CSIS 3280: Lecture 1

Course Overview

- Lessons, office hour, tests will be online
  - Have access to a decent computer with good speaker and internet connection
- Online class
  - Ask question in the chat or raise your hand
  - It is ‘normal’ class but online. I appreciate it if you show your video most of the time during the class
  - Attendance will be called at random times.
    - You will lose your attendance if you don’t reply with your MIC or turn off your video during the roll call
- Exam:
  - Tests will be online → time will be limited and practical
  - It is the students’ responsibility to be available online with the required computing (software and hardware) requirement for the test
  - If you are not confident with your computing and networking resources, you should consider taking the exam in the College

Course Overview

- Office hour
  - Send an email to arrange an office hour time with me.
  - If you want to ask questions regarding your work, please send your files to me beforehand (preferably 1 day before)
- Communication: **use your Douglas student account** for inquiry to sarifb@douglascollege.ca
  - **Email sent using non DC account will be ignored**
- Environment setting
  - Please read the environment settings document
  - If you can, you may want to invest in a second monitor so that you can listen/interact with the online class in one screen and practice/code in another
- Course structure:
  - Lab submission is due before the next class→ basically your assignments, each 2% mark (see submission guideline)
  - Please see the course outline

Copying other students’ work

- Douglas college policy, http://www.douglascollege.ca/about-douglas/governance/policies
- Sharing the class’s exercises, assignments and tests is considered as cheating
- Submitted work (exam or assignment) that is **similar in style** as in other work(s) submitted by another student(s) in the same or previous semesters will be **considered as cheating**
  - College policy on academic dishonesty will be applied
- Person’s **digital signature** is searchable by employers (and the Federal Government of Canada).

Think about it.

#It’sNotWorthIt

Course Overview

- Final Exam period Aug 11 – Aug 19.
- Final exam:
  - All material
  - practical

Week	Date	Activity	Details and Deadlines
1	13-May-21	Course overview and logistics Code and environment setup (CH 2) Introduction to PHP (CH 3)	
2	20-May-21	Functions (CH 4) Arrays (CH 5)	**Lab 1**
3	27-May-21	**QUIZ 1 (Lecture 1 ~ 2)** Date and Time (CH 12) Forms (CH 13)	**Lab 2** **May 25: Last day to withdraw without W record. Last day to add a course**
4	03-Jun-21	Exeption Handling (CH 8) File handling and Uploads (CH 10, 15)	**Lab 3**
5	10-Jun-21	Object Oriented PHP (CH 6)	**Lab 4**
6	17-Jun-21	Strings and RegEx (CH 9) **M I D T E R M (Lecture 1 ~ 6)**	**Lab 5**
7	24-Jun-21	Advance OOP (CH 7)	**Midterm (Lecture 1 ~ 6)**
8	01-Jul-21	College Closed (Canada Day)	
9	08-Jul-21	Intro. to SQL and Web Database Design (CH 22, 25) Using PHP with SQL (CH 27)	**Lab 6**
10	15-Jul-21	PDO and CRUD Operations with PDO (CH 28)	**Lab 7** **Jul 12: Last day to withdraw with W record**
11	22-Jul-21	**QUIZ 2 (Lecture 7 ~ 10)** Authentication (CH 14), Session (CH 17)	**Lab 8**
12	29-Jul-21	JSON and Web Services (CH 18)	**Lab 9**
13	05-Aug-21	MVC and Introduction to Framework (CH 21)	**Lab 10**

PHP Intro

- Created in 1990. The latest version is PHP 8!
- The most popular web scripting language → 82% of the sites
- Practical, can be embedded to web pages easily → messy code!
- No need to include libraries, but supported by thousands of functions
- Powerful text/string manipulation, form processing, date formatting, etc
    - Used in system administration, web development, prototyping, general scripting, and many more
- Highly extensible: support tons of databases and file formats
- Low (hardware) operational cost
- Fast learning curve and used to be a procedural script
    - We'll **learn** the object oriented PHP!

PHP environment

- You need the following
  - **PHP**
  - Database: **MySQL**, MariaDB, PostgreSQL, etc
  - Web server: **Apache**, NGINX
  - Text editor, IDE: Notepad++, Sublime Text, **Visual Studio Code**
- Please look at the PHP environment setting document
- In the lab and windows computer:
  - WAMP
  - Visual Studio Code
  - Xdebug for Visual Studio Code
- For MAC machine, please use XAMPP (https://www.apachefriends.org/index.html)
- Linux
- Cloud: AWS, GCP?

STOP!!

- We will learn how to write how to code PHP following the good programming practice
  - Separation of concern
  - Files naming
  - Directory structure
  - Code structure and cleanliness
  - Comments

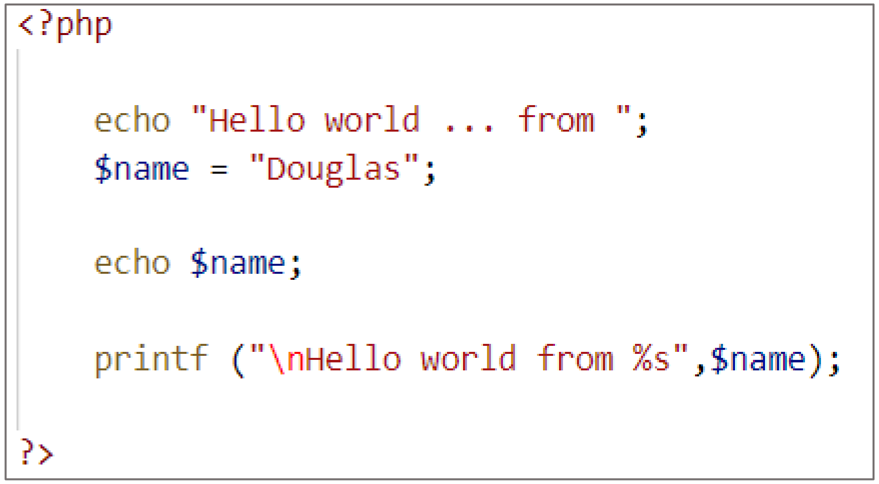
- **No messy code**
  - Submitted work that is not following the coding and lab/assignment guideline will be given up to 100% mark discount

PHP Tags

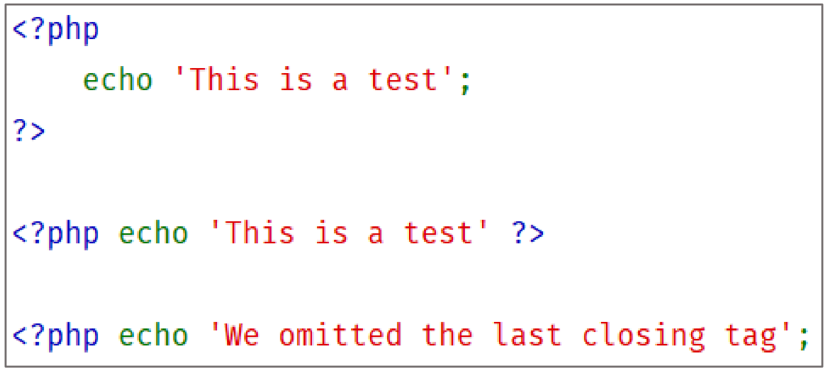
- PHP files are text files that are interpreted, PHP code can be declared inside any text file by using the following tags:
- **<?php … ?>** -- Cannonical PHP Open and Close tags
- <?= … ?> -- shortcircuit syntax
- <? … ?>
  - Short tags (SGML Style), only work if short_open_tag is enabled in php.ini or php is installed with –enable-short-tags
- <% … %>
  - ASP Style tags, removed in PHP 7
- <script language = “PHP”> … </script>
  - HTML Style tags, removed in PHP 7

Syntax

- Expressions are terminated by semicolons**;**
- Expressions are made up of tokens such as numbers (3.194), strings (.text.), variables (**$**yntax) and constants (true).
- Some special words if, then, else, case, switch, foreach etc… are words for controlling the code flow, i.e., loops, conditional statements and Boolean logic.
- The php closing tags implies a semicolon → please avoid doing this for consistency



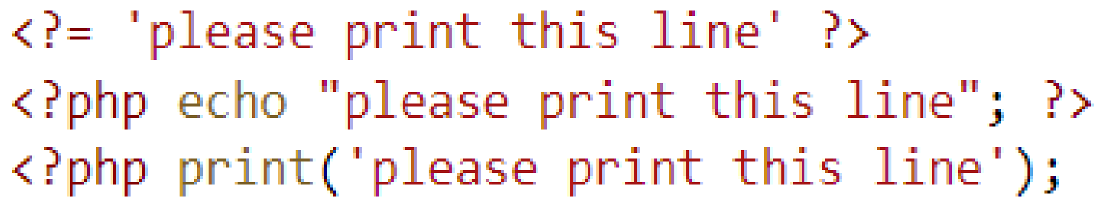
```
<?php

    echo "Hello world ... from ";
    $name = "Douglas";

    echo $name;

    printf ("\nHello world from %s",$name);

?>
```



```
<?php
    echo 'This is a test';
?>


<?php echo 'This is a test' ?>


<?php echo 'We omitted the last closing tag';
```

Short circuit syntax

- Short echo tag is used to quickly escape from and to php
  - <?= 'please print this line' ?>
- The following syntax produce the same output



```
<?= 'please print this line' ?>
<?php echo "please print this line"; ?>
<?php print('please print this line');
```

- Note on printing statements (more later)
  - The difference between echo and print: echo has no return value and is a bit faster.

Comments

Make good comments, not bad?

Good code needs minimal number of comments!

Bad comments make me sad.

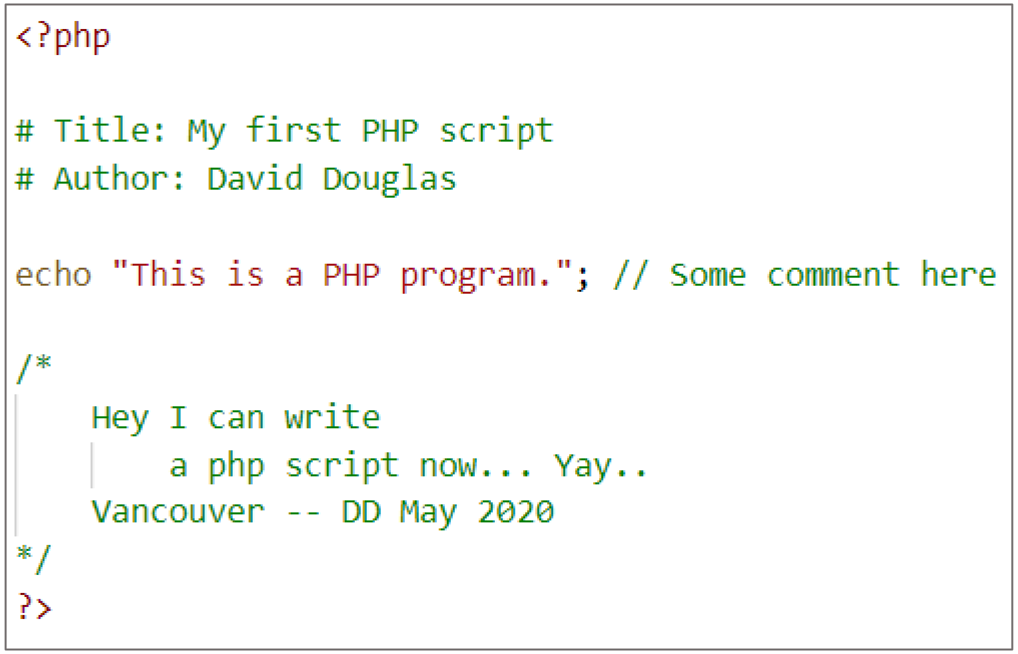
Please do not write your letter in the comment ☺

Your comments should explain what you are trying to do, if you are beginning its ok to use a lot of comments, typically you will not be faulted for putting too many comments. It is a good practice to add your initials to the end of your comment:

```
//This is my comment –DD May 7, 2020
```

Comments

```
//This Comment
#That Comment



/*
    Here is a
    multi-line
    comment. Comment.
    */
```



```
<?php

# Title: My first PHP script
# Author: David Douglas

echo "This is a PHP program."; // Some comment here

/*
    Hey I can write
        a php script now... Yay..
    Vancouver -- DD May 2020
*/
?>
```

Demo code:
syntax.php,
html_php.php

Tips

Instead of working in C:/wamp64/www folder:

- Create Apache's virtual host
- Create symbolic links in windows

Creating symbolic links in Windows

see https://www.educative.io/edpresso/mklink-windows-10

Below is the command to create a symbolic link C:\Documents\CSIS3280 → C:\wamp64\www\CSIS3280-002

```
mklink /d C:\wamp64\www\CSIS3280-002 C:\Documents\CSIS3280
```

Any files/folders in the Document folder can be seen at http://localhost/CSIS3280-002

Note: You need to run the cmd with the admin priviledge

Printing Data: Echo

- Echo accepts a list of arguments, does not require the parentheses, and does not return any value

```php
<?php
$a = "The value is: ";
$b = 5;
echo $a, $b;
?>
```

What's the difference between these two codes?

```php
<?php
$a = "The value is: ";
$b = 5;
echo "$a $b";
?>
```

- When working with double quoted strings, we can embed the variables directly into strings
- Use curly bracket to provide visual cue separating between variables and static strings

```php
<?php
$b = 5;
echo "The value is {$b}";
?>
```

✓

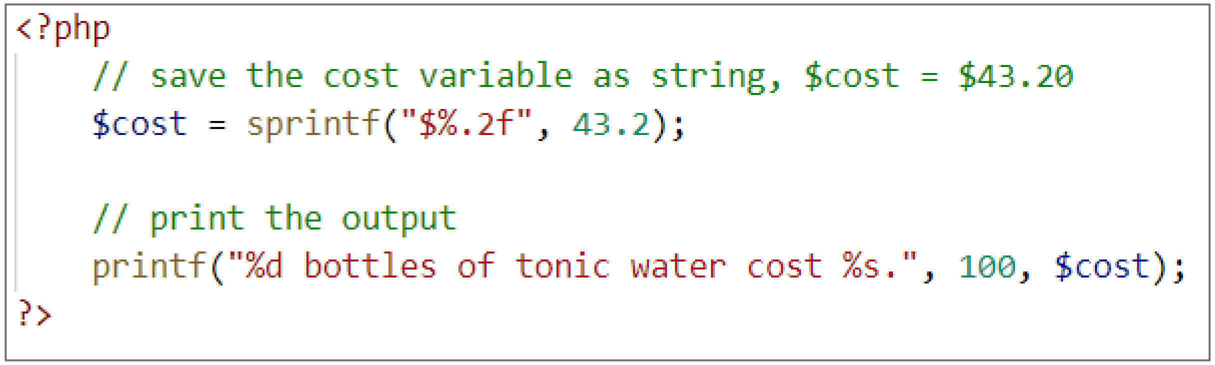
Printing Data: print and printf

- print is similar with echo in terms of speed and functionality
- printf ≈ C’s printf
  - Format the output with some specifier
  - A slightly slower than echo
  - Good if you want to combine several variables

Type	Description
%b	Argument considered an integer; presented as a binary number
%c	Argument considered an integer; presented as a character corresponding to that ASCII value
%d	Argument considered an integer; presented as a signed decimal number
%f	Argument considered a floating-point number; presented as a floating-point number
%o	Argument considered an integer; presented as an octal number
%s	Argument considered a string; presented as a string
%u	Argument considered an integer; presented as an unsigned decimal number

Printing Data: sprintf

- sprintf is similar to printf but the output is assigned to a string



```php
<?php
    // save the cost variable as string, $cost = $43.20
    $cost = sprintf("$%.2f", 43.2);

    // print the output
    printf("%d bottles of tonic water cost %s.", 100, $cost);
?>
```

Data Types (textbook pp. 56 – 63)

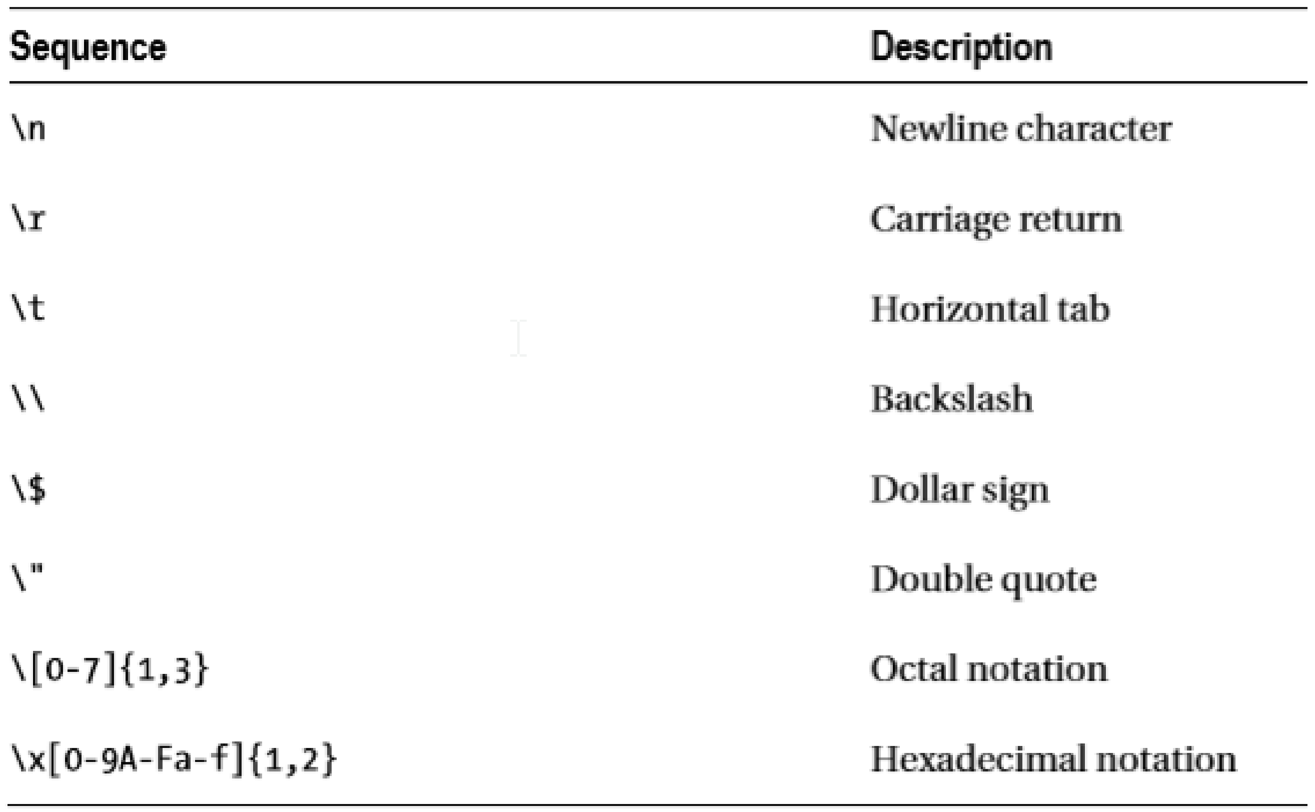
- Integers – whole numbers: 1,5, -6, 45565
- Float – floating-point numbers: 4.5676, 4.0, 8.7e4
- Booleans – true or false, nothing else.
  - Be careful when assigning number to a Boolean variable. Assigning 0 or '0' means false. Others are true
- NULL – NULL – NOTHING – NADA – NEHI
- Strings – sequences of characters called ‘strings’. Double quoted “strings” are more flexible
- Arrays – named indexes of collections- store data using numeric indexes or associative indexes (more on this next class).
- Objects – Hold instantiated classes

String Interpolation

- Double Quotes – denote a string
  - Very cumbersome when dealing with HTML because HTML has a lot of quotes in it regardless
  - It is very flexible in the php code since you can put the variables within the quotes
- Single Quotes – denote a string
  - A lot easier to deal with as they occur less frequently in HTML and therefore do not have to be **escaped**.
  - Php variables that are put within single quote will not be treated as variables
- **Choose one style and be consistent!**

Escape Sequences

- Escape sequences are special characters that can be used to either interpret text literally or for special characters



Sequence	Description
`\n`	Newline character
`\r`	Carriage return
`\t`	Horizontal tab
`\\`	Backslash
`\$`	Dollar sign
`\"`	Double quote
`\[0-7]{1,3}`	Octal notation
`\x[0-9A-Fa-f]{1,2}`	Hexadecimal notation

Identifier

- Always pick good names for user defined variables, functions and objects.
  - Must **begin with a letter or underscore**.
  - Can only consist of letters, numbers and underscores
  - Can be any length
  - Are **case sensitive**
  - Can’t be identical to any of PHP’s pre-defined keywords.

Variables

- Identifiers where data is stored!
- Value is always its most recent assignment
- Always start with a “$”
- Assigned with the assignment operator “=“
- Weak typed (variables do not need to be instantiated)
  - **But you should** as a matter of good practice
- If you use a variable before assigning it, it will have a default value
- PHP does casting for you! (most of the time)…
  - Please read type casting in the textbook pp. 60 - 62

Variables

- Value Assignment:
  - Most common
  - Copies the variable contents on assignment

- Reference Assignment
  - Less common
  - “Alias” or “Pointer” to a variable
  - Equals operator with an ampersand sign appended to it:

```
$value1 = "Hello";
$value2 = &$value1;
```

Demo code:
echo.php, printf.php

Variable scope

- **Local Variable** - Variables defined inside of functions, only visible inside the function.
- **Function Parameters** – arguments accepted in the function declaration. These arguments accept values that come from outside the function, once passed in they are no longer valid after the function exits.
- **Static Variables** – These variables out-live a functions execution and are available the next time a function is run.
- ~~**Global Variables** – accessed by any part of the program, you must explicitly declare the global variable inside the function which it is modified.~~
- **Super Global Variables** – these are made available as part of the environment; these are pre-defined and usually environment specific. For example: $_SERVER, $_GET and $_POST

Constant

- A constant is a value that cannot be modified
- Practically these are used to configure applications and often placed into a configuration file (for example: config.inc.php)
- Constants are defined using the **define** or **CONST** keyword.

```
        define("PI", 3.141592);
```

Or

```
        const PI = 3.141592;
```

Expression

- Phrases representing a particular action in a program.
- have at least operand and one or more operators

```
$a = 5;                        // assigns integer value 5 to the variable $a
$a = "5";                      // assigns string value "5" to the variable $a
$sum = 50 + $a;                // assigns sum of 50 + $a to $sum. "5" is converted to number
$a += 10;                      // adds variable $a by 10
$first = "Douglas";            // assigns "Douglas" to the variable $first
$name = $first . " James";     // assigns concatenated $first with " James" to variable $name
$inventory++;                  // increments the variable $inventory by 1
// open a file OR DIE??
$file = fopen("filename.txt", 'r') OR die("File does not exist!");
```

Operands

- The input of the expression
- The values or variables you are starting with

```
$a++; // $a is the operand
$sum = $val1 + val2; // $sum, $val1 and $val2 are operands
```

Operators

- The symbol that specifies the action in the expression such as adding, or decrementing.
  - Some operators will convert the type of the operand.

- Type
  - Assignment – these are for modifying variables
  - Arithmetic – these do math.
  - Comparison – these compare.
  - Logical or Relational – these do logic.
  - Conditional – these decide.

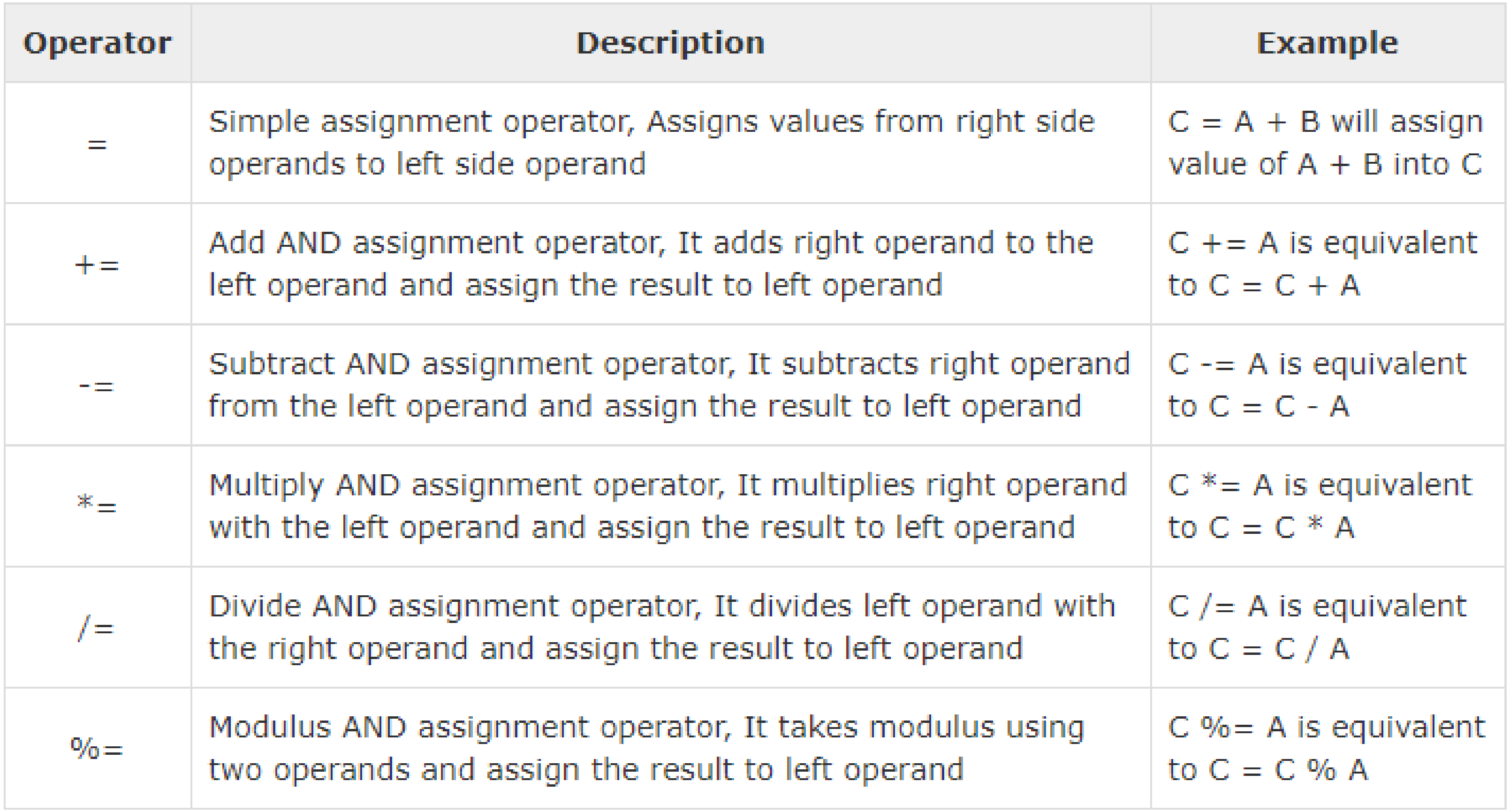
Operator Precedence and Associativity

- Operator Precedence
  - The order in which operators evaluate the operands around them.
  - Think BEDMAS (Brackets, Exponents, Division, Multiplication, Addition, Subtraction)
  - Also generally arithmetic >>> logical >>> assignment

- Operator Associativity
  - Associativity is how operations of the same precedence are evaluated.
    - Left to Right (most operator)
    - Right to Left (negation, increment, assignment)

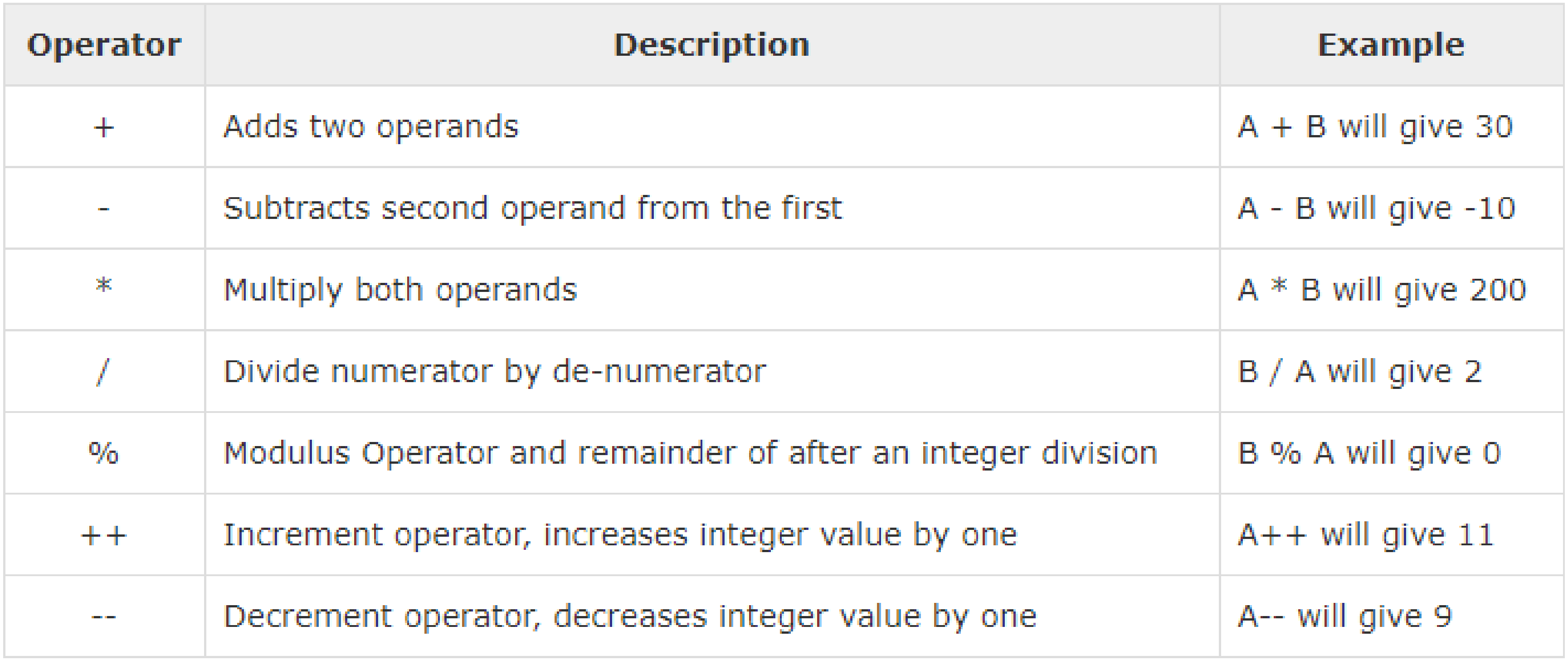
- Please view https://www.php.net/manual/en/language.operators.precedence.php

Assignment Operators



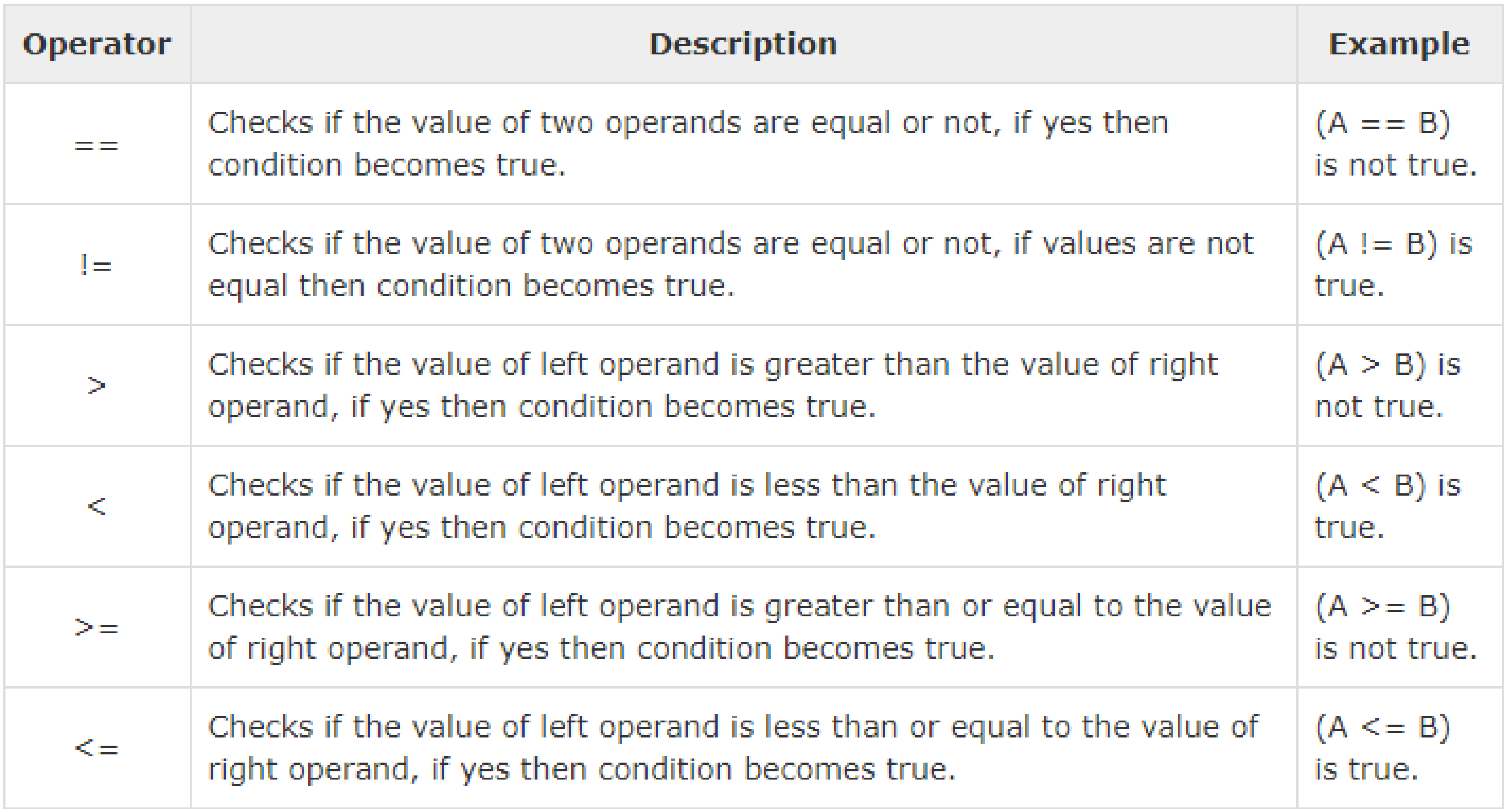
Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assign value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A

Arithmetic – these do math.



Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiply both operands	A * B will give 200
/	Divide numerator by de-numerator	B / A will give 2
%	Modulus Operator and remainder of after an integer division	B % A will give 0
++	Increment operator, increases integer value by one	A++ will give 11
--	Decrement operator, decreases integer value by one	A-- will give 9

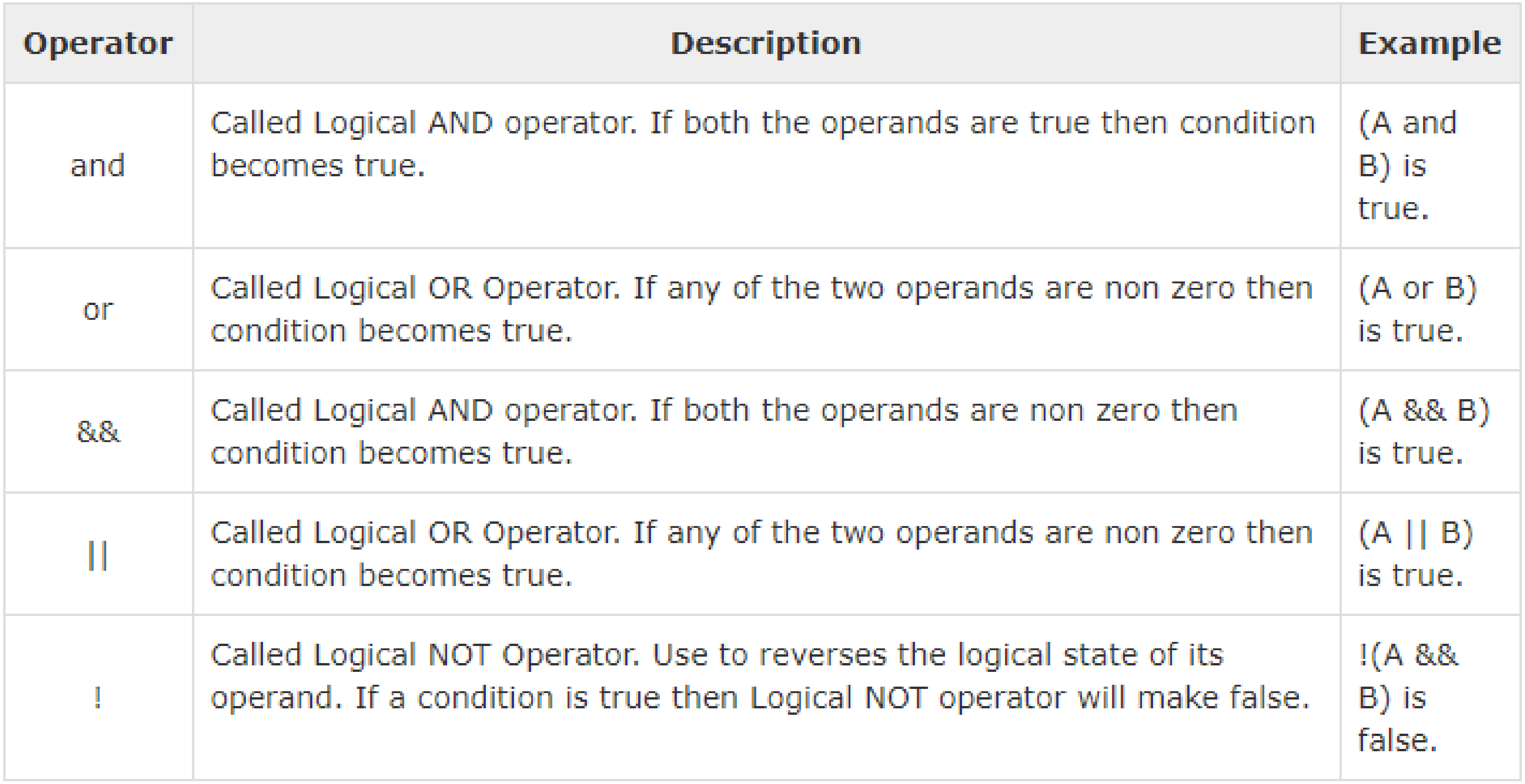
Comparison – these compare.



Operator	Description	Example
==	Checks if the value of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

- **New!** Php spaceship operator <==>
  - https://www.tutorialspoint.com/php7/php7_spaceship_operator.htm

Logical Operators



Operator	Description	Example
and	Called Logical AND operator. If both the operands are true then condition becomes true.	(A and B) is true.
or	Called Logical OR Operator. If any of the two operands are non zero then condition becomes true.	(A or B) is true.
&&	Called Logical AND operator. If both the operands are non zero then condition becomes true.	(A && B) is true.
||	Called Logical OR Operator. If any of the two operands are non zero then condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is false.

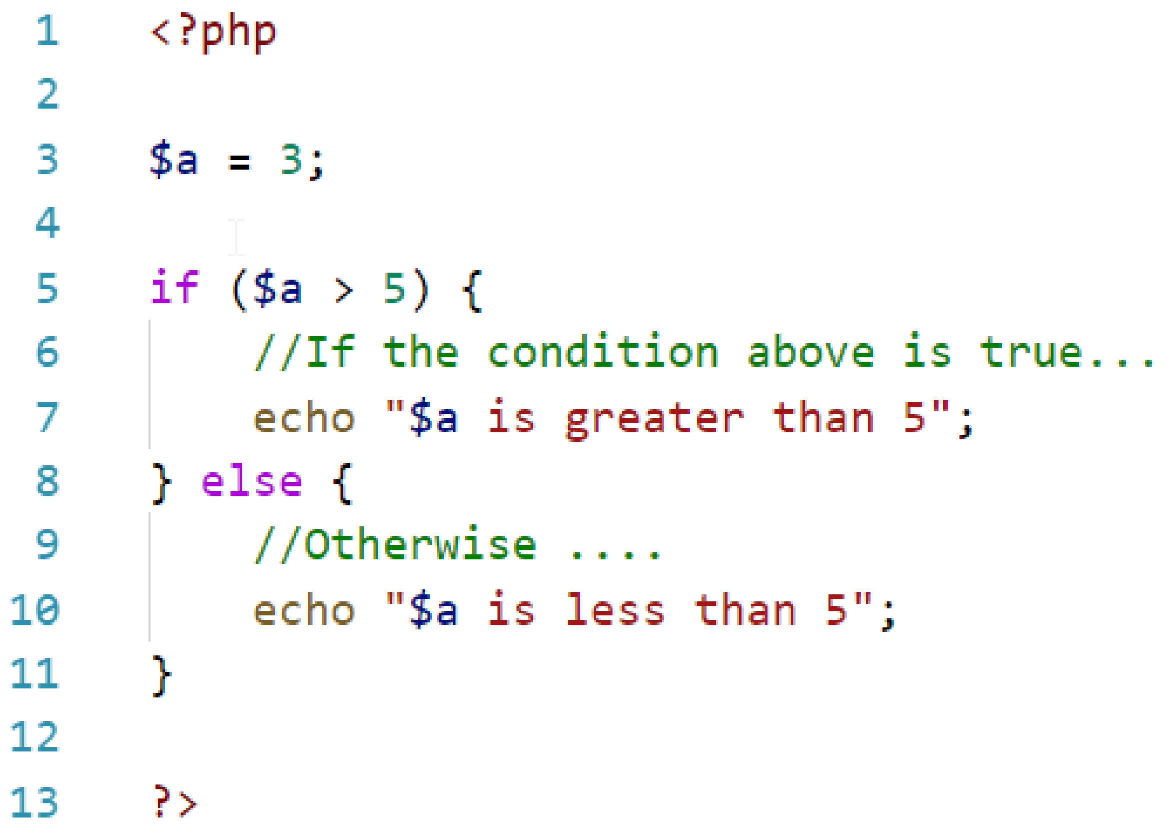
Conditional

- Evaluate two Boolean expressions and return the result.

```php
<?php

$a = 1;
$b = 2;

//If 1 is less than two, assign the result variable to 1 or 2.
$result = ($a < $b ) ? "one is less than two" : "two is less than one?";
echo $result;

?>
```

If/else statements

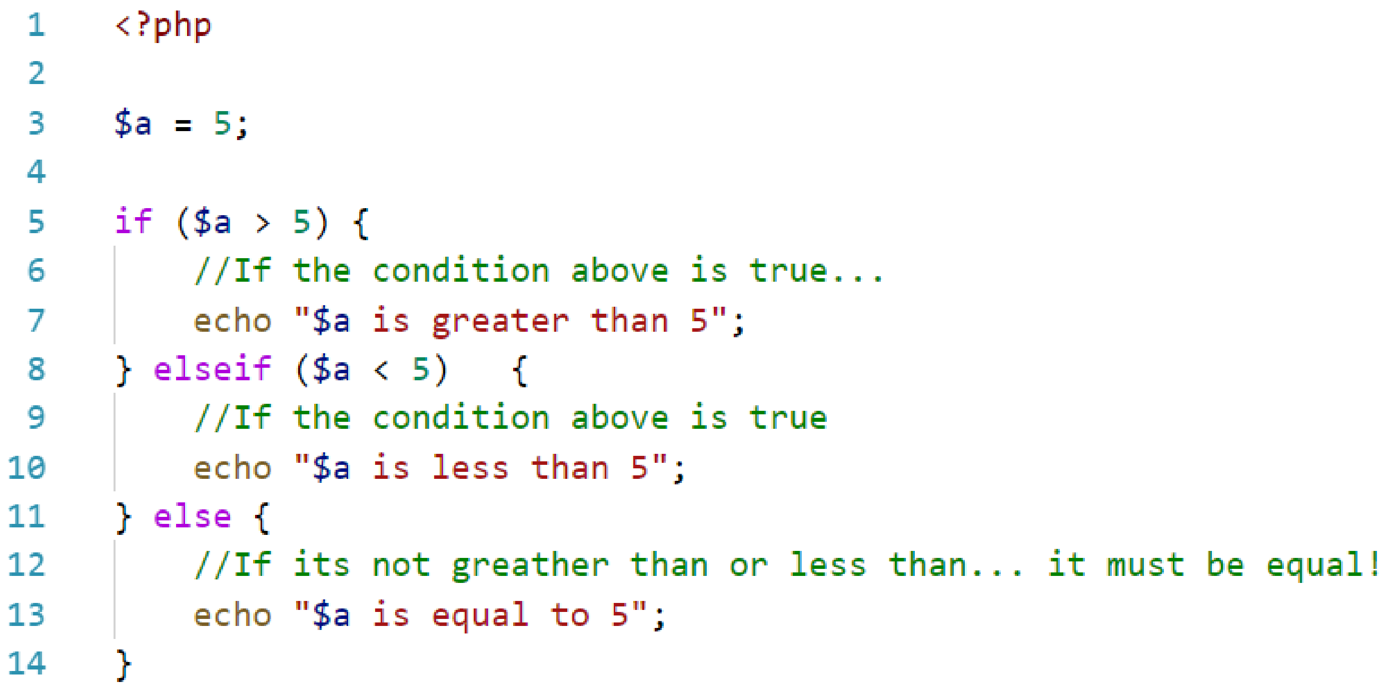
- It is not fun if there’s no choice, rite?



```php
<?php

$a = 3;

if ($a > 5) {
    //If the condition above is true...
    echo "$a is greater than 5";
} else {
    //Otherwise ....
    echo "$a is less than 5";
}

?>
```

Else-if statements

- Else-if can be used to chain If statements – don't overuse this.



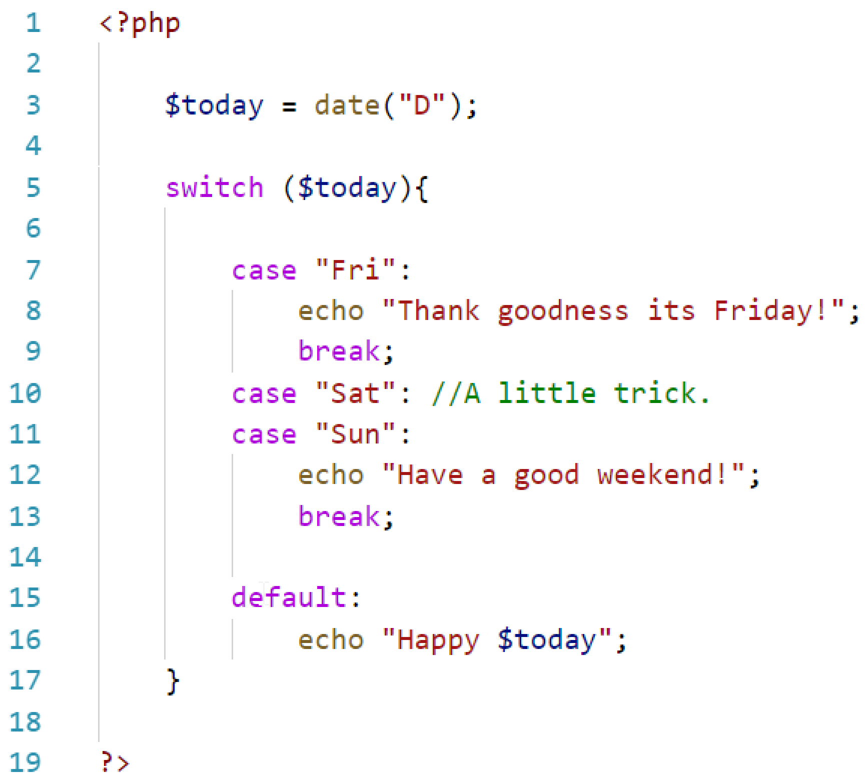
```php
<?php

$a = 5;

if ($a > 5) {
    //If the condition above is true...
    echo "$a is greater than 5";
} elseif ($a < 5)   {
    //If the condition above is true
    echo "$a is less than 5";
} else {
    //If its not greather than or less than... it must be equal!
    echo "$a is equal to 5";
}
```

Demo code:
ifelse.php, ifelseif.php

Case statement

- Switch is like choosing a path to take from multiple ways
  - Think of a road intersection!

- Switch structure needs break statements
  - Why??



```php
<?php

    $today = date("D");

    switch ($today){

        case "Fri":
            echo "Thank goodness its Friday!";
            break;
        case "Sat": //A little trick.
        case "Sun":
            echo "Have a good weekend!";
            break;

        default:
            echo "Happy $today";
    }

?>
```

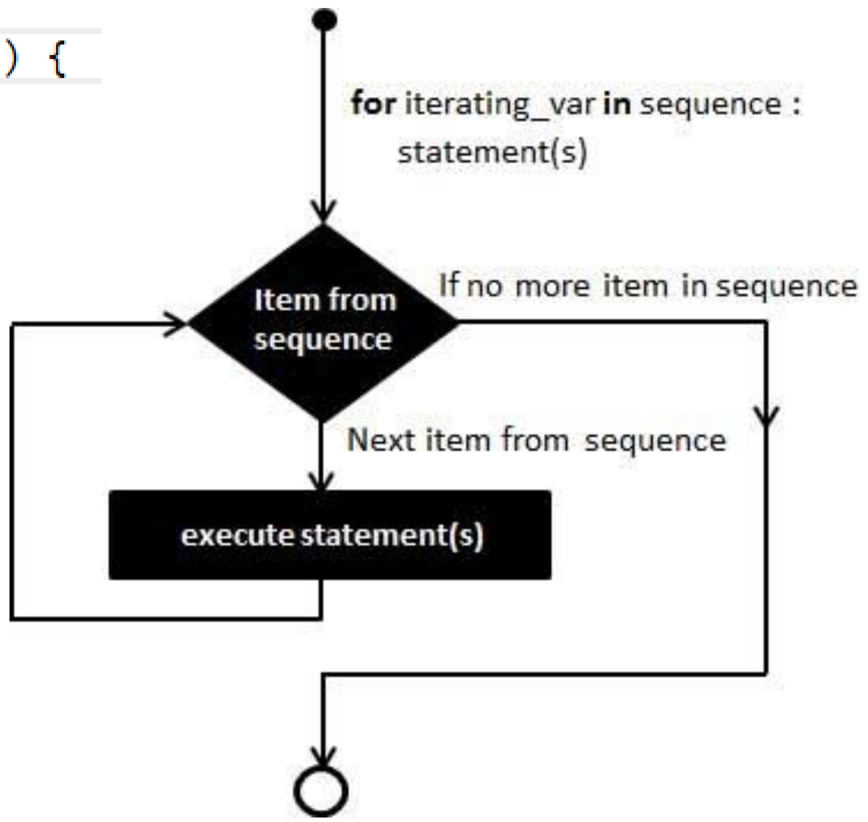
Loops

- Sometimes you need to repeat several tasks using a structure called loops:
  - for – pre-test loop
  - while – pre-test loop
  - do while – post-test loop

for loop

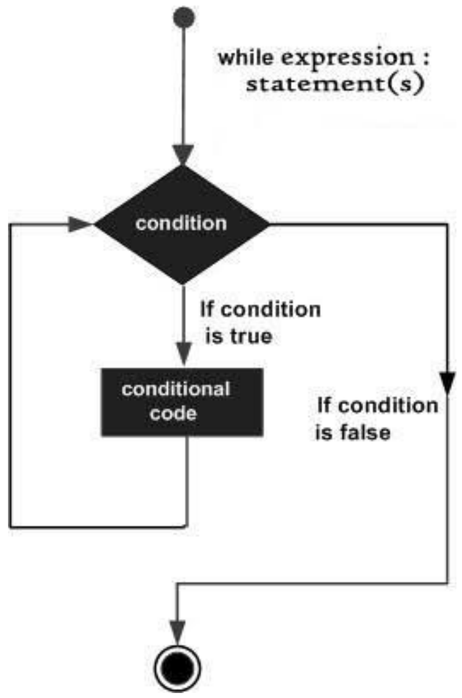
- The initialization, comparison and update in a single line of code

```php
<?php

    for ($v = 10; $v > 0; $v--) {
        echo $v  . "\n";
    }

?>
```



while

```php
<?php

$v = 0;

while ($v <= 10) {

    echo $v . "\n";
    $v++;
}

?>
```



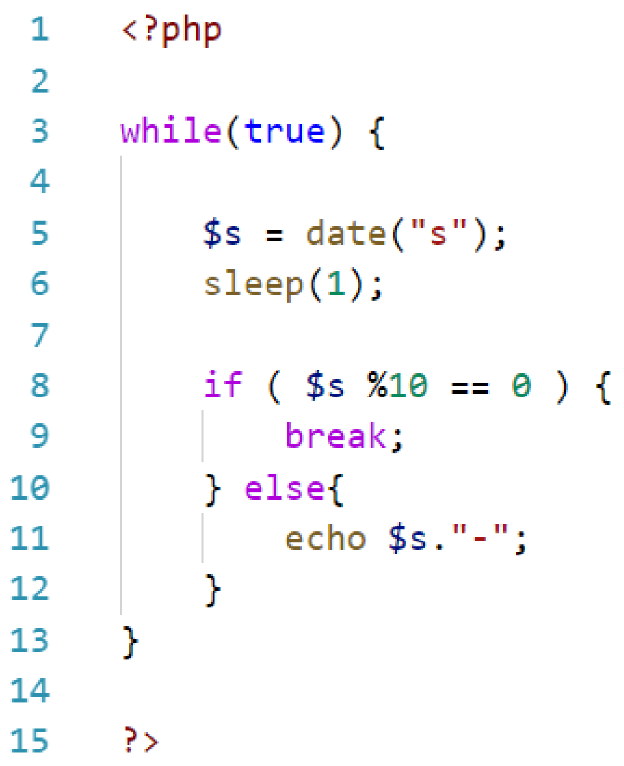
do…while

```php
<?php
$start = date("s");
do {

    //do stuff
    echo ".";
    sleep(1);

} while (date("s") % 10 != 0)

?>
```

break; and continue;

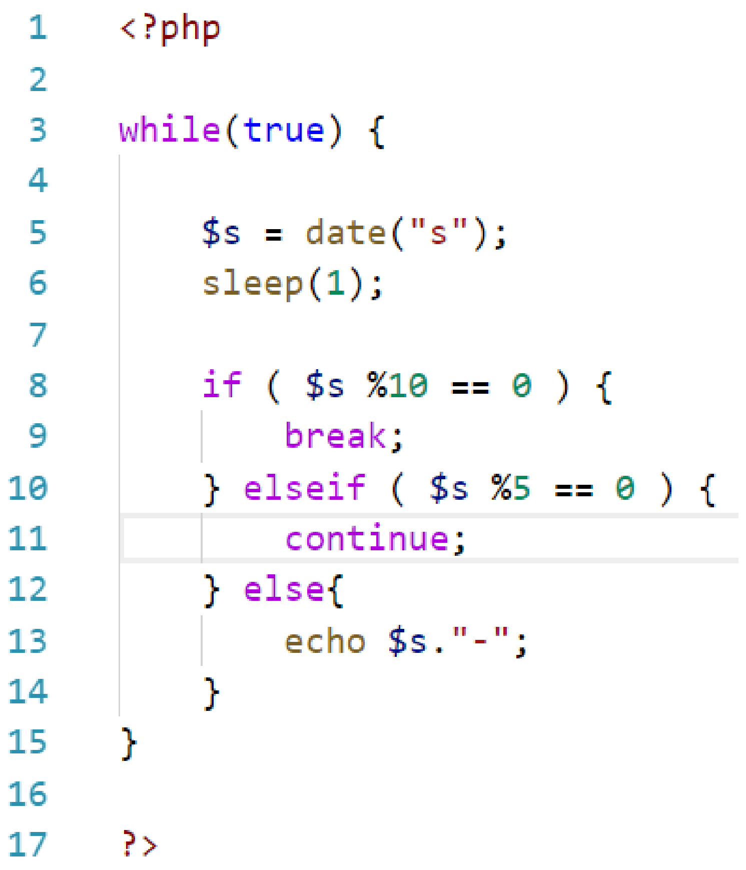
- Use break; to ‘**break out**’ of a block statement, e.g. loop
- Use continue; to **skip** the current iteration (cycle) of the loop **and continue the next iteration**.

break;



```php
<?php

while(true) {

    $s = date("s");
    sleep(1);

    if ( $s %10 == 0 ) {
        break;
    } else{
        echo $s."-";
    }
}

?>
```

continue;

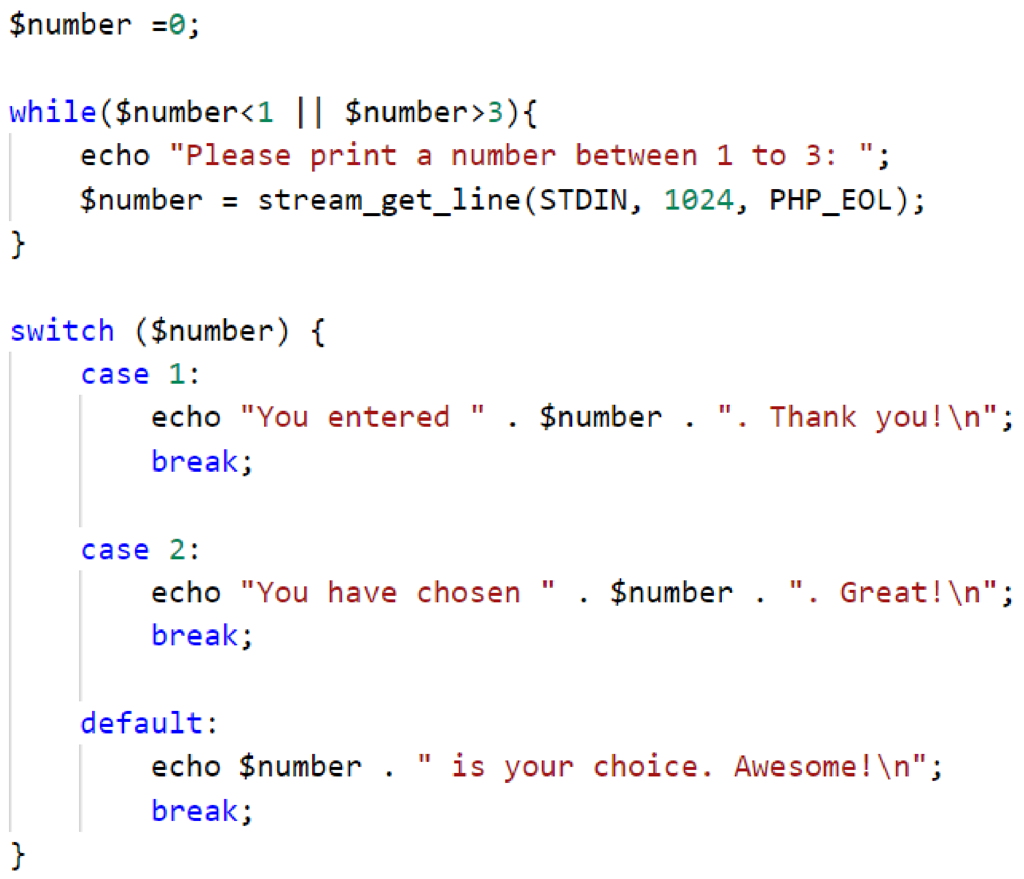


```php
<?php

while(true) {

    $s = date("s");
    sleep(1);

    if ( $s %10 == 0 ) {
        break;
    } elseif ( $s %5 == 0 ) {
        continue;
    } else{
        echo $s."-";
    }
}

?>
```

Demo code: for_break.php, for_continue.php, while.php, dowhile.php

Prompt input from the user

- You can use fgets() or stream_get_line()
  - stream_get_line() is faster than fgets() and you can specify a different end of line character



```php
$number =0;

while($number<1 || $number>3){
    echo "Please print a number between 1 to 3: ";
    $number = stream_get_line(STDIN, 1024, PHP_EOL);
}

switch ($number) {
    case 1:
        echo "You entered " . $number . ". Thank you!\n";
        break;

    case 2:
        echo "You have chosen " . $number . ". Great!\n";
        break;

    default:
        echo $number . " is your choice. Awesome!\n";
        break;
}
```

Demo code:
while_switch.php

Lab 1

- Download Lab1 file from the Blackboard
- You need to submit it next week in the morning of the day of next class
- **Make sure to follow the submission guideline!**