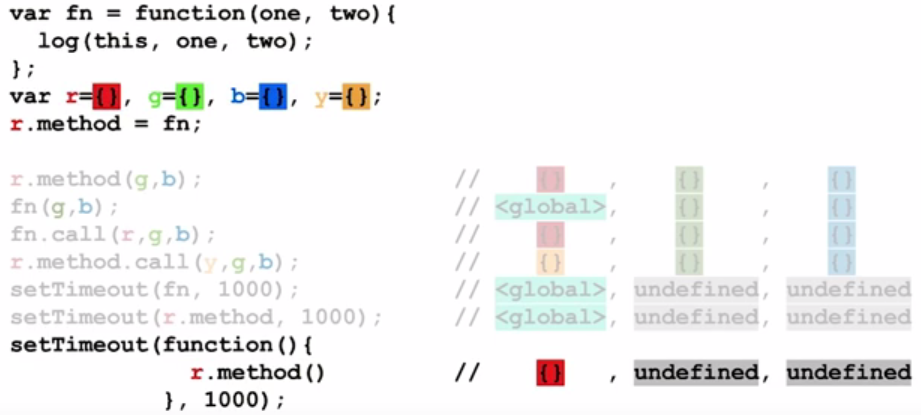
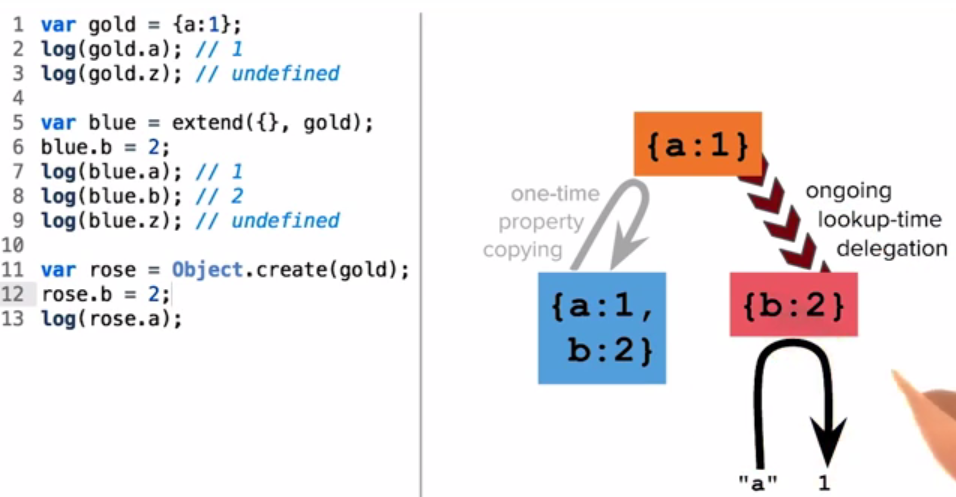
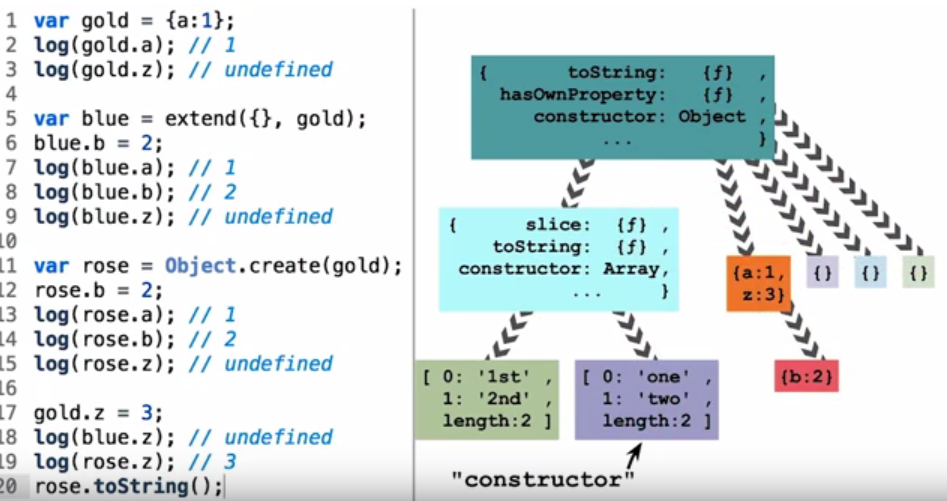
This Keyword

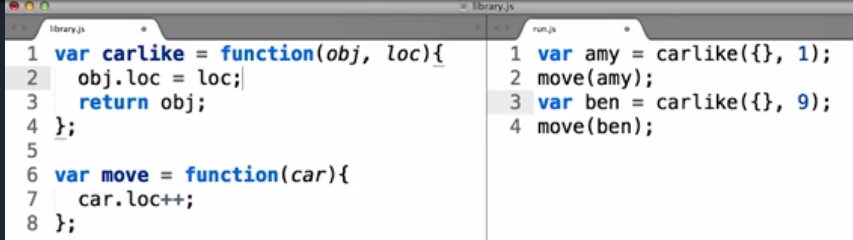


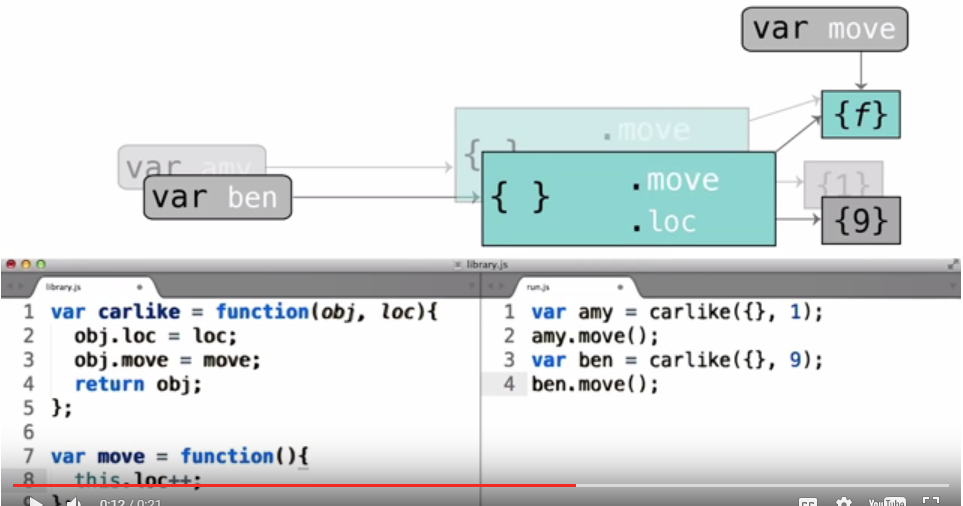
Prototype Chains



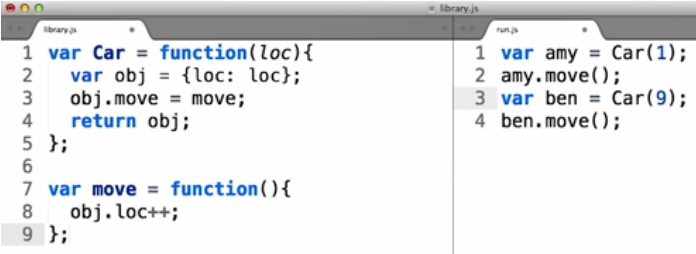


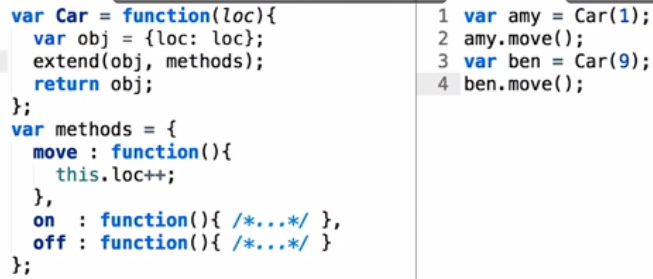
Code Reuse



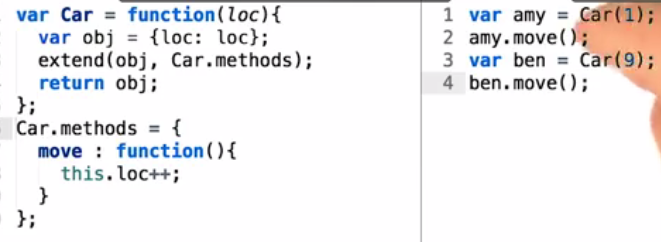


Classes

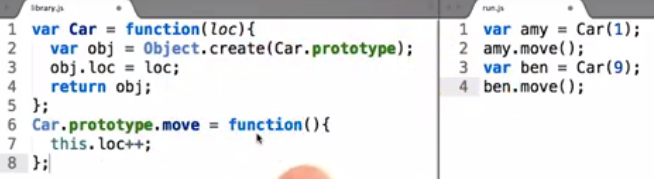


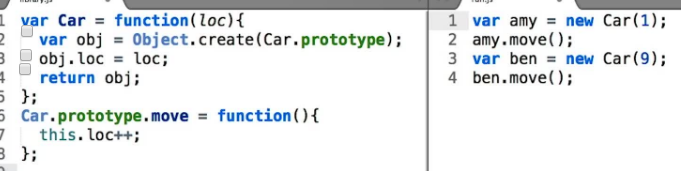


removed the car methods from global scope from the code below

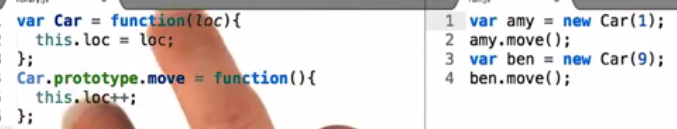
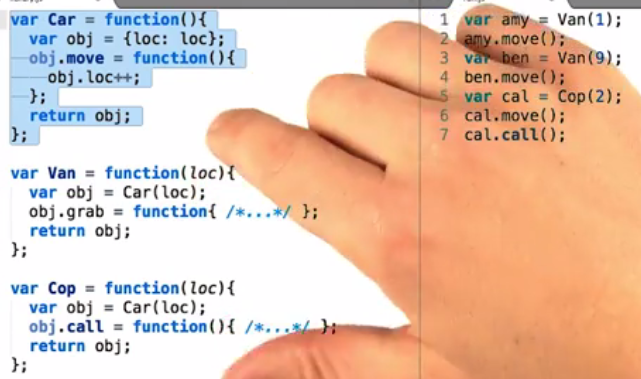


Prototypal Classes

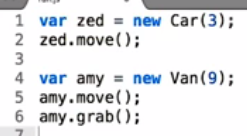




Sudo Classical

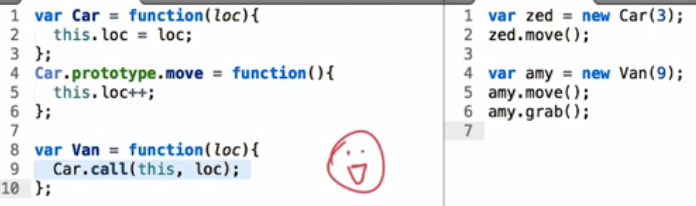
Pseudo classical Subclasses



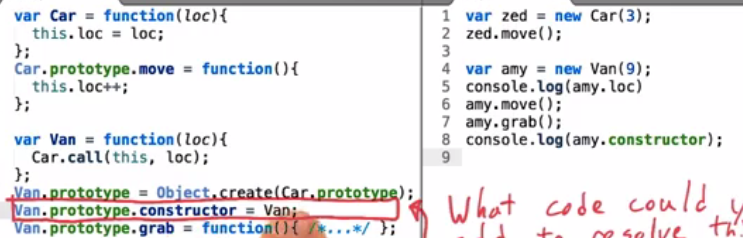
The prototype function group will allow any object to access the function:

Car.prototype.move = function() {  
this.loc++;

};

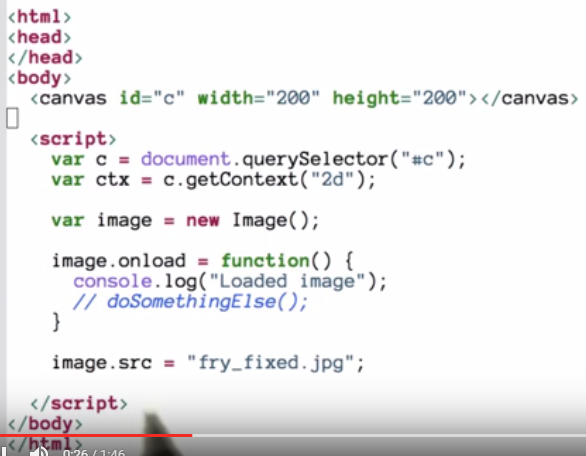


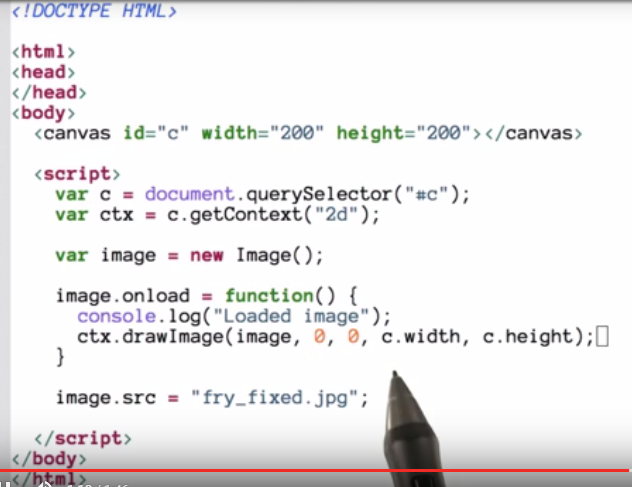




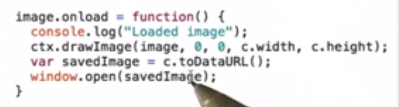
HTML5 Canvas Basics

* Setting up a canvas and creating an image





Save the photo you would need to create your own server because to serve a tainted image you would need to do so.



There are couple of ways to get around this error and one (optionally) is to start a local web server. Below, I've outlined some different options for simple HTTP web server with instructions for setting them up.

**Python SimpleHTTPServer**

* Windows, Mac, Linux
* [**Download here**](https://www.python.org/downloads/) (you may already have it installed!)

All you need is Python and a command line. Python comes with Mac and Linux but Windows users will need to download it. There are two main versions of Python, Python 2 and Python 3. Both are perfectly fine for setting up a simple server.

**Instructions**

1. Open the terminal (Mac and Linux) or command prompt (Windows).
2. cd to a directory where you've saved an HTML file. For example, cd ~/Documents/mysite/. (Mac and Linux: here's more on [**cd**](https://kb.iu.edu/d/afsk#cd) if you aren't familiar with the terminal. Windows: here's more info on [**cd**](http://www.digitalcitizen.life/command-prompt-how-use-basic-commands) for you).
3. Run python --version. If Python is installed, you'll see "Python X.Y.Z". The "X" will be 2 or 3, indicating Python 2 or 3. If nothing shows up or the command produces an error, I recommend that you [**download Python**](https://www.python.org/downloads/).
4. If you have **Python 2**, run python -m SimpleHTTPServer 8000. If you have **Python 3**, run python -m http.server 8000.
5. Navigate your browser to http://localhost:8000/. If there is a file called index.html in the directory where you ran the command from step 4, then it should automatically show up! If not, you should see the files in that directory listed. Click on an HTML file and watch it load! Congrats! You've got a server running.

(What was that 8000? It's a network port, which computers use to organize their network traffic.)

**MAMP**

* Windows and Mac
* [**Download here**](https://www.mamp.info/) (you only need the free version!)

This is a simple [**GUI**](https://en.wikipedia.org/wiki/Graphical_user_interface) for serving a directory.

**Installation**

Check out the [**MAMP website installation instructions**](https://documentation.mamp.info/documentation/mamp/).

**Node**

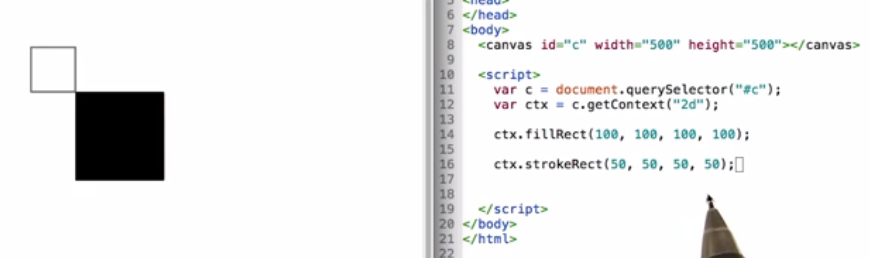
* Windows, Mac, Linux
* Requires [**Node**](https://nodejs.org/) and npm (comes with node)

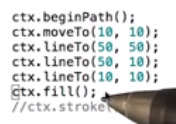
Similar to the Python technique above, this is a simple command line tool. This technique uses Node, which is a popular JavaScript runtime (i.e, a piece of software that runs arbitrary JavaScript outside of the browser). Node is incredibly popular for a number of uses, including servers. While Node rarely shows up in front-end work, every web developer will likely encounter its package installer, npm. npm (Node Package Manager) is a command line tool that makes it easy to install and manage other command line tools, libraries and frameworks.

This strategy may take a little more time to install than Python, but it's worth trying so that you get a head-start with npm. That being said, if you try this and run into problems, it's not a big deal - just use a different strategy.

**Installation**

1. Run node --version from the terminal or command line. If nothing shows up or you get an error, [**install Node**](https://nodejs.org/).
2. Run npm install -g http-server.
3. Serve files with http-server ~/Documents/mysite -p 8000 (replace ~/Documents/mysite with the path to your project's directory!).
4. Navigate your browser to http://localhost:8000 to test!





So far, we’ve been drawing everything using exact coordinates. This is fine for a couple shapes but breaks down if you need to draw a bunch of objects.

Canvas2D allows you to translate (move), rotate, or scale objects.

### Scaling

scale(x,y) multiplies the x and y values by a given factor so

ctx.scale(2,3);

will make all values twice as large on the x axis and three times as large on the y axis.

### Translation

translate(x,y) moves all subsequent draw commands by x number of pixels on horizontally and y pixels vertically.

ctx.translate(20,40); moves all elements drawn after it 20 pixels to the rights and 40 pixels down.

### Rotation

ctx.rotate(angleRadians) rotates an object a certain number of radians (generally) about its center. You may have learned about radians in school but here's a handy formula to convert a value from degrees to radians.

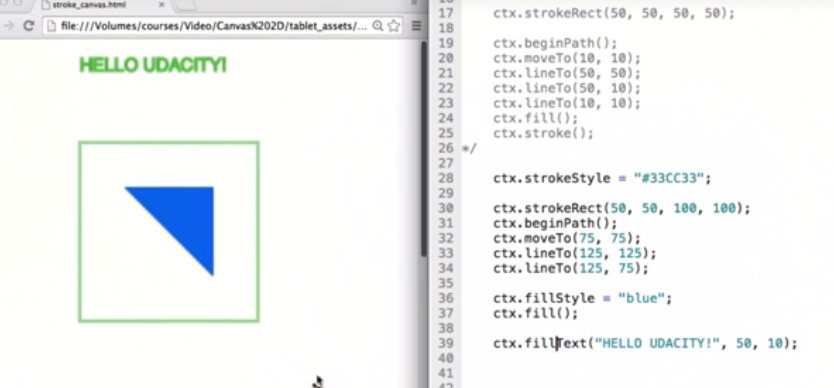
radians = degrees \* (Math.PI/180)

Don't ask us why everything in Computer Graphics uses radians. We have no idea. :)

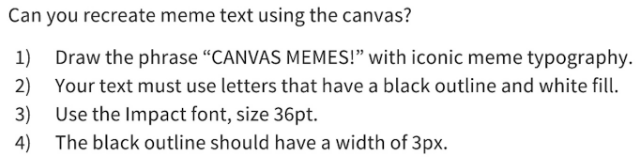
### Order of operations

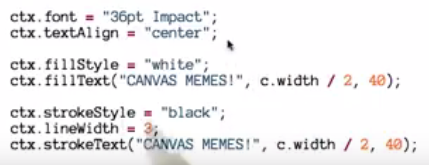
You should generally scale objects first, rotate them next, and then finally translate last. There are times when you'd want to rotate around an arbitrary point instead of an object's center, that's out of scope for this lesson.

It’s important to note that whatever transformations apply for all subsequent objects until you reverse them.



Classic Meme text:







From Pixels to automation:

# Silent Film Vignette

Don't worry if you can't follow all the code in these links right now. Try to look at them from a high level and don't worry about the syntax too much.

Tanner Helland outlines a bunch of different algorithms for computing grayscale [**here**](http://www.tannerhelland.com/3643/grayscale-image-algorithm-vb6/) in pseudocode and Visual Basic. Mozilla has a great tutorial on using how to do a [**green screen effect**](https://developer.mozilla.org/en-US/docs/Web/HTML/Manipulating_video_using_canvas).

The visual effects and resources that we didn't create in code are listed below

## Attributions

### City Background

**Title**: What Happened on Twenty-Third Street, New York City (1901)  
**Director**: Edwin S. Porter  
**Production Company**: Edwin S. Porter  
**Sponsor**: Edison Mfg. Co.  
[**https://archive.org/details/What\_Happened\_1901**](https://archive.org/details/What_Happened_1901)

### Train Background

**Title**: Freight Train (1898)  
**Producer**: The Edison Manufacturing Co. and Thomas A. Edison, Inc.  
[**https://archive.org/details/EdisonMotionPicturesCollectionPartOne1891-1898**](https://archive.org/details/EdisonMotionPicturesCollectionPartOne1891-1898)  
[**https://archive.org/download/EdisonMotionPicturesCollectionPartOne1891-1898/1898Freight\_train.mpg**](https://archive.org/download/EdisonMotionPicturesCollectionPartOne1891-1898/1898Freight_train.mpg)

### Music

**Title**: Old Fasioned Auto Piano.wav  
**Producer**: Razzvio  
[**http://www.freesound.org/people/Razzvio/sounds/79572/**](http://www.freesound.org/people/Razzvio/sounds/79572/)

### Title Card Design

**Name**: Silent Movie The End Title Card HD  
**Creator**: Farrin N. Abbott / CopyCatFilms

### Filters and Effects

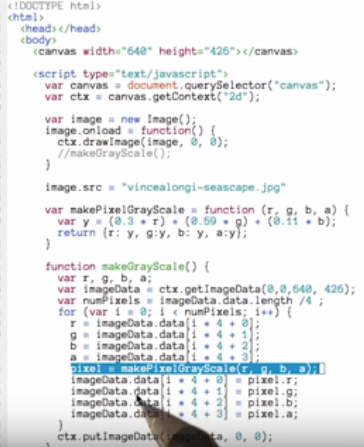
Night Vision Scope

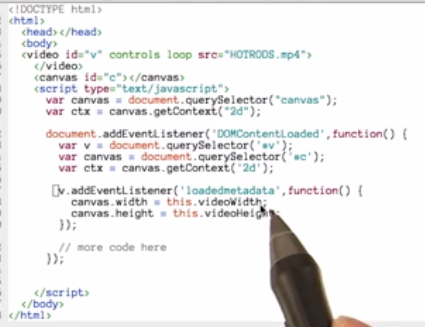
Creator: Andrew Kramer / Video Copilot [**http://www.videocopilot.net/blog/2012/11/new-tutorial-simulated-scopes/**](http://www.videocopilot.net/blog/2012/11/new-tutorial-simulated-scopes/)

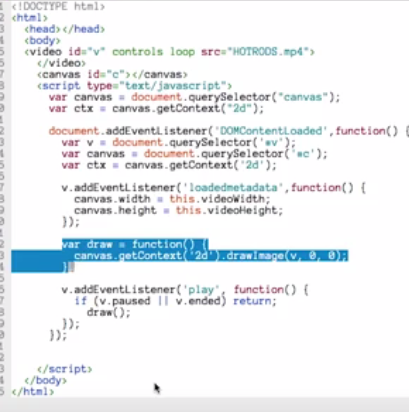


Grayscale:

I used an algorithm similar to one of those listed earlier in the link from [**Tanner Helland**](http://www.tannerhelland.com/3643/grayscale-image-algorithm-vb6/). Check this post titled [**"Human Vision and Color Perception"**](http://micro.magnet.fsu.edu/primer/lightandcolor/humanvisionintro.html) for more information about how the human eye works.

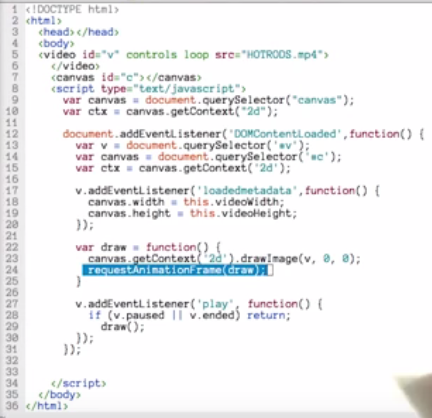






# RequestAnimationFrame

Paul Irish's [**post**](http://www.paulirish.com/2011/requestanimationframe-for-smart-animating/) about requestAnimationFrame is an evergreen primer for when and how to use it. **[Kirupa](http://www.kirupa.com/html5/animating_with_requestAnimationFrame.htm" \t "_blank)** and [**Mozilla's MDN**](https://developer.mozilla.org/en-US/docs/Web/API/window.requestAnimationFrame) are great resources as well.



function draw() {

// request to execute this function at the next earliest convenience

requestAnimationFrame(draw);

processInput();

moveObjectsAndEnemies();

drawAllTheThings();

}

### Processing Keyboard Input

While it isn't too difficult to process keyboard presses by hand, I rather stand on the shoulders of giants and use open source projects that have perfected a library serving the thing I want to do. One such library is **[Kibo](https://github.com/marquete/kibo" \t "_blank)**.

Kibo allows you to reference keys by their common names('a', '3', 'up') instead of their keycodes greatly simplifying your code. You can also attach events to pressing or releasing a key as well as modifier keys or wildcards.

var k = new Kibo();

k.down(['up', 'w'], function() {

// Do something cool on the canvas

});

k.up(['enter', 'q'], function() {

// Do other stuff.

});

### Processing Mouse Input

Like many other DOM elements, the canvas can accept click and mousedown events. We do however have to do a little work to figure out where exactly in the canvas the user has clicked. Mouse click events return clientX and clientYpositions that are global to the browser window. Every element knows where it is positioned relative to the browsers (0,0) position (offsetLeft and offsetTop).

To get the canvas-relative of a click, you need to subtract the offsetLeft and offsetTop values from clientX and clientY. Check out the example code below.

var c = document.querySelector("canvas");

function handleMouseClick(evt) {

x = evt.clientX - c.offsetLeft;

y = evt.clientY - c.offsetTop;

console.log("x,y:"+x+","+y);

}

c.addEventListener("click", handleMouseClick, false);

[**Kibo.js**](https://github.com/marquete/kibo) - a JavaScript library for processing keyboard input.

var canvas, context;

init();

animate();

function init() {

canvas = getCanvas();

context = canvas.getContext( '2d' );

}

function animate() {

requestAnimationFrame( animate );

draw();

}

function draw() {

var time = new Date().getTime() \* 0.002;

var x = Math.sin( time ) \* 96 + 38;

var y = Math.cos( time \* 0.9 ) \* 96 + 38;

context.fillStyle = 'rgb(245,245,245)';

context.fillRect( 0, 0, 255, 255 );

context.fillStyle = 'rgb(255,0,0)';

context.beginPath();

context.arc( x, y, 10, 0, Math.PI \* 2, true );

context.closePath();

context.fill();

}

# Readme

* [**README #1 - factory\_girl**](https://github.com/thoughtbot/factory_girl)
* [**README #2 - can.viewify**](https://github.com/zkat/can.viewify)
* [**README #3 - create-your-own-adventure**](https://github.com/udacity/create-your-own-adventure)