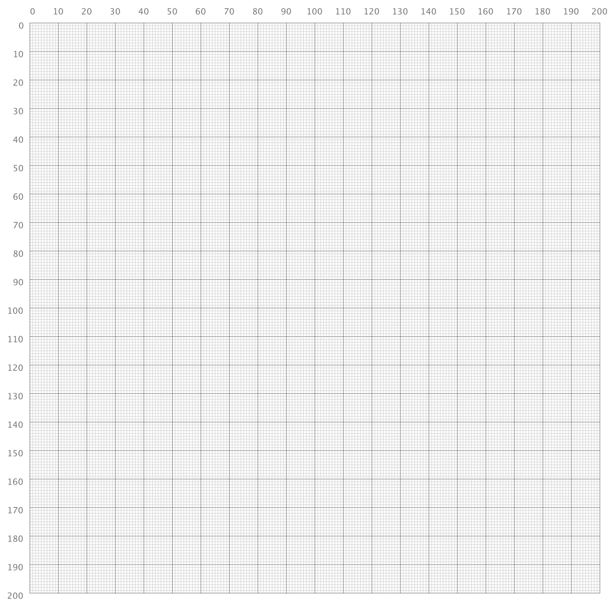
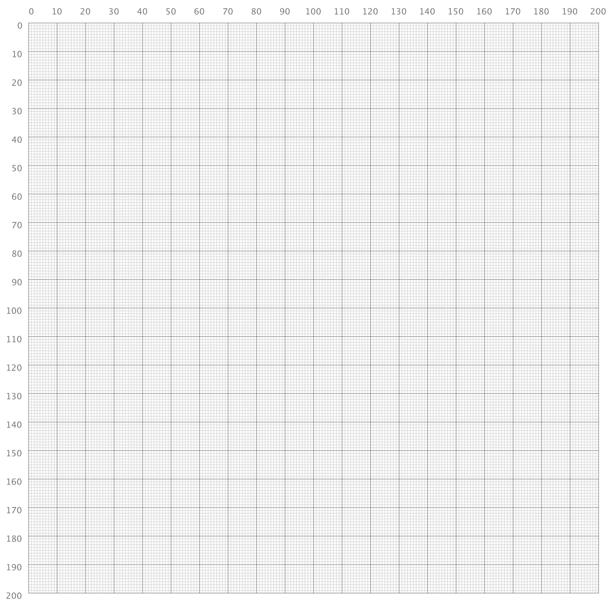
CreativeJS for non-coders: notes

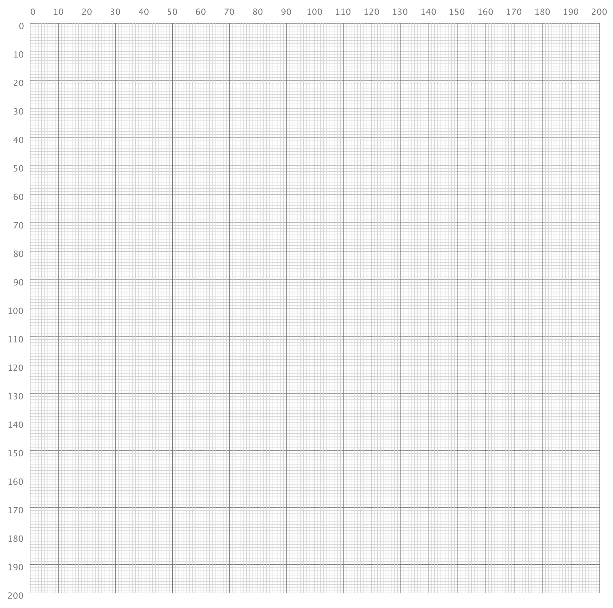
March 2012 • seb.ly

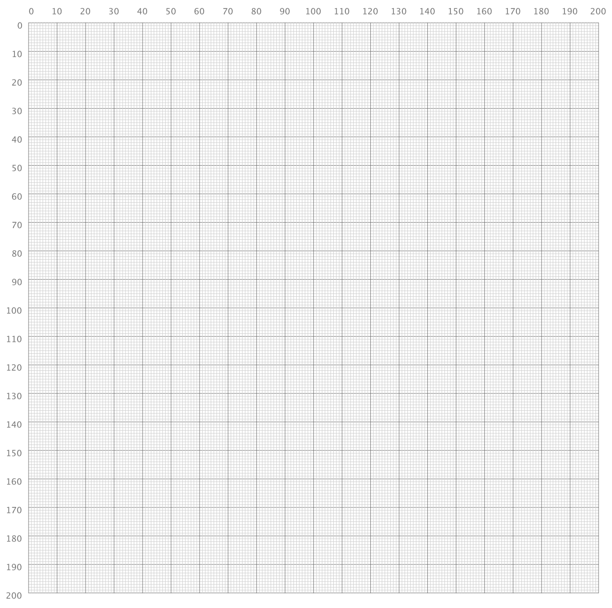
Graphics

Coordinates



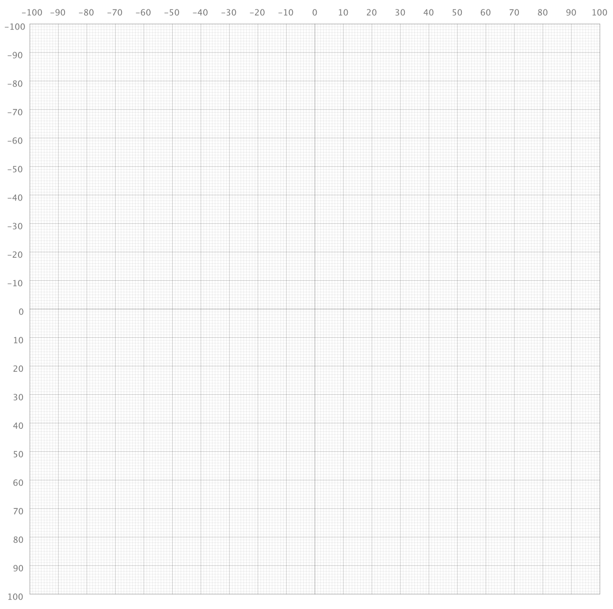


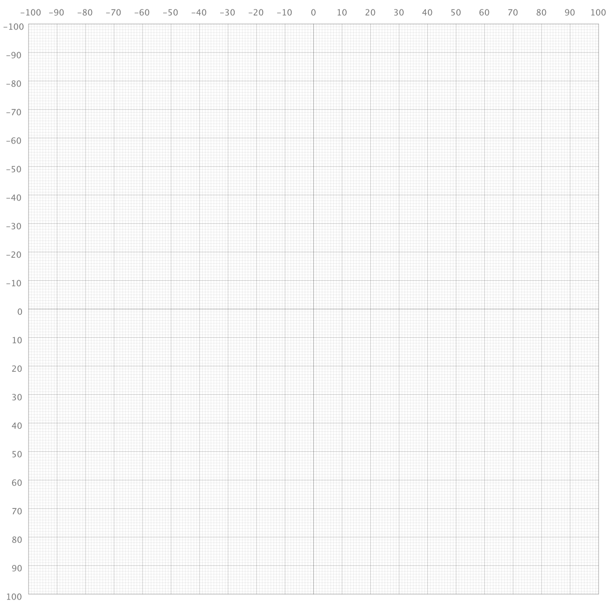


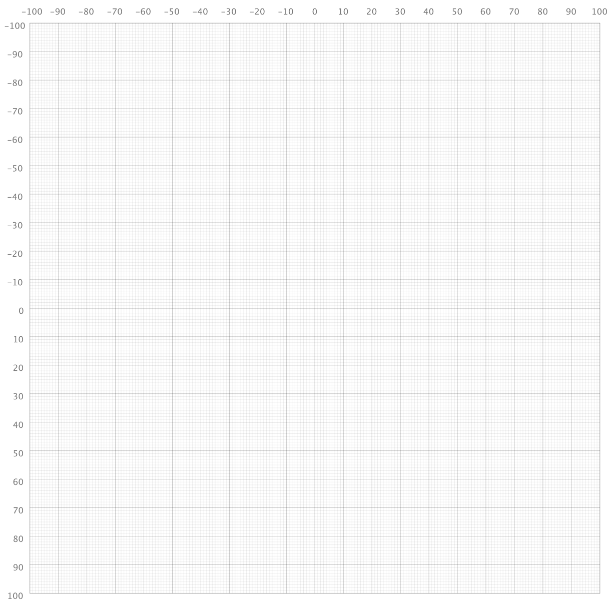


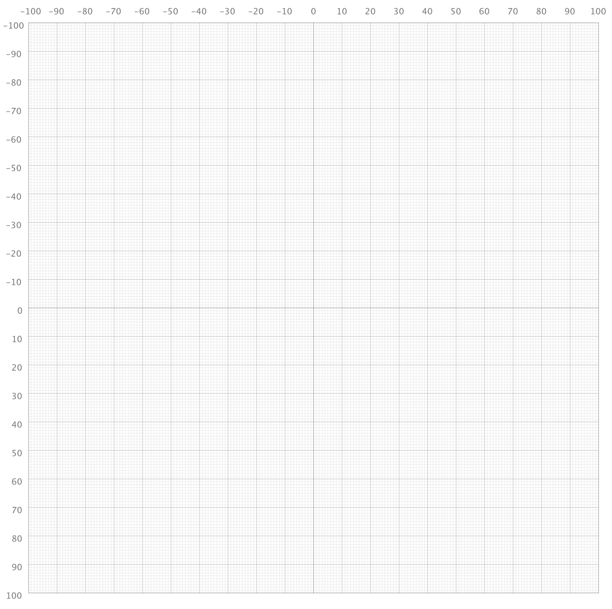
Graphics

Coordinates









Rectangles

In the following examples, *c* refers to the canvas context that we're drawing to, *x* and *y* specify the top left position of the rectangle.

`c.clearRect(x, y, width, height);`	clears a rectangle - the area goes completely transparent.
`c.fillRect(x, y, width, height);`	draw a solid rectangle with the current fill colour.
`c.strokeRect(x, y, width, height);`	draw an outline rectangle with the current stroke colour.
`c.rect(x, y, width, height);`	define the path for a rectangle without drawing it. (See paths)

Circles and Ellipses

For these functions, *x* and *y* are the centre of the circle or ellipse.

`c.fillCircle(x, y, radius);`	draw a solid circle with the current fill colour
`c.strokeCircle(x, y, radius);`	draw an outline circle with the current stroke colour
`c.circle(x, y, radius);`	define the path for a circle without drawing it. (See paths)
`c.fillEllipse(x, y, width, height);`	draw a solid ellipse with the current fill colour
`c.strokeEllipse(x, y, width, height);`	draw an outline ellipse with the current stroke colour
`c.ellipse(x, y, width, height);`	define the path for an ellipse without drawing it. (See paths)

Lines

`c.line(x1, y1, x2, y2);`	draw a line between the two points
`c.lineWidth = width;`	set the stroke width

Line and fill colour

`c.strokeStyle = 'colour';`	set all subsequent lines to be drawn with the specified colour
`c.fillStyle = 'colour';`	set all subsequent fills to be the specified colour (see colours)

Paths

You can define a custom path and then later stroke and/or fill it.

`c.beginPath();`	starts a new path
`c.moveTo(x, y);`	moves to a new position
`c.lineTo(x, y);`	adds a line to the path to the position specified
`c.closePath();`	closes the current path
	warning - it doesn't reset the path! Use beginPath() again to restart

Add a quadratic bezier curve to the path:

```
c.quadrativeCurveTo(controlX, controlY, endX, endY);
```

Add a cubic bezier curve to the path:

```
c.bezierCurveTo(controlX1, controlY1, controlX2, controlY2, endX, endY);
```

Add an arc to the path (note that the angles are required in radians - see *radians* function)

```
c.arc(x, y, radius, startAngle, endAngle, drawAntiClockwise);
```

`c.fill();`	Fill the current path with the colour set with fillStyle
`c.stroke();`	Stroke the current path with the colour set with strokeStyle

Colours

Common colours : *aqua*, *black*, *blue*, *fuchsia*, *gray*, *green*, *lime*, *maroon*, *navy*, *olive*, *purple*, *red*, *silver*, *teal*, *white*, and *yellow*.

```
c.strokeStyle = 'green';
c.fillStyle = 'gray';
```

Define colours with 3 digit hex values

`'#rgb'`	where r is red, g is green and b is blue, hex values go from 0 to F.
`'#F00'`	full red and no green or blue which makes red
`'#0F0'`	green
`'#FFF'`	red, green and blue all at full brightness, which makes white

Define colours with 6 digit hex values

`'#rrggbb'`	
`'#FF00FF'`	magenta
`'#FFAB00'`	orange
`'#FFFF00'`	yellow
`'#000066'`	dark blue

Define colours with rgb

`rgb(r, g, b)`	values are between 0 and 255.
`rgb(255, 255, 255)`	white
`rgb(100,100,100)`	grey

Define colours with rgba

alpha determines how transparent the colour is and goes from 0 (fully transparent) to 1 (fully opaque).

`rgba(r, g, b, alpha)`	
`rgba(255, 255, 255, 0.5)`	half transparent white

Define colours with hsl and hsla

`hsl(hue, saturation, lightness)`

hue is in the range from 0 to 360 (like the colour wheel), *saturation* and *lightness* from 0% to 100%.

`hsl(0, 100%, 50%)`	hue is 0 (red), 100% saturation, and 50% lightness gives a pure red
`hsl(50, 0%, 50%)`	hue is ignored as we have 0% saturation to give a mid grey.

`hsla(hue, saturation, lightness, alpha)`

Same as *hsl*, but with an added *alpha* value that goes from 0 to 1, as in *rgba()*.

Variables

Declaration and assignation

Before you use a variable you have to declare it, and it will have a value of 'undefined' until you assign (or set) it.

`var x;`	declare a variable called x
`var x = 0;`	make a variable called x and set its value to 0
`var day = 'Monday';`	make a variable called day and set its value to '*Monday*'
`var x, y, z;`	declare multiple variables
`var x = 0, y = 3, z = 4;`	declare and assign multiple variables
`var     x = 0,`	multiple variable declaration and assignment split up on several lines to
`        y = 3,`	make it easier to read
`        z = 4;`	

Variable names

```
var position = 50;
var lastName = 'Smith';

var firstName = 'Harry';
var aVeryLongVariableName = 'probably shouldn't have variable names that are too long... ';
```

Variable types

Numbers, ie *24*, *58*, *1.5*, *-40*, *348429561.24*
Strings, ie '*hello*', '*abc*', '*Monday*'. A very long sentence can be stored in a string.
Booleans - *true* or *false*.

`var greeting = 'hello';`	a string
`var xPosition = 25;`	a number
`var isActive = true;`	a boolean

Changing variables

Adding

`x = x + 2;`	let *x* be equal to *x* plus 2
`x += 2;`	another way of adding 2 to *x*
`x ++;`	add 1 to *x*

Subtracting

`counter = counter - 4;`	let *counter* equal counter minus 4
`counter -= 4;`	another way to subtract 4 from *counter*

Multiplication

`y = y * 5;`	let *y* equal *y* multiplied by 5
`y *= 5;`	shorthand version that does the same thing
`y *= x;`	*y* becomes *y* multiplied by *x*

Division

`counter = counter / 2;`	let *counter* be equal to *counter* divided by 2
`counter /= 2;`	another way to divide *counter* by 2

Adding to strings:

```
var greeting = 'Hello';
greeting = greeting + ' Sarah';
```

greeting now becomes *'Hello Sarah'*

Joining strings together:

```
var name = 'Sarah';
var greeting = 'Hello '+name+', welcome.';
```

greeting becomes *'Hello Sarah, welcome.'*

Loops

While loops

```
while(x < 10) {        // while x is less than 10
       // do this stuff
       x++;   // add 1 to x
}
```

For loops

Start at 0, keep going while *i* is less than 5 and add one to *i* each time around.

```
for(var i = 0; i<5; i++) {
       // the value of i goes from 0 to 5
}
```

Start at 0, keep going while *i* is less than or equal to 10 and add 2 to *i* each time

```
for(var i = 0; i<=10; i+=2) {
       // the value of i goes from 0 to 10 stepping by 2 every time
}
```

Start at 100, keep going while *i* is greater than 0 and subtract 1 from *i* each time.

```
for(var i = 100; i>0; i--) {
       // the value of i goes from 100 to 1
}
```

Functions

Make a custom function that draws 2 circles :

```
function drawCircles() {
	c.fillCircle(40, 20, 10);
	c.fillCircle(80, 20, 10);
}
```

Call that function:

```
drawCircles();
```

Define a function that draws 2 circles at a specific y position

```
function drawCirclesAtY(yposition) {
	c.fillCircle(40, yposition, 10);
	c.fillCircle(80, yposition, 10);
}
```

Call the function and pass 100 into it

```
drawCirclesAtY(100);  // circles are drawn with a y position of 100
```

Define a function that adds two numbers together and returns the result

```
function addNumbers (num1, num2) {
	return num1 + num2;
}
```

Call the function and store the result in a variable

```
var sum = addNumbers(5, 10);  // sum variable now holds 15
```

if statements (conditionals)

If x is equal to 5

```
if(x==5) {
       // code between these brackets is run if x is equal to 5.
       // note the double equals == to compare, as opposed to single equals to set.
}
```

Example using if and else

```
if(name == 'John') {
       message = “Hello John”;
} else {
       message = “Hello stranger”;
}
```

Example using if and else if

```
if(numFriends > 100) {
       message = “You're really popular!”;
} else if (numFriends > 50) {
       message = “You're quite popular.”;
} else {
       message = “Loser.”;
}
```

Testing for more than one thing - && (AND) and || (OR)

If *mouseX* is greater than 100 AND *mouseX* is less than 200:

```
if((mouseX > 100) && (mouseX<200)) {
       message = “mouse is in the middle”;
}
```

If *mouseX* is greater than 100 OR *mouseY* is greater than 100:

```
if((mouseX > 100) || (mouseY > 100)) {
       message = “mouse is bottom or right”;
}
```

Full list of comparisons :

==	is equal to
!=	is not equal to
<	is less than
>	is greater than
<=	is less than or equal to
>=	is greater than or equal to

Testing boolean variables

`if(mouseDown)` is the same as `if(mouseDown == true)`

`if(!mouseDown)` is the same as `if(mouseDown == false)`

Arrays

Make a new array

```
var favouriteNumbers = [3, 7, 21];
var friends = ['John', 'Sarah', 'Mike', 'Alex'];
var particles = [];    // an empty array
```

Access elements in an array

```
var firstFriend = friends[0]; // the first element in the array - 'John'
var lastFriend = friends[3];  // 'Alex'
```

Find how many items are in the array

```
var numFriends = friends.length;      // numFriends is now 4
```

Push an item onto the end of an array

```
friends.push('Anna'); // array now contains 'John', 'Sarah', 'Mike', 'Alex', 'Anna'
```

Take an item off the end of an array

```
friends.pop();                        // array now contains 'John', 'Sarah', 'Mike', 'Alex'
var myRemovedFriend = friends.pop()   // array now contains 'John', 'Sarah', 'Mike' and
                                      // myRemovedFriend is 'Alex'
```

Add an item onto the front of an array

```
friends.unshift('Anna');              // array now contains 'Anna', 'John', 'Sarah', 'Mike'
```

Take an item off the front of an array

```
friends.shift();                              // array now contains 'John', 'Sarah', 'Mike'
var myRemovedFriend = friends.shift();        // array now contains 'Sarah', 'Mike' and
                                              // myRemovedFriend is 'John'
```

Objects

Make a custom object:

```
var person = { name : 'Val', eyes : 'hazel', hair : 'brown', age : 32 };
```

Access elements in the object :

```
console.log(person.name);     // outputs 'Val'
person.age++                  // age is now 33
```

Add new element to the object

```
person.shoeSize = 8;
```

More Graphics

Drawing text into the canvas

`c.fillText(text, x, y);`	draws the text at the x and y position specified
`c.font = 'sizepx fontname';`	change the default font and size

Drawing images into the canvas

`c.drawImage(image, x, y);`	draws the image at full size with the top left at *x* and *y*
`c.drawImage(image, x, y, width, height);`	add *width* and height to specify the *image* size

Draw part of an image

```
c.drawImage(image, sx, sy, swidth, sheight, x, y, width, height);
```

sx, *sy*, *swidth* and *sheight* are the position and size of the part of the image that you want to draw.

Coordinate transformations

`c.translate(x, y);`	Move the origin by x pixels along and y pixels down
`c.rotate(angle);`	Rotate the coordinate system around the origin by *angle* radians
`c.scale(xscale, yscale);`	Scale the coordinate system. *xscale* and *yscale* are unit values, in other words, 1 is 100%, and 0.5 is 50%.

Save and restore the draw state

```
c.save();
c.restore();
```

CreativeJS 'training wheels'

Here are some useful functions that I've provided - they're not built in to JavaScript. All you need to do is include the creativejs.js script in your html body tag and the functionailty will be available to you.

```
<script src='js/creativejs.js'></script>
```

The draw function.

```
function draw(){
        // I've added code that automatically calls this function 60 times a second
}
```

Mouse position and button state

`mouseX`	The current x position of the mouse
`mouseY`	The current y position of the mouse
`lastMouseX`	The previous x position of the mouse
`lastMouseY`	The previous y position of the mouse

`mouseDown`	is true if the left mouse button is currently pressed

The onMouseDown function.

```
function onMouseDown(){
        // I've added code that calls this function once when the mouse is pressed
}
```

Keyboard interactions

```
function onKeyDown(e){
        // I've added code that calls this function once when any key is pressed
        // find the code of the key that was pressed with e.keyCode
}
```

Random

`random()`	returns a random value between 0 and 1
`random(max)`	returns a random value between 0 and *max*
`random(min, max)`	returns a random value between *min* and *max*

`randomInteger(max)`	returns a random integer (whole number) between 0 and *max*
`randomInteger(min, max)`	returns a random integer (whole number) between *min* and *max*

If you'd rather use the built-in random function use :

`Math.random()`	returns a random value between 0 and 1

Angles - conversion between degrees and radians

`degrees(radians);`	returns *radians* converted to degrees
`radians(degrees);`	returns *degrees* converted to radians

Map and clamp functions.

The *map* function converts a number between a range from *min1* to *max1* on to the range between *min2* and *max2*.

```
map(value, min1, max1, min2, max2);
```

`map(5, 0, 10, 0, 100);`	map 5 between 0 and 10 on to a range from 0 to 100 - returns 50
`map(20, 10, 30, 0, 100);`	map 20 between 10 and 30 on to a range from 0 to 100 - returns 50
`map(0.5, 0, 1, 50, 100);`	map 0.5 between 0 and 1 on to a range from 50 to 100 - returns 75

The *clamp* function limits one value between minimum and maximum values.

`clamp(value, min, max);`	if *value* is less than *min*, return *min*, if *value* is greater than *max*, return *max*, otherwise, return *value*
`clamp(5, 0, 10);`	returns 5
`clamp(5, 10, 20);`	returns 10
`clamp(30, 10, 20);`	returns 20

More maths

`Math.round(number);`	returns the nearest integer (whole number) to *number*
`Math.floor(number);`	returns *number* rounded down to the nearest integer
`Math.ceil(number);`	returns *number* rounded up to the nearest integer

`Math.max(number1, number2);`	returns the number that is the largest
`Math.min(number1, number2);`	returns the number that is the smallest

`Math.sqrt(number);`	returns the square root of *number*

Modulus (%)

The modulus operator returns what is left over when the first number is divided by the second

`24 % 10;`	returns 4: 24 divided by 10 is 2 with 4 left over
`18 % 5;`	returns 3: 18 divided by 5 is 3 with 3 left over
`3 % 2;`	returns 1
`4 % 2;`	returns 0 - any odd number % 2 is 1, and any even number % 2 is 0