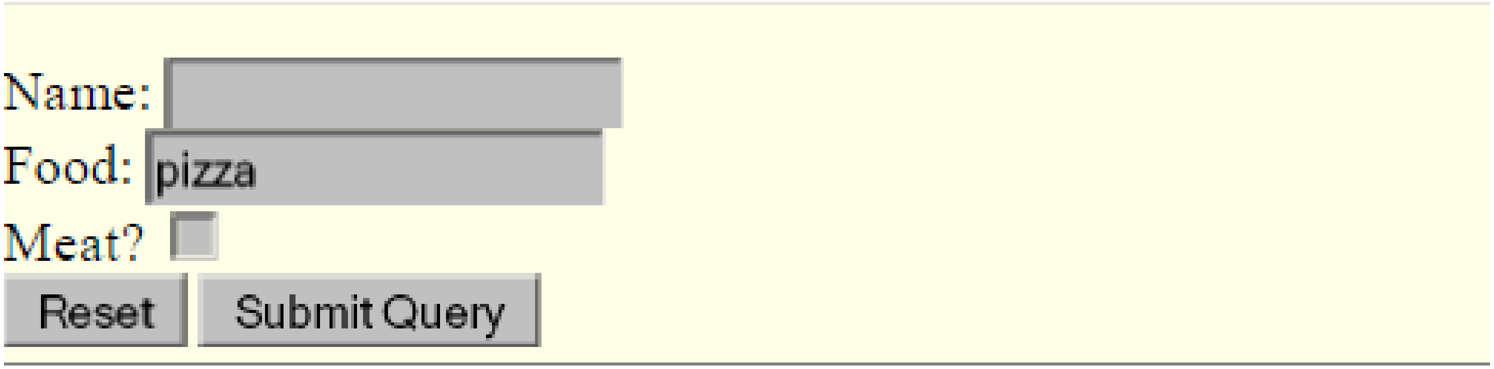
More forms

Reset Buttons

```html
Name: <input type="text" name="name" /> <br />
Food: <input type="text" name="meal" value="pizza" /> <br
/>
<label>Meat? <input type="checkbox" name="meat" /></label>
<br />
<input type="reset" />                                    HTML
```



- specify custom text on the button by setting its value attribute

Grouping input: `<fieldset>`, `<legend>`

```
<fieldset>
	<legend>Credit cards:</legend>
	<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
</fieldset>                                                    HTML
```

- fieldset groups related input fields, adds a border; legend supplies a caption

Common UI control errors

- “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

Styling form controls

```
input[type="text"] {
        background-color: yellow;
        font-weight: bold;
}                                                    CSS
```

- attribute selector: matches only elements that have a particular attribute value
- useful for controls because many share the same element (input)

Hidden input parameters

```html
<input type="text" name="username" /> Name <br />
<input type="text" name="sid" /> SID <br />
<input type="hidden" name="school" value="UW" />
<input type="hidden" name="year" value="2048" />
```
HTML

- an invisible parameter that is still passed to the server when form is submitted
- useful for passing on additional state that isn't modified by the user

Submitting data

Problems with submitting data

```html
<form action="http://localhost/test1.php" method="get">
<label><input type="radio" name="cc" /> Visa</label>
<label><input type="radio" name="cc" /> MasterCard</label>
<br />
Favorite Star Trek captain:
<select name="startrek">
        <option>James T. Kirk</option>
        <option>Jean-Luc Picard</option>
</select> <br />
</form>
```
HTML

- the form may look correct, but when you submit it...
- `[cc] => on, [startrek] => Jean-Luc Picard`
- How can we resolve this conflict?

The `value` attribute

```html
<label><input type="radio" name="cc" value="visa" />
Visa</label>
<label><input type="radio" name="cc" value="mastercard" />
MasterCard</label> <br />
Favorite Star Trek captain:
<select name="startrek">
      <option value="kirk">James T. Kirk</option>
      <option value="picard">Jean-Luc Picard</option>
<input type="submit" value="submit" />
</select> <br />
```
HTML

- value attribute sets what will be submitted if a control is selected
- `[cc] => visa, [startrek] => picard`

URL-encoding

- certain characters are not allowed in URL query parameters:
  - examples: " ", "/", "=", "&"
- when passing a parameter, it is URL-encoded
  - “Xenia's cool!?" → “Xenia%27s+cool%3F%21"
- you don't usually need to worry about this:
  - the browser automatically encodes parameters before sending them
  - the PHP $_REQUEST array automatically decodes them
  - ... but occasionally the encoded version does pop up (e.g. in Firebug)

Submitting data to a web server

- though browsers mostly retrieve data, sometimes you want to submit data to a server
  - Hotmail: Send a message
  - Flickr: Upload a photo
  - Google Calendar: Create an appointment
- the data is sent in HTTP requests to the server
  - with HTML forms
  - with **Ajax** (seen later)
- the data is placed into the request as parameters

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

HTTP `GET` vs. `POST` requests

- 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 `POST` example

```
<form action="http://localhost/app.php" method="post">
<div>
        Name: <input type="text" name="name" /> <br />
        Food: <input type="text" name="meal" /> <br />
        <label>Meat? <input type="checkbox" name="meat"
/></label> <br />
        <input type="submit" />
<div>
</form>                                             HTML
```

GET or POST?

```
if ($_SERVER["REQUEST_METHOD"] == "GET") {
        # process a GET request
...
} elseif ($_SERVER["REQUEST_METHOD"] == "POST") {
        # process a POST request
...
}
```
PHP

- some PHP pages process both GET and POST requests
- to find out which kind of request we are currently processing, look at the global `$_SERVER` array's "`REQUEST_METHOD`" element

Uploading files

```
<form action="http://webster.cs.washington.edu/params.php"
method="post" enctype="multipart/form-data">
      Upload an image as your avatar:
      <input type="file" name="avatar" />
      <input type="submit" />
</form>                                              HTML
```

- add a file upload to your form as an input tag with type of file
- must also set the `enctype` attribute of the form

Processing form data in PHP

"Superglobal" arrays

Array	Description
$_REQUEST	parameters passed to any type of request
$_GET, $_POST	parameters passed to GET and POST requests
$_SERVER, $_ENV	information about the web server
$_FILES	files uploaded with the web request
$_SESSION, $_COOKIE	"cookies" used to identify the user (seen later)

- PHP "superglobal" arrays contain information about the current request, server, etc.
- These are special kinds of arrays called associative arrays.

Associative arrays

```
$blackbook = array();
$blackbook[“xenia”] = "206-685-2181";
$blackbook[“anne”] = "206-685-9138";
...
print “Xenia's number is " . $blackbook[“xenia"] . ".\n";
```
PHP

- associative array (a.k.a. map, dictionary, hash table) : uses non-integer indexes
- associates a particular index "key" with a value
  - key “xenia" maps to value "206-685-2181"

Example: exponents

```php
<?php
       $base = $_REQUEST["base"];
       $exp = $_REQUEST["exponent"];
       $result = pow($base, $exp);
?>
<?= $base ?> ^ <?= $exp ?> = <?= $result ?>
```
PHP

- What should we do to run this with xampp?

Example: Print all parameters

```php
<?php
foreach ($_REQUEST as $param => $value) {
  ?>
  <p>Parameter <?= $param ?> has value <?= $value ?></p>
  <?php
}
?>
```
PHP

- What should we do to run this with xampp?

Processing an uploaded file in PHP

- uploaded files are placed into global array `$_FILES`, not `$_REQUEST`
- each element of `$_FILES` is itself an associative array, containing:
  - `name`: the local filename that the user uploaded
  - `type`: the MIME type of data that was uploaded, such as image/jpeg
  - `size` : file's size in bytes
  - `tmp_name` : a filename where PHP has temporarily saved the uploaded file
    - to permanently store the file, move it from this location into some other file

Uploading files

```
<input type="file" name="avatar" />
                                                HTML
```

- example: if you upload tobby.jpg as a parameter named avatar,
  - $_FILES["avatar"]["name"] will be “tobby.jpg"
  - $_FILES["avatar"]["type"] will be "image/jpeg"
  - $_FILES["avatar"]["tmp_name"] will be something like "/var/tmp/phpZtR4TI"

```
Array
(
    [file1] => Array
        (
            [name] => MyFile.txt (comes from the browser,
so treat as tainted)
            [type] => text/plain  (not sure where it gets
this from - assume the browser, so treat as tainted)
            [tmp_name] => /tmp/php/php1h4j1o (could be
anywhere on your system, depending on your config
settings, but the user has no control, so this isn't
tainted)
            [error] => UPLOAD_ERR_OK  (= 0)
            [size] => 123   (the size in bytes)
        )
   [file2] => Array
        (
            [name] => MyFile.jpg
            [type] => image/jpeg
            [tmp_name] => /tmp/php/php6hst32
            [error] => UPLOAD_ERR_OK
            [size] => 98174
        )
)
```

Processing uploaded file example

```
$username = $_REQUEST["username"];
if (is_uploaded_file($_FILES["avatar"]["tmp_name"])) {
move_uploaded_file($_FILES["avatar"]["tmp_name"],
"$username/avatar.jpg");
	print "Saved uploaded file as
$username/avatar.jpg\n";
} else {
	print "Error: required file not uploaded";
}
```
PHP

- functions for dealing with uploaded files:
  - is_uploaded_file(filename)
    returns TRUE if the given filename was uploaded by the user
  - move_uploaded_file(from, to)
    moves from a temporary file location to a more permanent file

Including files: `include`

```
include("header.php");
```
PHP

- inserts the entire contents of the given file into the PHP script's output page
- encourages modularity
- useful for defining reused functions needed by multiple pages