# CS316 class notes

Raphael Finkel

April 21, 2022

## 1 Intro

- Class 1: 1/21/2022
- Handout 1 — My names
  - Mr. / Dr. / Professor / —
  - Raphael / Rafi / Refoyl
  - Finkel / Goldstein
- Plagiarism — read aloud from handout 1
- Assignments on web.
- E-mail list: [class email address]; instructor uses to reach students.
- All students have MultiLab accounts, although you may use any computer you like to do assignments. MultiLab machines, such as `cor.cs.uky.edu` and `pen.cs.uky.edu`, are behind a firewall, so if you are off campus, you must establish a Virtual Private Network (VPN) to access them.
- All students have MySQL accounts, which will be useful later in the semester.

## 2 History

- ArpaNet: around 1973

- Internet Protocol (IP) provides a way to name every computer by a 32-bit identifier, conventionally written as four 8-bit integers in base 10, such as `128.163.146.21`. (That's IPv4; IPv6 is different.)
- Transfer-control protocol (TCP) (Vinton Cerf and Robert Kahn) around 1974, sits above IP and provides reliable, in-order transmission of data through a network. (Behind the scenes: establishing a connection, serial numbers, positive acknowledgements, timeouts, retransmissions)
- Application-level protocols are built above TCP
  - File Transfer Protocol (FTP: Abhay Bhushan, 1971)
  - Simple Mail Transfer Protocol (SMTP: Ray Tomlinson 1971)
  - Network Time Protocol (NTP: David Mills, 1985)
  - Hypertext Transfer Protocol (HTTP: Tim Berners-Lee, 1989)
  - Domain Name System (DNS: Paul Mockapetris 1983)
  - Secure Shell protocol (SSH: Tatu Ylönen 1995)
- Many protocols are officially specified in RFC ("Request For Comment") documents stored by the Internet Engineering Task Force (IETF).

# 3 Overall structure of Web applications

- We will mostly consider Web programs based on **clients** communicating with **web servers** using HTTP.
  - The **client** is typically a **browser**, such as *Firefox*, *Opera*, *Safari*, or *Chrome*, running as a program on the **user**'s computer. It can also be a non-interactive program like *curl* or *wget*.
  - The **web server** is a typically a daemon (long-running unattended) program such as Apache, Internet Information Services (MicroSoft IIS), Tomcat, or nginx ("engine-X") running on an host computer.
  - The CS web server (`www.cs.uky.edu`) runs Apache; it is hosted on `bud.cs.uky.edu`.
- The client sends **requests** to the web server using HTTP, and the web server sends **responses** back to the client, again using HTTP.

- The web server does not maintain state between requests; every request/response pair is self-contained.

# 4 A deeper look

- The web server can ask the client to store opaque data called **cookies** and present them back to the web server on subsequent requests; the web server can use the contents of the cookies to verify authentication and look up stored history.
- Browser requests are for **pages**.
- The web server interprets a typical page request as naming a file to send back as the response, but if the file has the necessary permissions, the web server treats the file as a program and packages its output into the response.
- The web server generally doesn't try to interpret the content of its responses. It's up to the client to interpret that content. In particular, web servers don't understand Hypertext Markup Language (HTML), but browsers do, and they parse the HTML content to format and present pages to the user.
- The user can explicitly specify a request by a Universal Resource Locator (URL) of this form:
  `protocol://hostName/file/path?parameters`
  - `protocol`: often `http` or `https`. The latter works the same, but the protocol includes cryptographic means to authenticate the web server (so the client can be sure of its identity) and a guarantee of privacy and integrity of the communication.
  - `hostName`: Either an IP name, like `128.163.146.21`, or a DNS name, like `www.cs.uky.edu`. Either of these host names refers to our CS Department web server, which can access MultiLab files. The `hostName` can include an optional colon followed by a **port number**; if there is no port number, the default port (80 for `http`, 443 for `https`) is implied.
  - `file/path`: This component can be of any length.
  - `parameters`: The web server provides this optional component to the program (if any) that is named by the `file/path` component. Under Unix, it does so by placing the parameters

in the environment of the program or as input to the program. The format of the parameters component is typically of the form `var1=value1&var2=value2` ... .

- Web pages often contain URLs in **links** that a user may click, which initiates a browser action very similar to manually entering a URL in the appropriate area in the browser window.

# 5 Unix: files and permissions

- Class 2: 1/13/2022
- Every file and directory in Unix is associated with (among other attributes) the **owner**, the **group owner**, and nine permissions: (read, write, and execute) for each of (owner, group, and other).
- These permissions are often represented as three octal numbers representing nine bits. For instance, `764` represents `111-110-100`, which in turn means the owner may read, write, and execute the file; failing that, members of the group may read and write (but not execute) the file; failing that, other users (the "world") may read but not write or execute the file.
- Many programs in Unix, such as `ls` and `chmod`, represent the same information in nine letters, such as `rwxrw-r--` .
- If a file is not executable but is readable (and all its ancestor directories are world-executable), the web server responds to a URL representing that file by sending the file (with suitable HTTP headers) back to the browser. We call that file a **static** page.
- Our CS web-server configuration stipulates that if the file is executable by the owner, not writable by anyone else, and is accessible to the web server (permissions at least `--x---r--`), the web server invokes that file as a program running under the owner's privileges and sends its output (which is typically HTML preceded by HTTP headers) as the response to the request. We call that situation a **dynamic** page.
- Say your login name is *abc123@uky.edu*. You can login to any MultiLab machine (such as `pen.cs.uky.edu`; you need to use the campus VPN) with login name simply *abc123*. You can say `ssh abc123@pen.cs.uky.edu` to make the connection.

- In your home directory, you need a subdirectory called `HTML`, with suggested permissions `751`. If you have a readable file in that subdirectory called `foo.html`, the URL that accesses it (from anywhere in the world) is `http://www.cs.uky.edu/~abc123/foo.html`.
- There are other ways to make a dynamic page, in particular, **server-side includes**, in which the web server substitutes dynamic information such as today's date for specific codes in an otherwise static page.

# 6 Hypertext Markup Language (HTML)

- All browsers can display pages encoded in HTML.
- HTML is derived from a more general scheme, Extensible Markup Language (XML).
- HTML has gone through several generations; currently, HTML means HTML5, released in 2014. A 6th version is in progress; many browsers already support some of its new features. See `http://html6test.com/index.html`.
- HTML pages are text with additional "markup" to control formatting.
- The markup is in the form of **tags**, which are always surrounded by brokets (angle brackets). Example: `<span>`.
- Most markup encloses a region of text. Tags for that sort of markup have closing tags to indicate the end of the region Closing tags are also surrounded by brokets; they start with '`/`'. Example: `</span>`.
- Tagged regions may nest (subject to some constraints), but they may not overlap.
- A properly constructed HTML document contains some required tags; other tags are optional.
- Example 1
- Browsers are very forgiving of missing closing tags, often inserting them automatically.
- Example 2

# 7 Browser developer tools

- Whenever you build a web page, use the browser to check for warnings. Open the **developer tools**: Control-Shift-I on FireFox.
- The layout and invocation method varies among Firefox, Chrome, and others.
- Inspector: see the HTML.
- Console: See JavaScript warnings and run JavaScript code directly.
- Network: See what pages are loaded, where they come from, and how long access took.
- HTML validator: see errors and warnings; send the page to a web service for more thorough validation; automatically clean up the HTML.

# 8 HTML tags

- Full list: `https://www.w3schools.com/tags/default.asp`
- Example that has all the tags mentioned here: Example 3
- Required regions
  - `<html>`
  - `<head>` (within `<html>`)
  - `<title>` (within `<head>`)
  - `<body>` (within `<html>`)
- Comments: `<!-- ... -->`
- Regions inside `<head>`
  - `<style>` style sheet organized by tag
  - `<script>` JavaScript program
- Non-regions inside `<head>`
  - `<meta>` metadata: character set, keywords, viewports
  - `<link>` reference to external style sheet

- Class 3: 1/18/2022
- Regions inside `<body>`, generally nestable; closing tag is sometimes optional.
  - `<h1>` .. `<h6>` headings of various sizes
  - `<p>` paragraph
  - `<a>` anchor; typically a **hyperlink**
  - `<span>` inline text region
  - `<div>` vertically separate text region
  - `<pre>` pre-formatted text
  - `<table>` table
  - `<tr>` table row (within `<table>`)
  - `<td>` table entry (within `<tr>`)
  - `<ul>` unnumbered list
  - `<ol>` ordered (numbered) list
  - `<li>` list element (within `ol` or `ul`)
- Non-regions inside `<body>`; nothing nests inside
  - `<img>` image
  - `<br>` line break
- We will see other tags later.

# 9 HTML attributes

- **Attributes** modify the way browsers present tags or provide essential additional information.
- Attributes are placed inside the brokets delimiting a tag and have this syntax:
  `tagname='tag value'`
- One may use single or double quotes; if the tag value is a single word, quotes are optional. Don't use fancy quotes; use only ASCII single quote and double quote. Tag values must not contain any embedded HTML tags.
- Each tag has a defined set of **attributes** it can accept, and each attribute has a defined syntax for its tag values. (For instance, the `width` attribute for `<img>` has a value like `20`, not `20px`.)

- Some attributes are obligatory.
  - `<img>` requires `src`.
- Some attributes are highly recommended.
  - `<html>` should have `lang`.
  - `<img>` should have `alt`.
- The most common non-obligatory attributes
  - `id` uniquely names the tagged region so it can be referred to elsewhere.
  - `style` modifies how the browser presents the region.
  - `class` identifies regions that share formatting.
- Browsers ignore unrecognized attributes, but the validator notices them.

# 10 Styles

- The philosophy is that HTML tags describe **content**; styles describe **presentation**.
- Syntax: `property:value;` (Spaces are optional.)
- A single element may have multiple style components.
- There are hundreds of style properties; we only discuss a few.
- Text modifications
  - `color`: a name, like `green`, a hex value, like `#00FF00`, or an RGB value, like `rgb(0,255,0)`
  - `background-color`: similar to `color`
  - `text-align`: `center`, `left`, `right`, or `justify`.
  - `vertical-align`: `top`, `middle`, or `bottom`
  - `text-indent`: a length, such as `20px`
- Font modifications
  - `font-family`: for instance, `"Times New Roman", Times, serif`

  - `font-style`: `normal`, `italic`, or `oblique`
  - `font-weight`: `normal` or `bold`

- Box model: each region is in a box with internal **padding**, a **border**, and an external **margin**, all of which can be styled.

  - `padding`: a length, such as `5px`
  - `border`: such as `12px solid blue`
  - `margin`: a length, such as `10px`
  - each of these can be specified separately for `top`, `bottom`, `left`, and `right`, such as `margin-top:10px;`

# 11 Cascading Style Sheets (CSS)

- Class 4: 1/20/2022
- The `style` attribute on a tag is useful when a single element needs special formatting.
- If many elements need the same formatting, there is a better approach: a `<style>` section within the `<head>` section with a tag-specific or class-specific set of formatting rules.
- The format of styles in the `<style>` subsection:
  *selector* {*style* ...}
- Example 3 from earlier has styles, both inline and in a `style` section.
- The *selector* can be (this list is not exhaustive)

  - a tag name (such as `p` or `h3`)
  - a class name, with a prepended dot, such as `.myRed`, which applies to all tags in the page that have the attribute `class="myRed"`
  - an id name, with a prepended hash, such as `#myRed`, which applies to the unique tag in the page that has the attribute `id="myRed"`.
  - (seldom needed:) a combination of the above possibilities separated by operators for descendant, child, adjacent sibling, general sibling.

- Browsers apply a cascading set of rules to determine formatting; that's why style sheets are called Cascading Style Sheets (CSS). The rules can be complicated, but this is a reasonable summary:
  - For each property, the browser uses the first rule it finds.
  - Step 1: the `style` attribute in the element's tag.
  - Step 2: an entry in the `<style>` region of the document targeting the element's id (such as `#myRed`), class (such as `.myRed`), or tag (such as `p`), and in that order.
  - Step 3: repeat steps 1 and 2 for each surrounding tag, moving from closest (local) to farthest (global).
  - Any inherited property value that is marked `!important` takes precedence over this usual order.
- You can discover the CSS for any element by using the inspector in the browser's developer tools.
- Example 4: Nested regions with styles.

# 12 Common Gateway Interface (CGI)

- CGI specifies how browsers send requests to web servers in response to user actions (typically, clicking on a *submit* button).
- The simplest invocation is via a `<form>` tag in the web page. Steps:
  - The user clicks a button, **activating** the form.
  - The browser collects values of the **input fields** from the form and sends them to the web server along with a URL that refers to an executable file (that is, a program).
  - The web server presents the input values to the specified program, which we call a **CGI-bin program** or simply a **CGI program**.
  - The CGI program produces output comprised of an HTTP header and HTML content.
  - The web server sends that output to the browser as the response.
  - The browser displays the new web page.
- Example 5 invokes `echo.cgi`.
- Class 5: 1/25/2022

- I have added a simple audio and video entry to this example. Accessing media: `<video>` (only MP4, WebM, and Ogg) `<audio>` (only MP3, Ogg, and WAV) Safari does not play Ogg.
- Notice the extra care that *echo.cgi* takes to prevent code-injection attacks. Disable that code and try this input:
  ```
  red<span style='color:green;'>green</span>
  ```
- There are worse attacks. Always untaint inputs that are used to generate outputs of CGI programs.
- One can also call the CGI program directly in a fancy URL: `www.cs.uky.edu/~raphael/courses/CS316/examples/echo.cgi?parameter1=hello`
- The CGI program
  - The CGI program may be written in any language.
  - Many languages provide libraries for accessing the CGI parameters.
  - The permissions (on Unix) of all the directories in the path from the file-system root to the program must be *executable* by *other* and must not be writable except by *user*.
  - The permissions on the CGI program file itself must be executable by *owner* (which must be the same as the user under whose HTML directory the program is located) and not writable by anyone except by *owner*.
  - The CS web server runs the CGI program under the owner's identity, so it has access to anything the owner can access.
  - The CS web-server configuration requires that the CGI file have a name with suffix `.cgi` or `.php`.
  - A Unix convention lets the first line of a text CGI file specify what programming language software is meant to interpret it. That is why `#!/usr/bin/perl` is at the top of the executable text file *echo.cgi*.
- The web server invokes the CGI program after setting several environment variables. Among them:
  - `REQUEST_METHOD`: either `GET` or `POST`.
    - `GET` applies to requests directly in the URL and to requests coming from `<form>` regions that specify it.

  - `GET` responses can be cached; the expectation is that the same request again should produce the same response.
  - It's usually better to use `POST`, which sends the parameters in a separate message, where they are less subject to manipulation and leaking information (like passwords).
  - There are also `PUT` and `DELETE` methods, but they are rarely used.
- `QUERY_STRING`: the parameters to a `GET` request, separated by the `&` or `;` symbol.
- `CONTENT_LENGTH`: the length of the parameter list passed to standard input for a `POST` request. This list has the same format as the `QUERY_STRING`.

# 13 Input tags

- A full list of input tags for forms is available at `www.w3schools.com/html/html_form_input_types.asp`.
- Example 6 uses many of them.

# 14 Command-line, non-interactive clients

- *curl*:

```
curl -d param1='hello world' -d param2='goodbye world' \
http://www.cs.uky.edu/~raphael/courses/CS316/examples/fullEcho.cgi
```

- *wget*:

```
wget --post-data='param1=hello world&param2=goodbye world' -O - \
http://www.cs.uky.edu/~raphael/courses/CS316/examples/fullEcho.cgi
```

- Both have many options, such as providing cookies, using `GET` instead of `POST`, and logging.
- Since all accessible CGI programs can be invoked by these clients, you cannot rely on the browser to constrain the values of fields such as `date`. The CGI program must be cautious.

# 15 Languages for CGI programs

- Class 6: 1/27/2022
- Many languages make it easier for CGI programs to get the parameters, direct runtime errors to the browser, and fill in HTML templates.

| language | parameters | errors | templates |
|---|---|---|---|
| Perl | `CGI` | `CGI::Carp` | `HTML::Template` |
| Python | `cgi` | `cgitb` | `Chameleon`, *Flask* |
| C/C++ | `cgicc` | | `ClearSilver` |
| Java | | | `jsp` |
| PHP | built-in | | built-in |
| Ruby | | | ERB |

- Example 7: Perl. Perl predates the WorldWideWeb by a few years.
- PHP: PHP Hypertext Preprocessor
  - `PHP` is a scripting language based on a built-in template facility.
  - A PHP manual is at `https://www.php.net/manual/en/`; a simplified description is at `https://www.w3schools.com/php/default.asp`.
  - The program looks like an HTML web page decorated with commands, all of the form `<?php ... ?>`
  - Example 8: PHP example.
  - Unfortunately, it's hard to debug PHP programs.
  - It has many functions for file manipulation, gathering `<form>` data, handling cookies, interacting with databases, and encryption.
  - To check syntax: `php -l FILENAME.php`
  - To provide GET or POST parameters to a PHP program run from the command line, put the following near the top of the PHP code:

```
1 /* If started from the command line, wrap parameters */
2 if (!isset($_SERVER["HTTP_HOST"])) {
3    parse_str($argv[1], $_GET);
4    parse_str($argv[1], $_POST);
5    parse_str($argv[1], $_REQUEST);
6 }
```

Then you can invoke the program from the command line this way:
`php FILENAME.php 'param1=value1&param2=value2'`

- Debugging
  - Always run your CGI programs on the command line first to check for errors. There are ways to provide parameters, too.
  - There is an error log on `www.cs.uky.edu`, but you don't have access to it.
  - You can often have the CGI program report errors as part of the web page it returns; in Perl, you can say
    `use CGI::Carp qw(fatalsToBrowser warningsToBrowser);`
  - Unix standard line termination is \n, but for MicroSoft it's \r\n. You might need to remove extraneous \r characters so that the first line (which looks like `#!/path/to/processor`) ends with \n.
  - For our assignments, you must test your code on the MultiLab to make sure that any languages and libraries your code requires are available.
  - In-class exercise: Modify example 8 to include Saturday and Sunday and add one more message to the set of randomly-selected ones.
  - See this solution.

# 16 A little more about tables

- Class 7: 2/1/2022
- Table: colspan and rowspan
- Example 9: Table example.

# 17 Local vs remote execution

- **Local execution**: In the browser.
- $\oplus$: offloads work from the web server, can avoid sending private information via the net.

- $\ominus$: dependent on version of tools used, such as JavaScript and libraries, which must be available to the browser. (Example: web applets in Java used to work, but they don't anymore.)
- **Remote execution**: In the web server (via CGI).
- $\oplus$: uses full power of server; not dependent on particular browser.
- $\ominus$: less portable, puts load on the web server.

# 18 JavaScript

- Officially called ECMAScript (European Computer Manufacturers Association): first released 1997.
- High-level, just-in-time compiled, looks like C or Java, but is in many ways different from both.
- Advanced features: dynamic typing, prototype-based object orientation, first-class functions, anonymous functions.
- All standard browsers can run JavaScript programs.
- One can write standalone programs in JavaScript, too; the processor is called *nodejs*. See `https://nodejs.org/docs/v8.10.0/api/`.
- JavaScript is quite error-prone, although more recent versions are less so.
- Example 10: JavaScript
- Class 8: 2/3/2022
- Snow day; only a Zoom review session
- Class 9: 2/8/2022
- Differences between *nodejs* and JavaScript
    - JavaScript runs inside the browser in a sandbox: no access to the file system or to programs outside the browser. It understands the DOM (Document Object Model; see Section 20 on page 17) and provides variables and methods specific to the DOM.
    - *Nodejs* runs on its own, with full access to the file system and to external programs. It does not have the DOM, although one can bring in a module (*jsdom*) that does have the DOM.

# 19 Database access: SQL

- Databases (DBs) are the subject of CS405; we only touch the surface here.
- DB servers include mySQL, MariaDB, SQL Server, MS Access, Oracle, Sybase, Informix, Postgres.
- These all support **relational** DBs: information is organized into **relations**, also called **tables**, with a **schema** detailing the columns and their types. Each **entry** is a row in a DB table.
- The standard language for communicating with a relational DB is SQL (Structured Query Language), with commands for storing, manipulating, and retrieving data.
- The highly popular SQLite C library "implements a small, fast, self-contained, high-reliability, full-featured, SQL database engine."
- SQL Reference.
- A standard organization is for the browser (the client) to communicate with a controller (a CGI program) that accesses a DB (the back end) and provides feedback to the browser.
- Many languages (like Perl, PHP, Java) have libraries intended for interacting with DBs, often called **Database Connectors**, such as JDBC (Java) PHP Data Objects (PDOs: PHP), and DBI (Perl).
- The database connector provides a way to safely pass values to the DB without worrying that the value might include an SQL insertion, typically by a two-step process.

  - Prepare the SQL statement, which uses **?** in place of all values to be inserted or modified in the database.
  - Execute the prepared statement, providing all the actual values to associate with the **?** references.

  The database connector makes sure that the actual values are stored verbatim and are never treated as SQL statements themselves.
- See `https://xkcd.com/327/` for a funny use of SQL injection.
- Example 11: SQL for Kratylos

# 20 Document Object Model (DOM)

- A JavaScript program running in a page in the browser can inspect, modify, insert, and delete any HTML and CSS on that page.
- Class 10: 2/10/2022
- The HTML is represented by a hierarchy of objects, and there are built-in methods that return objects based on CSS selectors, such as tag, id, name, and class.
- The DOM is pre-declared for JavaScript, including both variables and functions.
- `https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction` Reference
- Example 12, to be run in the developer console for Example 4.
- Typically, a JavaScript program is invoked as a side-effect of clicking a button, but there are other triggers as well, including interaction with the development console.
- Many examples: `https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Examples`.
- Exercise (in class): In the developer console, write a routine that causes the font size of the `inner` class to grow ten times.
- See this solution.

# 21 Scripts inside HTML

- Class 11: 2/15/2022
- The browser runs scripts when it loads the page.
- The `<script>` tags can be anywhere, but they are typically in the `<head>` region or the `<body>` region; the latter postpones execution until the body is loaded.
- The browser also invokes scripts as a side-effect of **events**, which can be specified as attributes of tags.
    - `onclick='some JavaScript code'`. The code is typically a function call. The parameter value `this` is the element itself in the DOM.

    - Besides `onclick`, there are many other triggers, such as `onmouseover`, `onmousedown`, `onmouseup`, `onfocus`.

- A script can bind new event handlers to elements.
- Example 13: Invoking scripts.
- Exercise (in class): Write a web page that includes JavaScript that takes two integers in input regions and displays their sum. This page should not use CGI; all computation should be in the browser.
- See this solution.

# 22 Asynchronous JavaScript and XML: AJAX

- Class 12: 2/17/2022
- Allows code on a web server to update a page without the browser needing to reload it.
- Simple idea.
    - The web page includes JavaScript that builds a request, sends it to the web server, awaits a response, and acts on the response.
    - The message is encapsulated in a `XMLHttpRequest` object. Since about 2017, this awkward construction was replaced with the far simpler `fetch()` routine.
    - The message is sent to a URL on the same server as the one that provided the page. That URL can refer to a file (which can be executable).
    - The web server returns the file (or invokes it as a program, giving it any optional information in the `fetch` parameters.
    - The JavaScript in the web page interprets the result and uses it to update the contents of the web page.
- Purposes
    - Update a web page without reloading. Example: suggesting valid completions of partial words in a form.
    - Request and receive data from the web server after the page is loaded.
    - Send data to the server in the background.

- Example 14: AJAX.
- AJAX transmits *asynchronously* to avoid blocking and to allow other scripts to continue while the JavaScript waits.
- Therefore the result of `fetch()` is a *promise*; the JavaScript code can **`await`** its eventual content.
- The code can access its content by `response.text()`.
- If the response text is a string comprising valid JSON, `JSON.parse()` can convert it to a JavaScript object. However, the syntax of the JSON string is very restrictive: every name and every string value must be surrounded by double quotes; single, missing, or non-ASCII quotes are not acceptable. Alternatively, the code can access the content by `response.json()`, but only if it hasn't "consumed" the content by `response.text()`.
- Efficiency and security
  - Every AJAX request generates a round-trip message on the network, which itself requires multiple underlying TCP packets.
  - The web server must handle each AJAX request, often calling a program.
  - The browser usually does not require CPU while a request is pending.
  - The browser refuses to send AJAX *except* to the web server generating the current page (`window.location.host`) to avoid leaking sensitive information.
  - The user can inspect and manipulate the web page, both content and script, so the program on the web server must not trust the formatting or content of the data it receives in an AJAX request. The web server must never send secrets to the browser, because they are then visible to the user.
- Exercise (in class): Write a web page with a button that, when clicked, causes an AJAX call to `http://www.cs.uky.edu/~raphael/courses/CS316/examples/example.07.cgi`, extracts the date from the response, and displays it.
- See this solution.
- Class 13: 2/22/2022

- To pass form parameters in `fetch()`:
  - GET method (the default): add the parameters to the URL in the first parameter of `fetch()`.
  - POST method: include a second parameter to `fetch()`:

```
{
   method: "POST",
   headers: new Headers({
      'Content-Type': 'application/x-www-form-urlencoded',
   }),
   body: "param1=value1&param2=value2...",
}
```

- To return a JSON string from CGI to AJAX, you can (optionally) have the CGI program set the HTML response header `Content-Type: application/json; charset=utf-8`. Make sure that all field names are quoted (with ordinary, not smart, double quotes). In PHP, use the `header()` function to get this non-default behavior.

# 23 JQuery

- JQuery, first released in 2007, is a JavaScript **framework**, that is, a library that abstracts the DOM, makes JavaScript programming easier and uniform across browsers, and acts as a basis for many other JavaScript libraries. JQuery was especially important before JavaScript introduced `querySelector()` and `fetch()`.
- The current version (released 3/2021) is 3.6.0.
- Features
  - Avoiding browser-specific differences.
  - Manipulating the DOM and CSS
  - Binding events to code
  - Handling effects and animations
  - Simplifying AJAX
  - Enabling third-party plugins for many other tasks, such as data tables.

- To enable JQuery, either download a personal copy from the official site, `https://jquery.com/` or link to a version provided by a content-delivery network (CDN) such as Cloudflare or Google. In either case, link to it in the `<head>` or at the end of the `<body>` section:

```
<script
  src="https://code.jquery.com/jquery-3.6.0.min.js"
  integrity="sha384-vtXRMe3mGCbOeY7l30aIg8H9p3GdeSe4IFlP6G8JMa7o7lXvnz3GFKzPxzJdPfGK"
  crossorigin="anonymous"></script>
```

  This code uses a "minified" version and checks that it hasn't changed. The "anonymous" `crossorigin` prevents sending cookies.

- JQuery provides a third way to access the DOM (the first way is direct JavaScript, the second is XPath).
  - All DOM references return objects that respond to further JQuery methods.
  - Simplest: `$(`*`node`*`)` turns any DOM node into a JQuery object.
  - Objects are usually specified by a **selector**: `$("`*`selector`*`")`
  - Multiple space-separated selectors are valid; each selects descendants of the objects specified by the previous selectors.
  - Multiple comma-separated selectors are valid; the result is the union of all selected objects.
  - The selectors include all CSS selectors (which also work for `querySelector()`.
  - Selector based on any attribute value: `$("[`*`name=value`*`]")`. There are variations on this form to select only those elements whose named attribute has a value satisfying some simple patterns.

- Example 17: JQuery, to be used with `examples/example.17.html`. (Examples 15 and 16 pertain to XML, which we are not covering.)
- Class 14: 2/24/2022
- Further examples from Example 17.
- One can query and modify any property (attributes, CSS, content) of selected elements.
- Exercise (in class): Write a single JavaScript statement you can enter (via the developer tools console) into Example 7 that uses a jQuery expression to identify all the `<li>` tags whose content contains an "`s`" and are odd children of their parent and then changes the font

color of their content to green and their font to Arial. This statement should not include any anonymous functions.

- See this solution.
- Exercise (in class): Write a web page using jQuery that shows a picture of an orange of width 20px (see Example 6) that when you click changes its size over 5 seconds to 200px. After that, one more click should change its size over 1 second to 300px. After that, click should have no effect. Use an `<img>` tag with `src` and `width` attributes.
- Class 15: 3/01/2022
- See this solution.

# 24 Review for midterm

# 25 Midterm exam

- Class 16: 3/03/2022 Exam
- Class 17: 3/08/2022 After-exam review

# 26 Fonts

- The fonts available to the browser depend on fonts installed on the user's computer. Sometimes the web-page designer wants to use a different font.
- Class 18: 3/10/2022
- CSS can include a font and refer to it via a URL:

```
@font-face {
  font-family: myName;
  src: url(location.woff);
}
```

- Browsers generally accept fonts in several formats: *woff*, *woff2*, *ttf*, *otf*.
- The `@font-face` style may also list `font-style`, `font-weight`, `unicode-range`.

- Google and other sites provide many fonts; see `http://fonts.google.com`.
- See Example 18.
- You can build a small font with special characters that you need; see `https://icomoon.io/app/`.
- Warning: Don't over-use strange fonts.
- Exercise (in class): Write a web page with some text that loads 5 fonts, and has some text in a `<span>` region. The text should change to a randomly-selected different font every time the cursor enters the text. You might want to use the JavaScript functions `Math.floor()` and `Math.random()`.
- Class 19: 3/22/2022
- See this solution.

# 27 More HTML and CSS

- See Example 19.
- More CSS: shadow effects, tooltips
- CSS selectors: hover, link/active/visited, focus
- Responsive web design: Viewport
  - By default, the browser shrinks the entire page to fit the width of the browser, often making the content too small on a smartphone.
  - a `<meta name='viewport'>` tag in the `<head>` can specify more useful values for the viewport.
  - `width=device-width` says that the page width should be the device width.
  - `initial-scale=2.0` says that the browser should set the initial zoom to that value.
- Responsive web design: @media
  - surrounds some CSS in the `<style>` section and enables it only if the browser has some property: media type (print, screen, speech, all) and feature (such as dimensions, resolution).

  - syntax:

```
@media not|only mediatype and (mediafeature and|or|not mediafeature) {
CSS-Code;
}
```

- CSS display: none (removes element) |inline (like span) |block (like p) |...
- CSS visibility: visible |hidden (still takes up space) |...
- Accessing media: `<video>` (only MP4, WebM, and Ogg) `<audio>` (only MP3, Ogg, and WAV) Safari does not play Ogg.
- Spellcheck and autocomplete; not available in all browsers.
- Exercise (in class): Build a web page with a table whose elements all have border shading (gray, 5px horizontal, 5px vertical, blurring in 5px), with a 4×4 grid of values, but the inner 4 grid cells contain a single entry: the letter X. The letter should be centered vertically and horizontally in each cell. Pad each cell with 10px on all sides. Extra: the big middle cell should have a tooltip saying "Y".
- Class 20: 3/24/2022
- See this solution.

# 28 Bootstrap: CSS

- Bootstrap is both a CSS library and a JavaScript library. Some of its facilities require other libraries. It used to require jQuery, but it no longer does.
- The current version (Spring 2022) is Bootstrap 5.1.3; as with all libraries and languages, versions change and new features appear.
- Standard reference.
- Main features
  - Provides **design templates** to make elements like buttons and tables prettier.
  - Supports **responsive web design**: pages that automatically adjust to look good on devices of various widths, ranging from phones to wide-panel displays.

- In the `<head>` section, include a `viewport` directive to begin the responsive web design and the Bootstrap CSS:

```
<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
<link rel="stylesheet"
  href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
  integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3"
  crossorigin="anonymous">
```

- At the end of the page, before the closing `</body>`, include JavaScript libraries for Bootstrap and Popper (a tooltip-positioning package); jQuery used be be required, but both Bootstrap and Popper have stopped using it.

```
<script
   src="https://code.jquery.com/jquery-3.5.1.slim.min.js"
   integrity="sha384-DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkfj"
   crossorigin="anonymous"></script>
<script
   src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.10.2/dist/umd/popper.min.js"
   integrity="sha384-7+zCNj/IqJ95wo16oMtfsKbZ9ccEh31eOz1HGyDuCQ6wgnyJNSYdrPa03rtR1zdB"
   crossorigin="anonymous"></script>
<script
   src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.min.js"
   integrity="sha384-QJHtvGhmr9XOIpI6YVutG+2QOK9T+ZnN4kzFN1RtK3zEFEIsxhlmWl5/YESvpZ13"
   crossorigin="anonymous"></script>
```

- Bootstrap provides its own CSS for all standard elements to enforce its own look and feel.
- See Example 20.
- It also provides classes that further enhance the CSS. Example 21.
- Class 21: 3/29/2022
- Responsive design has **breakpoints**:

| **name** | **abbreviation** | **dimension** |
|---|---|---|
| extra small | none | $<$ 576px |
| small | sm | $\geq$ 576px |
| medium | md | $\geq$ 768px |
| large | lg | $\geq$ 992px |
| extra large | xl | $\geq$ 1200px |
| extra extra large | xxl | $\geq$ 1400px |

  reference to any breakpoint refers to screens of that size unless there is an explicit override for a bigger size.

- The grid system: 12 grids across, but items can be configured to use a different number of grids on differently sized devices
- Forms: no special classes needed.
- Buttons: Various options.
- Exercise (in class): Build a web page using Bootstrap that has a table spanning columns 7 to 12 in the grid scheme, with the first and third rows each containing X X X (in three table columns) and the second row containing a single form. The form should have an input text region and a submit button. Feel free to make it fancy.
- See this solution.
- Modals (requires JavaScript): floating interactive region that hides the rest of the page.
- Class 22: 3/31/2022
- Navbars.
    - Used for navigation; the elements are typically links.
    - Can collapse to a smaller size for narrow displays.
    - Usually occupy the entire width.
    - Can be fixed to the top or bottom of the screen.
- Tooltips
    - The simplest HTML-based tooltip is to place on the surrounding element (typically a `button`) the attribute `title="`*`content`*`"`
    - Bootstrap tooltips require the `Popper.js` library.
    - Opt-in: the web page must have JavaScript that initializes them. They have a `data-bs-toggle="tooltip"` attribute, so you can use jQuery to initialize them all:

```
document.querySelectorAll('[data-bs-toggle="tooltip"]').
   forEach(function (elt)
{
   new bootstrap.Tooltip(elt);
});
```

    - A few gotchas; read the documentation concerning hidden or disabled elements and multiple-line hyperlinks.

    - The tooltip can be placed on any side, but Popper overrides the placement if the result would be invisible.
    - The tooltip can itself have style, but then the element needs `data-bs-html="true"`.
    - For keyboard and users of assistive technology (like screen readers): Only put tooltips on interactive, focusable elements like buttons.
- Pagination: typically a decoration on list items.
- Progress bars
- Exercise (in class): Build a web page using Bootstrap that has a table of 3×3 X characters and a progress bar, starting at 0. Each of the 4 side Xs (that is, not on corners) has a tooltip saying Y. The tooltips face inward (for instance, the one on the right side has a tooltip facing left). Every time you click on an X, the progress bar advances by 10%.
- Class 23: 4/5/2022
- See this solution.

# 29 DataTables

- DataTables is jQuery plugin (that means it depends on jQuery) to provide interactive tables.
- The manual is here.
- The tables can be formatted with Bootstrap.
- There are many options, some of which are shown in the example web page.
- This example also demonstrates favicon and scrollbar customization and a shaking image.
- See how Kratylos uses DataTables: `www.kratylos.org/~kratylos/project.cgi`
- Class 24: 4/7/2022
- Exercise (in class): Build a web page using DataTables that shows the contents of the Unix file `/etc/services`. You also need to build a back-end CGI script that formats this file in JSON. (You may

use `https://www.cs.uky.edu/~raphael/courses/CS316/examples/services.cgi`)

- See this solution.

# 30 Typescript

- Reference
- TypeScript is an extension of JavaScript that is careful about types.
  - It uses *static* typing as opposed to JavaScript's *dynamic* typing.
  - Static type checking helps prevent programming errors.
  - Static type checking is important for large programming projects.
- TypeScript can infer the type of an expression, so when you declare a variable and give it a value, TypeScript knows the type of that variable.
- TypeScript knows the types of the elements of the DOM.
- You can also define a type (it's called an **interface**) and explicitly type a variable declaration and function parameters and return type.
- Types can be unions:

```
function getLength(obj: string | string[]) {
  return obj.length;
}
```

- TypeScript supports generic types:

```
interface Backpack<Type> {
  add: (obj: Type) => void;
  get: () => Type;
}
declare const backpack: Backpack<string>;
const object = backpack.get();
backpack.add(23); // erroneous
```

- Unlike JavaScript, TypeScript is not built into the browser. If you have a TypeScript file, the *tsc* compiler can convert it to JavaScript as it checks the types.

# 31 IndexedDB

- It is possible to avoid DB manipulation on the server by putting the DB in the client. A good example is IndexedDB, "a low-level API for client-side storage of significant amounts of structured data".
- IndexedDB is not relational; it is **indexed**, which means that entries are key-value pairs; the key is considered the index, and the value can be any object.
- The programmer must build a database schema. Then a JavaScript program can open an internal connection to the database and then retrieve and update data via transactions.
- Most browsers other than Internet Explorer support IndexedDB.
- IndexedDB follows a **same-origin** policy: only the site that installs data can access or modify it.
- All operations are bundled within a **transaction**, which has a well-defined life span, and which is failure- and synchronization-atomic.
- Operations are asynchronous; their completion signals an event that the program can bind to a callback function, typically `onsuccess` and `onerror`.
- Class 25: 4/12/2022
- Queries on an index generally produce a **cursor** that can iterate through the resulting entries.
- IndexedDB Reference.
- IndexedDB example.

# 32 HTML canvas

- Class 26: 4/14/2022
- A `<canvas>` element lets you draw text and graphics.
- It requires JavaScript to actually perform the drawing.
- The JavaScript can build arcs, circles, polygons, rectangles, text.
- The *jCanvas* jQuery plugin makes programming easier.
- The items can be bound to events like `click` and `mouseenter`.
- Canvas example.

- Exercise (in class): Build a 300×300 canvas with a blue circle that moves away from the cursor whenever the cursor tries to enter it but always remains somewhere inside the canvas.
- Class 27: 4/19/2022
- See this solution.

# 33 HTML SVG

- SVG (Scalable Vector Graphics) is a more recent component of HTML that has the same abilities as Canvas.
- Graphic elements are placed in `<svg>` regions, and the code is part of those regions, part of the DOM, and not dependent on JavaScript.
- The graphic items are scalable and not dependent on screen resolution.
- Documentation: `https://www.w3schools.com/graphics/svg_intro.asp`.
- SVG is far more complex than the HTML canvas; you typically want to use an SVG editor (such as `https://svgeditoronline.com/editor/`) instead of building your own.
- Inside of `<svg>` you can place drawing commands such as `<circle>`, `<line>`, and `<rect>`, but most drawing is done by grouping elements with `<g>` and using the `<path>` tag.
- SVG examples.
- Continue examples
- In-class exercise: Use SVG to make a red square with a blue circle inside.
- See this solution.
- SVG `viewbox` attribute to scale contents.
- Continue examples

# 34 REST

- Web programming is an example of **REST**: REpresentational State Transfer.

- This term was introduced in 2000 by Roy Fielding.
- Basic characteristics of a RESTful interface
  - Client-server architecture. The server implementation is independent of the client implementation.
  - Statelessness. Each request from client to server contains all the information needed to handle the request, including credentials, which must be stored on the client.
  - Cacheability: Cacheable responses can be stored by the client to avoid repeated requests. The server should indicate whether a response is cacheable.

# 35 Security worries

- CGI vulnerabilities
  - The CGI program must sanitize inputs.
    - Server-side validation and sanitation is much safer than client-side, because a user can use the Developer Console to run any JavaScript and to modify any HTML, working around any client-side validation and sanitizing.
    - CGI programs that allow GET can be invoked by URL, bypassing the browser altogether, and the URL can provide any values for parameters.
    - A user can invoke the CGI program via POST and even provide cookies with programs like *curl*.
  - **Code-injection attacks**: Parameters can include program fragments, which the CGI program might unwittingly send to the command interpreter or to an SQL processor.
    - Example of the shell-code injection attack: To compute a file's digest using *sha256sum*, the CGI program might pass the file name to the command interpreter, and the file name might be `foo; rm *`. To sanitize: remove shell metacharacters: {& ; ' " >< !} (or only accept good characters).
    - A field to an SQL query might inject SQL code. Example: See `https://xkcd.com/327/`. To sanitize: remove SQL metacharacters: {; ' " ‘ }, or always use a DB connector method separating the SQL from the parameters.

- **Cross-site scripting** (XSS) attacks
  - **Stored XSS**: Data stored and retrieved includes HTML `<script>` regions, causing the retrieving page (run by an unsuspecting user) to leak sensitive information by invoking arbitrary JavaScript. That code could, for instance, make an HTTP request with parameters to a malicious site.
  - **Reflected XSS**: Malicious `<script>` code can be embedded in the parameters of a `GET` URL and can be obfuscated with alternatives such as `%3C` or `&#60;` (both represent <).
  - **DOM-based XSS**: JavaScript takes a value from an input field and invokes `eval()` or `innerHTML` with it, but a malicious user has placed a `<script>` in that input field.

  To sanitize: remove or feather HTML metacharacters: {<> `&`}; use JavaScript `.text()` instead of `.html()`.

- Social engineering, typically via email
  - **Phishing**: Convincing a user (typically via email) to reveal private information such as passwords by directing the user to what appears to be a legitimate site.
  - **Spear phishing**: similar, but targeted to specific individuals, typically with email that appears to come from a trusted person.
  - **Whale phishing**: similar, but targeting a specific high-profile target person.
  - **Scam**: an attempt to get banking information and pre-payments to release large (but non-existent) winnings/gifts/inheritances.
  - **Spam**: an unwanted advertisement, possibly added by a malicious agent on a website that has a public "comment" section.
- Remote attacks
  - **Buffer overflow**: sending invalid data to a server daemon that it is not programmed to handle, leading to server failure or escalation of privilege.
  - **Brute-force attacks**: repeatedly trying to connect to restricted services (like ssh or email) by trying different user names and passwords.

- **Denial of service (DOS)**: overloading the server by repeated expensive requires.
- **Distributed denial of service (DDOS)**: like DOS, but with a coordinated attack from many sources.

# 36 Defense methods

- Class 28: 4/21/2022
- Use `https` instead of `http` as the protocol. This connection provides **authentication** (you can be sure of the identity of the server), **privacy** (intruders cannot decrypt the conversation) and **integrity** (intruders cannot modify the conversation without destroying it).
  - The browser sends the server `ClientHello`, containing a list of cryptographic protocols (such as `TLS 1.3`) and cipher suites (such as `TLS_AES_128_GCM_SHA256`) it supports.
  - The web server returns `ServerHello`, containing a **certificate**, which provides cryptographic proof of the identity of the web server, and which protocol and suite it wants to use.
  - A certificate is part of the PKI (Public Key Infrastructure), a distributed web of trust.
  - The user can ask the browser to display the certificate.
  - If the browser does not trust the certificate, it prevents browsing, although (1) the user can usually choose to continue anyway, (2) the connection will be secure (read-write protected), although without authentication, and (3) subject therefore to man-in-the-middle (MIM) attack.
  - See `https://ssllabs.com` for results testing web servers and browsers.
  - Browsers display a lock symbol near the URL when they are connected via `https`; never provide personal information to sites that do not have that symbol.
  - The CGI program can check the environment variable `SERVER_PORT` to verify that it is 443. If it isn't, the CGI can respond with a redirection instead of a normal response: `<html><head><meta http-equiv="Refresh" content="0;url=https://THIS_FILE>"`

    - The web site can include an `.htaccess` file that forces `https`, and the web server can be configured to only allow `https` connections.
    - The web server must be configured to present the certificate and to list the security protocols it accepts.

- A website can use `<meta>` or the HTTP header to specify a **Content Security Policy (CSP)** such as disallowing all JavaScript, disallowing JavaScript from a different site, or restricting the source for particular kinds of content. Example:
  `<meta http-equiv="Content-Security-Policy" content="default-src 'self'; img-src https://*;">`
  This setting disallows JavaScript from outside sources and requires the `https` protocol for all images.
- Use `ssh` instead of older techniques such as `telnet` and `rlogin`. See `https://sshaudit.com`. Test `glo.cs.uky.edu` and `www.engr.uky.edu`.
- Passwords

    - Log brute-force attempts and employ firewall rules to temporarily or permanently disallow connections from bad actors. See `~raphael/src/f2b`.
    - Store passwords as cryptographic digests.
    - There are laws regarding proper storage and backups of personal information.
    - For critical applications, consider 2-factor authentication (**what you know**: the password, **what you have**: a phone, a fingerprint, a retinal scan)
    - Users should use a password vault that can generate random passwords, fresh for every application, and can save them all under the security of a single master password.
    - Master password: first letters of some phrase, with extra numbers and symbols, or at least 3 words joined together, such as 'phlegmaticcircusbicycle". See `https://xkcd.com/936/`.

- Sandboxes: If you can build virtual machines (VMs) using VirtualBox or something like it, construct several such machines: (1) general surfing (2) purchases (3) banking.

- Firewalls: allow/disallow internet connections based on source/destination IP, port, protocol. Hard to set up, but very effective.
- Scanning: use a web application vulnerability scanner, such as one listed at `https://owasp.org/www-community/Vulnerability_Scanning_Tools`, to verify that web sites are free of obvious security problems.
- Further information and practice in finding vulnerabilities: `www.hackthissite.org/`.

# 37 React

- There are many high-level frameworks available, such as **Angular** and **Vue.js**. The most popular at the moment is **React**, a JavaScript library developed by Jordan Walke at Facebook and first released in 2013. The current version is 16 (April 2022).
- React uses **JSX** (JavaScript XML), a JavaScript syntax extension that looks like HTML. All JSX expressions must contain a single element, which may contain sub-elements.
- The **babel** plugin translates JSX code to JavaScript as needed.
- React relies on some advanced features of JavaScript, enumerated at `https://www.w3schools.com/react/react_es6.asp`.
- React **renders** pieces of HTML; those pieces can be strings, functions, or classes.
- The easiest method is to build a JavaScript function that returns HTML. Its name must start with a capital letter. You can then cause the HTML returned by the function to be **rendered** as a new tag inside a given element. You can render the same function more than once. For very simple HTML, you can place it as the value of a string and render the string.
- See `examples/example.26.html`