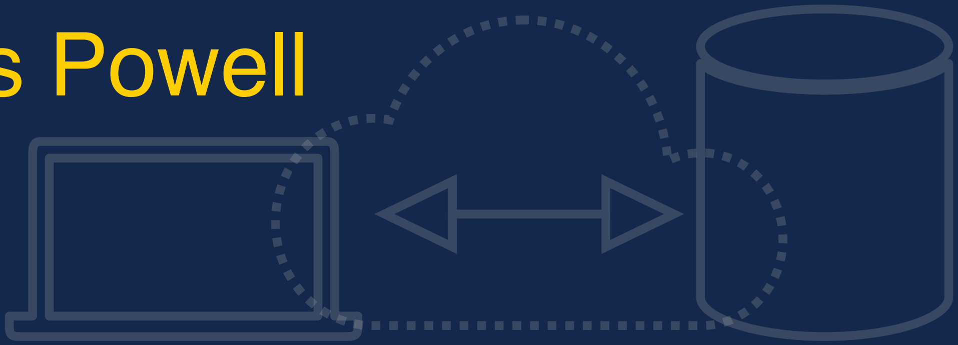
# Server-Side Scripting Environments

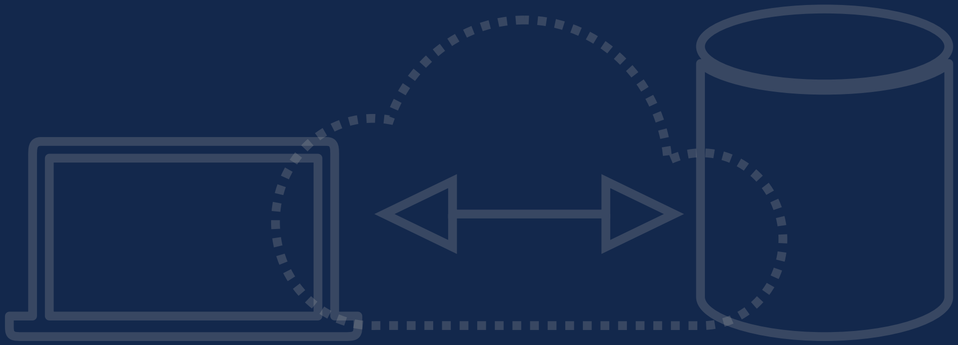
with Thomas Powell



# Server Side Scripting Intro

- Server-side scripting offers a balance between complexity and performance
  - Not as hard as server-modules (or maybe CGI), but not as performance oriented as server-modules

- The general idea is to add scripting directives with template files (often thought of as modified HTML files)
  - The scripted pages are intercepted by a Web server on the way out and the code converted into the appropriate output – generally HTML

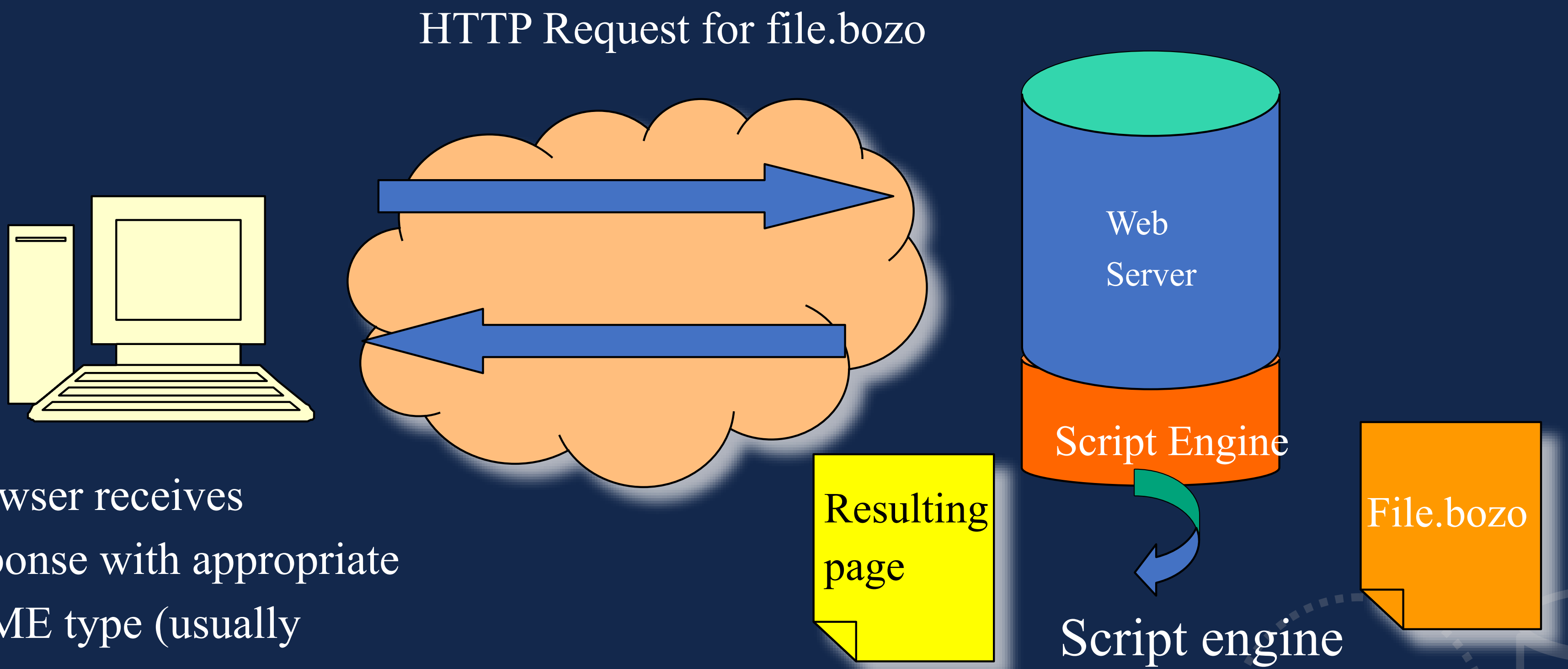


# Implementing a Server-Side Scripting Language

- Consider the idea of writing a BozoScript server module that looks to see if a file is being requested with an extension .bozo

- The server module either intercepts the page, parses and produces or a result or passes it on without parsing
    - The trigger for interception is generally a particular file extension (ex. php), but may be a path also
    - You could intercept everything particularly .html which might hide implementation but a performance cost would be incurred

# How Server-Side Scripting Works Contd.



HTTP Request for file.bozo

Web Server

Script Engine

Resulting page

File.bozo

Browser receives response with appropriate MIME type (usually text/html) and renders

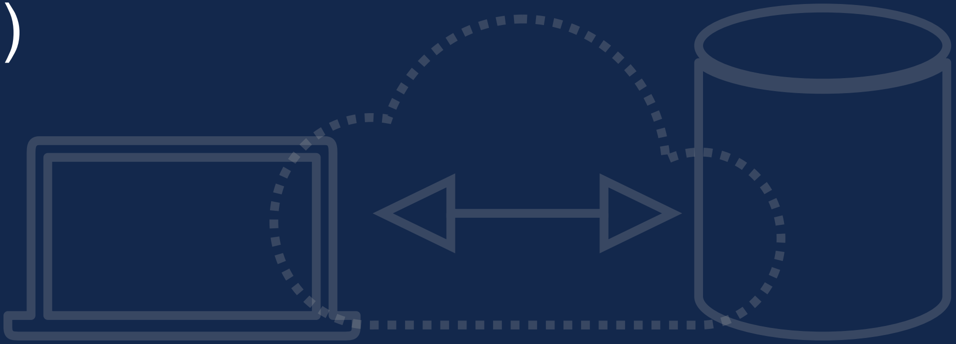
Script engine intercepts and evaluates “BozoScript” code

# Server-Side Scripting Language Characteristics

- When looking at server-side scripting languages you see some common characteristics
  - Syntax variations
    - Tag like or script like
  - Ease of use and focus on troubles of Web programming over complexity and scale concerns
    - Session management, form handling, database programming
  - Aims to assist in code-UI separation
    - Through syntax, separation of presentation and logic, etc.

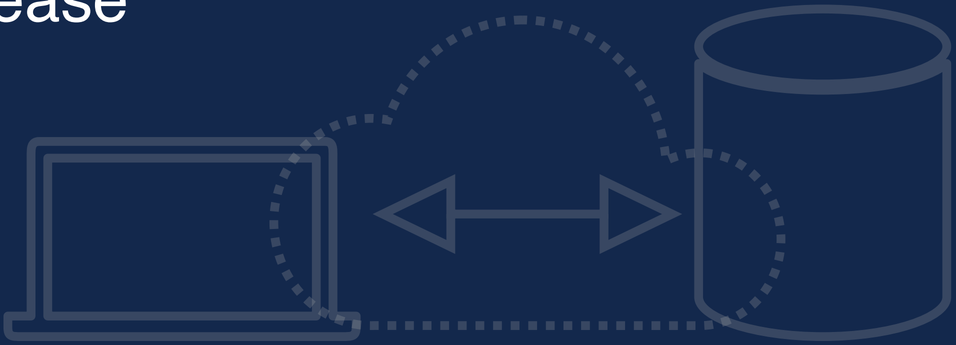
# Server-Side Scripting Language Characteristics

- Generations of Server-side scripting?
  - Zero Generation: SSI (server-side includes)
  - 1st Generation: Classic ASP, ColdFusion, PHP
  - 2nd Generation: JSP, ASP.NET
  - 3rd Generation: Declarative style- Flex –OR- 4GL like - Ruby on Rails
  - 4th Generation: Shared Client / Server Side which implies all JavaScript based for now (EJS)



# Server-Side Scripting Language Characteristics

- Yet for all their supposed differences server-side scripted environments suffer from a tension of ease and performance.
  - First generation suffered with performance issues
  - 2nd generation efforts addressed large system design and performance at the cost of ease
  - A thrust of 3rd generation was more ease
  - 4th generation has been about single language approaches and performance more so than ease



# Parsed Script Intro

- We review a few of the basic issues with the first generation server-side scripting languages here before studying PHP more in-depth.

- Server Side Includes (SSI)
  - .shtml files
  - In the form of a comment
  - `<!--#include file=“footer.html”-->`
  - SSI is common to nearly every Web server, though not always enabled.  Some Web servers may have unique features
- Already even with SSI you see the main issues
  - How do you indicate the file has some script in it (file extension)
  - Within the file how do you delimit or separate template from script (in this case HTML comment)
  - Performance hit vs. the productivity gain
- Some editors provided this same type of template style without the use of server-side page composition

# ColdFusion Markup Overview

- ColdFusion initially invented by Allaire currently owned by Adobe
  - Simple server-parsed tag based scripting language that is very focused on database access and web specific duties
  - ColdFusion requires a special Application Server
  - Like other server parsed scripting solutions the app server is engaged when a file of a particular extension (.cfm) is requested

- **Cold Fusion markup looks like basic HTML tags** - Markup focus
  - Upside: Like React and Web Components
  - Downside: awkwardness of tags

- Example of Legacy Web Tech - still used in many places though far from popular

# ColdFusion Markup Overview

- You need to specify basic SQL (Structured Query Language) statements to use CF
- Given a DB table called *Positions* with the fields *Position-Num, JobTitle, Location, Description*, *Hiring Manager*, and *PostDate.* It is easy to query the table using statements like

```
SELECT * FROM Positions

SELECT * FROM Positions WHERE Location="Austin"

SELECT *
    FROM Positions
    WHERE ((Location="Austin" OR
         (Location="Los Angeles") AND
         (Position="Game Tester"))
```

# <CFQUERY>

```
<CFQUERY NAME="ListJobs"
      DATASOURCE="CompanyDataBase">
           SELECT * FROM Positions
</CFQUERY>
```

*Note: The DATASOURCE attribute is set equal to CompanyDataBase, which is the ODBC data source that contains a database called Company, which contains the Positions table from which data is pulled.*

**<CFQUERY>** supports attributes like **NAME**, **DATASOURCE**, **MAXROWS**, **USERNAME**, **PASSWORD**, **TIMEOUT**, and **DEBUG**

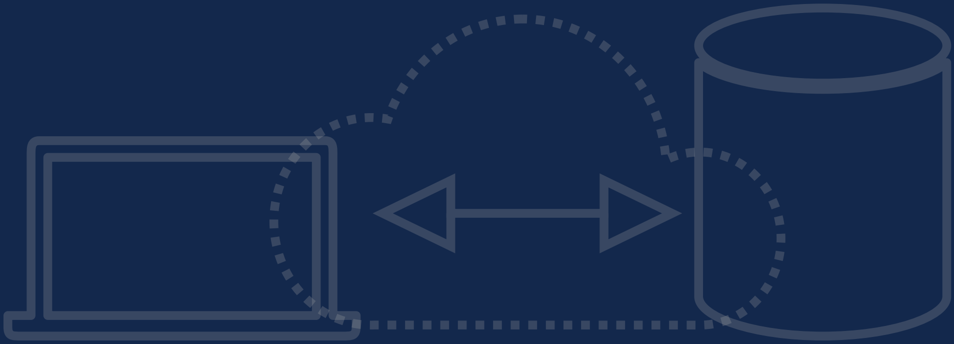
# <CFOUTPUT>

- Once you have data for example from variables, calculations or a from a database query you can output it with **<CFOUTPUT>**

```
<CFOUTPUT QUERY="ListJobs">
   <hr noshade><br>
   Position Number: <b>#PositionNum#</b><br><br>
   Title: #JobTitle#<br><br>
   Location: #Location#<br><br>
   Description: #Description#
 </CFOUTPUT>
```

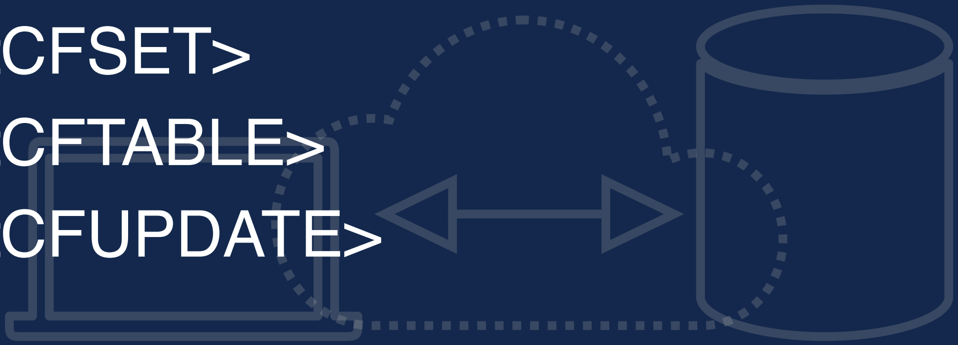
- Notice the use of # symbols to relate the fields of the database with output “slots” Also notice how HTML and database fields are intermixed.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<CFQUERY NAME="ListJobs" DATASOURCE="CompanyDataBase">
SELECT * from Positions
</CFQUERY>
<html>
<head><title>Demo Company Job Listings</title></head>
<body bgcolor="#FFFFFF">
<h2 align="center">Job Listings</h2>
<hr>
<CFOUTPUT QUERY="ListJobs">
<hr noshade><br>
Position Number: #PositionNum#<br><br>
Title: #JobTitle#<br><br>
Location: #Location#<br><br>
Description: #Description#
</CFOUTPUT>
<hr>
<address>
Demo Company, Inc.
</address>
</body>
</html>
```



# Common CFML Elements

- <CFABORT>
- <CFAPPLICATION>
- <CFCOL>
- <CFCONTENT>
- <CFCOOKIE>
- <CFERROR>
- <CFFILE>
- <CFHEADER>
- <CFIF>
- <CFINCLUDE>
- <CFINSERT>
- <CFLOCATION>
- <CFLOOP>
- <CFMAIL>
- <CFOUTPUT>
- <CFPARAM>
- <CFQUERY>
- <CFREPORT>
- <CFSET>
- <CFTABLE>
- <CFUPDATE>



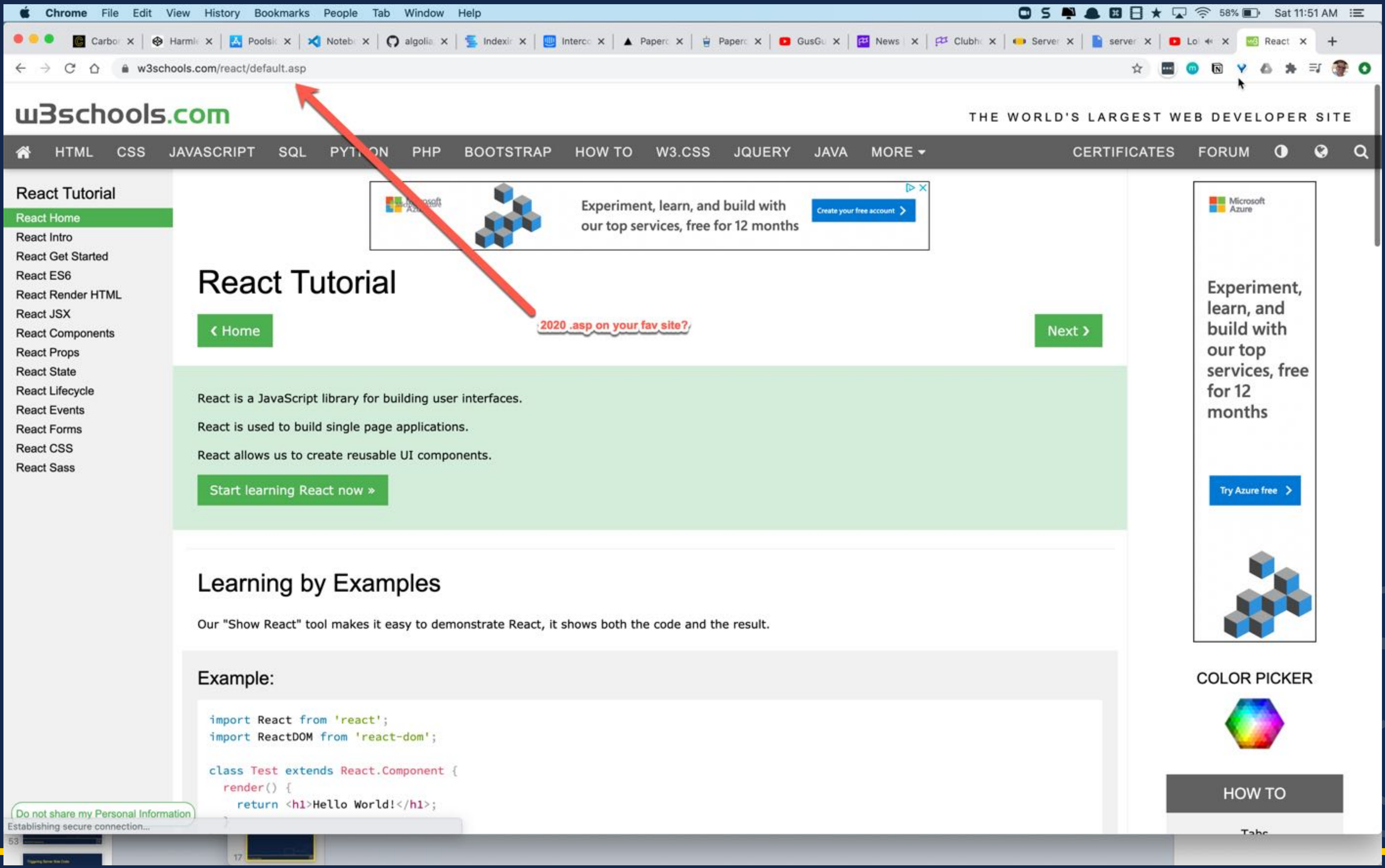
# CFM Summary

- Very high level and focused on building DB driven sites
- Tag like so integrates well with HTML
  - Resurgence of component systems including React suggests the Tag approach isn't necessarily a bad idea
- While some devs dislike tag systems we have seen many systems try to offer both approaches (ex. JSP, ASP.NET, etc.)

# ASP Intro

- Microsoft’s Active Server Pages (ASP) was (and still is to some degree) a very popular server parsed scripting framework
  - Bundled with Microsoft Web servers at the time - **Note this distribution advantage which is a common way systems win**
  - Indicated by files ending in .asp
  - Does not specify language used is a framework
    - Could be used with VBScript or Jscript (MS JavaScript clone)
  - **Syntax is more code like** and slightly less friendly than PHP and ColdFusion
  - With the rise of ASP.NET many classic ASP folks unable to make the complexity leap moved over to PHP or even Ruby

# Like ColdFuion - ASP Lives On?


Chrome File Edit View History Bookmarks People Tab Window Help
58% Sat 11:51 AM
Carbo Harm Pools Noteb algolia Indexi Interc Paper Paper GusGu News Clubh Server server Lo React
w3schools.com/react/default.asp
w3schools.com
THE WORLD'S LARGEST WEB DEVELOPER SITE
HTML CSS JAVASCRIPT SQL PYTHON PHP BOOTSTRAP HOW TO W3.CSS JQUERY JAVA MORE
CERTIFICATES FORUM
Microsoft Azure
Create your free account
Try Azure free
COLOR PICKER
HOW TO
Tabs
Do not share my Personal Information
Establishing secure connection...

2020 .asp on your fav site?

React Tutorial

- React Home
- React Intro
- React Get Started
- React ES6
- React Render HTML
- React JSX
- React Components
- React Props
- React State
- React Lifecycle
- React Events
- React Forms
- React CSS
- React Sass

Experiment, learn, and build with our top services, free for 12 months

## React Tutorial

❮ Home Next ❯

React is a JavaScript library for building user interfaces.

React is used to build single page applications.

React allows us to create reusable UI components.

Start learning React now »

## Learning by Examples

Our "Show React" tool makes it easy to demonstrate React, it shows both the code and the result.

### Example:

```
import React from 'react';
import ReactDOM from 'react-dom';

class Test extends React.Component {
  render() {
    return <h1>Hello World!</h1>;
```

# ASP Example

*Note <script> tag showing location of execution and language and script is delimited by <% %>*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
  <script language="VBScript" runat="Server"></script>
  <html>
  <head>
  <title>ASP Example</title>
  </head>
  <body>
  <h1>Breaking News</h1>

  <% = date() %>

<p>Today the stock of a major software company <br>
  reached an all time high, making the Demo Company CEO<br>
  the world's first and only trillionaire.</p>
  </body>
  </html>
```

# ASP Summary

- ASP syntax can get more familiar to programmers, for example consider a similar database example

```
<%
     Set Conn = Server.CreateObject("ADODB.Connection")
    Conn.Open ODBCPositions
    SQL = "SELECT JobTitle, Location, Description,
HiringManager,
PostDate FROM Positions"
     Set RS = Conn.Execute(SQL)
     Do While Not RS.EOF
%>
```

- Note the script + objects concept here.
- We can sub the VBScript for JSscript and it might seem super modern!
- Take-away is more the syntax rawness and the markup - code divide

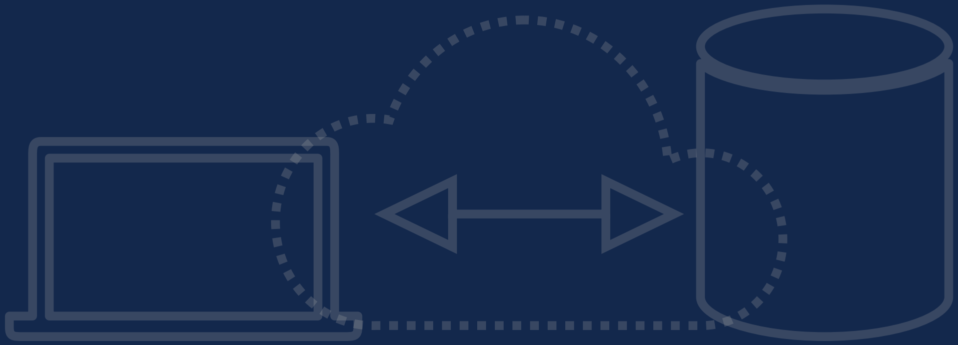
# Classic ASP is Back to the Future?

- Today server-side scripting is taking a hard turn towards JavaScript

- Obvious reasons
  - Fixed client-side language - no other choice in browser
  - **All things JavaScript** - Better to use one language than 2 or 3 or ...
  - Callback / Event Style Architecture makes sense for some types of network apps

- Interestingly classic ASP was often written in JScript (it was language neutral and usually VBScript or JScript), it looks a lot like some “new” stuff. **Common trend where there is little newness if you are historically aware though there is a point of timing for mass uptake**

# Introduction to PHP

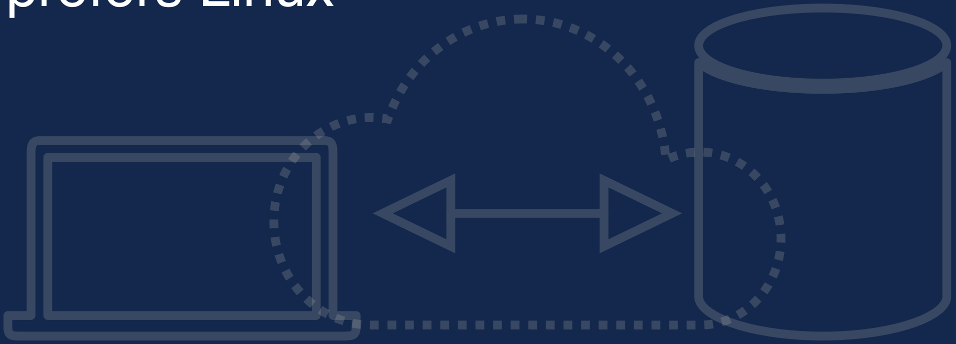


# PHP Intro

- PHP (www.php.net) is an open source server-side scripting language that works with a wide range of Web servers and operating systems.
- The language is relatively easy to learn, but includes many synonyms (discuss) and broad ecosystem so it can be daunting
- Some believe the language is probably not suited for extremely large system development though clearly Facebook has made it work?
- According to Netcraft PHP may be the most popular server-side technology on the Web. Note: File extension survey – be careful does this oversample people using WordPress implemented in PHP vs those who raw code?

# Why PHP?

- It’s easy
  - Syntax is familiar and forgiving
- It’s flexible
  - Doesn’t force a particular coding style
- It’s powerful
  - Lots of functions, lots of free code, extensible and works with lots of other technologies (ex. DBs, payment gateways, JSON parse,etc.)
- It’s free
  - Runs on just about anything* *prefers Linux
- Lots of people use it
  - Lots of assistance



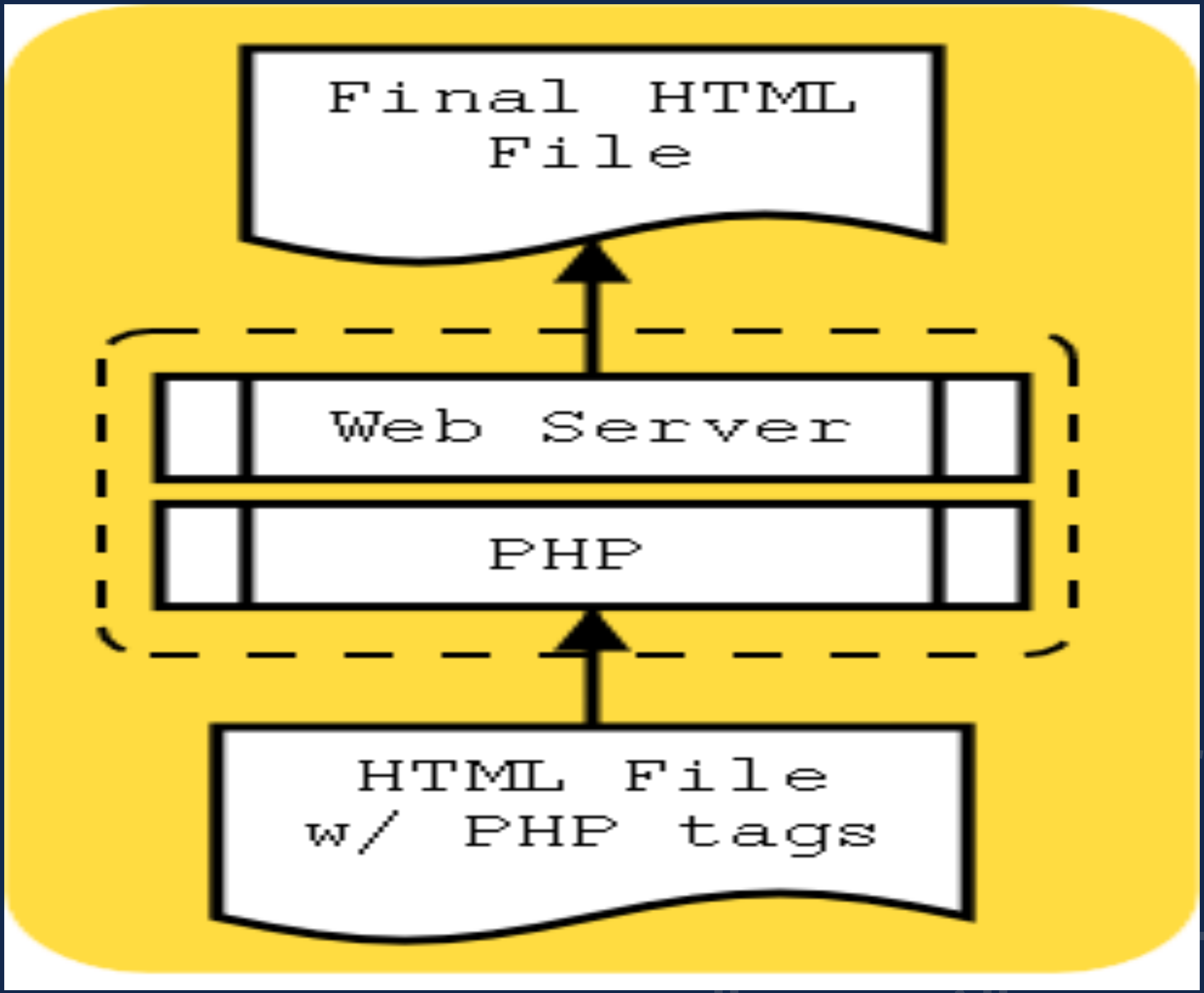
# Why not PHP?

- Open source stigma
  - Who do I call if it is broken?  Who is to blame (sue)?*
  - Who would pay for programmers in it
- It's too easy and super messy?
  - Type of people who use it often don't follow standard programming practices leading to "spaghetti code"
- Lack of quality and standardization
  - Too many frameworks, too many versions, platform quirks, etc.  Is this unique to PHP?  Think NodeJS!

# How PHP Works

- PHP files are usually an intermixture of PHP code with HTML
    - **Is the PHP is contained in the HTML or the HTML in the PHP?**
        - Factors - amount, emphasis, and developer bias
- PHP files are intercepted and evaluated by a scripting engine (generally implemented as a server module/ ISAPI though can be served CGI style) when requested producing the appropriate page
- The file extension .php is used to trigger the script engine though the extension is configurable

# Same Idea Simplified



# PHP Helloworld

```
<!DOCTYPE html>
<html lang=“en”>
    <head>
      <meta charset=“utf-8”>

      <title>Hello World PHP Style</title>
    </head>
<body>

<?php
  print "<p>Hello World from PHP!</p>“;
?>

<p>Hello World from HTML?</p>

</body>
</html>
```

# PHP Embedding Methods

- Typically PHP included in XML style
  - `<?php      ?>`
  - Short style `<?      ?>` is possible though not recommended and may require php.ini file change
  - If you want to include PHP in XHTML or XML files this is the best approach though `<script>` inclusion discussed next may also validate if appropriate CDATA directives are used

- Alternatively use the `<script>` tag
  - `<script language="php">    </script>`
  - 
    ```
    <script language="php">
    //<![CDATA[
       PHP code here
    //]]>
    </script>
    ```

# PHP Embedding Methods

- ASP style
  - `<%          %>`
  - Goal to provide easy upgrade path for ASP developers
  - To use this inclusion you will likely need to modify your server's php.ini file

- Direct echo
  - Within HTML you often find PHP variables and such being output directly using `<?=    ?>` For example `<input type="text" name="magic" value="<?= $foo; ?>" />`

# PHP External File Inclusion

- A valuable use of PHP is to include common site elements from a standard location such as headers, footers, navigation, and so on.

- There are numerous ways to include things in PHP including `readfile()`, `include()`, and `require()`.
  - It is also possible to use lower level file functions to perform inclusions.

- The `readfile()` function takes a specified file and writes its contents to output.
  - `readfile(‘footer.html’)`

# PHP File Inclusion methods

- You can also include content from another file using include

  ```
  <?php include ‘navbar.html’;  ?>
  ```

  - If included files are not found the page will continue to process but a warning will be issued
  - Note the file contents is not just included it is evaluated
  - Included file contents must be wrapped in `<?php ?>` since parsing will return to HTML mode upon inclusion

- The `require` statement is identical to `include`, but if the file is not found the page is aborted

  ```
  <?php require ‘global.inc’ ?>
  ```

# Determining PHP Settings

- PHP has a variety of settings for inclusion methods, loaded functions, etc.
  - Use function call **`phpinfo();`** to have the engine tell you its configuration
  - Usually you will want to locate the php.ini to make some changes (if you have rights to do so)
    - You will not on most lower end hosted sites or ones that you may not admin but rely on an IT group to help you
  - A simple call to **`phpversion()`** can give you simple version info
    - Example: **`echo 'Current PHP version: ' . phpversion();`**

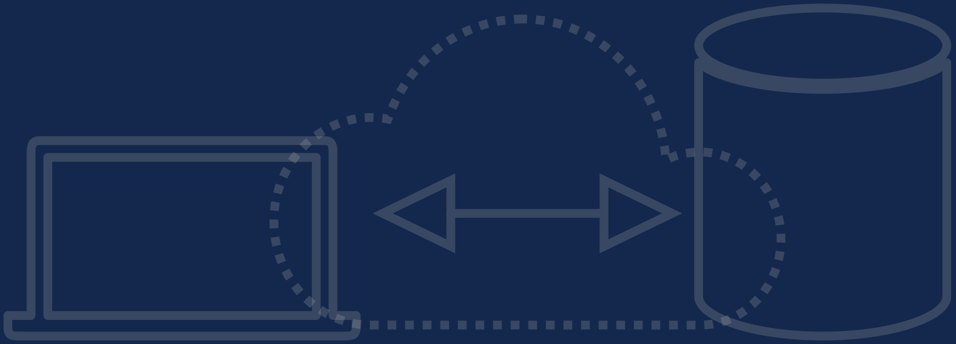
# Determining PHP Settings Contd.

- Part of the power of PHP comes from all the various modules that can be added in
- A simple way to see what is added is to make a call to `get_loaded_extensions()` from your script
  - Example: `print_r(get_loaded_extensions());`
- You can further look at the functions within a module named *modname* with `get_extension_funcs(modname);`
  - Example:

    ```
    print_r(get_extension_funcs('standard'));
    ```

# PHP Language Overview – Case Sensitivity

- PHP is **somewhat** case-sensitive (ugh!)
  - Built-in constructs and keywords like while, class, if, and so on are not case-sensitive
  - User defined classes and functions are not case-sensitive
  - Variables are case sensitive
- My case sensitivity groan is not limited to PHP, the intermixture of HTML, CSS, JS and other tech also has case and permissiveness conflict. Advice: Always pick a strict approach and stick with it!



# PHP Language Overview – Statements and Blocks

- PHP statements are terminated by a semi-colon.
- Blocks are indicated by curly braces `{ }`
- Note: The final statement before a closing script delimiter does not require a semi-colon.

# PHP Language Overview – White Space

- PHP is generally white space insensitive, though when intersecting with other languages (e.g. HTML, JavaScript, etc.) via output you should be cautious.

- Recall HTML is mostly white-space insensitive, CSS also can be this way, JS not as much.  However, in HTML white space handling can be overridden with tags (**`<pre>`**) and CSS (**`white-space: pre`**)

# PHP Language Overview – Intermixing HTML & PHP

- You’ll notice that people often use PHP to printout small snippets of XHTML or other client side tech.

```
<?php print “<h1>Hey there!</h1”; ?>
```

- Alternatively they might do this instead

```
<h1>
  <?php print “hey there!” ; ?>
</h1>
```

- You can see the issue of the PHP in the HTML or the HTML in the PHP
- For me the choice is a function of type (site vs app) and more importantly amount (code vs markup) in terms of what I emphasize.

# PHP Language Overview – Comments

- PHP supports 3 forms of comments
  - UNIX Shell style:

    ```
    # I am a single line comment
    ```

  - Standard C style:

    ```
    /* I am a multi-line
       comment */
    ```

  - C++ style:

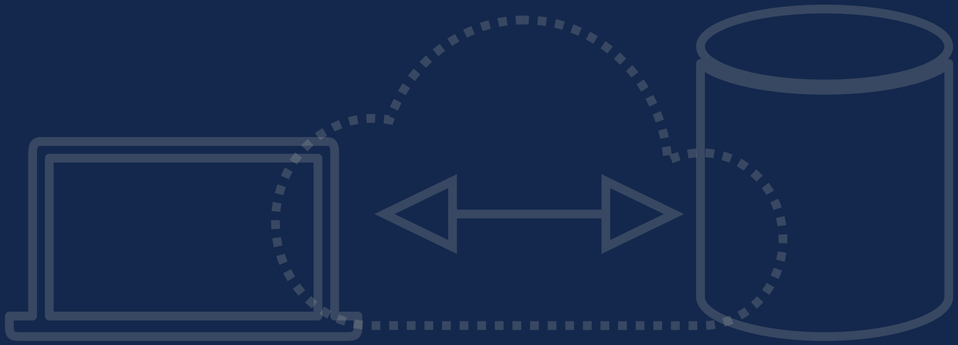
    ```
    // I am a single line comment
    ```

- Note: Comments are stripped from output (good thing) by the interpreter.

- Question: Are comments and formatting a good thing in your PHP source?

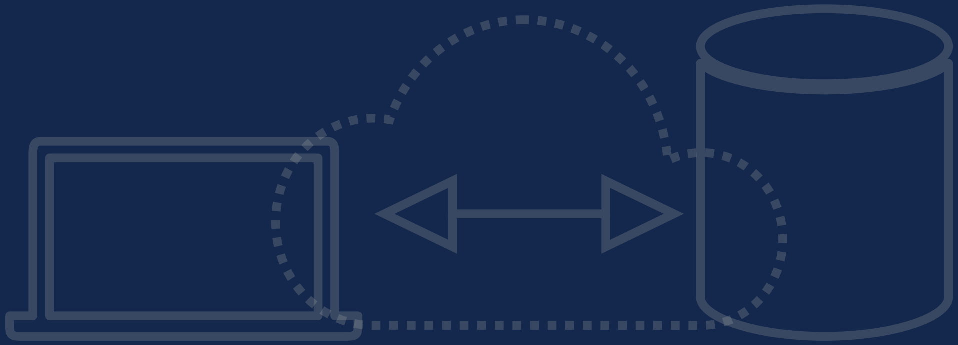
# PHP Language Overview

- Variable names in PHP always begin with a `$` and are case sensitive so `$Name` is not equivalent to `$NAME` or even `$NaMe`.
- Legal identifiers in PHP start with a letter or underscore and may be followed by these characters as well as digits
  - Good Variable Names
    - `$name`
    - `$_browser`
    - `$ThreeStooges`
  - Bad Variable Names
    - `$3Stooges`
    - `$$fun var`
    - `$|foo`



# PHP Language Overview

- PHP has eight data types:
  - 4 scalars
    - Integers
    - Floating-points
    - Strings
    - Booleans
  - 2 collections
    - Arrays
    - Object
  - 2 specials
    - Resource
    - Null



# PHP Language Overview

- Integers
  - Decimal, octal, and hex all allowed
    - Good: 754, -3, 0755 (octal), 0xFF (hex)
  - Range typically -2,147,483,648 to 2,147,483,648
    - Usually similar to the long type in C
    - When you exceed allowed range the data should be converted to floating point
  - Use `is_int()` or `is_integer()` to test a value to be an integer

- Floating Points
  - Normal style 3.14, -0.54 and scientific notation 0.314E1 or 1.56E10
  - Be wary of normal comparison problems with floats
  - Use `is_float()` or `is_real( )` to test a value to be a float

# PHP Language Overview

- Strings
  - Sequence of characters of any length (no char type)
  - Single and double quotes to delimit but be careful there are differences
  - Careful within `“”` a variable like `$name` is expanded to its value but within `‘’` its is treated as a string
    - `‘ ‘` printing should be faster executing since it doesn’t have to interpolate variables
  - Escape sequences a la C used within `“”` like `\n`, `\”`, `\t`, `\\`, etc.
    - You might find `\$` to be quite useful
  - Within `‘` only `\\` and `\’` are recognized escape sequences
  - Use `is_string()` to test if a value is a string
  - Numerous string manipulation functions as expected

# PHP Language Overview

- Booleans
  - False values are defined as
    - The keyword `false`
    - Numbers like 0 and 0.0
    - Empty string “” and “0” string
    - An array with zero elements
    - Object with no values or function
    - Null
  - True is everything else so,
    - True
    - Any number but 0 and 0.0
    - Non-empty strings
  - Use `is_bool( )` function to test for the data type

# PHP Language Overview

- Arrays
  - Zero indexed
  - Use `array( )` construct to make create an array
    - `$stooges = array(‘Larry’, ‘Curly’, ‘Moe’);`
    - `print $stooges[0];`
    - Use `foreach` to iterate over an array

```
foreach ($stooges as $name)
       {
          print “Hello $name <br />”;
       }
```

  - Numerous functions for array manipulation like `sort()`
  - Use `is_array( )` to test for the type

# PHP Language Overview

- Objects
  - PHP does support OOP but still has a way to go - define a class with the `class` keyword

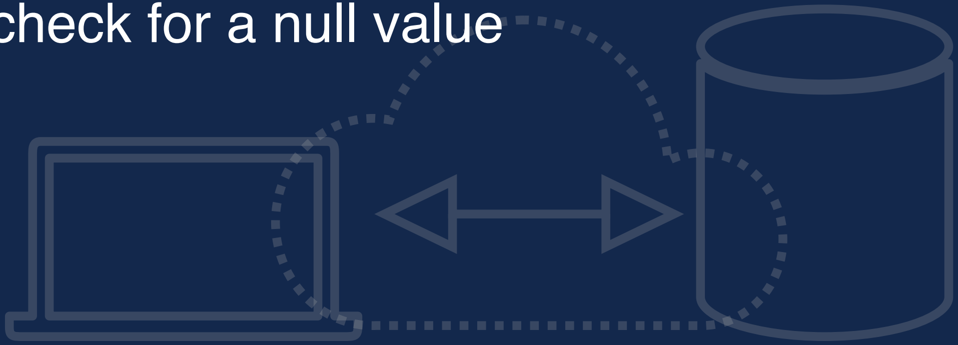
```
 class Dog {
     var $name = ‘’;
     function name ($thename == NULL)
       {
         if (! is_null($thename)) {
          $this->name = $thename;
          }
        return $this->name;
       }
   }
$dog1 = new Dog;
$dog1->name(‘Tucker’);
$dog2 = new Dog;
$dog2->name(‘Angus’);
```

- Use `is_object( )` to detect if something is an object type

# PHP Language Overview

- Resources
  - A special data type used in some sense like a file handle (it is an integer under the covers)
  - Often used with a database connection
  - Function `is_resource( )` used to check the data type

- Null
  - Simple null value indicating a lack of data
  - Somewhat useful for garbage collection hinting
  - The function `is_null( )` used to check for a null value



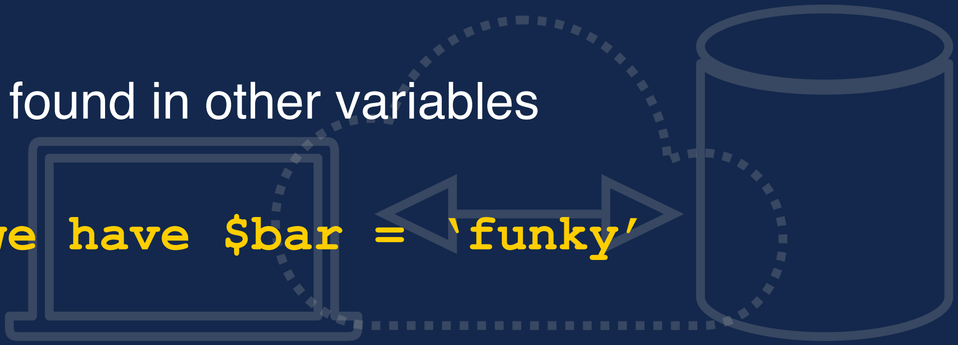
# PHP Language Overview

- Variables
  - Prefixed by a dollar sign **$**
    - **$browser** , **$name** , **$FAVORITE_NUMBER**
  - Variables do not have to be declared before use, they are defined upon first use
  - There is no need to limit a variable to a particular type, PHP allows a variable to hold any data type
    - You can cast variables though to a particular type to be safe

```
$myName = (string) “Thomas”;
$favNum = (string) $favNum;
```

  - Unset variables act as null
  - Variables can be created out of strings found in other variables

```
$foo = ‘bar’;
$$foo = ‘funky’;    # now we have $bar = ‘funky’
```



# PHP Language Overview

- Variables can have aliases
  - `$foo =& $bar  # now $foo is an alias to $bar`
  - Reference values can be somewhat tricky in that you can modify values indirectly
  - More confusion comes if you unset a value as it may retained in aliases

```
$foo = "bar";
$foo2 =& $foo;
unset($foo);
print $foo2;  # shows bar
```

  - Aliases values can be used in functions for parameter passing and to avoid large copy tasks

# PHP Language Overview

- You can define constants in PHP using the define construct

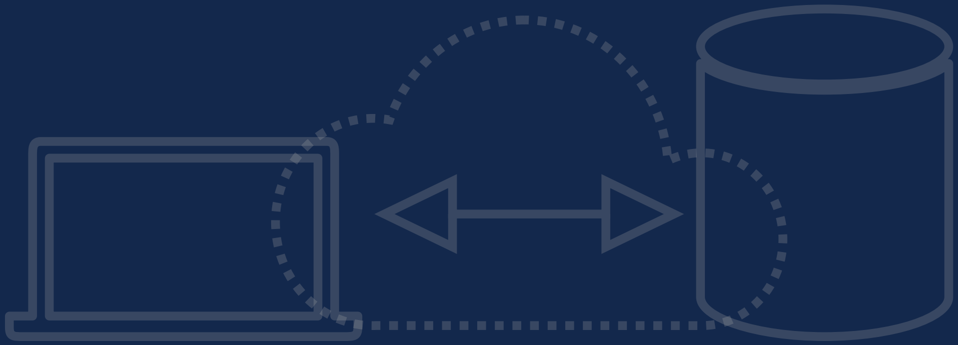
```
define('NAME', 'Thomas A. Powell');
// now you can use NAME anywhere
echo 'HI' , NAME
```

  - Convention for constants is to use all caps
  - Constants can only contain scalar data
  - C programmers should watch out it isn't `#define`
  - Careful with collisions particular if using other people's code
    - `get_defined_constants()` function may be useful
  - Some built in "MAGIC constants" `__LINE__` , `__FILE__`, `__FUNCTION__`, `__CLASS__`, `__METHOD__` mostly used for debugging

# PHP Language Overview

- Scoping
  - Variables declared (first use) within a function are local to that function
  - Variables declared outside a function are considered global
  - Variable within a function using the keyword static are local to the function and retain value between calls

- Function parameters are of course local to the function they are defined for



# PHP Language Overview

- Garbage Collection
    - Given the described variable and scope system it is pretty obvious that PHP garbage collects
    - You can use NULL or more appropriately unset( ) to add in memory collection
    - Question - Why might this not be that important to you?

# PHP Language Overview

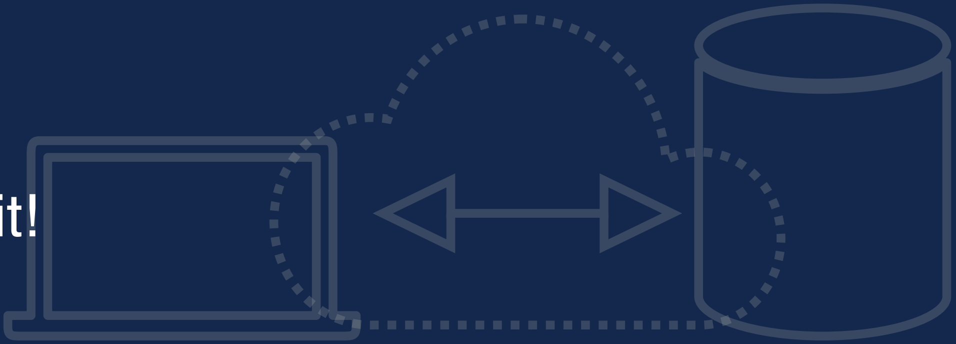
- Operators Overview
  - Generally similar to most languages but in mixed expression you need to watch out for type conversions
  - You can cast yourself using (int), (float), (string), etc. in front of the variable you want to convert

    ```
    $a = “5”;    $b = (int) $a;
    ```

- Specific operators
  - Arithmetic: `+`,`-`,`*`,`/`,`%`
  - String concatenation: `.`
    - Awkward here if you think OOP style
  - Increment/Decrement: `++`, `--`
    - You can autoincrement/decrement letters!

# PHP Language Overview

- More operators
  - Comparison: ==, ===, !=, >, >=, <, <=
    - You can also use <> for !=
  - Bitwise: ~, &, |, ^, <<, >>
  - Logical: &&, ||, !
    - Also supports and, or, xor
  - Assignment: =, +=, -=, /=, *=, %=, .=
  - Conditional: ? :
- Misc. operators
  - @ for error suppression
  - ` ` for shell execution. Watch it!



# PHP Language Overview

- Statements are pretty much the same as most imperative languages but you see some funny syntax here and there

```
if (expression)
   statement or block
if (expression)
   statement or block
else
   statement or block
```

- `elseif` is also supported
- Special `:` form is used in place of `{ }` and then has an `endif`

# PHP Language Overview

- Switch statement
  - Standard style with block

```
   switch ($grade) {
    case ‘A’: echo “Great!”;
                   break;
    case ‘B”: echo “Good”;
                   break;
    …
    default: echo “Error”;
  }
```

  - Substitute `{` for `:` and `}` for `endswitch` if you like

```
  switch ($grade) :
       case …
    endswitch;
```

  - Without break statement you get standard switch fall-through effect to create an “or” selector

# PHP Language Overview

- While loop
  - Standard style

    ```
    while (expression)
        statement or block;
    ```

  - Use `continue` statement to skip back to the loop condition check
  - Use `break` statement to break a set of braces
    - Interesting PHP allows you to pass a number to break to indicate the number of blocks to exit.  Most languages provide a label target

  - A `do/while` loop is also supported and causes the loop to execute at least once

# PHP Language Overview

- For loop
  - Standard style

```
for (start; check; increment;)
    statement or block;
```

  - Alternate style

```
for (start; check; increment;)
    statement1;
     …
    statement n;
endfor;
```

  - Remember you can string statements together in loop start, check, etc. using the comma operator
  - `break` and `continue` statements used to control for loops

# PHP Language Overview

- Foreach loop
  - Allows you to iterate over elements in an array

```
foreach ($array as $current)
    statement or block;
```

  - Alternate style

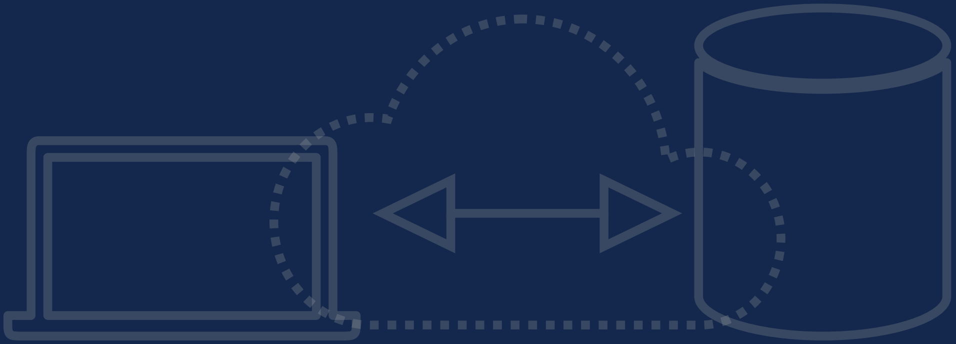
```
  foreach ($array as $current)
    statement1;
     …
    statement n;
endforach;
```

  - You can also loop and access both key and value

```
 foreach ($array as $key => $value)
    statement1;
     …
    statement n;
endforach;
```

# PHP Language Overview

- Return
  - A return statement will exit a function or if at top level the script
- Exit
  - The `exit( )` statement will exit the current process/script
  - You can pass a value indicating a status code or error message
  - It is a synonym for `die( )`
    - `die(“This script just croaked!”);`



# Calling Functions

- Calling functions whether user defined or system defined is the same

```
$result=function_name([ parameter,… ] );
```

- Functions of course do not need to take parameters (though the very often do) or return a value that is used (though it typically is)

- Example:

```
 $len = strlen(“UCSD”); // $len = 4
```

# Defining Functions

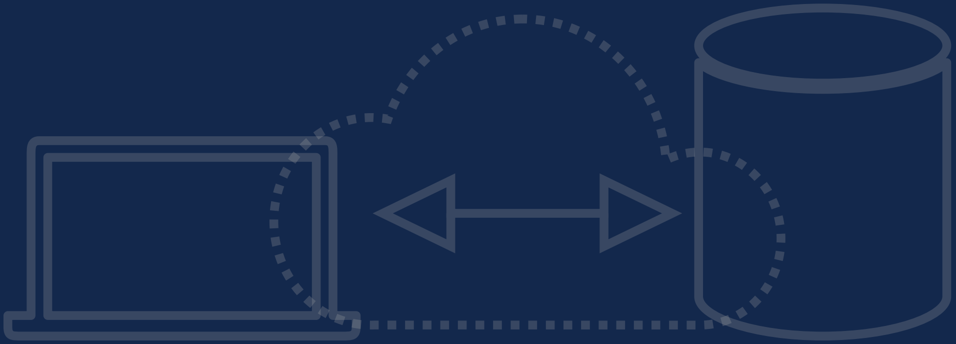
- Basic function definitions of the form

```
function [&]function_name(parameter-list)
  {
      statement(s) often with return(s)
  }
```

- Example

```
function add2($x)  {
  return $x+2;
}

$result = add2(5);
echo $result;
```



# Returning Values

- You can return a single value with return, multiple values of course can come within an array.
- Normally a single value is passed back by copy, but for performance you may find a reference indication to be useful
  - The optional `&` in a function definition causes return values to be passed by reference rather than by value (a copy)

# Local and Global Variables

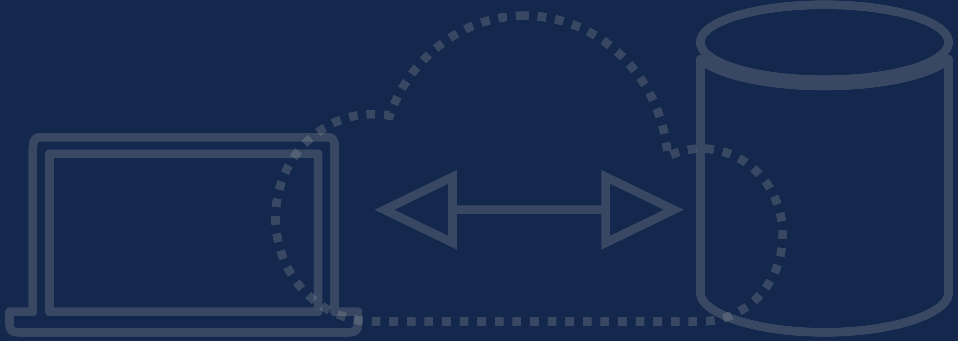
- Variables defined within a function are local and not accessible outside the function.
- Global variables are not usable inside a function either unless prefixed by the keyword global
- There are “superglobals” like `$_GET[ ]` that you can access without the **global** keyword

```
$a = 1;
function foo()
  {
   $a = 2;  // changed a local a
  }
function foo2()
  {
   global $a = 2;  //change the global a
  }
```

# Static Variables

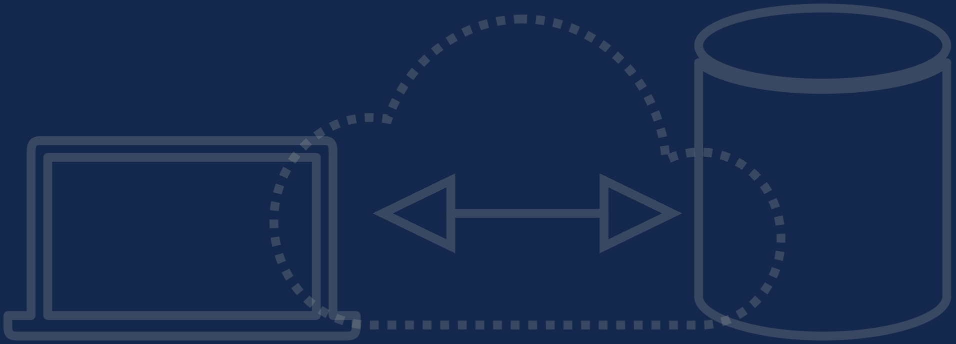
- A variable with the keyword `static` in front of in a function will preserve its values between calls

```
function foo()
   {
    static $a = 1;
    return $a++; // keeps growing
   }
```



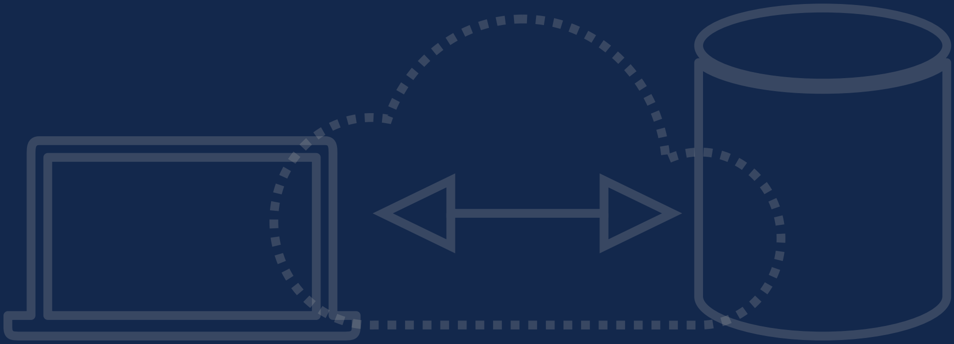
# Pass by Value and Reference

- Normally parameters are passed by value but if you add an `&` into the parameter name it will be passed by reference
  - Given `function funref(&$x) { }` and `function funval($x)` the function `funref` can effect changes directly on `$x` where as the standard `funval` cannot



# Advanced Parameters

- Default parameters can be set by putting an assignment in the parameter list.
  - Default parameters should come after standard parameters
  - Correct: `function foo($x,$y=5) { }`
  - Incorrect: `function foobar($x=5,$y) { }`
- Note: you may call functions and forget to pass parameters, but warnings will be issued



# Advanced Parameters Contd.

- A function with variable parameters can be accessed via the `func_num_args( )`, `func_get_arg()` and `func_get_args( )` functions from within the function itself.
  - The `func_get_arg( )` takes a number for the particular argument to fetch while `func_get_args()` returns an array of all the arguments
  - Example:

```
function foo()
  {
   print “I have func_num_args()”;
   print “Argument 1 = func_get_arg(1)”;
   print_r(func_get_args());
  }
```

# Variable Functions

- A variable function means that given a variable with `$x` with ()s attached, the PHP engine will look to call a function called whatever the value of `$x` is
  - This can be used to implement some elegant (or confusing) solutions for function tables, callbacks, and the like similar to how some use `eval()` in JavaScript
  - Example:

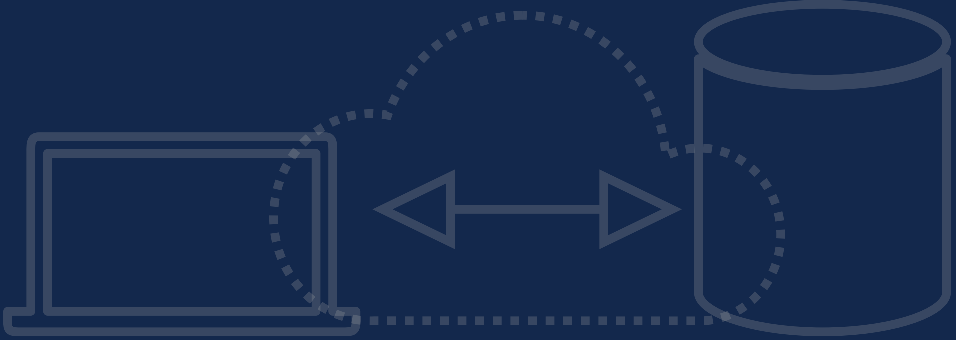
```
$x = ‘foo’;
$x(); // calls $foo assuming it exists
```

# Anonymous Functions

- You can create an anonymous function using `create_function`
  - `$myfunction = create_function(args, code);`
  - This may be useful within other function calls or objects

- Example:

```
 $add2 = create_function(‘$x,$y’, ‘return $x+
$y’);
 echo $add2(5,10);
```



# Classes

- In PHP, we define a class with the ‘class’ keyword followed by the class name:

```
class BankAccount{


}
```

- When we use the ‘class’ keyword, we are creating a datatype.  In order to use the type, we must create a variable with the ‘new’ operator.

```
$myAccount = new BankAccount();
```

# Properties and Methods

- A class can contain properties and methods.  Our BankAccount example will need to have a balance and a way to change the balance.

```
class BankAccount{
    var $balance;

    function setBalance($balance){
        $this->balance = $balance;
    }

    function getBalance(){
        return $this->balance;
    }
}
```

- In this example, the $balance variable is a property (unfortunately public here) and setBalance() and getBalance() are the methods.

# Methods and $this

- We can call setBalance after we create an instance of BankAccount.

```
$myAccount = new BankAccount();
$myAccount->setBalance(100);
```

- In the setBalance() method, we see the keyword '$this'

```
function setBalance($balance){
        $this->balance = $balance;
 }
```

- $this is a reference to the calling object.  In our example above, $this would be pointing to the $myAccount object.  This allows us to later reference the class property through another method call.

```
$myAccount = new BankAccount();
$myAccount->setBalance(100);
 echo $myAccount->getBalance();   //prints 100
```

# Constructors

- Constructors are optional in PHP classes.
- If a constructor is defined, it will always be called when an object is created.
- A constructor is declared by creating a class method called __construct.

```
function __construct(){
        $this->balance = 50;
}
```

- There can only be one constructor per class.

# Constructors Cont.

- It is also possible to pass a constructor a variable.

```
function __construct($start){
        $this->balance = $start;
}
```

- When creating the object, we would pass a value to the new operator.

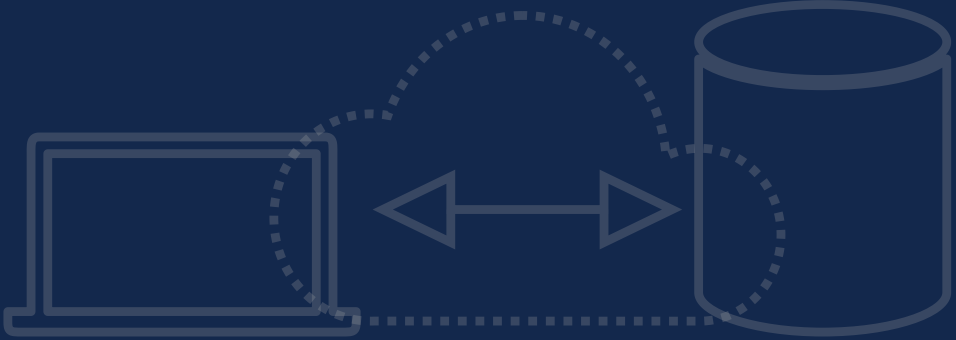
```
$myAccount = new BankAccount(25);
```

- Example: http://phpclass.pint.com/examples/OOP/classconstruct.php

# Inheritance

- One important feature of OOP is inheritance. PHP supports inheritance with the “extends” keyword.

```
class CheckingAccount extends BankAccount{
      var $pin;
   function setPin($pin){
           $this->pin = $pin;
      }
   function getPin(){
             return $this->pin;
      }
}
```



# Inheritance Cont.

- Now, any object of the type CheckingAccount will have two class properties: pin and balance.

```
$myAccount = new CheckingAccount(0);
$myAccount->setBalance(100);
$myAccount->setPin(“3333");
```

- Note: Even though the CheckingAccount class does not have a constructor, we pass in a value to the new operator. This is because the BankAccount constructor will be used.

# Access Control

- Access control is set for the class properties with the keywords ‘private’, ‘public’, and ‘protected’.
- Private: Only the class that declares the property can access it.
- Public: Everyone can access the property.
- Protected: The class that declares the property and any classes the inherit or are inherited from the class can access the property.

# Access Control cont.

- We use the appropriate keyword instead of var when declaring the variables:

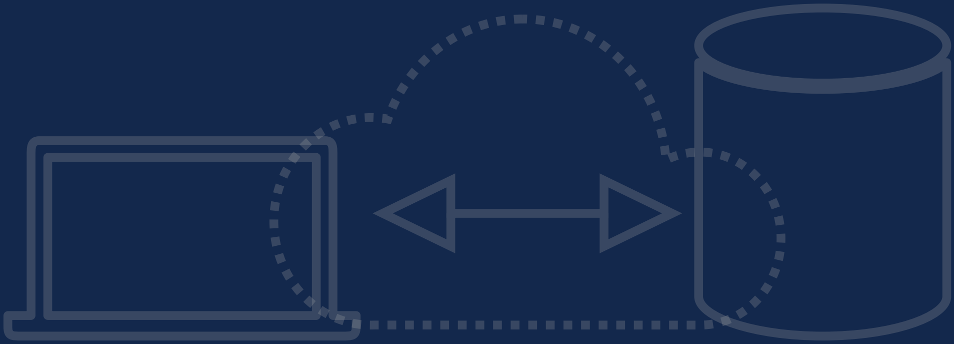
```
protected $balance;
private $pin;
public $name;
```

- Now, we can access $myAccount->name directly, but we get an error if we try

```
$myAccount->pin;
```

- Methods set access control in a similar way. Simply precede ‘function’ with the appropriate keyword.
- Constructors must always be public.

```
public function __construct()
```



# Static Variables

- We use the ‘static’ keyword to declare a class property or method that we wish to access outside of an instantiation of the class.

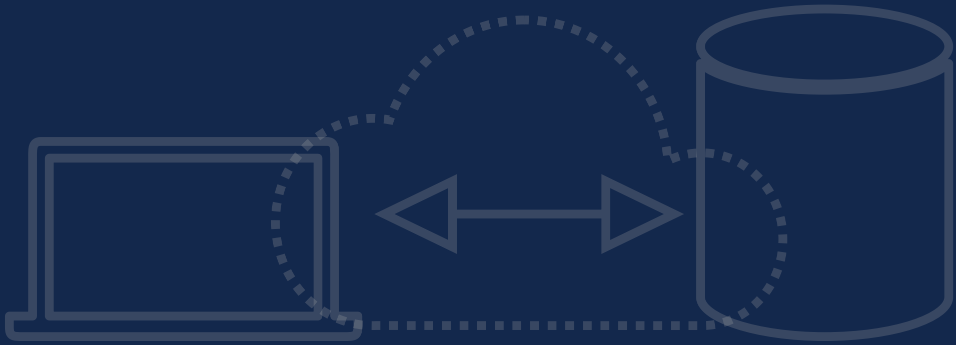
```
public static $bankname = "Wells Fargo";
```

- Inside of the class, we can reference the property with the ‘self’ keyword and the scope operator :: .

```
function getBank(){
    return self::$bankname;
}
```

- Outside of the class, we can reference the property by calling the class name along with the scope operator

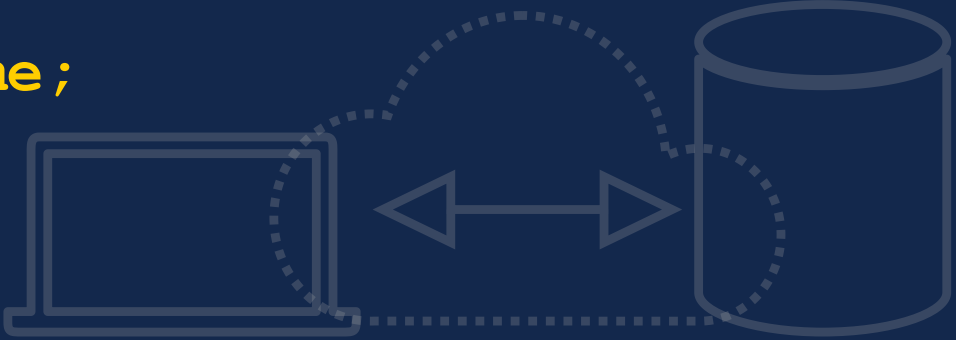
```
echo BankAccount::$bankname
```



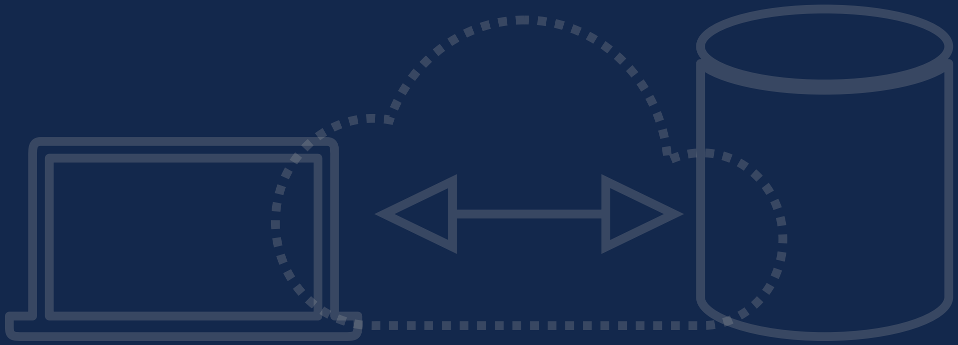
# Constant Variables

- Class Constants are declared in a similar manner to static variables. However, the value can not be changed. Note that the variable is not proceeded with a ‘$’.

```
class BankAccount{
 const bankname = ‘Wells Fargo';
 function getBank(){
         return self::bankname;
     }
 }
echo BankAccount::bankname;
```



# Ecosystem Example
# A Few PHP Packages



# Built-in String Functions

- PHP has a tremendous number of string functions
  - See http://us3.php.net/manual/en/ref.strings.php for a complete list

- A few you might commonly use include
  - `strlen()` – string length
  - `strtolower()`, `strtoupper()`, `ucfirst()`, `ucwords()` – casing conversions
  - `trim()`, `ltrim()`, `rtrim()` – remove white space from strings or left/right parts of the string respectively
  - `addslashes()`, `stripslashes()` – escape characters and remote escapes
  - `strip_tags()` – strip the tags from the string
  - `strtok()` – tokenize a string
  - `substr()` – substringing (many others)
  - `strpos()` and others to find things within a string
  - `urlencode()` and `urldecode()` – not quite string function listed but very related

- Insight - As a Web driven language PHP is really good at handling strings in and out

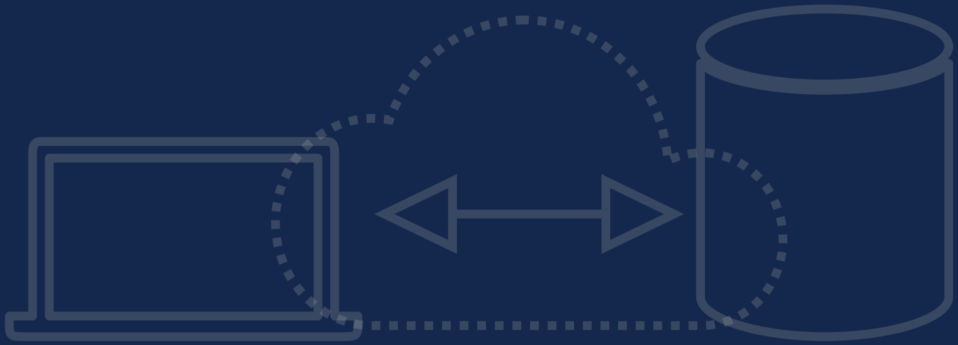
# Improved Printing

- There are many useful printing routines beyond simple print() and echo()
  - `number_format()`
  - `money_format()`
  - `printf()`
    - Also `sprintf()` useful just to do a conversion
  - Print to here idea from Perl
  - Useful debugging: `print_r()` and `var_dump()`

- Insight - lots of template specific ideas in PHP for output, “” interpolation, formatting, include files, etc.

# Dynamic Graphics

- You don’t have to just output HTML you can do graphics too!
- Make sure your version of PHP has graphics support installed. If it does you may be able to manipulate images quite easily in common formats like GIF and JPEG
- Manual: http://www.php.net/manual/en/ref.image.php



# Helloworld with GD

```php
<?php
    header ("Content-type: image/png");
    $img_handle = ImageCreate (120, 30) or die ("Cannot Create image");
    $back_color = ImageColorAllocate ($img_handle, 255, 255, 0);
    $txt_color = ImageColorAllocate ($img_handle, 255, 0, 0);
    ImageString ($img_handle, 31, 5, 5,  "Hello world!", $txt_color);
    ImagePng ($img_handle);
?>
```

Note: Using `header()` this must be the first and only output from the file – more on this later

Syntax:

- `resource imagecreate` ( int x_size, int y_size )
- `int imagecolorallocate` ( resource image, int red, int green, int blue )
- `bool imagestring` ( resource image, int font, int x, int y, string s, int col )
- `bool imagepng` ( resource image [, string filename] )

# Some Basic GD Examples

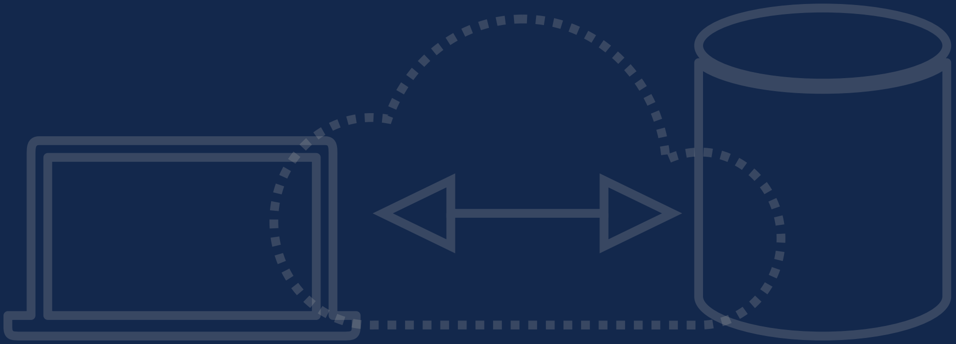
- Basic creation of lines, basic shapes, and fills both flood and object style
  - `imageline($im, x1,y1,x2,y2,color)`
  - `imagerectangle ( $im, x1, y1, x2, y2, color )`
  - `imageellipse($im, cx,cy height, width, color)`
  - `imagepolygon($im, array of points, numpoints, color)`
  - `imagefilledpolygon($im,array of points, numpoints, color)`
  - `imagefill($im,x,y,color)  // starts flood fill at x,y`
- Insight: The ecosystem of a language is a key aspect of what makes the environment legitimate for people and efficient to use. Hopefully, you are attracted to the ease of dev!

# Go Explore

- Visit PHP.net and see what is available.  You'll see common DBs easily supported, useful format handling (JSON, XML), the ability to handle HTTP (for bots and scraping), and much much more

- Caution: You need to make sure that your PHP instance has these things installed. Use `phpinfo()` to check if needed.

# Coding Web Applications



# Traditional Form Handling in PHP

- In traditional PHP if you had a page with a form with fields like

```
<form action=“page2.php” method=“get”>
   <input type=“text” name=“username” />
   <input type=“text” name=“age” />
   <input type=“submit” value=“send” />
</form>
```

Then after submission on page2.php you should automatically have variables $username and $age which corresponded to whatever the user had filled in on the previous page.

- So in page2.php you might simply have

```
<?php
 print “Hi $username you are $age years old!”;
 if ($DEBUG) doStuff();
?>
```

The ease of DX here hurts our security.  This tension continues on. Interestingly this initial decision of PHP taints it to this day

# Registered Globals Issues

- For security and program safety issues this style is typically not available by default in many PHP installations though you can enable it by setting register_globals to on in the php.ini file.
  - Issue: Injecting data directly into your program via GET or POST
  - Result: Trigger unintended actions (ex. Code flow, debug statements, etc.)
- Checking for setting
  - `ini_get('register_globals');`
  - Use with @ to suppress errors in an if to decide what to do
  - Note corresponding `ini_set()` does not allow `register_globals` to be set at runtime

Look you can go ahead and make it insecure…all the environment often have overrides where you might hurt self

# Safer Form Handling

- Rather than relying on registered globals use the superglobal variable arrays `$_GET`, `$_POST` or `$_REQUEST` which includes GET, POST, and Cookie vals

- For example to fetch out the values in the next page (page2.php) we might use code like if a GET method was used

```
$username = $_GET[‘username’];
$age = $_GET[‘age’];
```

  or alternatively regardless of method

```
$username = $_REQUEST[‘username’];
$age = $_REQEUEST[‘age’];
```

- The added complexity forces a safer coding posture

# Importing Request Variables

- For those folks that really like the style of coding that registered globals provides you can use the `import_request_variables()` function.
  - Syntax: `import_request_variables("what" [,prefix]);`
  - Where what is a string that contains a string containing g, p, and/or c indicating to import GET, POST, or COOKIE values respectively
  - The prefix value is optional and will append the string specified in front of any imported values.  For safety sake it is encouraged
  - Example: `import_request_variables(“gp","r_");`

# Self Posting

- One design pattern that PHP developers often use with forms is the idea of a single file form/response
- The basic idea is to have the page fork depending on if its collecting to user data or responding to the input.  It can be extended of course to address validation as well
- Pseudo code looks something like

```
if  form data not collected
  print form fields and allow user to submit
else if invalid form data
 print form fields with errors and allow resubmit
   else
      perform form action (and often redirect)
```

# Self Posting Contd.

- An easy way to set the action of a self posting form is to use the PHP_SELF value in the form action

  -
    ```
    <form action=" <?= $_SERVER[' PHP_SELF' ]
              ?>"  method=" GET or POST"  >
    ```

  - Often times the trigger is to look for the existence of a particular variable being set either in a submit button or <input type=“hidden”> or to look at the method the page is called with GET or POST

  - You may also note the troubling issue with form posting that happens particularly with POST.

# What Could Go Wrong?

- Well just about everything, you should assume nothing works
  - What happens if the fields aren’t filled in that are needed?
  - What happens if you get the wrong type for fields?
  - What about if someone figures out how to send fields they shouldn’t or avoid maxlength you might have set?
  - Could data be badly formatted?

- **General Web Dev Rule: Never trust user inputs (query string, body or headers!)**

# Basic Validations

- `isset()`
  - Make sure a variable is set before you play with it
- `trim()`
  - Trim a variable of extra spaces
- `empty()`
  - Determines if a value is empty
- `Type casting`
  - Cast a variable the way you want it
  - `$age = (integer) _POST[‘age’];`
  - `Settype($age, “integer”);`
- Explicit range checking
  - If statements
  - Regular expressions

# Form Action

- Up until now we have echoed form data, we try a simple example to show what’s next by sending an email based upon form data
  - Basic syntax:

```
bool mail ( string to, string subject, string
  message [, string additional_headers [, string
  additional_parameters]] )
```

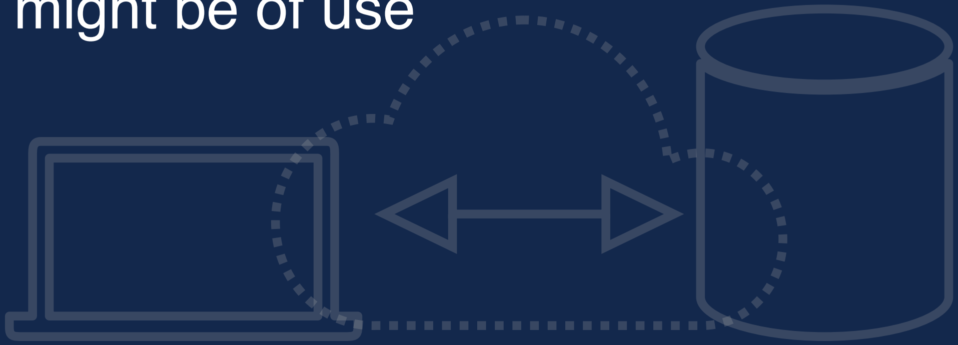
  - Example:

```
mail( “tpowell@example.org”, "Feedback Form Results",
$message,
   "From: $email"
```

Manual entry - http://us3.php.net/manual/en/function.mail.php

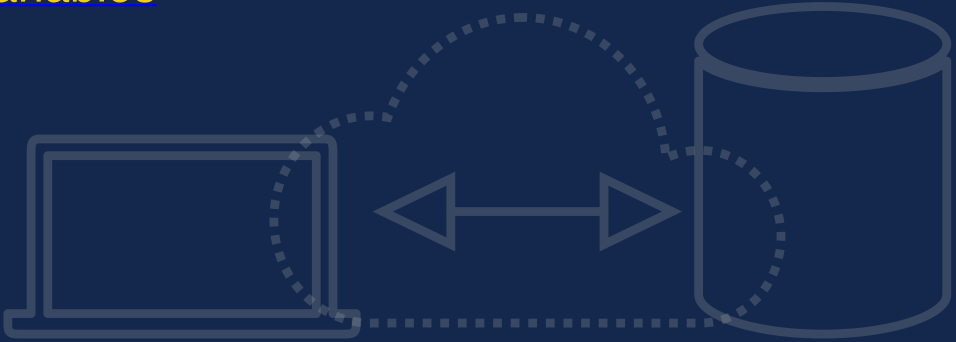
# Reading Headers and Environment

- If you recall from our lengthy HTTP discussion there is more than just reading the query string or message body (as in a POST), you can also gain valuable insight from the various headers transmitted in a request
  - Useful headers reminder – User Agent, Accept, Accept-Language, etc.
- You also find that the environment in which the transaction takes place might be quite useful
  - Time and IP address in particular might be of use



# Reading Headers and Environment Contd.

- Most of the interesting items are in the **`$_SERVER[]`** super global array
  - All the HTTP headers will be prefixed with **`HTTP_`**
    - Examples – **`HTTP_ACCEPT`**, **`HTTP_REFERER`**, **`HTTP_USER_AGENT`**, etc.
  - Others are variables
    - **`PHP_SELF`**, **`REMOTE_ADDR`**, **`REMOTE_HOST`**, **`REQUEST_METHOD`** but should be familiar from the "environment variables from CGI" - Remember as I have shown a few times the surface area is just the HTTP "conversation" and the common web data types (HTML, CSS, JSON, etc.) so that you see such commonality should be expected not surprising!
  - Reference: http://www.php.net/reserved.variables



# Header Output

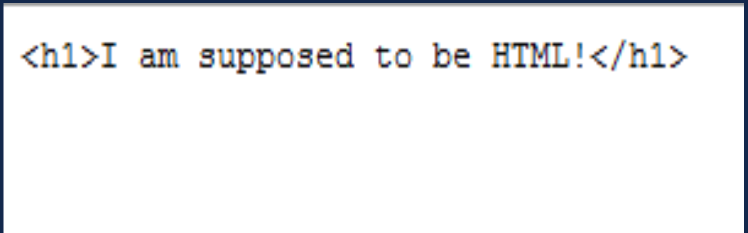
- You can output HTTP headers as you like using the header() function.

```
header(“Content-Type: text/plain”);
```

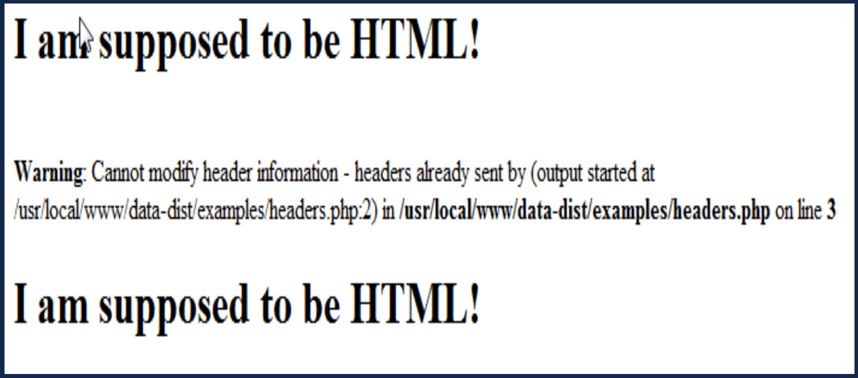
- If you understand the typical HTTP data stream you can see that this must be output first for it to work
  - Any screen output before this line will cause PHP to do its normal duty creating an HTML header stream and application of the function later should result in an error
- Consider the following used right and wrong

```
<?php
   header("Content-type: text/plain");
  ?>
  <h1>I am supposed to be HTML!</h1>
```

# Header Trouble



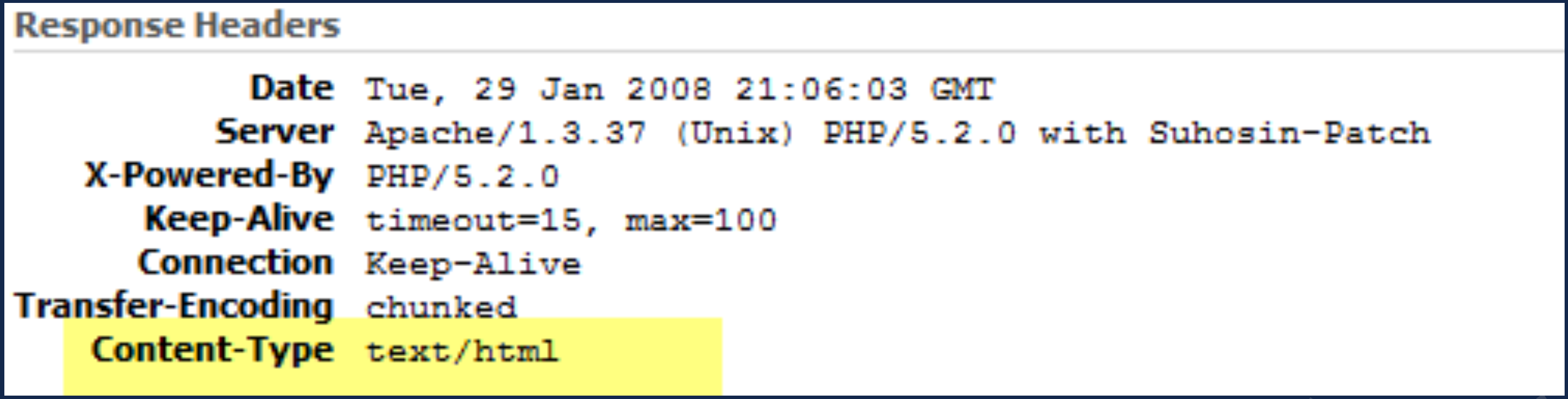
```
<h1>I am supposed to be HTML!</h1>
```



# I am supposed to be HTML!

**Warning**: Cannot modify header information - headers already sent by (output started at /usr/local/www/data-dist/examples/headers.php:2) in **/usr/local/www/data-dist/examples/headers.php** on line **3**

# I am supposed to be HTML!



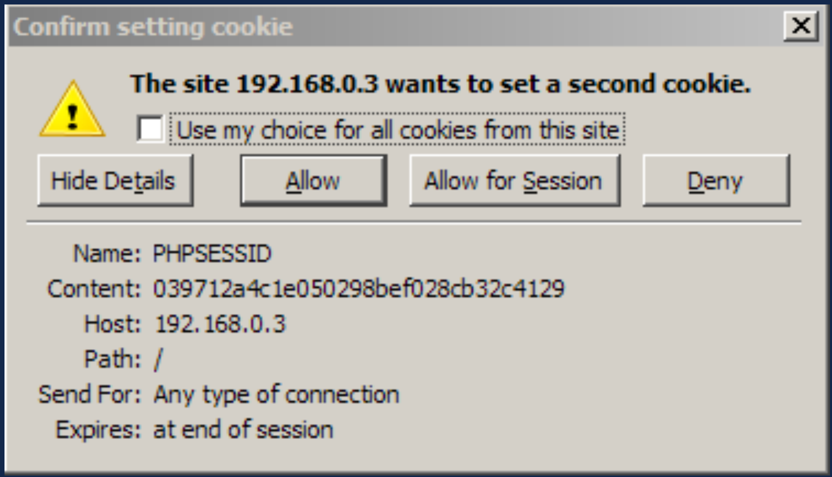
**Response Headers**

```
             Date  Tue, 29 Jan 2008 21:06:03 GMT
           Server  Apache/1.3.37 (Unix) PHP/5.2.0 with Suhosin-Patch
     X-Powered-By  PHP/5.2.0
       Keep-Alive  timeout=15, max=100
       Connection  Keep-Alive
Transfer-Encoding  chunked
     Content-Type  text/html
```

Danger: As we have seen respecting the Law of Three order doesn't seem an issue, as buffering allows us not to think about it.  This of course has a performance and scale cost.

# Sessions in PHP

- Use `session_start()` and PHP will issue a cookie with name PHPSESSID and some value
- You will use `session_start()` most likely on every page in your app



The site 192.168.0.3 wants to set a second cookie.

☐ Use my choice for all cookies from this site

Name: PHPSESSID
Content: 039712a4c1e050298bef028cb32c4129
Host: 192.168.0.3
Path: /
Send For: Any type of connection
Expires: at end of session

- It is possible to set a php.ini setting of session.auto_start so that you don’t need to use `session_start()` all the time

- You can specify a different value other than PHPSESSID before by calling `session_name(‘MYID’);` where MYID is your value and then calling `session_start();`

- To create a session variable use `$_SESSION[‘`*`var`*`’] = ‘`*`value`*`’;`

  Example: `$_SESSION[‘stooge’] = ‘moe’;`

# Session Example in PHP

```
<?php
   session_start();
   $_SESSION['fav_stooge']='Moe';
   $_SESSION['num_stooges']=3;
?>
<html
<head>
<title>Session Fun</title>
</head>
<body>
<h1>Just set some session values</h1>

<a href="session2.html">Next page</a>
 <!-- eventually read this later on session3.php -->

</body>
</html>
```

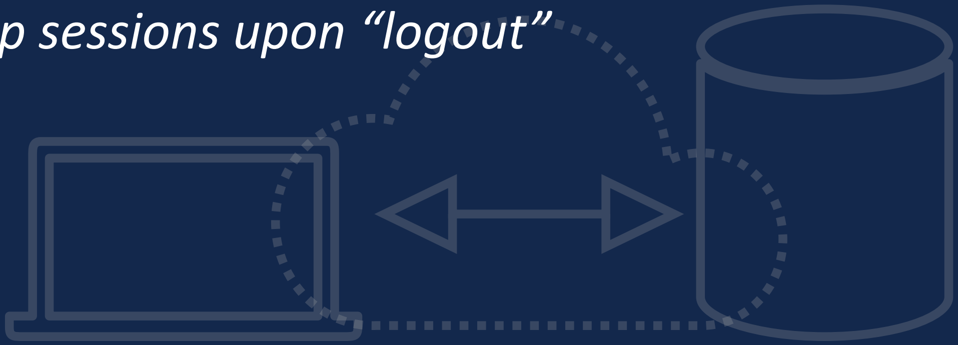
# Session Example in PHP Contd.

```
<html
<head>
<title>Session Fun</title>
</head>
<body>
<h1>Reading session variables</h1>
<?php
  session_start();
  print "There are ".$_SESSION['num_stooges'] ."
 and my favorite is ". $_SESSION['fav_stooge'];

?>
</body>
</html>
```

# Deleting Session Info from PHP

- To delete an individual session variable in PHP use `unset($_SESSION[‘thevar’]);` just as you would unset an arbitrary value in PHP
- To delete all session variables then just blow out the entire array like so `$_SESSION = array();`
- Removing all session data from the server can be accomplished with `session_destroy();`
- *Note: That to do anything with session values including destroying them you still have to call* `session_start()` *first.*
- *Just like cookies you really ought to clean up sessions upon “logout”*



# PHP Session Details

- By default sessions uses memory cookies that expire upon browser close. You can use the `session_set_cookie_params(`*`expiration`*`,’path’,’domain’,`*`secure`*`)` function to set the typical cookie values you might want to do
  - Example:

```
session_name(‘mysid’);
session_set_cookie_params(3600,’/whoismgr’,’www.xyz.com’);
session_start();
```

- Note that you do not have to manually calculate the time from EPOCH here you specify the number of seconds into the future before the cookie expires.

# Summary

- Initial scripting languages on the server side offer a trade-off in ease of development, performance and safety
- Later generations try to address these issues but significant impact ease based productivity
- PHP lives on with heavy use despite some shortcomings
- PHP demonstrates that “winning” solutions may not be the most elegant
  - Represents an cultural tension between “legitimacy and efficiency” that effects uptake of tech
- **Generally I think that in the life cycle of web dev PHP still provides a very good choice to get going quickly especially for mundane tasks like form handling, CRUD apps, and basic DB driven sites.**