and maintainability.

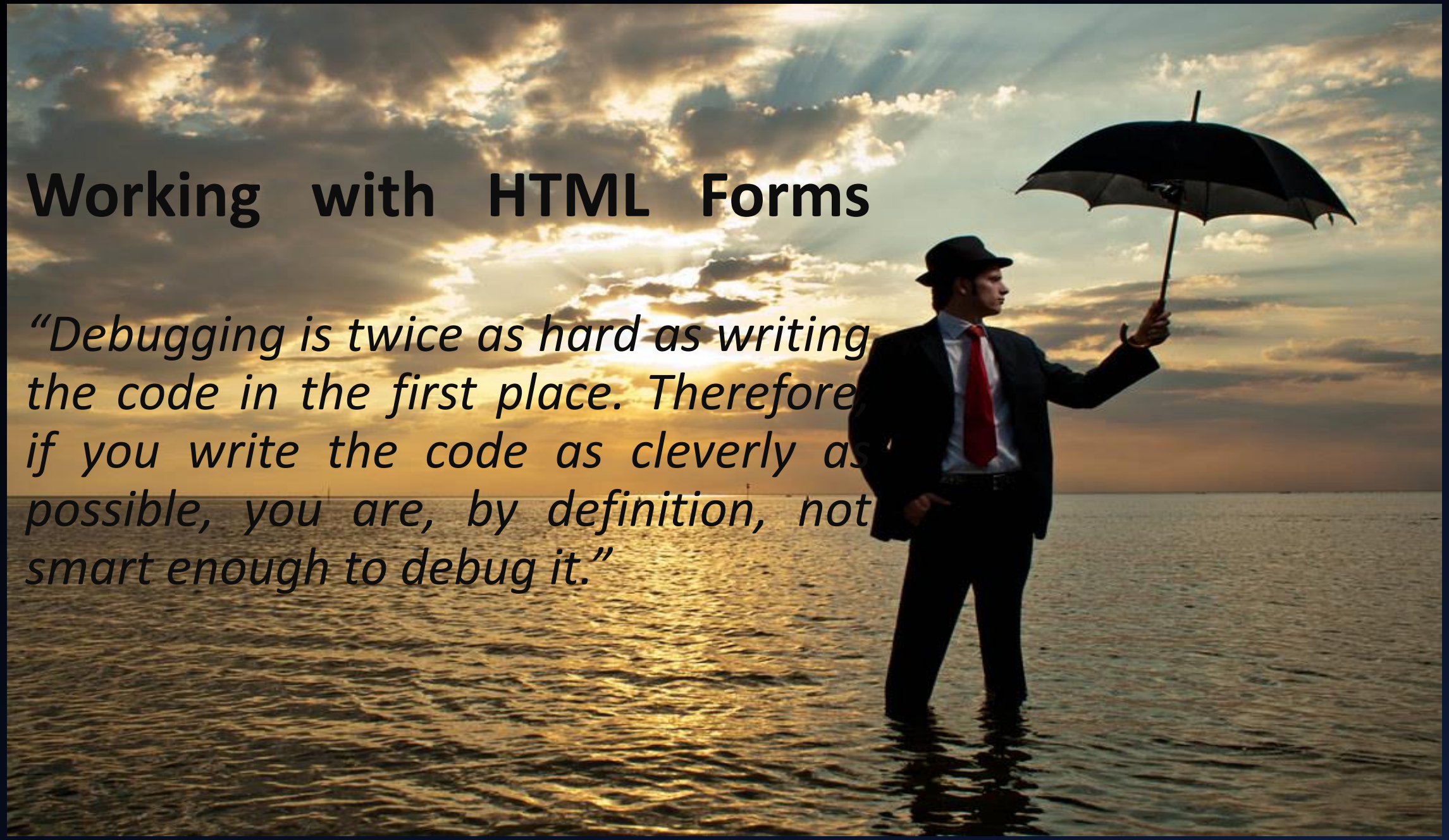
Global Variables are Evil!

- I cannot overemphasize this point that global variables are evil. The problem with global variables is all of your code (at a minimum, the code within a single script) has access to it and could change its value.
- This practice makes debugging and maintenance difficult. Unfortunately, it is a lazy substitute for proper design.

Further Advice On Writing Good Functions

Further Advice On Writing Good Functions

Working with HTML Forms



THANK YOU !

NO MATTER WHICH FIELD OF WORK YOU WANT TO GO IN, IT IS OF GREAT IMPORTANCE TO LEARN AT LEAST ONE PROGRAMMING LANGUAGE.