CSE3026: Web Application Development
Forms

Scott Uk-Jin Lee

6.1: Form Basics

Web data

- most interesting web pages revolve around data
  - examples: Google, IMDB, Digg, Facebook, YouTube, Rotten Tomatoes
  - can take many formats: text, HTML, XML, multimedia
- many of them allow us to access their data
- some even allow us to submit our own new data
- most server-side web programs accept **parameters** that guide their execution

Query strings and parameters

```
URL?name=value&name=value...
```

```
http://www.google.com/search?q=Obama
http://example.com/student_login.php?username=lee&id=1234567
```

- **query string**: a set of parameters passed from a browser to a web server
  - often passed by placing name/value pairs at the end of a URL
  - above, parameter `username` has value `lee`, and `sid` has value `1234567`
- PHP code on the server can examine and utilize the value of parameters
- a way for PHP code to produce different output based on values passed by the user

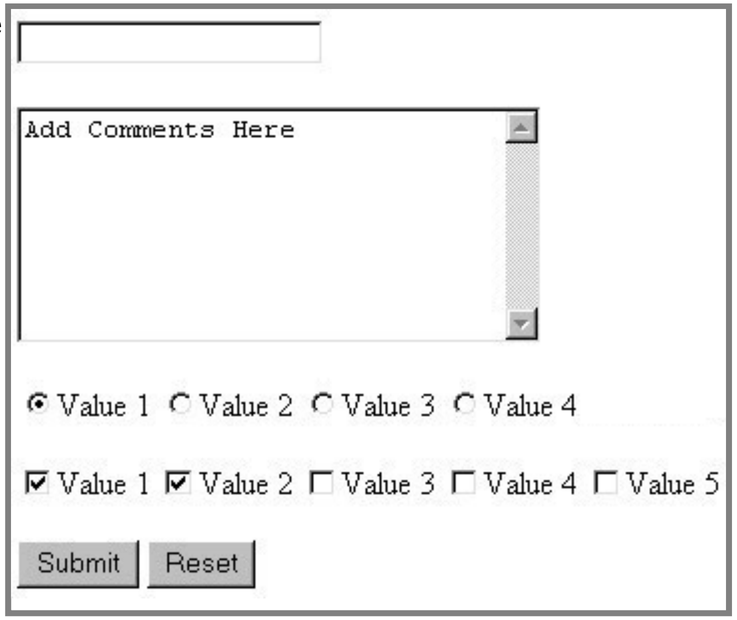
Query parameters: `$_GET`, `$_POST`

```php
$user_name = $_GET["username"];
$id_number = (int) $_GET["id"];
$eats_meat = FALSE;
if (isset($_GET["meat"])) {
  $eats_meat = TRUE;
}
```
PHP

- `$_GET["`*parameter name*`"]` or `$_POST["`*parameter name*`"]` returns a GET/POST parameter's value as a string
- parameters specified as `http://....?`*name*`=`*value*`&`*name*`=`*value* are GET parameters
- test whether a given parameter was passed with `isset`

HTML forms

- **form**: 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**
- JavaScript can be used to create interactive controls (seen later)



HTML form: `<form>`

```
<form action="destination URL">
  form controls
</form>
```
HTML

- required `action` attribute gives the URL of the page that will process this form's data
- when form has been filled out and **submitted**, its data will be sent to the `action`'s URL
- one page may contain many forms if so desired

Form example

```
<form action="http://www.google.com/search">
  <div>
    Let's search Google:
    <input name="q" />
    <input type="submit" />
  </div>
</form>
```
HTML

Let's search Google: [ ] [제출]

output

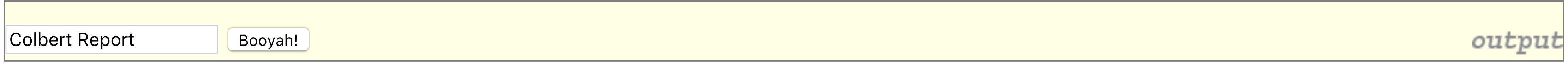
- must wrap the form's controls in a block element such as `div`

6.2: Form Controls

Form controls: `<input>`

```html
<!-- 'q' happens to be the name of Google's required parameter -->
<input type="text" name="q" value="Colbert Report" />
<input type="submit" value="Booyah!" />
```

HTML

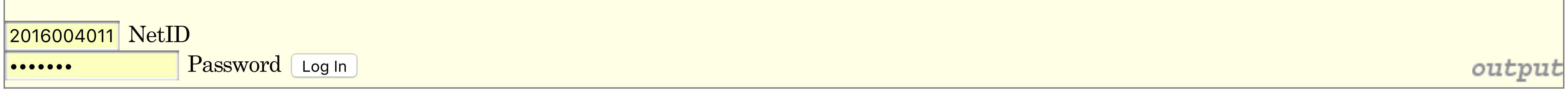


output

- `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

Text fields: `<input>`

```html
<input type="text" size="10" maxlength="8" /> NetID <br />
<input type="password" size="16" /> Password
<input type="submit" value="Log In" />
```
HTML

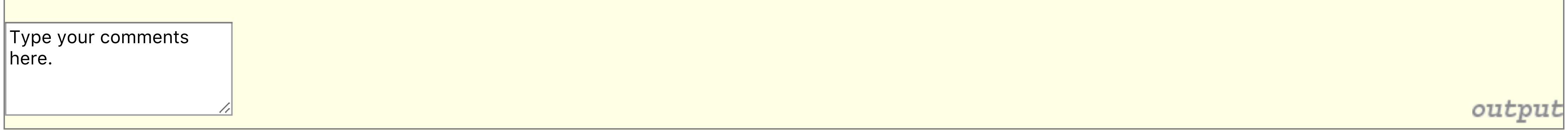

2016004011 NetID
••••••• Password

- `input` attributes: `disabled`, `maxlength`, `readonly`, `size`, `value`
- `size` attribute controls onscreen width of text field
- `maxlength` limits how many characters user is able to type into field

Text boxes: `<textarea>`

a multi-line text input area (inline)

```html
<textarea rows="4" cols="20">
Type your comments here.
</textarea>
```
HTML


Type your comments here.

- initial text is placed inside `textarea` tag (optional)
- required `rows` and `cols` attributes specify height/width in characters
- optional `readonly` attribute means text cannot be modified

Checkboxes: `<input>`

yes/no choices that can be checked and unchecked (inline)

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

☐ Lettuce ☑ Tomato ☑ Pickles [제출]

output

- 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://domain_name/params.php?tomato=on&pickles=on`
- use `checked="checked"` attribute in HTML to initially check the box

Radio buttons: `<input>`

sets of mutually exclusive choices (inline)

```html
<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
```
HTML

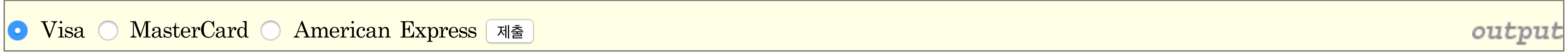
◉ Visa ○ MasterCard ○ American Express [제출]

output

- 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`

Text labels: `<label>`

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

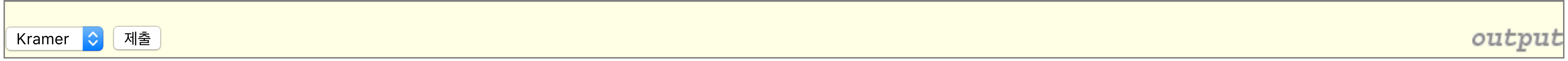


- associates nearby text with control, so you can click text to activate control
- can be used with checkboxes or radio buttons
- `label` element can be targeted by CSS style rules

Drop-down list: `<select>`, `<option>`

menus of choices that collapse and expand (inline)

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

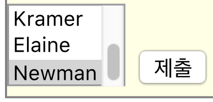


- `option` element represents each choice
- `select` optional attributes: `disabled`, `multiple`, `size`
- optional `selected` attribute sets which one is initially chosen

Using `<select>` for lists

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

HTML



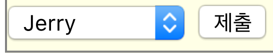
output

- optional `multiple` attribute allows selecting multiple items with shift- or ctrl-click
  - must declare parameter's name with `[]` if you allow multiple selections
- `option` tags can be set to be initially `selected`

Option groups: `<optgroup>`

```html
<select name="favoritecharacter">
  <optgroup label="Major Characters">
    <option>Jerry</option>
    <option>George</option>
    <option>Kramer</option>
    <option>Elaine</option>
  </optgroup>
  <optgroup label="Minor Characters">
    <option>Newman</option>
    <option>Susan</option>
  </optgroup>
</select>
```

HTML

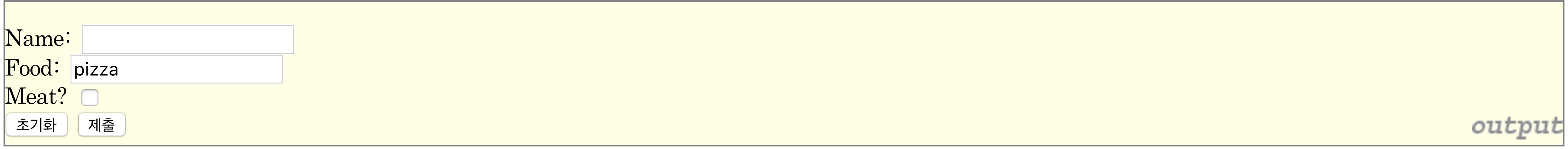


output

- What should we do if we don't like the bold italic?

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


Name:
Food: pizza
Meat?

- when clicked, returns all form controls to their initial values
- specify custom text on the button by setting its `value` attribute

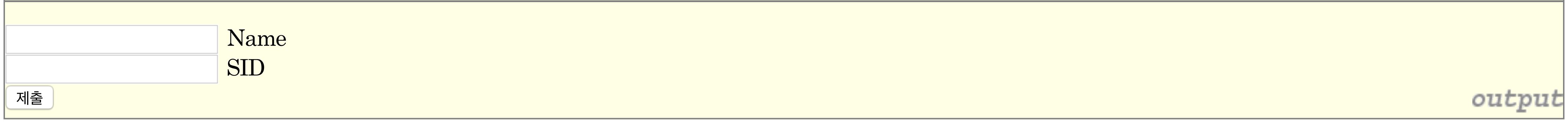
Common UI control errors

- "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

Hidden input parameters

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

HTML



output

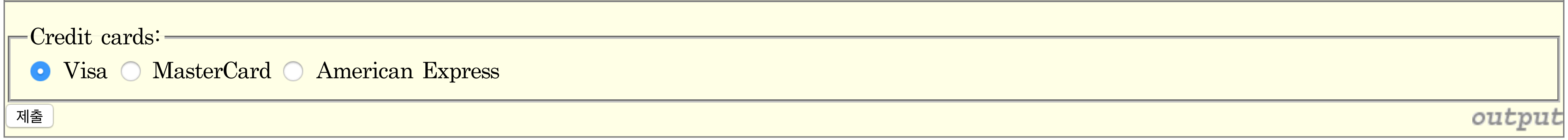
- an invisible parameter that is still passed to the server when form is submitted
- useful for passing on additional state that isn't modified by the user

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

groups of input fields with optional caption (block)

```html
<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



output

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

Styling form controls

```
element[attribute="value"] {
   property : value;
   property : value;
   ...
   property : value;
}
```
CSS

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

```
Borat
```
output

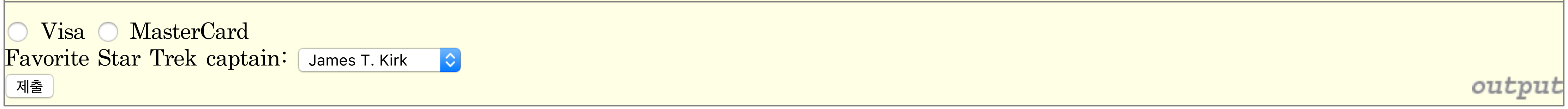
- **attribute selector**: matches only elements that have a particular attribute value
- useful for controls because many share the same element (`input`)

6.3: Submitting Data

Problems with submitting data

```html
<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 />
```

HTML

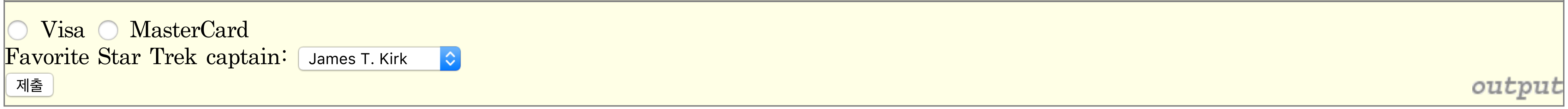


- this form submits to our handy params.php tester page
- the form may look correct, but when you submit it...
- **`[cc] => on`**`, [startrek] => Jean-Luc Picard`

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>
</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** (reference table)
  - `"Scott's cool!?"` → `"Scott%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
  - ... but occasionally the encoded version does pop up (e.g. in Firebug)

Submitting data to a web server

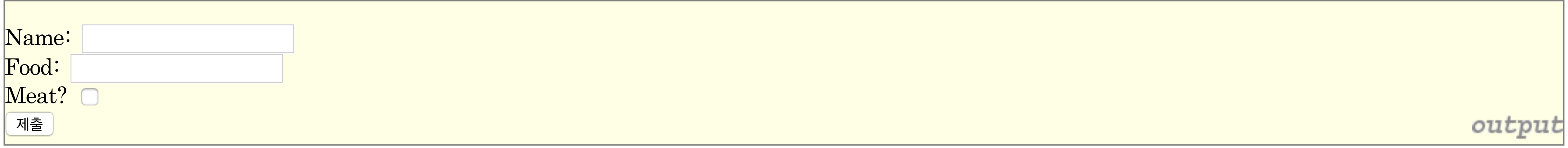
- though browsers mostly retrieve data, sometimes you want to submit data to a server
  - Hotmail: Send a message
  - Flickr: Upload a photo
  - Google Calendar: Create an appointment
- the data is sent in HTTP requests to the server
  - with HTML forms
  - with **Ajax** (seen later)
- the data is placed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

```html
<form action="http://foo.com/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


Name:
Food:
Meat?

output

GET or POST?

```php
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

6.4: Processing Form Data in PHP

"Superglobal" arrays

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

- 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["scott"] = "031-400-5238";
$blackbook["jaejin"] = "031-400-4754";
...
print "Scott's number is " . $blackbook["scott"] . ".\n";
```
PHP

- **associative array** (a.k.a. **map**, **dictionary**, **hash table**) : uses non-integer indexes
- associates a particular index "key" with a value
  - key `"scott"` maps to value `"031-400-5238"`
- syntax for embedding an associative array element in interpreted string:

```
print "Scott's number is {$blackbook['scott']}.\n";
```
PHP

Uploading files

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

HTML

Upload an image as your avatar: 파일 선택 선택된 파일 없음 제출

output

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

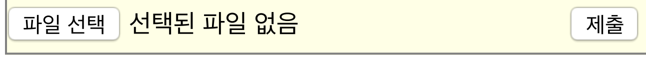
- it makes sense that the form's request method must be `post` (an entire file can't be put into a URL!)
- form's `enctype` (data encoding type) must be set to `multipart/form-data` or else the file will not arrive at the server

Processing an uploaded file in PHP

- uploaded files are placed into global array `$_FILES`, not `$_POST`
- 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 details

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


파일 선택 선택된 파일 없음 제출

output

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

Processing uploaded file, example

```
$username = $_POST["username"];
if (is_uploaded_file($_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
- proper idiom: check `is_uploaded_file`, then do `move_uploaded_file`

Including files: `include`

```
include("filename");
```

PHP

```
include("header.php");
include("shared-code.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

Extra stuff about associative arrays

Creating an associative array

```
$name = array();
$name["key"] = value;
...
$name["key"] = value;
```
PHP

```
$name = array(key => value, ..., key => value);
```
PHP

```
$blackbook = array("scott"  => "031-400-5238",
                   "jaejin" => "031-400-4754",
                   "cathy"  => "031-400-7777");
```
PHP

- can be declared either initially empty, or with a set of predeclared key/value pairs

Printing an associative array

```
print_r($blackbook);
```
PHP

```
Array
(
    [scott] => 031-400-5238
    [jaejin] => 031-400-4754
    [cathy] => 031-400-7777
)
```
output

- `print_r` function displays all keys/values in the array
- `var_dump` function is much like `print_r` but prints more info
- unlike `print`, these functions require parentheses

Associative array functions

```php
if (isset($blackbook["scott"])) {
  print "Scott's phone number is {$blackbook['scott']}\n";
} else {
  print "No phone number found for Scott Lee.\n";
}
```
PHP

name(s)	category
`isset`, `array_key_exists`	whether the array contains value for given key
`array_keys`, `array_values`	an array containing all keys or all values in the assoc.array
`asort`, `arsort`	sorts by value, in normal or reverse order
`ksort`, `krsort`	sorts by key, in normal or reverse order

`foreach` loop and associative arrays

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

```
scott's phone number is 031-400-5238
jaejin's phone number is 031-400-4754
cathy's phone number is 031-400-7777
```
output

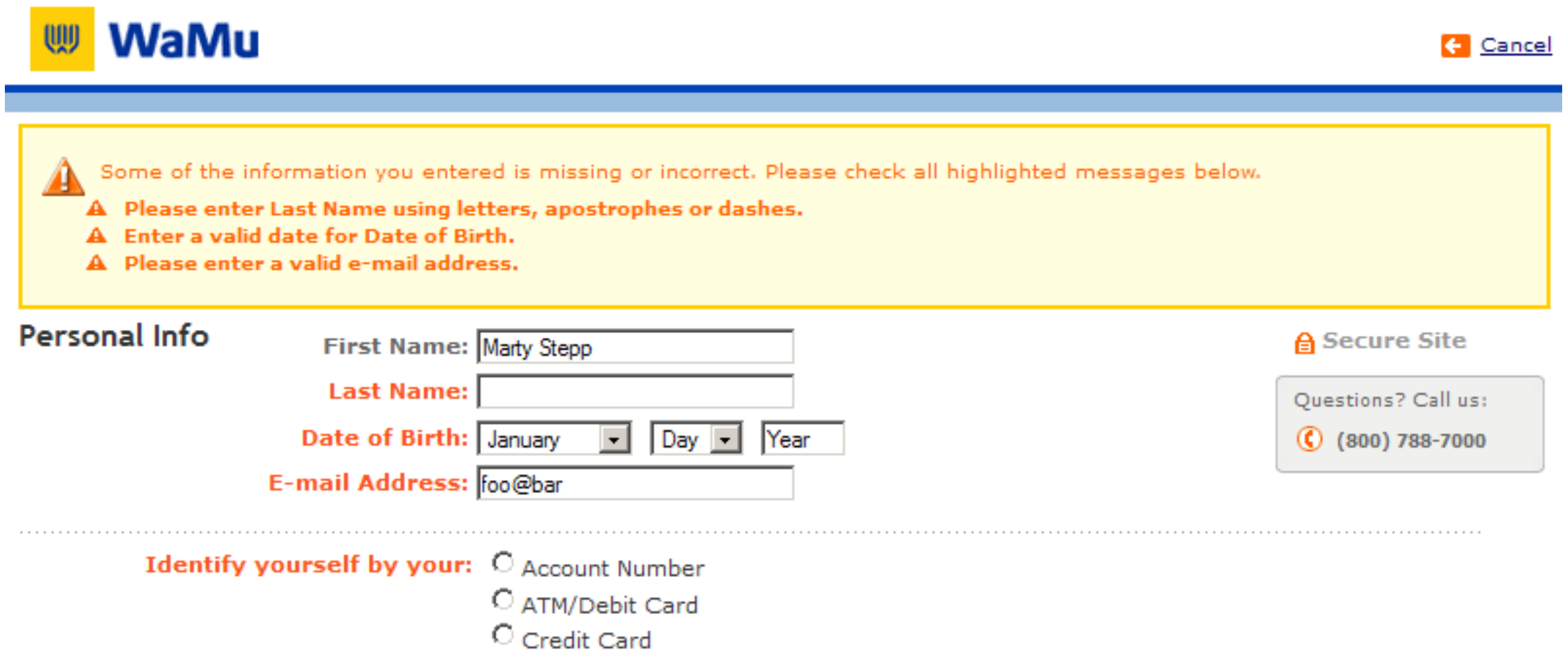
- both the key and the value are given a variable name
- the elements will be processed in the order they were added to the array

Form Validation

What is form validation?

- **validation**: ensuring that form's values are correct
- some types of validation:
  - preventing blank values (email address)
  - ensuring the type of values
    - integer, real number, currency, phone number, Social Security number, postal address, email address, date, credit card number, ...
  - ensuring the format and range of values (ZIP code must be a 5-digit integer)
  - ensuring that values fit together (user types email twice, and the two must match)

A real form that uses validation


Cancel

Some of the information you entered is missing or incorrect. Please check all highlighted messages below.

- Please enter Last Name using letters, apostrophes or dashes.
- Enter a valid date for Date of Birth.
- Please enter a valid e-mail address.

Personal Info

First Name:	Marty Stepp
Last Name:	
Date of Birth:	January, Day, Year
E-mail Address:	foo@bar

Secure Site

Questions? Call us: (800) 788-7000

Identify yourself by your: ○ Account Number ○ ATM/Debit Card ○ Credit Card

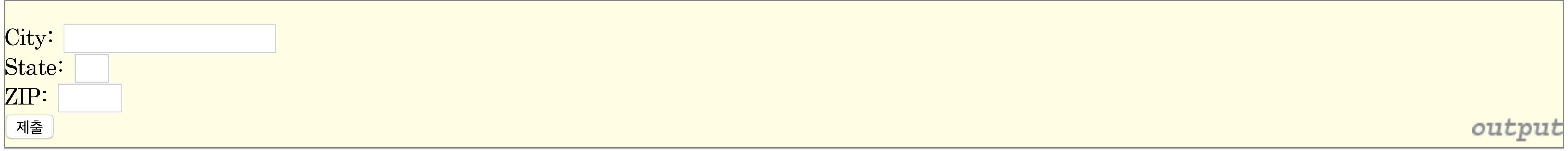
Client vs. server-side validation

Validation can be performed:

- **client-side** (before the form is submitted)
  - can lead to a better user experience, but not secure (why not?)
- **server-side** (in PHP code, after the form is submitted)
  - needed for truly secure validation, but slower
- both
  - best mix of convenience and security, but requires most effort to program

An example form to be validated

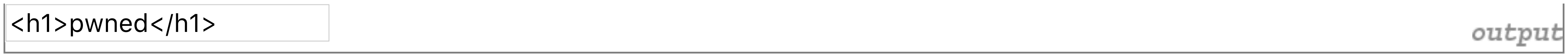
```html
<form action="http://foo.com/foo.php" method="get">
    <div>
        City:  <input name="city" /> <br />
        State: <input name="state" size="2" maxlength="2" /> <br />
        ZIP:   <input name="zip" size="5" maxlength="5" /> <br />
        <input type="submit" />
    </div>
</form>
```
HTML


City:
State:
ZIP:

output

- Let's validate this form's data on the server...

One problem: Users submitting HTML content


<h1>pwned</h1>

output

- A user might submit information to a form that contains HTML syntax
- If we're not careful, this HTML will be inserted into our pages (why is this bad?)

The `htmlspecialchars` function

`htmlspecialchars`	returns an HTML-escaped version of a string

- text from files / user input / query params might contain <, >, &, etc.
- we could manually write code to strip out these characters
- better idea: allow them, but **escape** them

```
$text = "<p>hi 2 u & me</p>";
$text = htmlspecialchars($text);   # "&lt;p&gt;hi 2 u &amp; me&lt;/p&gt;"
```
PHP

Basic server-side validation code

```
$city  = $_POST["city"];
$state = $_POST["state"];
$zip   = $_POST["zip"];
if (!$city || strlen($state) != 2 || strlen($zip) != 5) {
    print "Error, invalid city/state/zip submitted.";
}
```
PHP

- *basic idea*: examine parameter values, and if they are bad, show an error message and abort. But:
  - How do you test for integers vs. real numbers vs. strings?
  - How do you test for a valid credit card number?
  - How do you test that a person's name has a middle initial?
  - (How do you test whether a given string matches a particular complex format?)

Regular expressions

```
/^[a-zA-Z_\-]+@(([a-zA-Z_\-])+\.)+[a-zA-Z]{2,4}$/
```

- **regular expression** ("regex"): a description of a pattern of text
  - can test whether a string matches the expression's pattern
  - can use a regex to search/replace characters in a string
- regular expressions are extremely powerful but tough to read
  (the above regular expression matches email addresses)
- regular expressions occur in many places:
  - Java: `Scanner`, `String`'s `split` method
  - supported by PHP, JavaScript, and other languages
  - many text editors (TextPad) allow regexes in search/replace

Basic regular expressions

```
/abc/
```

- in PHP, regexes are strings that begin and end with `/`
- the simplest regexes simply match a particular substring
- the above regular expression matches any string containing `"abc"`:
  - YES: `"abc"`, `"abcdef"`, `"defabc"`, `".=.abc.=."`, ...
  - NO: `"fedcba"`, `"ab c"`, `"PHP"`, ...

Wildcards: `.`

- A dot `.` matches any character except a `\n` line break
  - `/.oo.y/` matches `"Doocy"`, `"goofy"`, `"LooNy"`, ...
- A trailing `i` at the end of a regex (after the closing `/`) signifies a case-insensitive match
  - `/mart/i` matches `"Marty Stepp"`, `"smart fellow"`, `"WALMART"`, ...

Special characters: `|`, `()`, `\`

- `|` means *OR*
  - `/abc|def|g/` matches `"abc"`, `"def"`, or `"g"`
  - There's no *AND* symbol. Why not?
- `()` are for grouping
  - `/(Homer|Marge) Simpson/` matches `"Homer Simpson"` or `"Marge Simpson"`
- `\` starts an **escape sequence**
  - many characters must be escaped to match them literally: `/ \ $ . [ ] ( ) ^ * + ?`
  - `/<br \/>/` matches lines containing `<br />` tags

Quantifiers: `*`, `+`, `?`

- `*` means 0 or more occurrences
  - `/abc*/` matches `"ab"`, `"abc"`, `"abcc"`, `"abccc"`, ...
  - `/a(bc)*/` matches `"a"`, `"abc"`, `"abcbc"`, `"abcbcbc"`, ...
  - `/a.*a/` matches `"aa"`, `"aba"`, `"a8qa"`, `"a!?xyz__9a"`, ...
- `+` means 1 or more occurrences
  - `/a(bc)+/` matches `"abc"`, `"abcbc"`, `"abcbcbc"`, ...
  - `/Goo+gle/` matches `"Google"`, `"Gooogle"`, `"Goooogle"`, ...
- `?` means 0 or 1 occurrences
  - `/a(bc)?/` matches `"a"` or `"abc"`

More quantifiers: `{min,max}`

- `{`*min*`,`*max*`}` means between *min* and *max* occurrences (inclusive)
  - `/a(bc){2,4}/` matches `"abcbc"`, `"abcbcbc"`, or `"abcbcbcbc"`
- *min* or *max* may be omitted to specify any number
  - `{2,}` means 2 or more
  - `{,6}` means up to 6
  - `{3}` means exactly 3

Anchors: `^` and `$`

- `^` represents the beginning of the string or line;
  `$` represents the end
  - `/Jess/` matches all strings that contain `Jess`;
    `/^Jess/` matches all strings that *start with* `Jess`;
    `/Jess$/` matches all strings that *end with* `Jess`;
    `/^Jess$/` matches the exact string `"Jess"` only
  - `/^Mart.*Stepp$/` matches `"MartStepp"`, `"Marty Stepp"`, `"Martin D Stepp"`, ...
    but NOT `"Marty Stepp stinks"` or `"I H8 Martin Stepp"`
- (on the other slides, when we say, `/PATTERN/` matches `"text"`, we really mean that it matches any string that contains that text)

Character sets: `[]`

- `[]` group characters into a **character set**; will match any single character from the set
  - `/[bcd]art/` matches strings containing `"bart"`, `"cart"`, and `"dart"`
  - equivalent to `/(b|c|d)art/` but shorter
- inside `[]`, many of the modifier keys act as normal characters
  - `/what[!*?]*/` matches `"what"`, `"what!"`, `"what?**!"`, `"what??!"`, ...
- What regular expression matches DNA (strings of A, C, G, or T)?
  - `/[ACGT]+/`

Character ranges: `[start-end]`

- inside a character set, specify a range of characters with `-`
  - `/[a-z]/` matches any lowercase letter
  - `/[a-zA-Z0-9]/` matches any lower- or uppercase letter or digit
- an initial `^` inside a character set negates it
  - `/[^abcd]/` matches any character other than a, b, c, or d
- inside a character set, `-` must be escaped to be matched
  - `/[+\-]?[0-9]+/` matches an optional `+` or `-`, followed by at least one digit
- What regular expression matches letter grades such as A, B+, or D- ?
  - `/[ABCDF][+\-]?/`

Escape sequences

- special escape sequence character sets:
  - `\d` matches any digit (same as `[0-9]`); `\D` any non-digit (`[^0-9]`)
  - `\w` matches any "word character" (same as `[a-zA-Z_0-9]`); `\W` any non-word char
  - `\s` matches any whitespace character (, `\t`, `\n`, etc.); `\S` any non-whitespace
- What regular expression matches dollar amounts of at least $100.00 ?
  - `/\$\d{3,}\.\d{2}/`

Regular expressions in PHP (PDF)

- regex syntax: strings that begin and end with `/`, such as `"/[AEIOU]+/"`

function	description
`preg_match(`*regex*`, `*string*`)`	returns `TRUE` if *string* matches *regex*
`preg_replace(`*regex*`, `*replacement*`, `*string*`)`	returns a new string with all substrings that match *regex* replaced by *replacement*
`preg_split(`*regex*`, `*string*`)`	returns an array of strings from given *string* broken apart using given *regex* as delimiter (like `explode` but more powerful)

PHP form validation w/ regexes

```
$state = $_POST["state"];
if (!preg_match("/^[A-Z]{2}$/", $state)) {
    print "Error, invalid state submitted.";
}
```
PHP

- `preg_match` and regexes help you to validate parameters
- sites often *don't* want to give a descriptive error message here (why?)

Regular expression PHP example

```php
# replace vowels with stars
$str = "the quick    brown        fox";

$str = preg_replace("/[aeiou]/", "*", $str);
                         # "th* q**ck    br*wn        f*x"

# break apart into words
$words = preg_split("/[ ]+/", $str);
                         # ("th*", "q**ck", "br*wn", "f*x")

# capitalize words that had 2+ consecutive vowels
for ($i = 0; $i < count($words); $i++) {
    if (preg_match("/\\*{2,}/", $words[$i])) {
        $words[$i] = strtoupper($words[$i]);
    }
}                        # ("th*", "Q**CK", "br*wn", "f*x")
```

PHP

- notice how \ must be escaped to \\