



JavaScript *for Geographers*

@jgravois

@patrickarlt

slides: <http://bit.ly/2oSM11A>

Agenda

1. Fundamentals
2. Patterns
3. Put it to use
4. Fatigue

JS can be overwhelming, but you're
more equipped than you think!

declaring variables

```
var dog;  
  
> undefined  
  
// new  
let nifty;  
const notGonnaChange;  
  
> undefined
```

defining values

```
var dogName = 'spot';  
var age = 21;  
var canBark = true;
```

value type is **not** explicitly declared

arithmetic operators

```
(age / 7) // 3
```

```
5 + 5 // 10
```

```
3 - 2 // 1
```

```
3 * 2 // 6
```

arithmetic operators

```
12 % 5 // 2 (modulus)
```

```
var x = 5;  
x++ // 6
```

```
var y = 3;  
y-- // 2
```

```
'foo' + 'bar' // 'foobar'
```

comparison - operators

```
3 === 3 // true
3 === '3' // false

'foo' !== 'bar' // true

3 > 2 // true
3 >= 2 // true
```


logical operators

```
// logical 'and'  
true && anotherTruthy  
> true
```

```
// 'or'  
true || somethingFalsy  
> true
```

```
// 'not'  
!somethingTruthy  
> false
```

arrays

```
var dogs = ['Spot', 'Lassie'];  
  
dogs[0] // 'Spot'  
  
dogs.push('Fido');  
  
dogs.length // 3
```

functions

```
function dogYears(age) {  
  return age * 7;  
}
```

```
dogYears(3);  
> 21
```

anonymous functions

```
function () {  
  return 2*2;  
}
```

arrow functions

```
function (age) {  
  return age*2;  
}  
// ===  
age => {  
  age*2;  
}  
// ===  
age => age*2;
```

objects

```
let dog = {  
  age: 7,  
  canBark: true,  
  _ssshhh: 'top secret'  
}  
  
> Object {age: 7, canBark: true, _ssshhh: 'top secret' }
```

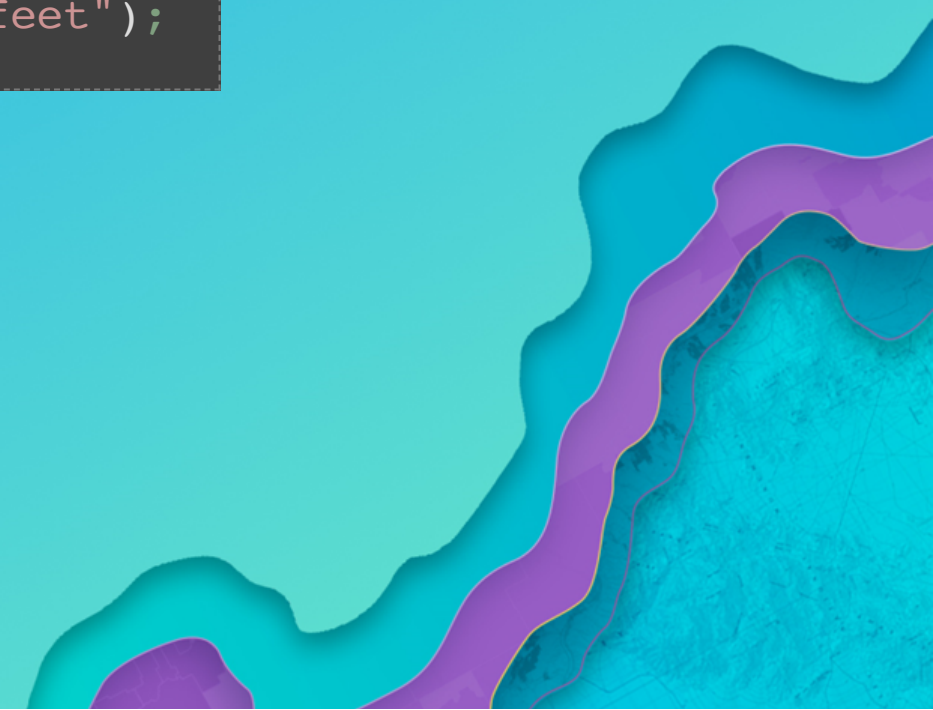
objects

```
<script src="./doglibrary.js"></script>
```

```
let spot = new Dog()  
> Object { canBark: true }  
spot.age = 21;
```

objects - methods

```
// Buffer point by 1000 feet  
var ptBuff = geometryEngine.buffer(point, 1000, "feet");
```



for loops

```
for (var i = 0; i < 6; i++) {  
  console.log(i);  
}
```

```
> 0  
> 1  
> 2  
> 3  
> 4  
> 5
```

looping through an array

```
var dogs = ['Spot', 'Lassie', 'Fido'];  
  
for (var i = 0; i < dogs.length; i++) {  
    console.log(dogs[i]);  
}  
  
> 'Spot'  
> 'Lassie'  
> 'Fido'
```

JavaScript is *Asynchronous*

- JavaScript is *single threaded*
- Only does 1 thing at a time
- Lots of things might happen at once
- This is the "Event Loop"

JavaScript Event Loop

1. Executes one function at a time
2. Run the entire function
3. Start the next function

Demo

Callbacks

```
<button id="button">Click Me!</button>
```

```
let button = document.getElementById('button');  
button.addEventListener('click', function () {  
  console.log('The button was clicked');  
});
```

Callback are functions that are called when things happen.

Demo

Promises

```
let user = fetch('https://randomuser.me/api/')
  .then(processResponse)
  .then(doSomethingWithUser)
  .catch(anyErrors);

function processResponse (response) {
  return response.json();
}

function doSomethingWithUser (user) {
  console.log(user); // prints a bunch of user info
}

function anyErrors (error) {
  console.error('what have you done!', error);
}
```

Promises represent a future value that will be "resolved".

Demo

Function Scope

```
var message = 'Hello World!';  
  
function go () {  
  console.log(message);  
}  
  
go();
```

When functions are called they remember the variables around them, this is referred to as "lexical scope".

What is `this`?

```
var user = {
  firstName: "Casey",
  lastName: "Jones",
  fullName: function () {
    console.log(this.firstName + " " + this.lastName);
  }
}

person.fullName() // > Casey Jones
```


What is `this`?

The value of `this` depends on how the function was called.

Demo



lets set up a JS development environment

- do i have a web server running?
- `demo.html`

debugging

Get familiar with your dev tools!

- `console.log` - print things to the console
- `debugger` - stops the application so you can look around

the DOM

- select elements
- listen for events
- change elements

A simple form; Finished example;

sharing JavaScript - modules

As applications grow it is helpful to divide code into different files to organize.

You can just use `<script>` tags for small apps.

ES2015 Modules

```
import { something } from 'some-module';
```

This is the future as you learn JavaScript you will encounter this more often.

AMD Modules (JS API)

```
require([
  "esri/Map",
  "esri/views/MapView",
], function (Map, MapView) {
  // Map and MapView have been loaded!
});
```

`require` is a fancy way of adding `<script>` tags to load code on demand.

lets put all this to use!

- [../sample-code/tasks-query/](#)
- [../sample-code/chaining-promises/](#)

The JavaScript Language

Updates every year (ES2015, ES2016, ES2017).

2015 had LOADS of new features.

2017 had ~2 new features.

the JavaScript ecosystem

- Module Formats - CommonJS, AMD, ES 2015
- Compilers - Babel, TypeScript
- Bundlers - Rollup, WebPack, SystemJS
- Minifiers - UglifyJS

the JavaScript ecosystem

You don't know what you don't know.

and that is great.

JavaScript Fatigue

Look, it's easy. Code everything in Typescript. All modules that use Fetch compile them to target ES6, transpile them with Babel on a stage-3 preset, and load them with SystemJS. If you don't have Fetch, polyfill it, or use Bluebird, Request or Axios, and handle all your promises with await.

We have very different definitions of easy.

How it feels to learn JavaScript in 2016

Fight JavaScript Fatigue

- The JS API is MORE than enough for simple apps
- Add tools when you **KNOW** you will benefit from using them
- Too many tools === Lots of complexity to manage

learn more!

- You Don't Know JS
- MDN: Learn web development
- MDN
- Eloquent JavaScript
- <http://wesbos.com/>

please, *please*, **please** fill out a session survey

1. Download the Esri Events App
2. Select Dev Summit
3. Search for "JavaScript for Geographers"
4. Leave feedback!

idea, question, issue, or success story?

@geogangster / @patrickarlt

Slides: <http://bit.ly/2oSM11A>



esri

THE
SCIENCE
OF
WHERE