cs3157: php lecture (mon-25-apr-2005)

- today:
  - history and background
  - your first php program
  - basics
  - writing your own functions
  - arrays
  - classes
  - I/O
  - on-line documentation
    - `http://www.php.net` (php home page)
    - `http://www.php.net/manual/en/` (online reference)

history and background

- developed in the latter 1990's
- originally created as "Personal Home Page" tools, by Rasmus Lerdorf
- at first, was a quick tool for embedding sql queries in a web page (v1.0)
- then structured code was added (v2.0), but with a buggy language parser
- official release (v3.0) fixed parser bugs - June 1998
- by Jan 1999, 100,000 web pages were using php!!!
- php is better than cgi because:
  - it runs as part of the web server process and doesn't require forking (unlike cgi)
  - it runs faster than cgi
  - it's faster to write...
- php was designed to run with apache web server on unix
  - but also runs on windows and mac
- it's free!

- php is coded in C
  - has a well-defined API
  - extensible
- the way it runs:
  - a php engine is installed as part of a web server
  - the engine runs the php script and produces html, which gets passed back to the browser

your fi rst program(s)

- `hello.php` (plain php)
- `hello2.php` (php embedded in html)
- `hello3.php` (uses `<?php` start tag)

basics

- php start and end tags: `<? ... ?>`
- also: `<?php ... ?>`
- semi-colon ends a statement (like C)
- string constants surrounded by quotes (`"`) or (`'`)
- you can embed multiple php blocks in a single html file
- variable names are preceded by dollar sign (`$`)
- user input is through html forms
- the language is case-sensitive, but calls to built-in functions are not (not sure if that's true for all built-in functions)
- identifiers are made of letters, numbers and underscore (_); and cannot begin with a number
- expressions are just like in C

data types

- integers
- floating-point numbers
- strings
- loosely typed (you don't have to declare a variable before you use it)
- conversion functions: `intval, doubleval, strval, settype`
- `settype( <value>, <newtype> )` where `newtype="integer", "double" or "string"`
- typecasting: `(integer), (string), (double), (array), (object)`

operators

- mathematical: `+, -, *, /, %, ++, --`
- relational: `<, >, <=, >=, ==, !=`
- logical: `AND, &&, OR, ||, XOR, !`
- bitwise: `&, |, ˆ (xor), ˜ (ones complement), >>, <<`
- assignment: `=,` =, -=, *=, /=,
- other:

  `.` → concatenate
  `->` → references a class method or property
  `=>` → initialize array element index

conditionals (1)

- if/elseif/else:

```
if ( <expression1> ) {
  <statement(s)>
}
elseif ( <expression2> ) {
  <statement(s)>
}
else {
  <statement(s)>
}
```

conditionals (2)

- tertiary operator:

```
<conditional-expression> ?
  <true-expression> : <false-expression>;
```

- switch:

```
switch( <root-expression> ) {
  case <case-expression>:
    <statement(s)>;
    break;
  default:
    <statement(s)>;
    break;
}
```

loops

- while

```
while ( <expression> ) {
  <statement(s)>;
}
```

- do-while

```
do {
  <statement(s)>;
} while ( <expression> );
```

- for

```
for ( <initialize> ; <continue> ; <increment> ) {
  <statement(s)>;
}
```

- break:
  - execution jumps outside innermost loop or switch

other

- `exit()` function
  - halts execution, meaning that no more code (php or html) is sent to the browser
- built-in constants
  - `PHP_VERSION`
  - `__FILE__`, `__LINE__`
  - `TRUE` = 1, `FALSE` = 0
  - `M_PI` = pi (3.1415927....)

writing your own functions

- declared just like C:

```
function <name> ( args ) {
  <body>
  [return <value>]
}
```

- called just like C
- arguments (and local variables) are local, and don’t exist when you exit the function; but you can use “static” to declare a variable so that when you call a function again, the value is retained
- use the “global” statement to declare global variables that you want to be able to access from within a function, or the `GLOBALS` array (which is like a perl hash) e.g., `GLOBALS[’username’]`
- recursion is okay, but be careful!
- example: `colors.php`

arrays

- indexed using `[...]`
- indeces can be integers or strings (like a perl hash)
- when strings are indeces, it's called an "associative array"
- `array()` function can be used to initialize an array
- e.g., `$var = array( value0, value1, value2, ... );`
- use the `=>` operator to define the index:

```
$var = array( 1=>value1, value2, ... );
$var = array( "a"=>value1, "b"=>value2, ... );
```

- multidimensional arrays are okay (like C)
- example: `arrays.php`

classes

- defining a class:

```
class <class-name> {
  // declare properties
  // declare methods
}
```

- use just like java and c++
- example: `myclass.php` and `userclass.php`
- note use of `include` statement

I/O

- get input from html forms using

```
$_POST['<name>']
$_GET['<name>']
$_REQUEST['<name>']
```

- file I/O
  - basically just like C:

```
$fp = fopen( "filename","w" );
fwrite( $fp,"stuff" );
fclose( $fp );
```

  - note that `fopen` second argument mode is like C)