UNIT – 2
OPERRATORS AND CONTROL STRUCTURE

UNIVERSITY
OF PETROLEUM
& ENERGY STUDIES

Dr. P S V S Sridhar

Assistant Professor (SS)

Centre for Information Technology

PHP - Operators

- Operators are used to manipulate or perform operations on variables and values.
- There are many operators used in PHP
  - Assignment Operators
  - Arithmetic Operators
  - Comparison Operators
  - String Operators
  - Combination Arithmetic & Assignment Operators

Assignment Operators

Assignment operators are used to set a variable equal to a value or set a variable to another variable's value.

Assignment of value is done with the "=", or equal character.

```
$my_var = 4;
$another_var = $my_var
```

Now both $my_var and $another_var contain the value 4.

Operator	Example	Is The Same As
=	x=y	x=y
+=	x+=y	x=x+y
-=	x-=y	x=x-y
*=	x*=y	x=x*y
/=	x/=y	x=x/y
%=	x%=y	x=x%y

Arithmetic Operators

Operator	Description	Example	Result
+	Addition	x=2 x+2	4
-	Subtraction	x=2 5-x	3
*	Multiplication	x=4 x*5	20
/	Division	15/5	3
		5/2	2.5
%	Modulus (division remainder)	5%2 10%8 10%2	1 2 0
++	Increment	x=5 x++	x=6
--	Decrement	x=5 x--	x=4

Bit wise operators

- Bitwise "and" operator **&**
- Bitwise "or" operator **|**
- Bitwise "exclusive or" operator **^**
- Bitwise "ones complement" operator **~**
- Shift left << 0000000000001010 shift left to 3 as 0000000001010**000**
- Shift right >> 0000000000001010 shift right to 3 as **000**0000000000001

	a	0	0	1	1
	b	0	1	0	1
and	a & b	0	0	0	1
or	a | b	0	1	1	1
exclusive or	a ^ b	0	1	1	0
one's complement	~a	1	1	0	0

Comparison Operators

- Comparisons are used to check the relationship between variables and/or values.
- Comparison operators are used inside conditional statements and evaluate to either true or false
- Assume $x = 4 and $y = 5

Operator	English	Example	Result
==	Equal To	$x == $y	false
!=	Not Equal To	$x != $y	true
<	Less Than	$x < $y	true
>	Greater Than	$x > $y	false
<=	Less Than or Equal To	$x <= $y	true
>=	Greater Than or Equal To	$x >= $y	false

- **===** **Identical (If arguments are equal to each other and of the same type but false otherwise) (**1 === "1" results in false. 1 === 1 results in true.)

String Operators

- Period "." is used to add two strings together, or more technically, the period is the concatenation operator for strings.

PHP Code:

```
$a_string = "Hello";
$another_string = " Billy";
$new_string = $a_string . $another_string;
echo $new_string . "!";
```

Display:

```
Hello Billy!
```

Combination arithmetic & Assignment operators

- In programming it is a very common task to have to increment a variable by some fixed amount.
- The most common example of this is a counter. you want to increment a counter by 1
  - $counter = $counter + 1;
  - However, there is a shorthand for doing this.
  - $counter += 1;

Operator	English	Example	Equivalent Operation
+=	Plus Equals	$x += 2;	$x = $x + 2;
-=	Minus Equals	$x -= 4;	$x = $x - 4;
*=	Multiply Equals	$x *= 3;	$x = $x * 3;
/=	Divide Equals	$x /= 2;	$x = $x / 2;
%=	Modulo Equals	$x %= 5;	$x = $x % 5;
.=	Concatenate Equals	$my_str.="hello";	$my_str = $my_str . "hello";

Pre/Post-Increment & Pre/Post-Decrement

- Shorthand for the common task of adding 1or subtracting 1 from a variable.
- To add one to a variable or "increment" use the "++" operator:
- $x++; Which is equivalent to $x += 1; or $x = $x + 1;
- To subtract 1 from a variable, or "decrement" use the "--" operator:
- $x--; Which is equivalent to $x -= 1; or $x = $x - 1;

Logical Operators

Operator	Description	Example
&&	and	x=6 y=3 (x < 10 && y > 1) returns true
||	or	x=6 y=3 (x==5 || y==5) returns false
!	not	x=6 y=3 !(x==y) returns true

The ternary operator

- One of the most elegant operators is the ?: (question mark) operator.
- test_expr ? expr1 : expr2
- The value of this expression is **expr1** if test-expression is true; otherwise, **expr2**
- Eg.

$max_num = $first_num > $second_num ? $first_num : $second_num;

Operator precedence

++ -- (*cast*)
/ * %
+ -
< <= => >
== === !=
&&
||
= += -= /= *= %= .=
and
xor
or

- Comparison operators have higher precedence than Boolean operators.

String Manipulation

The Concatenation Operator

- There is only one string operator in PHP.
- The concatenation operator (.) is used to put two string values together.
- To concatenate two variables together, use the dot (.) operator:

```
<?php
$txt1="Hello World";
$txt2="1234";
echo $txt1 . " " . $txt2;
?>
```

The output of the code above will be:

```
Hello World 1234
```

Using the strlen() function

- The strlen() function is used to find the length of a string.

```
<?php
echo strlen("Hello world!");
?>
```

The output of the code above will be:

```
12
```

$length = strlen(strlen("John")); - First inner strlen is executed . The result is strlen(4), strlen() expects a string therefore converts the integer 4 to the string "4“, the result is 1.

Using the strpos() function

- The strpos() function is used to search for a string or character within a string.
- If a match is found in the string, this function will return the position of the first match. If no match is found, it will return FALSE.

Let's see if we can find the string "world" in our string:

```
<?php
echo strpos("Hello world!","world");
?>
```

The output of the code above will be:

```
6
```

The reason that it is 6, and not 7, is that the first position in the string is 0, and not 1.

Strcmp()

- Strcmp() compares two strings byte by byte until it finds a difference. It returns –ve number if the first string is less than the second and +ve number if second string is less. It returns 0 if they are identical.
- strcasecmp() works the same way excepts that the equality comparison is case insensitive. Strcasecmp(“hey!”, “HEY!”) should returns 0.
- The strstr() function takes a string to search in and a string to look for (in that order). If it succeeds, it returns the portion of the string that starts with (and includes) the first instance of the string it is looking for. If the string is not found, a false value is returned.

```
$string_to_search = “showsuponceshowsuptwice”;
$string_to_find = “up”;
print(“Result of looking for $string_to_find” .
strstr($string_to_search, $string_to_find) . “<br>”);
$string_to_find = “down”;
print(“Result of looking for $string_to_find” . strstr($string_to_search, $string_to_find));
```

which gives us:

Result of looking for up: uponceshowsuptwice

Result of looking for down:

Simple Inspection, Comparison, and Searching Functions

Function	Behavior
`strlen()`	Takes a single string argument and returns its length as an integer.
`strpos()`	Takes two string arguments: a string to search, and the string being searched for. Returns the (0-based) position of the beginning of the first instance of the string if found and a false value otherwise. It also takes a third optional integer argument, specifying the position at which the search should begin.
`strrpos()`	Like `strpos()`, except that it searches backward from the end of the string, rather than forward from the beginning. The search string must only be one character long, and there is no optional position argument.
`strcmp()`	Takes two strings as arguments and returns 0 if the strings are exactly equivalent. If `strcmp()` encounters a difference, it returns a negative number if the first different byte is a smaller ASCII value in the first string, and a positive number if the smaller byte is found in the second string.
`strcasecmp()`	Identical to `strcmp()`, except that lowercase and uppercase versions of the same letter compare as equal.
`strstr()`	Searches its first string argument to see if its second string argument is contained in it. Returns the substring of the first string that starts with the first instance of the second argument, if any is found — otherwise, it returns false.
`strchr()`	Identical to `strstr()`.
`stristr()`	Identical to `strstr()` except that the comparison is case independent.

Substr()

Substr() returns a new string

```
$alphabet_test = “abcdefghijklmnop”;
print(“3: “ . substr($alphabet_test, 3) . “<BR>”);
print(“-3: “ . substr($alphabet_test, -3) . “<BR>”);
print(“3, 5: “ . substr($alphabet_test, 3, 5) . “<BR>”);
print(“3, -5: “ . substr($alphabet_test, 3, -5) . “<BR>”);
print(“-3, -5: “ . substr($alphabet_test, -3, -5) . “<BR>”);
print(“-3, 5: “ . substr($alphabet_test, -3, 5) . “<BR>”);
```

This gives us the output:

3: defghijklmnop
-3: nop
3, 5: defgh
3, -5: defghijk
-3, -5:
-3, 5: nop

Start:
A positive number - Start at a specified position in the string
A negative number - Start at a specified position from the end of the string
0 - Start at the first character in string

Length:
A positive number - The length to be returned from the start parameter
Negative number - The length to be returned from the end of the string

Str_replace(), substr_replace()

Str_replace() replace all instances of a particular substring with an alternate string.

$first_edition = “Burma is similar to Rhodesia in at least one way.”;
$second_edition = str_replace(“Rhodesia”, “Zimbabwe”,$first_edition);
$third_edition = str_replace(“Burma”, “Myanmar”,$second_edition);
print($third_edition);

gives us:
Myanmar is similar to Zimbabwe in at least one way.

Substr_replace() picks out portions to replace by matching to a target string
print(substr_replace(“ABCDEFG”, “-“, 2, 3));
gives us:
AB-FG (CDE portion of the string is replaced with the single -)

Str_repeat() string that is the appropriate number of copies.
print(str_repeat(“cheers “, 3));
gives us:
cheerscheerscheers

Strrev() reverse order

Substring and String Replacement Functions

Function	Behavior
`substr()`	Returns a subsequence of its initial string argument, as specified by the second (position) argument and optional third (length) argument. The substring starts at the indicated position and continues for as many characters as specified by the length argument or until the end of the string, if there is no length argument. A negative position argument means that the start character is located by counting backward from the end, whereas a negative length argument means that the end of the substring is found by counting back from the end, rather than forward from the start position.
chop(), or rtrim()	Returns its string argument with trailing (right-hand side) whitespace removed. Whitespace is a blank space, `\n`, `\r`, `\t`, and `\0`.
`ltrim()`	Returns its string argument with leading (left-hand side) whitespace removed.
`Trim()`	Returns its string argument with both leading and trailing whitespace removed.
`Str_ replace()`	Used to replace target substrings with another string. Takes three string arguments: a substring to search for, a string to replace it with, and the containing string. Returns a copy of the containing string with *all* instances of the first argument replaced by the second argument.
`Substr_ replace()`	Puts a string argument in place of a position-specified substring. Takes up to four arguments: the string to operate on, the string to replace with, the start position of the substring to replace, and the length of the string segment to be replaced. Returns a copy of the first argument with the replacement string put in place of the specified substring. If the length argument is omitted, the entire tail of the first string argument is replaced. Negative position and length arguments are treated as in `substr()`.

Case functions

strlower() returns in all lowercase string

```
<?php
$original = “They DON’T KnoW they’re SHOUTING”;
$lower = strtolower($original);
echo $lower;
?>
```

they don’t know they’re shouting

strupper() returns in all upper case string

```
<?php
$original = “make this link stand out”;
echo(“<B>strtoupper($original)</B>”);
?>
```

ucfirst() first letter capital

ucwords() capitalizes the first letter of each word in a string.

Arrays and Strings

explode(): The function explode converts string into array format .

We can use explode() to split a string into an array of strings

$inp = "This is a sentence with seven words";

$temp = explode(“ “, $inp);

print_r($temp);

Output:

Array([0] => This [1] => is [2] => a [3] => sentence [4] => with [5] => seven [6] => words)

implode(): To combine array of elements in to a string

- $p = array("Hello", "World,", "I", "am", "Here!");
- $g = ***implode(" ", $p)***;
- **Output:** Hello World, I am Here!

str_pad() : function pads a string to a new length.

str_pad(string,length,pad_string,pad_type)

Parameter	Description
string	**Required. Specifies the string to pad**
length	**Required. Specifies the new string length. If this value is less than the original length of the string, nothing will be done**
pad_string	**Optional. Specifies the string to use for padding. Default is whitespace**
pad_type	**Optional. Specifies what side to pad the string.Possible values:** **•STR_PAD_BOTH - Pad to both sides of the string. If not an even number, the right side gets the extra padding** **•STR_PAD_LEFT - Pad to the left side of the string** **•STR_PAD_RIGHT - Pad to the right side of the string. This is default**

```
<?php
$str = "Hello World";
echo str_pad($str,20,".");
?>
```

Hello World.....

```php
<?php
$str = "Hello World";
echo str_pad($str,20,".",STR_PAD_LEFT);
?>
```

.....Hello World

```php
<?php
$str = "Hello World";
echo
str_pad($str,20,".:",STR_PAD_BOTH);
?>
```

.:.:Hello World.:.:.

Branching/Conditional Control Structures

If-else

The syntax for if is:

if (*test)*
statement-1

with an optional else branch:

if (*test)*
statement-1
else
statement-2

elseif statement

If(test)
Statement-1
Elseif (test)
Statement -2

switch

switch(*expression)*
{
case *value-1:*
statement-1;
statement-2;
...
[break;]
case *value-2:*
statement-3;
statement-4;
...
[break;]
...
[default:
default-statement;]
}

Looping

while
while (*condition)*
statement

Do while
do *statement*
while (*expression);*

for
for (*initial-expression; termination-check; loop-end-expression)*
statement

For each
foreach (*array as value)*
{
code to be executed;
}

```
<?php
$arr=array("one", "two", "three");
foreach ($arr as $value)
{
echo "Value: " . $value . "<br />";
}
?>
```

foreach(array as key => value)
{
code to be executed;
}

```
<?php
$arr=array("one", "two", "three");
foreach ($arr as  $key=>$value)
{
echo "Value: " . $key.  $value . "<br />";
}
?>
```

Loop control

Break and continue

Break and continue offer an optional side exit from all the looping constructs, including while, do-while, and for:

- The break command exits the innermost loop construct that contains it.
- The continue command skips to the end of the current iteration of the innermost loop that contains it.

For example, the following code:

```
for ($x = 1; $x < 10; $x++)
{
// if $x is odd, break out
if ($x % 2 != 0)
break;
print("$x ");
}
```

prints nothing, because 1 is odd, which terminates the for loop immediately.

```
for ($x = 1; $x < 10; $x++)
{
// if $x is odd, skip this loop
if ($x % 2 != 0)
continue;
print("$x ");
}
```

Prints 2 4 6 8

Additional Material

Math functions

Abs : Absolute value

```
$abs = abs(-4.2); // $abs = 4.2; (double/float)
$abs2 = abs(5);   // $abs2 = 5; (integer)
$abs3 = abs(-5);  // $abs3 = 5; (integer)
```

Acos, acosh, asin, asinh, atan, atanh, cos, cosh, sin, sinh

Base_convert: Convert a number between arbitrary bases

```
$hexadecimal = 'A37334';
echo base_convert($hexadecimal, 16, 2);                    //10100011011100110010100
```

Bindec: Converts binary to decimal

```
echo bindec('110011') . "\n";        //51
echo bindec('000110011') . "\n";  //51
echo bindec('111');                        //7
```

Ceil: Rounds franctions up

```
echo ceil(4.3);    // 5
echo ceil(9.999);  // 10
echo ceil(-3.14);  // -3
```

Math functions

Decbin, dechex, decoct, hexdec, octdec

Deg2rad: converts number from degrees to the radian equivalent.

```
echo deg2rad(45); // 0.785398163397
```

Exp: Calculates the exponent of e

Expm1: Returns exp(number) -1

Floor: Rounds fractions down

```
echo floor(4.3);   // 4
echo floor(9.999); // 9
echo floor(-3.14); // -4
```

Getrandmax: Shows the largest possible random value

Rand: Generate random integer

```
echo rand() . "\n";      //7771
echo rand() . "\n";      //22264
echo rand(5, 15);      //71
```

Math functions

Hypot: Retuns sqrt(num1 * num1 + num2 * num2)

Is_nan: Determines whether a value is not a number. Returns TRUE if val is 'not a number', else FALSE

Log10: Base 10 logarithm

Log: Returns the natural logarithm

Max , min: Finds the highest value/lowest value

```
echo max(1, 3, 5, 6, 7);  // 7
echo max(array(2, 4, 5)); // 5
```

Pi: gets value of pi

Pow: Exponential expression

Round: Rounds a float

Sqrt: Square root

Converting to and from Strings

```
<?php
$float = 3.1415;
echo (string) $float;       // 3.1415
echo strval($float);                    // 3.1415
$value = 1 + "19.2";
echo $value;                                    //20.2
$text = "3.0";
$value = (float) $text;
echo $value/2.0;                        //1.5
?>
```

String to array:

str_split: Convert a string to an array

Parameters

String The input string. ***split_length*** Maximum length of the chunk.

```
<?php

$str = "Hello Friend";

$arr1 = str_split($str);
$arr2 = str_split($str, 3);

print_r($arr1);
print_r($arr2);

?>
```

Array ([0] => H [1] => e [2] => l [3] => l [4] => o [5] => [6] => F [7] => r [8] => i [9] => e [10] => n [11] => d)

Array ([0] => Hel [1] => lo [2] => Fri [3] => end)

str_word_count:

str_word_count — Return information about words used in a string

Parameters

String The string

Format Specify the return value of this function. The current supported values are:

0 - returns the number of words found

1 - returns an array containing all the words found inside the string

2 - returns an associative array, where the key is the numeric position of the word inside the string and the value is the actual word itself

Charlist A list of additional characters which will be considered as 'word'

```
<?php

$str = "Hello fri3nd, you're
      looking          good today!";

print_r(str_word_count($str, 1));
print_r(str_word_count($str, 2));
print_r(str_word_count($str, 1, 'àáãç3'));

echo str_word_count($str);

?>
```

```
Array ( [0] => Hello [1] => fri [2] => nd [3] => you're [4] =>
looking [5] => good [6] => today )

Array ( [0] => Hello [6] => fri [10] => nd [14] => you're [29] => looking
[46] => good [51] => today )

Array ( [0] => Hello [1] => fri3nd [2] => you're [3] => looking [4] =>
good [5] => today )

7
```

Thanks