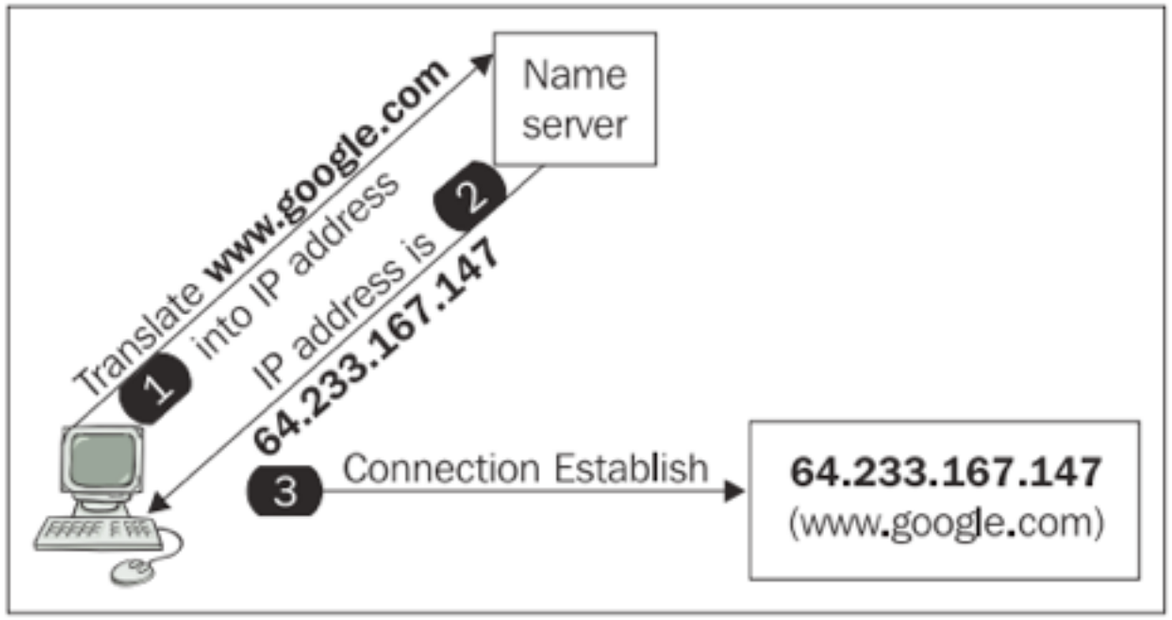
Lecture 2 Overview

- Day1/Lab1 Questions
- Review Student Videos
- Todays Topic (**Forms**)
- Lecture

Todays Topics

- Some code hints
- Forms
- Client side validation
- Security
- Firebug hack
- Start your editors

DNS Lookup



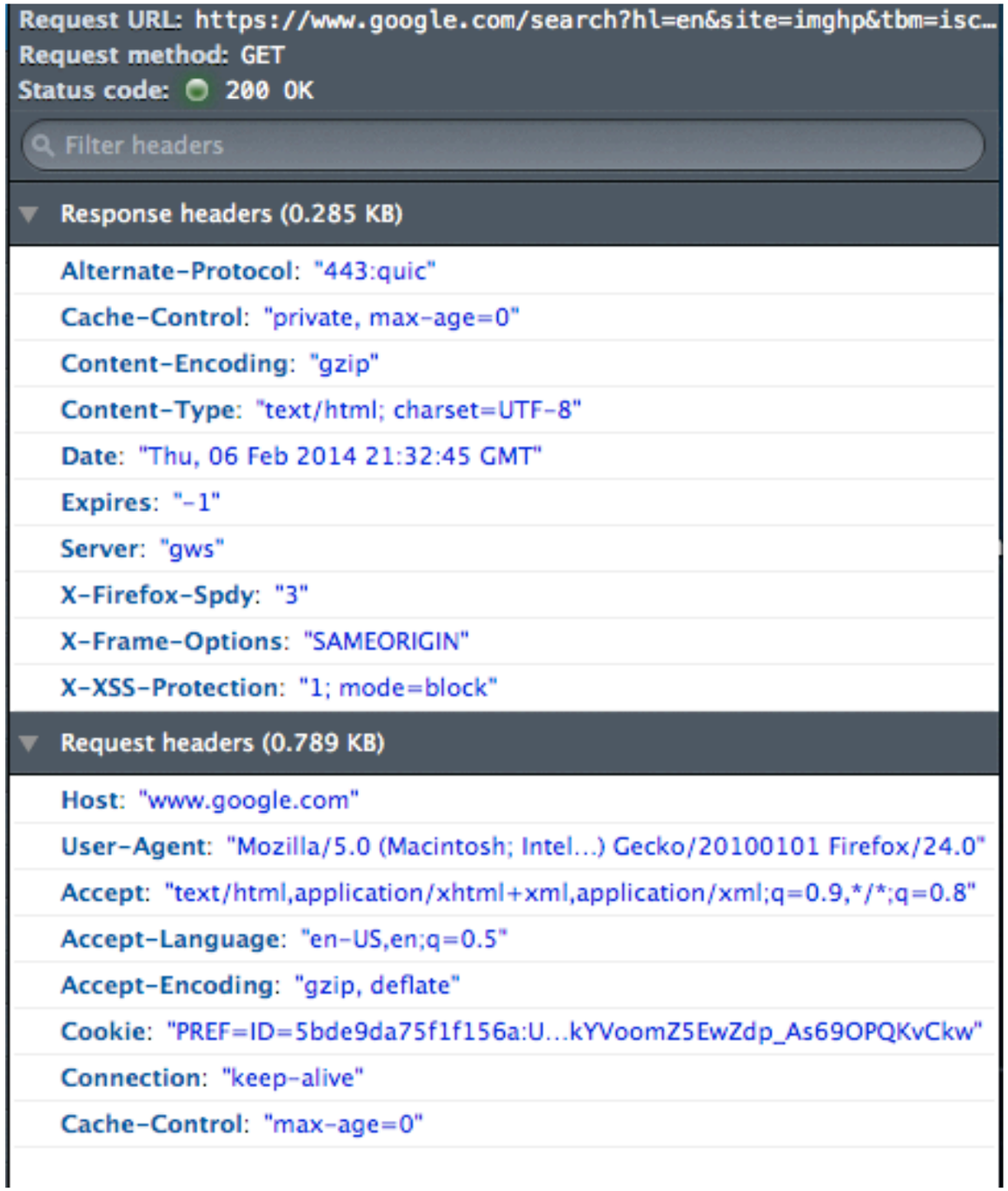
Source: http://www.tech-juice.org/2011/06/22/the-dns-protocol-explained/

Apache Config

```
<VirtualHost *:80>
ServerName www.domain.tld
ServerAlias domain.tld *.domain.tld
DocumentRoot /www/domain
</VirtualHost>

<VirtualHost *:80>
ServerName www.otherdomain.tld
DocumentRoot /www/otherdomain
</VirtualHost>
```

Host Headers



Router/Firewall
Public IP: 50.60.12.14
Internal LAN IP:
192.168.1.1

Web Host
IP: 192.168.1.5
Apache: Port 80

Use Network Address Translation (NAT) to forward requests from
Public IP 50.60.12.14 port 80 to
internal IP 192.168.1.5 port 80.

Security: Remember that any port opened up through firewall exposes a part of your Internal network to outside traffic. Best Practices?
pfsense.org - Open Source Firewall

PHP Comparison Operators

http://us1.php.net/manual/en/language.operators.comparison.php

Use to compare variables - strings, integers, boolean, etc.
If you compare a number with a string or the comparison involves numerical strings, then each string is converted to a number and the comparison performed numerically.

Example	Name	Result
$a == $b	Equal	TRUE if *$a* is equal to *$b* after type juggling.
$a === $b	Identical	TRUE if *$a* is equal to *$b*, and they are of the same type.
$a != $b	Not equal	TRUE if *$a* is not equal to *$b* after type juggling.
$a <> $b	Not equal	TRUE if *$a* is not equal to *$b* after type juggling.
$a !== $b	Not identical	TRUE if *$a* is not equal to *$b*, or they are not of the same type.
$a < $b	Less than	TRUE if *$a* is strictly less than *$b*.
$a > $b	Greater than	TRUE if *$a* is strictly greater than *$b*.
$a <= $b	Less than or equal to	TRUE if *$a* is less than or equal to *$b*.
$a >= $b	Greater than or equal to	TRUE if *$a* is greater than or equal to *$b*.

PHP Control Structures

http://us1.php.net/manual/en/control-structures.elseif.php

If/else/elseif

```
<?php
if ($a > $b) {
  echo "a is greater than b";
} else {
  echo "a is NOT greater than b";
}
?>
```

elseif, as its name suggests, is a combination of if and else. Like else, it extends an if statement to execute a different statement in case the original if expression evaluates to FALSE. However, unlike else, it will execute that alternative expression only if the elseif conditional expression evaluates to TRUE.

```
<?php
if ($a > $b) {
    echo "a is bigger than b";
} elseif ($a == $b) {
    echo "a is equal to b";
} else {
    echo "a is smaller than b";
}
?>
```

Exercise: Conditionals

Assign letter grades based on points earned. Using if/else/elseif statements, create a function that returns a letter grade based on the following point breakdowns:
100-90=A
80-89=B
79-70=C
69-60=D
<60=F

To test your function, try it with these 5 point values and echo the result back out:
1. 94
2. 54
3. 89.9
4. 60.01
5. 102.1

Switch

http://us1.php.net/manual/en/control-structures.switch.php

The switch statement is similar to a series of IF statements on the same expression. A special case is the default case that matches anything that wasn't matched by the other cases.

```
if ($i == 0) {
    echo "i equals 0";
} elseif ($i == 1) {
    echo "i equals 1";
} elseif ($i == 2) {
    echo "i equals 2";
}

switch ($i) {
    case 0:
        echo "i equals 0";
        break;
    case 1:
        echo "i equals 1";
        break;
    case 2:
        echo "i equals 2";
        break;
    default:
       echo "i is not equal to 0, 1 or 2";
}
```

Exercise: Conditionals 2

Continue the previous exercise but this time use the Switch statement. Create a function that returns a letter grade based on the following point breakdowns:
100-90=A
80-89=B
79-70=C
69-60=D
<60=F

To test your function, try it with these 5 point values and echo the result back out for BOTH functions you created:
1. 94
2. 54
3. 89.9
4. 60.01
5. 102.1

While Loops

http://us1.php.net/manual/en/control-structures.while.php

A while statement tells PHP to execute the nested statement(s) repeatedly, as long as the while expression evaluates to TRUE. The value of the expression is checked each time at the **beginning** of the loop, so even if this value changes during the execution of the nested statement(s), execution will not stop until the end of the iteration (each time PHP runs the statements in the loop is one iteration). Sometimes, if the while expression evaluates to FALSE from the very beginning, the nested statement(s) won't even be run once.

```
$i = 1;
while ($i <= 10) {
    echo $i
    $i++;
   }
```

Do-While Loops

http://us1.php.net/manual/en/control-structures.do.while.php

do-while loops are very similar to while loops, except the truth expression is checked at the **end** of each iteration instead of in the beginning. The main difference from regular while loops is that the first iteration of a do-while loop is guaranteed to run (the truth expression is only checked at the end of the iteration), whereas it may not necessarily run with a regular while loop (the truth expression is checked at the beginning of each iteration, if it evaluates to FALSE right from the beginning, the loop execution would end immediately).

```
$i = 0;
do {
    echo $i;
} while ($i > 0);
```

For Loops

http://us1.php.net/manual/en/control-structures.for.php

for loops are the most complex loops in PHP.

```
for (expr1; expr2; expr3)
    statement
```

The first expression (expr1) is evaluated (executed) once unconditionally at the beginning of the loop.

In the beginning of each iteration, expr2 is evaluated. If it evaluates to TRUE, the loop continues and the nested statement(s) are executed. If it evaluates to FALSE, the execution of the loop ends.

At the end of each iteration, expr3 is evaluated (executed).

```
for ($i = 1; $i <= 10; $i++) {
    echo $i;
}
```

ForEach Construct

http://us1.php.net/manual/en/control-structures.foreach.php

The foreach construct provides an easy way to iterate over arrays. foreach works only on arrays and objects, and will issue an error when you try to use it on a variable with a different data type or an uninitialized variable. There are two syntaxes:

```
foreach (array_expression as $value)
    statement
```

On each iteration, the value of the current element is assigned to $value and the internal array pointer is advanced by one (so on the next iteration, you'll be looking at the next element).

```
foreach (array_expression as $key => $value)
    statement
```

Will additionally assign the current element's key to the $key variable on each iteration.

```php
$arr = array(
    "one" => 1,
    "two" => 2,
    "three" => 3,
    "seventeen" => 17
);

foreach ($arr as $key => $value) {
    echo "\$arr[$key] => $value.\n";
}
```

Exercise: Loops

Part 1: Create an array indexed by integers. Create 5 solid color values for the even numbers (starting at 0), then a shade of that color for the successive odd number.
E.g., [0] => "Red", [1] => "Pink", ...

Loop through the colors of the array and display the index number and color name.
E.g., "Color 1 is Red"

You can do this a number of ways - for this exercise there is no right or wrong expectation to approach the problem as long as it displays the colors in order and uses a form of loop.

Part 2: Repeat the same above, but display the colors in reverse order.

Part 3: Repeat Part 1 above, but only display the solid colors.

Superglobal Variables

http://us1.php.net/language.variables.superglobals.php

Several predefined variables in PHP are "superglobals", which means they are available in all scopes throughout a script. There is no need to do global $variable; to access them within functions or methods.

$_GET - An associative array of variables passed to the current script via the URL parameters.

```
<form method="GET" action="myscript.php">
<input type="text" name="name" />
</form>
echo 'Hello ' . $_GET["name"] . '!';
```

Assuming the user entered http://example.com/?name=Hannes

The above example will output something similar to:

```
Hello Hannes!
```

$_POST - An associative array of variables passed via the HTTP POST method.

```
<form method="POST" action="myscript.php">
<input type="text" name="name" />
</form>
echo 'Hello ' . $_POST["name"] . '!';
Hello Hannes!
```

Variable Handling and Validation

http://us1.php.net/manual/en/ref.var.php

We may need to check to see if a variable has been set, or validate the contents of the variable before storing it or taking other action.

is_null — Finds whether a variable is NULL
empty — Determine whether a variable is empty
isset — Determine if a variable is set and is not NULL
filter_var — Filters a variable with a specified filter
http://www.php.net/manual/en/function.filter-var.php

```
$email_a = 'joe@example.com';
$email_b = 'bogus';

if (filter_var($email_a, FILTER_VALIDATE_EMAIL)) {
    echo "This ($email_a) email address is considered valid.";
}
if (filter_var($email_b, FILTER_VALIDATE_EMAIL)) {
    echo "This ($email_b) email address is considered valid.";
}
```

preg_match - Searches subject for a match to the regular expression given in pattern.
http://us1.php.net/manual/en/function.preg-match.php

```
$subject = "abcdef";
$pattern = '/^def/';
preg_match($pattern, $subject, $matches, PREG_OFFSET_CAPTURE, 3);
```

File Uploads

http://us1.php.net/manual/en/function.move-uploaded-file.php
http://us1.php.net/manual/en/features.file-upload.post-method.php
The $_FILES[] is a superglobal that contains an array of files and their attributes as uploaded by the browser.
When files are uploaded through a POST method, PHP will store the uploaded files in a temporary location and with a temporary filename.

$_FILES['userfile']['name']
The original name of the file on the client machine.

$_FILES['userfile']['type']
The mime type of the file, if the browser provided this information. An example would be "*image/gif*". This mime type is however not checked on the PHP side and therefore don't take its value for granted.

$_FILES['userfile']['size']
The size, in bytes, of the uploaded file.

$_FILES['userfile']['tmp_name']
The temporary filename of the file in which the uploaded file was stored on the server.

$_FILES['userfile']['error']
The error code associated with this file upload. This element was added in PHP 4.2.0

File Uploads

Form type must include *multipart/form-data* as shown below.

```
<form enctype="multipart/form-data" action="__URL__" method="POST">
    <!-- Name of input element determines name in $_FILES array -->
    Send this file: <input name="userfile" type="file" />
    <input type="submit" value="Send File" />
</form>
```

```
$uploaddir = '/var/www/uploads/';
$uploadfile = $uploaddir . basename($_FILES['userfile']['name']);

echo '<pre>';
if (move_uploaded_file($_FILES['userfile']['tmp_name'], $uploadfile)) {
    echo "File is valid, and was successfully uploaded.\n";
} else {
    echo "Possible file upload attack!\n";
}

echo 'Here is some more debugging info:';
print_r($_FILES);

print "</pre>";
```

Exercise: File Uploads

Create a test script to upload a JPEG file to your web server. When upload is completed, echo back out the file attributes and display the JPEG image.

Redirects

http://us1.php.net/manual/en/function.http-redirect.php
Sends a response to browser to redirect it to another page. Also sends HTTP Status Code about whether it is a permanent or temporary redirection.
http://www.w3.org/QA/Tips/reback

bool http_redirect ([string $url [, array $params [, bool $session = false [, int $status = 0]]]])

Example #1 A http_redirect() example

```
<?php
http_redirect("relpath", array("name" => "value"), true, HTTP_REDIRECT_PERM);
?>
```

The above example will output:

```
HTTP/1.1 301 Moved Permanently
X-Powered-By: PHP/5.2.2
Content-Type: text/html
Location: http://www.example.com/curdir/relpath?name=value&PHPSESSID=abc

Redirecting to <a href="http://www.example.com/curdir/relpath?
name=value&PHPSESSID=abc">http://www.example.com/curdir/relpath?
name=value&PHPSESSID=abc</a>.
```

Today's Lab (Lab 2)

In this lab you will create an input form in HTML, validate the user input on the server side, and if the validation is successful store the form fields in an array and echo out the contents of the array. (You may have validated forms using client-side JavaScript before, but for this lab do the validation using PHP.)

Part 1: Create an HTML form using POST to capture a user contact request, including: First Name, Last Name, Address, City, State, Zip, Email address, Phone number, and Web URL. Capture the form data using *foreach* in an array and display the results back out as a formatted mailing address in a textarea (Line 1 - names, Line 2 - Address, Line 3 - City, state, zip), and the remaining fields below the textarea .

Part 2: For the form you created in Part 1, validate the State, Zip, Email address, Phone Number, and Web URL. Assume that the phone number is inputted in the format xxx-xxx-xxxx, and the zip code is either 5-digit or Zip+4 (xxxxx-xxxx). Validate that the user entered information for the names and address.
IF the user inputted data did not pass validation, then allow the user to change the information entered and display a message on the field to correct. *(The user should NOT have to retype all fields in again.)*

SUBMIT: Source code zipped and uploaded on FSO Lab 2; Link to screencast video explaining code uploaded on FSO Video 2.