

# JAVASCRIPT DEVELOPMENT

Sasha Vodnik, Instructor

# HELLO!

1. Pull changes from the `svodnik/JS-SF-13-resources` repo to your computer
2. Open the `08-advanced-jquery` folder in your code editor

---

**JAVASCRIPT DEVELOPMENT**

---

# **ADVANCED JQUERY**

# **LEARNING OBJECTIVES**

At the end of this class, you will be able to

- Use event delegation to manage dynamic content.
- Use implicit iteration to update elements of a jQuery selection
- Identify all the HTTP verbs & their uses.
- Describe APIs and how to make calls and consume API data.
- Access public APIs and get information back.

# **AGENDA**

- Event delegation
- Implicit iteration
- HTTP
- APIs

---

## ADVANCED JQUERY

---

# WEEKLY OVERVIEW

### WEEK 5

DOM & jQuery / Advanced jQuery

### WEEK 6

Ajax & APIs / Asynchronous JS & callbacks

Break

### WEEK 7

Advanced APIs / Project 2 lab

# **EXIT TICKET QUESTIONS**

1. What are some other frameworks and libraries out there that have animation/visuals?

# EXERCISE

---



## EXERCISE

### OBJECTIVE

---

- ▶ Manipulate the DOM and create DOM event handlers using jQuery

### LOCATION

---

- ▶ `starter-code > 0-dice-lab`

### TIMING

---

*20 min*

1. Use jQuery to create a page where every time the user hits the "Roll Dice" button, the screen randomly updates the two dice.
2. BONUS: Refactor your code to replace all jQuery with vanilla JavaScript.



# EXERCISE

---



## EXERCISE

### OBJECTIVE

---

- ▶ Manipulate the DOM and create DOM event handlers using jQuery

### LOCATION

---

- ▶ `starter-code > 1-bottles-lab`

### TIMING

---

*15 min*

1. Use jQuery to write the lyrics of 99 Bottles of Beer to the browser window.
2. Start by creating both an HTML file and a JavaScript file.
  - In your HTML file, you'll want to include an unordered list (`<ul>`) that will contain all of your lyrics.
  - Each line of the song should appear inside of a list item (`<li>`) within that unordered list.
3. BONUS: Refactor your code to replace all jQuery with vanilla JavaScript.

# **BEST PRACTICES**

---

**ADVANCED JQUERY**

---

# METHOD CHAINING

# CHAINING

without chaining:

```
let $mainCaption = $( '<p>' );  
let $captionWithText = $mainCaption.html( 'Today' );  
let $fullCaption = $captionWithText.addClass( 'accent' );
```

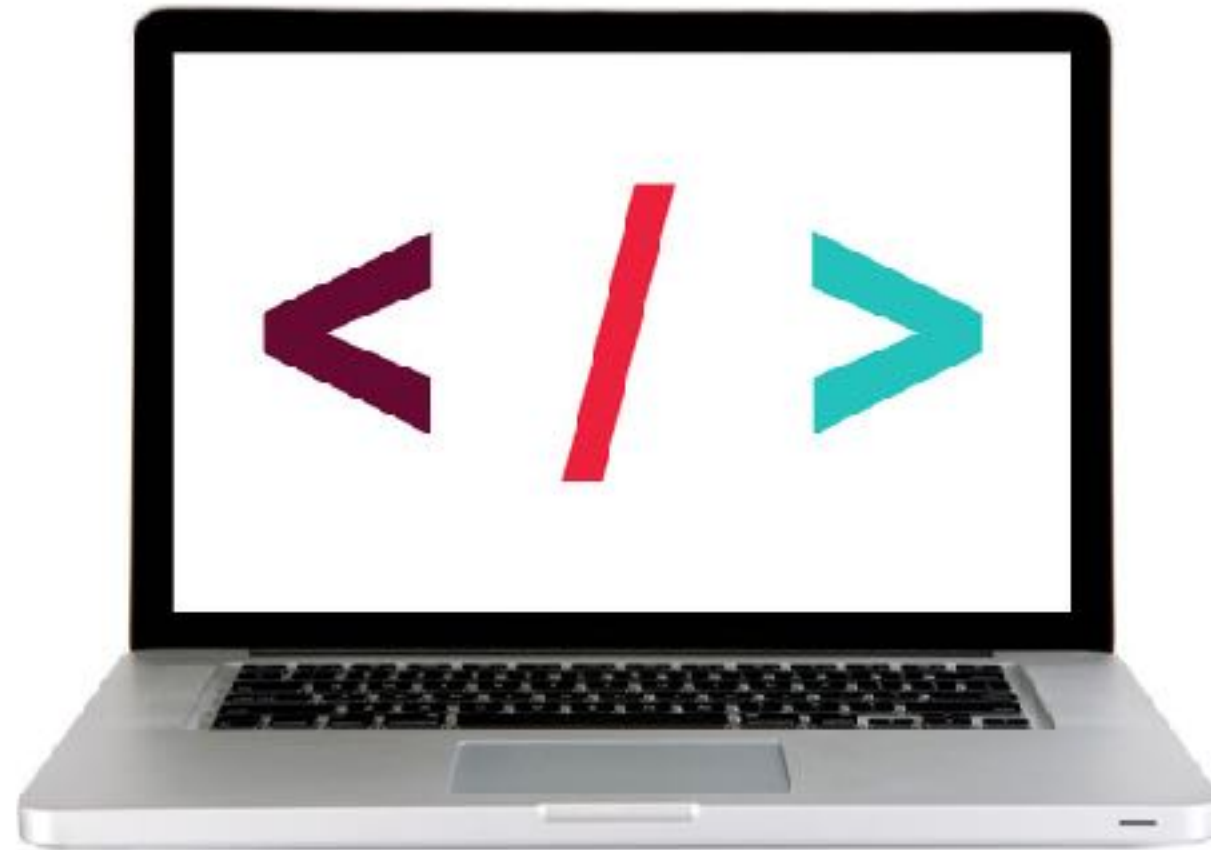
with chaining:

```
let $fullCaption = $( '<p>' ).html( 'Today' ).addClass( 'accent' );
```

---

## ADVANCED JQUERY

---



**LET'S TAKE A CLOSER LOOK**

# EXERCISE – CHAINING

---



## EXERCISE

### OBJECTIVE

---

- ▶ Use chaining to place methods on selectors.

### LOCATION

---

- ▶ `starter-code > 3-best-practices-exercise`

### TIMING

---

*3 min*

1. In your browser, open `index.html` and test the functionality.
2. Open `main.js` in your editor and complete items 1 and 2.
3. In your browser, reload `index.html` and verify that the functionality is unchanged.

# IMPLICIT ITERATION

# IMPLICIT ITERATION

## explicit iteration

```
$( 'li' ).each(function() {  
    $(this).removeClass( 'current' );  
});
```

jQuery .each() method works like a forEach loop



not necessary for  
element collections

## implicit iteration

```
$( 'li' ).removeClass( 'current' );
```

applying any method to a jQuery collection iterates through each element!



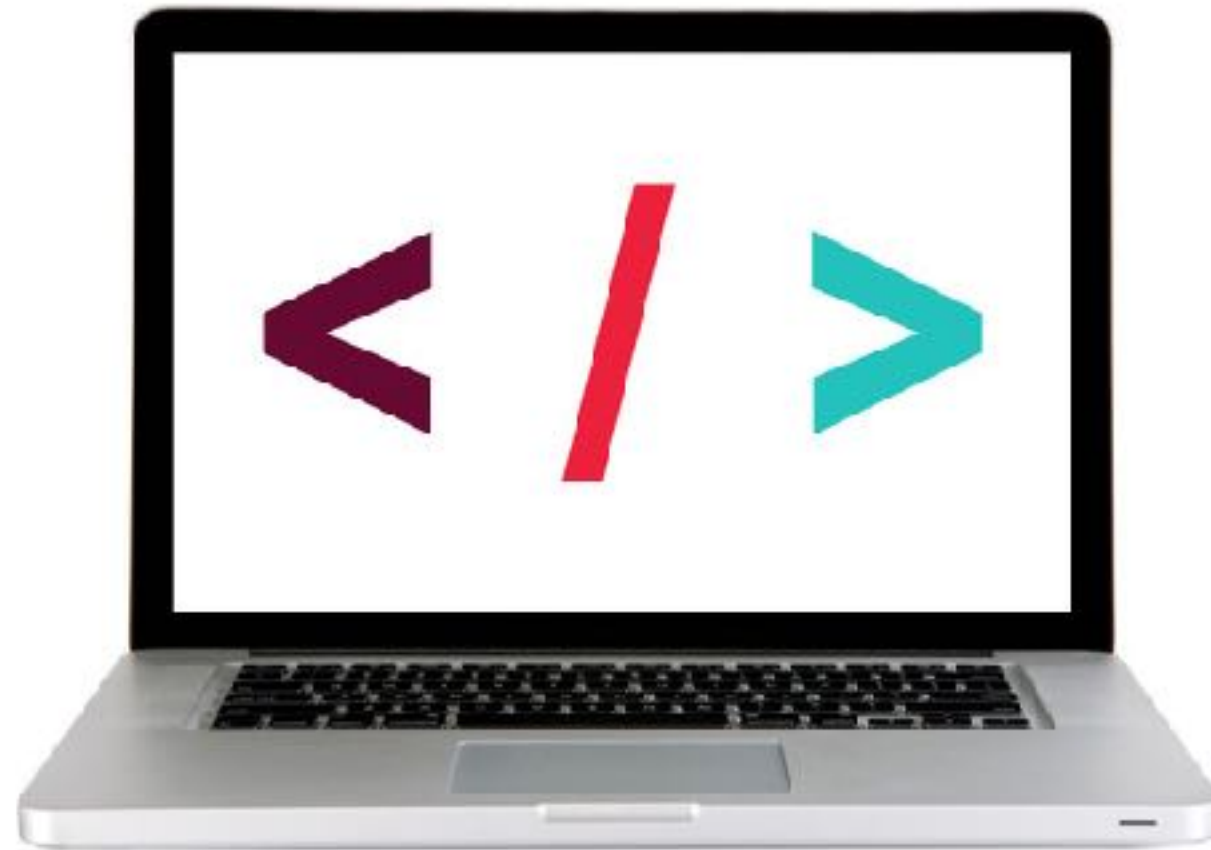
less code = best practice!



---

## ADVANCED JQUERY

---



**LET'S TAKE A CLOSER LOOK**

# EXERCISE – IMPLICIT ITERATION

---



## EXERCISE

### OBJECTIVE

---

- ▶ Use implicit iteration to update elements of a jQuery selection.

### LOCATION

---

- ▶ `starter-code` > `3-best-practices-exercise`

### TIMING

---

*5 min*

1. Return to `main.js` in your editor and complete item 3.
2. In your browser, reload `index.html` and verify that the functionality is unchanged.

---

**ADVANCED JQUERY**

---

# EVENT DELEGATION

# WITHOUT EVENT DELEGATION

1. load page

2. set event listener  
on list items

```
$( 'li' ).on( 'click', function() {
    addClass( 'selected' )
});
```

- item1
- item2
- item3

- |        |             |
|--------|-------------|
| •item1 | click event |
| •item2 | click event |
| •item3 | click event |

3. add a new list item

- |        |             |
|--------|-------------|
| •item1 | click event |
| •item2 | click event |
| •item3 | click event |
| •item4 |             |

click event is not automatically  
applied to the new li element



# WITH EVENT DELEGATION

1. load page

- item1
- item2
- item3

2. set event listener  
on *parent of* list items

selector changed to parent

new second argument specifies children

```
$('ul').on('click', 'li', function(){  
  addClass('selected')  
});
```

- item1
- item2
- item3

click event

click event

click event

3. add a new list item

- item1
- item2
- item3
- item4

click event

click event

click event

click event

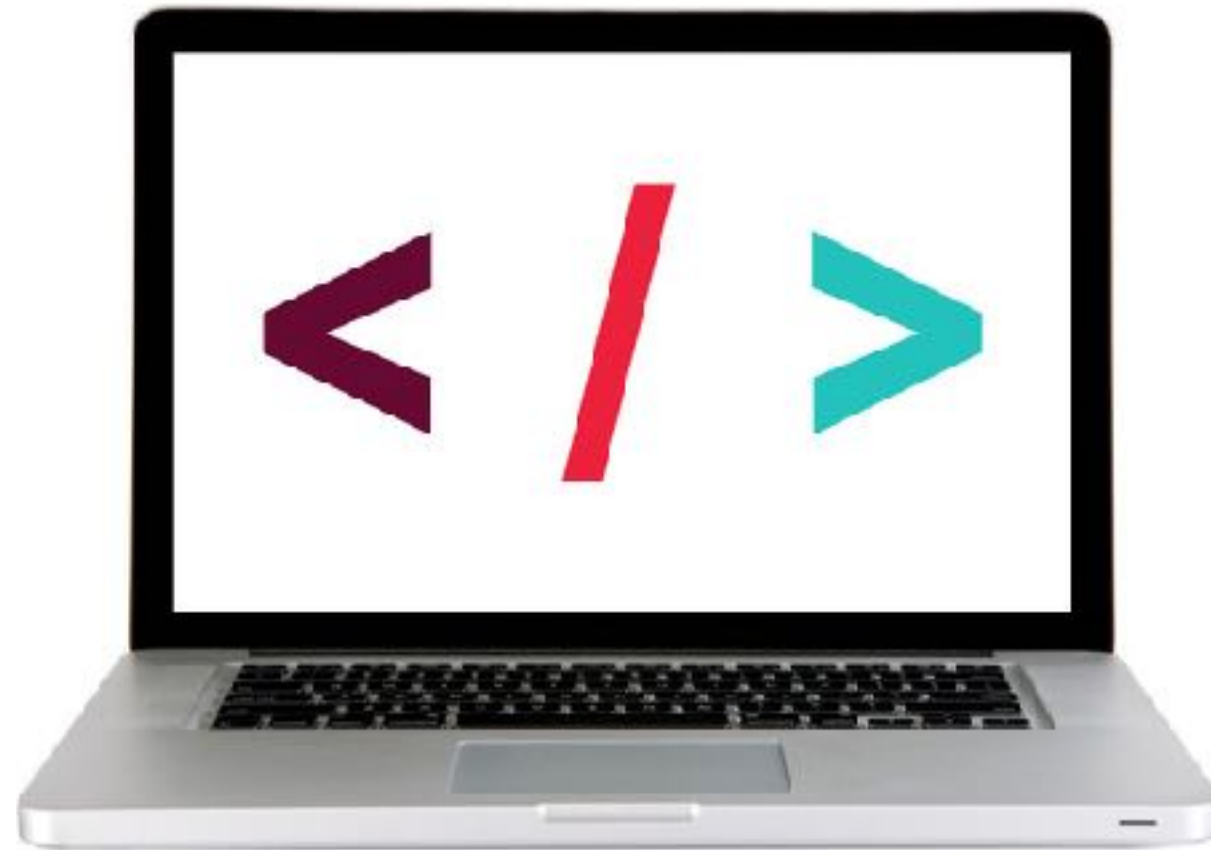
click event IS automatically applied to the new li element!

✓

---

## ADVANCED JQUERY

---



**LET'S TAKE A CLOSER LOOK**

# EXERCISE – EVENT DELEGATION

---



## EXERCISE

### OBJECTIVE

---

- ▶ Use event delegation to manage dynamic content.

### LOCATION

---

- ▶ `starter-code > 3-best-practices-exercise`

### TIMING

---

*10 min*

1. Return to `main.js` in your editor and complete item 4.
2. In your browser, reload `index.html` and verify that when you add a new item to the list, its “cross off” link works.
3. BONUS 1: When the user mouses over each item, the item should turn grey. Don't use CSS hovering for this.
4. BONUS 2: Add another link, after each item, that allows you to delete the item.

# ATTACHING MULTIPLE EVENTS WITH A SINGLE ON() STATEMENT



# ATTACHING MULTIPLE EVENTS WITH A SINGLE .ON() STATEMENT

- We could write a separate .on() statement for each event on an element:

```
var $listElement = $('#contents-list');

$listElement.on('mouseenter', 'li', function(event) {
    $(this).siblings().removeClass('active');
    $(this).addClass('active');
});

$listElement.on('mouseleave', 'li', function(event) {
    $(this).removeClass('active');
});
```

# ATTACHING MULTIPLE EVENTS WITH A SINGLE .ON() STATEMENT

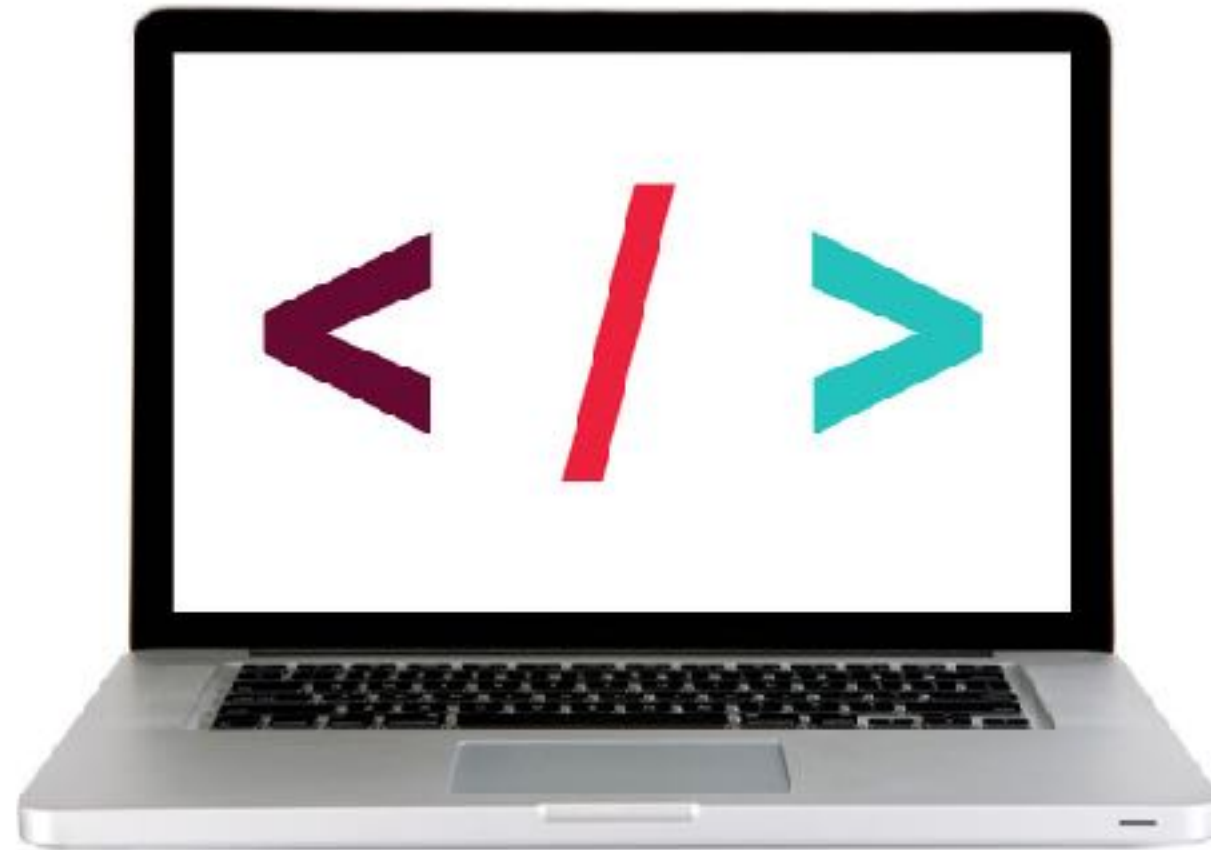
```
var $listElement = $('#contents-list');

$listElement.on('mouseenter mouseleave', 'li', function(event) {
    if (event.type === 'mouseenter') {
        $(this).siblings().removeClass('active');
        $(this).addClass('active');
    } else if (event.type === 'mouseleave') {
        $(this).removeClass('active');
    }
});
```

---

## ADVANCED JQUERY

---



**LET'S TAKE A CLOSER LOOK**

# EXERCISE – ATTACHING MULTIPLE EVENTS

---



## EXERCISE

### LOCATION

---

► starter-code > 4-multiple-events-exercise

### TIMING

---

*5 min*

1. In your browser, open index.html. Move the mouse over each list item and verify that the sibling items turn gray.
2. In your editor, open main.js and refactor the two event listeners near the bottom of the file into a single event listener for multiple events.
3. In your browser, reload index.html and verify that the functionality is unchanged.

# APIs

# WEB SERVICES

Your app



Web service



request for data



response containing data



your app can now  
incorporate data from  
the web service

my website  
content

# SASHA VODNIK

Instructor and Author on  
Programming and  
Technology


Home

Books


Content  
from  
Twitter  
added  
using  
Twitter API

### FOLLOW ME ON TWITTER


Tweets by @sashavodnik

 **Sasha Vodnik**  
@sashavodnik


Take the next step to secure your passwords, browsing, and networking in my workshop this Thursday! [@GGA\\_SF](#) [@GGA](#) [#onlinesecurity](#) [#infosecurity](#) [generalassemblyeducationacad...](#)

 **Securing Your Digital ...**  
Learn to keep your data ...  
[generalassembly](#)

Mar 4, 2018

 **Sasha Vodnik**  
@sashavodnik

AR makes so much more sense to me than what VR, and here are some reasons why! ⚡ "LukoVR's Augmented Reality Examples" by @jeramiaswhar.com/!moments6677...

 **LukoVR's Augmented Reality Examples**  
by [Jeremy Q...](#)

impersonating you and resetting your password to one they choose. The result is that they have access to your account, while you are locked out. To defend against this type of attack, many web services allow you to set up two factor authentication (2FA).

Continue reading →

Share this:



★ Like



One blogger likes this.

## Securing Your Digital Life, Part 1: Choosing a password manager

JANUARY 11, 2018

Configuring and using a password manager is a critical building block of your online security.

Continue reading →

Share this:



★ Like



One blogger likes this.

Kayak  
website  
content



KAYAK Hotels Flights Cars Packages More ▾

One-way ▾ 2 travelers ▾ Economy ▾

✈ Dublin (DUB) ↔ ✈ San Francisco (SFO) 📅 Thu 8/23 ✎

Our Advice **Buy now**

Prices are unlikely to decrease within 7 days ☹

Track Prices ☐

887 of 1411 flights Sorted by Recommended ▾

Fee Assistant ⓘ Carry-on bag - 0 + Checked bag - 0 +

**Orbitz Flight Deals – Price Guarantee**

Plus earn Orbitz Rewards

Orbitz.com | Sponsored [View Deal](#)

**BEST FLIGHTS** ⓘ

8/24 Fri	Multiple Airlines	11:45 am DUB	— — GPH	4:16 pm OAK (+1)	8h 30m	<b>\$360</b> KIWI.COM	<a href="#">View Deal</a>
8/26 Sun	WOW air	11:45 am DUB	— — KEF	5:15 pm SFO	13h 30m	<b>\$419</b> KAYAK	<a href="#">View Deal</a> ▾
8/23 Thu	Air Lingus	12:30 pm DUB	— — nonsstop	3:30 pm SFO	11h 00m	<b>\$851</b> OneTravel	<a href="#">View Deal</a> ▾

\$856 book early on KAYAK

Content  
from  
kiwi.com  
using API

Content  
from  
OneTravel  
using API



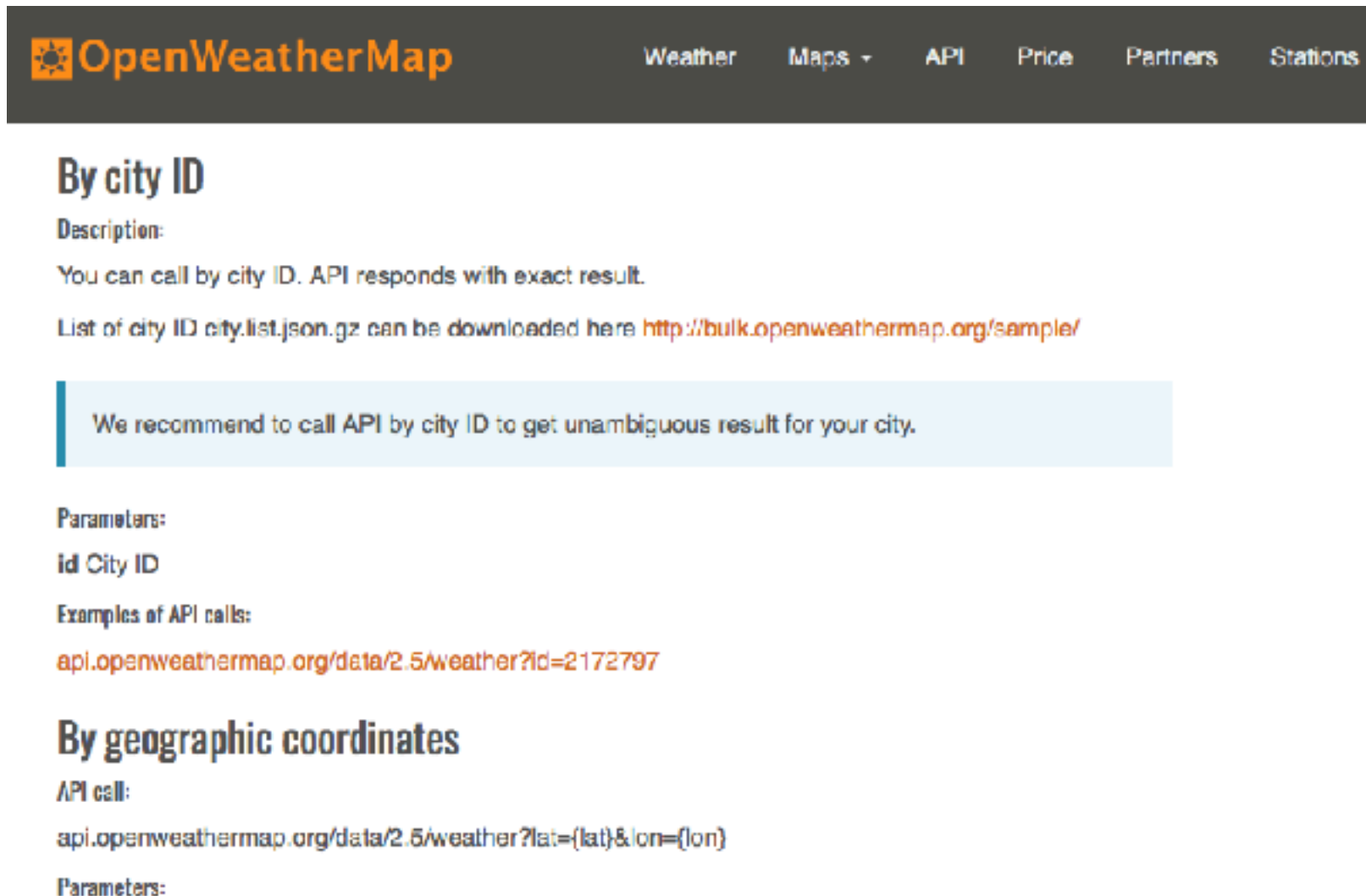
# WEB SERVICES



OMDb



# API = application programming interface



The screenshot shows the OpenWeatherMap website's API documentation. At the top is a dark navigation bar with the OpenWeatherMap logo and links for Weather, Maps, API, Price, Partners, and Stations. The main content area is titled 'By city ID' and includes a description, a link to a city ID list, a recommendation box, and API call examples. Below this, the 'By geographic coordinates' section is partially visible.

**OpenWeatherMap** Weather Maps API Price Partners Stations

## By city ID

Description:

You can call by city ID. API responds with exact result.

List of city ID city.list.json.gz can be downloaded here <http://bulk.openweathermap.org/sample/>

We recommend to call API by city ID to get unambiguous result for your city.

Parameters:

id City ID

Examples of API calls:

[api.openweathermap.org/data/2.5/weather?id=2172797](http://api.openweathermap.org/data/2.5/weather?id=2172797)

## By geographic coordinates

API call:

[api.openweathermap.org/data/2.5/weather?lat={lat}&lon={lon}](http://api.openweathermap.org/data/2.5/weather?lat={lat}&lon={lon})

Parameters:

# APIS IN THE REAL WORLD

- Most APIs are unique, like separate languages
- APIs for
  - devices (iPhone)
  - operating systems (macOS)
  - JavaScript libraries (jQuery API)
  - services (Slack)



MacOS



# WEB SERVICES



OMDb



# ENDPOINTS

- Addresses (URLs) that return data (JSON) instead of markup (HTML)

## By city ID

### Description:

You can call by city ID. API responds with exact result.

List of city ID `city.list.json.gz` can be downloaded here <http://bulk.openweathermap.org/sample/>

We recommend to call API by city ID to get unambiguous result for your city.

### Parameters:

id City ID

### Examples of API calls:

`api.openweathermap.org/data/2.5/weather?id=2172797`

## By geographic coordinates

### API call:

`api.openweathermap.org/data/2.5/weather?lat={lat}&lon={lon}`

### Parameters:

lat, lon coordinates of the location of your interest

### Examples of API calls:

`api.openweathermap.org/data/2.5/weather?lat=35&lon=139`

### API respond:

```
{
  "coord": {
    "lon": 139,
    "lat": 35
  },
  "sys": {
    "country": "JP",
    "sunrise": 1369769524,
    "sunset": 1369821849
  },
  "weather": [
    {
      "id": 804,
      "main": "clouds",
      "description": "overcast clouds",
      "icon": "04n"
    }
  ],
  "main": {
    "temp": 289.5,
    "humidity": 89,
    "pressure": 1013,
    "temp_min": 287.04,
    "temp_max": 292.04
  },
  "wind": {
    "speed": 7.31,
    "deg": 187.002
  },
  "rain": {
    "3h": 0
  },
  "clouds": {
    "all": 92
  },
  "dt": 1369824608
}
```

# **WHAT WE NEED TO KNOW TO USE AN API**

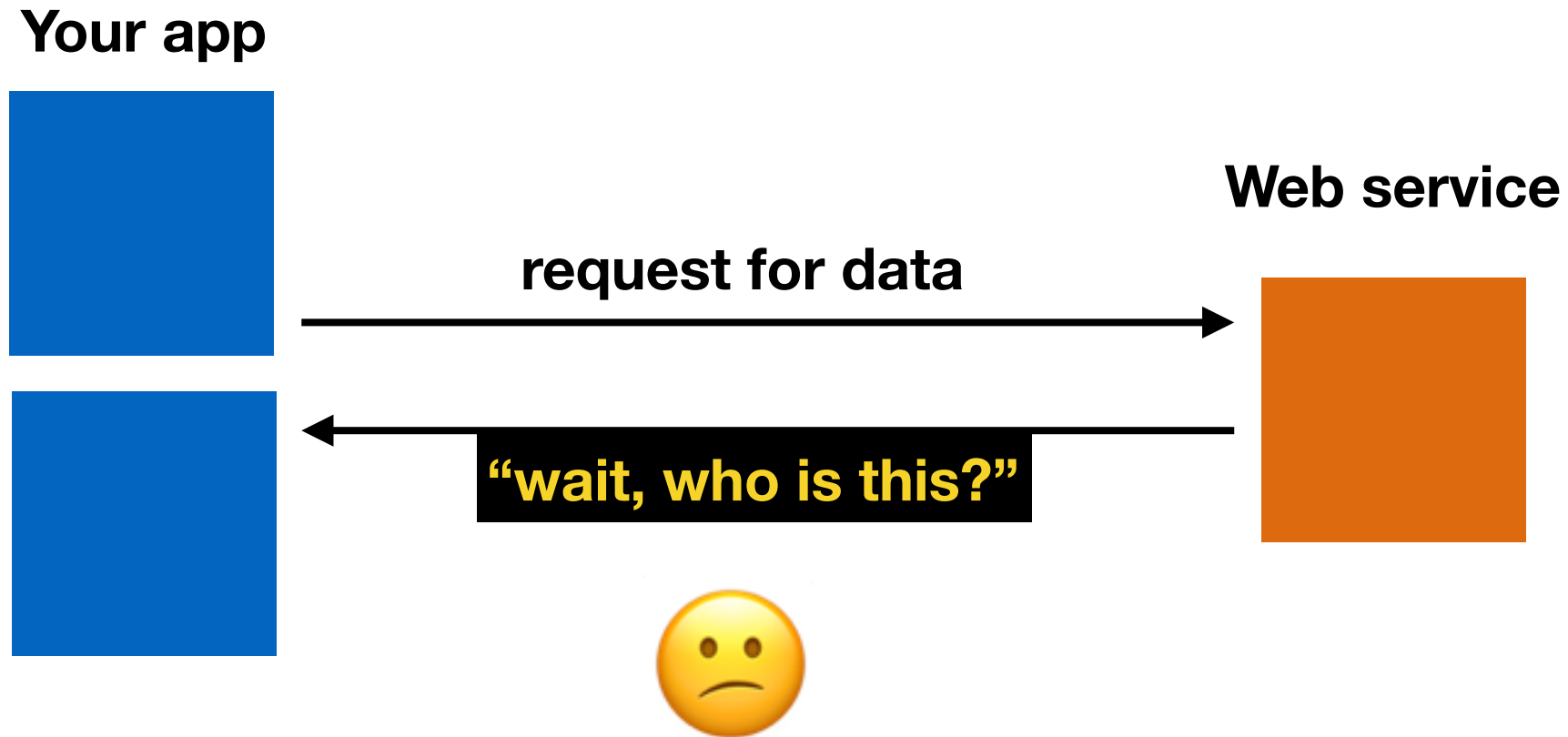


**TERMS OF  
SERVICE**


**HOW TO  
MAKE A  
REQUEST**

**HOW TO  
UNDERSTAND  
RESPONSE**

# AN API MIGHT REQUIRE AUTHENTICATION



# API KEY

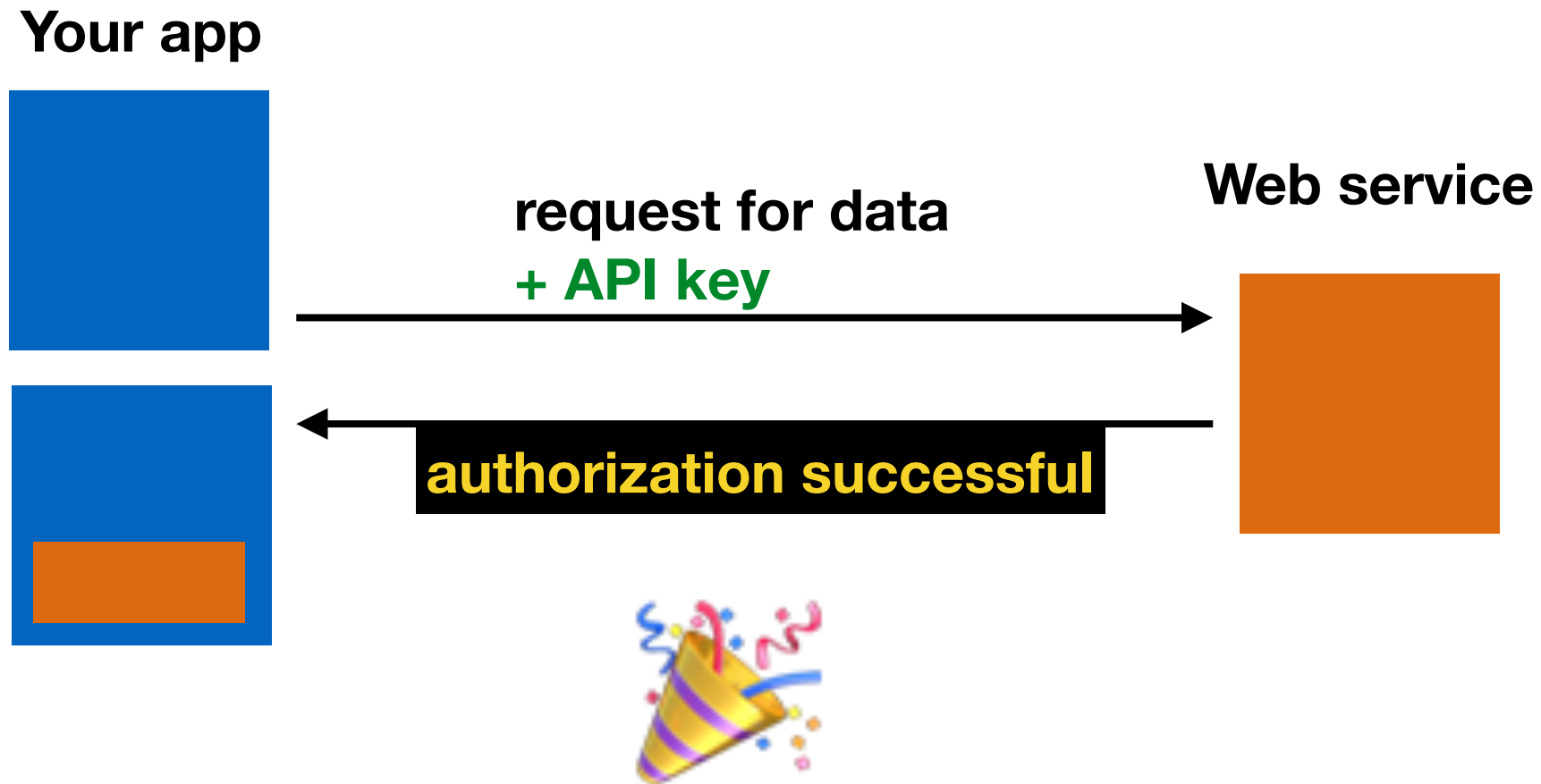


asd32f;li^6uq439#0587adf;lgk@

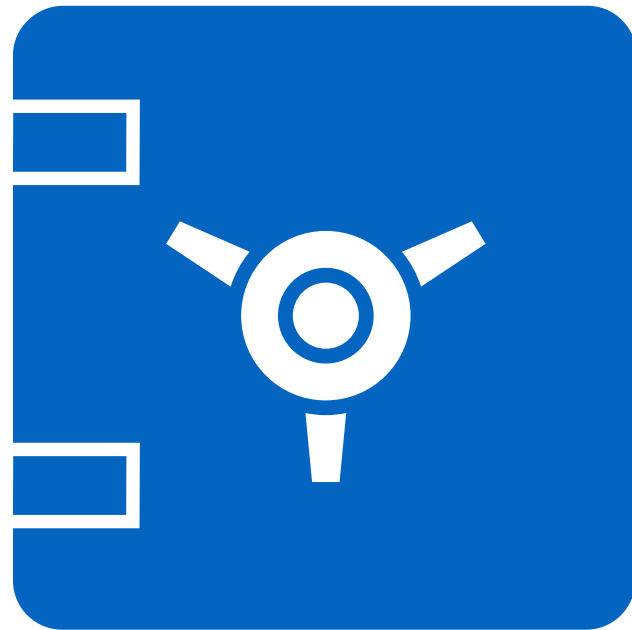
- unguessable character string
- connects your requests to your account



# API REQUEST WITH AUTHENTICATION

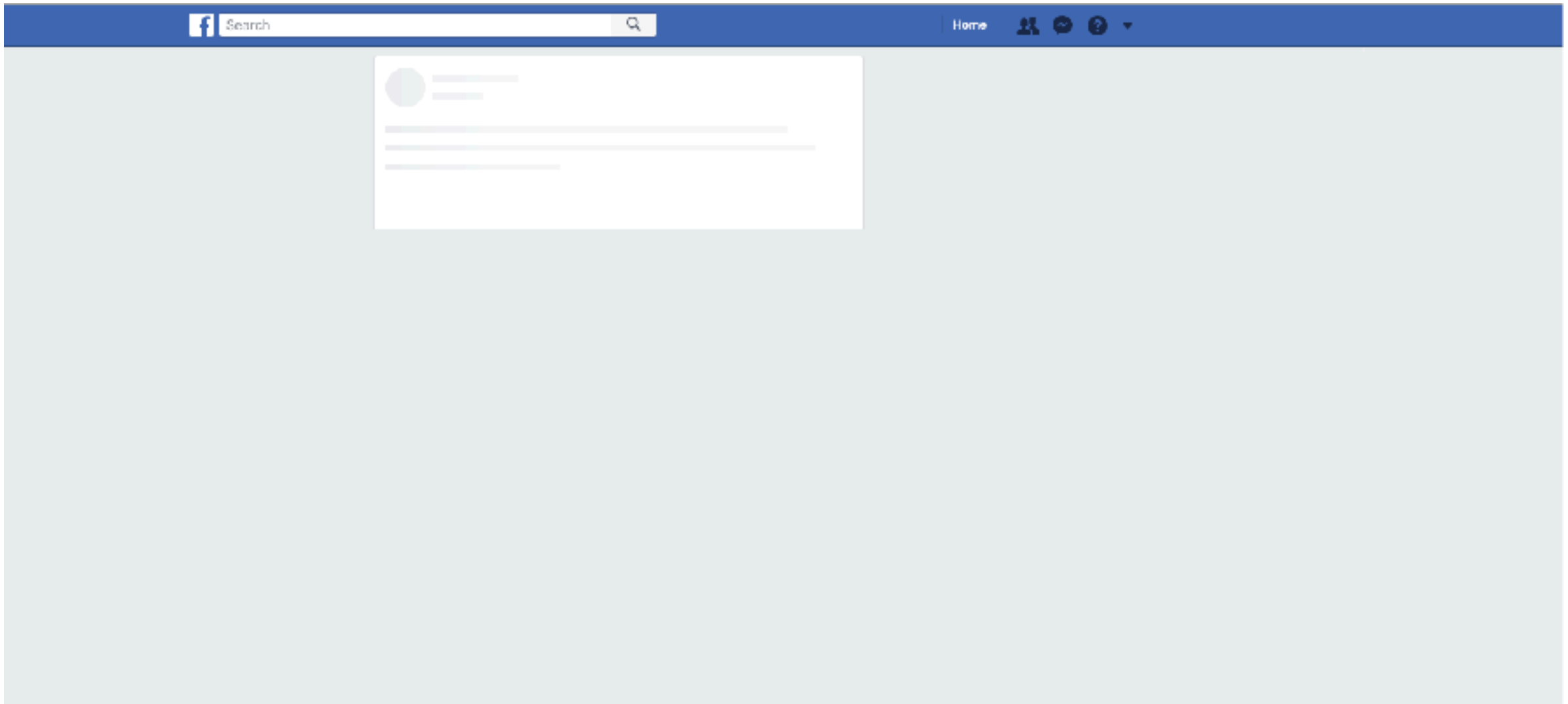


# **KEEP YOUR API CREDENTIALS PRIVATE**



- Don't post to a public code repo
- Don't share with other developers outside of your organization

# YOUR APP MIGHT EXPERIENCE A DELAYED RESPONSE

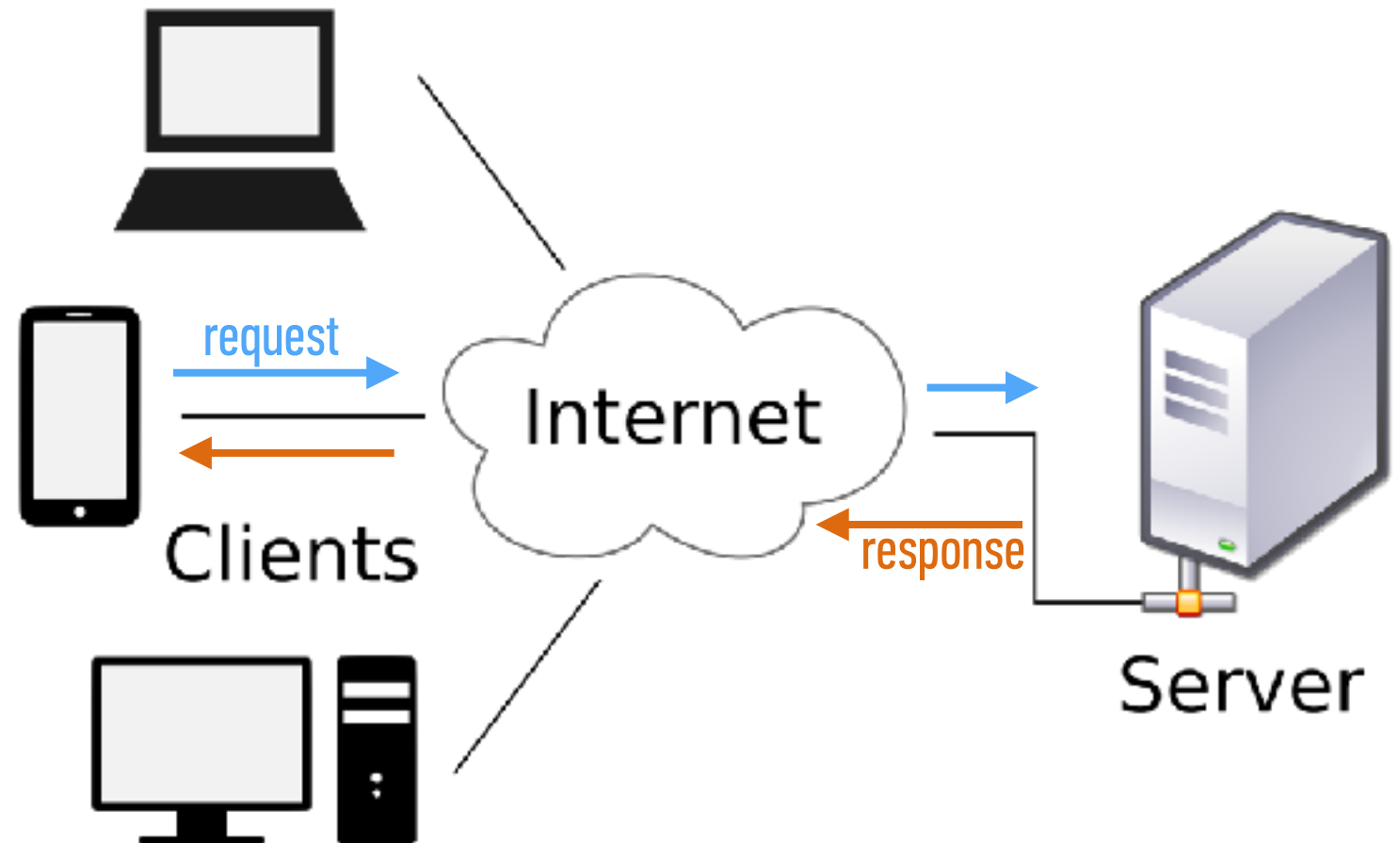


# YOUR REQUEST MAY RESULT IN AN ERROR



# REST (representational state transfer)

- architectural style of web applications
- transfers a representation of the state of a resource between the server and the client



# RESTful API

- adheres to REST architecture
- uses
  - a base URL
  - an Internet media type (such as JSON)
  - standard HTTP methods

## By geographic coordinates

API call:

`api.openweathermap.org/data/2.5/weather?lat={lat}&lon={lon}`

Parameters:

lat, lon coordinates of the location of your interest

Examples of API calls:

`api.openweathermap.org/data/2.5/weather?lat=35&lon=139`

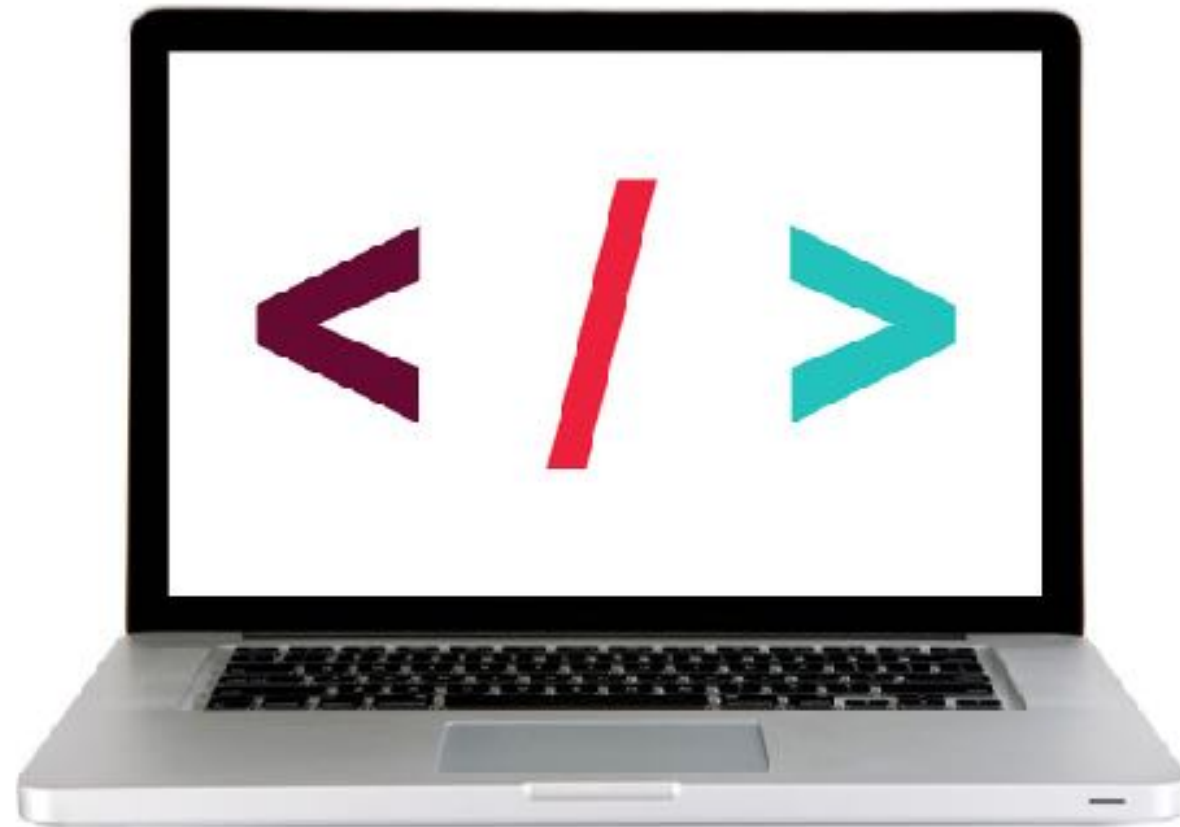
API respond:

```
{
  "coord": {
    "lon": 139,
    "lat": 35
  },
  "sys": {
    "country": "JP",
    "sunrise": 1369769524,
    "sunset": 1369821049
  },
  "weather": [
    {
      "id": 804,
      "main": "clouds",
      "description": "overcast clouds",
      "icon": "04n"
    }
  ],
  "main": {
    "temp": 289.5,
    "humidity": 89,
    "pressure": 1013,
    "temp_min": 287.04,
    "temp_max": 292.04
  },
  "wind": {
    "speed": 7.31,
    "deg": 187.002
  },
  "rain": {
    "3h": 0
  },
  "clouds": {
    "all": 92
  },
  "dt": 1369824698,
  "id": 1851632,
  "name": "Shuzenji",
  "cod": 200
}
```

---

## LET'S TAKE A CLOSER LOOK

---

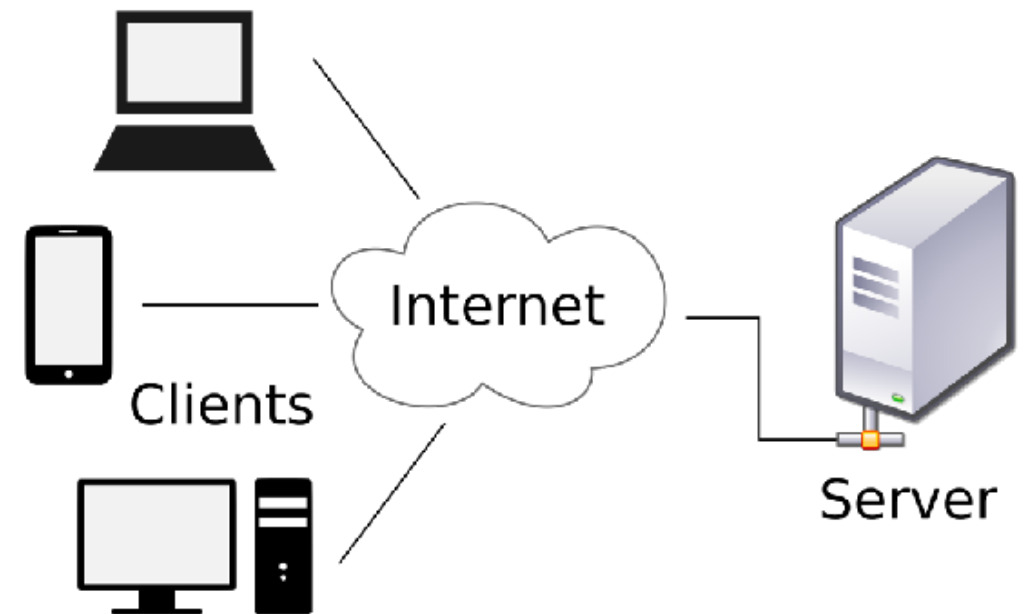


# HTTP



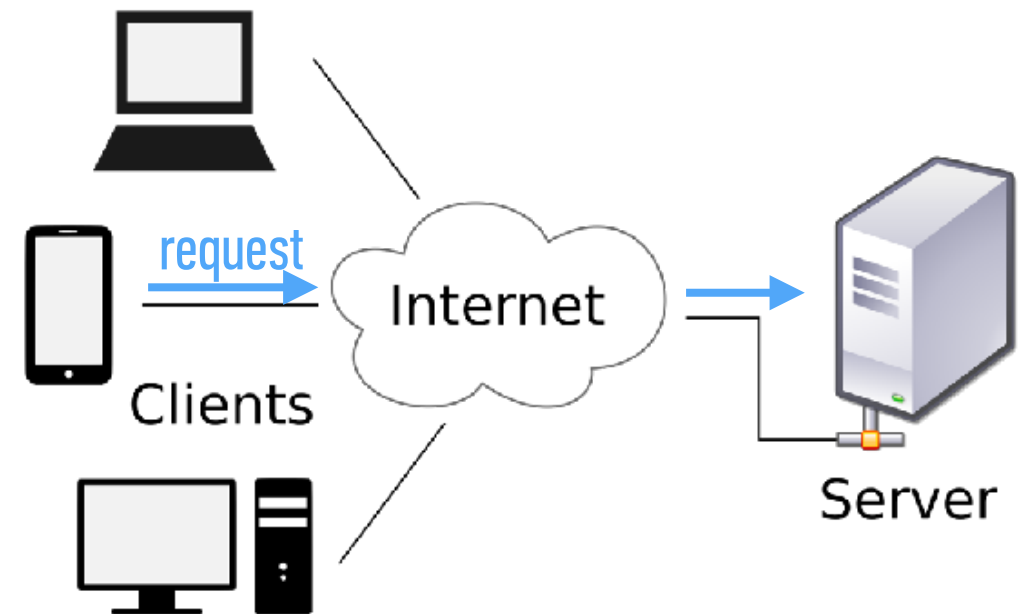
# HTTP (hypertext transfer protocol)

- System of rules for how web pages are transmitted between computers
- Defines the format of messages passed between HTTP clients and HTTP servers



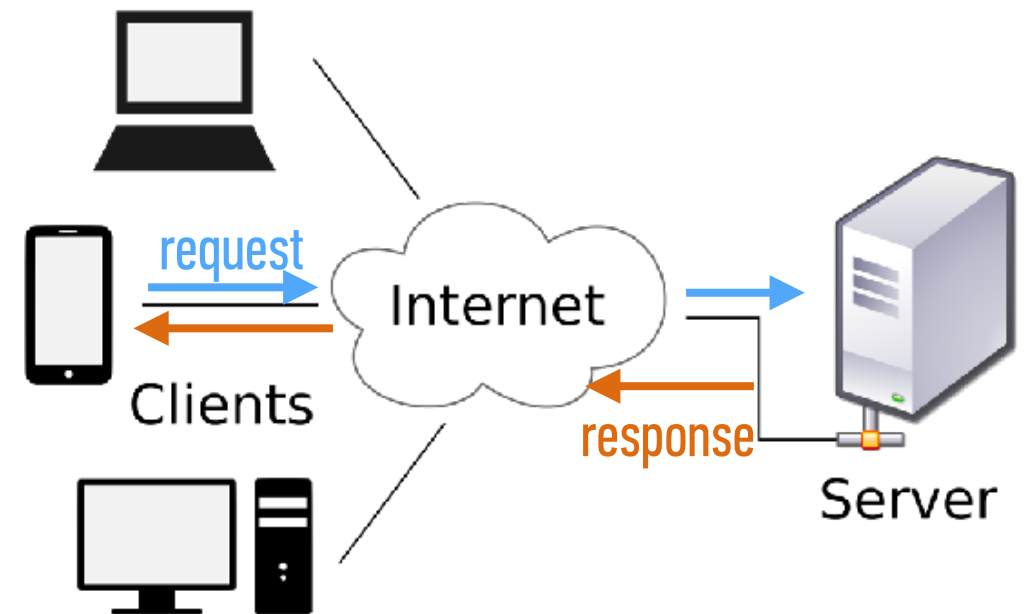
# HTTP (hypertext transfer protocol)

- A client sends a **request** to a server.

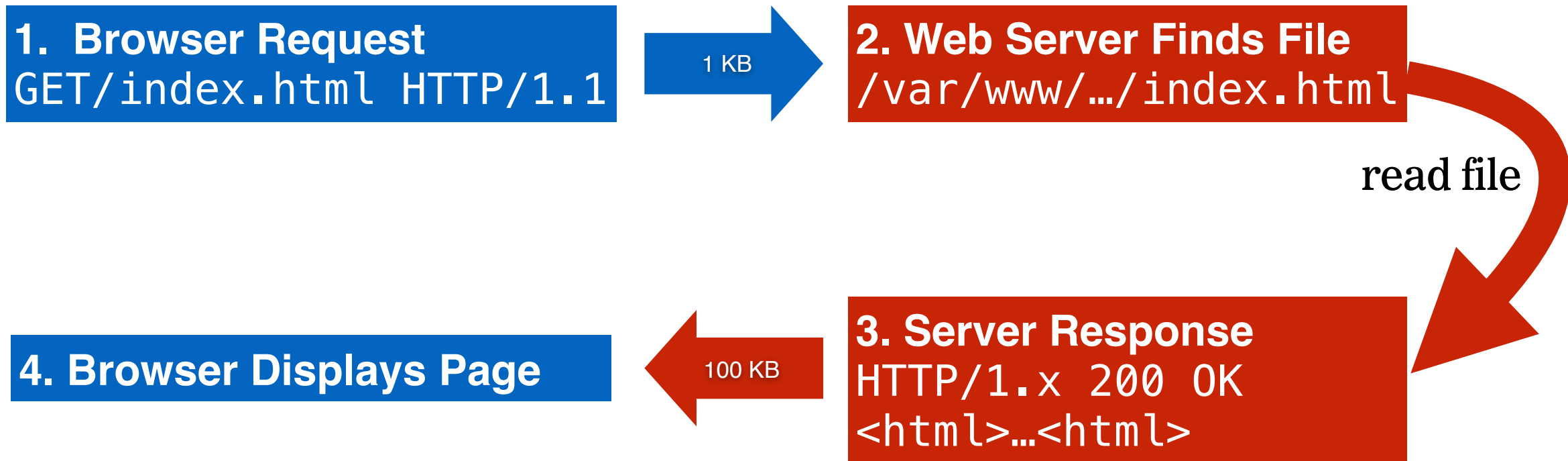


# HTTP (hypertext transfer protocol)

- A server sends a **response** back to a client.



# HTTP REQUEST AND RESPONSE



# HTTP (hypertext transfer protocol)

**HTTP client**

web browser



**HTTP server**

web server  
software



**Web service**

app that responds  
to HTTP requests



OMDb



# HTTP REQUESTS IN EVERYDAY LIFE

protocol

host

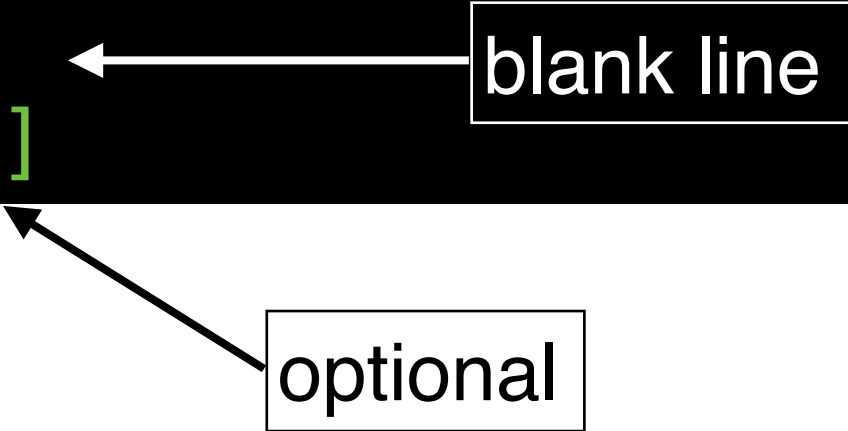
resource path

query

`https://www.domain.com/path/to/resource?a=b&x=y`

# HTTP REQUEST STRUCTURE

```
[http request method] [URL] [http version]  
[list of headers]  
[request body]
```



The diagram illustrates the structure of an HTTP request. It consists of four lines of text in a monospace font, colored green on a black background. The first line is '[http request method] [URL] [http version]'. The second line is '[list of headers]'. The third line is '[request body]'. A white arrow points from a box labeled 'blank line' to the space between the second and third lines. A black arrow points from a box labeled 'optional' to the third line.

blank line

optional

# HTTP REQUEST METHODS (“HTTP VERBS”)

GET	Retrieve a resource
POST	Create a resource
PATCH	Update an existing resource (use instead of PUT, which replaces)
DELETE	Delete a resource
HEAD	Retrieve the headers for a resource

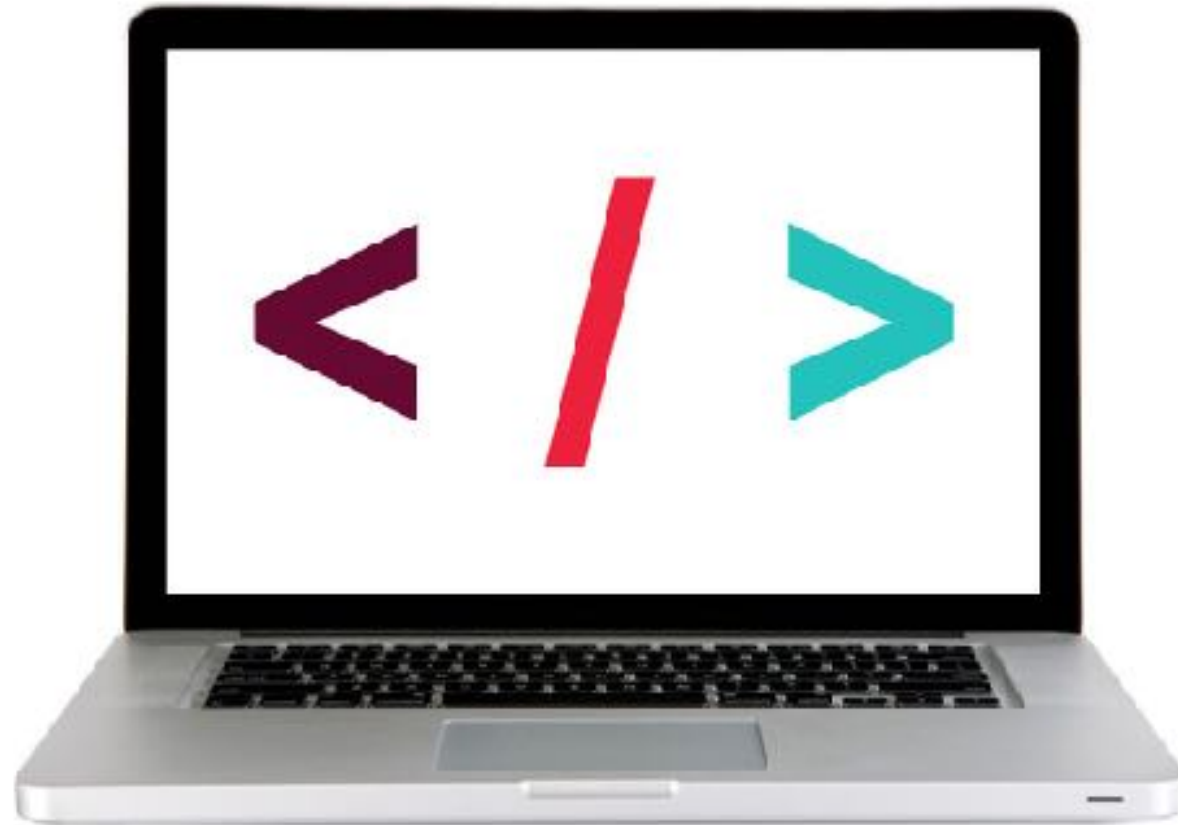
Most widely used



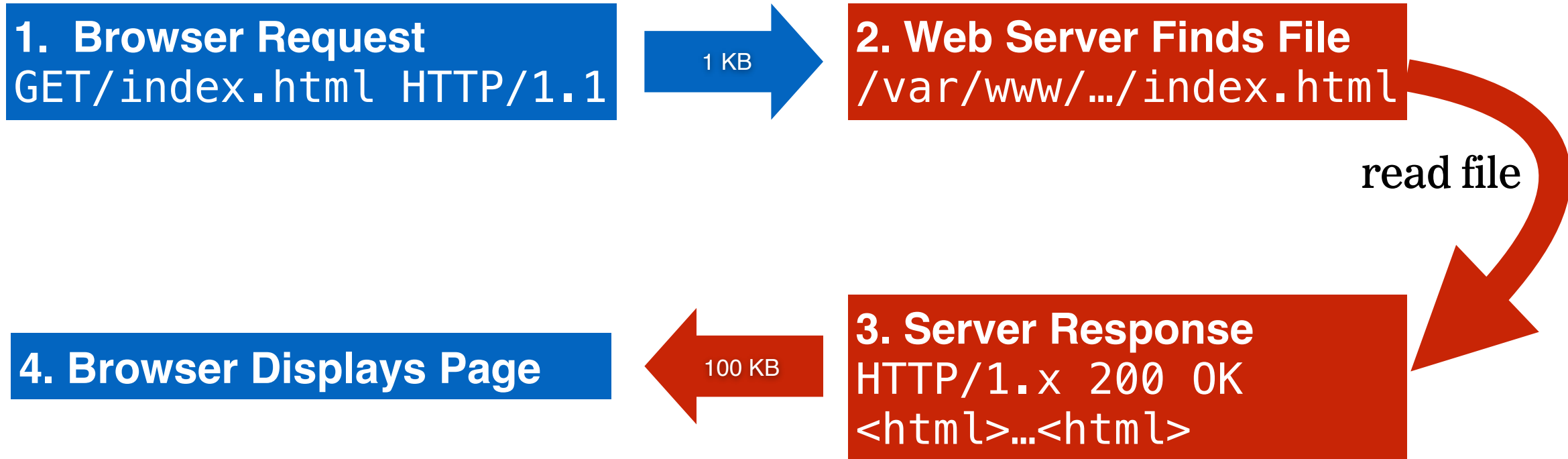
---

## LET'S TAKE A CLOSER LOOK

---



# HTTP REQUEST AND RESPONSE



# HTTP RESPONSE STRUCTURE

```
[http version] [status] [reason]  
[list of headers]  
[response body]
```

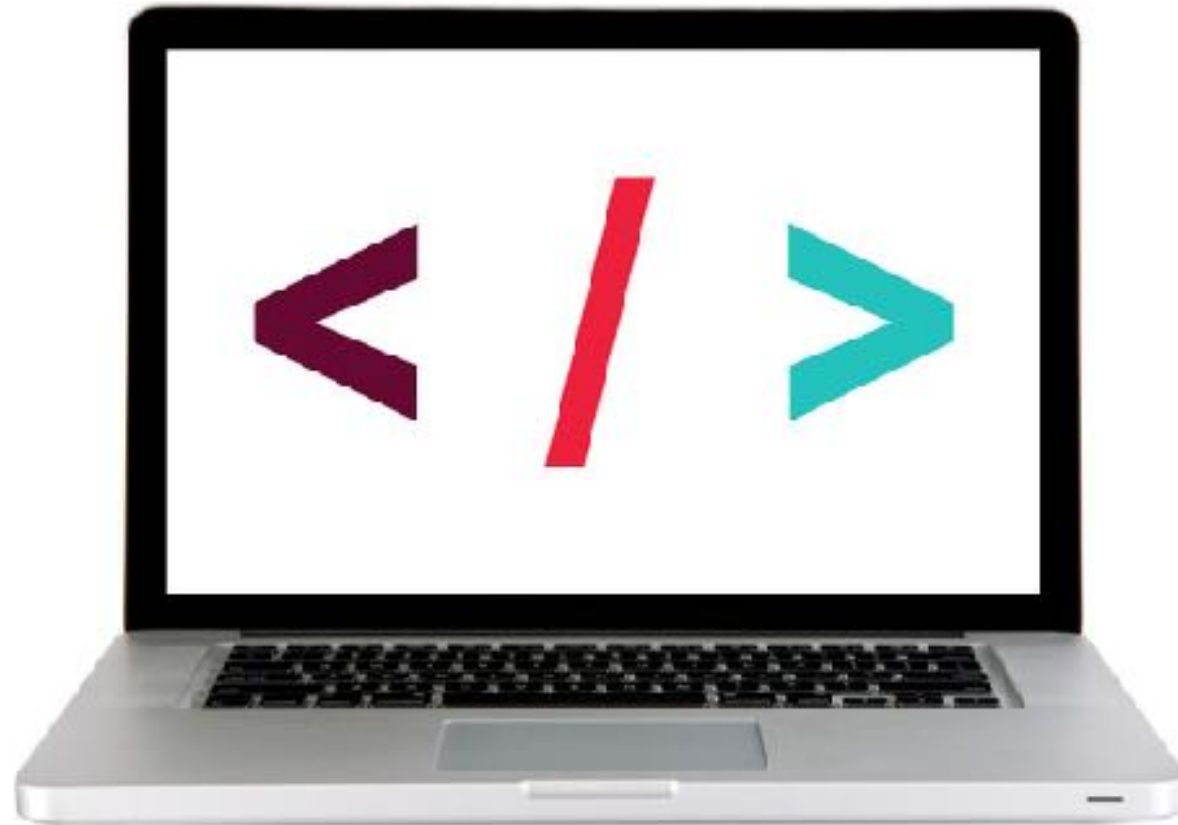
← blank line

↖ usually HTML, JSON, etc

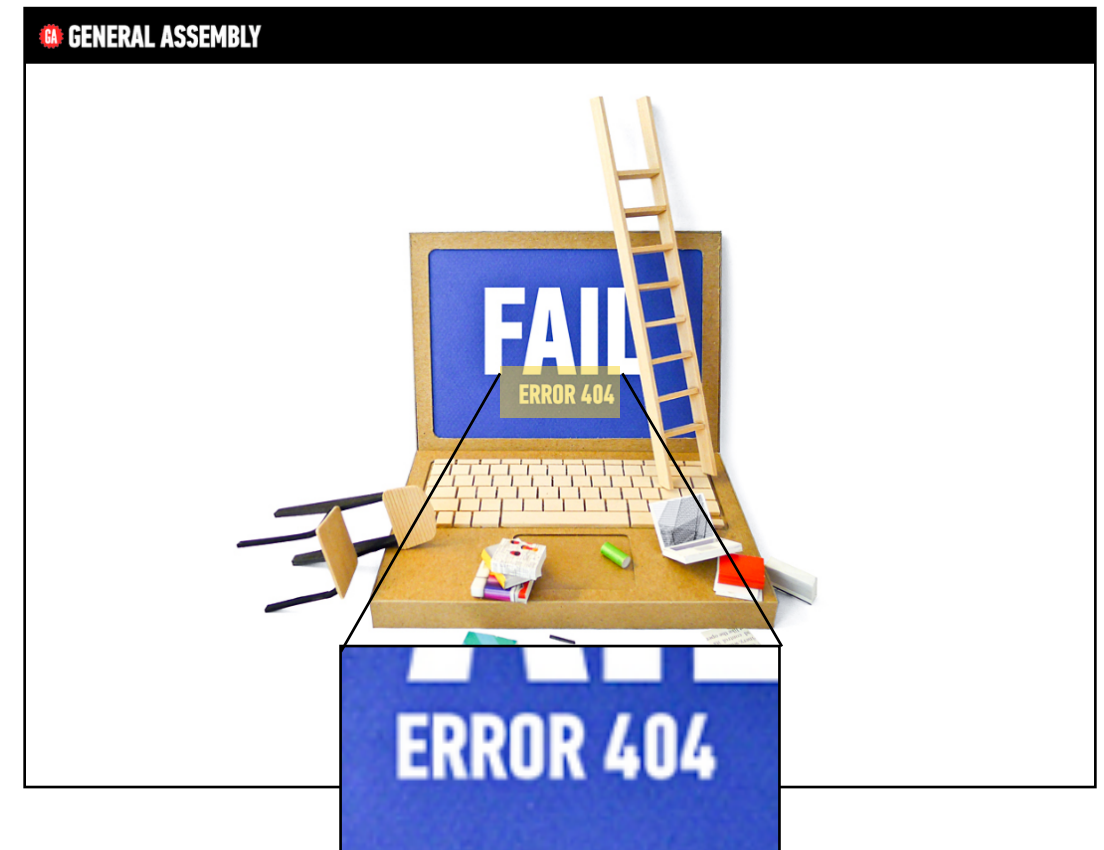
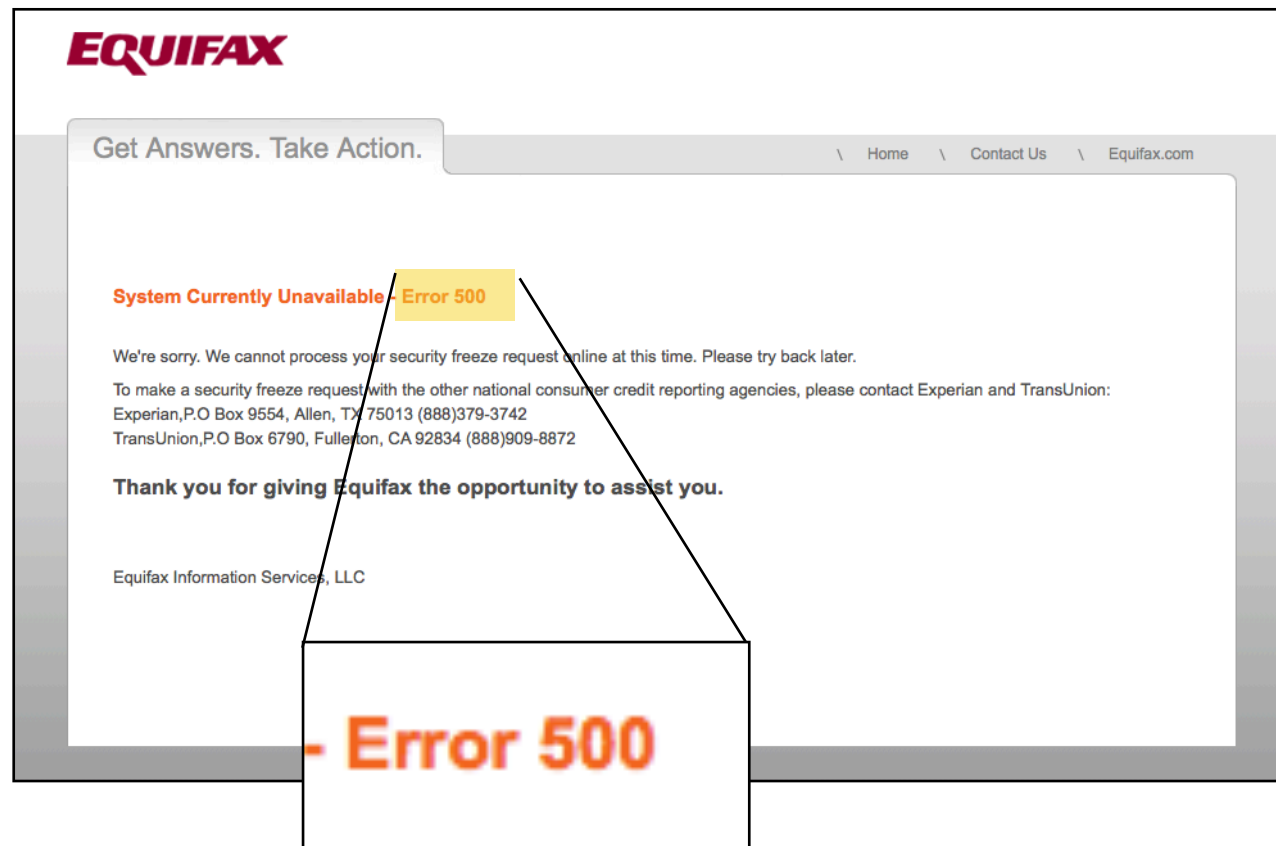
---

## LET'S TAKE A CLOSER LOOK

---



## HTTP STATUS CODES



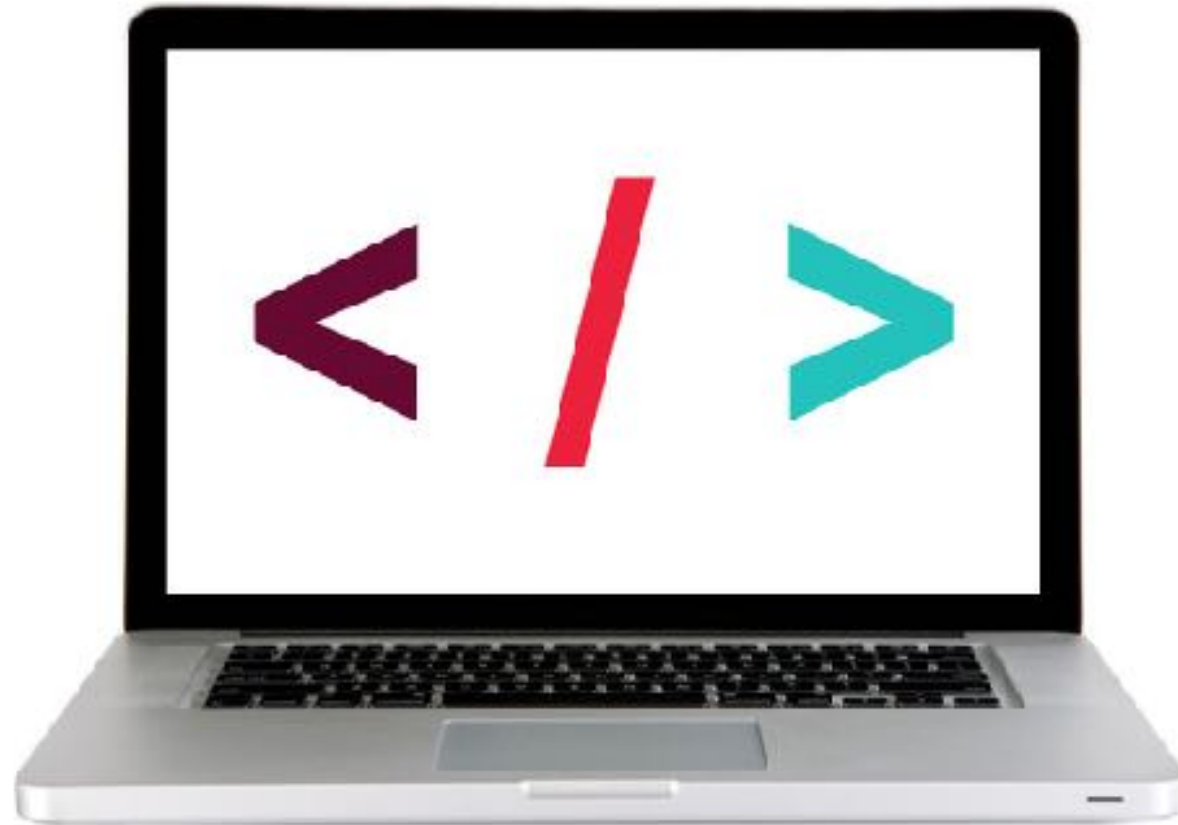
# HTTP STATUS CODES

200	Okay
301	Moved permanently
302	Moved temporarily
400	Bad request
403	Forbidden
404	Not found
500	Internal server error

---

## LET'S TAKE A CLOSER LOOK

---



# **Exit Tickets!**

**(Class #8)**



# **LEARNING OBJECTIVES – REVIEW**

- Use event delegation to manage dynamic content.
- Use implicit iteration to update elements of a jQuery selection
- Identify all the HTTP verbs & their uses.
- Describe APIs and how to make calls and consume API data.
- Access public APIs and get information back.

# **NEXT CLASS PREVIEW**

## **Ajax & APIs**

- Implement an Ajax request with Fetch.
- Create an Ajax request using jQuery.
- Reiterate the benefits of separation of concerns – API vs. Client.

# Q&A