IP - address → the address of the host server

TCP - service specifier ~~(443 for~~

- a single server can have multiple TCP numbers, one for each service (email, web, chat, etc)

Week 8

- The number is a port number
- e.g. 80 — HTTP (request for web page)
  443 — HTTPS (secure request)
  email, zoom etc. have their own predesignated numbers

TCP/IP → ~~####~~ - ~~###~~ ... 80

IP ↑ port at end

TCP also resends data when packets are lost

TCP/IP protocols support adaptive routing, so data can be sent via different paths depending on varying circumstances

Sophisticated Requests

GET /search?q=cats HTTP/1.1

↓

and version of typing cats into google search bar

like a CLI command

?q={word}

↓

like "what is the value of q?"

input: cats

[program]

↓

google search results

protocols
1 TCP/IP - address on outside of
"envelope"
~~DNS - used to translate url to IP~~
2 HTTP(S) - contents of envelope
like "Dear server"

↓
What's inside envelope below
HTTP header?

HTML!!
Hypertext Markup Language

<form>
action = "url" ← [illegible]
method = "get"

url parameters { <input name="" type="search"
<input type="submit" value="search"

Form tags are ubiquitous
on sites requiring user input
- open/close tags
- id or name attributes

③ - form tags
- POST (vs get) hides
the input info from url.

CSS - Cascading Stylesheets

<style> tag in header

OR

<link href="..." /> ← in header

style="font-size: large; text-align: center"

↑ separates properties

© → ©

↓

HTML Entity

symbols not on keyboard

You can use style sheets to create a list of classes that can be plugged in to an html file with <link href="styles.css" rel="stylesheet">

Javascript

when submit is pressed run this function

<form onsubmit = "greet();

don't submit form to server → return false;" >

<input id = "name"

this allows us to refer to it in our function → type = "text" >

<input type = "submit"

document.querySelector('#name').value

document ↑ special global variable in js. ↑ access to all things related to webpage

Selector ↑ method that allows us to select CSS selector (nodes from tree → the DOM) Document Object Model

We can also add
all kinds of attributes
autocomplete = "off"
document.querySelector. autofocus = "on"
etc.

^ .addEventListener('submit', greet)

↓

when the user does anything on the page

Or when anything happens on the page at all, such as finished loading

when submit is activated run the greet function (like the lambda function from week 7)

don't call greet

Similar to prototypes in C
↓
In order to use .addEventListener
on a DOM object, you
need to have a line ~~of code~~
~~that~~ listens for the ~~entire~~
DOM via("DOMContentLoaded")
↑
so it looks like this

```
function rest()
{
}
function listen()
{
    addEventListener
}
addEventListener('DOMContent...' listen)
```

↑ Function w/ addEventListener

Like python you can shorten functions as w/ lambda

This time you give write the **whole** function as the 2nd arg of the method.

```
add.EventListener('DOMContentLoaded' function()
    {
      ~~~~
    });
    ^
```

close parentheses to indicate end of method call

You can continue to nest functions in arguments.

PUT

↓

3rd request in addition

to POST and GET

↓

PUT sends to