Server programming: PHP

Lecture 9

Questions

1. What software is interpreting and executing JavaScript programs which we wrote?

2. Can we get data from server using client-side JavaScript?

3. Can we store data on server using only client-side JavaScript?

4. Are HTML pages standalone desktop applications? What is the main usage of HTML pages?

5. How can we pass data between several HTML pages?

6. Can we embed fonts into HTML page? Can we embed images?

Server pages

- The web server in this context is a software application that helps to deliver web pages on the request to clients using the Hypertext Transfer Protocol (HTTP).

- Server-side pages are **programs** written using one of many web programming languages/frameworks: PHP, Java/JSP, Ruby on Rails, ASP.NET, Python, Perl

- When client makes a request, the corresponding program is executed and its output is sent to the client as a file (mostly, HTML)

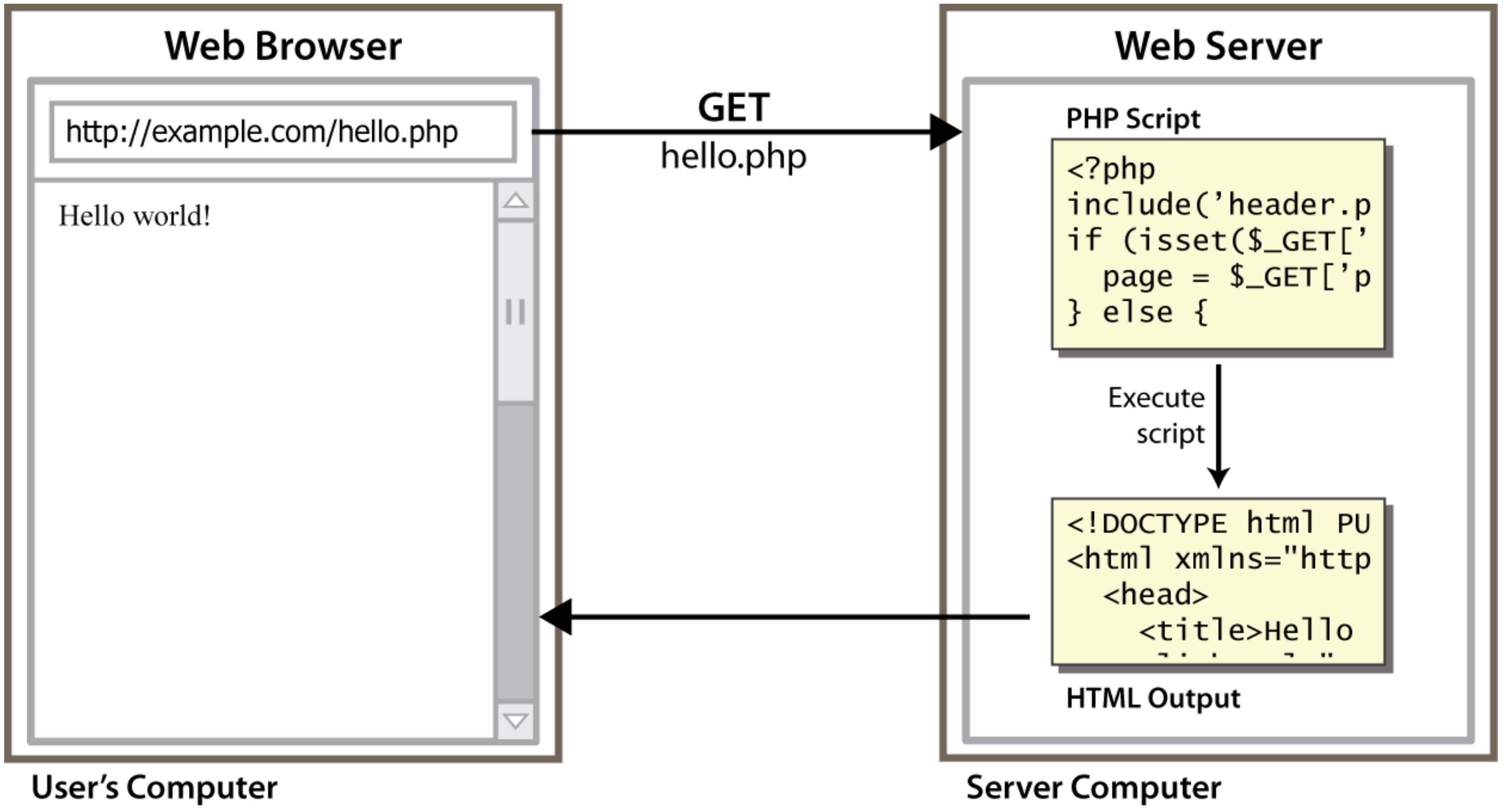
What is PHP

- PHP stands for "PHP Hypertext Preprocessor" - a server-side scripting language used to make web pages dynamic
- PHP allows you to manipulate web page content ***on the server*** just before a page is delivered to the client browser.
  - A PHP script runs on the server and can alter or generate HTML code at will.
  - An HTML web page is still delivered to the browser, which doesn’t know or care that PHP is involved in tweaking the HTML on the server.

PHP is the easiest among server pages

- free and open source: anyone can run a PHP-enabled server free of charge
- **compatible:** supported by most popular web servers
- **simple:** lots of built-in functionality; familiar syntax
- **available:** installed on most commercial web hosts
- **well-documented:** type php.net/*functionName* in browser Address bar to get docs for any function

Delivering PHP pages



```
<?php
include('header.p
if (isset($_GET['
  page = $_GET['p
} else {
```

```
<!DOCTYPE html PU
<html xmlns="http
  <head>
    <title>Hello
```

A new process is started on the server with each request, and this process lives several milliseconds, until the script has finished its execution

```php
<?php  print "Hello, world!";  ?>
```

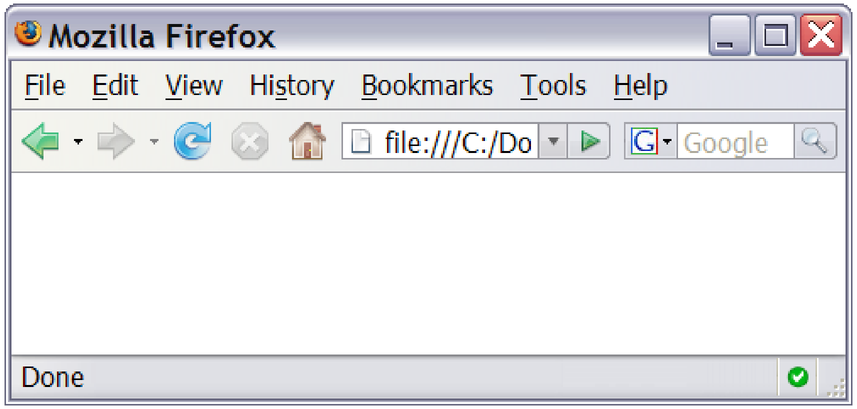
```
Hello, world!
```

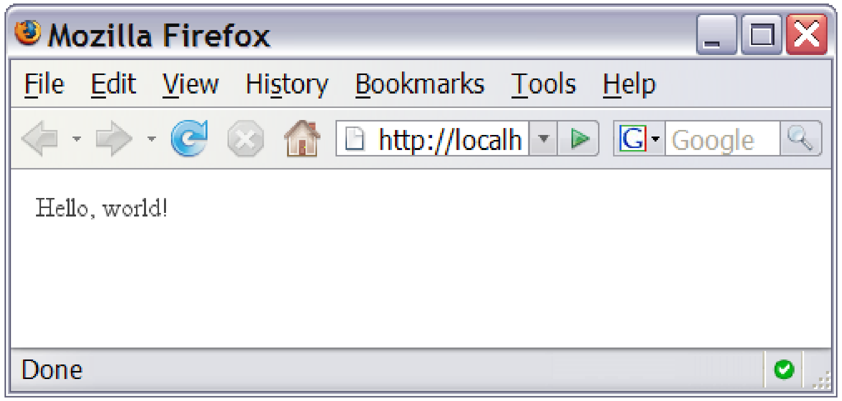
Output HTML page

- a block or file of PHP code begins with **<?php** and ends with **?>**
- PHP statements, function declarations, etc. appear between these endpoints

Test PHP page?

- you can't test your .php page by opening it from your local hard drive; you'll either see nothing or see the PHP source code
- **Web browsers know nothing about PHP and, therefore, have no ability to run PHP scripts.**
- if you upload the file to a PHP-enabled web server, requesting this .php file will run the program and send you back its output





Hello, world!

Is PHP installed in the lab?

1. Initial setup - these steps only need to be done once:
 Open terminal and set up public_html directory

```
$ umask 077
$ cd
$ chmod go+x .
$ mkdir public_html
$ chmod 755 public_html
```

2. Create file **test.php** with the following content and save it in public_html:

```
<?php
        phpinfo();
?>
```

3. You can now access **test.php** from the browser in the lab with the following url:

http://cscidb.csci.viu.ca/~USERNAME/test.php

How to test your pages at home using lab server

- In order to be able to see php pages in your home browser, you need to open (and keep open) ssh tunnel, by issuing the following forwarding command on a unix machine:

```
ssh -L 2222:cscidb.csci.viu.ca:80 username@otter.csci.viu.ca
```

where 2222 is an unused port on the local Linux system.

- Then you should be able to access your web page with the following URL:
  http://localhost:2222/~username/pagename.php

- So long as the SSH connection is alive, URLs starting with http://localhost:2222/ on the local Linux host will get routed to the web server on cscidb.csci.viu.ca.

Configuring SSH tunnel for Windows machine

- Install PuTTY SSH client from http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html

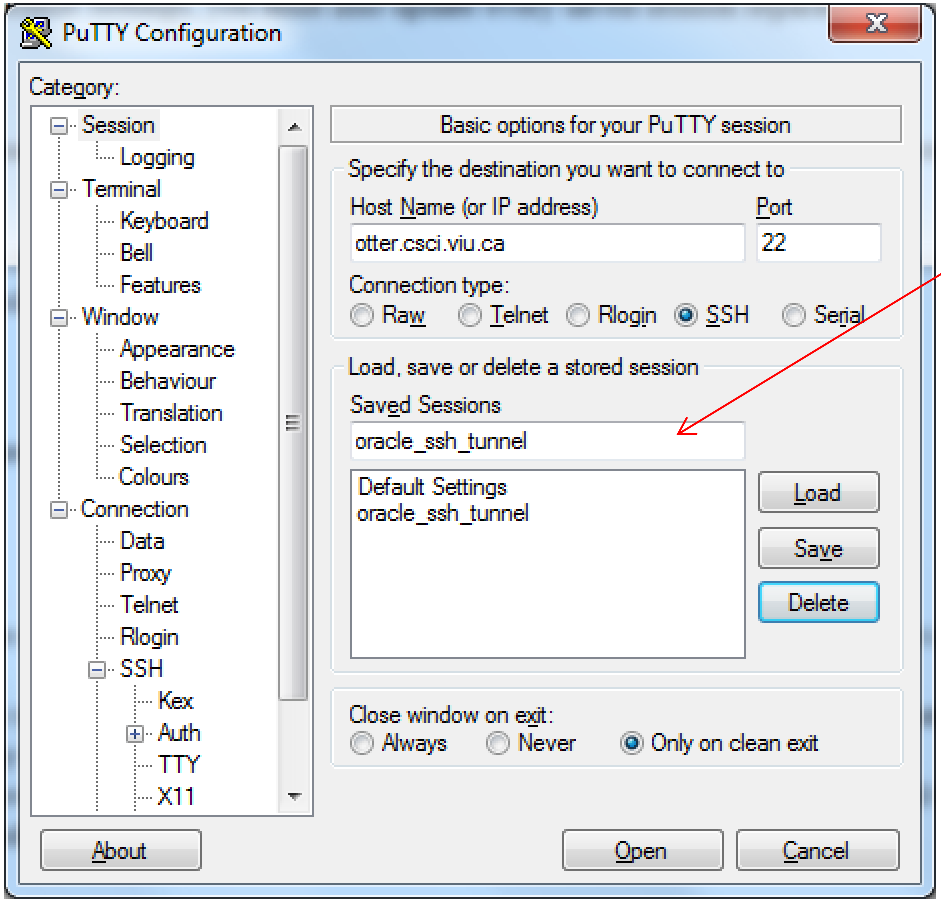
- In the "PuTTY Configuration" window:
  1. Select "Session" on the left hand side and in the "Host Name (or IP Address)" field type: otter.csci.viu.ca (port: 22). In the "Connection type" field select "SSH".
  2. In the left hand side under "Connection" select the little "+" sign to the left of "SSH" to expand the list of ssh options. Click on "Tunnels". On the right hand side just below "Add forwarded port:"

     In the "Source port" field put: 2222
     In the "Destination" field put: cscidb.csci.viu.ca:80
     Click "Add"

     Click "Open". Enter your username, then password.
  3. You can now put php files into your public_html and run them from browser as in the previous slide

- When you logout from otter, the connection forwarding will no longer work.

Saving configured SSH tunnel in PuTTY



Specify the destination you want to connect to

Host Name (or IP address): otter.csci.viu.ca

Port: 22

Connection type: Raw Telnet Rlogin [✓ SSH] Serial

Load, save or delete a stored session

Saved Sessions: oracle_ssh_tunnel

Default Settings
oracle_ssh_tunnel

Close window on exit: Always Never [✓ Only on clean exit]

- Set all configurations
- Type the name for the stored session in Saved Sessions input field
- Press Save
- Next time, select saved session from the list and press Load, then Open

PHP BASIC SYNTAX

Printing text onto the page

code

```
<?php
print "text";
print "Hello, World!\n";
print "Escape \"chars\" are the SAME as in Java!\n";
print "You can have line breaks in a string.";
print 'A string can use "single-quotes". It\'s cool!';
?>
```

output html

```
Hello, World! Escape "chars" are the SAME as in Java! You can have line breaks in a string. A string can use "single-quotes". It's cool!
```

some PHP programmers use the equivalent ***echo*** instead of ***print***

Arithmetic operators

- + - * / %
- . ++ --
- = += -= *= /= %= .=
- many operators auto-convert types: 5 + "7" is 12

Variables

```
$name = expression;

$user_name = "PinkHeartLuvr78";
$age = 16;
$drinking_age = $age + 5;
$this_class_rocks = TRUE;
```

- names are case sensitive; separate multiple words with _
- names always begin with $, on both declaration and usage
- implicitly declared by assignment (type is not written; a "loosely typed" language)

Types

- basic types: int, float, boolean, string, array, object, NULL
  - test what type a variable is with is_*type* functions, e.g. is_string
  - gettype function returns a variable's type as a string (not often needed)
- PHP converts between types automatically in many cases:
  - string → int auto-conversion on + ("1" + 1 == 2)
  - int → float auto-conversion on / (3 / 2 == 1.5)
- type-cast with (*type*):
  - $age = *(int)* "21";

int and float types

```
$a = 7 / 2;               # float: 3.5
$b = (int) $a;            # int: 3
$c = round($a);           # float: 4.0
$d = "123";               # string: "123"
$e = (int) $d;            # int: 123
```

- int for integers and float for reals
- division between two int values can produce a float

String type

```
$favorite_food = "Ethiopian";
print $favorite_food[2];                # h
```

- zero-based indexing using bracket notation
- string concatenation operator is . (period), not +
  - 5 + "2 turtle doves" produces 7
  - 5 . "2 turtle doves" produces "52 turtle doves"
- can be specified with "" or ''

Interpreted strings

```
$age = 16;
~~print "You are " . $age . " years old.\n";~~
print "You are $age years old.\n";
# Prints: You are 16 years old.
```

- strings inside " " are interpreted
  - variables that appear inside them will have their values inserted into the string
- strings inside ' ' are *not* interpreted:

```
print 'You are $age years old.\n';
# Prints: You are $age years old.\n
```

Interpreted strings

- strings inside " " are interpreted
  - variables that appear inside them will have their values inserted into the string
- if necessary to avoid ambiguity, can enclose variable in {}:

```
print "Today is your $ageth birthday.\n";
# $ageth not found
print "Today is your {$age}th birthday.\n";
```

String functions

```
$name = "Stefanie Hatcher";
$length = strlen($name); # 16
$cmp = strcmp($name, "Brian Le"); # > 0
$index = strpos($name, "e"); # 2
$first = substr($name, 9, 5); # "Hatch"
$name = strtoupper($name); # "STEFANIE HATCHER"
```

Splitting/joining (tokenizing) strings

```
$array = explode(delimiter, string);
$string = implode(delimiter, array);
$s  = "CSCI 311 M";
$a  = explode(" ", $s);     # ("CSCI", “311", "M")
$s2 = implode("...", $a);   # " CSCI...311..M“
```

- explode and implode convert between strings and arrays

bool (Boolean) type

```
$feels_like_summer = FALSE;
$php_is_rad = TRUE;

$student_count = 217;
$nonzero = (bool) $student_count;     # TRUE
```

- the following values are considered to be FALSE (all others are TRUE):
  - 0 and 0.0
  - "", "0", and NULL (includes unset variables)
  - arrays with 0 elements
- can cast to boolean using (bool)
- FALSE prints as an empty string (no output); TRUE prints as a 1

NULL

```
$name = "Victoria";
$name = NULL;
if (isset($name))
{
        print "This line isn't going to be reached.\n";
}
```

- a variable is NULL if
  - it has not been set to any value (undefined variables)
  - it has been assigned the constant NULL
  - it has been deleted using the unset function
- can test if a variable is NULL using the isset function
- NULL prints as an empty string (no output)

Comments

```
# single-line comment
// single-line comment
/* multi-line
comment */
```

- like Java, but # is also allowed a lot of PHP code uses # comments instead of //

Arrays

```
name = array();                          # create
$name = array(value0, value1, ..., valueN);

$name[index]                             # get element value
$name[index] = value;                    # set element value
$name[] = value;                         # append
```

Array examples

```
$a = array();     # empty array (length 0)
$a[0] = 23;       # stores 23 at index 0 (length 1)
$a2 = array("some", "strings", "in", "an", "array");
$a2[] = "Ooh!";   # add string to end (at index 5)
```

- to append, use bracket notation without specifying an index
- element type is not specified; can mix types
- the array in PHP replaces many other collections in Java list, stack, queue, set, map, ..

Array functions

```
$letters = array("A", "B", "C", "D", "E");
for ($i = 0; $i < count($letters); $i++) {
  $letters[$i] = strtolower($tas[$i]);
}                                          # ("a", "b", "c", "d", "e")
$first = array_shift($letters);            # ("b", "c", "d", "e")
array_pop($letters);                       # ("b", "c", "d")
array_push($letters, "m");                # ("b", "c", "d", "m")
array_reverse($letters);                 # ("m", " b", "c", "d")
sort($letters);                        # (" b", "c", "d ", "m")
$best = array_slice($letters, 1, 2);  # ("c", "d")
```

Sorting an array of objects: usort

```
function cmp($a, $b)
{
    return strcmp($a["fruit"], $b["fruit"]);
}

$fruits[0]["fruit"] = "lemons";
$fruits[1]["fruit"] = "apples";
$fruits[2]["fruit"] = "grapes";

usort($fruits, "cmp");
```

Iterating an array: The *foreach* loop

```
foreach ($array as $variableName) {
  ...
}
```

```
$stooges = array("Larry", "Moe", "Curly", "Shemp");
for ($i = 0; $i < count($stooges); $i++) {
  print "Moe slaps {$stooges[$i]}\n";
}
foreach ($stooges as $stooge) {
  print "Moe slaps $stooge\n";  # even himself!
}
```

- a convenient way to loop over each element of an array without indexes

for loop

```
for (initialization; condition; update) {
        statements;
}


for ($i = 0; $i < 10; $i++) {
        print "$i squared is " . $i * $i . ".\n";
}
```

if/elseif

```
if (condition) {
        statements;
}
elseif (condition) {
        statements;
}
else {
        statements;
}
```

NOTE: although *elseif* keyword is much more common, *else if* is also supported

PHP EMBEDDED INTO HTML PAGE

Printing HTML tags in PHP = bad style

```
<?php
print "<!DOCTYPE html >\n";
print "<html>\n";
print "  <head>\n";
print "    <title>My web page</title>\n";
...
for ($i = 1; $i <= 10; $i++) {
  print "<p> I can count to $i! </p>\n";
}
?>
```

PHP expression blocks: shortcut to insert dynamic values

<?= *expression* ?>

<h2> The answer is <?= *6 * 7* ?> </h2>

```
The answer is 42
```

- PHP expression block: evaluates and embeds an expression's value into HTML

<?= *expr* ?>   is equivalent to

<?php print *expr*; ?>

Embedded PHP

```
HTML content

 <?php
   PHP code
 ?>

HTML content

 <?php
   PHP code
 ?>

HTML content ...
```

- any contents of a .php file between <?php and ?> are executed as PHP code
- all other contents are output as pure HTML

Expression blocks example

```
<body>
  <?php for ($i = 99; $i >= 1; $i--) { ?>
    <p> <?= $i ?> bottles of beer on the wall, <br />
      <?= $i ?> bottles of beer. <br />
      Take one down, pass it around, <br />
      <?= $i - 1 ?> bottles of beer on the wall. </p>
  <?php } ?>
 </body>
```

Common errors: unclosed braces, missing = sign

```
<body>
        <p>Watch how high I can count:
        <?php for ($i = 1; $i <= 10; $i++) { ?>
                <? $i ?>
        </p>
</body>
</html>
```

- </body> and </html> above are inside the for loop, which is never closed
- if you forget to close your braces, you'll see an error about 'unexpected $end'
- if you forget = in <?=, the expression does not produce any output

Expression blocks can go inside HTML tags and attributes

```
<body>
  <?php for ($i = 1; $i <= 3; $i++) { ?>
    <h<?= $i ?>>This is a level <?= $i ?>
                                    heading.</h<?= $i ?>>
  <?php } ?>
 </body>
```

This is a level 1 heading.

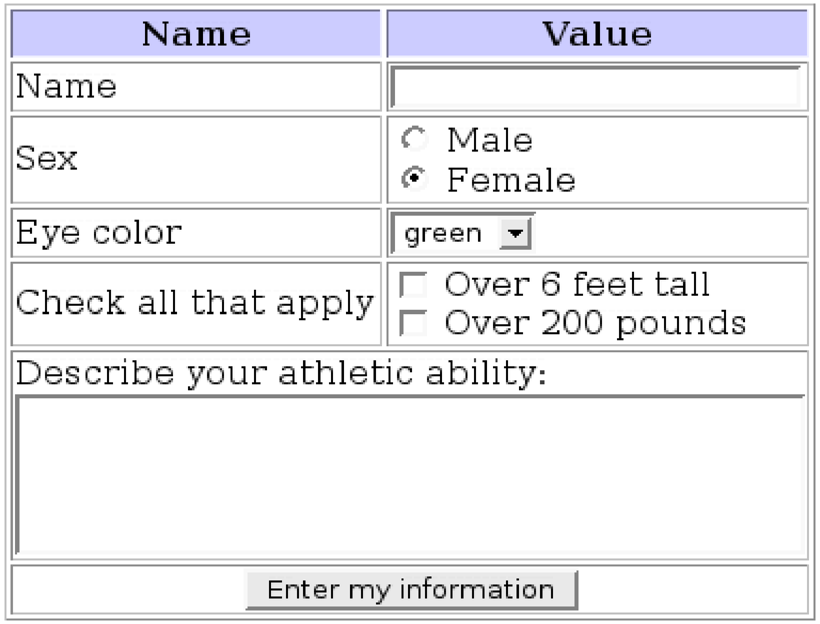
This is a level 2 heading.

This is a level 3 heading.

PHP AND HTML FORMS

HTML forms

- An **HTML form** on a web page allows a user to enter data that is sent to a server for processing.
- Form is a group of UI controls that accepts information from the user and sends the information to a web server
- The information is sent to the server as a query string attached to HTTP GET or POST request



Name	Value
Name	
Sex	○ Male ● Female
Eye color	green ▾
Check all that apply	☐ Over 6 feet tall ☐ Over 200 pounds
Describe your athletic ability:	
Enter my information	

HTTP GET vs. POST requests

- GET : asks a server for a page or data

  if the request has parameters, they are sent in the URL as a query string
- POST : submits data to a web server and retrieves the server's response

  if the request has parameters, they are embedded in the request's HTTP packet, not the URL
- For submitting data, a POST request is more appropriate than a GET
  - GET requests embed their parameters in their URLs
  - URLs are limited in length (~ 1024 characters)
  - URLs cannot contain special characters without encoding
  - private data in a URL can be seen or modified by users

Form elements: input

- **input** element is used to create many UI controls
  - an inline element that MUST be self-closed
- name attribute specifies name of query parameter to pass to server
- type can be button, checkbox, file, hidden, password, radio, reset, submit, text, ...
- value attribute specifies control's initial text

Checkboxes

```
<input type="checkbox" name="lettuce" /> Lettuce
<input type="checkbox" name="tomato" checked="checked" />
Tomato
<input type="checkbox" name="pickles" checked="checked" />
Pickles
```

- none, 1, or many checkboxes can be checked at same time
- when sent to server, any checked boxes will be sent with value on:
- http://viu.ca/params.php?tomato=on&pickles=on
- use checked="checked" attribute in HTML to initially check the box

Radio buttons

```
<input type="radio" name="cc" value="visa"
checked="checked" /> Visa
<input type="radio" name="cc" value="mastercard" />
MasterCard
<input type="radio" name="cc" value="amex" /> American
Express
```

- grouped by name attribute (only one can be checked at a time)
- must specify a value for each one or else it will be sent as value on

Dropdown list

```
<select name="favoritecharacter">
  <option>Jerry</option>
  <option>George</option>
  <option selected="selected">Kramer</option>
  <option>Elaine</option>
</select>
```

- option element represents each choice
- optional selected attribute sets which one is initially chosen

http://viu.ca/params.php?favoritecharacter%5B%5D=Kramer&favoritecharacter%5B%5D=Newman

Multiple selections

```
<select name="favoritecharacter[]" size="3"
multiple="multiple">
  <option>Jerry</option>
  <option>George</option>
  <option selected="selected">Kramer</option>
  <option>Elaine</option>
  <option selected="selected">Newman</option>
</select>
```

- optional multiple attribute allows selecting multiple items with shift- or ctrl-click
- must declare parameter's name with [] if you allow multiple selections

http://viu.ca/params.php?favoritecharacter%5B%5D=Kramer&favoritecharacter%5B%5D=Newman

%20 is space
%5B is '['
%5D is ']'
This is called percent encoding and is used in encoding special characters in the url parameter values

URL-encoding

- certain characters are not allowed in URL query parameters:

  examples: " ", "/", "=", "&"
- when passing a parameter, it is URL-encoded (reference table)

  "Joe's cool!?" → "Joe%27s+cool%3F%21"
- you don't usually need to worry about this:
- the browser automatically encodes parameters before sending them
- the PHP $_GET and $_POST arrays automatically decode them

Full refresh of the form fields

- I changed the form's HTML code ... but when I refresh, the page doesn't update!
  - By default, when you refresh a page, it leaves the previous values in all form controls
  - it does this in case you were filling out a long form and needed to refresh/return to it
  - if you want it to clear out all UI controls' state and values, you must do a full refresh
    - Firefox: Shift-Ctrl-R
    - Mac: Shift-Command-R

FILE OPERATIONS

Reading/writing files

contents of foo.txt	file("foo.txt")	file_get_contents("foo.txt")
Hello how r u? I'm fine	array("Hello\n", # 0 "how r u?\n", # 1 "\n", # 2 "I'm fine\n" # 3)	"Hello\n how r u?\n # a single \n # string I'm fine\n"

- file function returns lines of a file as an array (\n at end of each)
- file_get_contents returns entire contents of a file as a single string
- file_put_contents writes a string into a file, replacing its old contents

    if the file doesn't exist, it will be created

    file_put_contents can be called with an optional third parameter to append (add to the end) rather than overwrite

Example: display lines of file as a bulleted list

```
<ul>
<?php
        $lines = file("todolist.txt");
        foreach ($lines as $line) {
                print "<li>$line</li>\n";
        }
?>
</ul>
```

- *file* returns the lines of a file as an array of strings
- each ends with \n ; to strip it, use an optional second parameter:
  $lines = file("todolist.txt", *FILE_IGNORE_NEW_LINES*);

Unpacking an array into a set of variables: *list*

```
list($var1, ..., $varN) = array;
```

personal.txt

```
John Doe
(111) 685-2181
333-867-326
```

```
list($name, $phone, $sin) = file("personal.txt");
```

- the odd *list* function "unpacks" an array into a set of variables you declare
- when you know a file's exact length/format, use file and list to unpack it

Associative arrays

```
$blackbook = array();
$blackbook["lisa"] = "111-685-2181";
$blackbook["bart"] = "111-685-9138";
...
```

- print "Lisa's number is " . $blackbook["lisa"] . ".\n";
- associative array (a.k.a. map, dictionary, hash table) :
- uses non-integer indexes associates a particular index "key" with a value key
- "lisa" maps to value "111-685-2181“
- print_r function displays all keys/values in the array

foreach loop and associative arrays

```
foreach ($blackbook as $key => $value) {
  print "$key's phone number is $value\n";
}
```

- both the key and the value are given a variable name
- the elements will be processed in the order they were added to the array

So what is PHP

- PHP is just a scripting language add-on (mod_php) which is installed on top of a Web server software
- Web server software (such as Apache2) is able to serve pages and route user requests (HTTP requests and responses)
- PHP scripts are executed before serving pages, and after getting data from the user
- PHP can access files and databases **on server**, and dynamically generate a static content which is sent back to the client browser

Simple Web architecture with PHP

- PHP is a part of a simple and scalable architecture: you can just add more servers to handle an increasing load
- Processing of each new request is performed in a separate process
- Each PHP program runs and terminates in less 30 ms
- These programs cannot share data between users in a form other than writing to and reading from a persistent storage

PHP is of a little use for real-time interaction between multiple users

- Sharing data only through a slow persistent layer
- Server can’t immediately notify one user of something another user does.

Pull and push: two sides of a real-time interaction

- The act of periodically requesting information from the server is called ***polling*** because the client periodically polls the server for new information.
- We can do AJAX polling or full-page (normal http request) polling for the information from a server
- However, with PHP the server has no capability to reverse the equation and instantly notify the client when something happens (**pushing** data back to client).

How about on-line multi-player games with PHP?

- The most basic type of a multiplayer game you can build is the one in which the client does full-page or AJAX requests to update itself.
- PHP-enhanced web server is not a great architecture for games that need to have a lot of direct interactions between different players

Multi-player games with PHP?

- The best types of multiplayer games for polling only: when the actions of each player are mostly autonomous: players play primarily by themselves, but a central server keeps players honest by running a lot of the game logic on the server. (Example: *Mob Wars*).
- We need to enable the server to initiate requests, which is impossible to do with PHP

Conclusion

- You can use PHP and standard HTTP architecture to build multiplayer social games.
- There are limits to what type of game you can build without various hacks if you want the server to push data to the client.

REITERATION ON WEB SERVICES

Web services

web service: software functionality that can be invoked through the internet using common protocols

- like a remote function(s) you can call by contacting a program on a web server
- many web services accept parameters and produce results
- service's output can be text, HTML, XML, or other content types

RESTful web services

- **Re**presentational **S**tate **T**ransfer is intended to evoke an image of how a well-designed Web application behaves:

  presented with a network of Web pages (a virtual state-machine), the user progresses through an application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for his use.
- **REST** is initially described in the context of HTTP.
- RESTful applications maximize the use of the existing, well-defined HTTP interface, its built-in capabilities, and minimize the addition of new application-specific features on top of it.

Central principles of REST

- An important concept in REST is the existence of *resources* (sources of specific information), each of which is referenced with a global identifier (e.g., a URI in HTTP).
- In order to manipulate these resources, *components* of the network (user agents and origin servers) communicate via a standardized interface (e.g., HTTP) and exchange *representations* of these resources (the actual documents conveying the information).

REST vs. SOAP

- In addition to URIs, Internet media types, request and response codes HTTP has a rich vocabulary of operations:

  GET POST PUT DELETE etc.

- REST uses these operations and existing features of the well-defined HTTP protocol.
- SOAP RPC encourages each application designer to define new, application specific operations that supplant HTTP operations:

  getUsers()

  getNewUsersSince(date SinceDate)

  savePurchaseOrder(string CustomerID, string PurchaseOrderID) etc.

- This additive, "re-invention of the wheel" vocabulary — defined on the spot and subject to individual judgment or preference — disregards many of HTTP's existing capabilities, such as authentication, caching, and content-type negotiation.
- The advantage of SOAP over REST comes from this same limitation: Since it does not take advantage of HTTP conventions, SOAP works equally well over raw TCP, named pipes, message queues etc.

Web services with PHP

- As we already know, RESTful web services can be written in PHP and contacted by the browser through Ajax or JSONP

- The content provided by the service can be of many different types

Setting response content type with *header*

```
header("Content-type: type/subtype");
```

```
header("Content-type: text/plain");
print("This output will appear as plain text now!\n");
```

- by default, a PHP script's output is assumed to be HTML
- use the *header* function to specify non-HTML output
- must appear before any other output generated by the script

Content ("MIME") types

MIME type	related file extension
text/plain	.txt
text/html	.html, .htm, ...
text/css	.css
text/javascript	.js
text/xml	.xml
image/gif	.gif
image/jpeg	.jpg, .jpeg
video/quicktime	.mov
application/octet-stream	.exe

Example: Exponent web service

- Write a web service that accepts a base and exponent and outputs base raised to the exponent power.
- For example, the following query should output 81 :

http://example.com/exponent.php?base=3&exponent=4

- Solution:

```
header("Content-type: text/plain");
$base = $_GET["base"];
$exp = $_GET["exponent"];
$result = pow($base, $exp);
print $result;
```

Example: book sales web service (from JSONP lecture)

```
<?php
$filename = "../saleswithdates.json";
$data = file_get_contents($filename);// json string

if(array_key_exists('callback', $_GET)){

    header('Content-Type: text/javascript; charset=utf8');

    $callback = $_GET['callback'];
    echo $callback.'('.$data.');';

}else{
    // normal JSON string
    header('Content-Type: application/json; charset=utf8');
    echo $data;
}
?>
```