ΕΛΛΗΝΙΚΟ ΜΕΣΟΓΕΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

Διαδικτυακά Πολυμέσα και Γραφικά

Canvas

Στελιος Σφακιανάκης

Χειμερινό 2021

Τι είναι;

- Ο HTML5 Canvas είναι μια bitmap περιοχή μέσα σε ένα έγγραφο HTML που εισάγεται με το στοιχείο `<canvas>`.

- Ο προγραμματισμός του τι εμφανίζεται μέσα στην περιοχή του καμβά καθορίζεται αποκλειστικά μέσω του Javascript API (application programming interface)

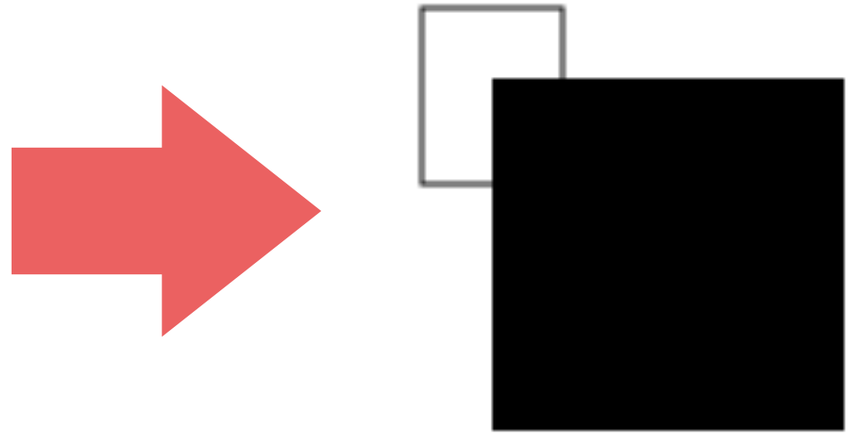
  - ..σε αντίθεση με το **SVG** (Scalable Vector Graphics) το οποίο είναι markup (όπως η HTML και η XML)

- Χρησιμοποιείται κυρίως για παιχνίδια, γραφικά, επεξεργασία εικόνων/φωτογραφιών στο Web, κλπ δηλαδή εργασίες που χειρίζονται pixels

  - Κυρίως 2D αλλά και 3D μέσω WebGL!

Παράδειγμα

```html
<html>
  <body>
    <canvas width="200" height="200"></canvas>
    <script>
      let cnv = document.querySelector('canvas');
      let ctx = cnv.getContext('2d');
      ctx.strokeRect(10,10, 40, 50);
      ctx.fillRect(30,30, 100, 100);
    </script>
  </body>
</html>
```



Ο καμβάς είναι ..low level

- Ο προγραμματιστής καθορίζει μέσω Javascript τη θέση σε συντεταγμένες `x`, `y` pixels των αντικειμένων που θέλει να 📌 εμφανιστούν στο επόμενο *frame*

Θα χρειαστούμε Μαθηματικά!!

  - Αντικείμενα: ευθείες γραμμές, καμπύλες, τόξο (π.χ. κύκλοι), παραλληλόγραμμα, κείμενο, εικόνες
    - Επίσης: χρώματα (περίγραμμα, "γέμισμα", διαβαθμίσεις), μετασχηματισμοί, ...

- Ότι αντικείμενα βάλουμε στον καμβά δεν υπάρχουν στο DOM

- Αν θέλουμε animation πρέπει να το κάνουμε μόνοι μας (ή μέσω βιβλιοθήκης που έφτιαξε άλλος) τοποθετώντας τα αντικείμενα στη νέα τους θέση πριν το επόμενο frame!

Υποστήριξη σε Browsers: 💕

Canvas (basic support) - LS

Method of generating fast, dynamic graphics using JavaScript.

Current aligned | Usage relative | Date relative | Filtered | All

Usage: Global — % of all users: 98.68% + 1.11% = 99.79%

IE	Edge *	Firefox	Chrome	Safari	Opera	Safari on iOS *	Opera Mini *	Android Browser *	Opera Mobile *	Chrome for Android	Firefox for Android	UC Browser for Android	Samsung Internet	QQ Browser	Baidu Browser	KaiOS Browser
6-8		1 2-3.5		1 3.1-3.2				1 2.1-2.3								
9-10	12-94	3.6-93	4-94	4-14.1	10-80	3.2-14.8		3-4.4.4	12-12.1				4-14.0			
11	95	94	95	15	81	15	2 all	95	64	95	92	12.12	15.0	10.4	7.12	2.5
		95-96	96-98	TP												

https://caniuse.com/canvas

Το στοιχείο <canvas>

```
<canvas id="canvasOne" width="500" height="300">
Your browser does not support HTML5 Canvas.
</canvas>
```

- Το χρειαζόμαστε πάντα. Μπορούμε να το έχουμε μέσα στο HTML ή να το δημιουργούμε δυναμικά με Javascript (δηλ. με `createElement` και π.χ. `appendChild`)

```
let canvas = document.createElement('canvas');
canvas.width = 500;
canvas.height = 300;
document.body.appendChild(canvas);
```

- Καλό είναι να του δίνουμε τις διαστάσεις (πλάτος, ύψος) μέσω των `width` και `height`. Default: 300 x 150 pixels

- Προσοχή: Αν οποιαδήποτε στιγμή θέσουμε τιμές στο `width` ή `height` property του canvas DOM object θα σβηστεί οτιδήποτε υπάρχει στον καμβά!

Το Context object

- Όλη η αλληλεπίδραση με τον καμβά γίνεται μέσω του context object, το οποίο το παίρνουμε από το canvas DOM object :

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

- Η παράμετρος "2d" σημαίνει ότι θέλουμε το διδιάστατο rendering context, που είναι instance του `CanvasRenderingContext2D`
  - Αν δώσουμε το "webgl" ως όρισμα παίρνουμε το 3διάστατο WebGL context (`WebGLRenderingContext`), που δεν θα μας απασχολήσει προς το παρόν

- Με το context στα χέρια μας, μπορούμε να .."ζωγραφίσουμε"!

```
context.fillStyle = "#ffffaa";
context.fillRect(0, 0, 500, 300);
```

Οι 2 εντολές..

- Στην πλειονότητα των περιπτώσεων, ξεκινάμε πάντοτε το προγραμματισμό με τις ακόλουθες 2 εντολές :

```
let canvas = document.getElementById('canvasid');
let context = canvas.getContext('2d');
```

- Όπου "canvasid" είναι το id του <canvas> element που έχουμε βάλει στο HTML έγγραφο μας

Ορθογώνια Παραλληλόγραμμα ("κουτιά")

- Το `strokeRect(x,y,width,height)` ζωγραφίζει ένα ορθ. παραλληλόγραμμο με την πάνω αριστερή γωνία να είναι στο (x,y) και να έχει πλάτος `width` και ύψος `height`

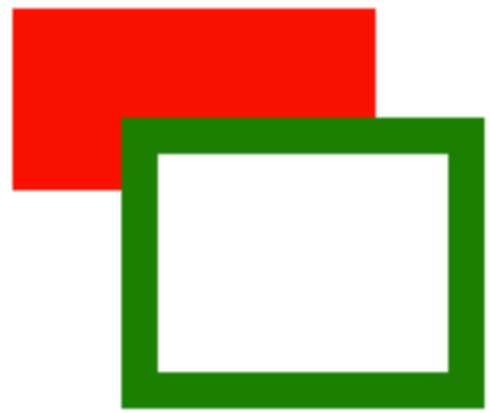
- Το `fillRect(x,y,width,height)` κάνει το ίδιο, αλλά γεμίζει το παραλ/μο με χρώμα

- Το `clearRect(x,y,width,height)` σβήνει ότι υπάρχει στο παραλ/μο

- Το χρώμα περιγράμματος (για το stroke) και "γεμίσματος" (για το fill) καθορίζονται από το `strokeStyle` και `fillStyle` αντίστοιχα

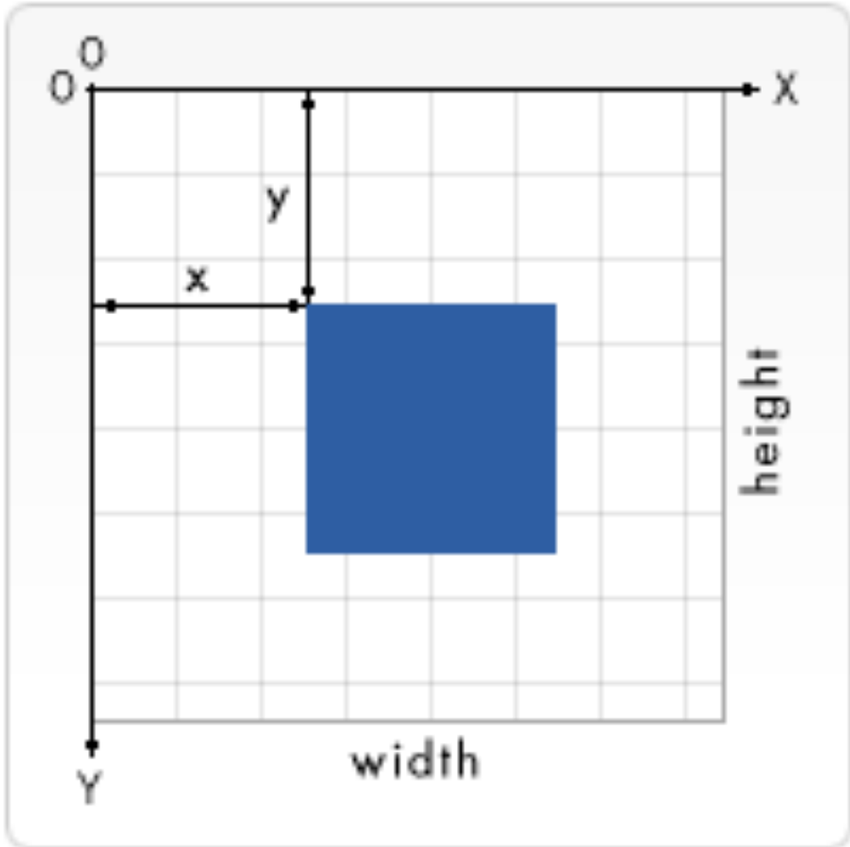
Παράδειγμα

```
context.fillStyle = 'red';
context.fillRect(10,10, 100, 50);

context.fillStyle = 'green';
context.fillRect(40,40, 100, 80);

context.clearRect(50,50, 80, 60);
```



Συντεταγμένες

- Οι συντεταγμένες αναφέρονται πάντα σε pixels
- Η αρχή των αξόνων (όπως ισχύει παραδοσιακά στους υπολογιστές) είναι στην πάνω αριστερή γωνία του καμβά
- Το `x` αυξάνει προς τα δεξιά και το `y` προς τα κάτω!
- Θα δούμε αργότερα πώς με μετασχηματισμούς μπορούμε να χρησιμοποιήσουμε διαφορετικό σύστημα συντεταγμένων



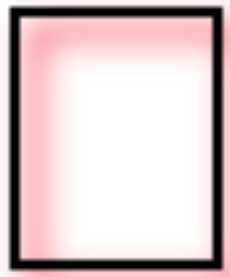
Μονοπάτια

- Το άλλο βασικό αντικείμενο μετά τα "κουτιά" είναι τα μονοπάτια
- Είναι λίστα από σημεία ενωμένα με γραμμές που μπορούν να έχουν διαφορετικό σχήμα, καμπύλες ή ευθείες, διαφορετικό πάχος (πλάτος) και χρώμα
- Προσοχή: Για να φτιάξουμε ένα μονοπάτι κάνουμε τα εξής:
  - Ξεκινάμε ένα νέο μονοπάτι με την `beginPath()`
  - Προσδιορίζουμε τι σημεία θα περιέχει και πώς θα είναι ενωμένα
  - Τελικά το "ζωγραφίζουμε" με την εντολή `stroke()` για γραμμή ή την `fill()` για "γέμισμα" με χρώμα

Εντολές για μονοπάτια

- `moveTo(x,y)`: μετακίνηση στο (x,y), ορισμός του ως τρέχον σημείο
- `lineTo(x,y)`: ευθεία γραμμή από το τρέχον σημείο στο (x,y)
- `arc(x,y,radius,startAngle,endAngle)`: τόξο κύκλου με κέντρο το (x,y), ακτίνα το radius, αρχική γωνία και τελική γωνία
  - Οι γωνίες μετριούνται σε ακτίνια (radians), π.χ. `Math.PI/2` (δηλ. π/2) για τις 90 μοίρες
- `bezierCurveTo(cp1x, cp1y, cp2x, cp2y, x, y)`: κυβική καμπύλη Bézier με αρχή το τρέχον σημείο, τελικό το (x,y) και σημεία ελέγχου (cp1x, cp1y) και (cp2x, cp2y)
- `quadraticCurveTo(cpx, cpy, x, y)`: τετραγωνική καμπύλη Bézier

Πάχος γραμμής και Σκιές



```
ctx.lineWidth = 2;      // Πάχος γραμμής : 2 pixels

// Σκιά:
ctx.shadowOffsetX = 4; // θέση σκιάς: 4 pixels πιο δεξιά
ctx.shadowOffsetY = 4; // θέση σκιάς: 4 pixels πιο χαμηλά
ctx.shadowBlur    = 7; // "Θαμπάδα"
ctx.shadowColor   = "red"; // Χρώμα σκιάς

ctx.strokeRect(10, 10, 40, 50);
```

```
ctx.shadowColor = 'red';
ctx.shadowBlur = 100;

ctx.arc(200, 200, 40, 0, 2*Math.PI);
ctx.fill();
```

"Καπέλα" (caps) σε γραμμές

- Το `lineCap` μπορεί να πάρει τις τιμες `butt`, `round`, kai `square`

- `butt` σημαίνει όχι έξτρα καπέλο

- Το `round` σημαίνει ότι σχηματίζεται ένα ημικύκλιο με διάμετρο όσο το `lineWidth` και στις δύο άκρες της γραμμής.

- Το `square` σχηματίζει παραλληλόγραμμο με πλάτος όσο το `lineWidth` και μήκος όσο το μισό του, και στις δύο άκρες της γραμμής.

```
ctx.lineWidth=20

ctx.beginPath();
ctx.strokeStyle="orange";
ctx.lineCap="butt";
ctx.moveTo(30, 30);
ctx.lineTo(300,  30);
ctx.stroke();

ctx.beginPath();
ctx.strokeStyle="crimson";
ctx.lineCap="round";
ctx.moveTo(30, 80);
ctx.lineTo(300,  80);
ctx.stroke();

ctx.beginPath();
ctx.strokeStyle="teal";
ctx.lineCap="square";
ctx.moveTo(30, 130);
ctx.lineTo(300,  130);
ctx.stroke();
```

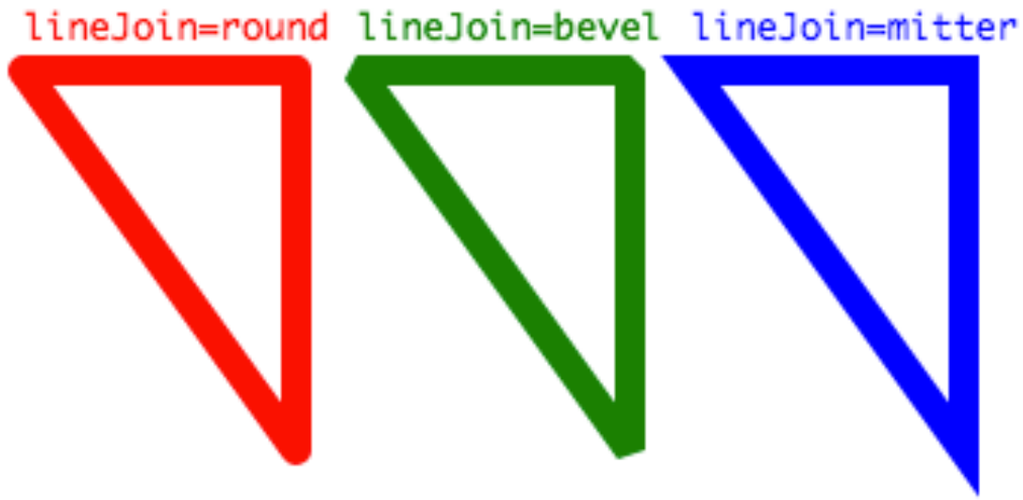
Ενώσεις γραμμών

- Το `lineJoin` μπορεί να παρει τις τιμες `round`, `bevel`, kai `miter`

- `round` σημαίνει "στρογγυλεμένες" ενώσεις

- `bevel` : διαγώνιες ακμές στις ενώσεις

- `miter` είναι το default και κάνει τις ενώσεις "μυτερές" :-)




```
ctx.lineJoin  = 'round';
ctx.lineWidth = 10;
ctx.strokeStyle = 'red';
ctx.beginPath();
ctx.moveTo(10 , 100);
ctx.lineTo(100, 100);
ctx.lineTo(100, 225);
ctx.closePath();
ctx.stroke();
```

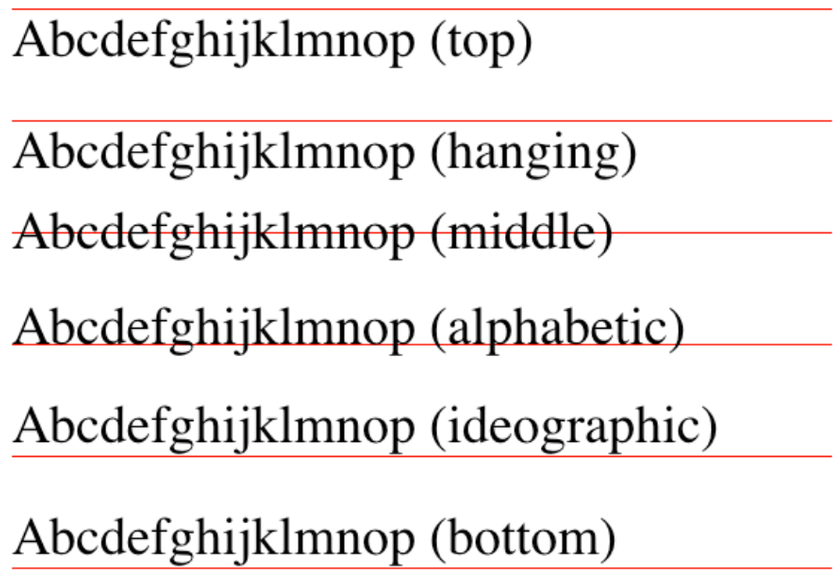
Κείμενο

```
ctx.font          = "18px Sans-Serif";
ctx.textAlign     = "center";
ctx.fillStyle     = "blue";
ctx.fillText  ("HELLO", 150, 200);
```

- `fillText(text, x, y)`: "γεμίζει" τα γράμματα του text με το τρέχον `fillStyle`. Το κείμενο εμφανίζεται στο σημείο (x,y) βάσει των ακόλουθων properties του context:
  - `font`: Ποια CSS γραμματοσειρά και μέγεθος θα χρησιμοποιηθεί (Default: "10px sans-serif")
  - `textAlign`: οριζόντια "ευθυγράμμιση" (alignment) του κειμένου (π.χ. `"center"`, `"left"`, `"right"`)
  - `textBaseline` : κάθετη "ευθυγράμμιση" (alignment) του κειμένου (π.χ. `"top"`, `"bottom"`, `"middle"`)
- Σε κάποιες περιπτώσεις είναι χρήσιμη `measureText(text)` που μας επιστρέφει σε ένα αντικείμενο πληροφορίες για τις διαστάσεις του κειμένου, π.χ. το `width`, βάσει του τρέχοντος `font`.

Παραδειγμα με textBaseline

```
const canvas =
document.getElementById('canvas');
const ctx = canvas.getContext('2d');

const baselines = ['top', 'hanging',
'middle', 'alphabetic', 'ideographic',
'bottom'];
ctx.font = '36px serif';
ctx.strokeStyle = 'red';

baselines.forEach(function (baseline,
index) {
  ctx.textBaseline = baseline;
  const y = 75 + index * 75;
  ctx.beginPath();
  ctx.moveTo(0, y + 0.5);
  ctx.lineTo(550, y + 0.5);
  ctx.stroke();
  ctx.fillText('Abcdefghijklmnop (' +
baseline + ')', 0, y);
});
```



https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/textBaseline

Χρώμα

- Για τον προσδιορισμό του χρώματος (π.χ. για τα strokeStyle, fillStyle) χρησιμοποιείται το "συντακτικό" του CSS, δηλ μπορούμε να δώσουμε:
  - Το όνομα του π.χ. `red`
  - Την "Red-Green-Blue" (RGB) τιμή του, π.χ. `rgb(243, 243, 160)`, ή την αντίστοιχη δεκαεξαδική τιμή, π.χ. `#f5f5dc`

https://developer.mozilla.org/en-US/docs/Web/CSS/color_value

Named	Numeric	Color name	Hex rgb	Decimal
		aliceblue	#F0F8FF	240,248,255
		antiquewhite	#FAEBD7	250,235,215
		aqua	#00FFFF	0,255,255
		aquamarine	#7FFFD4	127,255,212
		azure	#F0FFFF	240,255,255
		beige	#F5F5DC	245,245,220
		bisque	#FFE4C4	255,228,196
		black	#000000	0,0,0
		blanchedalmond	#FFEBCD	255,235,205
		blue	#0000FF	0,0,255
		blueviolet	#8A2BE2	138,43,226
		brown	#A52A2A	165,42,42
		burlywood	#DEB887	222,184,135
		cadetblue	#5F9EA0	95,158,160
		chartreuse	#7FFF00	127,255,0
		chocolate	#D2691E	210,105,30
		coral	#FF7F50	255,127,80
		cornflowerblue	#6495ED	100,149,237
		cornsilk	#FFF8DC	255,248,220
		crimson	#DC143C	220,20,60
		cyan	#00FFFF	0,255,255
		darkblue	#00008B	0,0,139
		darkcyan	#008B8B	0,139,139
		darkgoldenrod	#B8860B	184,134,11
		darkgray	#A9A9A9	169,169,169
		darkkhaki	#BDB76B	189,183,107
		darkmagenta	#8B008B	139,0,139
		darkolivegreen	#556B2F	85,107,47
		darkorange	#FF8C00	255,140,0
		darkorchid	#9932CC	153,50,204
		darkred	#8B0000	139,0,0
		darksalmon	#E9967A	233,150,122
		darkseagreen	#8FBC8F	143,188,143
		darkslateblue	#483D8B	72,61,139
		darkslategray	#2F4F4F	47,79,79
		darkslategrey	#2F4F4F	47,79,79
		darkturquoise	#00CED1	0,206,209
		darkviolet	#9400D3	148,0,211
		deeppink	#FF1493	255,20,147
		deepskyblue	#00BFFF	0,191,255
		dimgray	#696969	105,105,105
		dimgrey	#696969	105,105,105
		dodgerblue	#1E90FF	30,144,255
		firebrick	#B22222	178,34,34
		floralwhite	#FFFAF0	255,250,240
		forestgreen	#228B22	34,139,34
		fuchsia	#FF00FF	255,0,255
		gainsboro	#DCDCDC	220,220,220
		ghostwhite	#F8F8FF	248,248,255
		gold	#FFD700	255,215,0
		goldenrod	#DAA520	218,165,32
		gray	#808080	128,128,128
		green	#008000	0,128,0
		greenyellow	#ADFF2F	173,255,47
		grey	#808080	128,128,128
		honeydew	#F0FFF0	240,255,240
		hotpink	#FF69B4	255,105,180
		indianred	#CD5C5C	205,92,92
		indigo	#4B0082	75,0,130
		ivory	#FFFFF0	255,255,240
		khaki	#F0E68C	240,230,140
		lavender	#E6E6FA	230,230,250
		lavenderblush	#FFF0F5	255,240,245
		lawngreen	#7CFC00	124,252,0
		lemonchiffon	#FFFACD	255,250,205
		lightblue	#ADD8E6	173,216,230
		lightcoral	#F08080	240,128,128
		lightcyan	#E0FFFF	224,255,255
		lightgoldenrodyellow	#FAFAD2	250,250,210
		lightgray	#D3D3D3	211,211,211
		lightgreen	#90EE90	144,238,144
		lightgrey	#D3D3D3	211,211,211
		lightpink	#FFB6C1	255,182,193
		lightsalmon	#FFA07A	255,160,122
		lightseagreen	#20B2AA	32,178,170
		lightskyblue	#87CEFA	135,206,250
		lightslategray	#778899	119,136,153

https://drafts.csswg.org/css-color-3/#svg-color

Χρώματα RGB
(“red-green-blue”)

000000	000033	000066	000099	0000CC	0000FF
003300	003333	003366	003399	0033CC	0033FF
006600	006633	006666	006699	0066CC	0066FF
009900	009933	009966	009999	0099CC	0099FF
00CC00	00CC33	00CC66	00CC99	00CCCC	00CCFF
00FF00	00FF33	00FF66	00FF99	00FFCC	00FFFF
330000	330033	330066	330099	3300CC	3300FF
333300	333333	333366	333399	3333CC	3333FF
336600	336633	336666	336699	3366CC	3366FF
339900	339933	339966	339999	3399CC	3399FF
33CC00	33CC33	33CC66	33CC99	33CCCC	33CCFF
33FF00	33FF33	33FF66	33FF99	33FFCC	33FFFF
660000	660033	660066	660099	6600CC	6600FF
663300	663333	663366	663399	6633CC	6633FF
666600	666633	666666	666699	6666CC	6666FF
669900	669933	669966	669999	6699CC	6699FF
66CC00	66CC33	66CC66	66CC99	66CCCC	66CCFF
66FF00	66FF33	66FF66	66FF99	66FFCC	66FFFF
990000	990033	990066	990099	9900CC	9900FF
993300	993333	993366	993399	9933CC	9933FF

996600	996633	996666	996699	9966CC	9966FF
999900	999933	999966	999999	9999CC	9999FF
99CC00	99CC33	99CC66	99CC99	99CCCC	99CCFF
99FF00	99FF33	99FF66	99FF99	99FFCC	99FFFF
CC0000	CC0033	CC0066	CC0099	CC00CC	CC00FF
CC3300	CC3333	CC3366	CC3399	CC33CC	CC33FF
CC6600	CC6633	CC6666	CC6699	CC66CC	CC66FF
CC9900	CC9933	CC9966	CC9999	CC99CC	CC99FF
CCCC00	CCCC33	CCCC66	CCCC99	CCCCCC	CCCCFF
CCFF00	CCFF33	CCFF66	CCFF99	CCFFCC	CCFFFF
FF0000	FF0033	FF0066	FF0099	FF00CC	FF00FF
FF3300	FF3333	FF3366	FF3399	FF33CC	FF33FF
FF6600	FF6633	FF6666	FF6699	FF66CC	FF66FF
FF9900	FF9933	FF9966	FF9999	FF99CC	FF99FF
FFCC00	FFCC33	FFCC66	FFCC99	FFCCCC	FFCCFF
FFFF00	FFFF33	FFFF66	FFFF99	FFFFCC	FFFFFF

- 6 δεκαεξαδικά ψηφία, τα 2 πρώτα για το κόκκινο, τα 2 επόμενα για πράσινο, και τα 2 τελευταία για το μπλε

- Με 2 16αδικά ψηφία έχουμε μέχρι 256 διαφορετικές αποχρώσεις, άρα συνολικα 256 x 256 x 256 = 16.777.216 διαφορετικά χρώματα!

Animation στον καμβά

- Βασιζόμαστε στη οπτική ψευδαίσθηση όπου στατικές εικόνες διαδέχονται γρήγορα η μία την άλλη (“βήτα κίνηση”)

- Η βασική ιδέα λοιπόν είναι ότι για να δείξουμε κίνηση στο καμβά “ζωγραφίζουμε” στον καμβά και “πολύ γρήγορα” σβήνουμε τον καμβα και ξαναζωγραφίζουμε μια ελαφρώς διαφοροποιημένη εικόνα

- Με τη σωστή ταχύτητα εναλλαγής των εικόνων και την κατάλληλη ενημέρωση του περιεχομένου των εικόνων μπορούμε να έχουμε ενα πολύ καλό animation!

Animation: 3 συναρτήσεις

- Η ενημέρωση των περιεχομένων του καμβά για το animation effect θα πρέπει να γίνεται ξανα και ξανα για όσο χρόνο διαρκεί το animation.

- Αυτή η επαναληπτική διαδικασία δεν μπορεί να γίνει με ένα κλασσικό loop (π.χ. με `while`) (Γιατί;;)

- Αντίθετα, θα πρέπει να χρησιμοποιήσουμε κάποια από τις παρακάτω συναρτήσεις:

- `setInterval(<func>, <delay>)`: Προγραμματιζει την εκτέλεση της συνάρτησης `<func>` επαναληπτικά (ξανα και ξανά) με `<delay>` milliseconds καθυστέρηση μεταξύ των διαδοχικών κλήσεων

- `setTimeout(<func>, <delay>)`: Προγραμματιζει την εκτέλεση της συνάρτησης `<func>` μια φορά μετα από `<delay>` milliseconds

- `requestAnimationFrame(<func>)`: Προγραμματιζει την εκτέλεση της συνάρτησης `<func>` μια φορά πριν το επόμενο “repaint” που κάνει ο browser στα περιεχόμενα της σελίδας. Πότε θα κληθεί η συνάρτηση δεν είναι δεδομένο, π.χ. εξαρτάται από το refresh rate της οθόνης.

Γενικά, προτιμούμε αυτήν!!

Γενική δομή κώδικα για animation στον καμβα

- Κρατάμε τα δεδομένα κατάστασης για την απεικόνιση στον καμβα σε κατάλληλες δομές

- Μια συνάρτηση (ή μέρος κώδικα) για την αρχικοποίηση αυτών των δεδομένων

- Μια συνάρτηση για το “ζωγράφισμα” στον καμβά βασει των τρέχοντων δεδομένων κατάστασης, που κάνει τα εξής 3 βήματα:

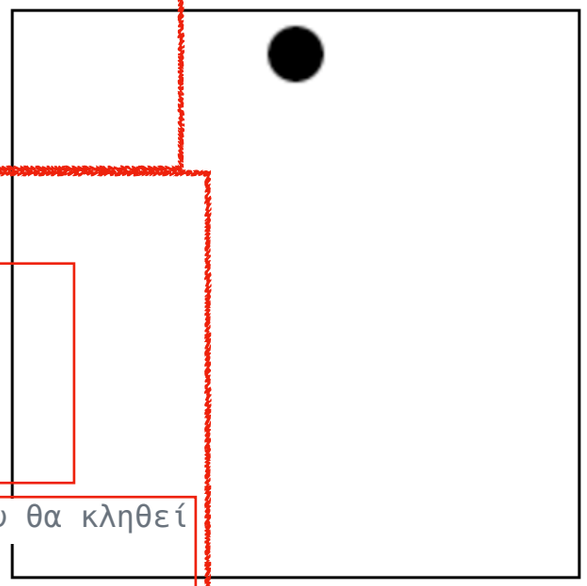
  - “Ζωγραφίζει” στον καμβά με βαση την τρέχουσα κατάσταση

  - Ενημερώνει τα δεδομένα κατάστασης ώστε να απεικονίζουν την κατάσταση στο επόμενο “frame”

  - Καλεί τη `requestAnimationFrame` (ή τη `setTimeout`) δίνοντας ως συνάρτηση τον “εαυτό” της

Παράδειγμα: σφαίρα που κινείται πάνω - κάτω

```
// Τα δεδομένα κατάστασης: x, y είναι οι συνεταταγμένες του κέντρου της
// ενώ dir είναι η κατευθυνση κίνησης, 'up' (πάνω) ή 'down' (κάτω):
let ball = {x: cnv.width/2, y: cnv.height, dir: 'up'};

// Η συνάρτηση για το animation:
function draw() {
    // 1. Ζωγραφίζουμε τη σφαίρα στην τρέχουσα θέση (x,y) που έχουμε κρατήσει στο αντικείμενο ball:
    cnv.width += 0; // Καθάρισμα του καμβα!
    ctx.beginPath();
    ctx.arc(ball.x, ball.y, r, 0, 2*Math.PI);
    ctx.fill();
    // 2. Ενημερώνουμε την "κάτασταση" της σφαίρας, δηλ. αλλάζουμε το y έτσι ώστε την επόμενη φορά που θα κληθεί
    // η συνάρτηση να ζωγραφιστεί σε μια διαφορετική θέση
    if (ball.dir === 'up') {
        // Αν η σφαίρα κινείται προς τα πάνω, μειώνουμε το y:
        ball.y -= dy;
        // Αν με τη μείωση του y η σφαίρα βγαίνει εκτός του πάνω ορίου του καμβα τότε αλλάζουμε φορά κίνησης
        // προς τα κάτω:
        if (ball.y - r <= 0)
            ball.dir = 'down';
    }
    else {
        // Αλλιώς, η σφαίρα κινείται προς τα κάτω, οπότε αυξάνουμε το y:
        ball.y += dy;
        // Αν με την αύξηση του y η σφαίρα βγαίνει εκτός του κάτω ορίου του καμβα τότε αλλάζουμε φορά κίνησης
        // προς τα πάνω:
        if (ball.y + r >= cnv.height)
            ball.dir = 'up';
    }
    // 3. Προγραμματίζουμε να ξανακληθει η συνάρτηση draw μετά από 20 milliseconds ή με το requestAnimationFrame:
    // setTimeout(draw, 20);
    requestAnimationFrame(draw);
}
// Πρώτη κλήση, για να ξεκινήσει το animation:
draw();
```



http://bit.ly/2S6L

Μετασχηματισμοί

- `translate(x,y)` : μετακίνηση της αρχής των συντεταγμένων στο σημείο (x,y)

- `rotate(angle)` : περιστροφή των αξόνων

- `scale(sx, sy)`: αλλάζει την κλίμακα ("scales") πολλαπλασιάζοντας τις συντεταγμένες. Αν δώσουμε αρνητικές τιμές κάνει "flip" (αντικατοπρισμό) με βάση τους άξονες

- Έχουν επίδραση σε όλες τις drawing εντολές που τις ακολουθούν

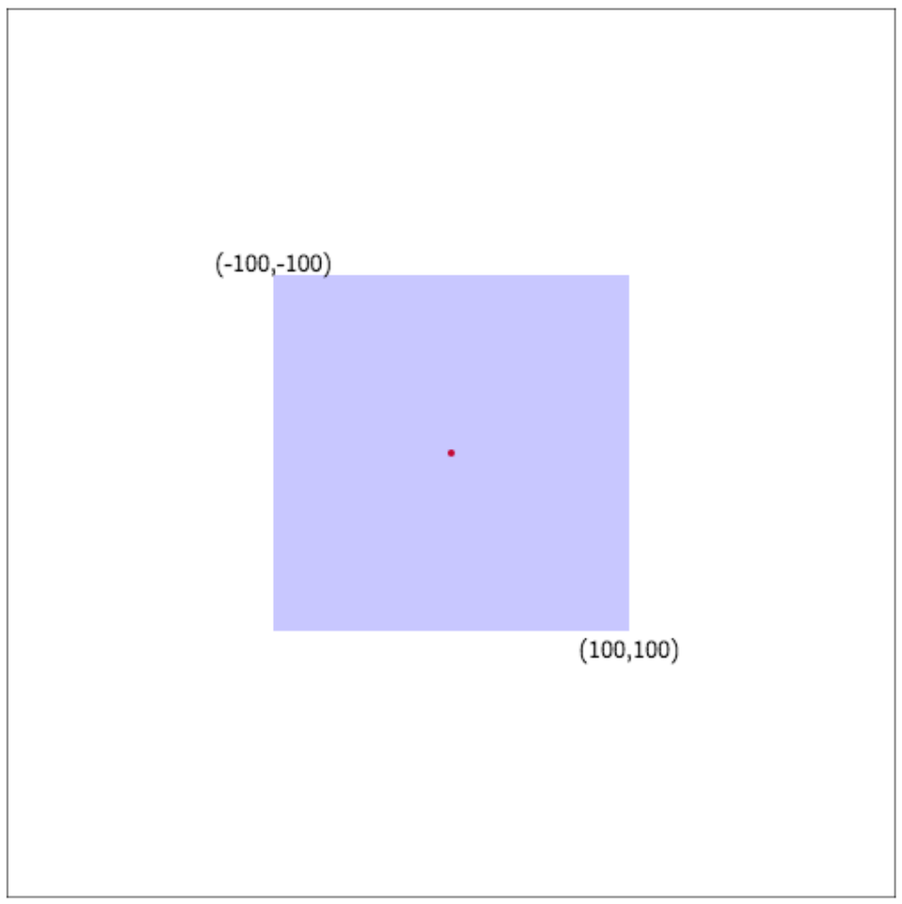
Μετασχηματισμοί Translate

- Μπορούμε να "μετακινήσουμε" το κέντρο των αξόνων για τις συντεταγμένες του καμβά σε όποιο σημείο (x,y) θέλουμε μέσα στον καμβά με την `translate(x,y)`
  - Π.χ. για είναι στο κέντρο του καμβά:

```
context.translate(canvas.width/2, canvas.height/2)
```

- Μετά από το translate όλες οι κλήσεις συναρτήσεων του context θα πρέπει να χρησιμοποιούν το νέο σύστημα συντεταγμένων
  - Αυτό σημαίνει ότι αν κάνουμε `context.moveTo(0,0)` μετά από το `context.translate(a,b)`, το τρέχον σημείο θα έχει συντεταγμένες (a,b) στο default σύστημα συντ/νων του καμβά

Παράδειγμα translate




```
ctx.translate(ctx.canvas.width/2, ctx.canvas.height/2);

ctx.beginPath();
ctx.fillStyle = 'red';
ctx.arc(0, 0, 2, 0, 2 * Math.PI); // center
ctx.fill();


ctx.fillStyle = "rgba(0, 0, 255, 0.2)";
let w = 200;
ctx.fillRect(-w/2, -w/2, w, w);
```

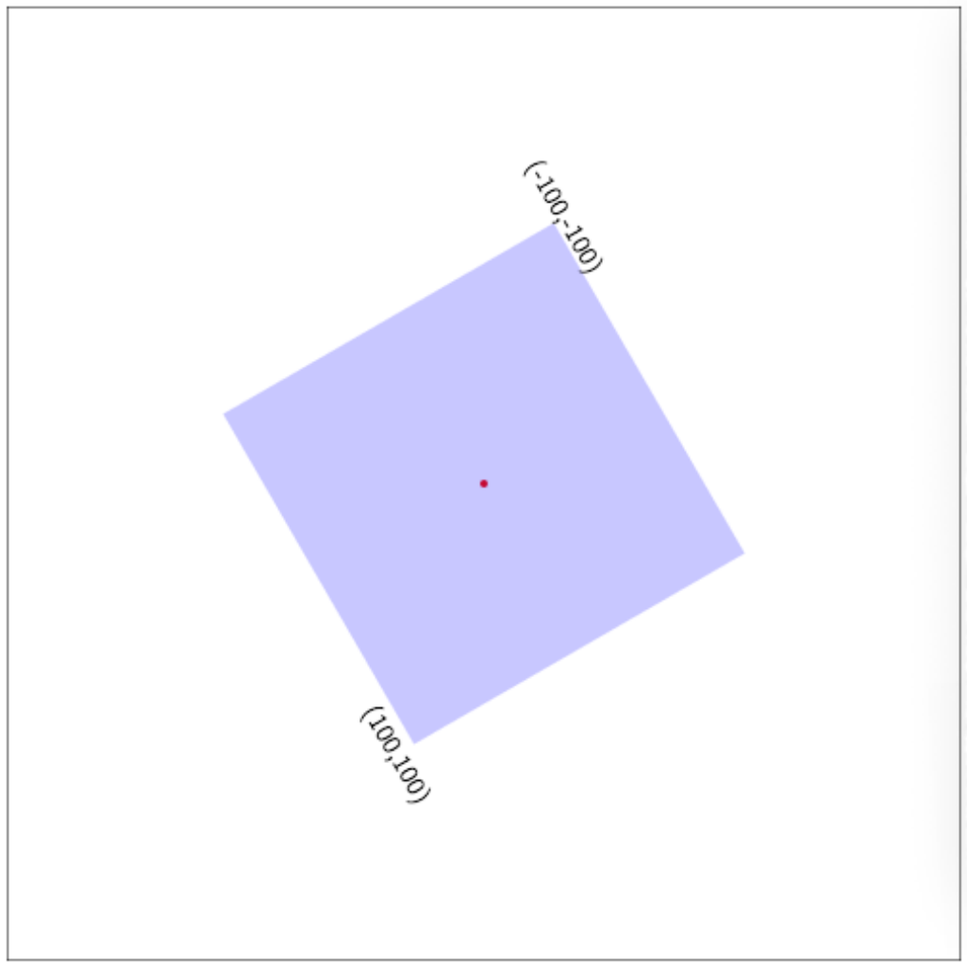
- Το τετράγωνο με πλάτος 200 pixels έχει το κέντρο του στο κέντρο του καμβά (κόκκινη τελεία). Η εντολή `fillRect` χρησιμοποιεί αρνητικές τιμές (-100, -100) για τις συντεταγμένες της πάνω αριστερής γωνίας του τετραγώνου

Περιστροφή

- Με την `rotate(angle)` μπορούμε να περιστρέψουμε το σύστημα συντεταγμένων κατά μία γωνία angle (σε ακτίνια, φυσικά)

- Αν θέλουμε να περιστρέψουμε ένα μόνο σχήμα μπορούμε να κάνουμε πρώτα ένα `translate` στο κέντρο του σχήματος

Παράδειγμα rotate




```
ctx.translate(ctx.canvas.width/2, ctx.canvas.height/2);

ctx.beginPath();
ctx.fillStyle = 'red';
ctx.arc(0, 0, 2, 0, 2 * Math.PI); // center
ctx.fill();

let angleInRadians = 60 * Math.PI / 180;
ctx.rotate(angleInRadians);

ctx.fillStyle = "rgba(0, 0, 255, 0.2)";
let w = 200;
ctx.fillRect(-w/2, -w/2, w, w);
```

Παράδειγμα rotate (2)

```
ctx.translate(ctx.canvas.width/2, ctx.canvas.height/2);

let theta = 10;
let w = 100;
for (let i = 0; i< 360/ theta; ++i) {
    let r = Math.floor(256 * Math.random());
    let g = Math.floor(256 * Math.random());
    let b = Math.floor(256 * Math.random());
    let color =  `rgba(${r}, ${g}, ${b}, 0.4)`;
    ctx.fillStyle = color;
    ctx.fillRect(0, 0, w, w);
    ctx.rotate( theta / 180 * Math.PI);
}
```

Πίνακας μετασχηματισμών

- Κάθε μετασχηματισμός (`translate`, `rotate`, `scale`) και συνδυασμοί τους υλοποιούνται με ένα πίνακα μετασχηματισμών που κρατιέται στο context object

- Οι σχετικές συναρτήσεις `getTransform` και `setTransform` ανακτούν και τροποποιούν τον πίνακα, αλλά δεν θα ασχοληθούμε περαιτέρω..

- Εκτός από την κλήση:

  ```
  ctx.setTransform(1, 0, 0, 1, 0, 0);
  ```

  που επί της ουσίας αναιρεί κάθε μετασχηματισμό και επαναφέρει τις default ρυθμίσεις

Τρέχουσα κατάσταση του καμβά: `save` και `restore`

- Ο χειρισμός του καμβά γίνεται μέσω του context object αλλά κάθε αλλαγή αναιρεί την προηγούμενη (π.χ. αν βάλουμε `lineWidth = 2` όλα τα strokes στη συνέχεια θα έχουν πάχος 2 pixel)

- Με τις `save()` και `restore()` μπορούμε να κάνουμε "τοπικές" αλλαγές στα settings του καμβά:

    - Η `save()` αποθηκεύει την τρέχουσα κατάσταση και η `restore()` επαναφέρει αυτή που είχε σωθεί τελευταία

    - Δουλεύουν όπως μια στοίβα (stack): Η `save()` προσθέτει στην κορυφή της στοίβας, και η `restore()` αφαιρεί και επαναφέρει την κατάσταση που είναι στην κορυφή.

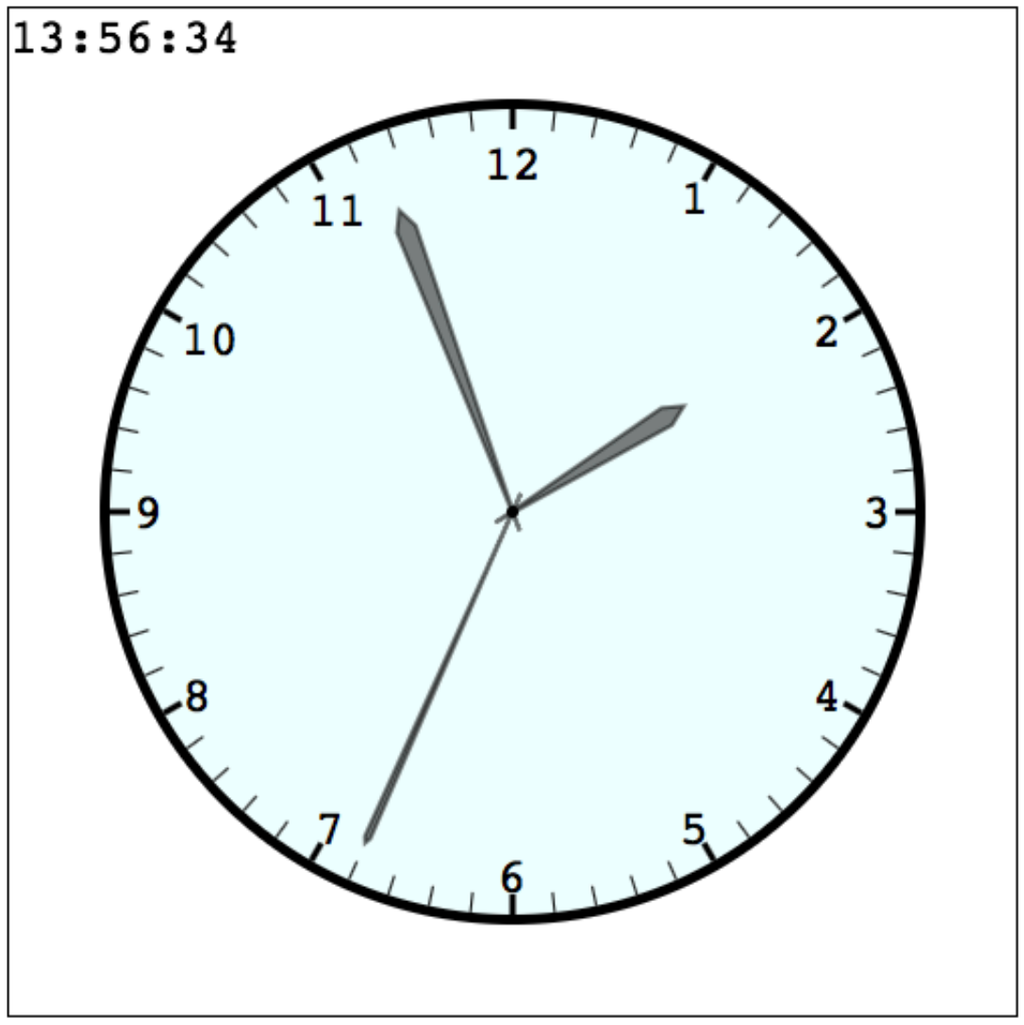
    - Οπότε πάντα πάνε "ζευγαράκι": κάθε `save()` πρέπει να έχει ένα `restore()`

- Η "κατάσταση" του καμβά περιλαμβάνει τους μετασχηματισμούς (π.χ. `translate`, `rotate`), τα styles (`strokeStyle`, `fillStyle`, `font`, `textAlign`, `textBaseline`), ρυθμίσεις για γραμμές (`lineWidth`, `lineCap`, `lineJoin`,...), σκιές (`shadowOffsetX`, `shadowOffsetY`, `shadowBlur`, `shadowColor`) κλπ.

Παράδειγμα: Ρολόι σε Καμβά

- Πλήρης κώδικας στο https://eclass.teicrete.gr/modules/document/file.php/TP319/ΘΕΩΡΙΑ/Ρολόι%20σε%20Καμβά/canvas_clock.html

- Εναλλακτικές υλοποιήσεις:
  - https://www.w3schools.com/graphics/canvas_clock.asp
  - https://dzone.com/articles/html5-canvas-clock-keeps
  - https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial/Basic_animations#An_animated_clock



Προσθήκη γραμμών για τα λεπτά περιφερειακά του κύκλου του ρολογιού. Έχουμε 60 λεπτά, άρα διαιρούμε τον κύκλο σε 2π/60 ή π/30 τόξα. Ξεκινάμε από το -π/2 για να βάλουμε το 12, επειδή οι γωνίες στον καμβά ξεκινάνε από την οριζόντια θέση και αυξάνουν ξεξιόστροφα (clockwise)

Ανά 5 γραμμές βάζουμε και το ψηφίο της ώρας

Κάνουμε translate στο κέντρο του ψηφίου. Για να το υπολογίσουμε καλούμε την measureText για μας επιστρέψει το πλάτος του ως string.

Θέλουμε το ψηφίο της ώρας να είναι σε οριζόντια θέση οπότε κάνουμε ένα αντίστροφο rotation πριν το γράψουμε

```
// ...
for (let i = 0; i < 60; i++) {
    let angle = -Math.PI / 2 + i * Math.PI / 30;
    ctx.save();
    ctx.rotate(angle);
    ctx.beginPath();
    let w = 10;
    ctx.lineWidth = i % 5 == 0 ? 3 : 1;
    ctx.moveTo(radius - w, 0);
    ctx.lineTo(radius, 0)
    ctx.stroke();
    // ανα 5 βαζουμε τον αριθμό της ώρας δηλ. 12, 1, 2, 3, ..., 11
    if (i % 5 == 0) {
        ctx.save();
        ctx.textAlign = "center";
        ctx.font = "18pt monospace";
        ctx.textBaseline = "middle";
        let hour = i == 0 ? 12 : i / 5; // Ξεκινάμε απο το 12
        let digit = hour.toFixed();
        let dt = ctx.measureText(digit).width / 2;
        ctx.translate(radius - w - dt - 2, 0);
        ctx.rotate(-angle);
        ctx.fillText(digit, 0, 0);
        ctx.restore();
    }
    ctx.restore();
}
// ...
```

Λίγη …τριγωνομετρία

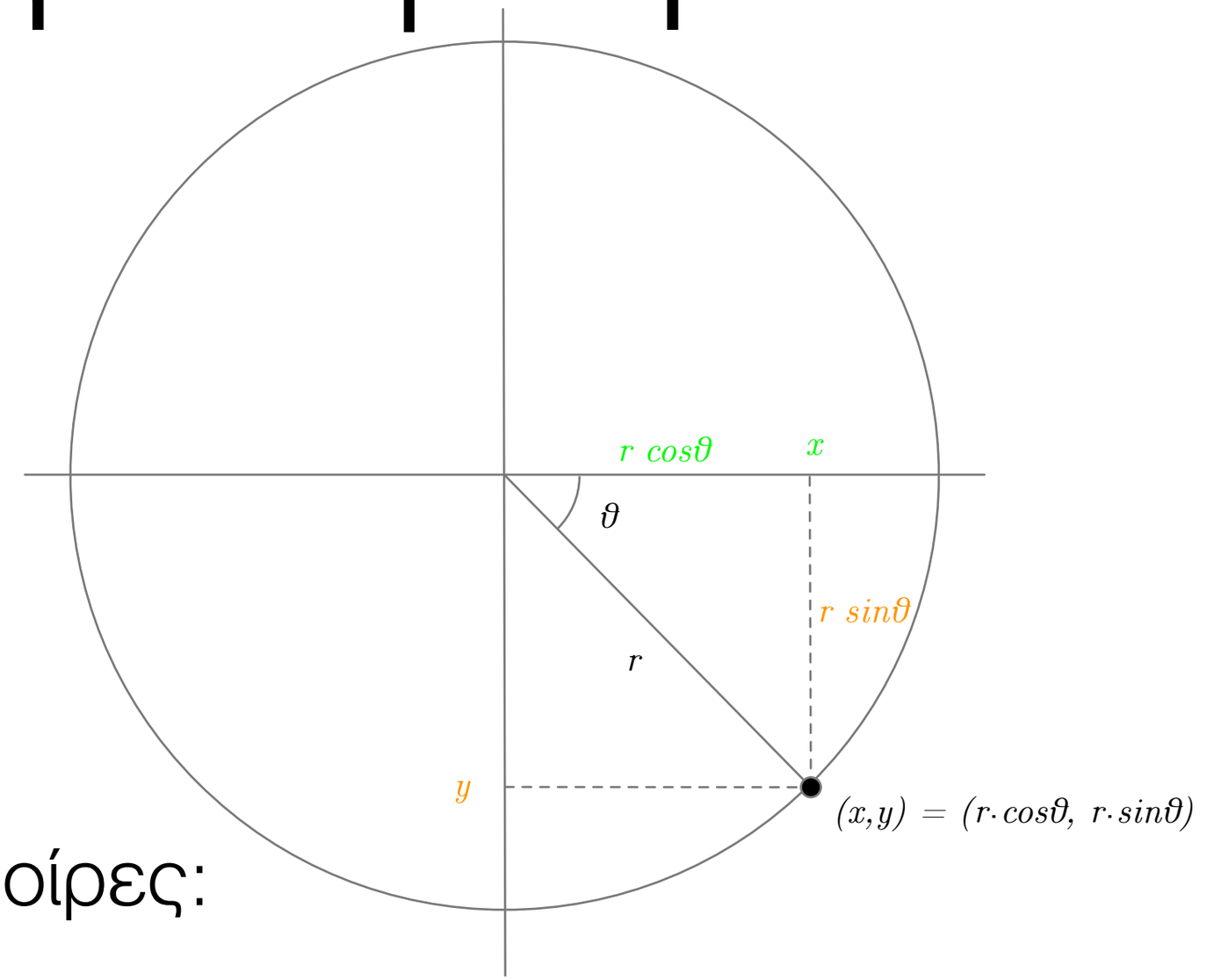
- Μερικές ταυτότητες:

$$x^2 + y^2 = r^2$$

$$(\cos\theta)^2 + (\sin\theta)^2 = 1$$

$$x = r \cdot \cos\theta$$

$$y = r \cdot \sin\theta$$



- Μετατροπή ακτινίων σε μοίρες:

180° = π ακτίνια

```
function degreesToRadians(degrees) {
    return Math.PI * degrees / 180;
}
```

Εικόνες ("Sprites")

- "Sprite": *a computer graphic which may be moved on-screen and otherwise manipulated as a single entity*.

- Πολλές φορές σε παιχνίδια συνδυάζουμε τα sprites σε μια μεγάλη εικόνα (αντί για πολλές μικρές) για λόγους performance, και στον κώδικα μας επιλέγουμε κάθε φορά ποιο τμήμα / sprite από την μεγάλη εικόνα θα δείξουμε

Εικόνες στον καμβά

- Αρχικά, θα πρέπει έχουμε τις εικόνες μέσα στο HTML έγγραφο, είτε στατικά (με `<img>`) ή να δημιουργούνται δυναμικά με Javascript (π.χ. `createElement` ή `new Image`)

- Όταν η εικόνα έχει φορτωθεί, χρησιμοποιούμε το `drawImage` function του context object:

  - `ctx.drawImage(img, x, y)`: Εμφανίζει την εικόνα `img` στις συντεταγμένες `x` και `y`

  - `ctx.drawImage(img, x, y, width, height)`: Εμφανίζει την εικόνα `img` στις συντεταγμένες `x` και `y` κάνοντας την και scale έτσι ώστε να έχει πλάτος `width` και ύψος `height`

  - `ctx.drawImage(image, sx, sy, sWidth, sHeight, dx, dy, dWidth, dHeight)`: Από την εικόνα `img` παίρνει το τμήμα (ορθ. παραλ/μο) που βρίσκεται στις συντεταγμένες `sx` και `sy` και έχει διαστάσεις `sWidth, sHeight` το κάνει scale έτσι ώστε να έχει πλάτος `dWidth` και ύψος `dHeight` και το εμφανίζει στις συντεταγμένες `dx, dy` του καμβά

Παράδειγμα

```
let img = new Image;
img.src = "battleship.png";
let x = ctx.canvas.width / 2 - 25;
let y = ctx.canvas.height - 50;
img.addEventListener('load', function() {
    ctx.drawImage(img, x, y, 50, 50);
});
```

- Η `drawImage` πρέπει να κληθεί όταν έχει φορτωθεί η εικόνα, γιαυτό βάζουμε event listener

- Οι συντεταγμένες `x` και `y` στην `drawImage` χρησιμοποιούνται για την πάνω αριστερή γωνία της εικόνας

- Βάζοντας το 50, 50 στην `drawImage` "αναγκάζουμε" την εικόνα να έχει αυτές τις διαστάσεις μέσα στον καμβά και έτσι μπορούμε να χειριστούμε καλύτερα τη θέση της