CS 5142
Scripting Languages

10/02/2013

Server-Side Scripting (PHP)

Outline

- PHP Data and Control
- Programming in the large

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`

Arrays

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

Array Library Functions

http://us.php.net/manual/en/ref.array.php

`range(low, high, [, step])`	Create array
`count(var [, mode])`	Size
`array_keys(input [, …])`	Indices
`array_values(input)`	Contents
`array_key_exists(key, array)`	Contains
`sort(array [, sort_flags])`	By values
`rsort(array [, sort_flags])`	Descending
`usort(array, cmp_function)`	By user function
`array_push(array, var [, …])`	Add to end
`array_pop(array)`	Remove from end
`array_walk(array, func [, data])`	Mapping

Control Statements

Conditional	**if (***expr***)** … [**elseif (***expr***)** …] [**else** …] **switch(***expr***){case** *expr***:**… … **default:** …**}**
Fixed iteration loops	**foreach ($***arr* **as $***val***)** … **foreach ($***arr***as $***key* **=> $***val***)** …
Indefinite loops	**for (***expr***;** *expr***;** *expr***)** … **while (***expr***)** … **do** … **while (***expr***);**
Unstructured control	**break** [*expr*]**;** **#** *expr* = loop levels to skip **continue** [*expr*]**;** **#** *expr* = loop levels to skip **exit** [*expr*]**;** **#** *expr* = error message **return** [*expr*]**;** **#** *expr* = return value
Directive	**declare (***directive***)** … **#** rarely used

Alternative Control Syntax

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

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

Also available for other control statements, e.g.:

- `while(`*expr*`):` … `endwhile;`
- `for(`*expr*`;` *expr*`;` *expr*`):` … `endfor;`
- `switch(`*expr*`):case` *expr*`:`… … `default:`… `endswitch;`

References

"Variable variable"

- Store name of one variable as string in other variable
- Also known as "soft reference"

```
$x = 123;
$r = 'x';
echo $$r;
$r = '';
```

"Alias"

- Make two variables refer to the same memory location
- Also known as "hard reference"

```
$x = 123;
$r = &$x;
echo $r;
unset $r;
```

Writing Subroutines

- Declaration: **`function`** [**`&`**]*id* **`(`***arg*`*`**`)`** **`{`** ... **`}`**
  - To return a value: **`return`** *expr* **`;`**
  - **`&`**: return alias for result (hard reference)
- Arguments: *arg* ::= [**`&`**]**`$`***id* [**`=`** *expr*]
  - Call-by-value, even for arrays
  - **`&`**: call-by-reference
  - [**`=`** *expr*]: optional parameter, default value
  - Empty **`(`***arg*`*`**`)`**: **`$my_array = func_get_args()`**
- Variable functions: same as variable variables

Anonymous Functions

- Creating new variable function from strings:

```
$x=create_function(args, code);
```

- Example script:

```
<?php
function callFn($fn, $x) { $fn($x); }
function printIt($it) { print "printIt $it\n"; }
#pass reference to named subroutine; prints "printIt Hello"
callFn("printIt", "Hello");
#pass reference to anonymous subroutine; prints "lambda Hi"
callFn(create_function('$it', 'print "lambda $it\n";'), "Hi");
  ?>
```

- Useful for call-backs, e.g.,
  `usort(array, cmp_function)`

Outline

- PHP Data and Control
- Programming in the large

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';`	Include file
`$a1 = new Apple(150, "green");` `$a2 = new Apple(150, "green");`	Constructor calls

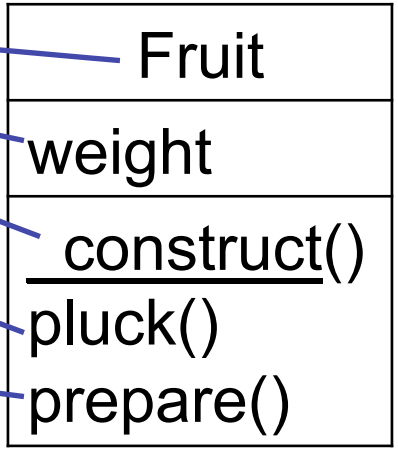
`$a2->color = "red";`
`$varvar = "weight";`
`$a2->$varvar = 220;`
	Property access

`echo $a1->prepare("slice") . " \n";` `echo $a2->prepare("squeeze") . " \n";`	Method calls

PHP 5

Defining Classes

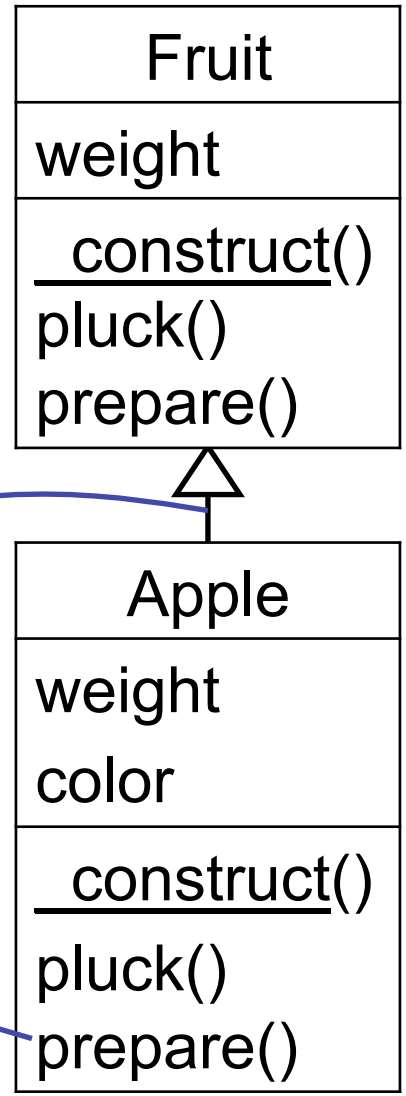
```
class Fruit {
  var $weight = 0;
  function __construct($weight) {
    $this->weight = $weight;
  }
  function pluck() {
    return "fruit(" . $this->weight . "g)";
  }
  function prepare($how) {
    return $how . "d " . $this->pluck();
  }
}
```



PHP 5

Inheritance in PHP

```
class Fruit {
  var $weight = 0;
  function __construct($weight) {
    $this->weight = $weight;
  }
  function pluck() {
    return "fruit(" . $this->weight . "g)";
  }
  function prepare($how) {
    return $how . "d " . $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: **`__destructor()`**
- Introspection: **`class_exists`**, **`get_declared_classes`**, **`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 EGPCS 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

Reference

PHP Documentation

- Language and library reference: http://www.php.net/manual/en/
- CIMS web scripting instructions: http://www.cims.nyu.edu/systems/userservices/webhosting/
- Book: Programming PHP, 2nd edition [safari]. Rasmus Lerdorf, Kevin Tatroe, and Peter MacIntyre. O'Reilly, 2006.
- Tutorial: http://www.w3schools.com/
- PEAR = PHP Extension+Application Repository: http://pear.php.net

Soap-box

Evaluating PHP

Strenghts

- 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

Administrative

Last Slide

- Today's lecture
  - Server-side scripting
  - PHP
- Next lecture
  - PHP
  - HTML5