

# Javascript, The Swiss Army Knife of Programming Languages

David Morcillo

30-11-2013

# Previously on JS Workshop. . .

Introduction to JS Hello World and Syntax

Good parts Objects, Functions, Inheritance and Arrays

Node.js Javascript platform and npm for back-end dependencies

Bower front-end dependencies

Grunt Javascript task runner

Basic HTML5 Canvas and requestAnimationFrame

# Git cheatsheet

`git init` Initialize git repository.

`git add .` Add all changes to stage.

`git commit -am` Commit changes

`git checkout <commit>` Checkout code to specific commit.

`git diff` Show changes between workspace and last commit

`git status -sb` Show current status of workspace and stage

`git log` Show history



# Problem

## Including scripts

```
<html>
  <head>
    <script src='js/game.js'></script>
    <script src='js/character.js'></script>
    <script src='js/player.js'></script>
    <script src='js/enemy.js'></script>
    <script src='js/knight.js'></script>
    <script src='js/soldier.js'></script>
    <script src='js/protector.js'></script>
    ...
```

# Problem

## Including scripts

```
<html>
  <head>
    <script src='js/game.js'></script>
    <script src='js/character.js'></script>
    <script src='js/player.js'></script>
    <script src='js/enemy.js'></script>
    <script src='js/knight.js'></script>
    <script src='js/soldier.js'></script>
    <script src='js/protector.js'></script>
    ...
```

- We need to remember the inclusion order
- Each module, function or object must be accessible through global scope if we want to use it as a dependency.

# Possible solution

## index.html

```
<html>
<head>
  <script src='js/built.js'></script>
  ...
```

## Gruntfile.js

```
...
concat: {
  options: {
    separator: ';',
  },
  dist: {
    src: [
      'js/game.js',
      'js/character.js',
      'js/player.js',
      'js/enemy.js',
      'js/knight.js',
      'js/soldier.js',
      'js/protector.js'
    ],
  },
},
...
```

# Require.js

## A javascript module loader

RequireJS is a JavaScript file and module loader. It is optimized for in-browser use, but it can be used in other JavaScript environments, like Rhino and Node. Using a modular script loader like RequireJS will improve the speed and quality of your code.





# Require.js: an example

## Without Require.js

```
var MYGAME = MYGAME || {},
    game    = MYGAME.game,
    entity  = MYGAME.entity;

MYGAME.crate = function (spec) {
  // code omitted
};
```

## With Require.js

```
define(function (require) {
  var game    = require('game'),
      entity  = require('entity'),
      crate;

  crate = function (spec) {
  };

  return crate;
});
```

# Require.js: getting started

## Get Require.js

Use `bower` to install it as a dependency of your project

# Require.js: getting started

## Get Require.js

Use `bower` to install it as a dependency of your project

## Include it

```
<html>
  <head>
    <script data-main='scripts/main' src='bower_components/requirejs/require.js'></script>
    ...
```

# Require.js: getting started

## Get Require.js

Use `bower` to install it as a dependency of your project

## Include it

```
<html>
  <head>
    <script data-main='scripts/main' src='bower_components/requirejs/require.js'></script>
    ...
```

## Define modules

```
define(function (require) {
  // code omitted
});
```

# Require.js: Lab

## Exercise

- `git checkout stage_6`
- Install your back-end dependencies with `npm install`
- Install your front-end dependencies with `bower install`
- Start `grunt watch` for auto linting
- Install Require.js and include the main entry point
- Refactor modules and functions using Require.js modules.



# Event Loop

## Event Loop

JavaScript has a concurrency model based on an “event loop”. This model is quite different than the model in other languages like C or Java.

```
while(queue.waitForMessage()) {  
  queue.processNextMessage();  
}
```

# Event Loop

## Event Loop

JavaScript has a concurrency model based on an “event loop”. This model is quite different than the model in other languages like C or Java.

```
while(queue.waitForMessage()) {  
  queue.processNextMessage();  
}
```

A very interesting property of the event loop model is that JavaScript, unlike a lot of other languages, never blocks.



# Event Loop

## Example

```
var now = +new Date();

setTimeout(function () {
  var dt = (+new Date()) - now;
  console.log('First timeout');
  console.log('Elapsed time: ' + dt + ' milliseconds');
}, 500);

setTimeout(function () {
  var dt = (+new Date()) - now,
      i = 0;
  console.log('Second timeout');
  console.log('Elapsed time: ' + dt + ' milliseconds');
  while(i < 10000000000) {
    i += 1;
  }
}, 250);
```

# Event Loop

## Example

```
var now = +new Date();

setTimeout(function () {
  var dt = (+new Date()) - now;
  console.log('First timeout');
  console.log('Elapsed time: ' + dt + ' milliseconds');
}, 500);

setTimeout(function () {
  var dt = (+new Date()) - now,
      i = 0;
  console.log('Second timeout');
  console.log('Elapsed time: ' + dt + ' milliseconds');
  while(i < 10000000000) {
    i += 1;
  }
}, 250);
```

## Output

```
Second timeout
Elapsed time: 252 milliseconds
First timeout
Elapsed time: 1271 milliseconds
```

# DOM Events

When an event happens, the browser sends the event to the related element. If you've set a handler (a function) on that element, it gets called with related event info which means you 'handled' the event.

# DOM Events

When an event happens, the browser sends the event to the related element. If you've set a handler (a function) on that element, it gets called with related event info which means you 'handled' the event.

## The old way

```
var blueSoldierSeat = document.getElementById('blue-soldier-seat');
blueSoldierSeat.onclick = function (event) {
    // code omitted
};
```

# DOM Events

When an event happens, the browser sends the event to the related element. If you've set a handler (a function) on that element, it gets called with related event info which means you 'handled' the event.

## The old way

```
var blueSoldierSeat = document.getElementById('blue-soldier-seat');
blueSoldierSeat.onclick = function (event) {
    // code omitted
};
```

## The classy way

```
var blueSoldierSeat = document.getElementById('blue-soldier-seat');
blueSoldierSeat.addEventListener('click', function (event) {
    // code omitted
});
```

# DOM Events

## The jQuery way #1

```
var $blueSoldierSeat = $('#blue-soldier-seat');  
$blueSoldierSeat.click(function (event) {  
    // code omitted  
});
```

# DOM Events

## The jQuery way #1

```
var $blueSoldierSeat = $('#blue-soldier-seat');  
$blueSoldierSeat.click(function (event) {  
    // code omitted  
});
```

## The jQuery way #2

```
var $blueSoldierSeat = $('#blue-soldier-seat');  
$blueSoldierSeat.on('click', function (event) {  
    // code omitted  
});
```

# Custom Events

We can use a library or implement our own functions for listening and triggering custom events.



# Custom Events

We can use a library or implement our own functions for listening and triggering custom events.

With jQuery on and trigger

```
var tree = {  
  apples: 10  
};  
  
$(document).on('apple fall', function (event) {  
  tree.apples -= 1;  
});  
  
$(document).trigger('apple fall');  
  
console.log(tree.apples); // Output: 9
```

# Custom Events

## With library IndigoUnited/events-emitter

```
// Require.js code omitted
var EventEmitter = require('events-emitter/EventsEmitter'),
    emitter      = new EventEmitter(),
    tree         = {
        apples: 10
    };

emitter.on('apple fall', function (event) {
    tree.apples -= 1;
});

emitter.emit('apple fall');

console.log(tree.apples); // Output: 9
```