PHP Syntax

PHP Syntax

- PHP is a great example of a commonly-used modern programming language.
  - C was first released in 1972, PHP in 1995.
- PHP is an excellent language choice for software that requires an easy way to do things that, in C, are really complicated.
  - String manipulation
  - Networking
- Fortunately, PHP is heavily inspired by C (in fact, the PHP interpreter is written in C) and so the syntax should be pretty familiar.

PHP Syntax

- To start writing PHP, open up a file with the .php file extension.

- All PHP code inside of that file must be enclosed in the PHP delimiters.

```
<?php

?>
```

- If not, when the code is run, everything outside of those delimiters will simply be printed out verbatim in your program.

PHP Syntax

- Unlike C, which is a **compiled** language, PHP is an **interpreted** language.
  - Get to skip the step of compiling our code, with the trade off that PHP code needs to be converted to 0s and 1s on the fly, which might slow the program down.

- The fact that the PHP interpreter effectively “discards” non-PHP can actually be used to our advantage though, particularly when we start to mix PHP and HTML in the context of web programming.

PHP Syntax

- **Variables**

- PHP variables have two big differences from C.
  - No type specifier.
  - All names start with $.

PHP Syntax

- **Variables**

- PHP variables have two big differences from C.
  - No type specifier.
  - All names start with $.

```
int x = 54;
```

PHP Syntax

- **Variables**

- PHP variables have two big differences from C.
  - No type specifier.
  - All names start with $.

~~`int x = 54;`~~

PHP Syntax

- **Variables**

- PHP variables have two big differences from C.
  - No type specifier.
  - All names start with $.

```
$x = 54;
```

PHP Syntax

- **Variables**

- PHP variables have two big differences from C.
  - No type specifier.
  - All names start with $.

```
string phrase = “This is CS50”;
```

PHP Syntax

- **Variables**

- PHP variables have two big differences from C.
  - No type specifier.
  - All names start with $.

~~`string phrase = “This is CS50”;`~~

PHP Syntax

- **Variables**

- PHP variables have two big differences from C.
  - No type specifier.
  - All names start with $.

```
$phrase = “This is CS50”;
```

PHP Syntax

- **Conditionals**

- All of the old favorites from C are still available for you to use.

PHP Syntax

- **Conditionals**

- All of the old favorites from C are still available for you to use.

```
if ($y < 43 || $z == 15)
{

}
```

PHP Syntax

- **Conditionals**

- All of the old favorites from C are still available for you to use.

```
if ($y < 43 || $z == 15)
{

}
else
{

}
```

PHP Syntax

- **Conditionals**

- All of the old favorites from C are still available for you to use.

```
if ($coursenum == 50)
{

}
else if ($name == “David”)
{

}
```

PHP Syntax

- **Conditionals**

- All of the old favorites from C are still available for you to use.

```
if ($coursenum == 50)
{

}
else if ($name == “David”)
{

}
```

PHP Syntax

- **Conditionals**

- All of the old favorites from C are still available for you to use.

```
if ($coursenum == 50)
{

}
elseif ($name == “David”)
{

}
```

PHP Syntax

- **Conditionals**

- All of the old favorites from C are still available for you to use.

```
$var = ‘A’;
$letter = ctype_alpha($var) ? true : false;
```

PHP Syntax

- **Conditionals**

- All of the old favorites from C are still available for you to use.

```
$state = readline(“Your state, please: ”);
switch($state)
{
   case “MA”: case “Mass.”: case “Massachusetts”:
      // stuff
      break;
   default:
      echo(“I don’t understand your input!”);
}
```

PHP Syntax

- **Conditionals**

- All of the old favorites from C are still available for you to use.

```
$state = readline(“Your state, please: ”);
switch($state)
{
   case “MA”: case “Mass.”: case “Massachusetts”:
      // stuff
      break;
   default:
      echo(“I don’t understand your input!”);
}
```

PHP Syntax

- **Loops**

- All of the old favorites from C are still available for you to use.

PHP Syntax

- **Loops**

- All of the old favorites from C are still available for you to use.

```
$counter = 1;
while($counter <= 100)
{
    print($counter);
    $counter++;
}
```

PHP Syntax

- **Loops**

- All of the old favorites from C are still available for you to use.

```
$counter = 1;
do
{
    print($counter);
    $counter++;
}
while($counter <= 100);
```

PHP Syntax

- **Loops**

- All of the old favorites from C are still available for you to use.

```
for ($counter = 1; $counter <= 100; $counter++)
    print($counter);
```

PHP Syntax

- **Arrays**

- Here's where things really start to get a lot better than C.

- PHP arrays are **not** fixed in size; they can grow or shrink as needed, and you can always tack extra elements onto your array and splice things in and out easily.

- Because PHP is loosely typed, your arrays can also mix different data types together.

PHP Syntax

- **Arrays**

- Declaring an array is pretty straightforward.

```
$nums = array(1, 2, 3, 4);
```

PHP Syntax

- **Arrays**

- Declaring an array is pretty straightforward… though there is more than one way to do it.

```
$nums = [1, 2, 3, 4];
```

PHP Syntax

- **Arrays**

- Tacking on to an existing array can be done a few ways:

```
$nums = [1, 2, 3, 4];
$nums[4] = 5;
```

PHP Syntax

- **Arrays**

- Tacking on to an existing array can be done a few ways:

```
$nums = [1, 2, 3, 4];
$nums[] = 5;
```

PHP Syntax

- **Arrays**

- Tacking on to an existing array can be done a few ways:

```
$nums = [1, 2, 3, 4];
array_push($nums, 5);
```

PHP Syntax

- **Arrays**

- Recall that our arrays also do not have to always contain elements of the same data type.

```
$nums = [1, true, 3, 4];
array_push($nums, “five”);
```

PHP Syntax

- **Associative arrays**

- PHP also has built in support for **associative arrays**, allowing you to specify array indices with words or phrases (*keys*), instead of integers, which you were restricted to in C.

PHP Syntax

- **Associative arrays**

```
$pizzas = [
    “cheese” => 8.99,
    “pepperoni” => 9.99,
    “vegetable” => 10.99,
    “buffalo chicken” => 11.99
];
```

PHP Syntax

- **Associative arrays**

```
$pizzas = [
    “cheese” => 8.99,
    “pepperoni” => 9.99,
    “vegetable” => 10.99,
    “buffalo chicken” => 11.99
];
```

PHP Syntax

- **Associative arrays**

```
$pizzas = [
    “cheese” => 8.99,
    “pepperoni” => 9.99,
    “vegetable” => 10.99,
    “buffalo chicken” => 11.99
];
```

PHP Syntax

- **Associative arrays**

```
$pizzas = [
    “cheese” => 8.99,
    “pepperoni” => 9.99,
    “vegetable” => 10.99,
    “buffalo chicken” => 11.99
];
```

PHP Syntax

- **Associative arrays**

```
$pizza[“cheese”] = 7.99;
```

PHP Syntax

- **Associative arrays**

```
$pizza[“cheese”] = 7.99;

if ($pizza[“vegetable”] < 12)
{

}
```

PHP Syntax

- **Associative arrays**

```
$pizza[“cheese”] = 7.99;

if ($pizza[“vegetable”] < 12)
{

}

$pizza[“bacon”] = 13.49;
```

PHP Syntax

- **Associative arrays**

- PHP also has built in support for **associative arrays**, allowing you to specify array indices with words or phrases (*keys*), instead of integers, which you were restricted to in C.

- But this creates a somewhat new problem… how do we iterate through an associative array? We don't have indexes ranging from [0, *n*-1] anymore.

PHP Syntax

- **Loops (redux)**

- PHP also includes a new kind of loop called **foreach** that allows you to iterate across any array, but is particularly useful for associative arrays.

```
foreach($array as $key)
{
    // use $key in here as a stand-in for $array[$i]
}
```

PHP Syntax

- **Loops (redux)**

```
$pizzas = [
    “cheese” => 8.99,
    “pepperoni” => 9.99,
    “vegetable” => 10.99,
    “buffalo chicken” => 11.99
];
```

PHP Syntax

- **Loops (redux)**

```
$pizzas = [
    “cheese” => 8.99,
    “pepperoni” => 9.99,
    “vegetable” => 10.99,
    “buffalo chicken” => 11.99
];
```

```
foreach($pizzas as $pizza)
{
   print($pizza);
   print(“\n”);
}
```

PHP Syntax

- **Loops (redux)**

```
$pizzas = [
    “cheese” => 8.99,
    “pepperoni” => 9.99,
    “vegetable” => 10.99,
    “buffalo chicken” => 11.99
];
```

```
foreach($pizzas as $pizza)
{
    print($pizza);
    print(“\n”);
}
```

PHP Syntax

- **Loops (redux)**

```
$pizzas = [
    “cheese” => 8.99,
    “pepperoni” => 9.99,
    “vegetable” => 10.99,
    “buffalo chicken” => 11.99
];
```

```
foreach($pizzas as $pizza)
{
   print($pizza);
   print(“\n”);
}
```

```
8.99
9.99
10.99
11.99
```

PHP Syntax

- **Loops (redux)**

```
$pizzas = [
    “cheese” => 8.99,
    “pepperoni” => 9.99,
    “vegetable” => 10.99,
    “buffalo chicken” => 11.99
];

foreach($pizzas as $topping => $price)
{

}
```

PHP Syntax

- **Loops (redux)**

```
$pizzas = [
    “cheese” => 8.99,
    “pepperoni” => 9.99,
    “vegetable” => 10.99,
    “buffalo chicken” => 11.99
];

foreach($pizzas as $topping => $price)
{
    print(“A whole ”);
    print($topping);
    print(“ pizza costs $”);
    print($price);
    print(“.\n”);
}
```

PHP Syntax

```
A whole cheese pizza costs $8.99.
A whole pepperoni pizza costs $9.99.
A whole vegetable pizza costs $10.99.
A whole buffalo chicken pizza costs $11.99.
```

PHP Syntax

- **Printing and variable interpolation**

- We've already seen a few ways to print out information to the user.
  - `print()`
  - `echo()`

- Suffice it to say, PHP has a lot of ways to print, including ways to print out substituted variables in the middle of strings.

PHP Syntax

- **Printing and variable interpolation**

```
print(“A whole ”);
print($topping);
print(“ pizza costs $”);
print($price);
print(“.\n”);
```

PHP Syntax

- **Printing and variable interpolation**

```
printf(“A whole %s pizza costs $%s.\n”,
        $topping, $price);
```

PHP Syntax

- **Printing and variable interpolation**

```
printf("A whole %s pizza costs $%s.\n",
       $topping, $price);
```

PHP Syntax

- **Printing and variable interpolation**

```
echo(“A whole {$topping} pizza costs
      \${$price}.\n”);
```

PHP Syntax

- **Printing and variable interpolation**

```
echo(“A whole {$topping} pizza costs
      \${$price}.\n”);
```

PHP Syntax

- **Printing and variable interpolation**

```
echo(“A whole {$topping} pizza costs
      \${$price}.\n”);
```

PHP Syntax

- **Printing and variable interpolation**

```
print(“A whole {$topping} pizza costs
        \${$price}.\n”);
```

PHP Syntax

- **Printing and variable interpolation**

```
print(“A whole ” . $topping .
      “ pizza costs $” . $price . “.\n”);
```

PHP Syntax

- **Printing and variable interpolation**

```
print(“A whole ” . $topping .
      “ pizza costs $” . $price . “.\n”);
```

PHP Syntax

- **Printing and variable interpolation**

```
print(“A whole ” . $topping .
      “ pizza costs $” . $price . “.\n”);
```

PHP Syntax

- **Functions**

- PHP has support for functions as well. Like variables, we don't need to specify the return type of the function (because it doesn't matter), nor the data types of any parameters (ditto).

- All functions are introduced with the `function` keyword.
  - Also, no need for `main()`, since the interpreter just reads from top to bottom!

PHP Syntax

- **Functions**

```
function hard_square($x)
{
    $result = 0;
    for ($i = 0; $i < $x; $i++)
        $result += $x;
    return $result;
}
```

PHP Syntax

```
<?php
    $x = readline("Give me a number to square: ");
    echo(hard_square($x) . "\n");

    function hard_square($x)
    {
        $result = 0;
        for ($i = 0; $i < $x; $i++)
            $result += $x;
        return $result;
    }
?>
```

PHP Syntax

- **Functions**

```
function hard_square($x = 10)
{
    $result = 0;
    for ($i = 0; $i < $x; $i++)
        $result += $x;
    return $result;
}
```

PHP Syntax

```
<?php
    echo(hard_square() . “\n”);

    function hard_square($x = 10)
    {
        $result = 0;
        for ($i = 0; $i < $x; $i++)
            $result += $x;
        return $result;
    }
?>
```

PHP Syntax

```
<?php
    echo(hard_square() . “\n”);

    function hard_square($x = 10)
    {
        $result = 0;
        for ($i = 0; $i < $x; $i++)
            $result += $x;
        return $result;
    }
?>
```

PHP Syntax

- **Including files**

- Just like C programs can consist of multiple files to form a single program, so can PHP programs tie files together.

```
#include <cs50.h>
```

PHP Syntax

- **Including files**

- Just like C programs can consist of multiple files to form a single program, so can PHP programs tie files together.

~~`#include <cs50.h>`~~

PHP Syntax

- **Including files**

- Just like C programs can consist of multiple files to form a single program, so can PHP programs tie files together.

```
require_once(__DIR__ . “cs50.php”);
```

PHP Syntax

- **Including files**

- Just like C programs can consist of multiple files to form a single program, so can PHP programs tie files together.

```
require_once(__DIR__ . “cs50.php”);
```

PHP Syntax

- Though PHP is primarily used (and was initially designed) as a language for web-based programming, it is a full language and can do nearly all things C can, which means it can be run from the command line.

- All that is required is that the PHP interpreter is installed on the system you wish to run your PHP programs on.

PHP Syntax

- To run your PHP program through the PHP interpreter at the command-line, simply type

```
php <file>
```

- and your program will run through the interpreter, which will execute everything inside of the PHP delimiters, and simply print out the rest.

PHP Syntax

- Recall that you don’t have to have only one set of PHP delimiters in your program. You can have as many as you’d like, opening and closing them as needed.

PHP Syntax

- You can also make your programs look a lot more like C programs when they execute by adding a **shebang** to the top of your PHP files, which automatically finds and executes the interpreter for you.

```
#!/usr/bin/php
```

- If you do this, you need to change the **permissions** on your file as well using the linux command `chmod` as follows:

```
chmod a+x <file>
```