

# Javascript, The Swiss Army Knife of Programming Languages

David Morcillo

23-11-2013

# About me



- [twitter.com/ultrayoshi](https://twitter.com/ultrayoshi)
- [github.com/ultrayoshi](https://github.com/ultrayoshi)

# 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

# Follow this workshop

## Slides

```
$ git clone git://github.com/ultrayoshi/js-workshop-slides
```

## Code

```
$ git clone git://github.com/ultrayoshi/js-workshop-code
```



# Features

- Loosely typed language

# Features

- Loosely typed language
- Object literal notation

# Features

- Loosely typed language
- Object literal notation
- Prototypal inheritance

# Features

- Loosely typed language
- Object literal notation
- Prototypal inheritance
- Global variables

# Features

- Loosely typed language
- Object literal notation
- Prototypal inheritance
- Global variables
- Functions are first class objects

# Features

## ECMAScript

The standard that defines JavaScript is the third edition of *ECMAScript Programming Language*.

# Features

## ECMAScript

The standard that defines JavaScript is the third edition of *ECMAScript Programming Language*.

ES5 is the standard one nowadays:

<http://kangax.github.io/es5-compat-table/>

# Features

## ECMAScript

The standard that defines JavaScript is the third edition of *ECMAScript Programming Language*.

ES5 is the standard one nowadays:

<http://kangax.github.io/es5-compat-table/>

ES6 is still on development:

<http://kangax.github.io/es5-compat-table/es6/>

# Hello World

## index.html

```
<html>
  <head>
    <script>
      document.writeln("Hello, world!");
    </script>
  </head>
  <body>
    </body>
  </html>
```

# Syntax

## Comments

Block comments formed with `/* */` and line-ending comments starting with `//`. Example:

```
/*
  We are learning Javascript and comments are very important
*/
document.writeln('Hello World!'); // Output: Hello World!
```

# Syntax

## Comments

Block comments formed with `/* */` and line-ending comments starting with `//`. Example:

```
/*
  We are learning Javascript and comments are very important
*/
document.writeln('Hello World!'); // Output: Hello World!
```

## Names

Starts with a letter or underscore and optionally followed by one or more letters, digits or underscores. Beware of some reserved words.

```
bullet          // valid      _mana          // valid
3force         // invalid    lucky42         // valid
rocket-launcher // invalid    grenade_launcher // valid
```

# Syntax

## Numbers

Single number type represented internally as 64-bit floating point.

```
42  
3.141516  
10e5  
1/0 // Output: Infinity  
0/0 // Output: NaN
```

# Syntax

## Numbers

Single number type represented internally as 64-bit floating point.

```
42  
3.141516  
10e5  
1/0 // Output: Infinity  
0/0 // Output: NaN
```

## Strings

Can be wrapped in single quotes or double quotes. It can contains 0 or more characters. All characters in Javascript are 16 bits wide.

```
“Hello World”  
'Hello World'  
“This is\n a multiline string”  
'You can write “ on single quotes string'
```

# Syntax

## Functions

```
function helloWorld (name) {  
    console.log('Hello ' + name + '!');  
}  
  
helloWorld('David'); // Output 'Hello David!'  
  
var myFunction = function () {  
    console.log('Hi there!');  
};  
  
myFunction(); // Output: 'Hi there!'
```

# Syntax

## Variables

Use the `var` keyword followed by a name to declare a variable. When used inside of a function, the `var` statement defines the function's private variables.

```
var player; // variable player declared on a global scope

function test() {
    var enemy; // Scoped to function test
}
```

# Syntax

## Strict (in)equality

```
10 == '10' // Output: true, auto type coercion
10 === '10' // Output: false strict equality
10 != '10' // Output: false, auto type coercion
10 !== '10' // Output: true strict inequality
```

# Syntax

## Strict (in)equality

```
10 == '10' // Output: true, auto type coercion
10 === '10' // Output: false strict equality
10 != '10' // Output: false, auto type coercion
10 !== '10' // Output: true strict inequality
```

## null and undefined

```
console.log(mario); // Error: mario is not defined

function exists (mario) {
  console.log(mario);
}

exists(); // Output undefined

console.log(null == undefined) // Output: true
console.log(null === undefined) // Output: false
```

# Syntax

## if, else

```
var testOk = true;

if (testOk) {
  console.log("Captain obvious");
} else {
  console.log("I'm bored");
}
```

Here are the *falsy* values:

- false
- null
- undefined
- The empty string
- The number 0
- The number NaN

All other values are *truthy*.

# Syntax

## switch

```
var weapon = 'rocketlauncher';

switch(weapon) {
  case 'pistol':
    console.log('piu piu');
    break;
  case 'shotgun':
    console.log('paaam!');
    break;
  case 'rocketlauncher':
    console.log('BOOOOM!');
    break;
  default:
    console.log('falcon punch!');
    break;
}
```

# Syntax

## while, do while

```
var counter = 0;
while (counter < 10) { // Ends when counter is equal to 10
    console.log(counter);
    counter += 1;
}

do {
    console.log(counter);
    i -= 1;
} while(counter > 0); // Ends when counter is equal to 0
```

# Syntax

## while, do while

```
var counter = 0;
while (counter < 10) { // Ends when counter is equal to 10
    console.log(counter);
    counter += 1;
}

do {
    console.log(counter);
    i -= 1;
} while(counter > 0); // Ends when counter is equal to 0
```

## for

```
var i;

for (i = 0; i < 10; i += 1)
    console.log(i);
}
```



# Objects

- Objects in Javascript are mutable keyed collections.

# Objects

- Objects in Javascript are mutable keyed collections.
- Arrays, functions and regular expressions are objects.

# Objects

- Objects in Javascript are mutable keyed collections.
- Arrays, functions and regular expressions are objects.
- A property name can be any string.

# Objects

- Objects in Javascript are mutable keyed collections.
- Arrays, functions and regular expressions are objects.
- A property name can be any string.
- Objects can inherit properties of another through its prototype.

# Objects

- Objects in Javascript are mutable keyed collections.
- Arrays, functions and regular expressions are objects.
- A property name can be any string.
- Objects can inherit properties of another through its prototype.

## Prototype

All objects created from object literals are linked to `Object.prototype`.

# Objects

- Objects in Javascript are mutable keyed collections.
- Arrays, functions and regular expressions are objects.
- A property name can be any string.
- Objects can inherit properties of another through its prototype.

## Prototype

All objects created from object literals are linked to `Object.prototype`. If we try to retrieve a property value from an object, and if the object lacks the property name, then Javascript attempts to retrieve the property value from the prototype object.

# Objects

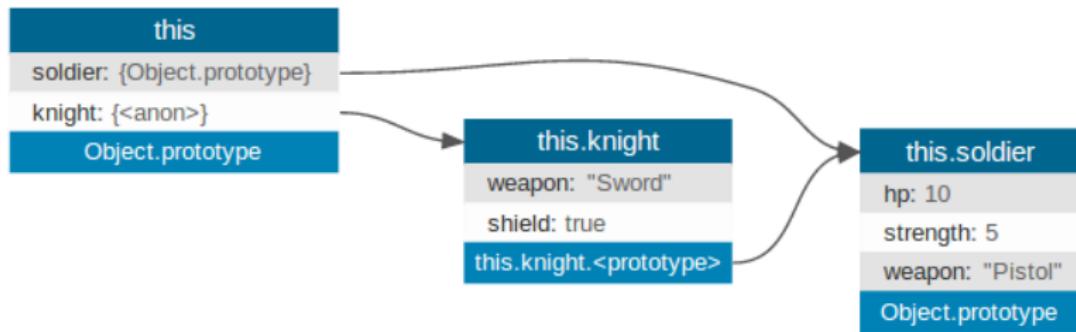
## Object.create

```
var soldier = {  
    hp: 10,  
    strength: 5,  
    weapon: 'Pistol'  
};  
  
var knight = Object.create(soldier);  
knight.weapon = 'Sword';  
knight.shield = true;  
  
console.log(knight.hp); // Output: 10  
console.log(knight['weapon']); // Output: 'Sword'  
console.log(knight.shield); // Output: true
```

---

Visit <http://www.objectplayground.com/> for a graphical explanation

# Objects



# Objects

## hasOwnProperty

```
console.log(knight.hasOwnProperty('hp')); // Output: false
console.log(knight.hasOwnProperty('shield')); // Output: true
```

# Objects

## hasOwnProperty

```
console.log(knight.hasOwnProperty('hp')) // Output: false
console.log(knight.hasOwnProperty('shield')) // Output: true
```

## for in

```
for (attr in knight) {
    if(knight.hasOwnProperty(attr)) {
        console.log('Knight property ' + attr + ' with value ' +
                    knight[attr]);
    }
}
// Knight property weapon with value 'Sword'
// Knight property shield with value true
```

# Objects

## hasOwnProperty

```
console.log(knight.hasOwnProperty('hp')) // Output: false
console.log(knight.hasOwnProperty('shield')) // Output: true
```

## for in

```
for (attr in knight) {
    if(knight.hasOwnProperty(attr)) {
        console.log('Knight property ' + attr + ' with value ' +
                    knight[attr]);
    }
}
// Knight property weapon with value 'Sword'
// Knight property shield with value true
```

## delete

```
console.log(knight.weapon) // Output: 'Sword'
delete knight.weapon;
console.log(knight.weapon) // Output: 'Pistol'
```

# Functions

Functions are the **fundamental modular unit** of Javascript. They are used for code reuse, information hiding, and composition.

The thing that is special about functions is that they can be invoked.

# Functions

Functions are the **fundamental modular unit** of Javascript. They are used for code reuse, information hiding, and composition.

The thing that is special about functions is that they can be invoked.

## Function.prototype

Functions are objects linked to Function.prototype.

# Functions



# Functions

Invoking a function suspends the execution of the current function, passing control and parameters to the new function. In addition to the declared parameters, every function receives two additional parameters: this and arguments.

# Functions

Invoking a function suspends the execution of the current function, passing control and parameters to the new function. In addition to the declared parameters, every function receives two additional parameters: `this` and `arguments`.

## `this`

Receive the value of the object invoking the function or the global object if it's called without an object.

# Functions

## Invocation (1/4): Method invocation pattern

```
var enemy = {  
  hp: 5,  
  rage: 0,  
  attack: function () {  
    this.rage += 1;  
  }  
};  
  
enemy.attack();  
console.log(enemy.rage); // Output: 1
```

# Functions

## Invocation (2/4): Function invocation pattern

```
physicsManager.collisionsDetected = 0;
physicsManager.checkCollision = function (entity1, entity2) {
    var bbCollision = function (bb1, bb2) {
        // Collision code omitted
        var collision = true;
        if (collision) {
            // WARNING: 'this' is the global object and not 'physicsManager'
            this.collisionsDetected += 1;
        }
        return collision;
    };

    bbCollision(entity1.getBB(), entity2.getBB());
};

if (physicsManager.checkCollision(enemy, player)) {
    player.takeDamage(enemy.strength);
}

console.log(physicsManager.collisionsDetected); // Output: 0
```

# Functions

## Invocation (2/4): Function invocation pattern (workaround)

```
physicsManager.collisionsDetected = 0;
physicsManager.checkCollision = function (entity1, entity2) {
    var that = this;

    var bbCollision = function (bb1, bb2) {
        // Collision code omitted
        var collision = true;
        if (collision) {
            that.collisionsDetected += 1;
        }
        return collision;
    };

    bbCollision(entity1.getBB(), entity2.getBB());
};

if (physicsManager.checkCollision(enemy, player)) {
    player.takeDamage(enemy.strength);
}

console.log(physicsManager.collisionsDetected); // Output: 1
```

# Functions

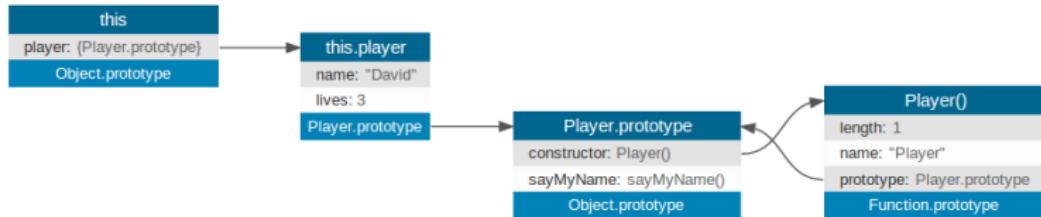
## Invocation (3/4): Constructor invocation pattern

```
var Player = function (name) {
    this.name = name;
    this.lives = 3;
};

Player.prototype.sayMyName = function () {
    console.log('My name is ' + this.name);
};

var david = new Player('David');
david.sayMyName(); // Output: 'My name is David'
```

# Functions



# Functions

## Invocation (3/4): Constructor invocation pattern (without new)

```
var Player = function (name) {
    this.name = name;
    this.lives = 3;
};

Player.prototype.sayMyName = function () {
    console.log('My name is ' + this.name);
};

var manfred = Player('Manfred'); // oops
try {
    manfred.sayMyName(); // raise an error because manfred is undefined
} catch (e) {
    console.log('[' + e.name + '] ' + e.message);
}

// Global variables feast
console.log(name); // Output: 'Manfred'
console.log(lives); // Output: 3
```

# Functions

## Invocation (4/4): Apply invocation pattern

```
var enemy = {  
    rage: 0,  
    attack: function () {  
        this.rage += 1;  
    }  
};  
  
var anotherEnemy = {  
    rage: 10  
};  
  
enemy.attack.apply(anotherEnemy, []);  
  
console.log(anotherEnemy.rage); // Output: 11
```

# Functions

## Arguments

```
function doActions() {  
    var i, l;  
  
    // WARNING: arguments is an Array-like object  
    for (i = 0, l = arguments.length; i < l; i += 1) {  
        console.log('Doing action ' + arguments[i]);  
    }  
  
    doActions('jump', 'attack');  
/*  
 * Output:  
 * 'Doing action jump'  
 * 'Doing action attack'  
 */
```

# Functions

## Closure

Javascript does have function scope. That means that the parameters and variables defined in a function are not visible outside of the function, and that a variable defined anywhere within a function is visible everywhere within the function.

```
var player = new Player();

function isGameOver() {
    var enemy = new Enemy();

    function checkHit() {
        return enemy.hit(player);
    }

    return checkHit();
}

isGameOver();
```

# Functions

## Immediately-Invoked Function Expression (IIFE)

```
var something = (function () {  
    // some code  
}());
```

# Functions

## Module pattern

```
var MYAPP = MYAPP || {};
(function (exports) {
    var detectedCollisions = 0;

    function checkBBCollision(bb1, bb2) {
        // some code
    }

    function checkCollision(entity1, entity2) {
        checkBBCollision(entity1.getBB(), entity2.getBB());
    }

    exports.physicsModule = {
        checkCollision: checkCollision
    };
})(MYAPP);
```

# Inheritance

Javascript provides a much richer set of code reuse patterns. It can ape the classical pattern, but it also supports other patterns that are more expressive.

# Inheritance

Javascript provides a much richer set of code reuse patterns. It can ape the classical pattern, but it also supports other patterns that are more expressive.

## Javascript is a class-free language

In classical languages, objects are instances of classes, and a class can inherit from another class. Javascript is a prototypal language, which means that objects inherit directly from other objects.

# Inheritance

## Pseudoclassical pattern

```
var Alien = function (name) {
  this.name = name;
};

Alien.prototype.talk = function () {
  console.log('>?saf? ' + this.name);
};

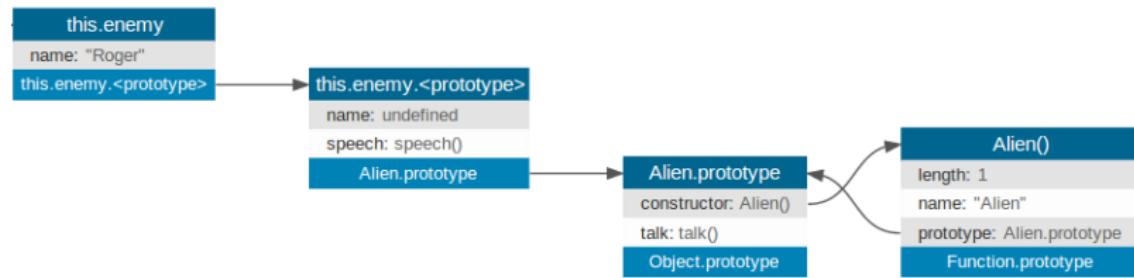
var SmartAlien = function (name) {
  this.name = name;
};

SmartAlien.prototype = new Alien();

SmartAlien.prototype.speech = function () {
  this.talk();
  console.log('...I mean, my name is ' + this.name);
};

var enemy = new SmartAlien('Roger');
enemy.speech();
// Output: '?saf? Roger
//           ...I mean, my name is Roger'
```

# Functions



# Inheritance

## Prototypal pattern

```
var alien = {  
    name: '%?&789',  
    talk: function () {  
        console.log('%?saf? ' + this.name);  
    }  
};  
  
var smartAlien = Object.create(alien);  
smartAlien.speech = function () {  
    this.talk();  
    console.log('...I mean, my name is ' + this.name);  
};  
  
var enemy = Object.create(smartAlien);  
enemy.name = 'Roger';  
enemy.speech();  
// Output: '%?saf? Roger  
//           ...I mean, my name is Roger'
```

# Inheritance

## Functional pattern

```
var alien = function (spec) {
  var that = {};

  var killHumans = function () { // Private access
    console.log('*Using ' + spec.weapon + '*');
  };

  that.talk = function () {
    console.log('%&78 ' + spec.name);
    if (spec.weapon) {
      killHumans();
    }
  };

  return that;
};

var enemy = smartAlien({ name: 'Roger' });
enemy.speech();
// Output: ??saf? Roger
//          *Using Pistol*
//          ...I mean, my name is Roger'
```

```
var smartAlien = function (spec) {
  spec.weapon = 'Pistol'; // Private access
  var that = alien(spec);

  that.speech = function () {
    that.talk();
    console.log('...I mean, my name is ' +
      spec.name);
  };
  return that;
};
```

# Arrays

Arrays doesn't exist

Javascript provides an object that has some array-like characteristics. It converts array subscripts into strings that are used to make properties.

# Arrays

## Arrays doesn't exist

Javascript provides an object that has some array-like characteristics. It converts array subscripts into strings that are used to make properties.

## Arrays literals

```
var enemies = [];  
  
console.log(enemies[9999]); // Output: undefined  
  
enemies[0] = 'Sigma';  
console.log(enemies[0]); // Output: 'Sigma'  
  
enemies[1] = 9000; // We can mix different types  
console.log(enemies[1]); // Output: 9000
```

# Arrays

## Remove elements

```
var enemies = ['Grassman', 'Bowser', 'Sephirot'],  
    players = ['David', 'Manfred', 'Joanmi'];  
  
delete enemies[1]; // Bad idea  
console.log(enemies[1]); // Output: undefined  
console.log(enemies.length); // Output: 3  
  
players.splice(1, 1); // Yeah!  
console.log(players[1]); // Output: 'Joanmi'  
console.log(players.length); // Output: 2
```



# What is Node.js?

## Website definition

Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.



# Installation

From source-code or pre-built installer

Visit <http://nodejs.org/download/> and choose the package for your platform.

# Installation

## From source-code or pre-built installer

Visit <http://nodejs.org/download/> and choose the package for your platform.

## Using nvm (UNIX environments)

Visit <https://github.com/creationix/nvm> and follow instructions.

# Installation

## From source-code or pre-built installer

Visit <http://nodejs.org/download/> and choose the package for your platform.

## Using nvm (UNIX environments)

Visit <https://github.com/creationix/nvm> and follow instructions.

## Check installation

```
$ node -v  
v0.8.21  
$ npm -v  
1.2.11
```

# Node packages

Node.js has a lot of packages that can be installed using npm. You can publish your own code as a node package and it will be available through npm.

# Node packages

Node.js has a lot of packages that can be installed using npm. You can publish your own code as a node package and it will be available through npm.

## Installing packages

```
$ npm install <package_name>
```



# What is Bower?

## Website definition

Bower is a package manager for the web. It offers a generic, unopinionated solution to the problem of front-end package management, while exposing the package dependency model via an API that can be consumed by a more opinionated build stack.



# Installation

## Using npm

```
$ npm install -g bower
```

# Installation

## Using npm

```
$ npm install -g bower
```

## Verify installation

```
$ bower -v  
1.2.7
```

# Commands

## bower init

Creates a bower.json file for including our application dependencies. Also, it's mandatory in order to register our application as a bower package or used on our internal projects through bower.

```
$ bower init
// Press ENTER for default answers
$ cat bower.json
{
  "name": "js-workshop-code",
  "version": "0.0.0",
  "homepage": "https://github.com/ultrayoshi/js-workshop-code",
  "authors": [
    "ultrayoshi <david@imesmes.com>"
  ],
  "license": "MIT",
  "ignore": [
    "**/.*",
    "node_modules",
    "bower_components",
    "test",
    "tests"
  ]
}
```

# Commands

## bower search

Find all packages or a specific package

```
$ bower search jquery
Search results:
jquery git://github.com/components/jquery.git
jquery-ui git://github.com/components/jqueryui
jquery.cookie git://github.com/carhartl/jquery-cookie.git
jquery-placeholder git://github.com/mathiasbynens/jquery-placeholder.git
jquery-file-upload git://github.com/blueimp/jQuery-File-Upload.git
jasmine-jquery git://github.com/velesin/jasmine-jquery
jquery.ui git://github.com/jquery/jquery-ui.git
jquery.scrollTo git://github.com/flesler/jquery.scrollTo.git
jquery-migrate git://github.com/appleboy/jquery-migrate.git
jquery-waypoints git://github.com/makewebthings/jquery-waypoints.git
...
```

# Commands

## bower search

Find all packages or a specific package

```
$ bower search jquery
Search results:
jquery git://github.com/components/jquery.git
jquery-ui git://github.com/components/jqueryui
jquery.cookie git://github.com/carhartl/jquery-cookie.git
jquery-placeholder git://github.com/mathiasbynens/jquery-placeholder.git
jquery-file-upload git://github.com/blueimp/jQuery-File-Upload.git
jasmine-jquery git://github.com/velesin/jasmine-jquery
jquery.ui git://github.com/jquery/jquery-ui.git
jquery.scrollTo git://github.com/flesler/jquery.scrollTo.git
jquery-migrate git://github.com/appleboy/jquery-migrate.git
jquery-waypoints git://github.com/makewebthings/jquery-waypoints.git
...
```

## bower home <package>

Opens a package homepage into your favorite browser

```
$ bower home jquery
```

# Commands

`bower install [package]`

Install a package locally

```
$ bower install jquery --save-dev
bower jquery#*           cached git://github.com/components/jquery.git#2.0.3
bower jquery#*           validate 2.0.3 against git://github.com/components/jquery.git#*
bower jquery#~2.0.3      install jquery#2.0.3

jquery#2.0.3 bower_components/jquery
```

--save-dev option add the package as a dependency of your application.  
Leave package name blank in order to install all your dependencies.

# Commands

`bower install [package]`

Install a package locally

```
$ bower install jquery --save-dev
bower jquery#*           cached git://github.com/components/jquery.git#2.0.3
bower jquery#*           validate 2.0.3 against git://github.com/components/jquery.git#*
bower jquery#~2.0.3      install jquery#2.0.3

jquery#2.0.3 bower_components/jquery
```

--save-dev option add the package as a dependency of your application.  
Leave package name blank in order to install all your dependencies.

`bower list`

List local packages

```
$ bower list
bower check-new    Checking for new versions of the project dependencies..
js-workshop-code#0.0.0 /home/david/code/js-workshop-code
jquery#2.0.3
```



# Simple exercise using jQuery

## Exercise

- git checkout boss\_stage\_1
- Install your web dependencies with bower install
- Create a function for creating character objects
- Create another function for creating player objects inheriting from characters
- Create another function for creating enemy objects inheriting from characters
- Use the jQuery sample code and complete 'Attack' and 'Drink potion' actions.
- (Optional) Use console.log to debug your actions



# What is Grunt?

## Definition

Grunt is a Javascript Task Runner. Simplify your life automating tedious tasks like minification, compilation, unit testing, etc. There are a lot of available Grunt plugins, take a look to the plugin directory at <http://gruntjs.com/plugins>.



# Installation

## Using npm

```
$ npm install -g grunt-cli
```

# Installation

## Using npm

```
$ npm install -g grunt-cli
```

## Gruntfile.js and package.json

Grunt needs a `Gruntfile.js` on your project's directory in order to work. Also, we need a `package.json` (similar to our `bower.json` but for our server-side dependencies) to add our Grunt plugins as a dependencies for our project.

```
$ npm init
// Default answers
$ npm install grunt --save-dev
```

# Gruntfile.js

## Basic Gruntfile.js

```
module.exports = function (grunt) {  
  grunt.registerTask('default', []);  
};
```

# Gruntfile.js

## Basic Gruntfile.js

```
module.exports = function (grunt) {  
  grunt.registerTask('default', []);  
};
```

## Run default task

```
$ grunt  
Done, without errors
```

# JSHint

JSHint is a tool that helps to detect errors and potential problems in your Javascript code.

# JSHint

JSHint is a tool that helps to detect errors and potential problems in your Javascript code.

## grunt-contrib-jshint

```
$ npm install grunt-contrib-jshint --save-dev
```

# JSHint

## Gruntfile.js

```
module.exports = function (grunt) {
  grunt.initConfig({
    jshint: {
      all: {
        options: {
          jshintrc: '.jshintrc'
        },
        files: {
          src: ['Gruntfile.js', 'js/**/*.js']
        }
      }
    }
  });
  grunt.loadNpmTasks('grunt-contrib-jshint');
  grunt.registerTask('default', ['jshint']);
};
```

# JSHint

## .jshintrc

```
{  
  "curly": true,  
  "eqeqeq": true,  
  "immed": true,  
  "latedef": true,  
  "newcap": true,  
  "noarg": true,  
  "sub": true,  
  "undef": true,  
  "unused": true,  
  "boss": true,  
  "eqnull": true,  
  "browser": true,  
  "node": true,  
  "expr": true,  
  "globals": {  
  }  
}
```

# JSHint

## Running task

```
$ grunt jshint:all
Running "jshint:all" (jshint) task
Linting js/main.js ...ERROR
[L1:C1] W117: '$' is not defined.
$(function ( {
[L29:C18] W117: '$' is not defined.
  var el = $($('#' + character.id));
[L50:C18] W117: '$' is not defined.
  var el = $("#" + character.id);

Warning: Task "jshint:all" failed. Use --force to continue.
Aborted due to warnings.
$ grunt jshint
// Same output
$ grunt
// Same output
```

# Watch

Running `grunt` or `grunt jshint` command manually is a bit painful. We can automate this process using another `grunt` plugin.

# Watch

Running grunt or grunt jshint command manually is a bit painful. We can automate this process using another grunt plugin.

## grunt-contrib-watch

```
$ npm install grunt-contrib-watch --save-dev
```

# Watch

## Gruntfile.js

```
module.exports = function (grunt) {
  grunt.initConfig({
    // other plugin configurations omitted
    watch: {
      jshint: {
        files: ['js/**/*.js'],
        tasks: ['jshint']
      }
    }
  });

  // other loading npm tasks omitted
  grunt.loadNpmTasks('grunt-contrib-watch');

  grunt.registerTask('default', ['jshint']);
};
```

# Watch

## Running task

```
$ grunt watch
Running "watch" task
Waiting...OK
>> File "js/main.js" changed.

Running "jshint:all" (jshint) task
Linting js/main.js ...ERROR
[L1:C1] W117: '$' is not defined.
$(function ( {
[L29:C18] W117: '$' is not defined.
  var el = $( '#' + character.id);
[L50:C18] W117: '$' is not defined.
  var el = $( '#' + character.id);

Warning: Task "jshint:all" failed. Use --force to continue.

Aborted due to warnings.
Completed in 0.646s at Sun Nov 17 2013 17:23:21 GMT+0100 (CET) - Waiting...
```

# Concat

You can concat your scripts file using grunt in order to make fewer HTTP requests.

# Concat

You can concat your scripts file using grunt in order to make fewer HTTP requests.

## grunt-contrib-concat

```
$ npm install grunt-contrib-concat --save-dev
```

# Concat

## Gruntfile.js

```
module.exports = function (grunt) {
  grunt.initConfig({
    // other plugin configurations omitted
    concat: {
      options: {
        separator: ';'
      },
      dist: {
        src: ['bower_components/jquery/jquery.js', 'js/main.js'],
        dest: 'build/built.js'
      }
    }
  });

  // other loading npm tasks omitted
  grunt.loadNpmTasks('grunt-contrib-concat');

  grunt.registerTask('default', ['jshint', 'concat']);
};
```

# Concat

## Running task

```
$ grunt concat
Running "concat:dist" (concat) task
File "dist/built.js" created.
```

Done, without errors.

# Uglify

You can optimize your scripts using grunt in order to minimize its size. Combine with concat for better results.

# Uglify

You can optimize your scripts using grunt in order to minimize its size. Combine with concat for better results.

## grunt-contrib-uglify

```
$ npm install grunt-contrib-uglify --save-dev
```

# Uglify

## Gruntfile.js

```
module.exports = function (grunt) {
  grunt.initConfig({
    // other plugin configurations omitted
    uglify: {
      build: {
        files: {
          'dist/built.min.js': 'dist/built.js'
        }
      }
    }
  });

  // other loading npm tasks omitted
  grunt.loadNpmTasks('grunt-contrib-uglify');

  grunt.registerTask('default', ['jshint', 'concat', 'uglify']);
};
```

# Uglify

## Running task

```
$ grunt uglify
Running "uglify:build" (uglify) task
File "dist/built.min.js" created.
```

Done, without errors.

```
$ ls -l dist
total 328
-rw-rw-r-- 1 david david 247260 Nov 17 18:05 built.js
-rw-rw-r-- 1 david david 84949 Nov 17 18:12 built.min.js
```



# Grunt's playground

## Exercise

- `git checkout boss_stage_2`
- Install your front-end dependencies using `bower install`
- Install your back-end dependencies using `npm install`
- Run `grunt watch`
- Fix all js lint errors
- Add a `reset.css` and a basic stylesheet to the application
- Install `grunt-contrib-csslint` and `grunt-contrib-cssmin`
- Concat css files and optimize them
- Run default task for creating a built and check the result



## Game design

We are going to create a simple game in the following stages using Javascript. The game will be a tiny MOBA (Multiplayer Online Battle Arena) and it will have the following features:

## Game design

We are going to create a simple game in the following stages using Javascript. The game will be a tiny MOBA (Multiplayer Online Battle Arena) and it will have the following features:

- Player choose a team: red or blue

# Game design

We are going to create a simple game in the following stages using Javascript. The game will be a tiny MOBA (Multiplayer Online Battle Arena) and it will have the following features:

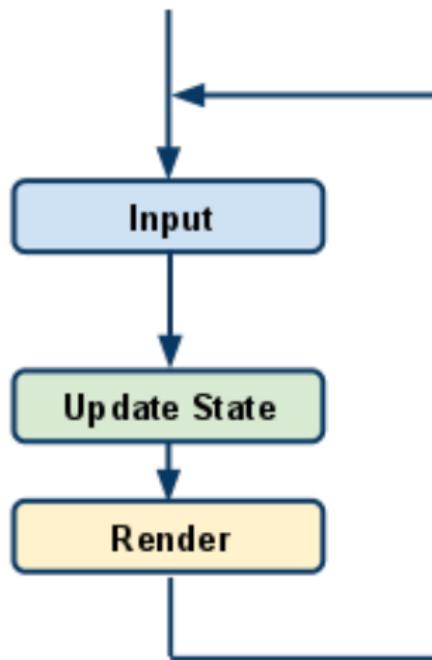
- Player choose a team: red or blue
- Player choose between 3 classes:
  - Soldier: low hp, ranged weapon, medium damage
  - Knight: medium hp, melee weapon, high damage
  - Protector: high hp, no weapon, can block enemy's attacks

# Game design

We are going to create a simple game in the following stages using Javascript. The game will be a tiny MOBA (Multiplayer Online Battle Arena) and it will have the following features:

- Player choose a team: red or blue
- Player choose between 3 classes:
  - Soldier: low hp, ranged weapon, medium damage
  - Knight: medium hp, melee weapon, high damage
  - Protector: high hp, no weapon, can block enemy's attacks
- **Objective:** Destroy other's team base

# Game loop



# HTML5

First of all, HTML5 is not a programming language, neither an API. It's the 5th revision of HTML but also an umbrella term about 100 specifications for the next generation web applications. Visit <http://platform.html5.org/> to have a global view about it.

# HTML5

First of all, HTML5 is not a programming language, neither an API. It's the 5th revision of HTML but also an umbrella term about 100 specifications for the next generation web applications. Visit <http://platform.html5.org/> to have a global view about it. We are going to use a small subset of these new APIs:

# HTML5

First of all, HTML5 is not a programming language, neither an API. It's the 5th revision of HTML but also an umbrella term about 100 specifications for the next generation web applications. Visit <http://platform.html5.org/> to have a global view about it. We are going to use a small subset of these new APIs:

- **Canvas:** Drawing graphics in 2D.

# HTML5

First of all, HTML5 is not a programming language, neither an API. It's the 5th revision of HTML but also an umbrella term about 100 specifications for the next generation web applications. Visit <http://platform.html5.org/> to have a global view about it. We are going to use a small subset of these new APIs:

- **Canvas:** Drawing graphics in 2D.
- **requestAnimationFrame:** Handling animations timings.

# HTML5

First of all, HTML5 is not a programming language, neither an API. It's the 5th revision of HTML but also an umbrella term about 100 specifications for the next generation web applications. Visit <http://platform.html5.org/> to have a global view about it. We are going to use a small subset of these new APIs:

- **Canvas:** Drawing graphics in 2D.
- **requestAnimationFrame:** Handling animations timings.
- **Websockets:** Two-way communication between browser and server.

# Introduction to the Canvas API

HTML5 introduces a new tag called `canvas`. Using Javascript we can interact with this element in order to draw 2D graphics in real time.

## Canvas element and 2d context

```
<canvas id='gameArea' width='200' height='200'></canvas>
<script>
var canvas = document.getElementById('gameArea'),
    ctx = canvas.getContext('2d');
</script>
```

# Introduction to the Canvas API

## Drawing lines

```
ctx.fillStyle = 'black';
```

```
ctx.beginPath();
ctx.moveTo(10, 10);
ctx.lineTo(100, 10);
ctx.stroke();
```

```
ctx.beginPath();
ctx.moveTo(10, 20);
ctx.lineTo(100, 20);
ctx.stroke();
```

```
ctx.beginPath();
ctx.moveTo(10, 30);
ctx.lineTo(100, 30);
ctx.stroke();
```

# Introduction to the Canvas API

## Drawing rects

```
ctx.fillStyle = "blue";
ctx.strokeStyle = "red";
ctx.fillRect(100, 100, 50, 50);
ctx.strokeRect(165, 165, 25, 25);
```

# Introduction to the Canvas API

## Drawing rects

```
ctx.fillStyle = "blue";
ctx.strokeStyle = "red";
ctx.fillRect(100, 100, 50, 50);
ctx.strokeRect(165, 165, 25, 25);
```

## Drawing arcs

```
ctx.beginPath();
ctx.fillStyle = "green";
ctx.strokeStyle = "orange";
ctx.arc(150, 50, 5, 0, 2 * Math.PI);
```

# Introduction to the Canvas API

## Drawing images

```
var crate = new Image();
crate.src = 'images/crate.png';
/*
 * onload callback function. Called when the
 * image is ready to be drawn.
 */
crate.onload = function () {
    ctx.drawImage(crate, 100, 100);
};
```

# Introduction to the Canvas API

## scale

```
// Draw crate 2x bigger
ctx.scale(2, 2);
ctx.drawImage(crate, 100, 100);
```

# Introduction to the Canvas API

## scale

```
// Draw crate 2x bigger
ctx.scale(2, 2);
ctx.drawImage(crate, 100, 100);
```

## translate

```
// Same as ctx.drawImage(crate, 100, 100);
ctx.translate(100, 100);
ctx.drawImage(crate, 0, 0);
```

# Introduction to the Canvas API

## scale

```
// Draw crate 2x bigger
ctx.scale(2, 2);
ctx.drawImage(crate, 100, 100);
```

## translate

```
// Same as ctx.drawImage(crate, 100, 100);
ctx.translate(100, 100);
ctx.drawImage(crate, 0, 0);
```

## rotate

```
ctx.rotate(45 * (Math.PI * 180));
ctx.drawImage(crate, 0, 0);
```

# Introduction to the Canvas API

## save and restore

```
ctx.save();
ctx.translate(100, 100);
ctx.translate(crate.width / 2, crate.height / 2);
ctx.rotate(45 * (Math.PI / 180));
ctx.translate(-crate.width / 2, -crate.height / 2);
ctx.drawImage(crate, 0, 0);
ctx.restore();

ctx.drawImage(create, 0, 0);
```

## requestAnimationFrame

Our game must run on 60 fps (frames per second) in order to be smoothly. It means, we must execute the game loop 60 times each second or 1 time each  $1000 / 60$  milliseconds.

## requestAnimationFrame

Our game must run on 60 fps (frames per second) in order to be smoothly. It means, we must execute the game loop 60 times each second or 1 time each  $1000 / 60$  milliseconds.

### Using setInterval

```
setInterval(function () {  
    console.log('Game loop!');  
}, 1000 / 60);
```

## requestAnimationFrame

Our game must run on 60 fps (frames per second) in order to be smoothly. It means, we must execute the game loop 60 times each second or 1 time each  $1000 / 60$  milliseconds.

### Using setInterval

```
setInterval(function () {  
    console.log('Game loop!');  
}, 1000 / 60);
```

### Using requestAnimationFrame

```
function gameLoop() {  
    requestAnimationFrame(gameLoop);  
    console.log('Game loop!');  
}  
  
gameLoop();
```



# Hello world game!

## Exercise

- `git checkout boss_stage_3`
- Install your front-end dependencies using `bower install`
- Install your back-end dependencies using `npm install`
- Run `grunt watch`
- Finish implementation of game module
- Finish implementation of entity constructor function
- Finish implementation of player constructor function
- Finish implementation of crate constructor function
- Complete main file