DT228/2 Web Development

Basic PHP

Philosphy of PHP

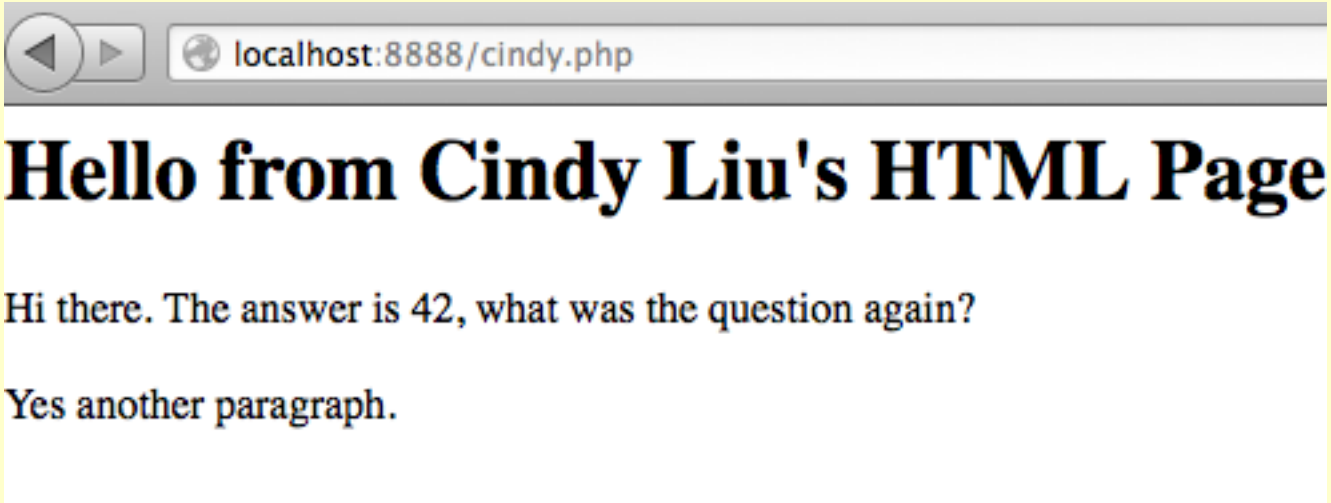
- You are a responsible and intelligent programmer
- You know what you want to do
- Some flexibility in syntax is OK - style choices are OK
- Lets make this as convienent as possible
- Sometimes errors fail silently

PHP examples

```
<h1>Hello from Cindy Liu’s HTML Page</h1>
<p>
<?php
echo "Hi there.\n";
$answer = 6 * 7;
echo "The answer is $answer, what ";
echo "was the question again?\n";
?>
</p>
<p>Yes another paragraph.</p>
```

PHP examples

```
<h1>Hello from Cindy Liu’s HTML Page</h1>
<p>
<?php
echo "Hi there.\n";
$answer = 6 * 7;
echo "The answer is $answer, what ";
echo "was the question again?\n";
?>
</p>
<p>Yes another paragraph.</p>
```

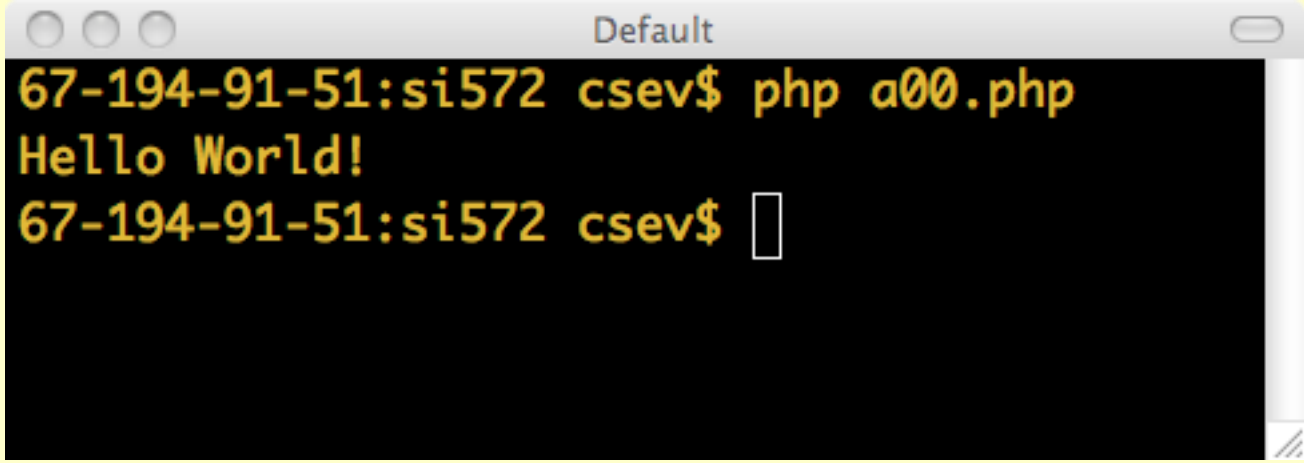


PHP From the Command Line

- You can run PHP from the command line - the output simply comes out on the Terminal

- It does not have to be part of a request-response cycle

```
<?php
echo("Hello World!");
echo("\n");
?>
```



Key Words

abstract and array() as break case catch class
clone const continue declare default do else
elseif end declare endfor endforeach endif
endswitch endwhile extends final for foreach
function global goto if implements interface
instanceof namespace new or private protected
public static switch $this throw try use var
while xor

http://php.net/manual/en/reserved.php

Variable Names

- Start with a dollar sign ($) followed by a letter or underscore, followed by any number of letters, numbers, or underscores

- Case matters

```
$abc = 12;                  abc = 12;
$total = 0;                 $2php = 0;
$largest_so_far = 0;        $bad-punc = 0;
```

http://php.net/manual/en/language.variables.basics.php

Variable Names Weirdness

Things that look like variables but are missing a dollar sign can be confusing

```
$x = 2;
$y = x + 5;
print $y;
```

5

```
$x = 2;
y = $x + 5;
print $x;
```

Parse error

Expressions

- Completely normal like other languages (+ - / *)
- More agressive implicit type conversion

```
<?php
$x = "15" + 27;
echo($x);
echo("\n");
?>
```

```
42
```

Output

- Echo is a language construct - can be treated like a function with one parameter. Without parenthesis, it accepts multiple parameters.

- Print is a function - only one parameter but parenthesis are optional so it can look like a language construct

```
<?php
   $x = "15" + 27;
   Echo $x;
   Echo ("\n");
   Echo $x, "\n";
   Print $x;
   Print "\n";
   Print ($x);
   print("\n");
?>
```

Conditional - if

- Logical operators (== != < > <= >= and or !)
- Curly braces

```
<?php
$ans = 42;
if ( $ans == 42 ) {
print "Hello world!\n";
} else {
print "Wrong answer\n";
}
?>
```

```
Hello World!
```

White space does not matter

```
<?php
$ans = 42;
if ( $ans == 42 ) {
print "Hello world!\n";
} else {
print "Wrong answer\n";
}
?>

<?php $ans = 42; if ( $ans == 42 ) { print
"Hello world!\n"; } else { print "Wrong
answer\n"; }
?>
```

What style do you prefer

```php
<?php
$ans = 42;
if ( $ans == 42 ) {
print "Hello
world!\n";
} else {
print "Wrong
answer\n";
}
?>
```

```php
<?php
$ans = 42;
if ( $ans == 42 )
{
print "Hello
world!\n";
}
else
{
print "Wrong
answer\n";
}
?>
```

Associative Arrays

- Like Python Dictonaries+Lists - but more powerful
- Can be key => value or simply indexed by numbers
- Ignore two-dimensional arrays for now...

Integer Indices

```
<?php
    $stuff = array("Hi", "There");
    echo $stuff[1], "\n";
?>
```

```
There
```

Integer Indices

```
<?php
  $stuff = Array();
  $stuff[] = "Hello";
  $stuff[] = "World";

  Echo $stuff[1], "\n";
?>
```

```
World
```

Integer Indices

```
<?php
  $stuff = Array();
  $stuff[2] = "Hello";
  $stuff[9] = "World";

  Echo $stuff[9], "\n";
?>
```

```
World
```

Key/Value

```
<?php
  $stuff = array("name" => "Liu",
                 "course" => "DT228");

  Echo $stuff["course"], "\n";
?>
```

```
DT228
```

Dumping an Array

- The function print_r() dumps out PHP data - it is used mostly for debugging

```
<?php
  $stuff = array("name" => "Liu",
                 "course" => "DT228");
  print_r($stuff)
  ;
?>
```

```
Array
(
[name] => Liu
[course] => DT228
)
```

Dumping an Array

- The function print_r() dumps out PHP data - it is used mostly for debugging

```
<?php
  $stuff = Array();
  $stuff[2] = "Hello";
  $stuff[9] = "World";
  print_r($stuff);
?>
```

```
Array
(
[2] => Hello
[9] => World
)
```

var_dump .vs. print_r

- The var_dump function displays structured information about variables/expressions including its **type** and **value**. Arrays are explored recursively with values indented to show structure. It also shows which array values and object properties are references

```
<?php
$stuff = array("name" => "Liu",
"course" => "DT228");
var_dump($stuff);
?>
```

```
array(2) {
["name"]=>
string(5) ”Liu"
["course"]=>
string(5) ”DT228"
}
```

var_dump is more verbose

```
<?php
  $thing = FALSE;
  echo("One\n");
  print_r($thing);
  echo("Two\n");
  var_dump($thing);
?>
```

```
One
Two
bool(false)
```

Looping Through an Array

```
<?php
  $stuff = array("name" => "Liu",
                 "course" => "DT228");
  foreach($stuff as $k => $v ) {
  echo "Key=", $k ,"Val=", $v ,"\n";
  }
?>
```

```
Key=name Val=Liu
Key=course Val=DT228
```

Variable Name Weirdness

- Things that look like variables but are missing a dollar sign as an array index are unpredictable...

```
$x = 5;
$y = Array("x" => "Hello");
print $y[x];
```

```
Hello
```

Strings

- String literals can use single quotes or double quotes
- The backslash (\) is used as an "escape" character
- Strings can span multiple lines - the newline is part of the String
- In double-quoted strings variable values are expanded

Single Quote

```
<?php
echo 'this is a simple string';

echo 'You can also have embedded newlines in
strings this way as it is
okay to do';

// Outputs: Arnold once said: "I'll be back"
echo 'Arnold once said: "I\'ll be back"';

// Outputs: This will not expand: \n a newline
echo 'This will not expand: \n a newline';

// Outputs: Variables do not $expand $either
echo 'Variables do not $expand $either';
?>
```

Double Quote

```
<?php
Echo "this is a simple string";

Echo "You can also have embedded newlines in
strings this way as it is
okay to do";

// Outputs: This will expand:
//          a newline
Echo "This will expand: \n a newline";

// Outputs: Variables do 12
$expand = 12;
Echo "Variables do $expand\n";
?>
```

Comments

```
<?php
echo 'This is a test';
// This is a c++ style comment
/* This is a multi line comment
yet another line of comment */
echo 'This is yet another test';
echo 'One Final Test';
# This is a shell-style comment
?>
```

Strings

- Those “\n” and others in Single Quote and Double Quote are designed to use in output plain strings

- In HTML, you should use “nl2br”, or just use “<br/>”

Expressions

- Expressions evaluate to a value. The value can be a string, number, boolean, etc...

- Expressions often use operations and function calls, and there is an order of evaluation when there is more than one operator in an expression

- Expressions can also produce objects like arrays

Expressions

Operator Precedence

High

Operator(s)	Type
( )	Parentheses
++ --	Increment/Decrement
!	Logical
* / %	Arithmetic
+ - .	Arithmetic and String
<< >>	Bitwise
< <= > >= <>	Comparison
== != === !==	Comparison
&	Bitwise (and references)
^	Bitwise
|	Bitwise
&&	Logical
||	Logical
? :	Ternary
= += -= *= /= .= %= &= != ^= <<= >>=	Assignment
and	Logical
xor	Logical
or	Logical

Low

Operators of Note

- Increment / Decrement (++ --)
- String concatenation (.)
- Equality (== !=)
- Identity (=== !==)
- Ternary (? :)
- Side-effect Assignment (+= -= .= etc.)
- Ignore the rarely-used bitwise operators (>> << ^ | &)

Increment / Decrement

- These operators allow you to both retrieve and increment / decrement a variable

- They are generally avoided in civilized code.

```
$x = 12;
$y = 15 + $x++;
echo "x is $x and y is $y \n";
```

```
x is 13 and y is 27
```

Increment / Decrement

- Instead, we could use the code below to serve the same purpose

```
$x = 12;
$y = 15 + $x;
$x = $x + 1;
echo "x is $x and y is $y \n";
```

```
x is 13 and y is 27
```

String Concatenation

- PHP uses the period character for concatenation because the plus character would instruct PHP to the best it could do to add the two things together, converting if necessary

```
$a = 'Hello ' . 'World!';
echo $a . "\n";                    Hello World!

$a = '12' . '13';
echo $a . "\n";                    1213
```

Equality(==) versus Identity(===)

- The equality operator (==) in PHP is far more agressive than in most other languages when it comes to data conversion during expression evaluation.

```
if ( 123 == "123" ) print ("Equality 1\n");
if ( 123 == "100"+23 ) print ("Equality 2\n");
if ( FALSE == "0" ) print ("Equality 3\n");
if ( (5 < 6) == "2"-"1" ) print ("Equality 4\n");
if ( (5 < 6) === TRUE ) print ("Equality 5\n");
```

Equality(==) versus Identity(===)

```
// "===" means that they are identical
// "==" means that they are equal
// "!=" means that they aren't equal.

          false   null    array()  0      "0"     0x0     "0x0"   "000"   "0000"
false     ===     ==      ==       ==     ==      ==      !=      !=      !=
null      ==      ===     ==       ==     !=      ==      !=      !=      !=
array()   ==      ==      ===      !=     !=      !=      !=      !=      !=
0         ==      ==      !=       ===    ==      ===     ==      ==      ==
"0"       ==      !=      !=       ==     ===     ==      ==      ==      ==
0x0       ==      ==      !=       ===    ==      ===     ==      ==      ==
"0x0"     !=      !=      !=       ==     ==      ==      ===     ==      ==
"000"     !=      !=      !=       ==     ==      ==      ==      ===     ==
"0000"    !=      !=      !=       ==     ==      ==      ==      ==      ===
```

Equality(==) versus Identity(===)

- The === operator works the same as the == operator but requires that its operands have not only the same value, but also the same data type.

```
$vv = "Hello World!";
echo "First:" . strpos($vv, "Wo") . "<br/>";
echo "Second: " . strpos($vv, "He") . "<br/>";
echo "Third: " . strpos($vv, "ZZ") . "<br/>";
if (strpos($vv, "He") == FALSE ) echo "Wrong A<br/>";
if (strpos($vv, "ZZ") == FALSE ) echo "Right B<br/>";
if (strpos($vv, "He")!== FALSE ) echo "Right C<br/>";
if (strpos($vv, "ZZ") === FALSE ) echo "Right D<br/>";
print_r(FALSE); print FALSE;
echo "Where were they?<br/>";
```

First:6
Second: 0
Third:
Wrong A
Right B
Right C
Right D
Where were they?

Beware FALSE variables. They are detectable but not visible...

Ternary

- The ternary operator comes from C. It allows conditional expressions. It is like a one-line if-then-else . Like all "contraction" syntaxes, we use it carefully.

```
$www = 123;
$msg = $www > 100 ? "Large" : "Small" ;
echo "First: $msg \n";
$msg = ( $www % 2 == 0 ) ? "Even" : "Odd";
echo "Second: $msg \n";
$msg = ( $www % 2 ) ? "Odd" : "Even";
echo "Third: $msg \n";?>
```

```
First: Large
Second: Odd
Third: Odd
```

Side-Effect Assignment

- These are pure contractions. Civilized programmers use them sparingly.

```
echo "\n";
$out = "Hello";
$out = $out . " ";
$out .= "World!";
$out .= "\n";
echo $out;
$count = 0;
$count += 1;
echo "Count: $count\n";
```

```
Hello World!
Count: 1
```