

# Extracting data from the web

## APIs and beyond



Scott Chamberlain  
Karthik Ram  
Garrett Grolemund

June 2016

**HELLO**

my name is

**Scott**



@sckottie

# Outline

## Part 1 - Collecting data from an API



**Scott Chamberlain**

Co-Founder

**R**OpenSci

# Outline

**Part 1** - Collecting data from an API

**Part 2** - Wrapping an API with R



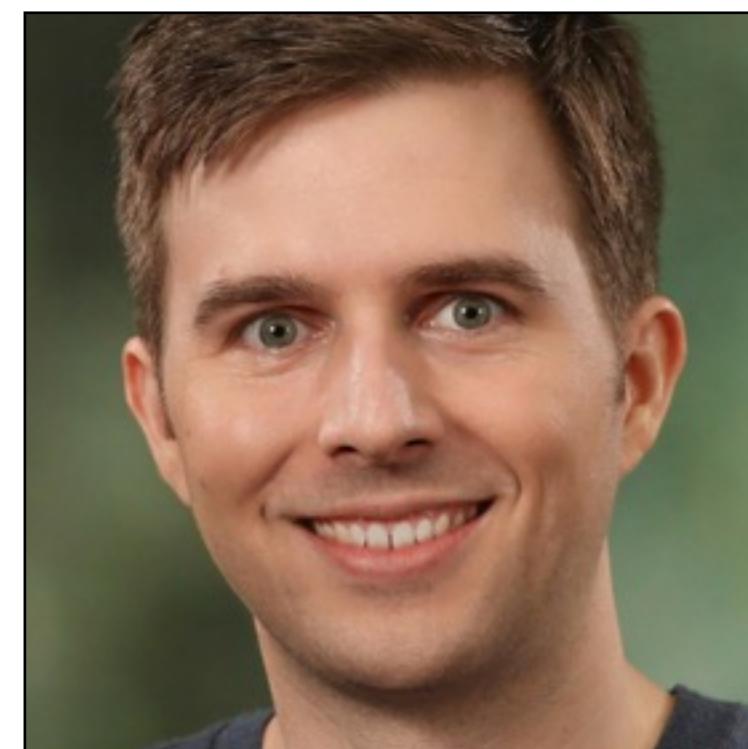
**Karthik Ram**  
Data Scientist and  
Ecologist and co-founder  
**R**OpenSci

# Outline

**Part 1** - Collecting data from an API

**Part 2** - Wrapping an API with R

**Part 3** - Scraping data without an API



**Garrett Grolemund**

Master Instructor and  
Data Scientist



# Your Turn

Introduce yourself to the people at your table.

Determine who among you has the most web development experience.



# APIs

# Intro

**HELLO**

my name is

**Scott**



@sckott / @ropensci / @pdxrlang

# Outline

- 1. What is an API?**
- 2. HTTP**
- 3. HTTP verbs**
- 4. HTTP structure**
- 5. Data formats**
- 6. Wrap up**

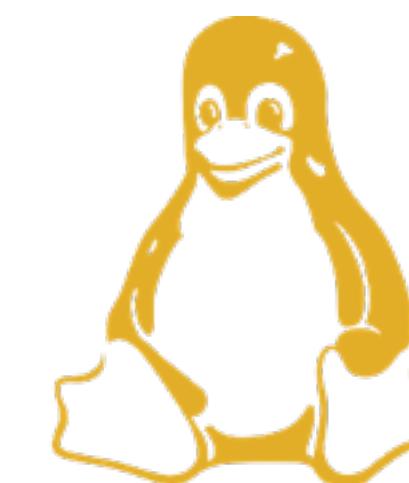
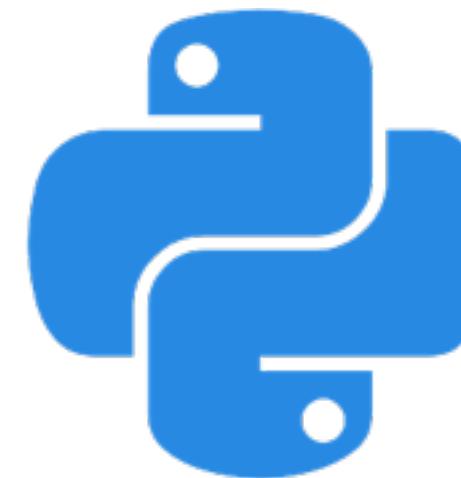
**What is an  
API?**

# An API is...

Programmatic instructions for how to interact  
with a piece of software

Can be the interface to:

- A software package in R/Python/etc.
- A public web API
- A database
- An operating system



# Most APIs are REST APIs

# **REST? WTF?**

**Representational State Transfer**

an architectural style in which most web APIs are constructed

[https://en.wikipedia.org/wiki/Representational\\_state\\_transfer](https://en.wikipedia.org/wiki/Representational_state_transfer)

# HTTP

**HyperText Transfer Protocol**

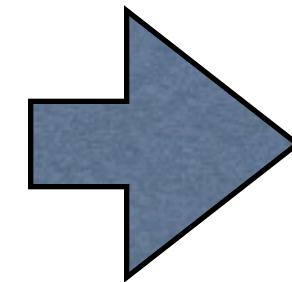
HTTP spec: <https://tools.ietf.org/html/rfc7235>

- Verbs for different actions
- Authentication
- Status codes
- Request and response format
- Most REST APIs use HTTP for data transfer

# But, what does it all look like?

Server

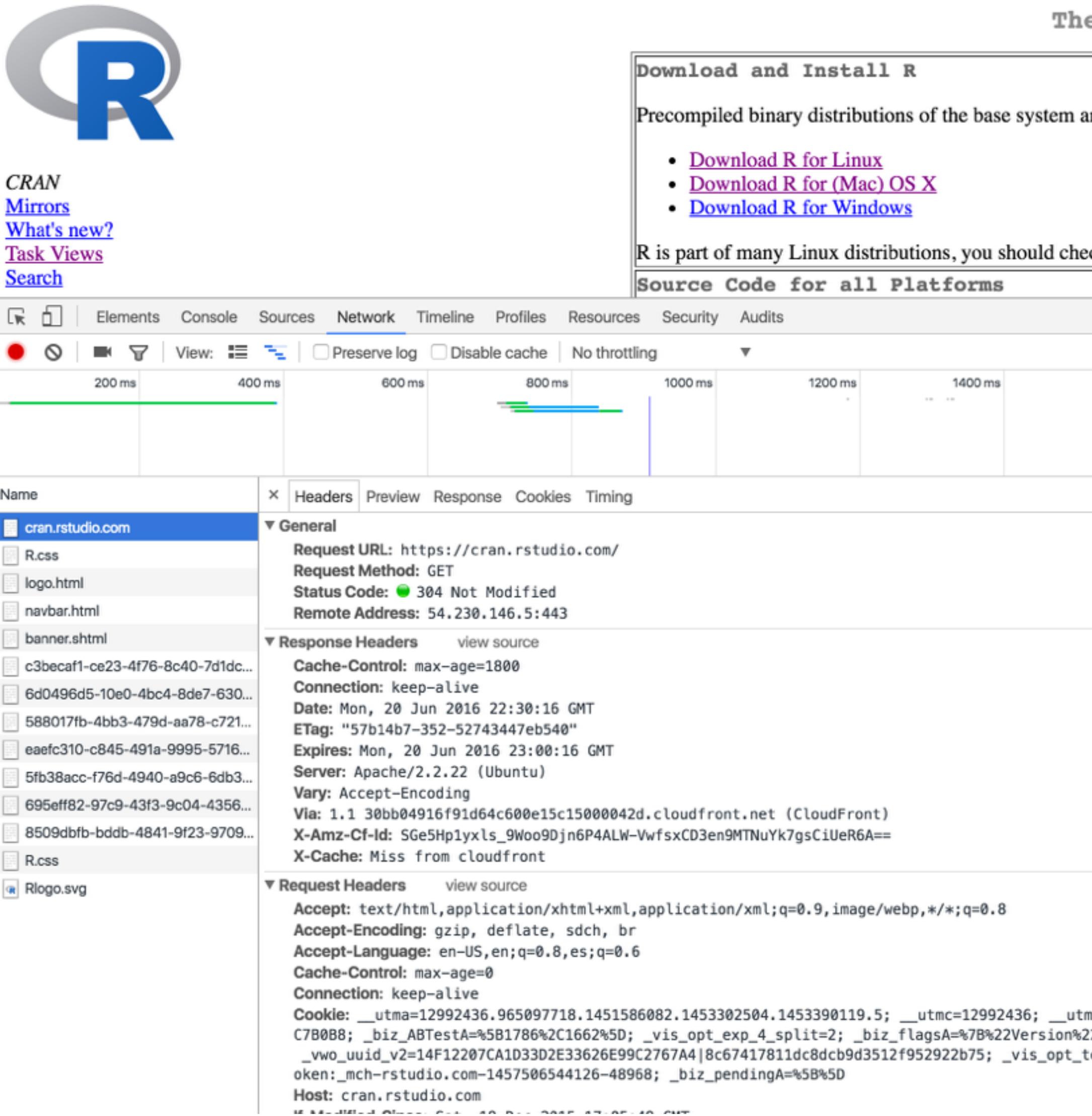
http server: nginx  
API: sinatra  
caching: redis  
database: postgresql



Client

R: httr  
ruby: faraday  
python: httpie  
browser: chrome

# HTTP is behind the scenes



The screenshot shows a browser developer tools interface, specifically the Network tab of the Chrome DevTools. The main content area displays a list of network requests made to the URL `https://cran.rstudio.com/`. The requests listed include:

- `R.css`
- `logo.html`
- `navbar.html`
- `banner.shtml`
- `c3becaf1-ce23-4f76-8c40-7d1dc...`
- `6d0496d5-10e0-4bc4-8de7-630...`
- `588017fb-4bb3-479d-aa78-c721...`
- `eaefc310-c845-491a-9995-5716...`
- `5fb38acc-f76d-4940-a9c6-6db3...`
- `695eff82-97c9-43f3-9c04-4356...`
- `8509dbfb-bddb-4841-9f23-9709...`
- `R.css`
- `Rlogo.svg`

For each request, detailed information is provided in the "General" section:

- Request URL: `https://cran.rstudio.com/`
- Request Method: GET
- Status Code: 304 Not Modified
- Remote Address: 54.230.146.5:443

Below the General section, Response Headers and Request Headers are expanded to show their respective details.

The Network tab also includes a timeline at the top showing the duration of each request, and a list of resources on the right side of the interface.

# HTTP in R

You've been using HTTP in R, e.g.:

- `install.packages()` -> uses `download.file()` under the hood -> which uses essentially http
- and many others ...

# Your Turn

## httr hello world

- If **httr** is not installed, install it
- Load **httr**
- Use **httr::GET()** to get data from any website.
  - Poke around at the resulting object.
  - Find the *headers*, the *status code*, and the *content*



```
x <- GET('https://google.com/')
```

```
x$status_code  
#> [1] 200
```

```
x$headers  
#>$date  
#>[1] "Thu, 23 Jun 2016 23:05:27 GMT"  
#> ...
```

```
x$content  
#>[1] 3c 21 64 6f 63 74 79 70 65 20 68 ...
```

# **HTTP Verbs & Requests**

# HTTP Verbs

GET

Read

POST

Create

PUT

Update

DELETE

Delete

# HTTP Verbs

GET

Retrieve whatever is specified by the URL

POST

Create resource at URL with given data

PUT

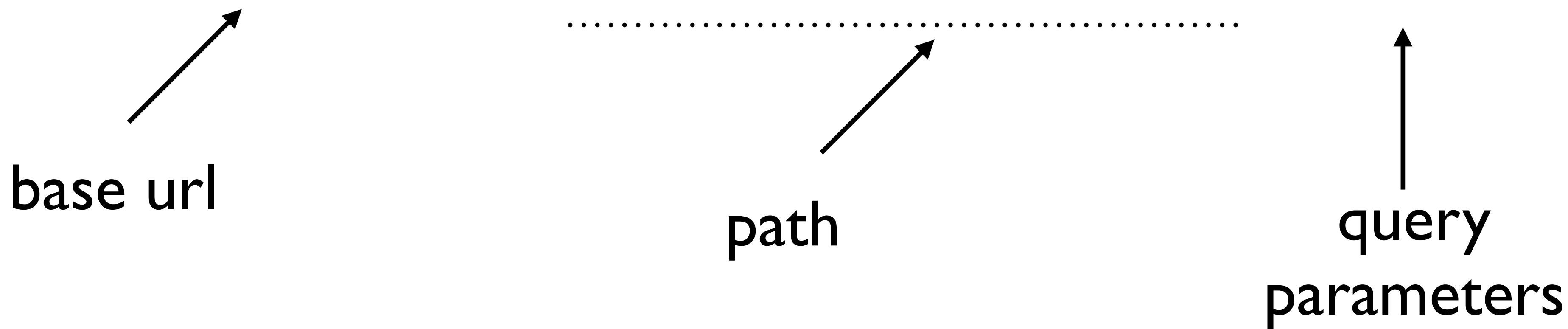
Update resource at URL with given data

DELETE

Delete resource at URL

# HTTP Verbs: GET

**GET** [https://api.github.com/repos/hadley/dplyr/issues?per\\_page=3](https://api.github.com/repos/hadley/dplyr/issues?per_page=3)



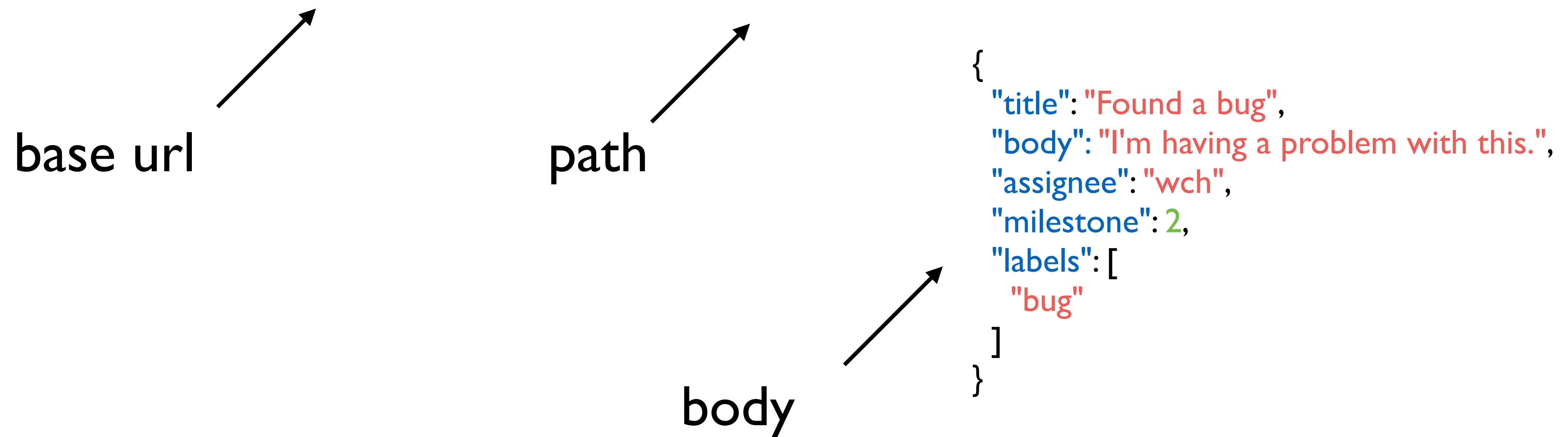
send to GitHub's servers



GitHub sends back data!

# HTTP Verbs: POST

**POST** <https://api.github.com/repos/hadley/dplyr/issues>



# HTTP Verbs: PUT

**PUT** https://api.github.com/repos/hadley/dplyr/issues/3

base url

path

body

{  
  "title": "Found a bug",  
  "body": "I'm having a problem with this.",  
  "assignee": "wch"  
}

issue  
#



# HTTP Verbs: DELETE

**DELETE**

`https://api.github.com/repos/sckott/foobar`

base url

path



# more HTTP Verbs

- HEAD - identical to GET, but just gets headers back
- PATCH - similar to PUT, but partially modify
- COPY - copy a resource from one URI to another
- OPTIONS - get what verbs supported for a URI
- a few others: TRACE, CONNECT

# Assembling Queries

## HTTP request components

- URL - where on the web do you want to make the request, including parameter values
- Method - what HTTP verb
- Headers - any metadata to modify the request
- Body - the data, very flexible, containing strings, files, binary, etc.

# Assembling Queries: in R

URL

`http://...`

Headers

`httr::add_headers()`

Method

`httr::GET()`  
`httr::POST()`  
`httr::PUT()`  
`httr::DELETE()`

Body

`httr::POST(body = list(foo = "bar"))`

...

# httpbin.org

## **httpbin(1): HTTP Request & Response Service**

Freely hosted in [HTTP](#), [HTTPS](#) & [EU](#) flavors by [Runscope](#)

### **ENDPOINTS**

- [`/`](#) This page.
- [`/ip`](#) Returns Origin IP.
- [`/user-agent`](#) Returns user-agent.
- [`/headers`](#) Returns header dict.
- [`/get`](#) Returns GET data.
- [`/post`](#) Returns POST data.
- [`/patch`](#) Returns PATCH data.
- [`/put`](#) Returns PUT data.
- [`/delete`](#) Returns DELETE data
- [`/encoding/utf8`](#) Returns page containing UTF-8 data.
- [`/gzip`](#) Returns gzip-encoded data.
- [`/deflate`](#) Returns deflate-encoded data.
- [`/status/:code`](#) Returns given HTTP Status code.
- [`/response-headers?key=val`](#) Returns given response headers.
- [`/redirect/:n`](#) 302 Redirects  $n$  times.
- [`/redirect-to?url=foo`](#) 302 Redirects to the *foo* URL.
- [`/relative-redirect/:n`](#) 302 Relative redirects  $n$  times.
- [`/absolute-redirect/:n`](#) 302 Absolute redirects  $n$  times.
- [`/cookies`](#) Returns cookie data.

# Your Turn

## httr verbs practice

- **GET** request to <https://httpbin.org/get>
- **POST** request to <https://httpbin.org/post>
- Try mismatching a httr method with a httpbin URL, what happens?

## Request Components

- Send a request with query parameters
- Send a request with a header
- Send a request with a body



```
url <- "https://httpbin.org/get"  
x <- POST(url)
```

```
x$status_code  
#>[1] 405  
METHOD NOT ALLOWED!!!!
```

```
x <- GET(url, query = list(a = 5))
```

```
x <- GET(url, add_headers(wave = "hi"))
```

```
x <- GET(url, body = list(a = 5))
```

# **HTTP**

# **Responses**

# HTTP response components

- **status** - status of the response
- **headers** - response headers, like content type, size of body, paging info, rate limit info, etc.
- **body/content** - many different types, compressed or not, binary or not, etc.

# status

- 3 digit numeric code
- One of 5 different classes of codes:
  - **1xx**: informational
  - **2xx**: success
  - **3xx**: redirection
  - **4xx**: client error
  - **5xx**: server error
- Info on status codes: [https://en.wikipedia.org/  
wiki/List\\_of\\_HTTP\\_status\\_codes](https://en.wikipedia.org/wiki/List_of_HTTP_status_codes)
- In R: [https://cran.rstudio.com/web/packages/  
httpcode/](https://cran.rstudio.com/web/packages/httpcode/) for HTTP status code look up

# status: beware

- Servers do not always give correct codes
- Clients may pass on these inappropriate codes
- i.e., Don't trust status codes alone - use in combination with other information:
  - content type
  - body length
  - etc.

# Your Turn

Look up different status codes by using

<https://http.cat/<HTTP STATUS CODE>>



# 418: “I’m a teapot”

<https://http.cat/418>



# headers

- Contain metadata about the **Request & Response**
- Some headers standardized
- Some headers custom for the web service
- Most headers **key:value** pairs
- Some headers just **value** without a key

# headers

<http://httpbin.org/get>

## request

GET /get HTTP/1.1  
Accept: \*/\*  
Accept-Encoding: gzip, deflate  
Connection: keep-alive  
Host: httpbin.org  
User-Agent: HTTPie/0.9.2

## response

HTTP/1.1 200 OK  
Access-Control-Allow-Credentials: true  
Access-Control-Allow-Origin: \*  
Connection: keep-alive  
Content-Length: 228  
Content-Type: application/json  
Date: Wed, 22 Jun 2016 16:12:04 GMT  
Server: nginx

# content / body

```
x <- GET('https://google.com/')
```

```
x$content
```

```
#> [1] 3c 21 64 6f 63 74 79 70 65 20 68 ...
```

```
content(x)
```

to extract data

raw bytes

More in Part II

# Your Turn

- Get status code from an httr response object
- Use httr to figure out what the code means
- From a http response: Get request & response headers -> Then extract content type



# Data Formats

# JSON

<http://www.omdbapi.com/?t=frozen&y=&plot=short&r=json>

```
{  
    "Title": "Frozen",  
    "Year": "2013",  
    "Rated": "PG",  
    "Released": "27 Nov 2013",  
    "Runtime": "102 min",  
    "Genre": "Animation, Adventure, Comedy",  
    "Director": "Chris Buck, Jennifer Lee",  
    "Writer": "Jennifer Lee (screenplay), Hans Christian Andersen (story inspired by \"The Snow Queen\" by), Chris Buck (story by), Jennifer Lee (story by), Shane Morris (story by)",  
    "Actors": "Kristen Bell, Idina Menzel, Jonathan Groff, Josh Gad",  
    "Plot": "When the newly crowned Queen Elsa accidentally uses her power to turn things into ice to curse her home in infinite winter, her sister, Anna, teams up with a mountain man, his playful reindeer, and a snowman to change the weather condition.",  
    "Language": "English, Icelandic",  
    "Country": "USA",  
    "Awards": "Won 2 Oscars. Another 70 wins & 56 nominations.",  
    "Poster": "http://ia.media-imdb.com/images/M/MV5BMTQ1MjQwMTE5OF5BMl5BanBnXkFtZTgwNjk3MTcyMDE@._V1_SX300.jpg",  
    "Metascore": "74",  
    "imdbRating": "7.6",  
    "imdbVotes": "410,734",  
    "imdbID": "tt2294629",  
    "Type": "movie",  
    "Response": "True"  
}
```

# jsonlite

<https://cran.rstudio.com/web/packages/jsonlite>

```
library(jsonlite)  
  
fromJSON('{"foo": "bar"}')  
  
#> $foo  
  
#> [1] "bar"
```

```
fromJSON('{"foo": "bar"}', FALSE)  
  
#> $foo  
  
#> [1] "bar"
```

```
fromJSON('[{"foo": "bar", "hello": "world"}]')  
  
#>  foo hello  
  
#> 1 bar world
```

# JSON

- **Javascript Object Notation**
  - Widely used in web APIs
  - Becoming de facto standard for data format for web APIs
  - less expressive than XML
  - but easier for humans to grok
  - jsonlite - the go to JSON pkg for R, to create and parse JSON

# XML

<http://www.omdbapi.com/?t=frozen&y=&plot=short&r=xml>

```
<root response="True">

    <movie title="Frozen" year="2013" rated="PG" released="27 Nov 2013"
    runtime="102 min" genre="Animation, Adventure, Comedy"
    director="Chris Buck, Jennifer Lee" writer="Jennifer Lee (screenplay),
    Hans Christian Andersen (story inspired by "The Snow Queen")
    by), Chris Buck (story by), Jennifer Lee (story by), Shane Morris (story by)"
    actors="Kristen Bell, Idina Menzel, Jonathan Groff, Josh Gad"
    plot="When the newly crowned Queen Elsa accidentally uses her power
    to turn things into ice to curse her home in infinite winter, her sister, Anna,
    teams up with a mountain man, his playful reindeer, and a snowman to
    change the weather condition." language="English, Icelandic"
    country="USA" awards="Won 2 Oscars. Another 70 wins & 56
    nominations." poster="http://ia.media-imdb.com/images/M/
    MV5BMTQ1MjQwMTE5OF5BMl5BanBnXkFtZTgwNjk3MTcyMDE@._V1_S
    X300.jpg" metascore="74" imdbRating="7.6" imdbVotes="410,734"
    imdbID="tt2294629" type="movie"/>

</root>
```

# XML

- Extensible Markup Language
  - Used to dominate in web APIs, no less common
  - Very expressive
  - hard for humans to grok
  - xml2 - the go to XML pkg for R, to create and parse XML

# xml2

<https://cran.rstudio.com/web/packages/xml2>

```
library(xml2)
```

```
res <- read_xml('<foo>bar</foo>')
xml_name(res)
#> [1] "foo"
```

```
xml_text(res)
#> [1] "bar"
```

# Your Turn

Get data for 3 movies in both JSON and XML format.

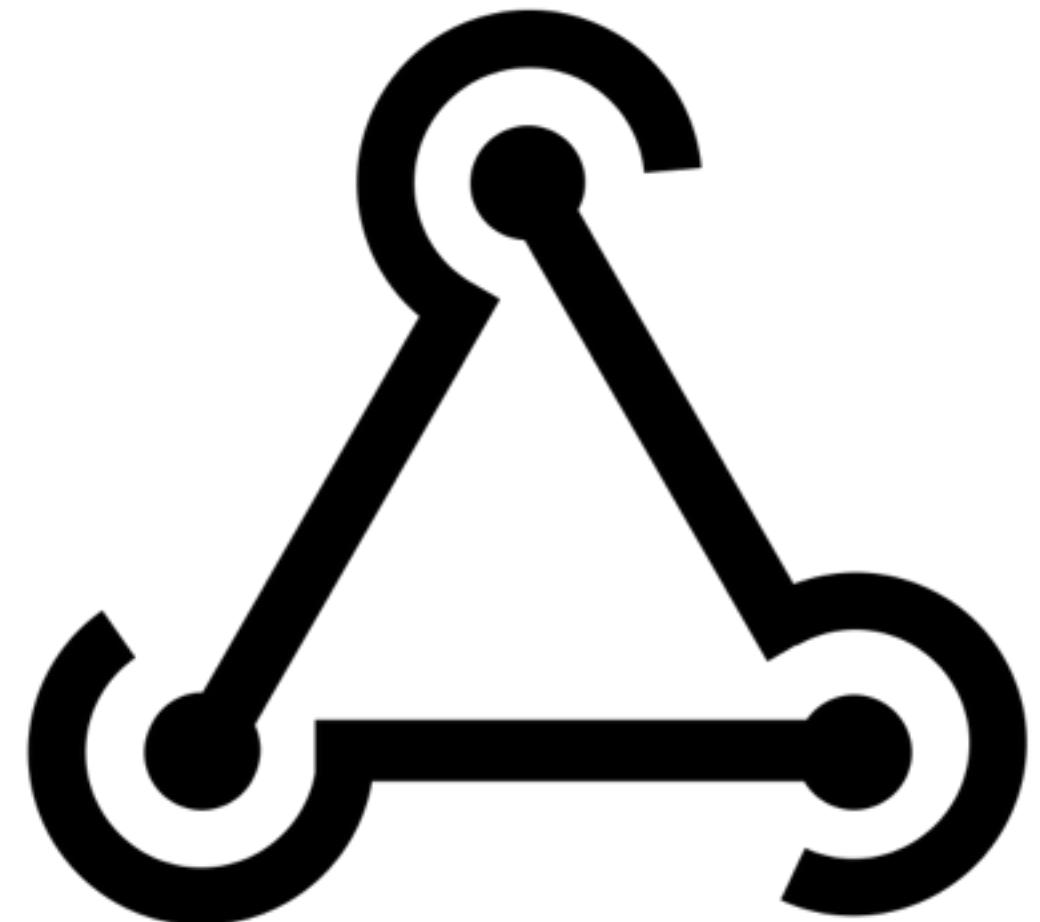
Parse each format to plain text and their parsed versions.



# API Resources

- httr introduction
- An Introduction to APIs from zapier.com
- HTTP status codes @wikipedia
- Oauth @wikipedia

# Part II: Getting data from the web



Karthik Ram

**HELLO**

my name is

**Karthik**

**@\_inundat**

1. Making **GET** requests
2. Extracting web content
3. Passing additional query parameters and API keys

# APIs and end points



# The Internet Movie database

Producers

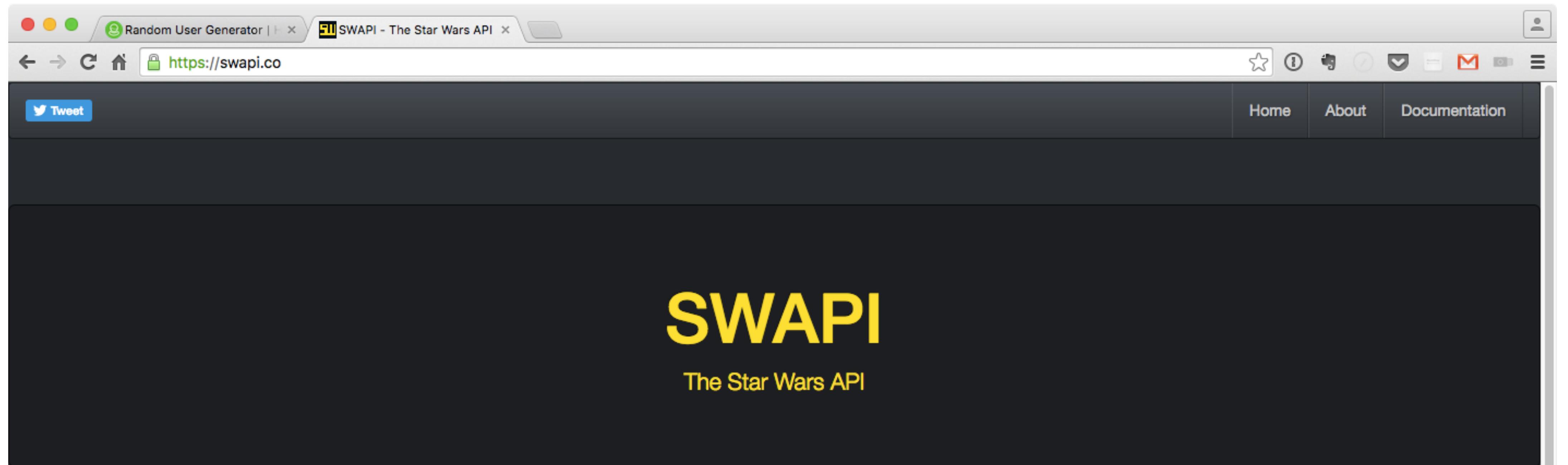
Movies

Actors



**End points**

# Writing a GET request



Try it now!

<http://swapi.co/api/>

[people/1/](#)

request

Need a hint? try [people/1/](#) or [planets/3/](#) or [starships/9/](#)

# Step 1: Write a web request

```
web_call <- GET('http://swapi.co/api/planets/1/')

web_call
```

```
> web_call
```

```
Response [http://swapi.co/api/planets/1/]
```

```
Date: 2016-06-22 20:35
```

```
Status: 200
```

```
Content-Type: application/json
```

```
Size: 805 B
```

# Now your turn

Run a **GET** request against these end points

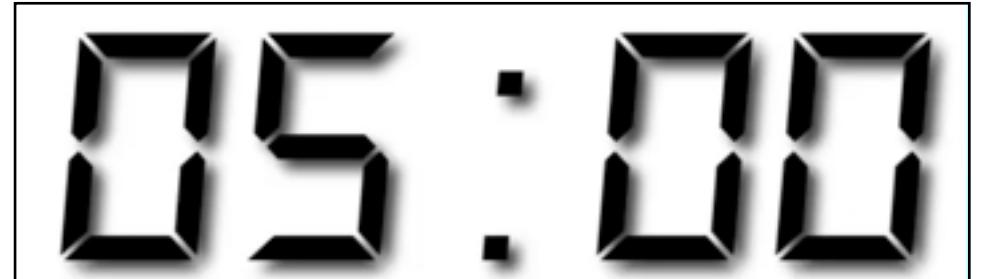
api.randomuser.me

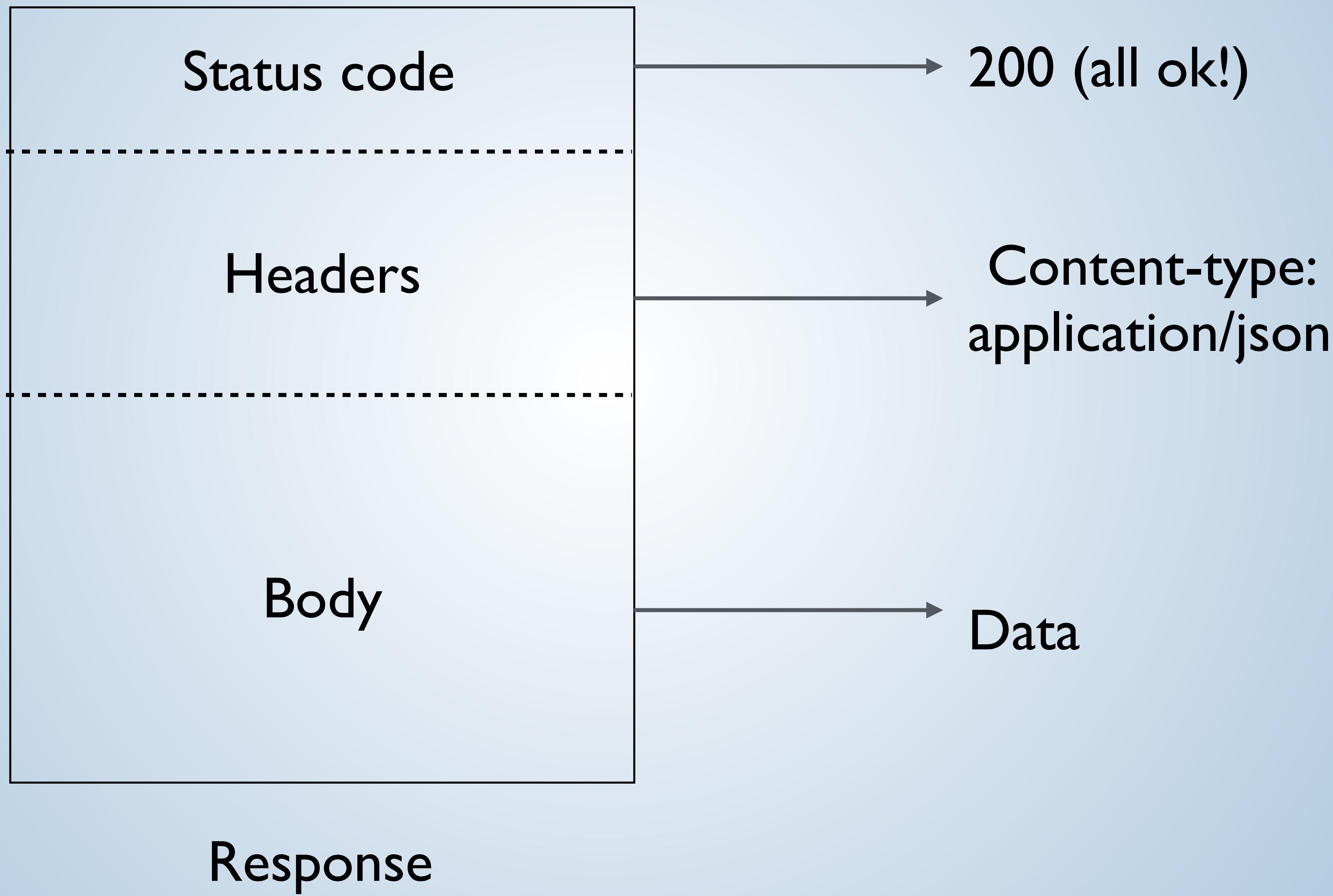
api.openweathermap.org/data/2.5/forecast?id=524901

What does your result object contain?

What are the response codes?

What do they mean?





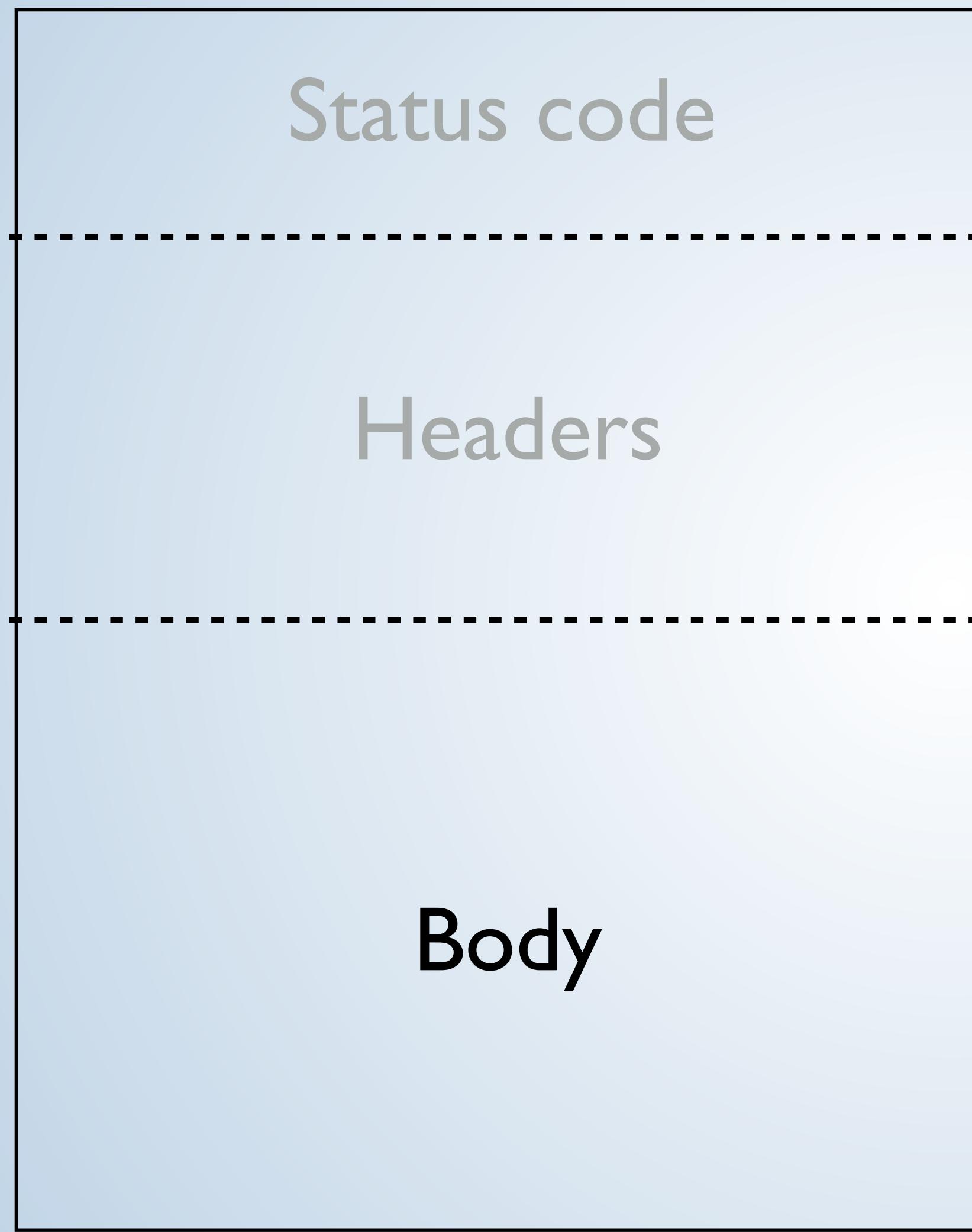
# 2: Extracting content

# Step 2: Extract content from a request

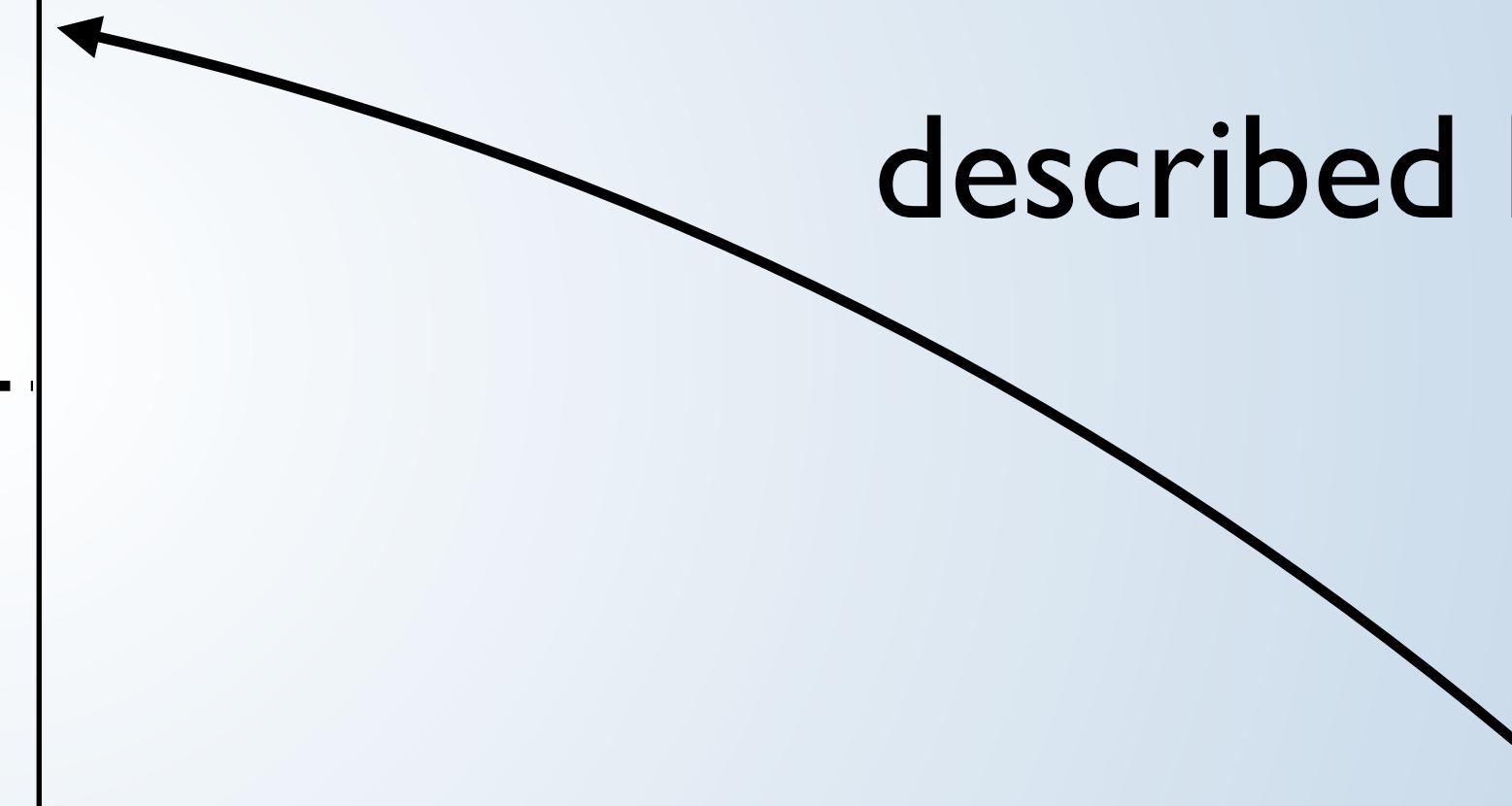
**text** - *a character vector*

**raw** - *as a raw object*

**parsed** - *parsed into a R object*



**described here**



**text, raw, parsed**



```
> call <- GET("http://google.com")
> result <- content(call, as = 'text')
> result
[1] "<!doctype html><html itemscope=\"\" itemtype=\"http://
schema.org/WebPage\" lang=\"en\"><head><meta content=\"Search the
world's information, . <truncated>
> class(result)
[1] "character"
```

insert example

# api.randomuser.me

Random User Generator | X

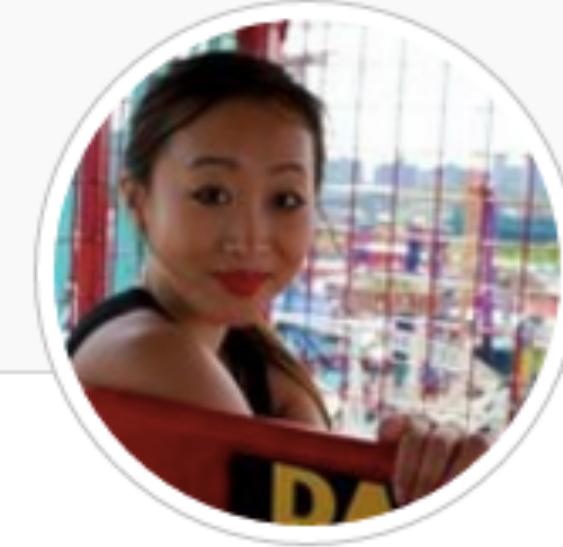
https://randomuser.me

Home User Photos Documentation Change Log Stats & Graphs Donate Copyright Notice Photoshop Extension

## RANDOM USER GENERATOR

A free, [open-source API](#) for generating random user data. Like Lorem Ipsum, but for people.

[Follow us @randomapi](#)



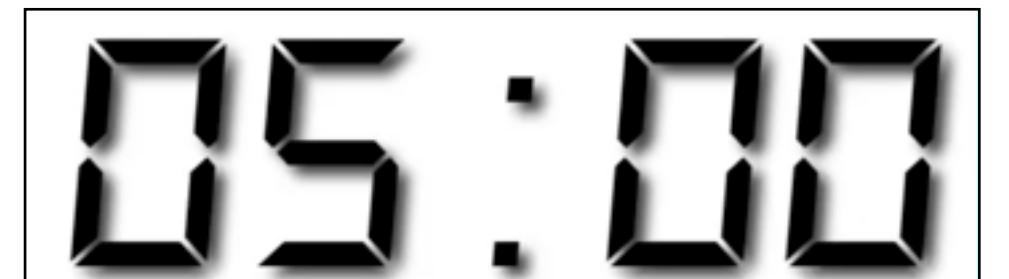
Hi, My name is  
Dawn Beck

# Exercise

Process the content from the random user API

Save the result into an object called **person**

Extract the content as text and parsed.



# Passing arguments

## Requesting Multiple Users

Random User Generator allows you to fetch up to 5,000 generated users in one request using the **results** parameter.

`http://api.randomuser.me/?results=5000`



**results:** Retrieve multiple names

## Specifying a gender

You can specify whether you would like to have only male or only female users generated by adding the **gender** parameter to your request. Valid values for the gender parameter are "male" or "female", or you may leave the parameter blank. Any other value will cause the service to return both male and female users.

`http://api.randomuser.me/?gender=female`



**gender:** specify a gender

## Seeds

Seeds allow you to always generate the same set of users. For example, the seed "foobar" will always return results for [Becky Sims](#) (for version 1.0). Seeds can be any string or sequence of characters.

`http://api.randomuser.me/?seed=foobar`



**seed:** Set a seed

## Formats

We currently offer the following data formats:

- JSON (default)

```
num_results <- 5
args <- list(results = num_results)
random_names <- GET("http://api.randomuser.me/",
query = args)
output <- content(random_names, as = 'parsed')
> length(output$results)
[1] 5
```

# Exercise

Look over the api documentation and see what other parameters can be passed

Modify the example to pass a gender to the request

Can you do this with both number of results and gender?





Personal Open source Business Explore

Pricing Blog Support

Search GitHub

## Personal API tokens

May 16, 2013



tclem

New Features

You can now [create your own personal API tokens](#) for use in scripts and on the command line.

Be careful, these tokens are like passwords so you should guard them carefully. The advantage to using a token over putting your password into a script is that a token can be revoked, and you can generate lots of them. [Head on over to your settings](#) to manage personal API tokens.

**Personal API Access Tokens**

These tokens are like passwords; guard them carefully.

4a68631afb82ba1a9f9c49892e0e3c82eaa7ef66

**Create new token**

**Delete**

Write a complete  
request to a  
proper API

# Open Weather example

The screenshot shows a web browser window with the title bar "Weather API". The address bar contains "openweathermap.org/api". The main content area displays the OpenWeatherMap website. At the top, there's a navigation bar with links for "Support Center", "Weather in your city" (with a search icon), "Sign In", "Sign Up", and temperature units ("°C" and "°F"). Below this is another navigation bar with links for "OpenWeatherMap", "Weather", "Maps", "API", "Price", "Partners", "Stations", "News", and "About". The main content area has a heading "Weather API" and a breadcrumb trail "Home / Weather API". A central text block states: "Our weather API is simple, clear and free. We also offer higher levels of support, please see our [paid plan options](#). To access the API you need to sign up for an [API key](#) if you are on a free or paid plan." Three main service sections are highlighted with blue borders: "Current weather data", "5 day / 3 hour forecast", and "16 day / daily forecast". Each section includes "API doc" and "Subscribe" buttons and a bulleted list of features.

## Current weather data

[API doc](#) [Subscribe](#)

- Access current weather data for any location including over 200,000 cities
- Current weather is frequently updated based on global models and data from more than 40,000 weather stations
- Data is available in JSON, XML, or HTML format
- Available for Free and all other paid accounts

## 5 day / 3 hour forecast

[API doc](#) [Subscribe](#)

- 5 day forecast is available at any location or city
- 5 day forecast includes weather data every 3 hours
- Forecast is available in JSON, XML, or HTML format
- Available for Free and all other paid accounts

## 16 day / daily forecast

[API doc](#) [Subscribe](#)

- 16 day forecast is available at any location or city
- 16 day forecasts includes daily weather
- Forecast is available in JSON, XML, or HTML format
- Available for Developer, Professional and Enterprise accounts

## Historical data

[API doc](#) [Subscribe](#)

- Through our API we provide city historical

## UV Index

[API doc](#) [Subscribe](#)

- Current UV index (Clear Sky) and historical

## Weather map layers

[API doc](#) [Subscribe](#)

- Weather maps include precipitation

# Exercise

- Get a API key from [openweathermap.org/api](http://openweathermap.org/api)
- Write a request for current weather data (<http://openweathermap.org/current>) for zip code 94708,us
- Pass
- Format results into a data.frame

Tip: Pass the API key as:

```
APPID <- '<your_api_key>'
```



Thanks!

# Web Scraping

**HELLO**

my name is

**Garrett**



@StatGarrett

# What if data is on a web page but there is no API?

[www.imdb.com/title/tt2294629/](http://www.imdb.com/title/tt2294629/)

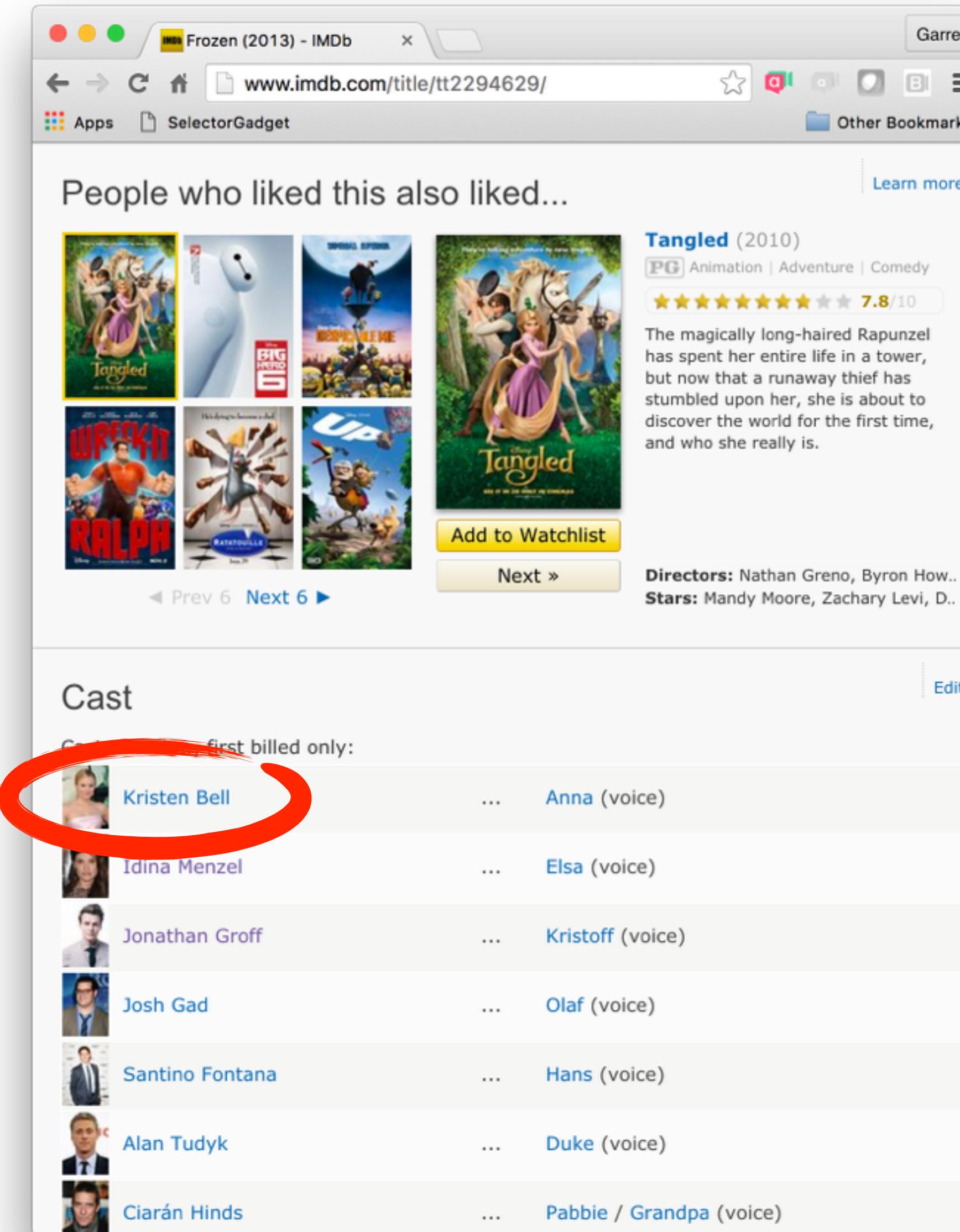
The screenshot shows a web browser window displaying the IMDb movie page for "Frozen (2013)". The page includes the movie's title, rating (7.6/10), and a thumbnail image. An advertisement for the "ALL-NEW PRIUS" is overlaid on the page, featuring a red Prius driving across a bridge with helicopters in the background. The ad text reads: "EXHILARATING DRIVING DYNAMICS. TIME FOR HYBRIDS TO HAVE FUN." A "LEARN MORE" button and a note stating "Prototype shown with options." are also visible.

IMDb Picks: June

# Outline

1. How to scrape a web page
2. rvest
3. selectorGadget
4. Practice with tables

# Strategy



People who liked this also liked...

**Tangled** (2010) PG Animation | Adventure | Comedy 7.8/10

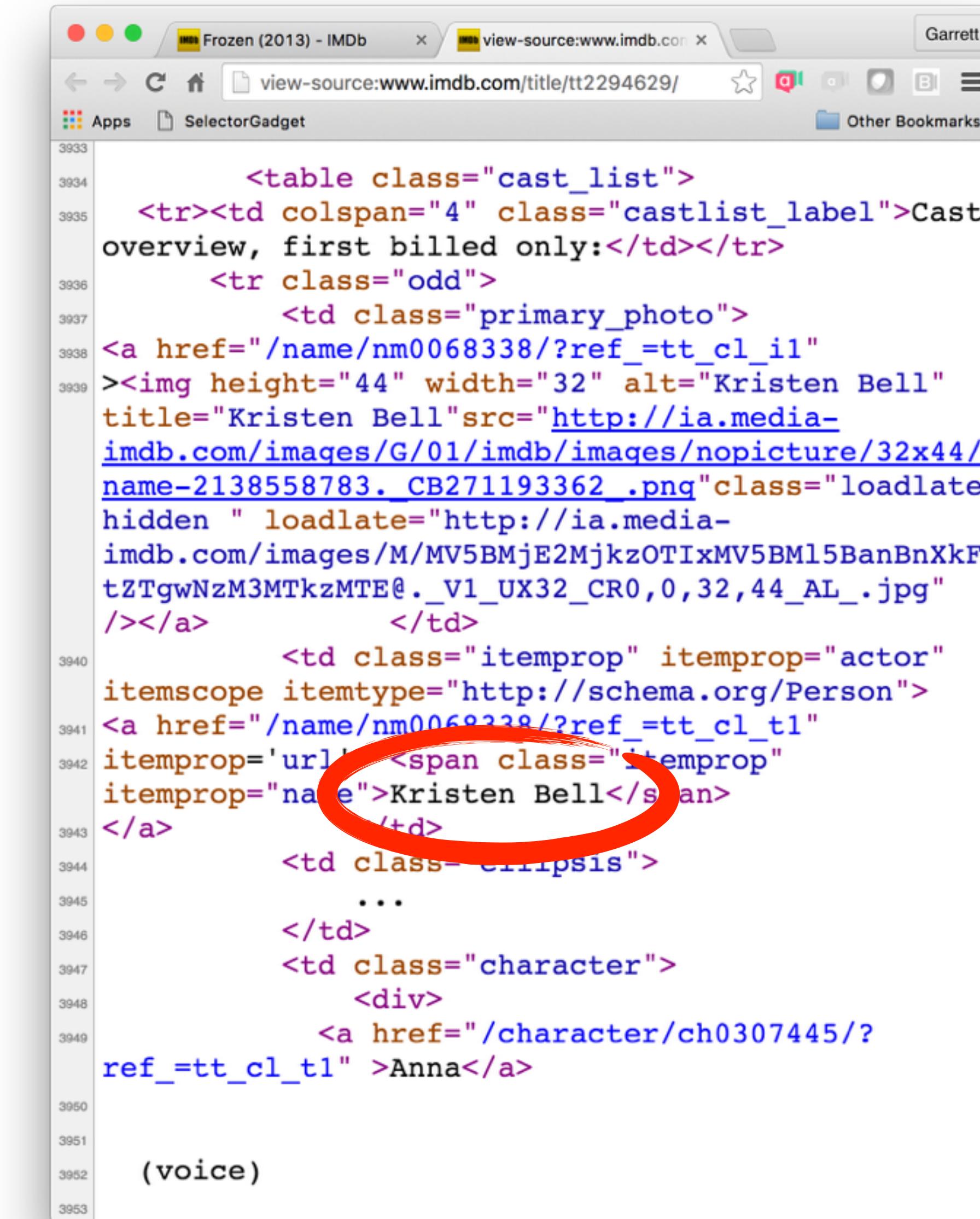
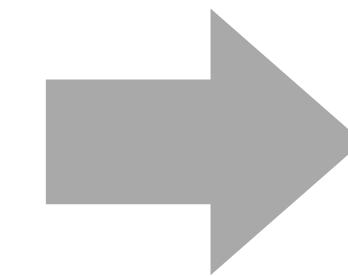
The magically long-haired Rapunzel has spent her entire life in a tower, but now that a runaway thief has stumbled upon her, she is about to discover the world for the first time, and who she really is.

Add to Watchlist Next »

◀ Prev 6 ▶

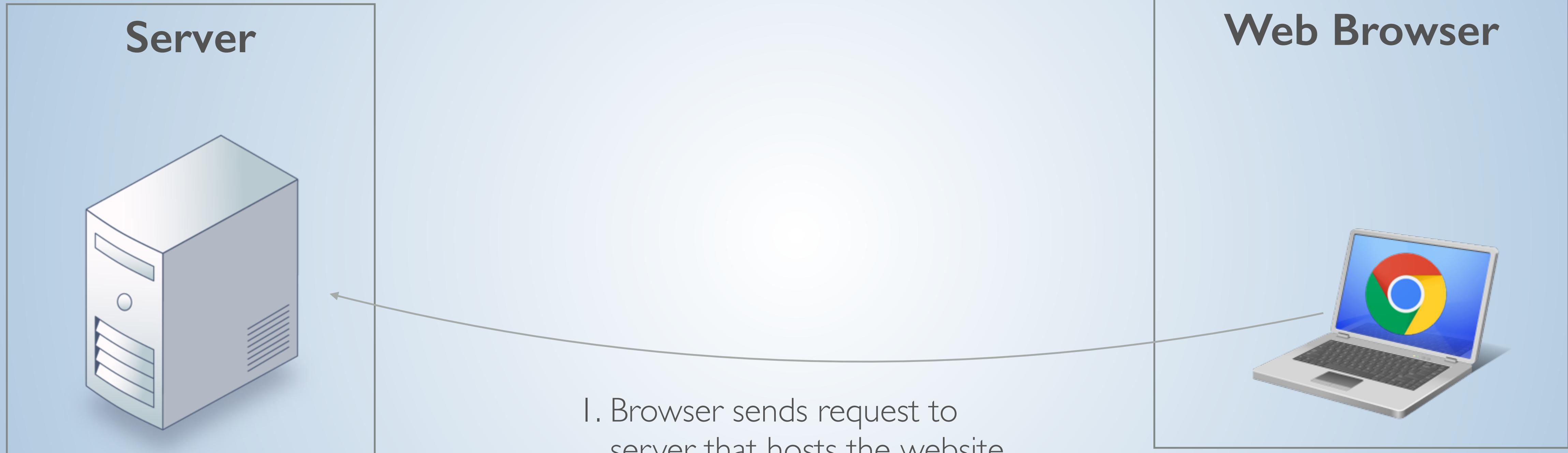
### Cast

	Cast	Character
	Kristen Bell	Anna (voice)
	Idina Menzel	Elsa (voice)
	Jonathan Groff	Kristoff (voice)
	Josh Gad	Olaf (voice)
	Santino Fontana	Hans (voice)
	Alan Tudyk	Duke (voice)
	Ciarán Hinds	Pabbie / Grandpa (voice)



```
3933 <table class="cast_list">
3934 <tr><td colspan="4" class="castlist_label">Cast
3935 overview, first billed only:</td></tr>
3936 <tr class="odd">
3937 <td class="primary_photo">
3938 <a href="/name/nm0068338/?ref_=tt_cl_i1"
3939 ></a> </td>
3940 <td class="itemprop" itemprop="actor"
3941 itemscope itemtype="http://schema.org/Person">
3942 <a href="/name/nm0068338/?ref_=tt_cl_t1"
3943 itemprop='url'><span class="itemprop"
3944 itemprop="name">Kristen Bell</span>
3945 </a> </td>
3946 <td class="ellipsis">
3947 ...
3948 </td>
3949 <td class="character">
3950 <div>
3951 <a href="/character/ch0307445/?ref_=tt_cl_t1" >Anna</a>
3952
3953 (voice)
```

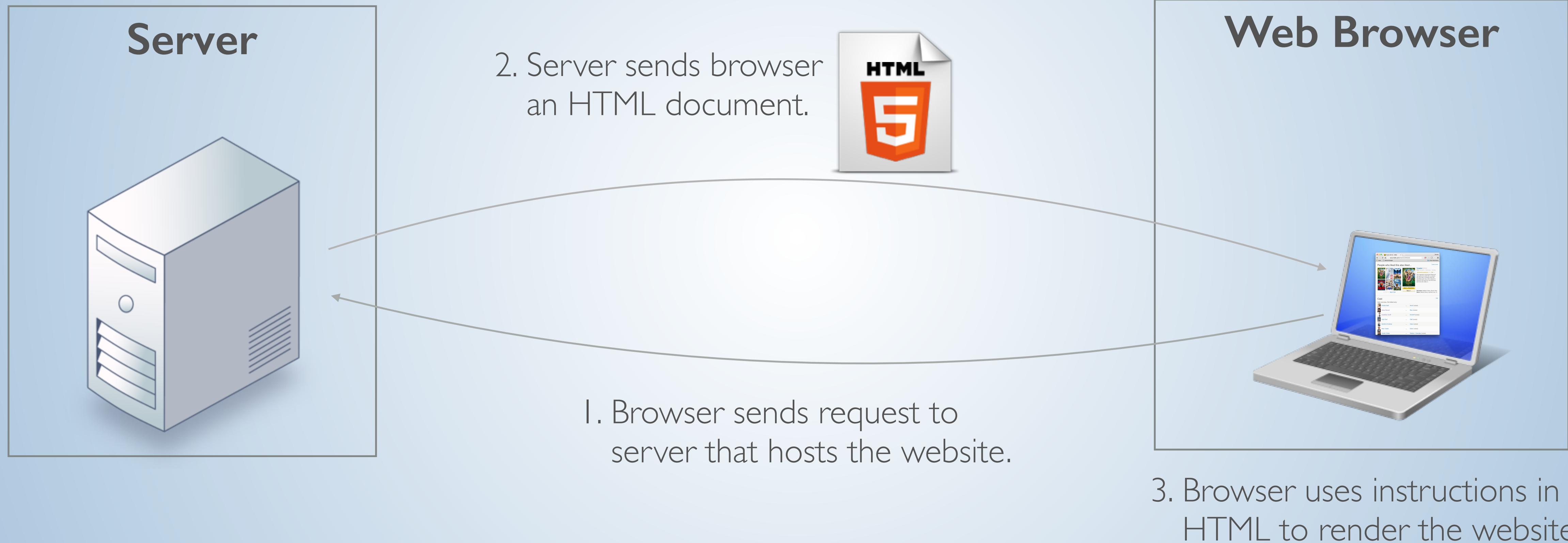
# *HTML (Review)*



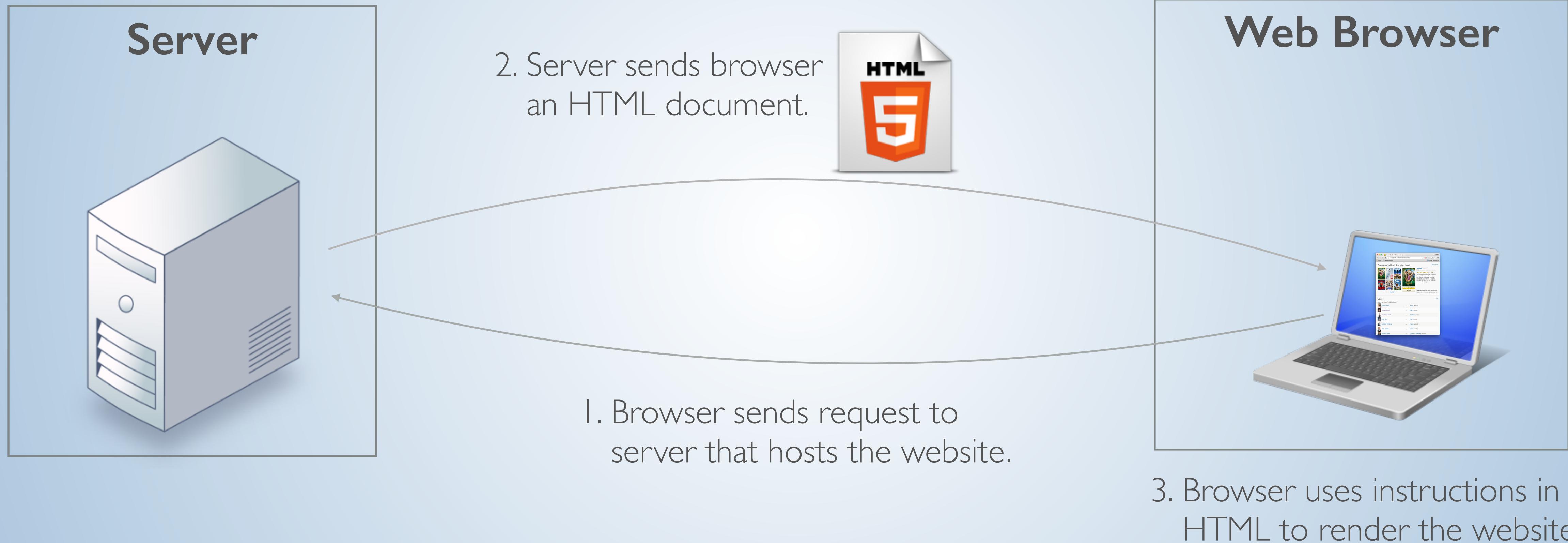
# *HTML (Review)*



# *HTML (Review)*



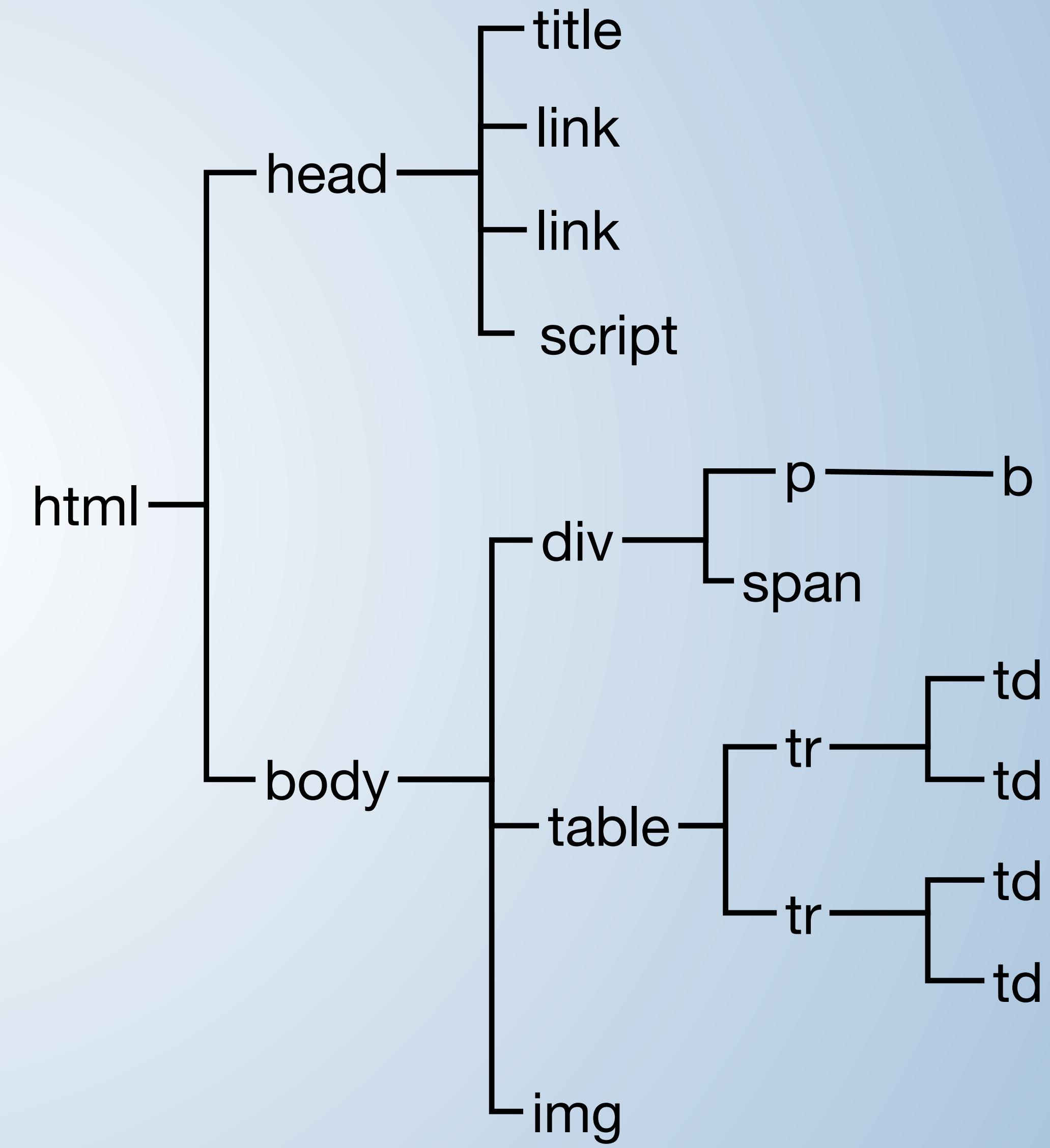
# *HTML (Review)*



# HTML (Review)



```
<html>
  <head>
    <title>Title</title>
    <link rel="icon" type="icon" href="http://a" />
    <link rel="icon" type="icon" href="http://b" />
    <script type="text/javascript">
      var ue_t0=window.ue_t0||+new Date();
    </script>
  </head>
  <body>
    <div>
      <p>Click <b>here</b> now.</p>
      <span>Frozen</span>
    </div>
    <table style="width:100%">
      <tr>
        <td>Kristen</td>
        <td>Bell</td>
      </tr>
      <tr>
        <td>Idina</td>
        <td>Menzel</td>
      </tr>
    </table>
    
  </body>
</html>
```



# HTML (Review)

Each element in the page is created by a tag.

```
<a href="http://github.com">GitHub</a>
```

tag name

attribute  
(name)

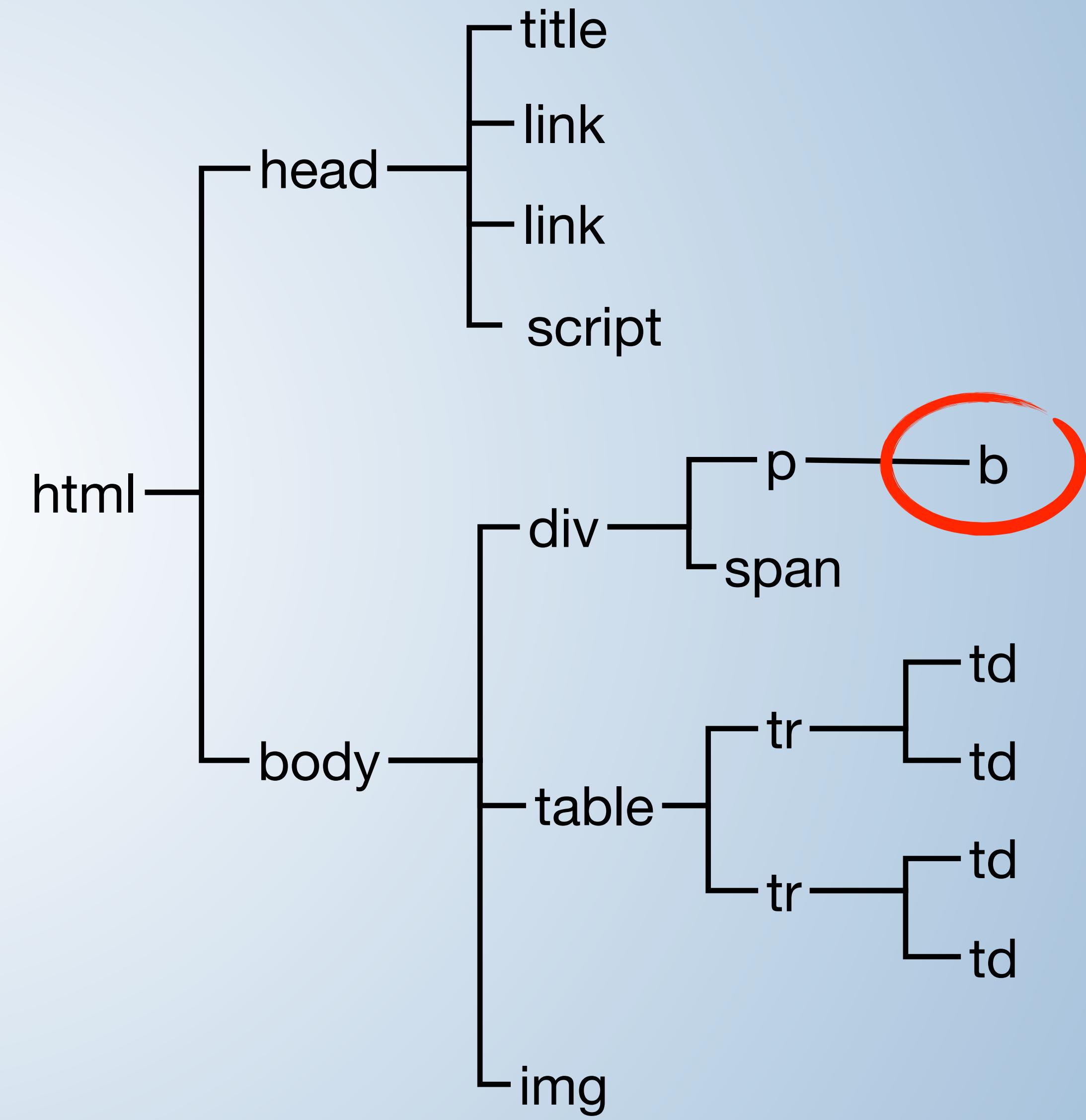
attribute  
(value)

content

# HTML (Review)



```
<html>
  <head>
    <title>Title</title>
    <link rel="icon" type="icon" href="http://a" />
    <link rel="icon" type="icon" href="http://b" />
    <script type="text/javascript">
      var ue_t0=window.ue_t0||+new Date();
    </script>
  </head>
  <body>
    <div>
      <p>Click <b>here</b> now.</p>
      <span>Frozen</span>
    </div>
    <table style="width:100%">
      <tr>
        <td>Kristen</td>
        <td>Bell</td>
      </tr>
      <tr>
        <td>Idina</td>
        <td>Menzel</td>
      </tr>
    </table>
    
  </body>
</html>
```



# Your Turn

Find the source code for the Frozen IMDB page



**Chrome**

Goto:  
1. View  
    a. Developer  
        i. View Source

03 : 00



**Explorer**

1. Right click  
    on page  
    a. Click View  
        Source



**Firefox**

Goto:  
1. Firefox  
    a. Web Developer  
        i. Page Source



**Safari**

Goto:  
1. Safari  
    a. Preferences  
        i. Advanced  
            a) Check "Show  
                Develop menu  
                in menu bar"  
2. Develop  
    a. Show Page  
        Source

# Your Turn



Navigate to the IMDB page for Frozen and open the source code.

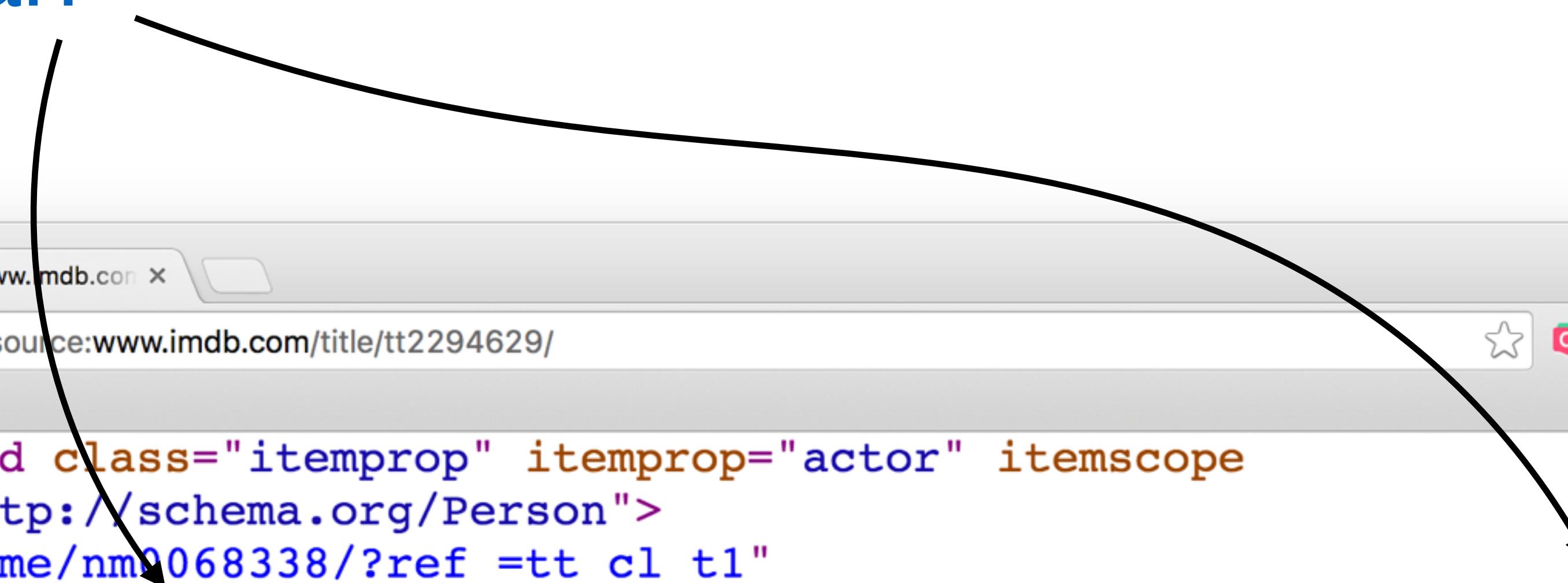
Locate the piece of HTML that inserts Kristen Bell into the Cast section.

Which HTML tag surrounds her name?

01:00

# Which HTML tag surrounds her name?

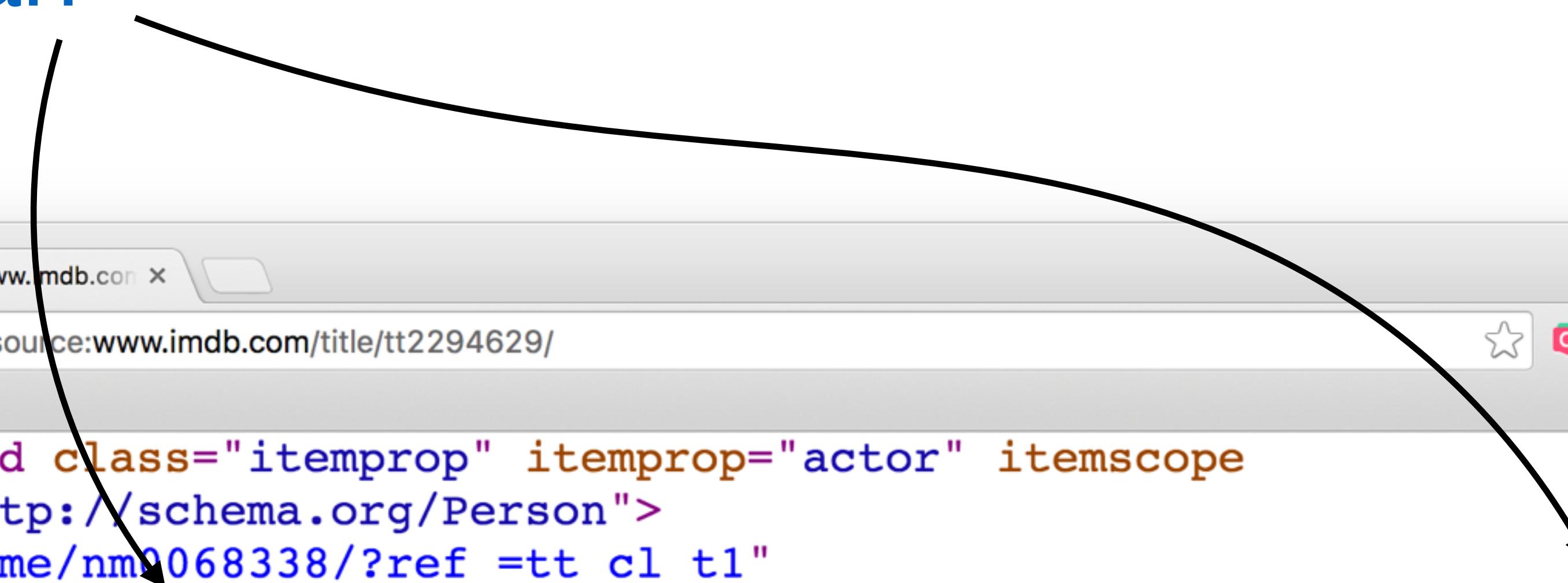
span



```
view-source:www.imdb.com x Garrett
view-source:www.imdb.com/title/tt2294629/
Apps SelectorGadget Other Bookmarks
3941 <td class="itemprop" itemprop="actor" itemscope
itemtype="http://schema.org/Person">
3942 <a href="/name/nm0068338/?ref_=tt_cl_t1"
3943 itemprop='url'> <span class="itemprop" itemprop="name">Kristen Bell</span>
3944 </a>
3945 <td class="ellipsis">
3946 ...
3947 </td>
3948 <td class="character">
3949 <div>
3950 <a href="/character/ch0307445/?ref_=tt_cl_t1" >Anna</a>
3951
3952
3953 (voice)
3954
```

# Which HTML tag surrounds her name?

span



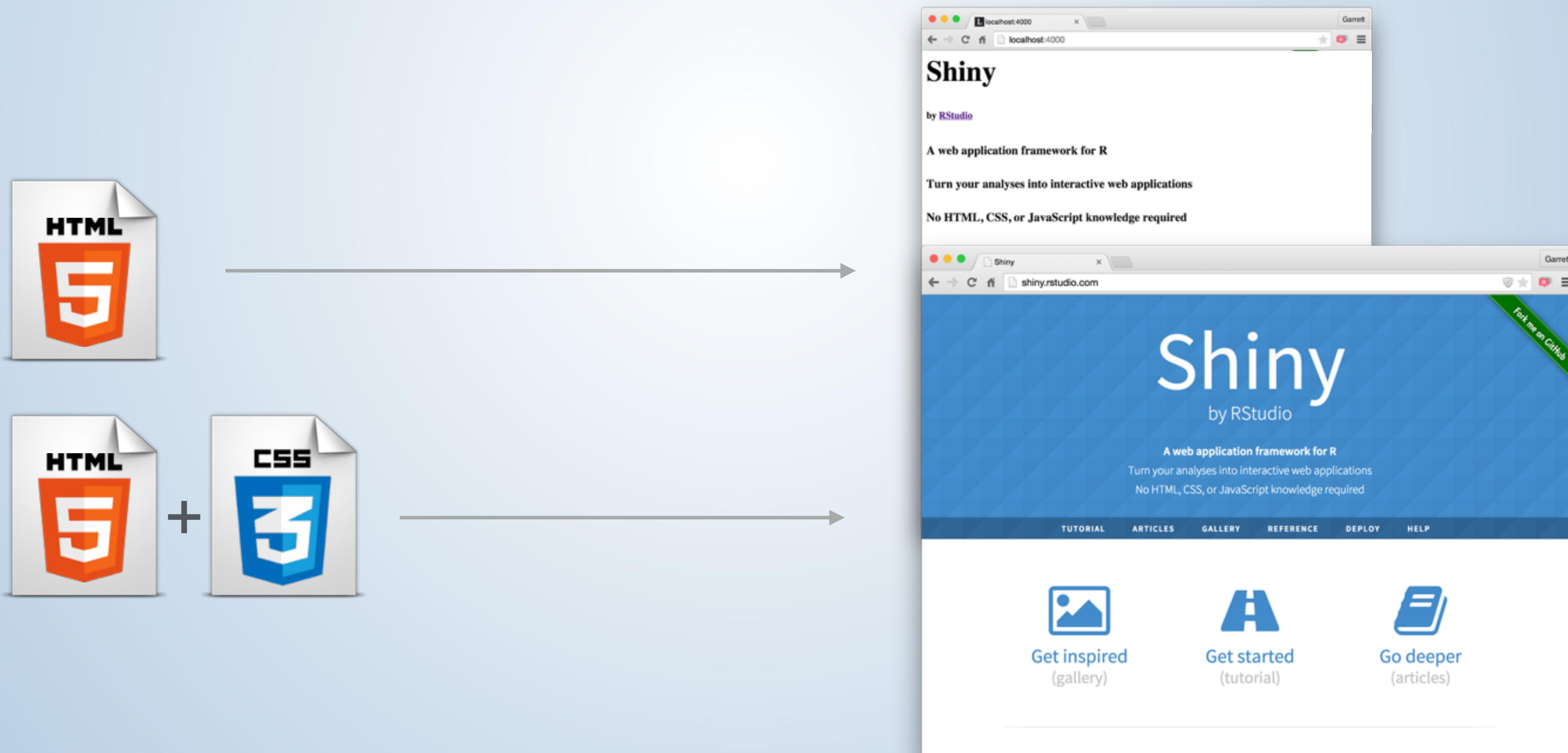
```
view-source:www.imdb.com x Garrett
view-source:www.imdb.com/title/tt2294629/
Apps SelectorGadget Other Bookmarks
3941 <td class="itemprop" itemprop="actor" itemscope
itemtype="http://schema.org/Person">
3942 <a href="/name/nm0068338/?ref_=tt_cl_t1"
3943 itemprop='url'> <span class="itemprop" itemprop="name">Kristen Bell</span>
3944 </a> </td>
3945 <td class="ellipsis">
3946 ...
3947 </td>
3948 <td class="character">
3949 <div>
3950 <a href="/character/ch0307445/?ref_=tt_cl_t1" >Anna</a>
3951
3952
3953 (voice)
3954
```

But how many `<span>` tags are there?

# css selectors

# CSS (Review)

Cascading Style Sheets (CSS) are a framework for customizing the appearance of elements in a web page.



# CSS (Review)



Shiny

by RStudio

A web application framework for R

Turn your analyses into interactive web applications

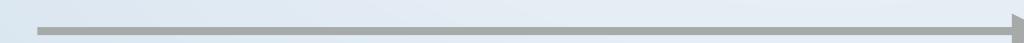
No HTML, CSS, or JavaScript knowledge required

- [Tutorial](#)
- [Articles](#)
- [Gallery](#)
- [Reference](#)
- [Deploy](#)
- [Help](#)

[Get inspired](#)  
(gallery)

[Get started](#)  
(tutorial)

[Go deeper](#)  
(articles)



Shiny

by RStudio

A web application framework for R

Turn your analyses into interactive web applications

No HTML, CSS, or JavaScript knowledge required

TUTORIAL ARTICLES GALLERY REFERENCE DEPLOY HELP

Get inspired  
(gallery)

Get started  
(tutorial)

Go deeper  
(articles)

# CSS (Review)



```
span {  
    color: #ffffff;  
}  
  
.num {  
    color: #a8660d;  
}  
  
table.data {  
    width: auto;  
}  
  
#firstname {  
    background-color: yellow;  
}
```

← selector

← styling

← selector

← styling

← selector

← styling

← selector

← styling

# CSS (Review)

A CSS script describes an element by its tag, class, and/or ID.

```
<span class="bigname" id="shiny">Shiny</span>
```

tag name

class  
(optional)

id  
(optional)

# CSS (Review)

A CSS script describes an element by its tag, class, and/or ID.

```
<span class="bigname" id="shiny">Shiny</span>
```

```
span
```

CSS selector for **ALL** elements with:

- the **span tag**

# CSS (Review)

A CSS script describes an element by its tag, class, and/or ID.

```
<span class="bigname" id="shiny">Shiny</span>
```

```
.bigname
```

CSS selector for **ALL** elements with:

- the **bigname class**

# CSS (Review)

A CSS script describes an element by its tag, class, and/or ID.

```
<span class="bigname" id="shiny">Shiny</span>
```

```
span.bigname
```

CSS selector for **ALL** elements with:

- the **span tag**

**AND**

- the **bigname class**

# CSS (Review)

A CSS script describes an element by its tag, class, and/or ID.

```
<span class="bigname" id="shiny">Shiny</span>
```

```
#shiny
```

CSS selector for **ALL** elements with:

- the **shiny id**

# CSS (*Review*)

<b>Prefix</b>	<b>Matches</b>
none	tag
.	class
#	id

# Your Turn

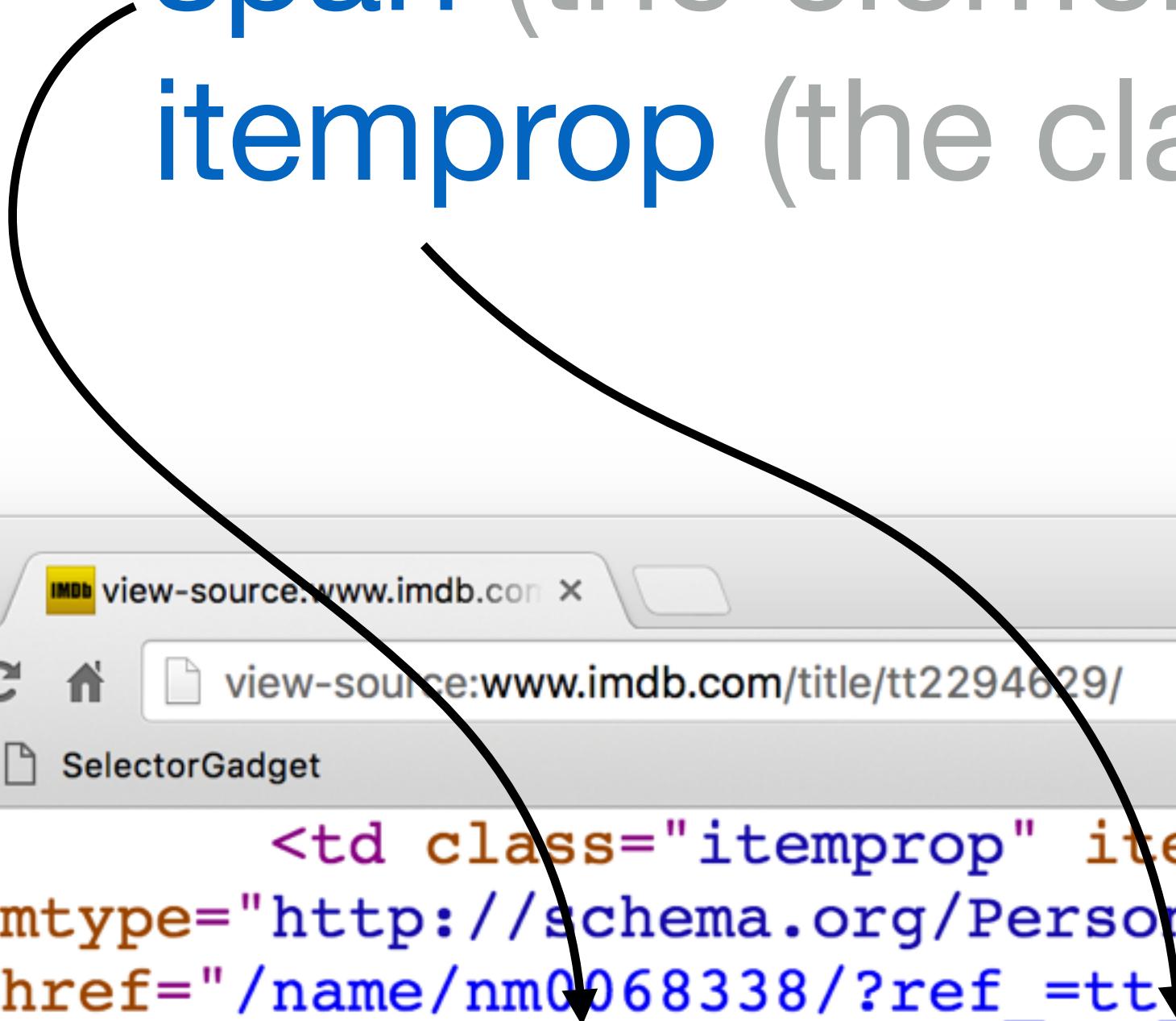
Which CSS identifiers are associated with Kristen Bell's name in the Frozen page? Write a CSS selector that targets them.



# Which CSS identifiers are associated with Kristen Bell's name?

span (the element)

itemprop (the class)



```
view-source:www.imdb.com
view-source:www.imdb.com/title/tt2294629/
Apps SelectorGadget Other Bookmarks
3941 <td class="itemprop" itemprop="actor" itemscope
itemtype="http://schema.org/Person">
3942 <a href="/name/nm0068338/?ref_=tt_cl_t1"
3943 itemprop='url'> <span class="itemprop" itemprop="name">Kristen Bell</span>
3944 </a>
3945 <td class="ellipsis">
3946 ...
3947 </td>
3948 <td class="character">
3949 <div>
```

# Which CSS identifiers are associated with Kristen Bell's name?

span (the element)

itemprop (the class)

```
view-source:www.imdb.com
view-source:www.imdb.com/title/tt2294629/
SelectorGadget
3941      <td class="itemprop" itemprop="actor" itemscope
3942      itemtype="http://schema.org/Person">
3943      <a href="/name/nm0068338/?ref_=tt_cl_t1"
3944          itemprop='url'> <span class="itemprop" itemprop="name">Kristen Bell</span>
3945          </a>
3946          <td class="ellipsis">
3947              ...
3948          </td>
3949          <td class="character">
3950              <div>
```

Span.itemprop

# Recap

Extract information from the HTML document.  
Identify information to extract with CSS selectors.

**rvest**

# rvest



A package that makes it easy to extract  
info from a webpage.

```
install.packages("rvest")
```

\* This will also install *xml2*, a package that *rvest* relies on.

# Basic Workflow

1. Download the HTML and turn it into an XML file with `read_html()`

```
library(rvest)  
frozen <- read_html("http://www.imdb.com/title/tt2294629/")
```

read\_html

URL

\* `read_html()` comes in the `xml2` package

## To examine contents

frozen

html\_structure(frozen)

as\_list(frozen)

xml\_children(frozen)

xml\_children(frozen)[[2]]

xml\_contents(xml\_children(frozen)[[2]])

# Basic Workflow

1. Download the HTML and turn it into an XML file with `read_html()`

2. Extract specific nodes with `html_nodes()`

```
itals <- html_nodes(frozen, "em")
```

XML

CSS selector

# Basic Workflow

1. Download the HTML and turn it into an XML file with `read_html()`
2. Extract specific nodes with `html_nodes()`
3. Extract content from nodes with `html_text()`,  
`html_name()`, `html_attrs()`, `html_children()`,  
`html_table()`

```
itals
```

```
## {xml_nodeset (1)}  
## [1] <em class="nobr">Written by<br><a href="/search/title?plot_author=DeAlan%2 ...
```

```
html_text(itals)
```

```
## [1] "Written by<br>DeAlan Wilson for ComedyE.com"
```

```
html_name(itals)
```

```
## [1] "em"
```

```
html_children(itals)
```

```
## {xml_nodeset (1)}  
## [1] <a href="/search/title?plot_author=DeAlan%20Wilson%20for%20ComedyE.com&a ...
```

```
html_attr(itals, "class")
```

```
## [1] "nobr"
```

```
html_attrs(itals)
```

```
## [[1]]
```

```
##   class
```

```
##   "nobr"
```

# Recap

- 1.** Download the HTML and turn it into an XML file with `read_html()`
- 2.** Extract specific nodes with `html_nodes()`
- 3.** Extract content from nodes with `html_text()`,  
`html_name()`, `html_attrs()`, `html_children()`,  
`html_table()`

# Your Turn

1. Read in the Frozen HTML.
2. Select the nodes that are both **spans** and class = " **itemprop**".
3. Extract the text from the nodes.

Do you collect the cast names and only the cast names?



```
library(rvest)
frozen <- read_html("http://www.imdb.com/title/tt2294629/")
cast <- html_nodes(frozen, "span.itemprop")
html_text(cast)

## [1] "Animation"                      "Adventure"
## [3] "Comedy"                          "Chris Buck"
## [5] "Jennifer Lee"                   "Jennifer Lee"
## [7] "Hans Christian Andersen"       "Kristen Bell"
## [9] "Idina Menzel"                   "Jonathan Groff"
## [11] "Kristen Bell"                   "Idina Menzel"
## [13] "Jonathan Groff"                "Josh Gad"
## [15] "Santino Fontana"                "Alan Tudyk"
## [17] "Ciarán Hinds"                  "Chris Williams"
## [19] "Stephen J. Anderson"           "Maia Wilson"
## [21] "Edie McClurg"                  "Robert Pine"
## [23] "Maurice LaMarche"              "Livvy Stubenrauch"
## [25] "Eva Bella"                     "snowman"
## [27] "sister love"                   "sister sister relationship"
## [29] "magic"                          "snow"
## [31] "Walt Disney Animation Studios" "Walt Disney Pictures"
```

But we've scraped  
too much info

**selectorGadget**

# selectorGadget

A GUI tool to identify CSS selector combinations



# To Install

1. Run `vignette("selectorgadget")`
2. Drag `Selectorgadget` link into your browser's bookmark bar

The image shows a composite of two screenshots. On the left, the RStudio interface displays an R script with the following code:

```
1 # Read in web page
2
3 library(rvest)
4 frozen <- read_html("http://www.imdb.com/title/tt2294629/")
5
6 # Look at web page
7
8 frozen
9 html_structure(frozen)
10 as_list(frozen)
11
12 xml_children(frozen)
13 xml_children(frozen)[[2]]
```

In the RStudio console below, the command `> vignette("selectorgadget")` is run, and the output shows a link to "Selectorgadget". This link is circled in red. A large red arrow originates from this circled link and points towards the top right corner of the image, where a screenshot of a web browser is shown.

The web browser screenshot shows the IMDb page for the movie "Frozen (2013)". The "SelectorGadget" bookmark is visible in the bookmark bar. The main content of the page includes a section titled "People who liked this also liked..." with movie posters for "Despicable Me", "Big Hero 6", "Up", and "Tangled". To the right of the poster for "Tangled" is a yellow "Add to Watchlist" button. Below the posters are navigation buttons "Prev 6" and "Next 6". To the far right, there is a summary of the movie "Tangled" with its director, stars, and a rating of 7.8/10.

# To Use

1. **Navigate** to a webpage
2. **Open** the SelectorGadget bookmark
3. **Click** on item to scrape
4. **Click** on yellow items You do not want to scrape
5. **Click** on additional items that you do want to scrape
6. **Copy** selector to use with `html_nodes()`

.fa-bolt

Clear (1)   Toggle Position   XPath   Help   X

CSS selector to use

start over

move gadget

show XPath

help

close gadget

# Your Turn

Install SelectorGadget in your browser.

Then use selectorGadget to find a CSS selector combination that identifies just the cast member names.

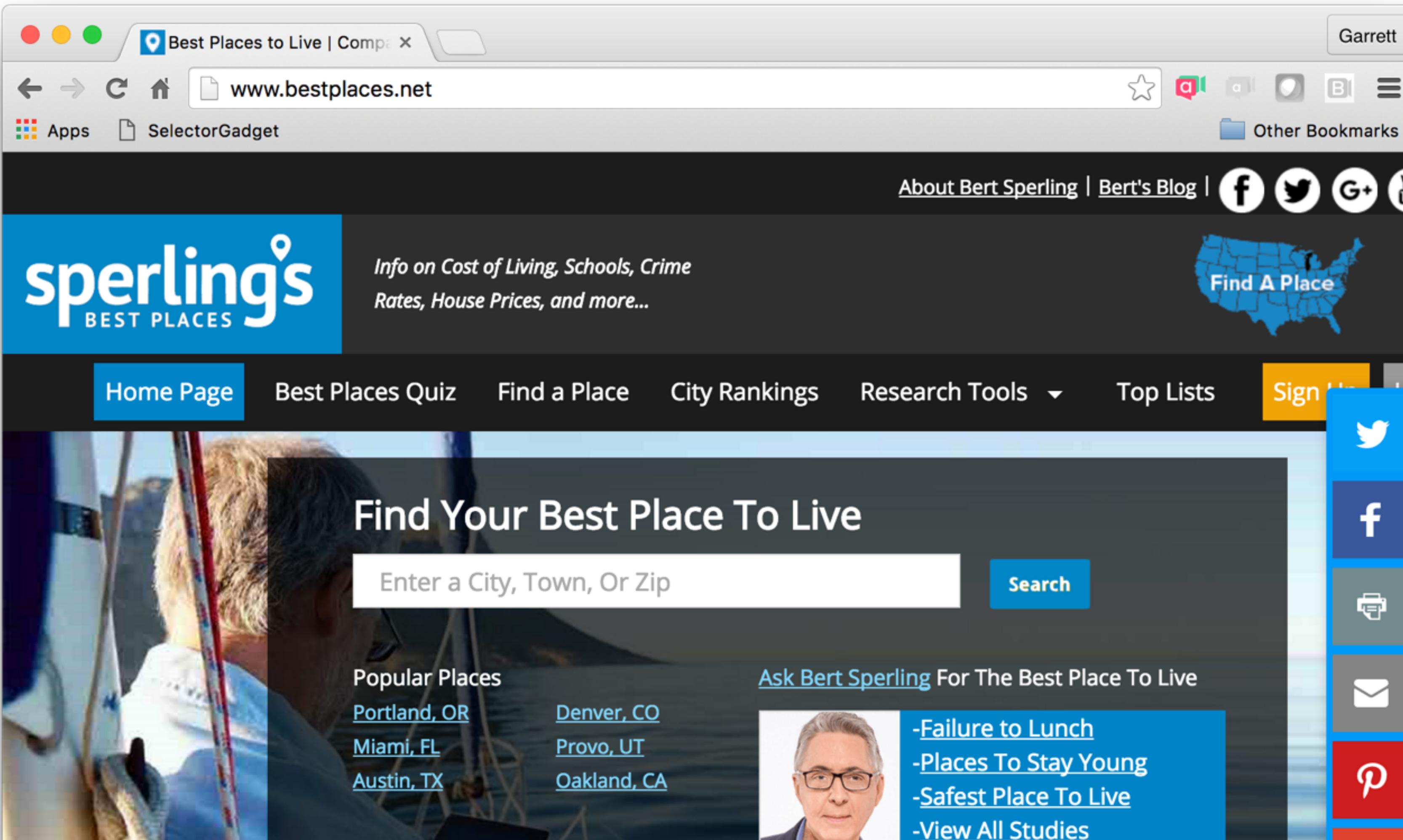


```
cast2 <- html_nodes(frozen, "#titleCast span.itemprop")
html_text(cast2)

cast3 <- html_nodes(frozen, ".itemprop .itemprop")
html_text(cast3)
```

# Bringing it home

<http://www.bestplaces.net/>



# Your Turn

Look up the cost of living for your hometown on <http://www.bestplaces.net/>. Then extract it with `html_nodes()` and `html_text()`.



```
kw <- read_html("http://www.bestplaces.net/cost_of_living/  
city/florida/key_west")  
  
col <- html_nodes(kw, css = "#mainContent_dgCostOfLiving  
tr:nth-child(2) td:nth-child(2)")  
html_text(col)  
  
kw %>%  
  html_nodes(css = "#mainContent_dgCostOfLiving tr:nth-  
child(2) td:nth-child(2")) %>%  
  html_text()
```

# tables

# Tables

Use `html_table()` to scrape whole tables of data as a data frame.

```
tables <- html_nodes(kw, css = "table")
html_table(tables, header = TRUE)[[2]]
```

# Your Turn

Visit the Climate tab for your home town. Extract the climate statistics of your hometown as a data frame with useful column names.



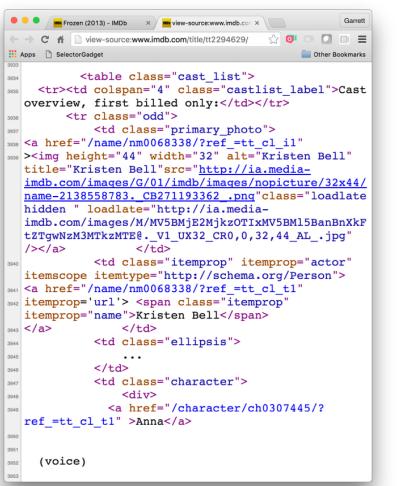
```
kw2 <- read_html("http://www.bestplaces.net/climate/city/  
florida/key_west")
```

```
climate <- html_nodes(kw2, css = "table")  
html_table(climate, header = TRUE)[[2]]
```

```
kw2 %>%  
  html_nodes(css = "table") %>%  
  html_table(header = TRUE) %>%  
  .[[2]]
```

# Recap

Pull from HTML, use HTML/CSS structure



**read\_html()** → **html\_nodes()** → **html\_text()**, etc.

**selectorGadget** for finding useful selector combinations




**html\_table()** for tables

thank you