# Basics Of Programming

Using a language to translate between you and the computer

# Variables

Storing Data for Later

# Variable Basics

- Container for storing data
- Represented by an identifier prefixed with $
- Identifier can have letters, numbers and underscores
- Identifier cannot start with a number
- Value: contents of a variable

# Creating and Assigning

- Assignment: putting something in a variable
- Variables are created by assigning something to them
- Assigning a variable to another variable will copy the variable

**Example 1-1**

```php
<?php

$number = 1;

$othernumber = $number;

echo $number;
echo $othernumber;

echo $missing;
```

Output:

11
Notice: Undefined variable: missing in {filename} on line 7

# Data Types

- scalar types
  - null
  - boolean
  - integer
  - float
  - string
- non-scalar types
  - resource
  - object
  - array

# Type Juggling

- Every variable has a type, but PHP does its best to ignore them
- PHP tries to convert values to the type it thinks you want

# Variable Tools

- isset() – does it exist?
- is_null() – is the value null?
- empty() – does it have data somewhere?
- unset() – Deletes the variable
- var_dump() – Prints the variable’s contents (great for debugging)

# Basic Data Types

Integers and Booleans and strings, oh my!

# Null

- Null is weird
- Indicates absence of a value, but existence of a variable
- is_null()

Example 2-1

```php
<?php

echo null;

$var = null;

echo $var;
```

Output:

# Boolean

- Usually shortened to bool (programmers are lazy)
- Contains either true or false
- is_bool() (NOT is_boolean(): that doesn’t exist)

**Example 2-2**

```php
<?php

echo true;

$var = false;

echo $var;
```

Output:

10

# Integer

- Usually shortened to int
- A number with no fractional part (no decimals)
- is_int()/is_integer()

**Example 2-3**

```php
<?php

echo 1;

$var = 5;

echo $var;
```

Output:

15

# Float

- Like int, but with a decimal point
- is_float()

**Example 2-4**

```php
<?php

echo 3.11;

$var = 8.1;

echo $var;
```

Output:

3.118.1

# String

- A string of text
- is_string()
- is_numeric()

Example 2-5

```php
<?php

echo 'foo';

$var = "bar";

echo $var;
```

Output:

```
foobar
```

# Single vs. Double Quotes

- Single quotes ('foo') for regular strings
- Double quotes ("bar") to interpolate
- Escape quotes with backslash (e.g.. ' \' ')
- Interpolation: inserting variables into a string

**Example 2-6**

```php
<?php

echo ' \' ';

$var = 'bar';

echo 'foo$var ';

echo "foo$var ";
```

Output:

```
' foo$var foobar
```

# Operators

Add, subtract, multiply, divide, concatenate?

# Operator Basics

- Operators operate on one or more pieces of data
- Work with variables or hardcoded values
- Multiple operators can be used together
- Operations can be separated with parentheses
- Result of an operation can be assigned to a variable

**Example 3-1**

```php
<?php

echo 5 + 2;

$var = 4;
echo $var * 2;

$var = 3 / (8 - 2);
echo $var;
```

Output:

780.5

# Basic Math

- + - Addition
- - - Subtraction
- * - Multiplication
- / - Division

**Example 3-2**

```php
<?php

echo 4 + 2;

echo 4 - 2;

echo 4 * 2;

echo 4 / 2;
```

Output:

6282

# Less Basic Math

- % - Modulus (remainder from division)
- ** - Exponent (like using ^ on your calculator)
- % ignores decimals, use fmod() for floats
- ** only works on PHP 5.6+, use pow() for old versions

Example 3-3

```php
<?php

echo 7 % 3.5;

echo fmod(7, 3.5);

echo 2 ** 3;

echo pow(2, 3);
```

Output:

1088

# Increment and Decrement

- Shortcuts for adding or subtracting one from a variable
- Increment (++) adds one to the variable
- Decrement (--) subtracts one from the variable
- Can be before the variable (++$var) or after ($var++)
- Does weird things when combined with other operators

**Example 3-4**

```php
<?php

$var = 5;

$var++;
echo $var;

--$var;
echo $var;
```

Output:

65

# Concatenation

- Links two strings together
- Operator is a dot (.)

**Example 3-5**

```php
<?php

echo 'foo' . 'bar';

$var = 'ghjkl';

echo 'asdf' . $var;
```

Output:

```
foobarasdfghjkl
```

# Assignment Operators

- Perform an operation and store the result with one operator
- Works with most operators
- +=, *=, .=, etc.
- For example, '$var += 3' does the same thing as '$var = $var + 3'

**Example 3-6**

```php
<?php

$var = 5;

$var += 3;
echo $var;

$var .= "4";
echo $var;
```

Output:

884

# Comparison operators

- Used to compare two values
- Return a Boolean value rather that a number
- Equal (==) and not equal (!= or <>)
- Greater than (>) and less than (<)
- Greater than or equal to (>=) and less than or equal to (<=)
- Identical (===) and not identical (!==)

Example 3-7

```php
<?php

echo 5 == 5;

echo 2 != 3;

echo 6 > 8;

echo 7 <= 7;
```

Output:

111

# Logical operators

- Used on Boolean values
- Not (!) flips the value
- And (&&) is true if both values are true
- Or (||) is true if either value is true
- Can be combined, just like other operators

**Example 3-8**

```php
<?php

echo !false;

echo false && true;

$var = true || false;

echo !$var;
```

Output:

1

# Conditionals

Deciding what to do

# If

- Executes a section of code if a condition is true
- Condition is contained in parentheses ()
- Code to execute is contained in braces {}

Example 4-1

```php
<?php

if (true) {
    echo 'foo';
}
if (true || false) {
    echo 'bar';
}
if ( !(6 <= 3) ) {
    echo 'foobar';
}
```

Output:

foobar

# Else

- Placed after an if statement
- Executes a block of code if the if statement was false

## Example 4-2

```php
<?php

if (true) {
    echo 'true';
} else {
    echo 'false';
}
if (false) {
    echo 'true';
} else {
    echo 'false';
}
```

Output:

truefalse

# elseif

- Placed after an if or another elseif statement
- Works just like if, but is only checked if the previous statement was false

## Example 4-3

```php
<?php

$var = 1;

if ($var == 0) {
    echo 'var is 0';
} elseif ($var == 1) {
    echo 'var is 1';
} elseif ($var == 2) {
    echo 'var is 2';
}
```

Output:

var is 1

# Switch

- Executes one of several sections of code depending on the value of a variable
- Can be used in place of long chains of elseifs
- Each section starts with “case value:” and ends with “break;”
- Works with integers, floats, and strings

Example 4-4

```php
<?php

switch (2) {
    case 1:
        echo “one";
        break;
    case 2:
        echo “two";
        break;
}
```

Output:

two

# Loops

Do it again (and again and again)

# While

- Repeats a block of code as long as a conditional is true
- Similar syntax to if

**Example 5-1**

```php
<?php

$var = 1;

while ($var < 5) {
    echo $var;
    $var++;
}
```

Output:

1234

# Do…While

- Similar to while, but checks the conditional at the end of the loop
- Used to make sure the loop runs at least once

**Example 5-2**

```php
<?php

$var = 5;

do {
    echo $var;
    $var++;
} while ($var < 3)
```

Output:

5

# For

- Executes an instruction before the loop starts
- Loops while a condition is true, executing an instruction at the end of each loop

**Example 5-3**

```php
<?php

for ($i = 1; $i < 10; $i++) {

    echo $i;

}
```

Output:

123456789

# Functions

Reusable chunks of code

# Function Basics

- Chunk of code that can be used in other places
- Data is given to the function via arguments
- Functions are 'passed' a copy of each argument
- Functions may return a value
- Can be used like a variable or on their own

**Example 6-1**

```php
<?php

echo pow(2,4);

$var = pow(1,5);

echo pow($var, 2);

echo pow(3, pow(4,2));
```

Output:

16143046721

# Defining a Function

- Naming rules are the same as variables
- Defined with the function keyword
- Can be declared and called anywhere… sort of

Example 6-2

```php
<?php

foobar();

function foobar() {
    echo 'foobar';
}

foobar();
```

Output:

```
foobarfoobar
```

# Defining a Function with Arguments

- Arguments are stored to variables
- Variables are listed in the parentheses in the definition
- Separate arguments with commas

Example 6-3

```php
<?php

function addandecho($num1, $num2) {

    echo $num1 + $num2

}

addandecho(2, 3);
```

Output:

5

# Returning Values

- A value can be returned with the return keyword
- Return values can be used like variables
- Any data type can be returned
- Function stops at return, nothing after it gets run

Example 6-4

```php
<?php

function addandreturn($num1, $num2) {
    return $num1 + $num2;
    echo 'I never get run!';
}

echo addandreturn(2, 3);
```

Output:

5

# The Standard Library

- All the functions built in to PHP are part of the standard library
- Information about all the functions in the standard library can be found at php.net/manual

# Files

Using permanent storage

# Include/Require

- Include allows you to use more than one file for your project
- Including a file allows you to use all the functions and variables defined in that file
- Include will just print a warning if something goes wrong, require will stop everything

# file_put_contents

- Writes a string to a file
- Takes two arguments: the path of the file to write and the data to write

# file_get_contents

- Used to get the contents of a file as a string
- Requires one argument, a string containing the path to the file

Example 8-1

```php
<?php

file_put_contents('foobar.txt', 'foobar');

echo file_get_contents('foobar.txt');
```

Output:

```
foobar
```

# file_exists

- Used to check if a file exists
- Takes one argument, a string containing the file path
- Returns a Boolean: true if the file exists, false if it doesn't

# unlink

- Deletes a file
- One argument, the path of the file as a string
- Returns true if the file was deleted, false otherwise

Example 8-2

```php
<?php

unlink('exists_test');

echo file_exists('exists_test');

file_put_contents('exists_test', 'foobar');

echo file_exists('exists_test');
```

Output:

1

# Arrays

Lists of data

# Array Basics

- An array stores multiple values of any type, even other arrays
- Each value has a unique key which is used to access it
- Each key/value pair is called an element
- Arrays can be used as a whole or one element at a time
- Array elements are accessed using square brackets [ ]
- Accessing a nonexistent array element will give a notice

**Example 9-1**

```php
<?php

$array = array();

$array[1] = 'foobar';
echo $array[1];

$array[2] = 4;
echo $array[2] + 3;
```

Output:

```
foobar7
```

# Creating Arrays

- Arrays can be created using square brackets or array()
- Keys are separated from values using the => operator
- Elements are separated by a comma
- Keys can be ints or strings, other types will be juggled if possible

## Example 9-2

```php
<?php

$array = [
    1.2 => 'foo',
    '3' => 'bar',
    'abc' => 123,
    false => 'asdf',
];

var_dump( $array );
```

Output:

```
array(4) {
  [1]=>
  string(3) "foo"
  [3]=>
  string(3) "bar"
  ["abc"]=>
  int(123)
  [0]=>
  string(4) "asdf"
}
```

# Numeric Indexes

- Values without keys will be assigned an integer key
- New array elements can be added with empty square brackets
- Individual array elements can be removed with unset()
- New keys count from the highest key that has been used in the array

## Example 9-3

```php
<?php

$array = [
    'foo',
    'bar',
];

var_dump( $array );
unset( $array[1] );
$array[] = 'asdf';
var_dump( $array );
```

Output:

```
array(2) {
  [0]=>
  string(3) "foo"
  [1]=>
  string(3) "bar"
}
array(2) {
  [0]=>
  string(3) "foo"
  [2]=>
  string(4) "asdf"
}
```

# foreach

- Special loop that goes through all the elements of an array
- Places the current value in a variable for easy access inside the loop
- Optionally places the current key in a different variable
- Adding or removing elements inside the loop will break things
- Key and value variables must be unset after the loop

Example 9-4

```php
<?php

$array = [
    'foo',
    'bar',
];
foreach($array as $key => $value) {
    echo $key . $value;
}
```

Output:

0foo1bar

# Object Oriented Programming

Using code to represent objects

# So, what is an object?

- Objects are collections of properties and methods
- Properties describe the object
- Methods make the object do stuff
- Objects are based on classes
- Classes describe the properties and methods

# Objects

Collections of data

# Object Basics

- Special non-scalar data type
- Stored in a variable… sort of

# Creating an object

- Objects are created, or instantiated, using a constructor
- Constructors are called using the 'new' keyword
- Constructor works like a function
- Returns a new object based on the class

**Example 10-1**

```php
<?php

$object1 = new stdClass();
$object2 = new stdClass();

var_dump( $object1 );
var_dump( $object2 );
var_dump( new stdClass() )
```

Output:

```
object(stdClass)#1 (0) {
}
object(stdClass)#2 (0) {
}
object(stdClass)#3 (0) {
}
```

# Cloning objects

- Objects can be copied using the 'clone' keyword
- Cloned objects can be used just like the original

Example 10-2

```php
<?php

$object = new stdClass();
$objectcopy = $object;
$objectclone = clone $object;

var_dump( $object );
var_dump( $objectcopy );
var_dump( $objectclone );
```

Output:

```
object(stdClass)#1 (0) {
}
object(stdClass)#1 (0) {
}
object(stdClass)#2 (0) {
}
```

# Deleting objects

- Objects are automatically destroyed when no longer in use
- Unset or change things object is 'stored' in

Example 10-3

```php
<?php

$object = new stdClass();

$objectcopy = $object;

unset($object);

$objectcopy = null;
```

Output:

# Classes

Making your own objects

# Class Basics

- Classes list properties and define methods
- Naming rules are the same as variables, but no $

# Defining a class

- Classes are defined using the 'class' keyword
- Definition goes inside braces { }
- Classes should be defined before they are used

Example 11-1

```php
<?php

class ExampleClass {

}

$object = new ExampleClass();

var_dump( $object );
```

Output:

```
object(ExampleClass)#1 (0)
{
}
```

# Creating a constructor

- PHP generates an empty constructor by default
- Custom constructors can be made using 'function __construct'
- The constructor definition goes inside the class
- Constructor works like a function
- Constructors can have arguments but no return value

Example 11-2

```php
<?php

class ExampleClass {
    function __construct() {
        echo 'Object created';
    }
}

$object = new ExampleClass();
```

Output:

Object created

# Creating a destructor

- Called when PHP destroys an object
- PHP generates an empty destructor by default
- Custom constructors can be made using 'function __destruct'
- Like a constructor, but can't have arguments

Example 11-3

```php
<?php

class ExampleClass {
    function __destruct() {
        echo 'Object destroyed';
    }
}

$object = new ExampleClass();
unset( $object );
```

Output:

Object destroyed

# Properties

Storing data inside objects

# Property Basics

- Variable inside an object
- Can be any data type, even a pointer to another object
- Work just like variables
- Can have a default value
- Can be public, private, or protected

# Creating properties

- Properties are defined inside the class
- Property definition consists of a visibility, a name, and optionally a default value.
- Properties without default values are null by default

Example 12-1

```php
<?php

class ExampleClass {

    public $property1;

    private $property2 = 'foobar';

    protected $property3;

}
```

Output:

# Accessing properties

- Accessed using the '->' operator
- $ is not used when accessing
- Work like variables

Example 12-2

```php
<?php

class exampleClass {
    public $property = 'foo';
}

$object = new exampleClass();
echo $object->property;
$object->property = 'bar';
echo $object->property;
```

Output:

foobar

# Methods

Making objects do things

# Method Basics

- Function inside an object
- Work just like functions
- Can take arguments and have returns
- Can be public, private, or protected

# Creating methods

- Methods are defined inside the class
- Defined like a function, but with a visibility

Example 13-1

```php
<?php

class ExampleClass {

    public function method1() {
    }

    private function method2() {
    }

}
```

Output:

# Calling methods

- Called using the -> operator
- Used just like a function

Example 13-2

```php
<?php

class exampleClass {
    public function foobar() {
        echo 'foobar';
    }
}

$object = new exampleClass();
$object->foobar();
```

Output:

foobar