SERVER-SIDE WEB PROGRAMMING
UNIT2: PROGRAMMING BASED ON EMBEDDED LANGUAGE

CFGS - DAW 2

JESUITAS
Sagrado Corazón
Jesuitas · Logroño

Index

3. Strings

- Most data you encounter as you program will be sequences of characters, or strings.
- For that reason, PHP has an extensive selection of functions for working with strings.

3.1. Quoting String Constants

- **Variable Interpolation**: Interpolation is the process of replacing variable names in the string with the values of those variables.

```php
$who = 'Kilroy';
$where = 'here';
echo "$who was $where";
```

```php
$n = 12;
echo "You are the {$n}th person";
```

3.1. Quoting String Constants

- In PHP strings are not repeatedly processed for interpolation. That means:

```php
$bar = 'this is not printed';
$foo = '$bar'; // single quotes
print("$foo"); // $bar will be printed
```

3.1. Quoting String Constants

- Single-quoted strings do not interpolate variables:

```php
$name = 'Fred';
$str = 'Hello, $name'; // single-quoted
echo $str; // Prints out: Hello, $name
```

  - The only escape sequences that work in single-quoted strings are \\ \':

```php
$name = 'Tim O\'Reilly';// escaped single quote
echo $name;
$path = 'C:\\WINDOWS'; // escaped backslash
echo $path;
$nope = '\n'; // not an escape
echo $nope;
```

3.1. Quoting String Constants

- Double-quoted strings interpolate variables and expand the many PHP escape sequences:

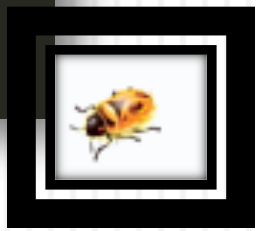
\"	Double quotes
\n	Newline
\r	Carriage return
\t	Tab
\\	Backslash
\$	Dollar sign
\{	Left brace
\}	Right brace
\[	Left bracket
\]	Right bracket
\0 through \777	ASCII character represented by octal value
\x0 through \xFF	ASCII character represented by hex value

3.1. Quoting String Constants

- You can easily put multiline strings into your program with a heredoc:

```php
<?php
    $clerihew = <<<EndOfQuote
    Sir Humphrey Davy
    Abominated gravy.
    He lived in the odium
    Of having discovered sodium.
EndOfQuote;
    echo $clerihew;
    //echo nl2br($clerihew);

?>
```



3.1. Quoting String Constants

- **When to use each one**:
  - Single quotes: when the character chain is constant.
  - Double quotes: when the character chain consists of a single line, require interpolation variables or spetial characters.
  - Heredoc: when the character chain consists of multiple lines and requiere interpolation variables or spetial characters. (Example: SQL sentences)

3.1. Quoting String Constants

A.2.12. Write a PHP program that sets the variable $first_name to your first name and $last_name to your last name. Print out a string containing your first and last name separated by a space. Also print out the length of that string.

A.2.13. Write a PHP program that computes the total cost of this restaurant meal: two hamburgers at $6.95 each, one chocolate milk shake at $2.90, and one cola at 75 cents. The sales tax rate is 8%.

3.2. Printing Strings

- echo:

```php
echo "Print";
echo ("Print");
echo "Print", "Print" , "Print" ;
// echo ("Print", "Print" , "Print"); This is not allowed
echo "\nPrint". "\nPrint" . "\nPrint"   ;
```

- print():

```php
if (print("test")) {
    print("It worked!");
}
```

3.2. Printing Strings

- printf(): outputs a string built by substituting values into a template:

```
$price = 5; $tax = 0.075;
printf('The dish costs $%.2f', $price * (1 + $tax));
```

3.2. Printing Strings

Specifier	Meaning
%	Displays the % character.
b	The argument is an integer and is displayed as a binary number.
c	The argument is an integer and is displayed as the character with that value.
d	The argument is an integer and is displayed as a decimal number.
e	The argument is a double and is displayed in scientific notation.
E	The argument is a double and is displayed in scientific notation using uppercase letters.
f	The argument is a floating-point number and is displayed as such in the current locale's format.
F	The argument is a floating-point number and is displayed as such.
g	The argument is a double and is displayed either in scientific notation (as with the %e type specifier) or as a floating-point number (as with the %f type specifier), whichever is shorter.
G	The argument is a double and is displayed either in scientific notation (as with the %E type specifier) or as a floating-point number (as with the %f type specifier), whichever is shorter.
o	The argument is an integer and is displayed as an octal (base-8) number.
s	The argument is a string and is displayed as such.
u	The argument is an unsigned integer and is displayed as a decimal number.
x	The argument is an integer and is displayed as a hexadecimal (base-16) number; lowercase letters are used.
X	The argument is an integer and is displayed as a hexadecimal (base-16) number; uppercase letters are used.

3.2. Printing Strings

A.2.14. Could you imagine how will the output look like? And Why?

```
printf('%.2f', 27.452);
printf('The hex value of %d is %x', 214, 214);
printf('Bond. James Bond. %03d.', 7);
$month=9;
$day=24;
$year=1983;
printf('%02d/%02d/%04d', $month, $day, $year);
printf('%.2f%% Complete', 3.2);
```

Here you will find the opposite from *trimming*… which is called *padding*.

3.2. Printing Strings

- printf():
  - The sprintf() function takes the same arguments as printf() but returns the built-up string instead of printing it.
  - This lets you save the string in a variable for later use:

```
$date = sprintf("%02d/%02d/%04d", $month, $day, $year);
```

3.2. Printing Strings

- print_r(): construct intelligently what is passed to it.

```php
$a = array('name' => 'Fred', 'age' => 35, 'wife' => 'Wilma');
print_r($a);
```

- Boolean values and NULL are not meaningfully displayed by print_r():

```php
print_r(true); // prints "1";
print_r(false); // prints "";
print_r(null); // prints "";
```

3.2. Printing Strings

- For this reason, var_dump() is preferred over print_r() for debugging. The var_dump() function displays any PHP value in a human-readable format:

```php
var_dump(true);
var_dump(false);
var_dump(null);
var_dump(array('name' => "Fred", 'age' => 35));
class P {
    var $name = 'Nat';
// ...
}
$p = new P;
var_dump($p);
```

3.3. Accessing Individual Characters


```php
$string = 'Hello';
for ($i=0; $i < strlen($string); $i++) {
    printf("The %dth character is %s\n", $i, $string[$i]);
}
```

3.4. Cleaning Strings

- Often, the strings we get from files or users need to be cleaned up before we can use them.
- Two common problems with raw data are:
  1. the presence of extraneous whitespace
  2. incorrect capitalization (uppercase versus lowercase).

3.4. Cleaning Strings

- Removing Whitespace:

$trimmed = trim(string [, charlist]);

$trimmed = ltrim(string [, charlist]);

$trimmed = rtrim(string [, charlist]);

- Default characters removed by trim(), ltrim(), and rtrim():

Character	ASCII value	Meaning
" "	0x20	Space
"\t"	0x09	Tab
"\n"	0x0A	Newline (line feed)
"\r"	0x0D	Carriage return
"\0"	0x00	NUL-byte
"\x0B"	0x0B	Vertical tab

3.4. Cleaning Strings


```
$title = " Programming PHP \n";
$str1 = ltrim($title); // $str1 is "Programming PHP \n"
$str2 = rtrim($title); // $str2 is " Programming PHP"
$str3 = trim($title); // $str3 is "Programming PHP"
```

```
// $_POST['zipcode'] holds the value of the submitted form parameter
// "zipcode"
$zipcode = trim($_POST['zipcode']);
// Now $zipcode holds that value, with any leading or trailing spaces
// removed
$zip_length = strlen($zipcode);
// Complain if the zip code is not 5 characters long
if ($zip_length != 5) {
    print "Please enter a zip code that is 5 characters long.";
}
```

```
if (strlen(trim($_POST['zipcode'])) != 5) {
    print "Please enter a zip code that is 5 characters long.";
}
```

3.4. Cleaning Strings

- Changing Case:

```
$string1 = "FRED flintstone";
$string2 = "barney rubble";
print(strtolower($string1));
print(strtoupper($string1));
print(ucfirst($string2));
print(ucwords($string2));
print(ucwords(strtolower($string1)));
```

- This will be useful in order to compare strings…

3.5. Comparing Strings

1. Exact Comparisons:
   - You can compare two strings for equality with the == and === operators

```php
$o1 = 3;
$o2 = "3";
if ($o1 == $o2) {
echo("== returns true<br>");
}
if ($o1 === $o2) {
echo("=== returns true<br>");
}
```

3.5. Comparing Strings

- The comparison operators (<, <=, >, >=) also work on strings:

```
$him = "Fred";
$her = "Wilma";
if ($him < $her) {
print "{$him} comes before {$her} in the alphabet.\n";
}
```

- However, the comparison operators give unexpected results when comparing strings and numbers:

```
$string = "PHP Rocks";
$number = 5;
if ($string < $number) {
    echo("{$string} < {$number}");
}
```

3.5. Comparing Strings

- To explicitly compare two strings as strings, casting numbers to strings if necessary, use the strcmp() function:

```
$n = strcmp("PHP Rocks", 5);
echo($n);
```

- A variation on strcmp() is strcasecmp(), which converts strings to lowercase before comparing them. Its arguments and return values are the same as those for strcmp():

```
$n = strcasecmp("Fred", "frED"); // $n is 0
```

3.5. Comparing Strings


```
if ($_POST['email'] == 'president@whitehouse.gov') {
    print "Welcome, US President.";
}

if (strcasecmp($_POST['email'], 'president@whitehouse.gov') == 0) {
    print "Welcome back, US President.";
}
```

A.2.15. Could you add some trimming in order to make it more secure? Test it creating a little form example.

3.5. Comparing Strings

- Another variation on string comparison is to compare only the first few characters of the string:

```
$relationship = strncmp(string_1, string_2, len);
$relationship = strncasecmp(string_1, string_2, len);
```

3.5. Comparing Strings

2. Approximate Equality:

- PHP provides several functions that let you test whether two strings are approximately equal:

```
$soundexCode = soundex($string);
$metaphoneCode = metaphone($string);
$inCommon = similar_text($string_1, $string_2 [, $percentage ]);
$similarity = levenshtein($string_1, $string_2);
$similarity = levenshtein($string_1, $string_2 [, $cost_ins, $cost_rep, $cost_del ]);
```

3.6. Manipulating and Searching Strings

1. Substrings:

- If you know where the data that you are interested in lies in a larger string, you can copy it out with the substr() function:

```
// Grab the first 30 bytes of $_POST['comments']
print substr($_POST['comments'], 0, 30);
// Add an ellipsis
print '...';
```

- For example, you may only want to display the beginnings of messages on a summary page.

3.6. Manipulating and Searching Strings

- To learn how many times a smaller string occurs in a larger one, use substr_count():

```
$sketch = <<< EndOfSketch
Well, there's egg and bacon; egg sausage and bacon; egg and spam;
egg bacon and spam; egg bacon sausage and spam; spam bacon sausage
and spam; spam egg spam spam bacon and spam; spam sausage spam spam
bacon spam tomato and spam;
EndOfSketch;
$count = substr_count($sketch, "spam");
print("The word spam occurs {$count} times.");
```

3.6. Manipulating and Searching Strings

- The substr_replace() function permits many kinds of string modifications:

```php
$greeting = "good morning citizen";
$farewell = substr_replace($greeting, "bye", 5, 7);
// $farewell is "good bye citizen"
$farewell = substr_replace($farewell, "kind ", 9, 0);
// $farewell is "good bye kind citizen"
$farewell = substr_replace($farewell, "", 8);
// $farewell is "good bye"
$farewell = substr_replace($farewell, "now it's time to say ", 0, 0);
// $farewell is "now it's time to say good bye"'
$farewell = substr_replace($farewell, "riddance", -3);
// $farewell is "now it's time to say good riddance"
$farewell = substr_replace($farewell, "", -8, -5);
// $farewell is "now it's time to say good dance"
```

3.6. Manipulating and Searching Strings

2. Miscellaneous String Functions:
   - *$string = **strrev**(string);*
   - *$repeated = **str_repeat**(string, count);*
   - *$padded = **str_pad**(to_pad, length [, with [, pad_type]]);*

A.2.16. Find out the functionality of these methods and write some examples.

3.6. Manipulating and Searching Strings

3. Decomposing a String: PHP provides several functions to let you break a string into smaller components.

   I. *Exploding and imploding:*

```php
$input = 'Fred,25,Wilma';
$fields = explode(',', $input);
// $fields is array('Fred', '25', 'Wilma')
$fields = explode(',', $input, 2);
// $fields is array('Fred', '25,Wilma')
```

```php
$fields = array('Fred', '25', 'Wilma');
$string = implode(',', $fields); // $string is 'Fred,25,Wilma'
```

3.6. Manipulating and Searching Strings

II. *Tokenizing:*

```
$string = "Fred,Flintstone,35,Wilma";
$token = strtok($string, ",");
while ($token !== false) {
    echo("{$token}<br />");
    $token = strtok(",");
}
```

1

2

- Call *strtok()* with two arguments to reinitialize the iterator. This restarts the tokenizer

3.6. Manipulating and Searching Strings

4. String-Searching Functions: All the string-searching functions return false if they can't find the substring you specified.
   - *strpos(), strrpos()*
   - *strstr(), strchr()*
   - *strspn(), strcspn()*

A.2.17. Find out the functionality of these methods and write some examples.

3.6. Manipulating and Searching Strings

- The parse_url() function returns an array of components of a URL:

```
$bits = parse_url("http://me:secret@example.com/cgi-bin/board?user=fred");
print_r($bits);
```

- The possible keys of the hash are scheme, host, port, user, pass, path, query, and fragment.

3.7. Regular expresions

- If you need more complex searching functionality than the previous methods provide, you can use regular expressions.
- A regular expression is a string that represents a pattern.

3.7. Regular expresions

1. The Basics:
   - Most characters in a regular expression are literal characters:
     - For instance: searching "/cow/" in the string "Dave was a cowhand" → you get a match because "cow" occurs in that string.
   - Some characters have special meanings in regular expressions:
     - For instance, a (^) at the beginning of a regular expression indicates that it must match the beginning of the string.

```
preg_match("/^cow/", "Dave was a cowhand"); // returns false
preg_match("/^cow/", "cowabunga!"); // returns true
```

3.7. Regular expresions

- Similarly, a dollar sign ($) at the end of a regular expression means that it must match the end of the string:

```
preg_match("/cow$/", "Dave was a cowhand"); // returns false
preg_match("/cow$/", "Don't have a cow"); // returns true
```

- A period (.) in a regular expression matches any single character:

```
preg_match("/c.t/", "cat"); // returns true
preg_match("/c.t/", "cut"); // returns true
preg_match("/c.t/", "c t"); // returns true
preg_match("/c.t/", "bat"); // returns false
preg_match("/c.t/", "ct"); // returns false
```

3.7. Regular expresions

- If you want to match one of these special characters (called a metacharacter), you have to escape it with a backslash:

```
preg_match("/\$5\.00", "Your bill is $5.00 exactly"); // returns true
preg_match("/$5.00", "Your bill is $5.00 exactly"); // returns false
```

3.7. Regular expresions

2. Character classes:

- To specify a set of acceptable characters in your pattern:

```
preg_match("/c[aeiou]t/", "I cut my hand"); // returns true
preg_match("/c[aeiou]t/", "This crusty cat"); // returns true
preg_match("/c[aeiou]t/", "What cart?"); // returns false
preg_match("/c[aeiou]t/", "14ct gold"); // returns false_
```

- You can negate a character class with a caret (^) at the start:

```
preg_match("/c[^aeiou]t/", "I cut my hand"); // returns false
preg_match("/c[^aeiou]t/", "Reboot chthon"); // returns true
preg_match("/c[^aeiou]t/", "14ct gold"); // returns false
```

3.7. Regular expresions

- You can define a range of characters with a hyphen (-). This simplifies character classes like “all letters” and “all digits”:

```
preg_match("/[0-9]%/", "we are 25% complete"); // returns true
preg_match("/[0123456789]%/", "we are 25% complete"); // returns true
preg_match("/[a-z]t/", "11th"); // returns false
preg_match("/[a-z]t/", "cat"); // returns true
preg_match("/[a-z]t/", "PIT"); // returns false
preg_match("/[a-zA-Z]!/", "11!"); // returns false
preg_match("/[a-zA-Z]!/", "stop!"); // returns true
```

3.7. Regular expresions

3. Alternatives:

- You can use the vertical pipe (|) character to specify alternatives in a regular expression:

```
preg_match("/cat|dog/", "the cat rubbed my legs"); // returns true
preg_match("/cat|dog/", "the dog rubbed my legs"); // returns true
preg_match("/cat|dog/", "the rabbit rubbed my legs"); // returns false
```

- You can combine character classes and alternation to, for example, check for strings that don't start with a capital letter:

```
preg_match("/^([a-z]|[0-9])/", "The quick brown fox"); // returns false
preg_match("/^([a-z]|[0-9])/", "jumped over"); // returns true
preg_match("/^([a-z]|[0-9])/", "10 lazy dogs"); // returns true
```

3.7. Regular expresions

4. Repeating Sequences:

- To specify a repeating pattern, you use something called a quantifier:

Quantifier	Meaning
?	0 or 1
*	0 or more
+	1 or more
{ *n* }	Exactly *n* times
{ *n* , *m* }	At least *n*, no more than *m* times
{ *n* , }	At least *n* times

```
preg_match("/ca+t/", "caaaaaaat"); // returns true
preg_match("/ca+t/", "ct"); // returns false
preg_match("/ca?t/", "caaaaaaat"); // returns false
preg_match("/ca*t/", "ct"); // returns true
```

3.7. Regular expresions

- With quantifiers and character classes, we can actually do something useful, like matching valid U.S. telephone numbers:

```
preg_match("/[0-9]{3}-[0-9]{3}-[0-9]{4}/", "303-555-1212"); // returns true
preg_match("/[0-9]{3}-[0-9]{3}-[0-9]{4}/", "64-9-555-1234"); // returns false
```