

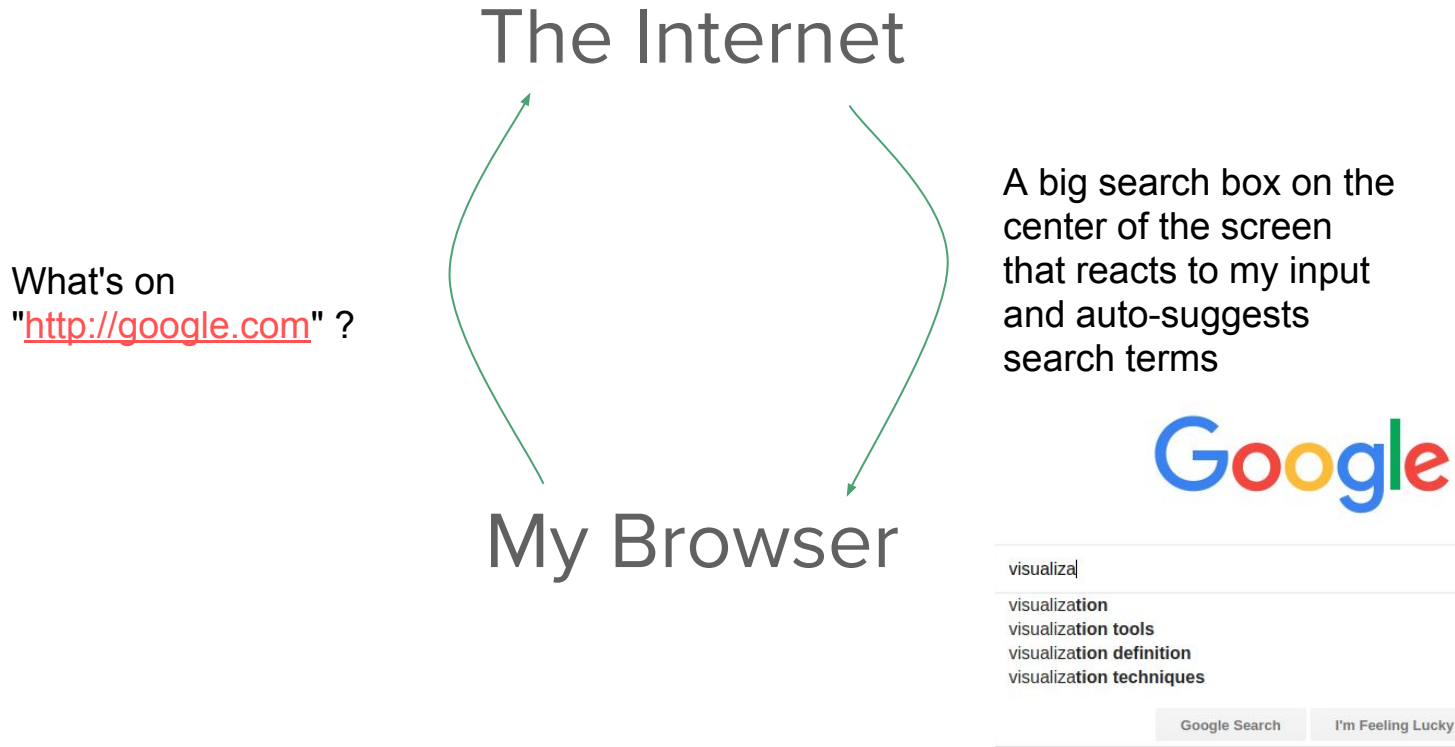
Web Development Techniques or: Everything you need to know to build a website using JavaScript

Lecture on September 21, 2017

Today's Class

- What does it mean to "open a website"
- Web languages - client, and DOM
- Web languages - server
- Test beds and distribution

What does it mean to open a website?



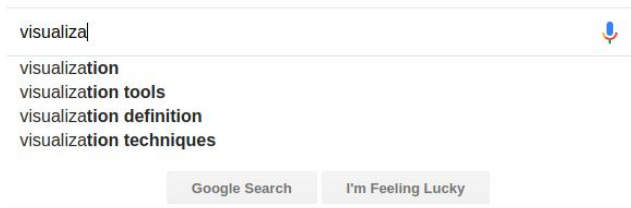
Request

The Internet



My Browser

A big search box on the center of the screen that reacts to my input and auto-suggests search terms

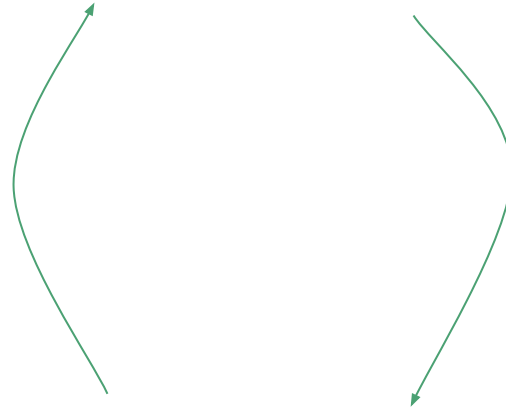


Request: What Does It Do? It specifies...

- Which server you want to communicate with: google.com
- Which protocol you're using to communicate: HTTP (vs HTTPS, FTP, GIT, SSH and many others)
- What type of communication you want
 - Do you want something? GET
 - Are you sending something new? POST
 - Are you changing something? PUT
 - Are you deleting something? DELETE
- What you want to communicate
 - What document are you requesting / creating / updating / deleting?
/ or /search?q=visualization&source=hp
 - What content are you creating / updating?

The Internet

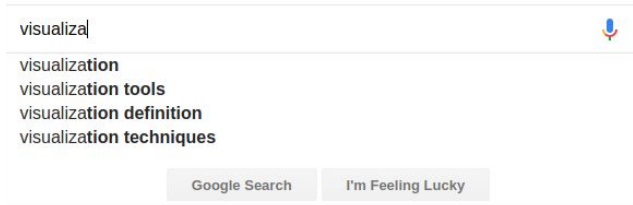
Request
GET google.com



My Browser

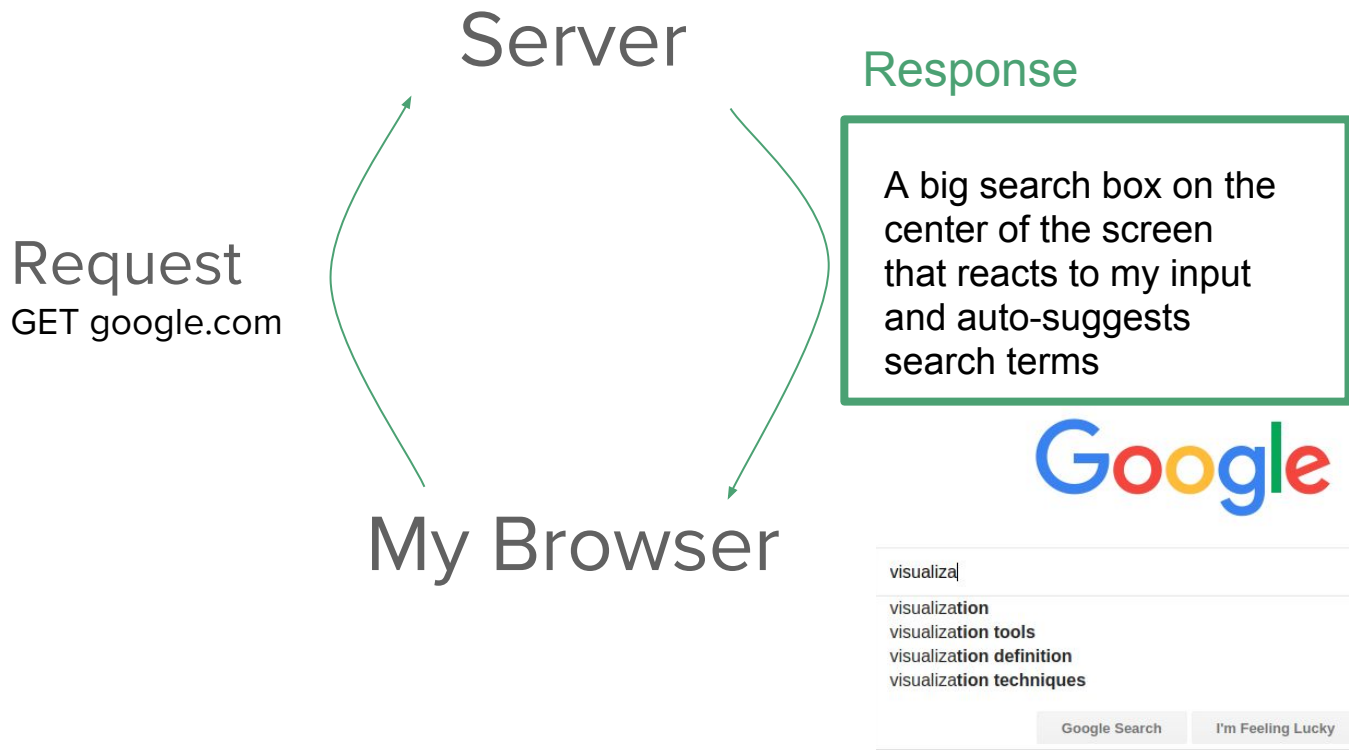
A big search box on the center of the screen that reacts to my input and auto-suggests search terms

Google



Internet: What does it do with my GET Request?

- Using a "phone book", it finds the address of the server that's supposed to respond to requests to "google.com". Said differently:
Domain Name Servers (DNS) resolve your host name (google.com) to an IP address.
- This way, your request is sent to a server at Google.
- Then, google's server parses your request
Eg, "/" = "Show them the start page", and "/search?q=visualization" = "Show them the results for a 'visualization' search"



What does the Response do?

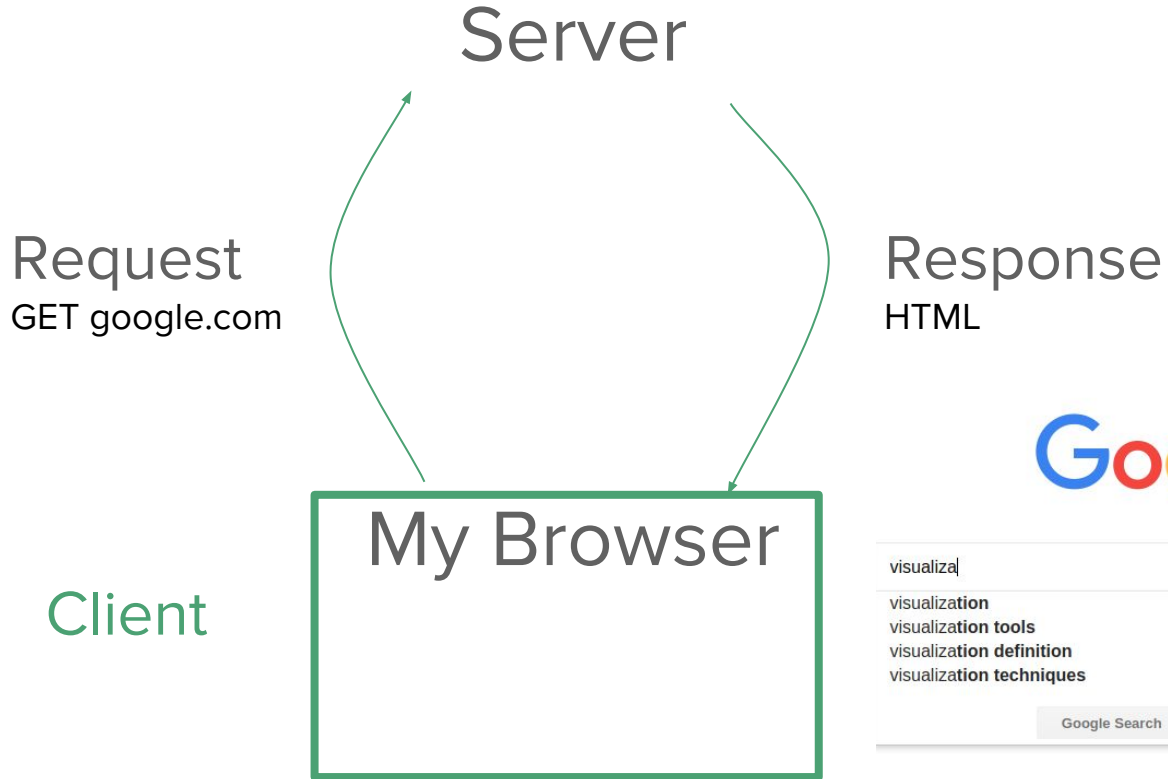
- It says whether the request was successful
 - 404
 - Not Found
 - 200
 - OK
 - 301
 - Moved Permanently
 - 304
 - Not Modified
 - 401/403?
 - Access Problem
 - 500
 - Internal Server Problem

What does the Response do?

- Request Status: 404 Not Found, 200 OK, 301 Moved Permanently, 304 Not Modified, 401/403 Access Problem, 500 Internal Server Problem
- Sends caching information (keep alive)
- Sends the response content

So, what is the "response content"? Or: What is a website?

- HTML
 - Hypertext Markup Language defines the **structure and content** of a web document
 - HTML is a type of XML
- CSS
 - Cascading Style Sheets define the **visual appearance** of the elements of a website
- JS
 - JavaScript defines the **behavior** of the website
 - Nothing to do with Java



What would a HTML-only page look like?

You can try it by saving the page, and removing the CSS.



[More](#)



[Shopping](#)

- [Wallet](#)
- [Finance](#)
- [Docs](#)
- [Books](#)
- [Blogger](#)
- [Contacts](#)
- [Hangouts](#)
- [Keep](#)
- [Classroom](#)

[Even more from Google](#)



[Change](#)

Micha Schwab

michaschwab@gmail.com

[Google+ Profile-Privacy](#)

[My Account](#)

Micha Schwab

michaschwab@gmail.com (default)

Google

What does my browser do with the response HTML?

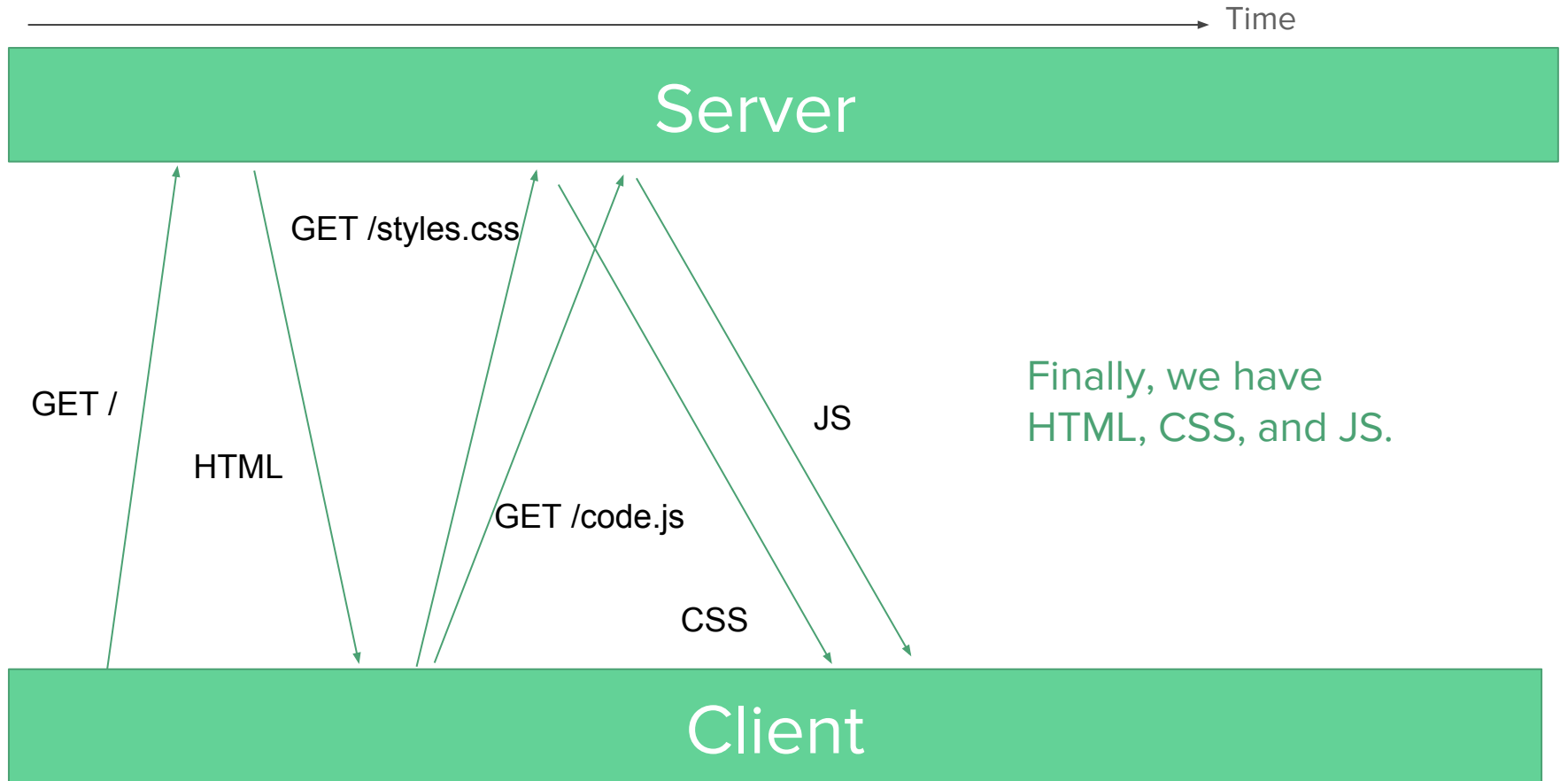
- Google's response looks something like this.
- It sends
 - Metainformation, like a description and a title
 - Some content
 - Links to more files, like JavaScript and CSS!
- The browser then requests those files as well.

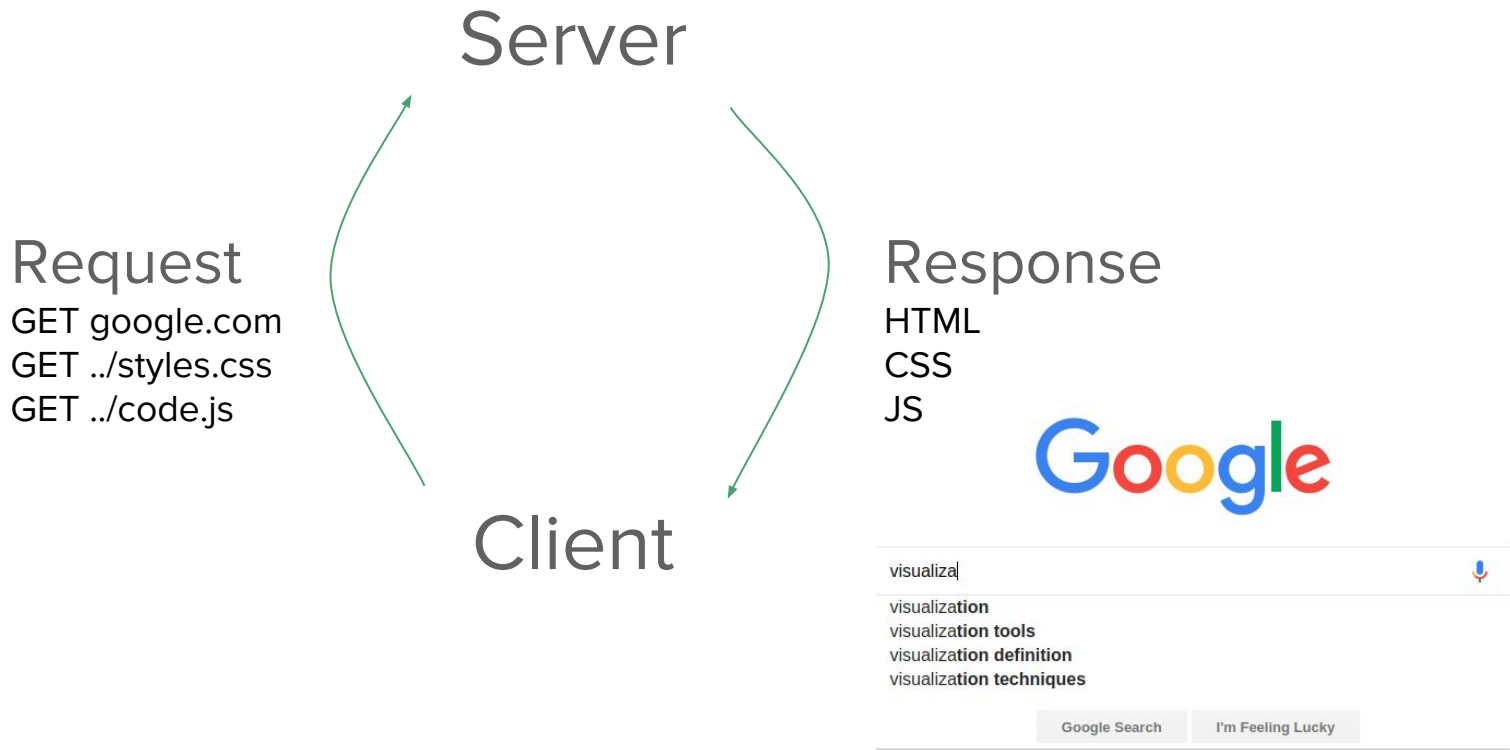
```
<!doctype html>
<html lang="en">
  <head>
    <meta name="description"
          content="Search the world's information,
          />
    <title>Google</title>
    <link href="styles.css" />
    <script src="javascript.js"></script>
  </head>
  <body>
    <!-- Some content here. -->
  </body>
</html>
```

How do the requests play out?

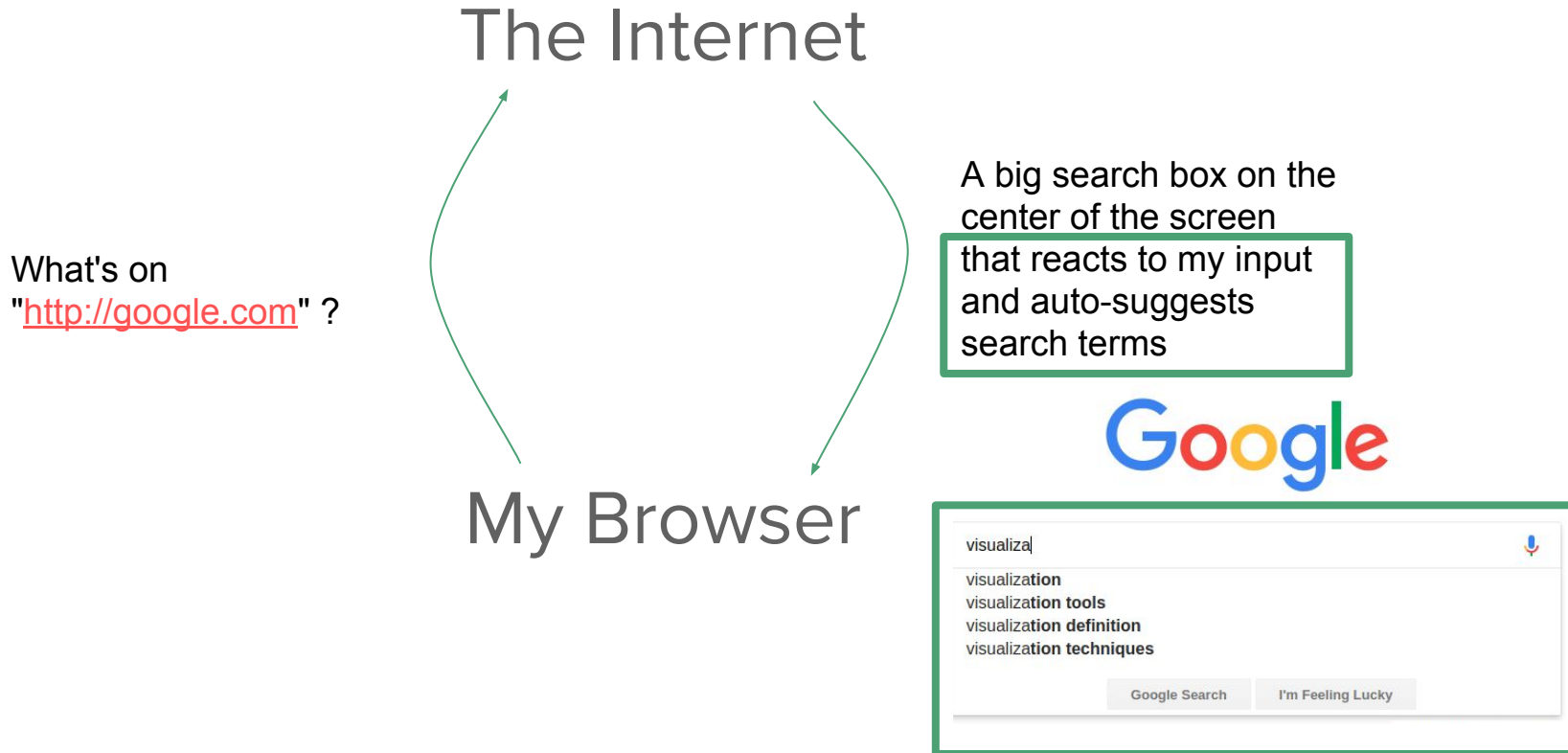
Let's take a look on a live demo of requests on [google.com](https://www.google.com)!

Client Server Ping Pong

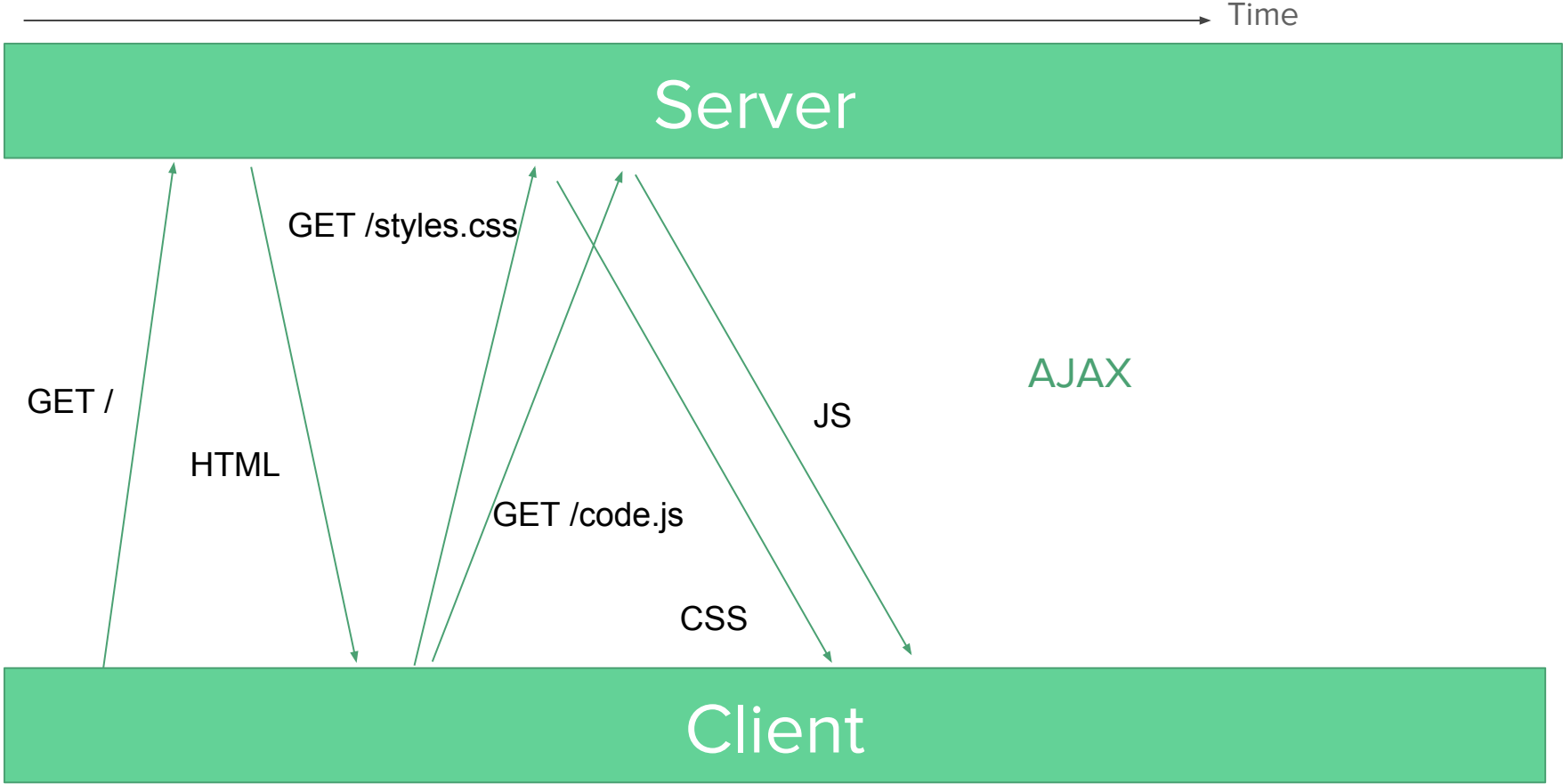




What of what you see below is still missing?



Client Server Ping Pong



AJAX

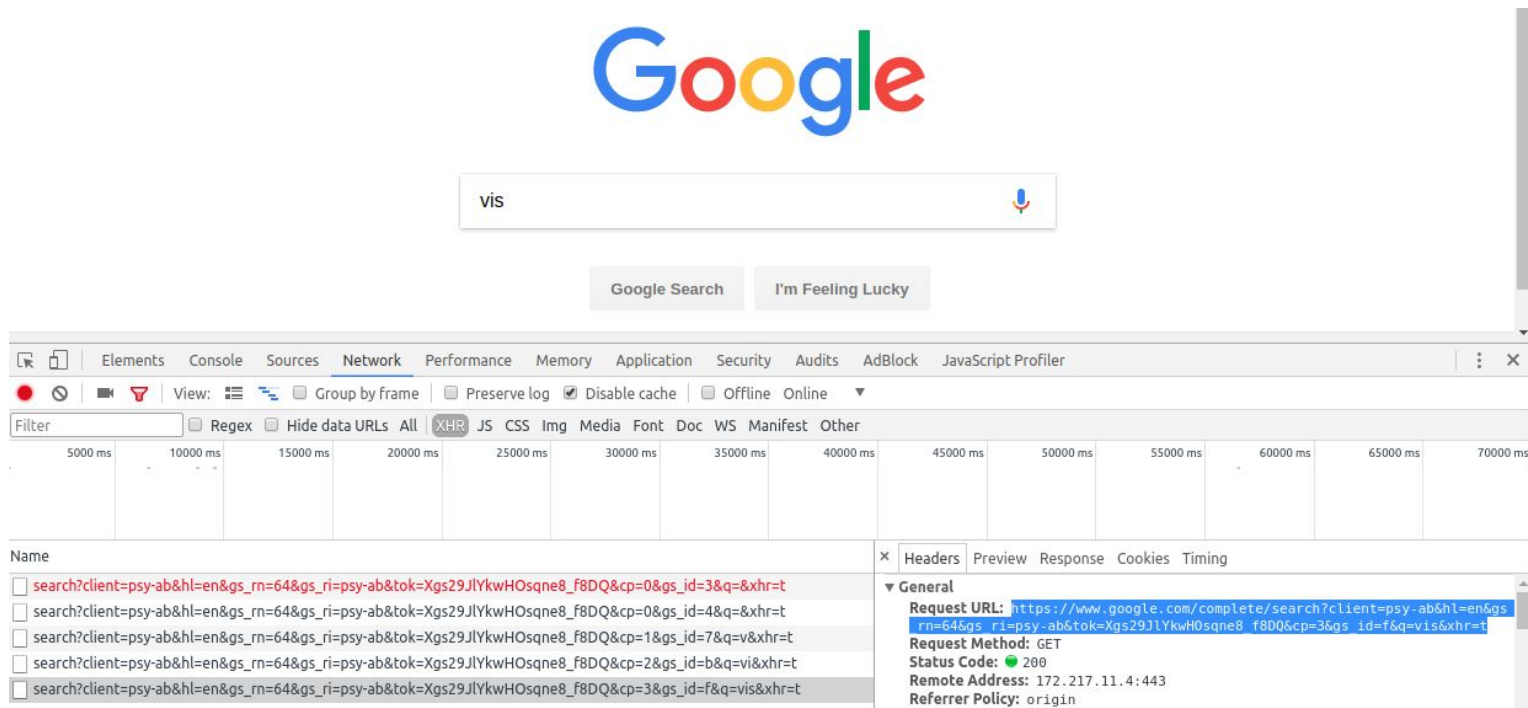
Asynchronous JavaScript and XML

Allows (re-) loading parts of a webpage, or loading additional content without re-opening the entire webpage

Can be initiated by **JS** (and other languages like Microsoft's ASP)

What do the AJAX requests look like?

Let's take a look!



The image shows a Google search page with the search bar containing the text "vis". Below the search bar are the "Google Search" and "I'm Feeling Lucky" buttons. The browser's developer tools are open to the Network tab, showing a list of requests. The selected request is a GET request to the URL: `https://www.google.com/complete/search?client=psy-ab&hl=en&gs_l=64&gs_ri=psy-ab&tok=Xgs29JlYkwH0sqne8_f8DQ&cp=0&gs_id=3&q=vis&xhr=t`. The request details are shown in the right pane, including the Request Method (GET), Status Code (200), Remote Address (172.217.11.4:443), and Referrer Policy (origin).

Google

vis

Google Search I'm Feeling Lucky

Elements Console Sources Network Performance Memory Application Security Audits AdBlock JavaScript Profiler

View: [Icons] Group by frame Preserve log [x] Disable cache [x] Offline Online [v]

Filter [] Regex [] Hide data URLs All [x] XHR JS CSS Img Media Font Doc WS Manifest Other

Name	Headers	Preview	Response	Cookies	Timing
<input type="checkbox"/> search?client=psy-ab&hl=en&gs_l=64&gs_ri=psy-ab&tok=Xgs29JlYkwH0sqne8_f8DQ&cp=0&gs_id=3&q=vis&xhr=t					
<input type="checkbox"/> search?client=psy-ab&hl=en&gs_l=64&gs_ri=psy-ab&tok=Xgs29JlYkwH0sqne8_f8DQ&cp=0&gs_id=4&q=&xhr=t					
<input type="checkbox"/> search?client=psy-ab&hl=en&gs_l=64&gs_ri=psy-ab&tok=Xgs29JlYkwH0sqne8_f8DQ&cp=1&gs_id=7&q=v&xhr=t					
<input type="checkbox"/> search?client=psy-ab&hl=en&gs_l=64&gs_ri=psy-ab&tok=Xgs29JlYkwH0sqne8_f8DQ&cp=2&gs_id=b&q=vi&xhr=t					
<input type="checkbox"/> search?client=psy-ab&hl=en&gs_l=64&gs_ri=psy-ab&tok=Xgs29JlYkwH0sqne8_f8DQ&cp=3&gs_id=f&q=vis&xhr=t					

General

Request URL: https://www.google.com/complete/search?client=psy-ab&hl=en&gs_l=64&gs_ri=psy-ab&tok=Xgs29JlYkwH0sqne8_f8DQ&cp=3&gs_id=f&q=vis&xhr=t

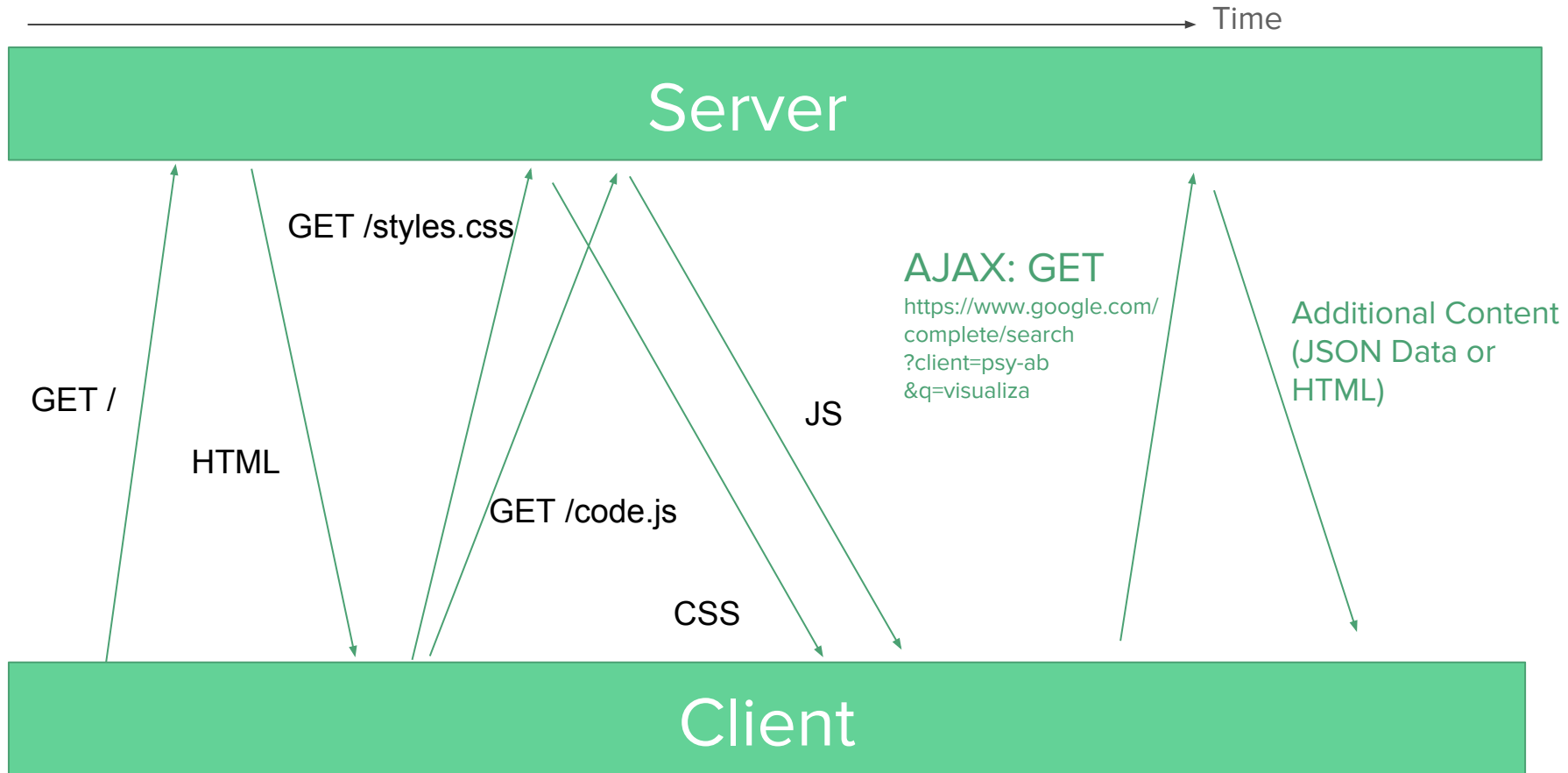
Request Method: GET

Status Code: 200

Remote Address: 172.217.11.4:443

Referrer Policy: origin

Client Server Ping Pong



Overview of web languages - local

- HTML, DOM
- CSS
- JS, JSON

Take a look in the inspector!

Do we need a server?

Almost always yes: Browsers have strong restrictions on what pages can do that were just opened from the file system.

- We need a server to distribute the website and keep it online
- We need a server to test on: local.

Options for web server languages

- Java
- Apache (static), or + PHP
- **Python**, e.g. [SimpleHTTPServer](#)
- **JavaScript**: NodeJS + [Express](#), e.g. [hello world](#)
- Databases: SQL, or NoSQL like Mongo. Firebase

If you have a lot of data and need to operate on parts of it, you will need to use AJAX and/or a database to look up and send relevant data

Hosting Locally

- show python server
- show express server

HTML, CSS, JS: How to code

- A bit much to cover in 1 class
- Let's start with the example from before:
- Every HTML document has this structure:
 - Head for meta information,
 - Body for content.
-

```
<!doctype html>
<html lang="en">
  <head>
    <meta name="description"
          content="Search the world's information,
          />
    <title>Google</title>
    <link href="styles.css" />
    <script src="javascript.js"></script>
  </head>
  <body>
    <!-- Some content here. -->
  </body>
</html>
```

HTML Elements - Some common ones

- **div** (divisions) are useful for structuring and positioning
- **p** (paragraphs) are used for text
- **h1-h5** are used for titles and subtitles
- **input** are used for text input, radio buttons, submit buttons and almost all other inputs (except for textarea)
- **a** (anchor) are used for links, both internal and external.
- **ul/ol and li** (unordered/ordered list and list items) are used for lists.
- **span** are used for styling parts of a text

Browsers have default settings for displaying each of these elements.

HTML5: Adds many elements, most of which are visually equivalent to divs, but add more semantic meaning - such as section. Good, but not required.

CSS: Cascading Style Sheets

- Applies styles to DOM elements and their children.
- To address an element, use
 - elementtype, e.g. `<input />` with `input { width: 200px }`
 - .classname, e.g. `<input class="search" />` with `.search { width: 200px }`
 - #element-id, e.g. `<input id="search" />` with `#search { width: 200px }`
- To address an element that fulfills two requirements, write them right after each other
 - e.g. `<input class="search" />` with `input.search { .. }`
- To address an element's children that fulfill some requirement, write them with a space between each other:
 - e.g. `<div class="search"><input /></div>` with `.search > input { .. }`

CSS: Some styles for this example

- text-align: left, center, right
- margin: 3px 2px bottom left, or margin-top: [X]px.
 - Same for padding
- width: [X]px
- background: #hex
- border: [width]px [solid/dashed/..] #hex
- font-weight: [normal/bold]
- color: #hex for text color
- box-shadow: [X]px [Y]px [BLUR]px #hex

Live coding time Step 1: CSS

Ok, with that CSS knowledge, let's start building google!

<http://tiny.cc/makegoogle>

- Click "Fork" to work on your own version of the jsfiddle
- Start styling it to look as close to possible to the original Google Page

JS

- Very wild and type-free and chaotic - unless you prevent it
- Very asynchronous!
- `document.getElementById`
- `element.addEventListener(eventType, callbackFunction)`
tells the element to call the `callbackFunction` whenever a event of type `eventType` is called.
- This example uses jQuery, which is very very common and used on most websites. It makes a lot of basic functionality easier.

d3: Data Driven Documents

"Data Binding"

<https://d3js.org/#enter-exit>

- enter() for new nodes
- exit() for removed nodes
- without any specific method to update.

```
d3.select("body")
  .selectAll("p")
  .data([4, 8, 15, 16, 23, 42])
  .enter().append("p")
    .text(function(d) { return "I'm number " + d + "!"; });
```

Updating nodes are the default selection—the result of the data operator. Thus, if you forget about the enter and exit selections, you will automatically select only the elements for which there exists corresponding data. A common pattern is to break the initial selection into three parts: the updating nodes to modify, the entering nodes to add, and the exiting nodes to remove.

```
// Update...
var p = d3.select("body")
  .selectAll("p")
  .data([4, 8, 15, 16, 23, 42])
  .text(function(d) { return d; });

// Enter...
p.enter().append("p")
  .text(function(d) { return d; });

// Exit...
p.exit().remove();
```

Live coding time

Ok, let's build google step 2: JS. I prepared a tiny server that sends back autocomplete suggestions, e.g:

<http://michaschwab.de:3001/autocomplete?q=visualiza> →

```
{"term":"visualiza","autocompletes":["visualiza 0","visualiza 1"]}
```

The suggestions are passed to a function "onAutocompletesChange". Use it to display the suggestions as links (<a>) in the "autocompletes" div.

<http://tiny.cc/makegoogle-two>

- Click "Fork" to work on your own version of the jsfiddle
- See what you get in the console after typing in the search field
- edit onAutocompletesChange to use the results!

Version Control: git (most basic)

- **clone** to "copy" an existing repository into your current folder. you can use this to create new ones too, after having them created by github.
- **add** to add files to be under version control
- **commit** to save your current changes as a "stage" of the project, but **this is local.**
- **push** to upload your local commits
- **pull** to download remote pushed commits

- **checkout** a file you have made local changes to to override/revert your local changes
- **stash** changes that are in conflict with remote changes before pulling and merging

Sandboxes / Distributing / Publishing

- [JSFiddle](#): Live coding. Recommended for debugging and asking for help
 - Create a minimal reproducible example of your problem.
 - Most of the time, this process helps find the bug before help is even asked.
- [Github Gist](#): Share code, e.g. [this example](#)
- + [Blocks](#): Show the website, e.g. [this example](#)
- Github Pages: Automatic live version of a project's code.

All of these are simple http servers and don't allow you to run anything server side.

Notes

JS Versions: ECMA5/6, CoffeeScript, TypeScript all fine by me

Editor: I really like webstorm. some like Atom (free), and many others

SVG: Scalable Vector Graphics

- Used for most interactive visualizations on the web
- Good: Everything is reflected in the DOM.
 - Easy debugging
 - using all our developer tools
 - easy to edit
- Bad: Performance scales with the number of elements
- Alternative: WebGL.

SVG elements

- basic shapes: circle, rect, line
- group elements that can be used to move groups of elements around together
- paths

Confusing amount of ways to position elements:

- `transform="translate(X, Y)"` for groups
- `x1="X", y1="Y"` for rects and lines
- `dx, dy, and x, and y, etc.` → Look up the spec for the specific element.

AngularJS - if you want!

Easier data binding

Not required for this course. Can make some things easier

Hope it helped!

- If you haven't coded using JS before, get some practice now before you have to code a nodelink visualization for your homework.
 - Maybe create some simple bar charts or something. You can also start looking at some d3 examples on <https://d3js.org/> even though d3 will be covered in more detail next week.
- I'm happy to help
 - Specially if you don't know what to learn, what to focus on, what approach to take, what to do.
 - If you need help with a bug, then I can help if you do your "homework" first. Don't ask me questions that have an easy google/stackoverflow answer.
 - If you send me a condensed jsfiddle, your chances of getting a prompt and helpful reply are increased at least tenfold.