Web Basics: Markup Languages, and the Request Response Cycle

Google

418. I'm a teapot.

The requested entity body is short and stout.
Tip me over and pour me out.

Brief Review

- We can build **services** on the Internet
  - Informational websites
  - Video streaming
  - Social media
  - High-performance computing
- A **Client** communicates with a **Server** that *hosts* the service
  - Client-Server based on some *protocol* that structures information
- We will start with **websites**
  - Static: Same content every time
  - Dynamic: Server *generates* “unique” content on every request

Brief Review: Python

- Python is an *interpreted*, *dynamically-typed* language common in Web Services
  - Server software often interacts with Python on the backend

- Python has a lot of supporting **packages**
  - "pip" is a *package manager* for Python: `pip install _____`
  - Project 1: `pip install jinja2`, then `import jinja2`
    - Why? Reuse code from others to accomplish a larger goal
    - e.g., `import json, json.loads()`
  - Conceptually equivalent to downloading a C library, then `#include`'ing a header
- A **virtual environment** is a way of isolating Python packages

One-Slide Summary: Markup, Protocols

- **Markup Languages** allow representing structured information
  - The markup language speaks to the *structure* and *appearance* of the document
  - The *hypertext markup language (HTML)* is used for building websites
    - HTML focuses on structure: headings, lists, links, images, etc.
  - *Cascading Style Sheets (CSS)* works with HTML to enhance the visual appearance of a document
    - CSS focuses on appearance: how big is the font? What color? Do links change color when you click?
- The **request response cycle** is our informal description of the interaction between the **client** and **server** while providing the service
  - **Uniform Resource Locators (URLs)** are like links. They provide a named way of accessing a resource
    - URLs can *encode* a lot of information (think: inputs from client can be encoded!)
  - The **Hypertext Transfer Protocol (HTTP)** is the standard for communicating websites from a **webserver** to a client **browser**

HyperText Markup Language: HTML

- HyperText Markup Language
  - “Hyper” meant advanced… in the olden days, folks would just use “plain” text
- Invented by Tim Berners-Lee in 1990
- Set of **tags** for rendering page

- HTML is a recursive language
  - Consider for this lecture:
    This allows creating a *tree* data structure that contains all tags in a document!

Markup language

- How do we improve the presentation of the plain text transmitted over HTTP ...

  `Academics+Admissions Research People Industry News+Awards Events`

- ... so that it looks like this?

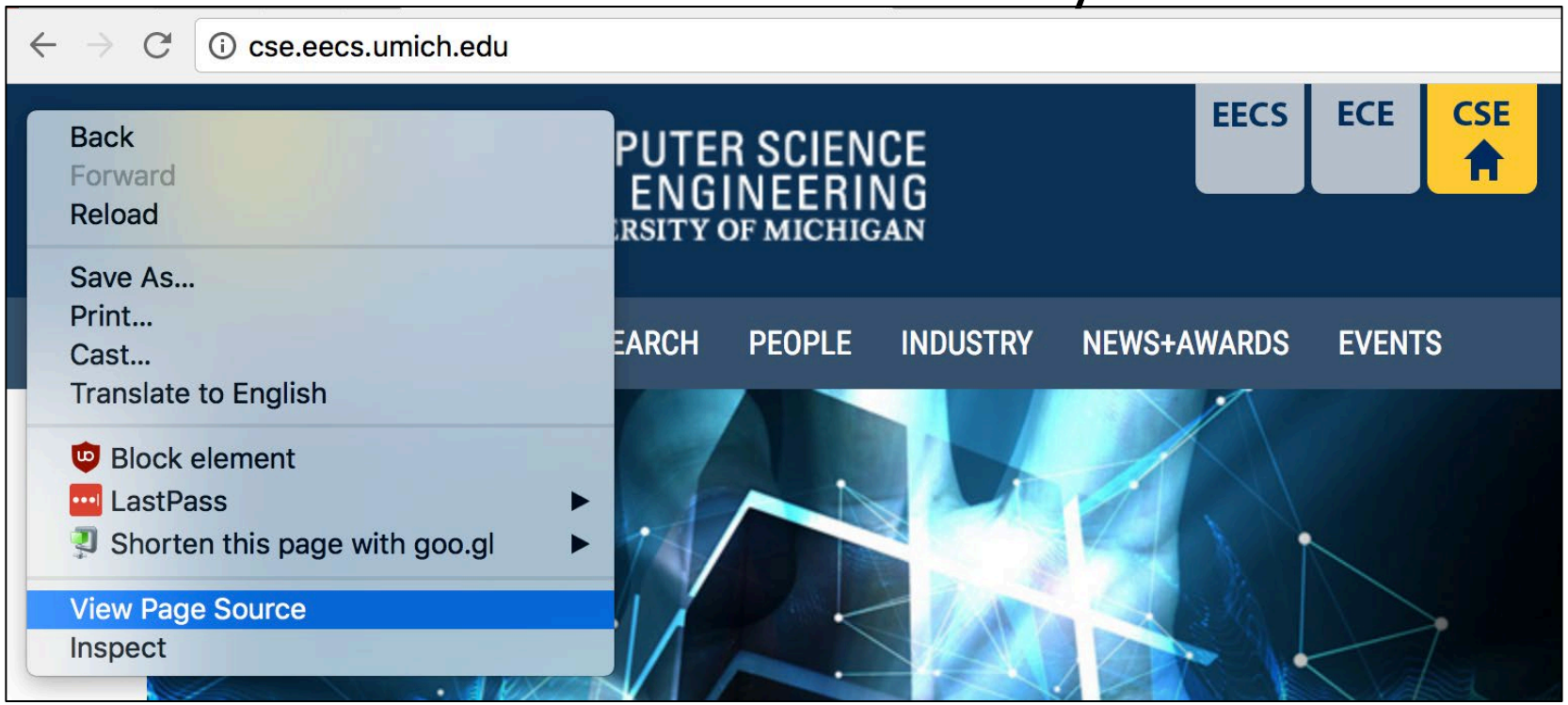
HTML

- Add **tags** as "mark up" to text
- End user sees structured document rather than a plain string!

```
<html>
  <head></head>
  <body>
    <nav>
    <ul>
      <li><a href="">Academics+Admissions</a></li>
      <li><a href="">Research</a></li>
      <li><a href="">People</a></li>
      <li><a href="">Industry</a></li>
      <li><a href="">News+Awards</a></li>
      <li><a href="/eecs/etc/events/cseevents.html">Events</a></li>
    </ul>
    </nav>
  </body>
</html>
```

Seeing HTML

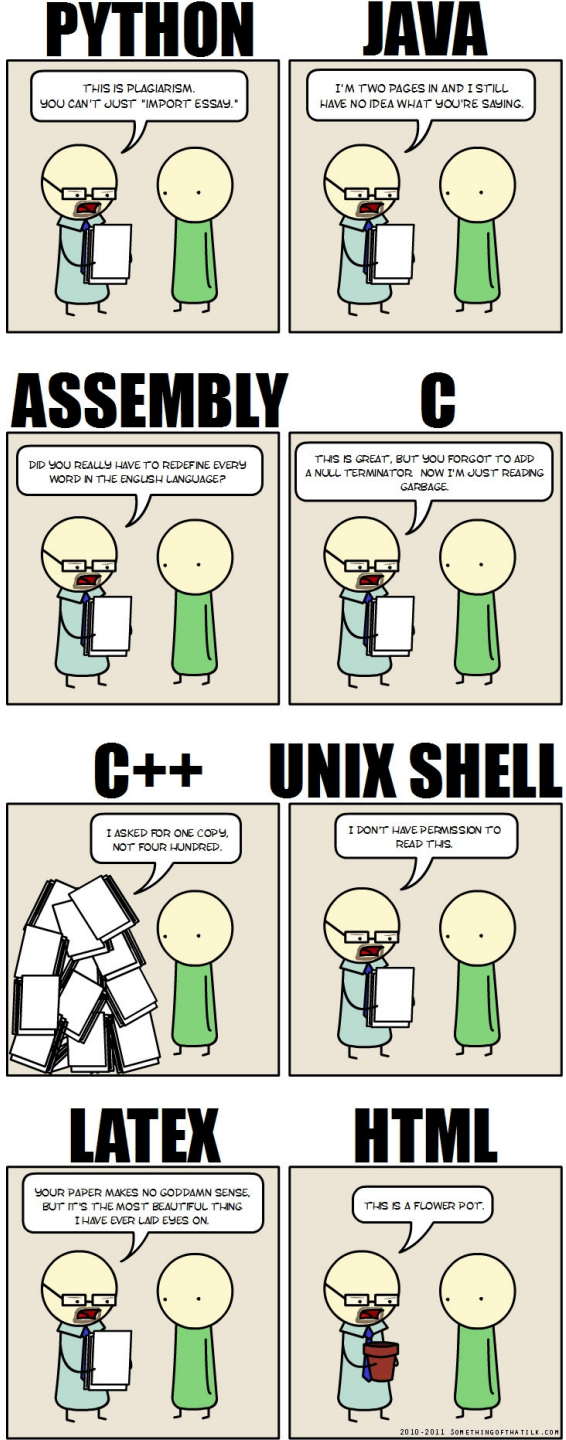
- See the HTML source with “View Source” in your web browser



Markup vs. programming language

- Markup language: data *presentation*
- Programming language: data *transformation*

- Markup language examples:
  - HTML, used for web pages
  - XML, used to communicate data between applications.
    - Less popular today (btw, Web Services involve fads!)
  - Markdown, used by GitHub
  - troff and nroff, used for man pages
  - TEX, used for publishing
- Programming language examples:
  - C/C++, Python, JavaScript, Bash (shell scripting)



Escape strings in HTML

- Some characters have special meaning in an application context
  - Example: “<“ in HTML
- What if you want to communicate such a character as an ordinary character?
  - Example: “I <3 chickens”
- Need an escape string
  - Example: “I **<**3 chickens”

XML: eXtensible Markup Language

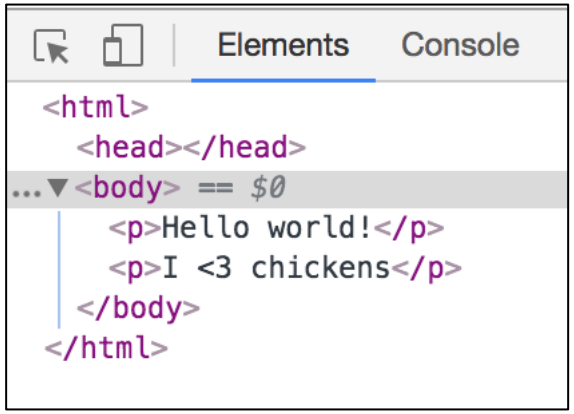
- Superset of HTML (generalized from HTML)
- No standard set of tags!!
- Define tags and use them
- Tags form a hierarchy of objects
  - Each open tag has a matching close
  - Must balance, like parentheses
  - Can build a tree of document objects
    - Document Object Model (DOM)
- JSON is the modern equivalent

Document Object Model (DOM)

- HTML tags form a tree

```
<html>
  <head></head>
  <body>
    <p>Hello world!</p>
    <p>I &lt;3 chickens</p>
  </body>
</html>
```

- This tree is called the Document Object Model (DOM)
- Inspect the DOM with
  - Chrome developer tools
  - Firefox developer tools



DOM Tree Nodes

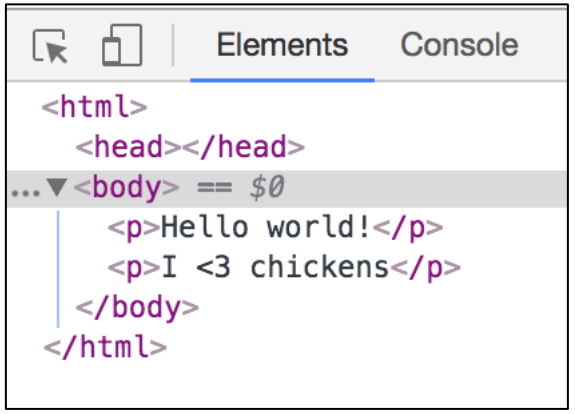
- Type
- Attributes
- Contents – text, other nodes

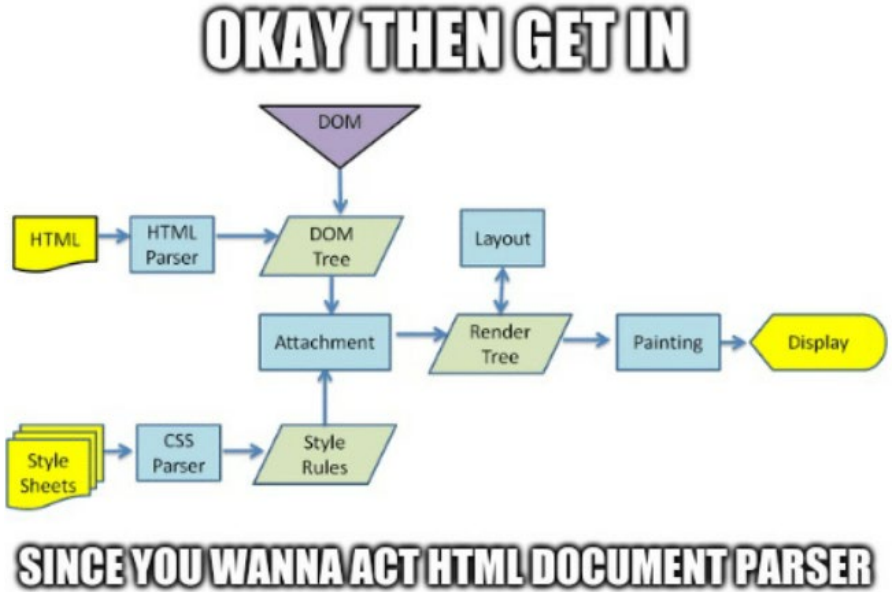
```
<a href="umich.edu"><h1>Hail</h1> to the victors</a>
```

https://software.hixie.ch/utilities/js/live-dom-viewer/

Document Object Model (DOM)

- The DOM is a data structure built from the HTML
- In the DOM, everything is a **node**
  - All HTML elements are element nodes
  - Text inside HTML elements are text nodes





HTML5

- HTML5 merges all that has happened over years related to HTML:
  - XHTML
  - Browser-specific extensions of HTML
- Finalized and published by W3C in 2014
- Check your HTML using a website
  - `https://validator.w3.org/nu/?doc=URL_GOES_HERE`
- Check your HTML at the CLI

```
$ pip install html5validator
$ html5validator --root YOUR_HTML_DIR/
```

CSS: Cascading Style Sheets

- **separate out** the *structure* of a document from its *appearance*
- This is sort of an “addon” to HTML
- HTML was around first, so CSS specifies appearances of HTML tags

```
body {
    background-color: red;
}
p a {
    color: blue;
}
```

```
(old HTML equivalent; don’t do this)
<body bgcolor=“red”>

<p><font color=“blue”>…</font></p>
```

CSS: Cascading Style Sheets

- **separate out** the *structure* of a document from its *appearance*
- This is sort of an “addon” to HTML
- HTML was around first, so CSS specifies appearances of HTML tags

```
body {
    background-color: red;
}
p a {
    color: blue;
}
```

```
(old HTML equivalent; don’t do this)
<body bgcolor=“red”>

<p><font color=“blue”>…</font></p>
```

Content, presentation, layout

- HTML has tags for all.
  - `<h1>` is content
  - `<b>` is presentation
- Good to separate these.
- Content is data; presentation is words, tables, organization; layout is visual.
- Use HTML for content and presentation, add CSS for layout.

Making requests: URLs

- URL -> Uniform Resource Locator

- What you type into the address bar of your web browser
  - https://en.wikipedia.org/wiki/URL

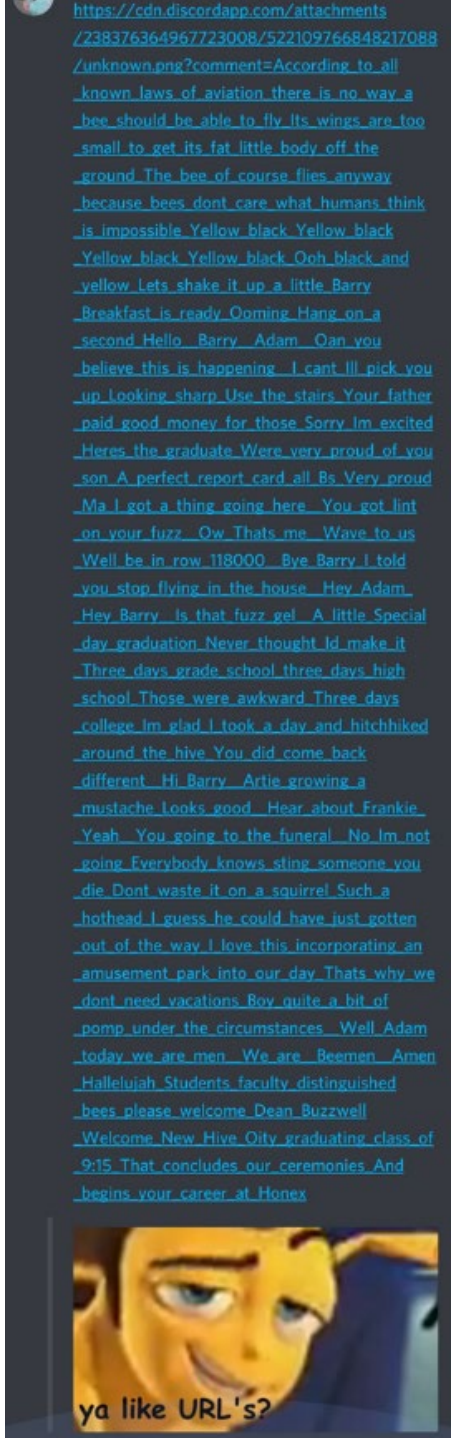
- Tells a web browser to make a request

URL encoding

```
$ curl --verbose http://cse.eecs.umich.edu/
```

```
protocol://server:port/path?query#fragment
```

- URLs have several parts
  - Protocol
  - Server
  - Port
  - Path
  - Query
  - Fragment



URL encoding: protocol

protocol://server:port/path?query#fragment

- `protocol` tells the server what protocol to use. In other words, what "language" to speak
- Example: unencrypted `http`

```
$ curl --verbose http://cse.eecs.umich.edu/
* Connected to cse.eecs.umich.edu
(141.212.113.143) port 80 (#0)
```

- Example: encrypted `https`

```
$ curl --verbose https://cse.eecs.umich.edu/
* Connected to cse.eecs.umich.edu
(141.212.113.143) port 443 (#0)
* TLS 1.2 connection using
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
* Server certificate: www.cse.umich.edu
```

URL encoding: server

`protocol://`**`server`**`:port/path?query#fragment`

- `server` helps locate the machine we want to talk to
- Example

```
$ curl --verbose http://cse.eecs.umich.edu/
* Connected to cse.eecs.umich.edu (141.212.113.143)
port 80 (#0)
```

- DNS lookup translates server name into an IP address

```
$ host cse.eecs.umich.edu
cse.eecs.umich.edu has address
141.212.113.143
```

URL encoding: port

`protocol://server:`**`port`**`/path?query#fragment`

- `port` is used to identify a specific service
- One host machine, several servers
  - Example: 80 is typically HTTP, 443 for HTTPS
- Check if a port is open

```
$ nc -v -z cse.eecs.umich.edu 80
cse.eecs.umich.edu [141.212.113.143] 80 (http)
open
$ nc -v -z cse.eecs.umich.edu 443
cse.eecs.umich.edu [141.212.113.143] 443 (https)
open
```

- See what ports are open

URL encoding: port

`protocol://server:`**`port`**`/path?query#fragment`

- See what ports are open

```
$ nmap cse.eecs.umich.edu
PORT      STATE    SERVICE
22/tcp    open     ssh
80/tcp    open     http
443/tcp   open     https
5666/tcp  open     nrpe
```

- WARNING: it's "not nice" to port scan!
  - Some websites might blacklist you for doing this

URL encoding: path

`protocol://server:port`**`/path`**`?query#fragment`

- `path` is a file name relative to the server root
- Default is `/index.html`
  `$ curl http://cse.eecs.umich.edu`
  is the same as
  `$ curl http://cse.eecs.umich.edu/index.html`
- A different path loads a different page
  `$ curl http://cse.eecs.umich.edu/eecs/faculty/csefaculty.html`

Absolute vs. relative paths

- Absolute path
  - Starts with a slash (/)
  - Examples:
    ```
    <img src="/uploads/9887e06812ef434d291e4936417d125cd594b38a.jpg">
    <a href="/u/awdeorio/">awdeorio</a>
    ```
- Relative path
  - Computed starting “present directory”. AKA, the path of the current page
  - Example:
    ```
    <img src="9887e06812ef434d291e4936417d125cd594b38a.jpg">
    <a href="awdeorio/">awdeorio</a>
    ```
- Prefer absolute path for automatically generated pages
  - Including P1
- Prefer to automatically generate links
  - Remember this on P2

URL encoding: query

`protocol://server:port/path`**`?query`**`#fragment`

- `query` string is a general-purpose set of parameters that the server (or specified resource on server) can use as it pleases
- You can encode lots of parameters! ?a=1&b=2&c=3

https://www.google.com/search?client=firefox-b-1-d&bih=531&biw=1056&hl=en&source=hp&ei=5xH8XpeWD8LOtAa5vojwBg&q=dank+memes&oq=dank+memes&gs_lcp=CgZwc3 180%

dank memes

- http://cse.eecs.umich.edu/eecs/etc/fac/CSEfaculty.html
- http://cse.eecs.umich.edu/eecs/etc/fac/CSEfaculty.html?match=All
- http://cse.eecs.umich.edu/eecs/etc/fac/CSEfaculty.html?match=Lecturer
- http://cse.eecs.umich.edu/eecs/etc/fac/CSEfaculty.html?match=Tenure

URL encoding: fragment

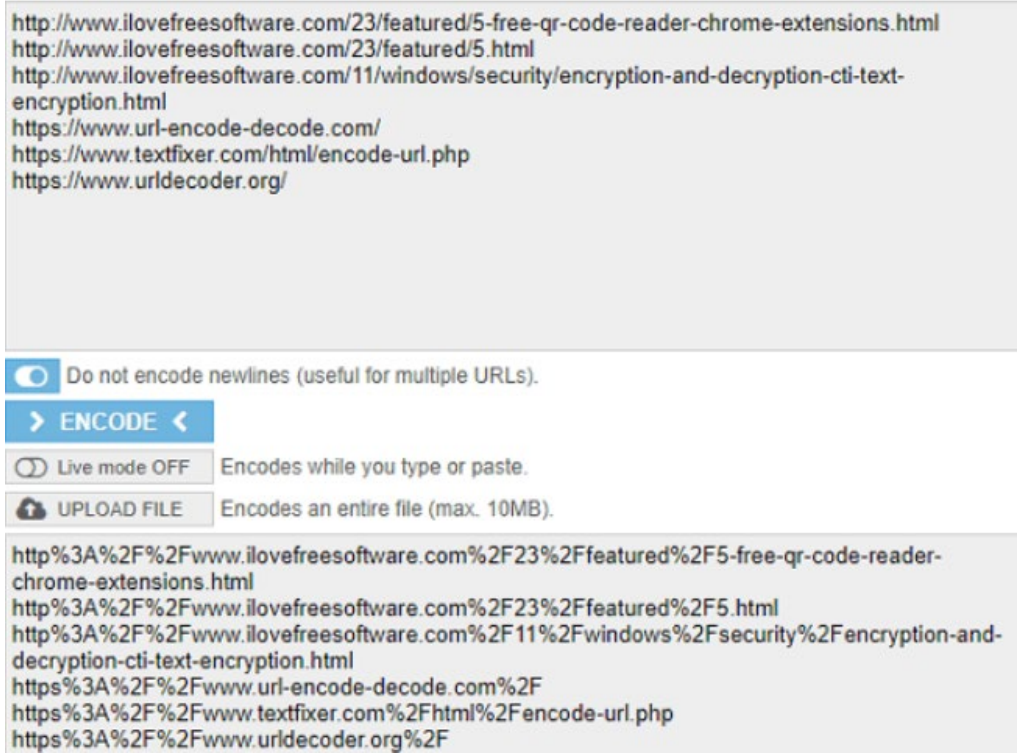
`protocol://server:port/path?query`**`#fragment`**

- `fragment` is identified at the client, ignored by server
- Example: navigate directly to the section labeled "Linking"
  http://en.wikipedia.org/wiki/World_Wide_Web**#Linking**

- Aside: The *anchor* tag <a> originally meant to specify places that fragments can go to!
- Create the anchor
  - `<a name="Linking">Some Text</a>`
- Elsewhere:
  - `<a href="#Linking">Jump to Some Text</a>`

Escape strings in URLS

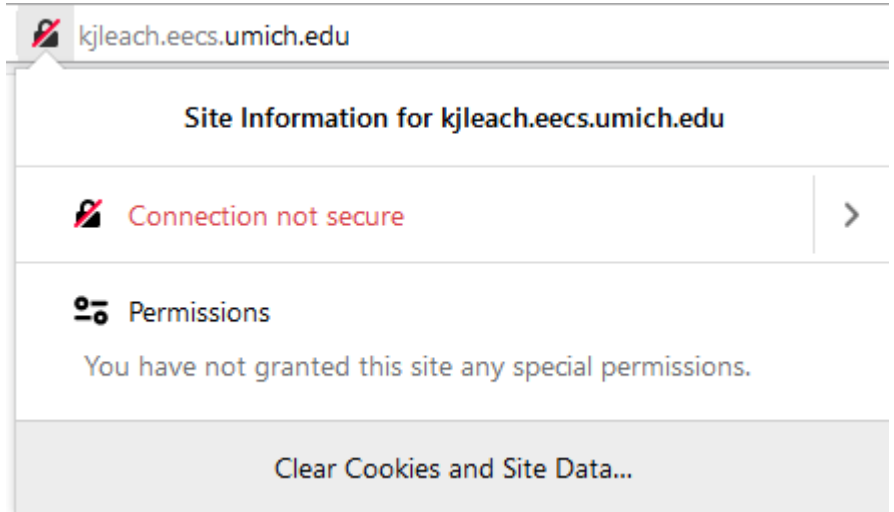
- Some characters have special meaning in an application context
  - Example: “ “ (a space) in a URL
    - Old servers would delimit inputs with spaces in HTTP requests!
- What if you want to communicate such a character as an ordinary character?
  - Example: `hello world.jpg`
- Need an escape string
  - Example: `hello%20world.jpg`

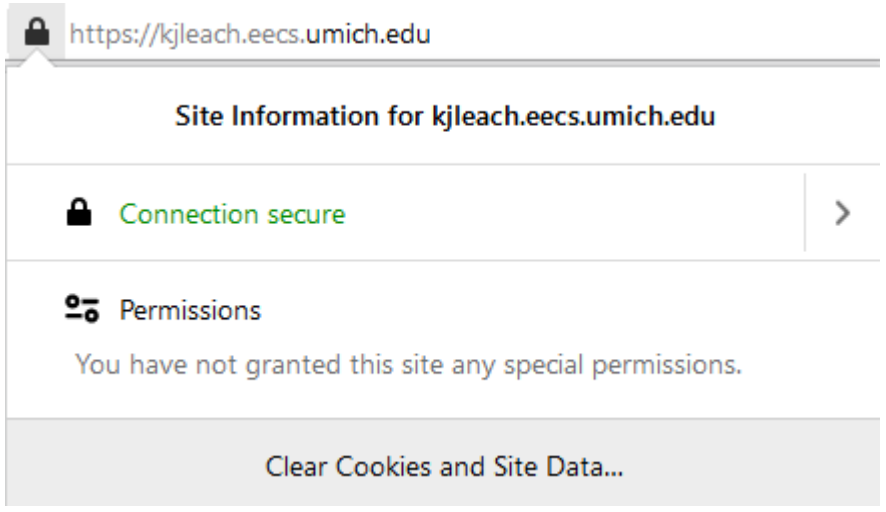


Request protocol: HTTP

- Protocol – language the client and server speak to each other

- HTTP: used to be used for most web pages
- HTTPS: encrypted version of HTTP, way more common today
  - Your browser will complain if you don't have https





Details of the request response cycle

```
$ curl --verbose http://cse.eecs.umich.edu/ > index.html
* Connected to cse.eecs.umich.edu (141.212.113.143) port 80 (#0)
> GET / HTTP/1.1
> Host: cse.eecs.umich.edu
> User-Agent: curl/7.54.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Tue, 12 Sep 2017 20:04:20 GMT
< Server: Apache/2.2.15 (Red Hat)
< Accept-Ranges: bytes
< Connection: close
< Transfer-Encoding: chunked
< Content-Type: text/html; charset=UTF-8
<
* Closing connection 0
```

HTTP

- Hypertext Transfer Protocol
- Request/response protocol
  - Client (your browser) opens connection to server and writes a request
  - Server responds appropriately
  - Connection is closed
  - That’s it

- Server can't open connection to client

- Completely stateless
  - Each request is treated as brand new
  - No state => no history

HTTP manual example - demo

```
$ telnet cse.eecs.umich.edu 80
Trying 141.212.113.143...
Connected to cse.eecs.umich.edu.
Escape character is '^]'.
GET /index.html HTTP/1.0
```

request

```
HTTP/1.1 200 OK
...
<!doctype html>
...
</html>
```

response

HTTP request methods

- The client's request contains what it wants the server to do
- GET: request a resource
  - Example: load a page
- HEAD: identical to GET, but without response body
  - Example: see if page has changed
- POST: send data to server
  - Example: web form

- Others we will cover later in the REST API lecture

HTTP request headers

```
$ curl --verbose http://cse.eecs.umich.edu/ > index.html
* Connected to cse.eecs.umich.edu (141.212.113.143) port 80 (#0)
> GET / HTTP/1.1
> Host: cse.eecs.umich.edu
> User-Agent: curl/7.54.0
> Accept: */*
```

- `Host` distinguishes between DNS names sharing a single IP address
  - Required as of HTTP/1.1
- `User-Agent`: which browser is making the request
- `Accept`: which content ("file") types the client will accept

User agent

- When a browser visits a page, it identifies itself with a `User-agent` string
  - For example, check yours out:
  - http://www.whatismyuseragent.net/

- Example from Google Chrome:
  - `Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/48.0.2564.103 Safari/537.36`
  - Previously used to indicate compatibility with the Mozilla rendering engine
  - During the "browser wars", some web sites would only send advanced features to some user agents

HTTP status code

- Response starts with a status code
  - 1XX: Informational
  - 2XX: Successful
  - 3XX: Redirection Error
  - 4XX: Server Error

Not Found

The requested URL /test was not found on this server.

Additionally, a 404 Not Found error was encountered while trying to use an ErrorDocument to handle the request.

- ```
  $ curl --verbose http://cse.eecs.umich.edu/
  > GET / HTTP/1.1
  < HTTP/1.1 200 OK
  ```
- ```
  $ curl --verbose http://cse.eecs.umich.edu/asdf
  > GET /asdf HTTP/1.1
  < HTTP/1.1 404 Not Found
  ```

HTTP response headers

- Headers accompany a response
- Most are optional

```
$ curl --verbose http://cse.eecs.umich.edu/
* Connected to cse.eecs.umich.edu
> GET / HTTP/1.1
...
< HTTP/1.1 200 OK
< Date: Tue, 12 Sep 2017 20:04:20 GMT
< Server: Apache/2.2.15 (Red Hat)
< Accept-Ranges: bytes
< Connection: close
< Transfer-Encoding: chunked
< Content-Type: text/html; charset=UTF-8
```

HTTP content type

- Content type describes the "file" type and encoding

```
$ curl --verbose http://cse.eecs.umich.edu/
* Connected to cse.eecs.umich.edu
> GET / HTTP/1.1
...
< HTTP/1.1 200 OK
...
< Content-Type: text/html; charset=UTF-8
```

HTTP content type

- Content type describes the "file" type and encoding

```
$ curl --verbose
http://cse.eecs.umich.edu/eecs/images/CSE-Logo-Mobile.png > CSE-
Logo-Mobile.png
* Connected to cse.eecs.umich.edu
> GET /eecs/images/CSE-Logo-Mobile.png HTTP/1.1
...
< HTTP/1.1 200 OK
< Content-Type: image/png
```

COMPUTER SCIENCE & ENGINEERING
UNIVERSITY OF MICHIGAN

MIME Types

`Content-Type: text/html; charset=UTF-8`

- MIME: Multipurpose Internet Mail Extensions
- Way to identify files
- Browser can open or display content correctly
- **`<type>/<subtype>`**

- https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types/Complete_list_of_MIME_types

Character encodings

`Content-Type: text/html;` **`charset=UTF-8`**

- Most people in the world use languages other than English
- ASCII -> char in C++

- 我太帅了 => ?
  - Encode with Unicode: \xE6\x88\x91\xE5\xA4\xAA\xE5\xB8\x85\xE4\xBA\x86
  - (think `char x = 0xE6;` in C; each of these is a byte in UTF-8)

- In olden times, you used to have to configure your browser ahead of time to know how to interpret a string – UTF-8 is a modern time saver!

Character Encodings

- Things look nasty if not specified!

Character encodings

- Check an HTML file for its reported encoding:

```
<!DOCTYPE html>
  <head>
    <meta charset="UTF-8">
    ...
```

- Check any file for its encoding on the file system

```
$ file index.html
index.html: HTML document text, ASCII text
$ file --mime index.html
index.html: text/html; charset=us-ascii
```

- UTF8 is a superset of ASCII
- UTF-8 is identical to ASCII for the values from 0 to 127

Character encodings and Python

- Strings in Python2 are ASCII encoded (default)
- Strings in Python3 are Unicode encoded (default)

- Much of the web is Unicode
  - Put another way: lots of the HTML out there is UTF-8 encoded
  - Why? Because the web is global!

- Unicode support was one reason for Python 3

HEAD request

- HEAD requests are useful for checking if a page has changed
- Good for caching stuff that doesn't change much and updating it
- Need a page that doesn't change much for this example

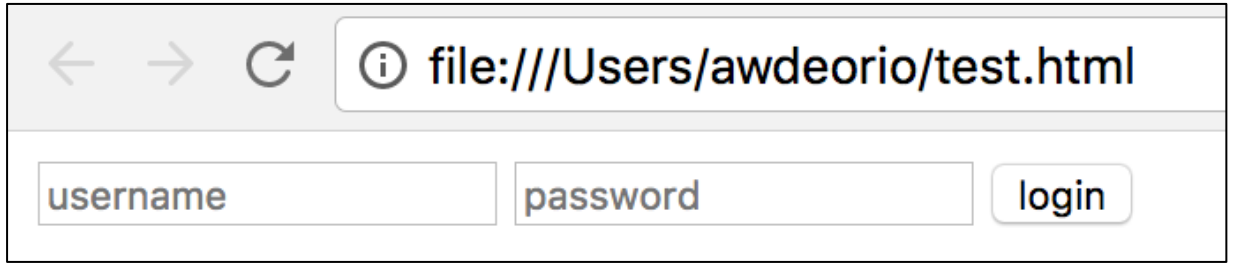
```
$ curl --head --verbose http://cse.eecs.umich.edu/eecs/images/CSE-Logo-Mobile.png > CSE-Logo-Mobile.png
* Connected to cse.eecs.umich.edu
> GET /eecs/images/CSE-Logo-Mobile.png HTTP/1.1
...
< HTTP/1.1 200 OK
< Content-Type: image/png
...
< Last-Modified: Thu, 28 May 2015 13:40:55 GMT
```

COMPUTER SCIENCE & ENGINEERING
UNIVERSITY OF MICHIGAN

POST request

- POST request sends data from the client to the server
- Commonly used with HTML forms

```
<html>
<body>
  <form action="" method="post" enctype="multipart/form-data">
    <input type="text" name="username" placeholder="username"/>
    <input type="password" name="password" placeholder="password"/>
    <input type="submit" value="login"/>
  </form>
</body>
</html>
```



HTTP versions

- See the HTTP version in the request and response

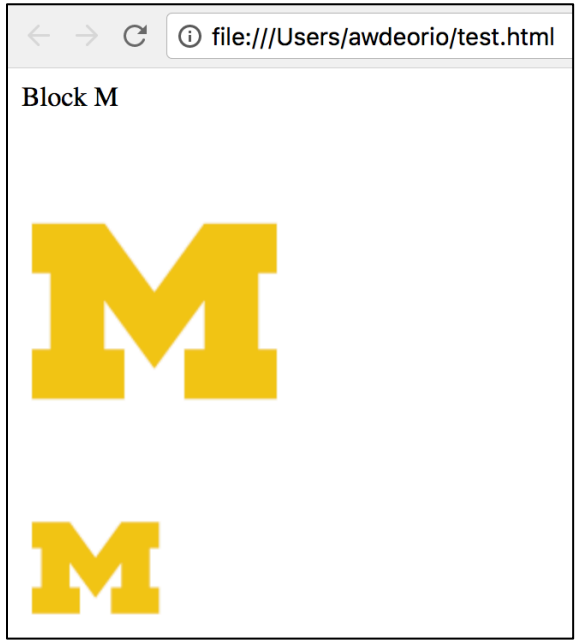
```
 $ curl --verbose http://cse.eecs.umich.edu/
* Connected to cse.eecs.umich.edu
> GET / HTTP/1.1
...
< HTTP/1.1 200 OK
```

- Three versions:
  - HTTP/1.0 (old)
  - **HTTP/1.1 (common)**
  - HTTP/2 (new)

HTTP/1.0 .vs HTTP/1.1

- How many TCP connections?

```
<html>
<body>
  <p>Block M</p>
  <img src="http://cse.eecs.umich.edu/eecs/images/CSE2017-hi-rev.png">
  <img src="http://cse.eecs.umich.edu/eecs/images/CSE2017-rev.png">
</body>
</html>
```

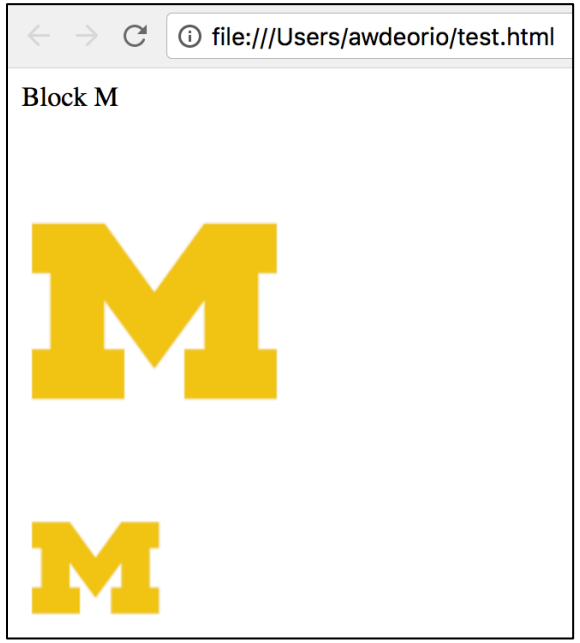


- HTTP/1.0: 3

HTTP/1.0 .vs HTTP/1.1

- How many TCP connections?

```
<html>
<body>
  <p>Block M</p>
  <img src="http://cse.eecs.umich.edu/eecs/images/CSE2017-hi-rev.png">
  <img src="http://cse.eecs.umich.edu/eecs/images/CSE2017-rev.png">
</body>
</html>
```



- HTTP/1.1: 1
  - Reuse one HTTP connection

HTTP/2

- Methods, status codes, etc. same as HTTP/1.1
- One new feature: server push
  - Server supplies data it knows a web browser will need to render a web page, without waiting for the browser to examine the first response.
  - Example: images from previous slide

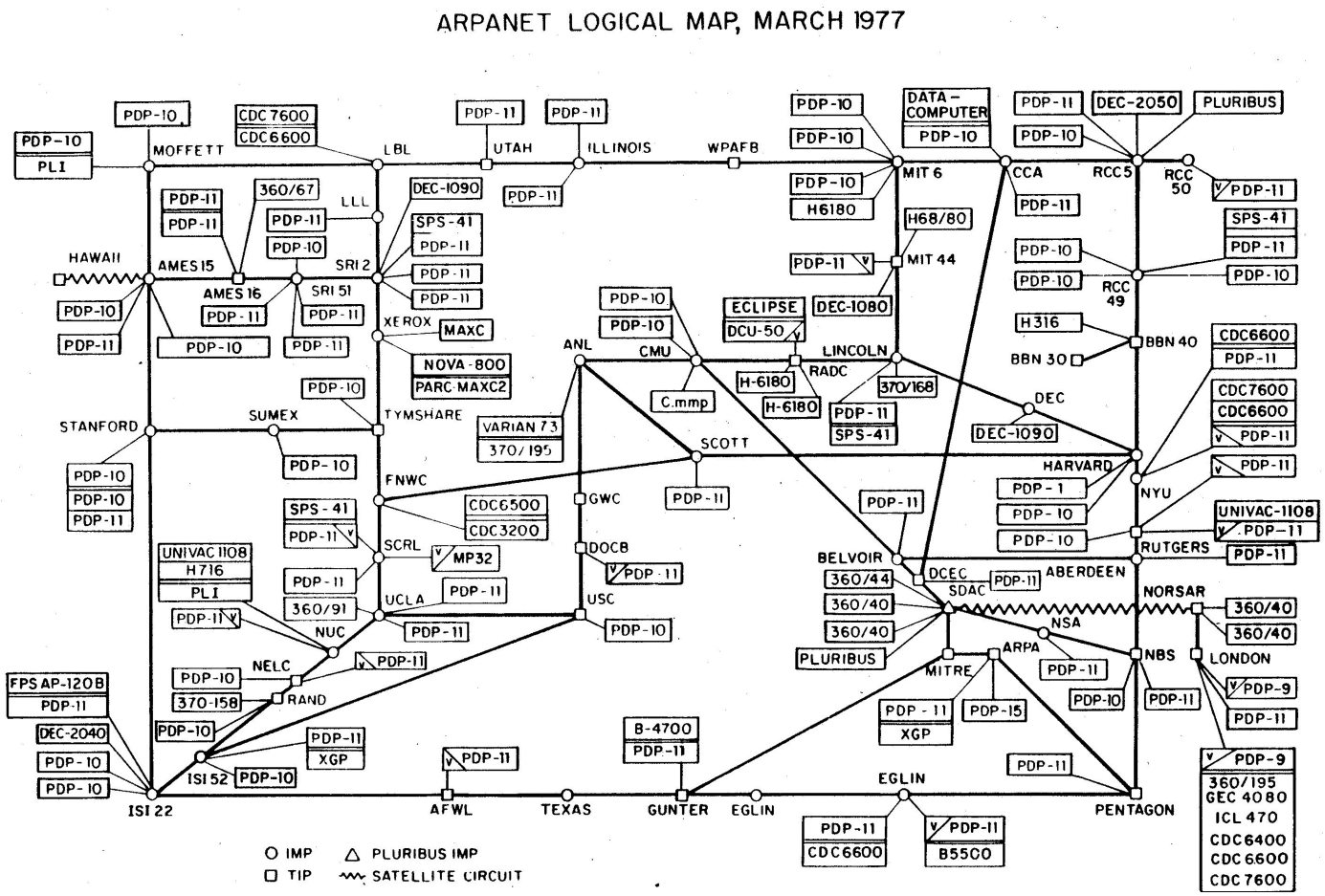
Paleolithic era

- 1965 Gordon Moore proposes law
- 1966 Design of ARPAnet

- 1969 First ARPAnet msg, UCLA -> SRI
- 1970 ARPAnet spans country, has 5 nodes
- 1971 ARPAnet has 15 nodes
- 1972 First email programs, FTP spec

The internet ramps up

- 1983 ARPAnet uses TCP/IP; design of DNS; 1000 hosts on ARPAnet



The internet ramps up

- 1983 ARPAnet uses TCP/IP; design of DNS; 1000 hosts on ARPAnet
- 1985 symbolics.com (computer mfg) is first registered domain name

- 1988 Robert Morris accidentally takes over the Internet
- 1989 100K hosts on Internet
- 1990 Cisco goes public; Tim Berners-Lee creates WWW at CERN; 3M Internet users world-wide

Modern age

- 1993 WWW Wanderer
  - First crawler
- 1995 Yahoo, Amazon
- 1998 Google & PageRank
- 2003 Skype

Modern age

- 2004 Facebook founded
- 2006 Twitter founded
- 2010 Instagram founded
- 2019 Facebook has 2.4 B monthly active uses
  - ~30% of humanity
  - https://investor.fb.com/investor-events/