INTRO TO PHP

DR. ANDREW BESMER

- About PHP
- Basic Syntax
- Variables
- Type Juggling
- Superglobals

ABOUT PHP

THE LANGUAGE

PHP: PHP Hypertext Preprocessor

- Recursive acronym
- Open source
- Available on most OS
- Portable code

SIMILARITIES

- Code similar to
  - C
  - Java
  - Perl
- Take some concepts with you or reuse existing
- Used to write dynamic server side web apps
  - Can be used to make cli scripts

RUNNING PHP

- You can run PHP several ways:
  - Through a web server like Apache

```
http://localhost/myFile.php
```

  - Directly from the command line

```
php myFile.php
```

INTERPRETED LANGUAGE

PHP is an interpreted language

- Pros
  - Program can change dynamically at runtime
  - Easier to program with, just code and run!
  - Easier to move from one OS/Architecture to another
- Cons
  - Without compiling syntax errors may go undetected
  - Can be problematic for performance
    - Can be mitigated via caching

BASIC SYNTAX

PHP MODE

- Can move in and out of “PHP mode” as you need
  - Move into using the start tag `<?php`
  - Can exit by using the end tag `?>`

```
<?php
//PHP mode
?>
```

- Execution depends on code location
  - Inside is code to be run on the server
  - Outside is ignored by interpreter and outputted[1]
- It is good practice to leave off the closing tag in a PHP only file

PHP MODE

- PHP is most frequently combined with HTML/CSS/JS for making web apps

```
<!DOCTYPE html>
<html>
    <head>
        <title>First App</title>
    </head>
    <body>
        <?php echo "Hi CSCI241!"; ?>
    </body>
</html>
```

- Can be used to do many other things
  - Generate PNG/JPG
  - Generate JSON/XML
  - Generate PDF

STATEMENTS

Statement: Instruction given within the language.

- When in PHP mode the interpreter will evaluate and execute each statement then move to the next
- Each statement **must** end with a semicolon `;`[2]

COMMENTS

- Single line comments use `//`

```php
<?php
// This is a single line comment
echo "This line is run!";
```

- Multi line comments use `/*` to start them and `*/` to end them

```php
<?php
/*
 This is a multi line comment
 echo "These statements were commented out.";
 echo "So they will not be executed or output";
*/
```

- A `#` can also be used but seldom is

```php
<?php
# I'm not used often
```

COMMENTS

WARNING

- Do not nest the multi line comments they end at the first `*/`

```php
<?php
/*
    echo 'This will not end well!'; /* Do you see the problem? */
*/
```

VARIABLES

VARIABLES

Variable: Named reference to a storage location in main memory (RAM) who's value can change.

- Which would you rather?

```
0x00BAB10C
```

```
$uberBlock
```

VARIABLES

- Variable names should start with a `$`
- Variable names
  - Are case sensitive
  - Must start with letter or _ followed by letters, numbers, or underscores
  - Variable names should be `$camelCasedForReadability`
  - `$this` is a reserved variable (more later)

VARIABLES

- PHP variables are loosely or weakly typed
  - It is not necessary to specify the type prior to using or initializing the variable
  - The type is determined by the language based on the context of use
  - Results can be predictably unexpected
- Variables are always assigned by value (more later)

PRIMITIVES

Primitive Data Type: A built in data type provided by the programming language being used.[3]

- PHP has a total of eight primitives
  - 4 scalar
  - 2 compound
  - 2 special
- In documentation for PHP you may see some “pseudo-types”, these are used for readability of documentation
  - `mixed`
  - `number`
  - `callback`

SCALARS

Scalar: A variable limited to a single value at a time.

- Differentiated from complex data types like `array` or `object`

BOOLEAN

`boolean`: A boolean value is either *`true`* or *`false`*
alternatively $B = \{0, 1\}$

- The following values are considered to be `false`[4]
  - `false`
  - `0`
  - `0.0`
  - `""` or `"0"`
  - `array()`
  - `NULL`
- All others are considered `true` including `-1`

INTEGER

integer: $\mathbb{Z} = \{\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots\}$

- Can be specified in binary, octal, decimal, or hex with a `+` or `-` indicating sign
  - A `0b` indicates binary
  - A `0` indicates octal
  - A `0x` indicates hex

```php
<?php
$bin = 0b10100011; //163 in decimal
$oct = 0123; //83 in decimal
$dec = 123; //123 in decimal
$hex = 0x64; //100 in decimal

var_dump($dec == $oct); //is bool(false)
```

INTEGER

- The maximum `integer` size can be determined using the constant `PHP_INT_SIZE`
- `PHP_INT_MAX` can be used for the maximum value
  - Returns the number of bytes allocated for the variable
  - **Usually** 8 bytes on 64 bit machines and 4 on 32 bit machines
  - No “unsigned” integers in PHP
    - Signing uses the first bit to indicate positive or negative
    - What would happen if PHP did support them?

INTEGER

- Invalid octal specification results in stopping at bad digit

```php
<?php
var_dump(011901); //Decimal 9!
```

- Variables do ~~not~~ need to be initialized
  - Always initialize your variables even though you dont “have to”

INTEGER OVERFLOW

- Unlike other languages `integer` overflow results in using a `float` using E notation
- Typed languages generally roll over because of 2's complement

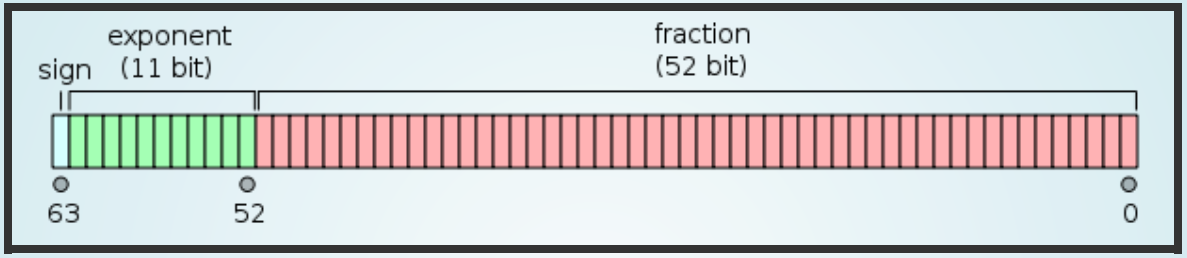
FLOAT

`float`: $\mathbb{R} = \{x | x \text{ is a real number}\}$

- Can be specified by providing E notation or number with decimal place
- Floats like integers depend on the platform, a max of ~ $1.8e308$ with ~14 digit precision per IEEE 64bit standard

FLOAT

- Conversion to binary results in loss of precision for some numbers



IEEE 754 Floating Point Format

This can lead to confusing results: for example, `floor((0.1+0.7)*10)` will usually return `7` instead of the expected `8`, since the internal representation will be something like

`7.9999999999999991118...`[5]

FLOAT

- Comparing floats for inequality can also be problematic
  - Can specify precision of equality[6]

```php
<?php
$a = 1.23456789;
$b = 1.23456780;
$epsilon = 0.00001;

if(abs($a-$b) < $epsilon) {
    echo "true";
}
```

NAN

NaN: Not a number, can be any number of values and should not be used for any purpose other than observing that the result of the operation was undefined or unrepresentable.

- Can check for `NaN` using `is_nan()`

STRING

`string`: A series of characters.

- Native strings only support ASCII (no native Unicode)

```php
<?php
$name = "Andrew Besmer"; //My name uses 13B
$name = "&rew Besmer"; //My 1337 name uses 11B
```

- The max string length is 2GB in PHP

STRING

- You can specify a string by using
  - Single quoted syntax
  - Double quoted syntax
  - heredoc syntax
  - nowdoc syntax
- Depending on the method used variables may be inserted into the string!

STRING

- Single quoted uses the `'` character to start and end the string

```php
<?php
$name = 'Andrew Besmer';
```

- Variables are not inserted into the string with single quotes

```php
<?php
$greeting = 'Hello';
$name = '$greeting Andrew Besmer';
echo $name; //Outputs: $greeting Andrew Besmer
```

STRING

- What if I wanted to set the name `Pat O'Neal`?
  - `\` can be used as an escape character
  - `\'` makes `'` and `\\` makes `\`

```php
<?php
$name = 'Pat O\'Neal';
echo $name; //Outputs: Pat O'Neal
```

STRING

- A double quote `"` can also be used to specify start and end of strings
- Variables are inserted into string

```php
<?php
$greeting = "Hello";
$name = "$greeting Andrew Besmer";
echo $name; //Outputs: Hello Andrew Besmer
```

STRING

- Other common escape characters

Sequence	Meaning
\r	CR
\n	LF
\t	HT
\v	VT
\\	backslash
\$	dollar sign
\"	double-quote

- Remember that a CRLF in HTML does nothing!

STRING

- You can also use heredoc syntax `<<<IDENTIFIER` which will accept a string until seeing the `IDENTIFIER;`
- Since a double quote is not used to start and stop the string definition it is not necessary to escape them
- Can optionally be specified using `<<<"IDENTIFIER"` more explicitly explaining what will happen in the string

```php
<?php
$greeting = "Hello!";
$longText = <<<EOF
$greeting
This is some longer text.
All of this will wind up in the string.
If can go on for many lines.
EOF;
```

STRING

- Nowdoc is similar to heredoc but is specified using `<<<'IDENTIFIER'` instead
- The `'` explicitly describes expected functionality

```php
<?php
$greeting = "Hello!";
$longText = <<<'EOF'
$greeting
This is some longer text.
All of this will wind up in the string.
If can go on for many lines.
EOF;
```

STRING

- A strings character can be accessed using array syntax `$name[0]`
- Note that arrays are 0 based
- More on arrays later
- Strings serve as PHP's byte
- An empty string is `NULL`

```php
<?php
$emptyString = ''; //NULL
```

STRING

WARNING

- Strings are concatenated using the `.` operator NOT the `+` operator which many other languages use

COMPOUND

- PHP has two compound data types
  - `array`
  - `object`
- We will learn more about both of these later

SPECIAL

- PHP has two special data types
  - `resource`
  - `NULL`

RESOURCE

`resource`: A variable to hold references to external resources, e.g. a opened files, database connections, etc…

NULL

`NULL`: Represents a variable with no value.

- A variable is `NULL` if
  - You explicitly assign `NULL`
  - If you have not set any value yet
  - The variable has been `unset()`
- `NULL` is not case senesitive
- Can check for it using `is_null()`

TYPE JUGGLING

TYPE JUGGLING

- PHP will auto convert the type depending on the context
- PHP does not change the variable itself but it's use in the expression and the resulting data type

```php
<?php
$test = "100"; //A string
$test = $test + 10; //An integer
$test = $test + 10.5; //A float
$test = $test + "15 hundred"; //A float 135.5
$test = 100 + "15 hundred"; //An integer 115
```

CASTING

- It may be beneficial for you to explicity set the datatype
- This can be accomplished by using casting
- Cast by putting the data type you desire in parathensis
  - `(int)`, `(integer)`
  - `(bool)`, `(boolean)`
  - `(float)`, `(double)`, `(real)`
  - `(string)`
  - `(array)`
  - `(object)`
  - `(unset)`
  - `(binary)`

CASTING

- For `boolean` see earlier for how values are determined to be `true` or `false`
- Examples[7]

```php
<?php
var_dump((bool) "");        // bool(false)
var_dump((bool) 1);         // bool(true)
var_dump((bool) -2);        // bool(true)
var_dump((bool) "foo");     // bool(true)
var_dump((bool) 2.3e5);     // bool(true)
var_dump((bool) array(12)); // bool(true)
var_dump((bool) array());   // bool(false)
var_dump((bool) "false");   // bool(true)
```

CASTING

- When converting from `boolean` to `integer`
  - `false` becomes `0`
  - `true` becomes `1`
- From `float` to `integer`
  - The number is rounded down **towards zero**
  - Numbers to large are `undefined` or `NaN` no errors are thrown
  - Warning!

```php
<?php
echo (int) ( (0.1+0.7) * 10 ); //Is 7 not 8!
```

CASTING

- From `object` to `float` results in a notice
- From `integer` to `float` can result in loss of precision

CASTING

- From `boolean` to `string`
  - `false` is `""`
  - `true` is `"1"`
- From `float` or `integer` to `string`
  - Textual representation to string form with E notation
- From `object` to `string`
  - Becomes `"Object"`
- From `array` to `string`
  - Becomes `Array`
- From `NULL` to `string`
  - Becomes `""`

CASTING

- From `string` to `number`
  - If it does not have `.`, `e`, or `E` in it becomes `integer` otherwise `float`
  - Valid part of `string` used then rest discarded
  - Nothing valid means `0`

String conversion to numbers from php 8

```php
<?php
$foo = 1 + "10.5";                // $foo is float (11.5)
$foo = 1 + "-1.3e3";              // $foo is float (-1299)
$foo = 1 + "bob-1.3e3";           // $foo is integer (1)
$foo = 1 + "bob3";                // $foo is integer (1)
$foo = 1 + "10 Small Pigs";       // $foo is integer (11)
$foo = 4 + "10.2 Little Piggies"; // $foo is float (14.2)
$foo = "10.0 pigs " + 1;          // $foo is float (11)
$foo = "10.0 pigs " + 1.0;        // $foo is float (11)
```

SUPERGLOBALS

$_GET/$_POST

- The superglobals `$_GET` and `$_POST` contain the name value pairs sent as part of a GET/POST request from your form[9]

```php
<?php
$firstName = $_POST["firstName"];
$lastName = $_POST["lastName"];
?>
<html>
    <head>
        <title>Hello <?php echo $firstName . $lastName . "!"; ?></title>
    </head>
    <body>


        <form method="POST" action="<?php echo $PHP_SELF; ?>">
            First Name: <input type="text" name="firstName"><br />
            Last Name: <input type="text" name="lastName">
            <input type="submit" value="submit" name="submit">
        </form>
    </body>
</html>
```

1. Exception is using conditionals for output, more on this later.↩
2. Exception is when closing the PHP tag. Don't do this!↩
3. Exceptions are sometimes made for complex data types.↩
4. SimpleXML objects from empty tags are also false.↩
5. http://www.php.net↩
6. http://www.php.net↩
7. http://www.php.net↩
8. http://www.php.net↩
9. XSS has been left in example for simplicity.↩