Lecture 4: Forms & Regular Expressions

CS472 Web Programming

Maharishi University of Management

Department of Computer Science

Assistant Professor Asaad Saad

Forms

In this lecture we will discuss how to generate and process user input. On the client side we will create HTML forms with different types of widgets that allow the users to submit different types of data, and on the server side we will look at processing these different types of data. *Life is found in layers.*

Query strings and parameters

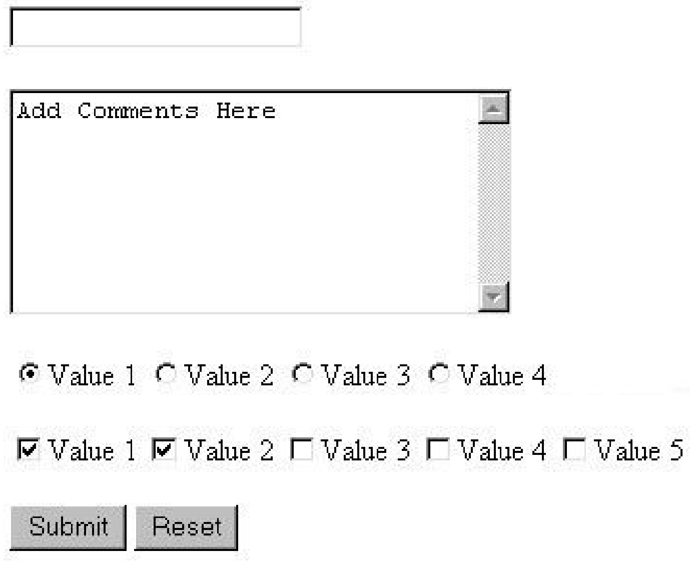
http://www.google.com/search?q=Obama
http://mum.edu/login.jsp?username=asaad&sid=1234567

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 “asaad”, and **sid** has value “1234567”

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>`**

The <**form**> tag is used to create an HTML form for user input.

The <**form**> element can contain one or more of the following form elements:
<**input**>, <**textarea**>, <**button**>, <**select**>, <**option**>, <**optgroup**>, <**fieldset**>, <**label**>

```
<form action="" method ="" enctype="" novalidate autocomplete>
    form controls
</form>
```

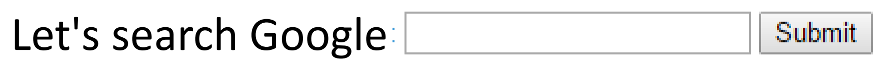
- **action** destination URL
- **method** get, post
- **enctype** application/x-www-form-urlencoded, multipart/form-data, text/plain
- **novalidate** (HTML5) specifies that the form should not be validated when submitted
- **autocomplete** (HTML5) on, off

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
  - **`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
- **`POST`** : submits data to a web server and retrieves the server's response
  - parameters are embedded in the request's HTTP packet, not the URL

Form Example

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



Main Point

An HTML form allows the user to send data (parameters) to the server. Forms are created with the <form> tag, and can be submitted with either an HTTP GET or POST method. *Every action has a reaction.*

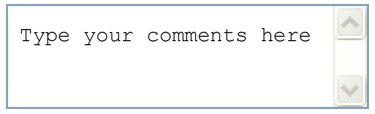
<input />

- **`input`** element is used to create inline input element and MUST be self-closed
- **`name`** attribute specifies name of query parameter to pass to server
- **`type`** attribute can be button, checkbox, color, date, datetime, datetime-local, email, file, hidden, image, month, number, password, radio, range, reset, search, submit, tel, text, time, url, week
- **`value`** attribute specifies control's initial text
- **`required`** `(HTML5)` if the input field must be filled out before submitting the form
- **`placeholder`** `(HTML5)` a short hint that describes the expected value of an <input> element
- **`pattern`** `(HTML5)` a regular expression that an <input> element's value is checked against
- **`size`** the width, in characters, of an <input> element
- **`maxlength`** the maximum number of characters allowed in an <input> element
- **`readonly`**

`<textarea>`

- The <**`textarea`**> tag defines a multi-line text input control.
- A textarea can hold an unlimited number of characters, and the text renders in a fixed-width font (usually Courier).
- The size of a textarea can be specified by the **`cols`** and **`rows`** attributes, or even better; through CSS' **`height`** and **`width`** properties.

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

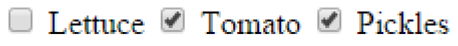

Type your comments here

Checkboxes

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

- none, 1, or many checkboxes can be checked at same time
- Use the `checked` attribute in HTML to initially check the box, the empty value syntax is equivalent to checked=""

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

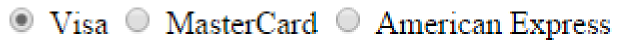

☐ Lettuce ✓ Tomato ✓ Pickles

Radio buttons

Sets of mutually exclusive choices (**inline**)

- 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`**

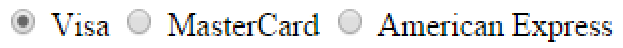
```
<input type="radio" name="cc" value="visa" checked /> Visa
<input type="radio" name="cc" value="mastercard" /> MasterCard
<input type="radio" name="cc" value="amex" /> American Express
```



Text labels: `<label>`

- 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

```
<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>
```

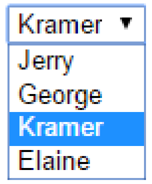


Drop-down list `<select>` and `<option>`

Menus of choices that collapse and expand (inline)

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

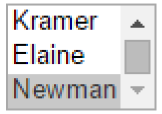
```
<select name="favoritecharacter">
    <option>Jerry</option>
    <option>George</option>
    <option selected>Kramer</option>
    <option>Elaine</option>
</select>
```



Using `<select>` for lists

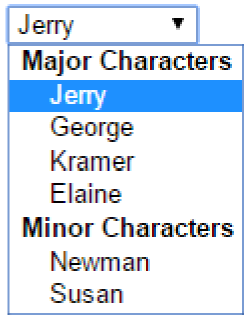
- optional **`multiple`** attribute allows selecting multiple items with shift- or ctrl-click
  - must declare parameter's name with [] if you allow multiple selections
- **`option`** tags can be set to be initially **`selected`**

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



Option groups: `<optgroup>`

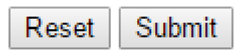
```
<select name="favoritecharacter">
    <optgroup label="Major Characters">
        <option>Jerry</option>
        <option>George</option>
        <option>Kramer</option>
        <option>Elaine</option>
    </optgroup>
    <optgroup label="Minor Characters">
        <option>Newman</option>
       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



Reset and Submit buttons

- When we click **`reset`** button, it returns all form controls to their initial values
- When we click **`submit`** buttons, it sends all data with the specified **`method`** (Get/Post) to the **`action`** page in the form
- Specify custom text on the button by setting its **`value`** attribute

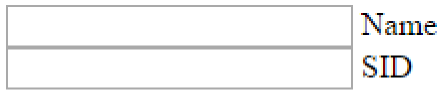
```
<input type="reset" />
<input type="submit" />
```



Hidden input parameters

An invisible parameter that is still passed to the server when form is submitted, it’s useful for passing on additional state that isn't modified by the user

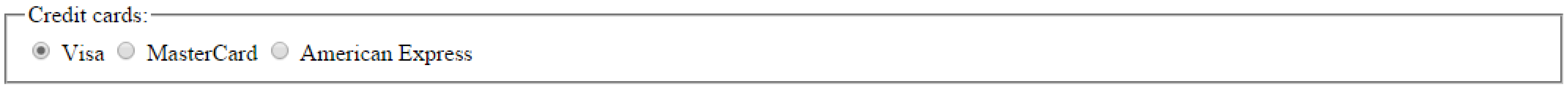
```
<input type="text" name="username" /> Name <br />
<input type="text" name="sid" /> SID <br />
<input type="hidden" name="school" value="MUM" />
<input type="hidden" name="year" value="2048" />
```



Grouping `<fieldset>, <legend>`

Groups of input fields with optional caption (block)

```
<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>
```


Credit cards:

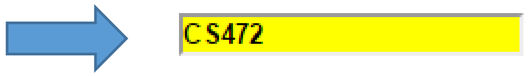
● Visa ○ MasterCard ○ American Express

Styling form controls

Because most input element are created using input tag, we target each group of elements using this CSS selector:

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

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



Main Point

HTML provides many different types of input widgets, including text fields, text areas, check boxes, radio buttons, and dropdown lists, this is also an area html 5 is expanding. *Harmony exists in diversity.*

pattern

The **pattern** attribute specifies a regular expression that the <**input**> element's value is checked against. The **pattern** uses the ECMAScript (i.e. Javascript) flavor of regex.

Note: The **pattern** attribute works with the following input types: **text**, **date**, **search**, **url**, **tel**, **email**, and **password**.

Tip: Use the global **title** attribute to describe the pattern to help the user.

```
<form action="demo_form.jsp">
      Country code:
      <input type="text" name="country_code" pattern="[A-Za-z]{3}“
                                           title="Three letter country code">
      <input type="submit">
</form>
```

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
- Regular expressions are extremely powerful but tough to read
  - (the above regular expression matches email addresses)
- Regular expressions are used in all languages:
  - Java, PHP ,JavaScript, HTML, C#, and other languages
- Many IDEs allow regexes in search/replace

Basic regular expressions

The simplest regexes simply matches any string that contains that text.

```
abc
```

The above regular expression matches any string containing "abc":

- YES: "abc", "abcdef", "defabc", ".=.abc.=.", ...
- NO: " ABC" , " fedcba", "ab c", "PHP", ...

- Regular expressions are case-sensitive by default.

Wildcards .

A dot `.` matches exactly **one-character** except a `\n` line break

`.oo.y` matches "Doocy", "goofy", "LooNy", ...

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"

The html5 spec states that ^ and & are implicit

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}
```

Advanced: Lookaround

Positive lookahead `(?=A)B`

Once a group starts with `?=` it means positive lookahead.

Find expression A first, if found then expression B follows.

Negative lookahead `(?!A)B`

Once a group starts with `?!` it means negative lookahead.

First check if expression A is not found, then check if expression B follows.

Examples

An <input> element with type="password" that must contain 8 or more characters that are of at least one number, and one uppercase and lowercase letter:

```
<form action="demo_form.jsp">
     Password:
     <input type="password" name="pw"
            pattern="(?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{8,}"
            title="Must contain at least one number and one uppercase and lowercase letter, and at least 8 or more characters">
     <input type="submit">
</form>
```

An <input> element with type="email" that must be in the following order: characters@characters.domain (characters followed by an @ sign, followed by more characters, and then a "."

```
<form action="demo_form.jsp">
     E-mail:
     <input type="email" name="email"
            pattern="[a-z0-9._+-]+@[a-z0-9.-]+\.[a-z]{2,3}">
     <input type="submit">
</form>
```

Examples

An <input> element with type="search" that CANNOT contain the following characters: ' or "

```
<form action="demo_form.jsp">
     Search: <!-- x27 is a single quote and \x22 is a double quote -->
     <input type="search" name="search"
            pattern="[^\x27\x22]+" title="Invalid input">
     <input type="submit">
</form>
```

An <input> element with type="url" that must start with http:// or https:// followed by at least one character:

```
<form action="demo_form.jsp">
     Homepage:
     <input type="url" name="website"
            pattern="https?://.+" title="Include http://">
     <input type="submit">
</form>
```

Bonus: Default page - index.html

How does your server know to call **`index.html`** file when you don't specify the file name?

```
http://www.mumstudents.org/~123456/
```

In Apache server, we have **`httpd.conf`** configurations file where we can update **`DirectoryIndex`** property and specify the order of our default pages, if none exists, then the server will throw a Forbidden error.

```
DirectoryIndex index.html index.php
```

Sessions and Cookies

- HTTP protocol is stateless, there is no way for the server to know the client if that client tried to send another request!
- A cookie is a small file located on our machines created by browsers. It’s a single key/value pair. Usually servers ask the browser to save this key/value information on clients machines.
- Browsers send all website related cookies with every request they make.
- All backend languages (on the server) have sessions, A session is a way for servers to save a temporary data (15-30 mins) like user information so they can recognize the user again.
- Usually some of the session information are saved as cookies on the client machine.

CONNECTING THE PARTS OF KNOWLEDGE WITH THE WHOLENESS OF KNOWLEDGE

Forms and Regular Expressions

1. It is a best practice to validate all data
2. The easiest way to write validation logic is with regular expressions

---

1. **Transcendental consciousness** is the ultimate in robustness and simplicity, as it is the unchanging silence at the basis of creation.
2. **Impulses within the Transcendental field:** all actions are achieved by the interplay of unity, by understanding the principles by which unity becomes diversity we can gain complete understanding of the relative.
3. **Wholeness moving within itself:** In Unity Consciousness, one experiences that both the unchanging silence of the absolute and the never ending diversity of the relative are the Self.