Outline

- HTML Basics
- PHP Basics
- Programming in the large

**About HTML**

- HyperText Markup Language
  - HyperText = contains clickable links
  - Markup Language = structuring and rendering instructions
- Used for most of the internet
  - Static HTML = simple file retrieved from server’s disk, or generated by server-side script, e.g., PHP
  - Dynamic HTML = HTML containing client-side script

**Related Languages**

- Markup languages: troff, LaTeX, texinfo
  - 1991, Tim Berners-Lee published description of HTML as application of SGML
  - HTTP = HyperText Transfer Protocol
  - Standards vs. Browser implementations
  - 1996, XML = eXtensible Markup Language
    - Simpler variant of SGML
    - Used to define markup languages, such as HTML
  - 2000, XHTML = HTML that conforms to XML

**How to Write + Run Code**

- See CIMS instructions: 
  http://www.cims.nyu.edu/systems/userservices/webhosting/
- Create web site:
  
  ssh access.cims.nyu.edu
  mkdir $HOME/public_html $HOME/public_html/php
  chmod 701 $HOME $HOME/public_html $HOME/public_html/php
  cd $HOME/public_html/php
  echo ‘<h1>Welcome to ’`pwd`’</h1>’ > index.html
  chmod 604 index.html
- Use a web browser to look at your web site:
  

**Structure of an HTML Document**

- Text Editor (e.g. Emacs)
- Web Browser (e.g. Firefox)
Lexical Peculiarities

- Comment: `<!-- ... -->`
- Element: `<table>`
- Start tag: `<table>`
- Attribute: `border=5`
- Contents: `<hr ...>`
- End tag: `</table>`
- Empty element: `<pre>`
- Entity: `&amp;`

HTML Documentation

- HTML Documentation:
  - [http://www.w3schools.com/html/](http://www.w3schools.com/html/)

Input and Output

```html
<html>
<head>
<title>Form test</title>
</head>
<body>
<form name="frmTest" action="script_url.php" method="get"> <!-- or: method="post" -->
  Text field: <input type="text" name="txt">
  <br/>
  Radio buttons:
  A <input type="radio" name="rdo" value="a"/>
  B <input type="radio" name="rdo" value="b"/>
  <br/>
  Check boxes:
  C <input type="checkbox" name="chk" value="c"/>
  D <input type="checkbox" name="chk" value="d"/>
  <br/>
  Submit button:
  <input type="submit" value="submit"/>
</form>
</body>
</html>
```

HyperText Transfer Protocol

- HTTP GET: parameters encoded in URL
- HTTP POST: parameters in message header

Outline

- HTML Basics
- PHP Basics
- Programming in the large

About PHP

- PHP: Hypertext Processor
  - Recursive acronym
  - HTML = HyperText Markup Language
- Claim to fame: simplicity
  - Script embedded in HTML generates HTML
  - Libraries for MySQL, Oracle, PDF, XML

## Conceptual Overview

### Derived from Source: [Server-Side Scripting (PHP)](http://www.g223033-002.nyu.edu/lectures/server-side-scripting-php.pdf)

#### Language Overview

**Related Languages**

- **PHP 5, current language version**
  - May 2004: Parser rewritten again: "Zend Engine"
  - May 2000: Engine rewritten by Zeev Suraski and Andy Gutsmans in Tel Aviv
  - June 1998: PHP/FI scripting language, Rasmus Lerdorf
  - April 1996: PHP = Personal Home Page Tools, Rasmus Lerdorf, CGI scripts written in C

**Secure Your Website!**

- **See CIMS instructions:**
  - Use a web browser, that will request authorization:
    - **HTTP GET or POST**
    - **serve HTML**
    - Run PHP, which creates HTML with result
  - **http://www.cs.nyu.edu/~your_cims_id/php/index.html**

- **Create a user name and password:**
  - Put the following in `$HOME/public_html/php/htaccess`:
    - **AuthType Basic**
    - **AuthUserFile /home/your_cims_id/.htpasswd**
    - **AuthName "Members ONLY"**
    - **require valid-user**
  - **chmod 604 $HOME/.htpasswd $HOME/public_html/php/htaccess**
  - **Use a web browser, that will request authorization:**
    - **http://www.cs.nyu.edu/~your_cims_id/php/index.html**

#### How to Write + Run Code

- **Put the following in $HOME/public_html/php/hello.php**
  - `<html><body>
  - `<php`
  - `<?php
  - `if(!empty($_GET['who'])) { echo "Hi, {$_GET['who']}."; }
  - `?>
  - `<form action="$_SERVER[PHP_SELF]" method=get>
  - `Who shall be greeted: <input type="text" name="who" />
  - `</form>
  - `<body></html>

- **Set the permissions:**
  - `chmod 604 $HOME/public_html/php/hello.php`

- **Use a web browser to look at your php script:**
  - **http://www.cs.nyu.edu/~your_cims_id/php/hello.php**

#### Concepts

##### Input and Output

- **Input:** EGPCS superglobals
  - `$ENV`, `$GET`, `$POST`, `$COOKIE`, `$SERVER`
  - `$REQUEST`: union of G,P,C
  - `$FILES`: contains uploaded files
  - `$SESSION`: persistent state across loads

- **Output:** printed text in HTML document
  - `echo`, `print()`, `printf()`
  - `var_dump()`, `print_r()`: print human-readable form for debugging; `warning`: problems with cycles
  - `phpinfo()`: prints lots of diagnostic information

##### Embedding Code in Web Pages

- **Browser doesn’t see PHP script, only result of running script on server**
- **PHP code gets replaced by its own output**
  - (printed by `echo` and other functions)
- **Four styles of embedding PHP:**
  - XML style (preferred!)
  - SGML style
  - ASP style
  - Script style

- **Variant of SGML style:**
  - `<script language="php">
  - `...
  - `</script>`

- **Self-Processing Pages**

  - **User**
  - **Browser**
  - **Server**

  - **HTTP GET**
  - **serve HTML**
  - **Run PHP, which creates HTML with form**

  - **HTTP GET or POST**
  - **same URL + data**
  - **Run PHP, which creates HTML with result**

  - **Same PHP script, different HTML response**
Lexical Peculiarities

- Embedded in HTML with `<?php ... ?>`
- Variables are case sensitive; classes, functions, and keywords are case insensitive
- All variables (including arrays) begin with dollar sign ($), can be interpolated in string
- Semicolon required even after last statement in block, optional only before `?>`
- Single-line comments, or up to `?>` //
- Multi-line comments, even across `?>` //
- Literals: `"s"`, `s`, multi-line string, `true`, `null`
- Heredocs: continues after closing tag

Type Conversions

<table>
<thead>
<tr>
<th>Value</th>
<th>bool</th>
<th>number</th>
<th>string</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>0</td>
<td>1</td>
<td>&quot;&quot;</td>
</tr>
<tr>
<td>true</td>
<td>1</td>
<td>true</td>
<td>Identity</td>
</tr>
<tr>
<td>0</td>
<td>false</td>
<td>1</td>
<td>Identity</td>
</tr>
<tr>
<td>1</td>
<td>true</td>
<td>Represent value as string</td>
<td></td>
</tr>
<tr>
<td>&quot;0&quot;</td>
<td>false</td>
<td>0</td>
<td>Identity</td>
</tr>
<tr>
<td>null</td>
<td>false</td>
<td>0</td>
<td>Numeric prefix</td>
</tr>
<tr>
<td>array</td>
<td>empty</td>
<td>Error</td>
<td>&quot;Array&quot;</td>
</tr>
<tr>
<td>object</td>
<td>empty</td>
<td>Error</td>
<td>&quot;Object&quot;</td>
</tr>
</tbody>
</table>

Variable Declarations

- Implicit: `echo $a + 1;` `$b = 5;`
  - Read NULL if non-existent
- Constant (global): `define("PI", "3.14");` `echo PI`
  - Not a variable, don't need dollar sign ($)
- Global, used locally: `global $g;` `$g = $g + 1;`
  - Otherwise, write creates new local $g
- Local, unlimited lifetime: `static $s;` `$s++;`
  - Otherwise, value forgotten after return

Operators

- `new` Create object
- `[]` Array subscript
- `++`, `--` Auto-increment / decrement
- `+=`, `-=`, `*=`, `/=` Additions, string concatenation
- `&`, `&`, , `&` Bitwise shift
- `|`, `|`, `|`, `|` Comparison
- `|`, `|`, `|`, `|` Logical negation
- `&`, `&`, `&`, `&` Logical AND
- `|`, `|`, `|`, `|` Logical OR
- `==`, `==`, `==`, `==` Identity (not all same precedence)
- `===`, `===`, `===`, `===` Identity (not all same precedence)
- `&&`, `&&`, `&&`, `&&` Logical (not all same precedence)
- `||`, `||`, `||`, `||` Logical (not all same precedence)
- `$, $, $, $` List separator

Arrays

- Creation: `$a = array(1, 2, 3);`
  - `$b = array('cat' => 'meow', 'dog' => 'woof')`
- Indexing: `e.g., $a[2], $b["dog"], $b[cat]`
  - `3` and `3` and `3` are the same key
  - Quotes around string keys are optional, and must be omitted for interpolation
  - Write to non-existent index inserts, e.g., `$a[3]=4;`
  - Write without index append, e.g., `$a[]=4;`
- Remove: `unset($a[2]), delete unset($a)`
- Multiple assign: `list($x, $y) = array(1,2,3);`
Array Library Functions

```
array_key_exists(key, search)
array_values(input)
array_walk(array, func [, data])
array_values(array)
array_push(array, var [, ,])
array_values(array)
```  

Control Statements

```
if (expr) ... elseif (expr) ... else ...
switch(expr){case expr: ... default: ...
for (expr; expr; expr) ...
foreach ($var as $val) ...
while (expr) ...
```  

Alternative Control Syntax

```
if ($x < $y):
echo "then branch";
$min = $x;
else:
echo "else branch";
$min = $y;
endif;
```

References

```
Variable variable
– Store name of one variable as string in other variable
– Also known as "soft reference"
```

```
Alias
– Make two variables refer to the same memory location
– Also known as "hard reference"
```

Writing Subroutines

- Declaration: function [6]int (arg*) { ... }
- To return a value: return expr;
- &: return alias for result (hard reference)
- Arguments: arg ::= [6]int = expr
- Call-by-value, even for arrays
- &: call-by-reference
- [= expr]: optional parameter, default value
- Empty (arg*)?: $my_array = func_get_args()
- Variable functions: same as variable variables
- Creating new variable function from strings:
  `$x=create_function('a','echo $x;'); $x('hi');`
Finding PHP Mistakes

• If script encounters error on server, the browser just gets empty HTML ⇒ not helpful!
• For compile errors: run at command line
  – php -f file  Parse and execute file
  – php -s -f file  Syntax highlighted source
  – php -l -f file  Lint (check syntax without running)
• For logic errors:
  – Use echo statements to see what gets executed
  – Use var_dump() calls to inspect data structures
• To view source in browser:
  ln -s script.php script.phps

Structure of a PHP Application

• Literal inclusion of code from file:
  require 'fileName';  fatal if non-existent
  include 'fileName';  warn if non-existent
  require_once 'fileName';  no effect if repeated
  include_once 'fileName';  no effect if repeated
• Use @include to suppress the warning
• Convention: fileName.extension .inc
• No separate scope / namespace for included code, may cause proliferation of globals

Using Objects

require_once 'Apple.inc';
$s1 = new Apple(150, "green");
$s2 = new Apple(150, "green");
$s1->color = "red";
$s2->weight = 220;
echo $s1->prepare("slice") . "<br/>
";
echo $s2->prepare("squeeze") . "<br/>
";

Defining Classes

class Fruit {
  var $weight = 0;
  function __construct($weight) {
    $this->weight = $weight;
  }
  function pluck() {
    return "fruit(\"weight\")";
  }
  function prepare($how) {
    return $how . $this->pluck();
  }
}
class Apple extends Fruit {
  var $color = "green";
  function __construct($weight, $color) {
    $this->weight = $weight;
    $this->color = $color;
  }
  function pluck() {
    return $this->color . " apple";
  }
}

Inheritance in PHP

class Fruit {
  var $weight = 0;
  function __construct($weight) {
    $this->weight = $weight;
  }
  function pluck() {
    return "fruit(\"weight\")";
  }
  function prepare($how) {
    return $how . $this->pluck();
  }
}
class Apple extends Fruit {
  var $color = "green";
  function __construct($weight, $color) {
    $this->weight = $weight;
    $this->color = $color;
  }
  function pluck() {
    return $this->color . " apple";
  }
}

More on Classes

• Modifiers: method (abstract, final), property (public, private, protected, const), both (static)
• Static member access: ::, self::, parent::
• Interface: like in Java, all methods implicitly abstract;
class can extend class and implement interfaces
• Access to non-existent property turns into method call: __get($propName) or __set($propName, $value)
• Called on object death: __destruct()
• Introspection: class_exists, get_declaredClasses, get_class_methods, get_class_vars, get_parent_class, is_object, get_class_method_exists, get_object_vars
Scopes and Visibility

- Locals: scope is entire function, not just block
- Globals:
  - "global $x;" is shorthand for "$x = &GLOBALS['x'];"
  - register_globals in php.ini causes EGPSC to be spilled into globals; that's bad for security
- Nested functions:
  - Inner function does not see outer locals/arguments
  - Inner function globally visible after first call to outer
- Modules: require/include don’t affect scoping
- Classes: public, private, protected properties

PHP Documentation

- CIMS web scripting instructions: http://www.cims.nyu.edu/systems/userservices/webhosting/
- Tutorial: http://www.w3schools.com/
- PEAR = PHP Extension+Application Repository: http://pear.php.net

Evaluating PHP

Strengths
- Simplicity
- Portability
- Large libraries
- Many database bindings
- Popularity

Weaknesses
- Error handling
- Lack of scalability
  - Compared to Java
- Low-level
  - Compared to Ruby on Rails or Google Web Toolkit

Administerial

Last Slide

- Pick up graded hw01, hw02, quiz1
- Look at example solutions
- Course evaluations (you evaluate me)
- Today’s lecture
  - HTML, HTTP
  - Server-side scripting
  - PHP
- Next lecture
  - Client-side scripting
  - JavaScript
- Next+1 lecture
  - Web applications
  - Databases