# COMP 431/531: Web Development

## Lecture 7: Modern JavaScript

Mack Joyner (mjoyner@rice.edu)

https://www.clear.rice.edu/comp431



# Announcements & Reminders

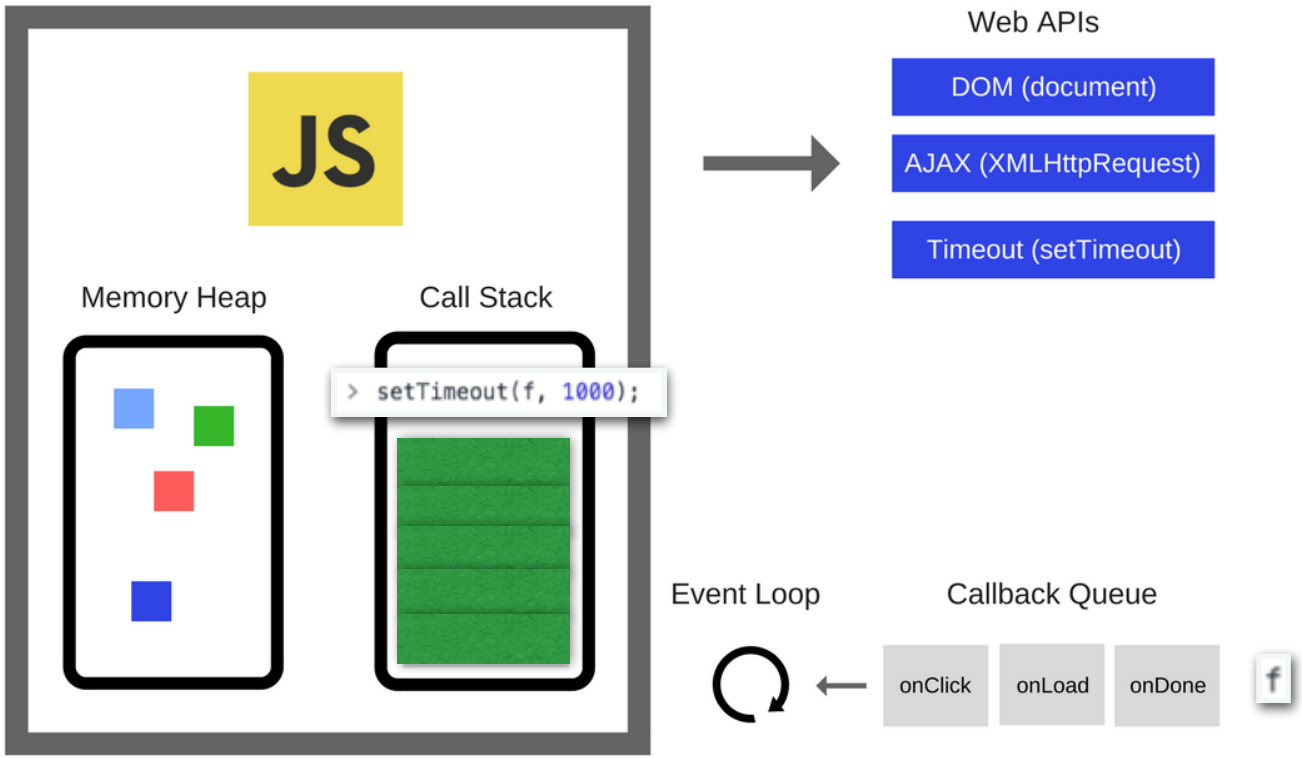
- Quiz #1 (JavaScript) is due Thursday, Sept. 18th at 11:59pm

- HW #3 (JavaScript Game) is due Thursday, Sept. 25th at 11:59pm

  classroom hw3 repo: https://classroom.github.com/a/wZT10Yqv

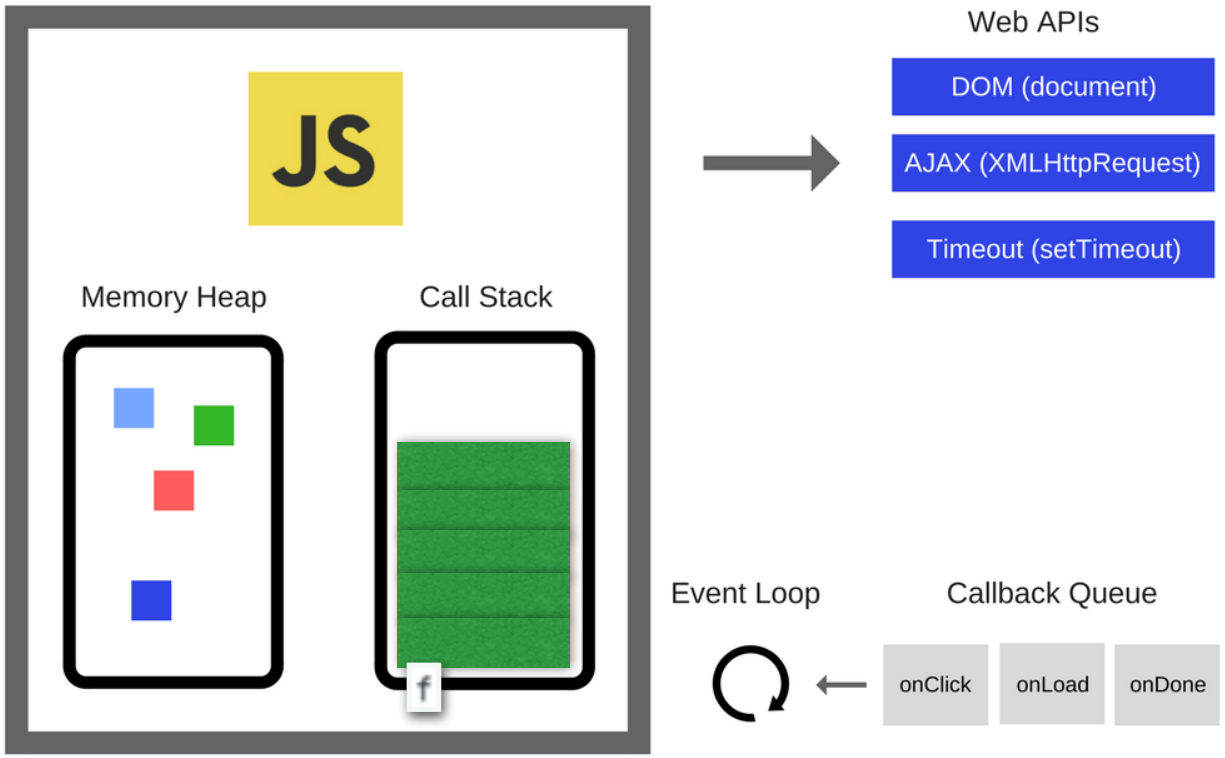


# How does JavaScript work?




Source: https://blog.sessionstack.com/how-does-javascript-actually-work-part-1-b0bacc073cf

# How does JavaScript work?



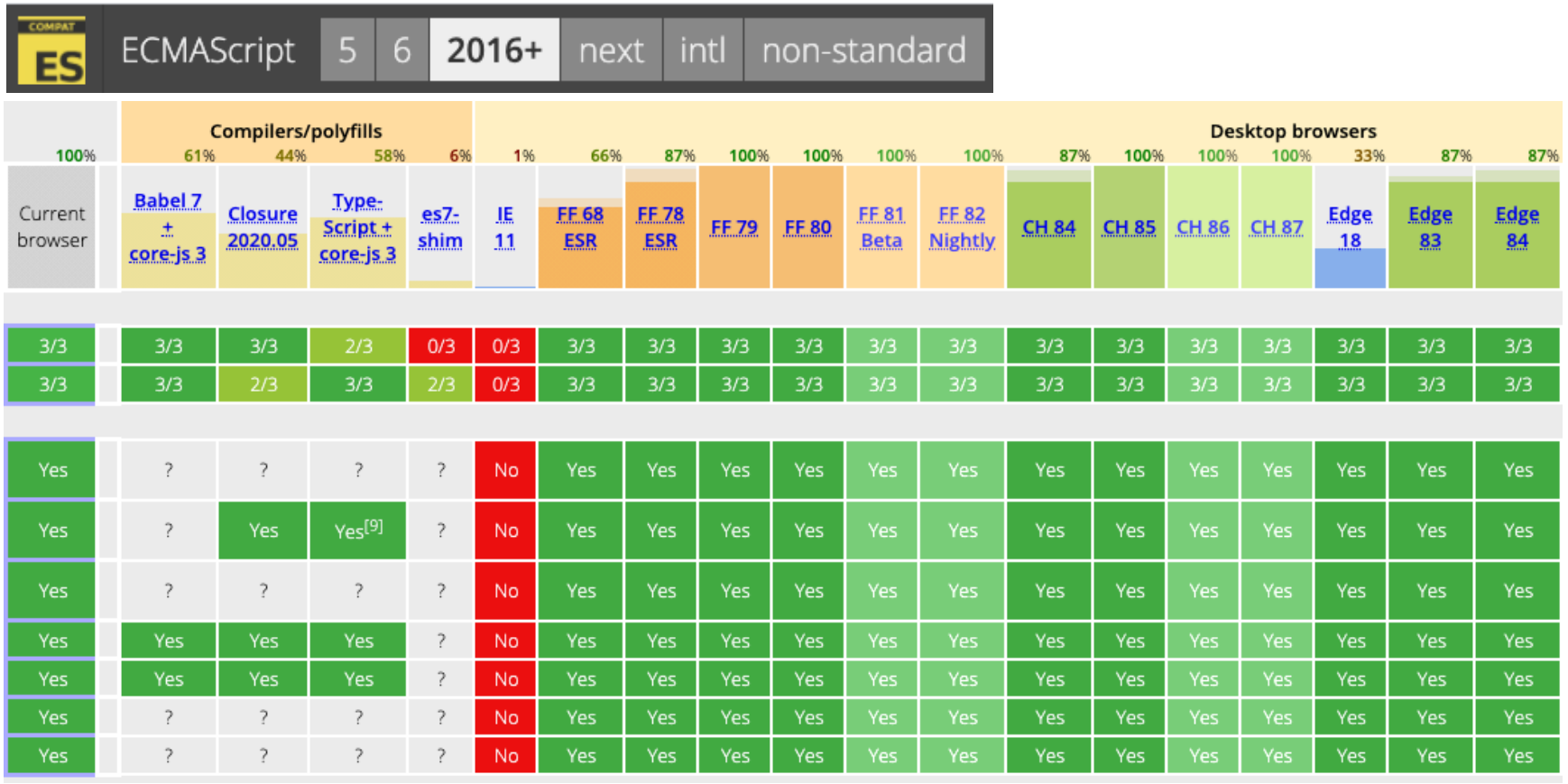

Source: https://blog.sessionstack.com/how-does-javascript-actually-work-part-1-b0bacc073cf

# jQuery Callback vs Chaining

```
$("#div3").click(function() {
    $(this)
        .animate( {opacity: 0}, 2500)
        .animate( {opacity: 1, fontSize: '1em' }, 500 )
        .hide(1000, function() {
            $(this).css({ backgroundColor: "blue" })
        })
        .show(1000)
        .animate( { fontSize: '2em' }, function() {
            $(this).css({ backgroundColor: "green" })
        } )
})
```



# ECMAScript



| ECMAScript | 5 | 6 | 2016+ | next | intl | non-standard |
|---|---|---|---|---|---|---|

| | Compilers/polyfills | | | | Desktop browsers | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 100% | 61% | 44% | 58% | 6% | 1% | 66% | 87% | 100% | 100% | 100% | 100% | 87% | 100% | 100% | 100% | 33% | 87% | 87% |
| Current browser | Babel 7 + core-js 3 | Closure 2020.05 | TypeScript + core-js 3 | es7-shim | IE 11 | FF 68 ESR | FF 78 ESR | FF 79 | FF 80 | FF 81 Beta | FF 82 Nightly | CH 84 | CH 85 | CH 86 | CH 87 | Edge 18 | Edge 83 | Edge 84 |
| 3/3 | 3/3 | 3/3 | 2/3 | 0/3 | 0/3 | 3/3 | 3/3 | 3/3 | 3/3 | 3/3 | 3/3 | 3/3 | 3/3 | 3/3 | 3/3 | 3/3 | 3/3 | 3/3 |
| 3/3 | 3/3 | 2/3 | 3/3 | 2/3 | 0/3 | 3/3 | 3/3 | 3/3 | 3/3 | 3/3 | 3/3 | 3/3 | 3/3 | 3/3 | 3/3 | 3/3 | 3/3 | 3/3 |
| Yes | ? | ? | ? | ? | No | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| Yes | ? | Yes | Yes[9] | ? | No | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| Yes | ? | ? | ? | ? | No | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| Yes | Yes | Yes | Yes | ? | No | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| Yes | Yes | Yes | Yes | ? | No | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| Yes | ? | ? | ? | ? | No | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| Yes | ? | ? | ? | ? | No | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |

https://compat-table.github.io/compat-table/es2016plus/

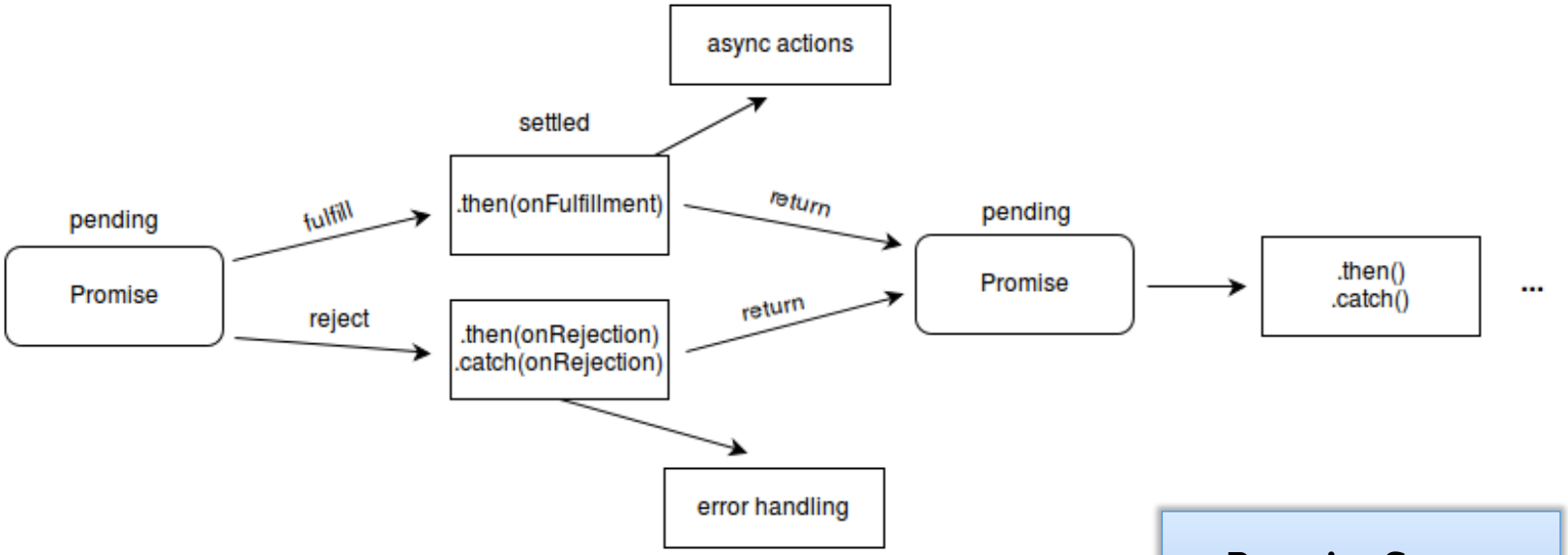


# Recent Evolution of JavaScript

- Jun 1997 - ECMAScript as ECMA-262 specification
- Dec 1999 - ECMAScript 3 = *JavaScript*
- Dec 2009 - ECMAScript 5 = *strict mode*
- Jun 2015 - ECMAScript 6 (aka ES2015) = *Harmony*
  - classes, modules, arrow functions, block scope let & const, promises, generators, …
- Jun 2017 - ES2017
  - concurrency, atomics, async/await
- ….



# Promises are Better than Callbacks




https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

*See also* http://www.html5rocks.com/en/tutorials/es6/promises/

**Promise States**
Pending
Fulfilled / Rejected
Settled

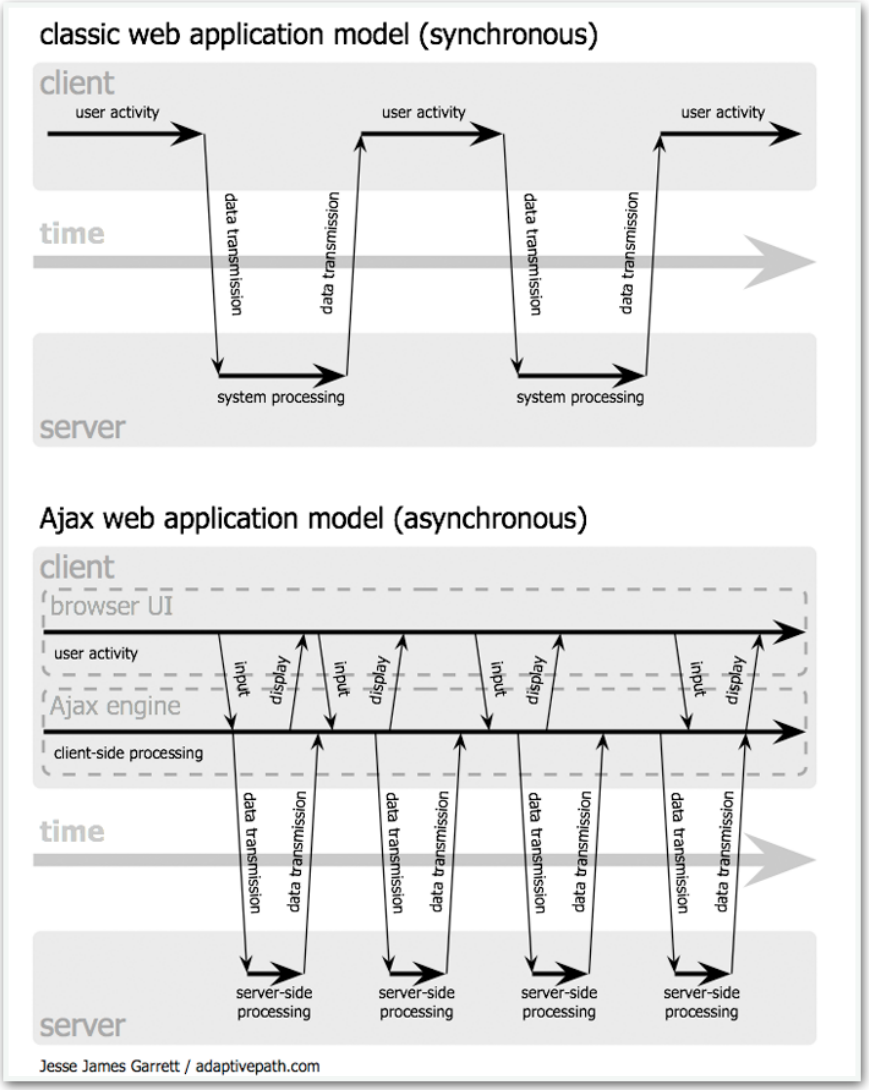


# Communicating with the Server

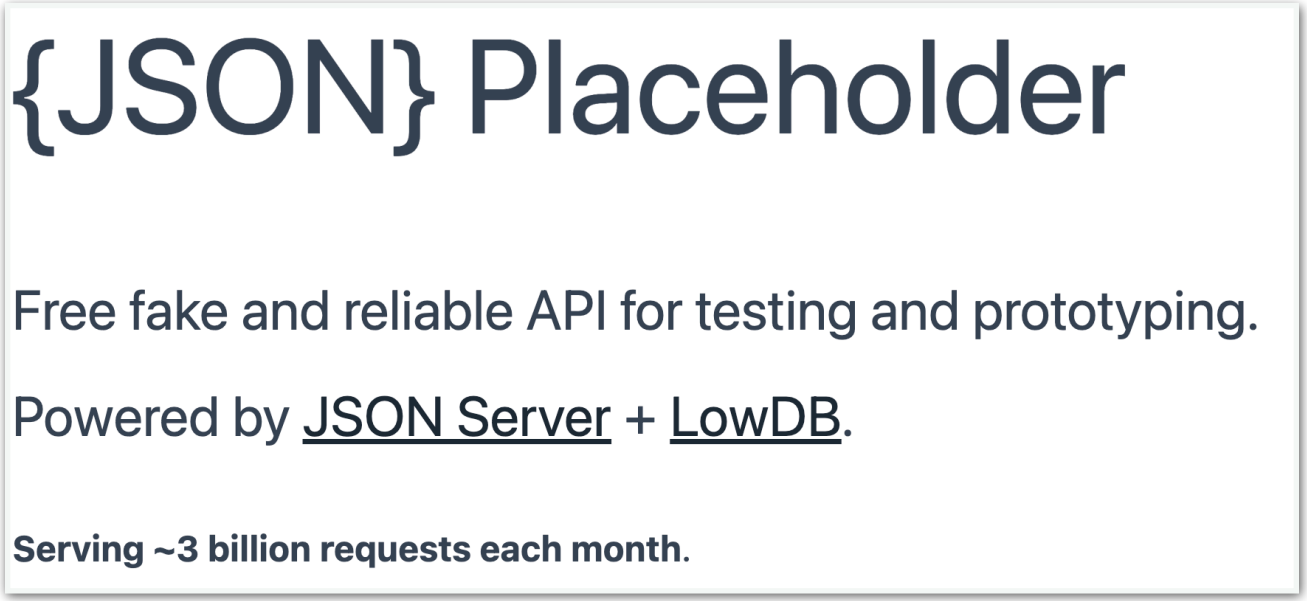
- GET, POST, etc…
- Always instantiated by the browser (i.e. user)
  - using a link
  - clicking on a button (e.g. submitting  a form)
- Convenient to use JavaScript to ask the server for data



# The first “A” in AJAX




# JSONPlaceholder (Server)




https://jsonplaceholder.typicode.com/

# Fetch API

```
> fetch
ƒ fetch() { [native code] }
> fetch('https://jsonplaceholder.typicode.com/posts').then(r => r.json())
Promise {<pending>}
  __proto__: Promise
  [[PromiseStatus]]: "resolved"
  [[PromiseValue]]: Array(100)
    0: {userId: 1, id: 1, title: "sunt aut facere repellat provident
    1: {userId: 1, id: 2, title: "qui est esse", body: "est rerum tem
    2: {userId: 1, id: 3, title: "ea molestias quasi exercitationem r
    3: {userId: 1, id: 4, title: "eum et est occaecati", body: "ullam
    4: {userId: 1, id: 5, title: "nesciunt quas odio", body: "repudiandae venia
    5: {userId: 1, id: 6, title: "dolorem eum magni eos aperiam quia", body: "u
    6: {userId: 1, id: 7, title: "magnam facilis autem", body: "dolore placeat
    7: {userId: 1, id: 8, title: "dolorem dolore est ipsam", body: "dignissimos
    8: {userId: 1, id: 9, title: "nesciunt iure omnis dolorem tempora et accusa
    9: {userId: 1, id: 10, title: "optio molestias id quia eum", body: "quo et
    10: {userId: 2, id: 11, title: "et ea vero quia laudantium autem", body: "c
    11: {userId: 2, id: 12, title: "in quibusdam tempore odit est dolorem", boc
    12: {userId: 2, id: 13, title: "dolorum ut in voluptas mollitia et saepe qu
    13: {userId: 2, id: 14, title: "voluptatem eligendi optio", body: "fuga et
    14: {userId: 2, id: 15, title: "eveniet quod temporibus", body: "reprehende
    15: {userId: 2, id: 16, title: "sint suscipit perspiciatis velit dolorum re
    16: {userId: 2, id: 17, title: "fugit voluptas sed molestias voluptatem prc
    17: {userId: 2, id: 18, title: "voluptate et itaque vero tempora molestiae"
    18: {userId: 2, id: 19, title: "adipisci placeat illum aut reiciendis qui",
    19: {userId: 2, id: 20, title: "doloribus ad provident suscipit at", body:
    20: {userId: 3, id: 21, title: "asperiores ea ipsam voluptatibus modi minim
```

***Inline function callbacks with arrow notation***

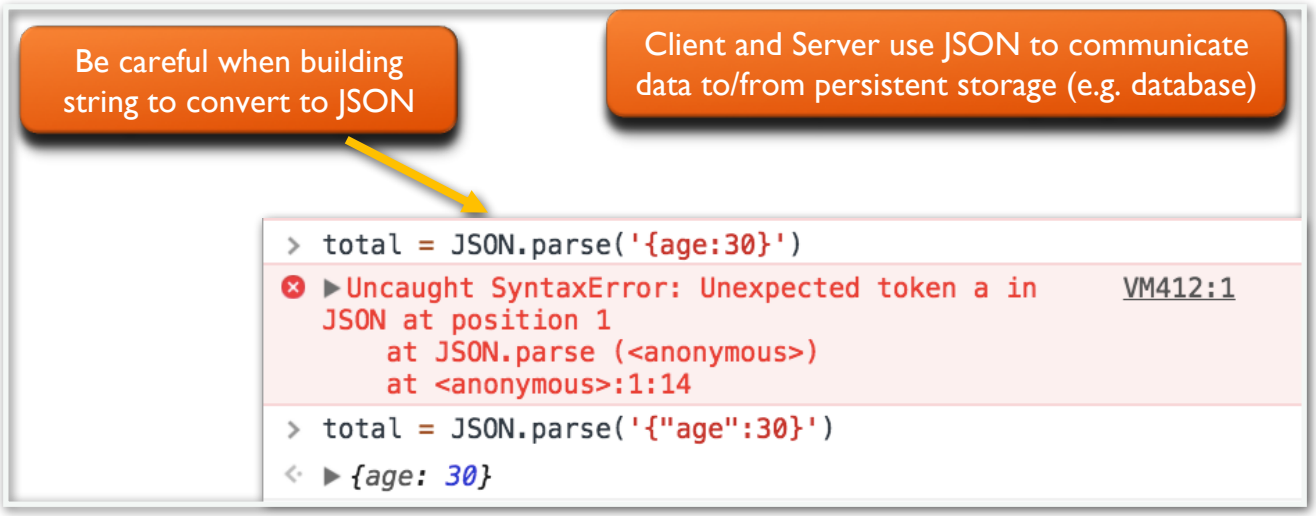
Same as:

```
.then(function(r) {
    return r.json()
})
```

`r.json()` returns a *Promise*, the next `then()` is called when `json()` resolves.



# JavaScript Object Notation (JSON)



# JavaScript Object Notation (JSON)

Object.keys(), Object.values() work for JSON

Error occurs when passing Object.keys() a non-JSON argument

```
function parseJSON(url) {

  //print out the JSON key, value pairs (assuming input is json here!)
  return fetch(url)
         .then(res => res.json())
         .then(res => console('keys: ' + Object.keys(res) + ', values: ' + Object.values(res)));
}
```



# Jasmine

- Jasmine is a test framework for JavaScript
- SpecRunner is the main engine for running tests
- Browser displays test results

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Jasmine Spec Runner v5.10.0</title>

  <link rel="shortcut icon" type="image/png" href="lib/jasmine-5.10.0/jasmine_favicon.png">
  <link rel="stylesheet" href="lib/jasmine-5.10.0/jasmine.css">

  <script src="lib/jasmine-5.10.0/jasmine.js"></script>
  <script src="lib/jasmine-5.10.0/jasmine-html.js"></script>
  <script src="lib/jasmine-5.10.0/boot0.js"></script>
  <!-- optional: include a file here that configures the Jasmine env -->
  <script src="lib/jasmine-5.10.0/boot1.js"></script>

  <!-- include source files here... -->
  <script src="src/Player.js"></script>
  <script src="src/Song.js"></script>

  <!-- include spec files here... -->
  <script src="spec/SpecHelper.js"></script>
  <script src="spec/PlayerSpec.js"></script>
```



# Jasmine Test Structure (spec)

Jasmine test description

Jasmine test to run

Jasmine test assertion

```
6   describe("Jasmine Inclass Fetch Exercises", () => {
7       const baseURL = 'https://jsonplaceholder.typicode.com';
8       const posts = `${baseURL}/posts`;
9
10
11          it('countWords should count the number of words in each post', (done) => {
12            countWords(posts)
13            .then(res => {
14               let wordCounts = Object.values(res);
15               expect(wordCounts[0]).toEqual(20);
16               expect(wordCounts[1]).toEqual(28);
17               expect(wordCounts[2]).toEqual(23);
18               done();
19            });
20          });
```



# Jasmine Source: countWords, getLastLargest

- Jasmine src directory contains JavaScript code
- Implement functionality to pass spec tests

```
// Inclass Fetch Exercise
// ======================

 function countWords(url) {
    return fetch(url)
       .then(res => res.json()).then(res => {
           let postWordCounts = {};

           // TODO: get the word count for each post
           //postWordCounts[postId] = postSize;


          // return an object { postId: wordCount }
          return postWordCounts;
       });
 }

 function getLastLargest(url) {
    return countWords(url)
       .then(postWordCounts => {
           let largestCountSoFar = 0;
           let largestCountPostId = 0;

           // TODO: get all the post id keys

           // TODO: now find the post id with longest post
           return largestCountPostId;
       });
 }
```



# Jasmine Test Failures

Jasmine 3.1.0

××

2 specs, 2 failures, randomized with seed 50617

Spec List | Failures

Jasmine Inclass Fetch Exercises > countWords should count the number of words in each post

Expected 2 to equal 20.

```
Error: Expected 2 to equal 20.
    at <Jasmine>
    at countWords.then.res (file:///Users/mjoyner/comp431/webdev/inclass_exercises/ic8/jasmine/spec/jasmine-inclass-fetch.spec.js:15:36)
```

Expected 4 to equal 28.

```
Error: Expected 4 to equal 28.
    at <Jasmine>
    at countWords.then.res (file:///Users/mjoyner/comp431/webdev/inclass_exercises/ic8/jasmine/spec/jasmine-inclass-fetch.spec.js:16:36)
```

Expected 6 to equal 23.

```
Error: Expected 6 to equal 23.
    at <Jasmine>
    at countWords.then.res (file:///Users/mjoyner/comp431/webdev/inclass_exercises/ic8/jasmine/spec/jasmine-inclass-fetch.spec.js:17:36)
```

Jasmine Inclass Fetch Exercises > getLastLargest should return the id of the last post with the most words

Expected 5 to equal 97.

```
Error: Expected 5 to equal 97.
    at <Jasmine>
    at getLastLargest.then.res (file:///Users/mjoyner/comp431/webdev/inclass_exercises/ic8/jasmine/spec/jasmine-inclass-fetch.spec.js:25:36)
```