P.PORTO

POLITÉCNICO
DO PORTO
ESMAD

COMPUTAÇÃO GRÁFICA
TSIW

Syllabus

- Maths, Physics and Animation:
  - Linear motion
  - Acceleration
  - Circular motion
  - Orientation
- Transformations and States

Maths, Physics and Animation

CONSTANT LINEAR MOVEMENT

- In animation, motion displacement is discretrized in pixels and time is controlled by the animation framerate
- **Constant linear movement**: object moves along a line, by the same number of pixels on each rendered frame

Maths, Physics and Animation

CONSTANT LINEAR MOVEMENT

```
let delta = 5; //5 pixels per frame (displacement or velocity)
window.setInterval(callback, 100); //100 milisseconds between frames
```

Velocity (constant) = 5 pixels / 100ms

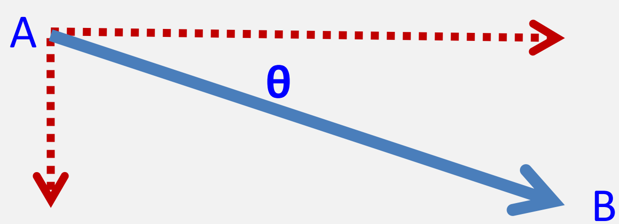
- Animation velocity can be controlled by the **object displacement** (in pixels) or the **animation framerate**
- For animations using `requestAnimationFrame`, time elapsed between frames can be determined by passing na argument to the callback and use it to control the movement
  - Check the example in https://developer.mozilla.org/en-US/docs/Web/API/window/requestAnimationFrame

Maths, Physics and Animation

CONSTANT LINEAR MOVEMENT

If one requires a movement between two points (A and B):

1. Determine the direction **θ** between A and B
2. Use it to control the displacement in X and Y directions (**dX** and **dY**)
3. Control the velocity using constant **V**



dX = V.cos(θ)

dY = V.sin(θ)

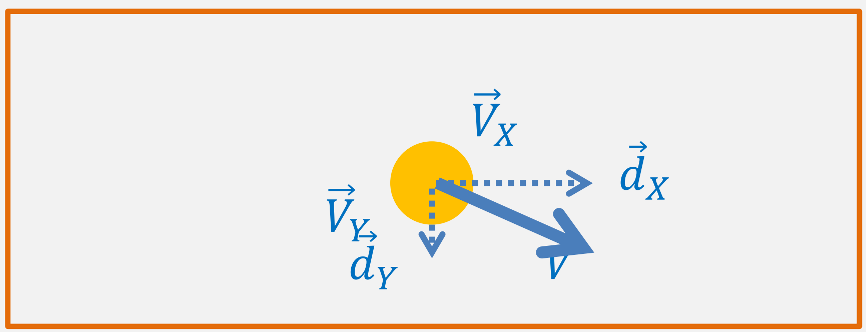
Direction θ = atan2(($B_Y - A_Y$) / ($B_X - A_X$))

.

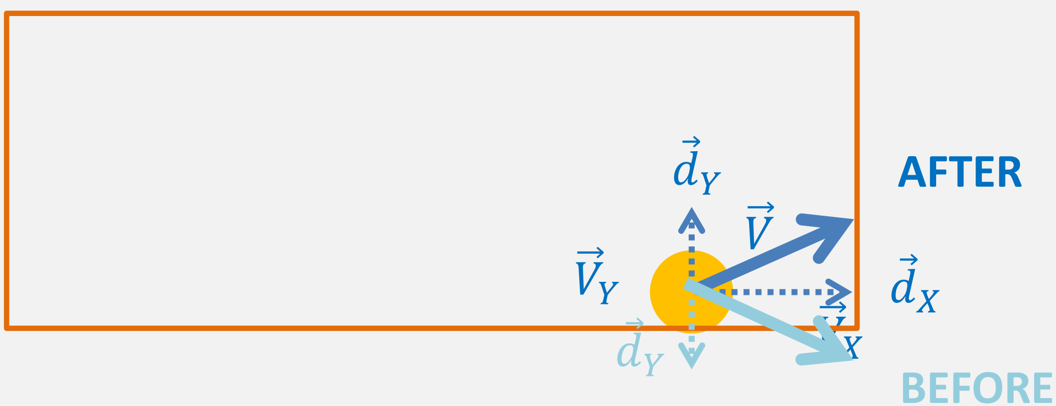
Maths, Physics and Animation

BOUNCING with the CANVAS LIMITS

- Initial state: an object moves linearly with a constant velocity V



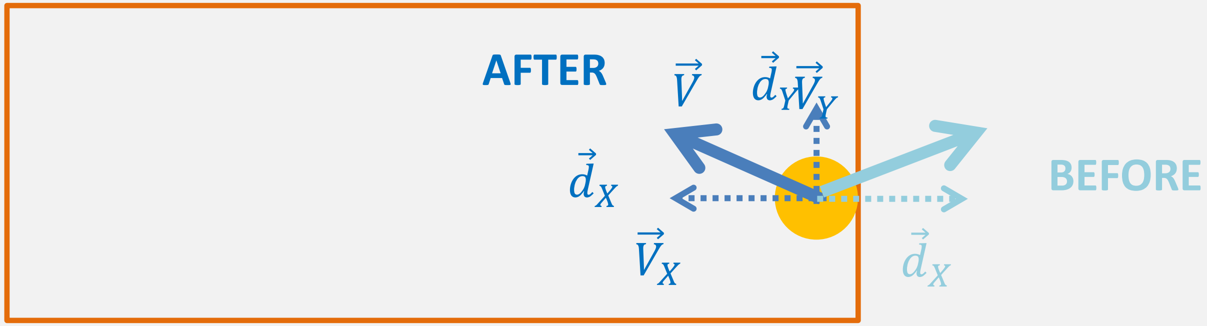
- Object colides with a horizontal Canvas boundary (vertical velocity is inverted)



Maths, Physics and Animation

BOUNCING with the CANVAS LIMITS

- Object colides with a vertical Canvas boundary (horizontal velocity is inverted)



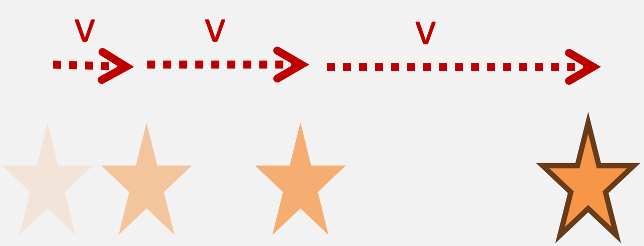
- Inversion is obtained by **inverting the signal** of the velocity (or displacement) variable

Maths, Physics and Animation

ACCELERATION

- Accelerated movement is when the object's velocity **varies** uniformly along time
- In digital animation: the objects velocity (displacement) is incremented (or decremented) by a constant value (acceleration), on each frame

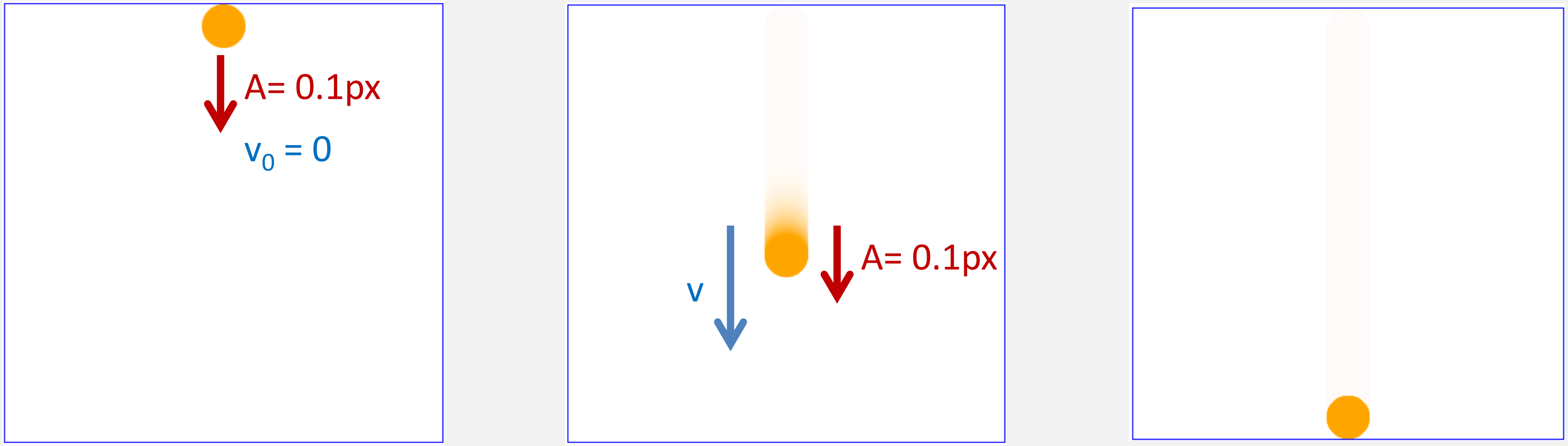
ΔV constant = acceleration



Maths, Physics and Animation

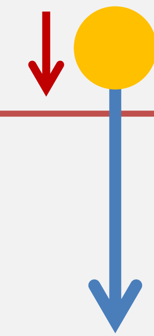
ACCELERATION – example 1: gravity

- Animation start: circle (20 pixels radius) at the top of the Canvas
- Constant acceleration of 0.1 pixels per frame
- Animation: 20ms per frame
- Circle stops when it reaches the bottom of the Canvas



Maths, Physics and Animation

ACCELERATION – example 1: gravity

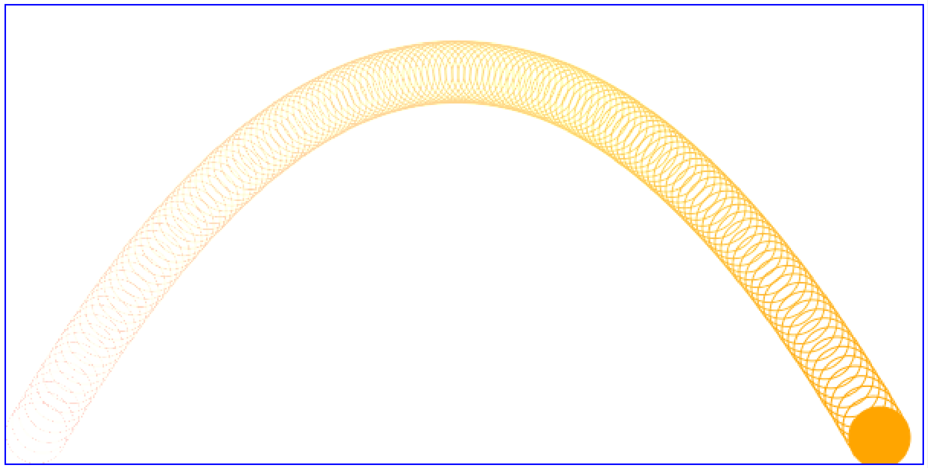



```
//CIRCLE: initialize a circle with properties and methods
let circle = {
     r: 20,  color: "orange",
     x: W / 2, y: 20, // position: Canvas middle top
     dY: 0,    // initial Y velocity (or Y displacement)
     a: 0.1,  // ACCELERATION (gravity = 0.1 pixels per frame)

     draw() { … },

     update() {
        if (this.y < H - this.r) { // if NOT at the Canvas bottom
          this.dY += this.a; // ACCELERATION: increase circle velocity in Y
          this.y += this.dY; // update circle Y position
        }
        else // adjust circle position so that it lies perfectly over the Canvas bottom
          this.y = H - this.r;
      }
};
```

Maths, Physics and Animation

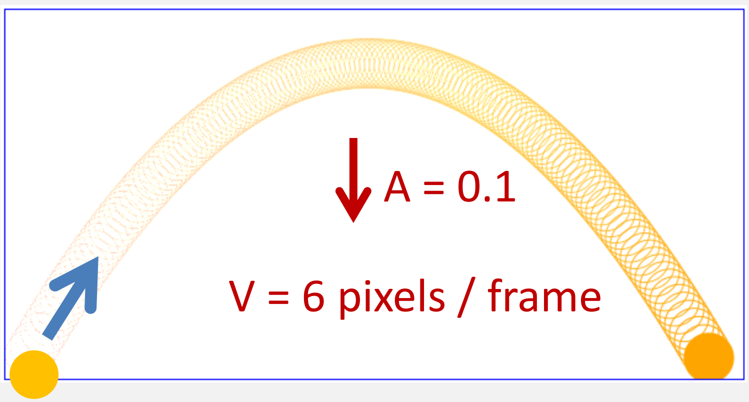
ACCELERATION – example 2: projectile

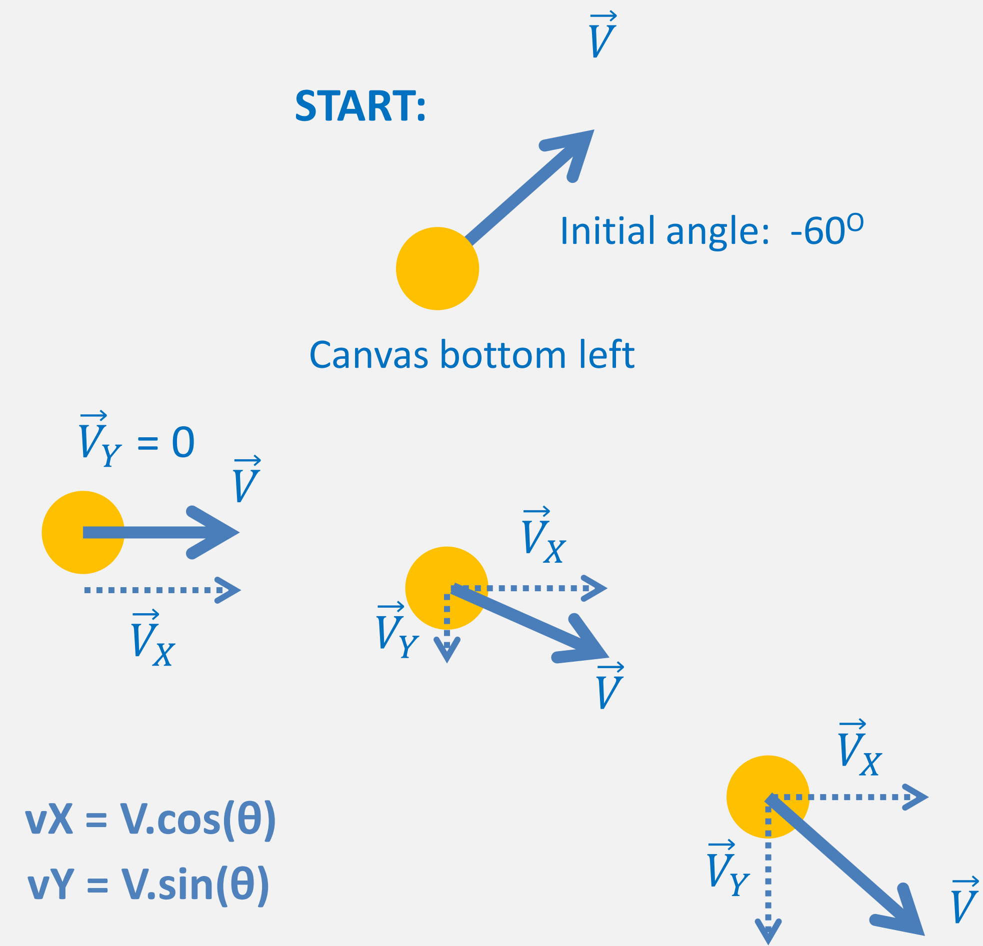
- Animation start: circle at the bottom left of the Canvas, with na initial velocity of 6 pixels per frame and direction of -60º
- Constant gravity (vertical acceleration) of 0.1 pixels per frame
- Animation: as fast as possible
- Circle stops when it reaches again the bottom of the Canvas



Maths, Physics and Animation

ACCELERATION – example 2: projectile



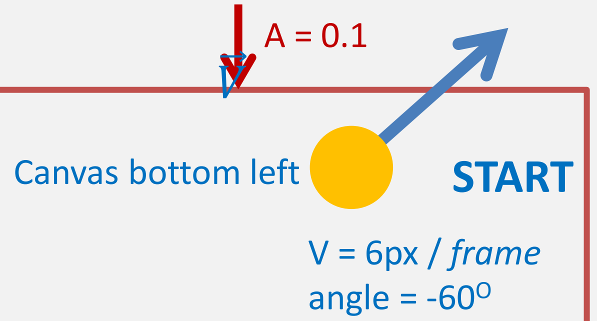


vX = V.cos(θ)

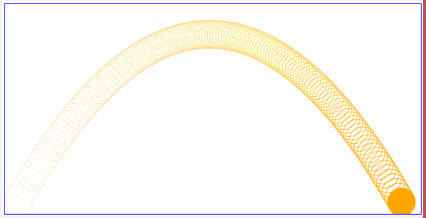
vY = V.sin(θ)

Maths, Physics and Animation

ACCELERATION – example 2: projectile



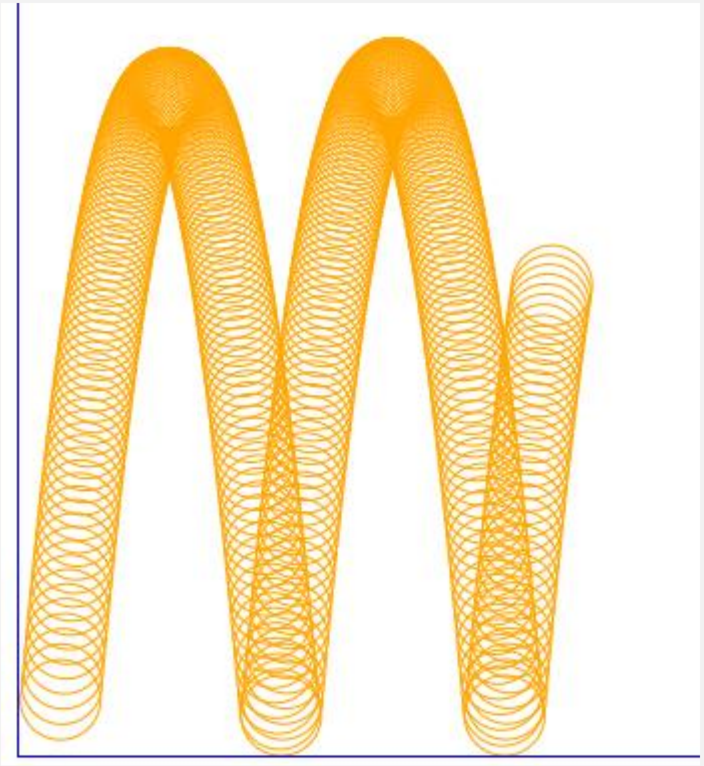

```
let circle = {
     r: 20,  color: "orange",
     x: 20, y: H - 20, // initial position: Canvas bottom left
     dX: 6 * Math.cos(-Math.PI/3), // initial X velocity
     dY: 6 * Math.sin(-Math.PI/3), // initial Y velocity
     a: 0.1,  // acceleration (gravity = 0.1 pixels per frame)

     draw() { … },

     update(){
        if (this.y > H - this.r) { // if circle hits the Canvas bottom
          this.y = H - this.r;   // adjust circle at the bottom
          this.dX = this.dY = 0; // stop circle movement (comment this line and analyse)
        }
        else {
          this.x += this.dX; // update circle X position (X - UNIFORM motion)
          this.dY += this.a; // increase circle velocity in Y ( Y - ACCELERATED motion)
          this.y += this.dY; // update circle Y position
        }
     }
};
```



Try yourself...

ACCELERATION – exercises

- Introduce bouncing to the previous two examples: gravity and projectile
  - For the projectile motion with boucing, consider:
    velocity = 8 pixels per frame
    initial angle = -85º

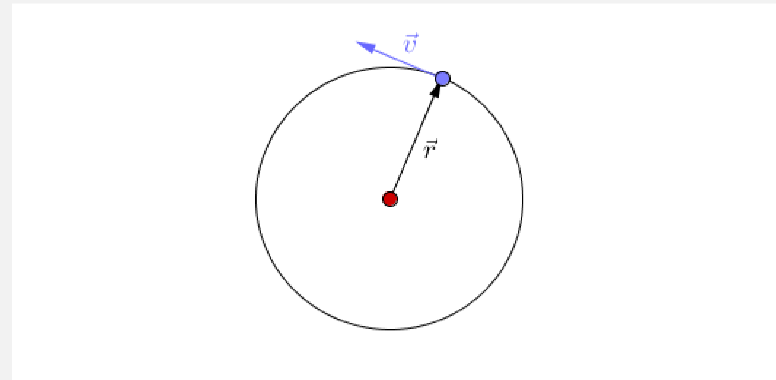


Do NOT stop the circle after reaching the Canvas bottom

Maths, Physics and Animation

CIRCULAR MOTION

- Circular motion is when an object moves along a **circular trajectory**

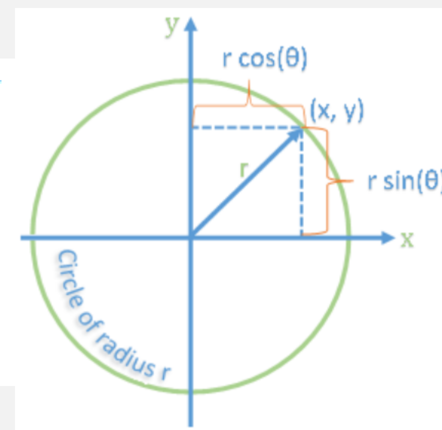


- Object positions are given by the circle parametric equations, where, per frame, **only the angle is altered**

The parametric equations of a circle with center (x_0, y_0) and radius r,

$$x = x_0 + r\cos t$$
$$y = y_0 + r\sin t$$

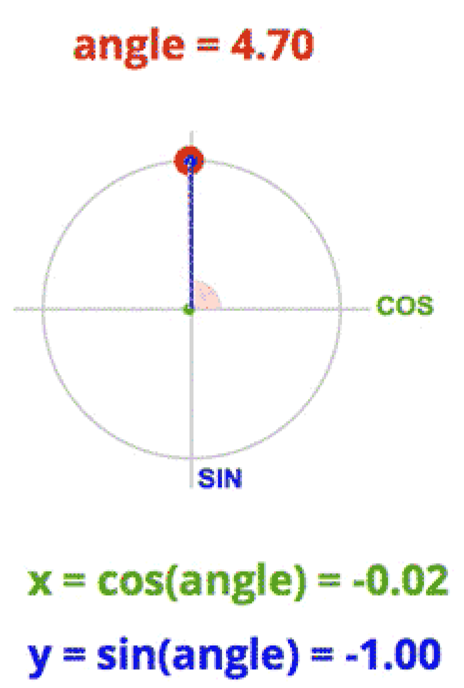
where, $0 \leq t < 2\pi$.



- The motion is called **constant** if the angle variation between frames is always the same

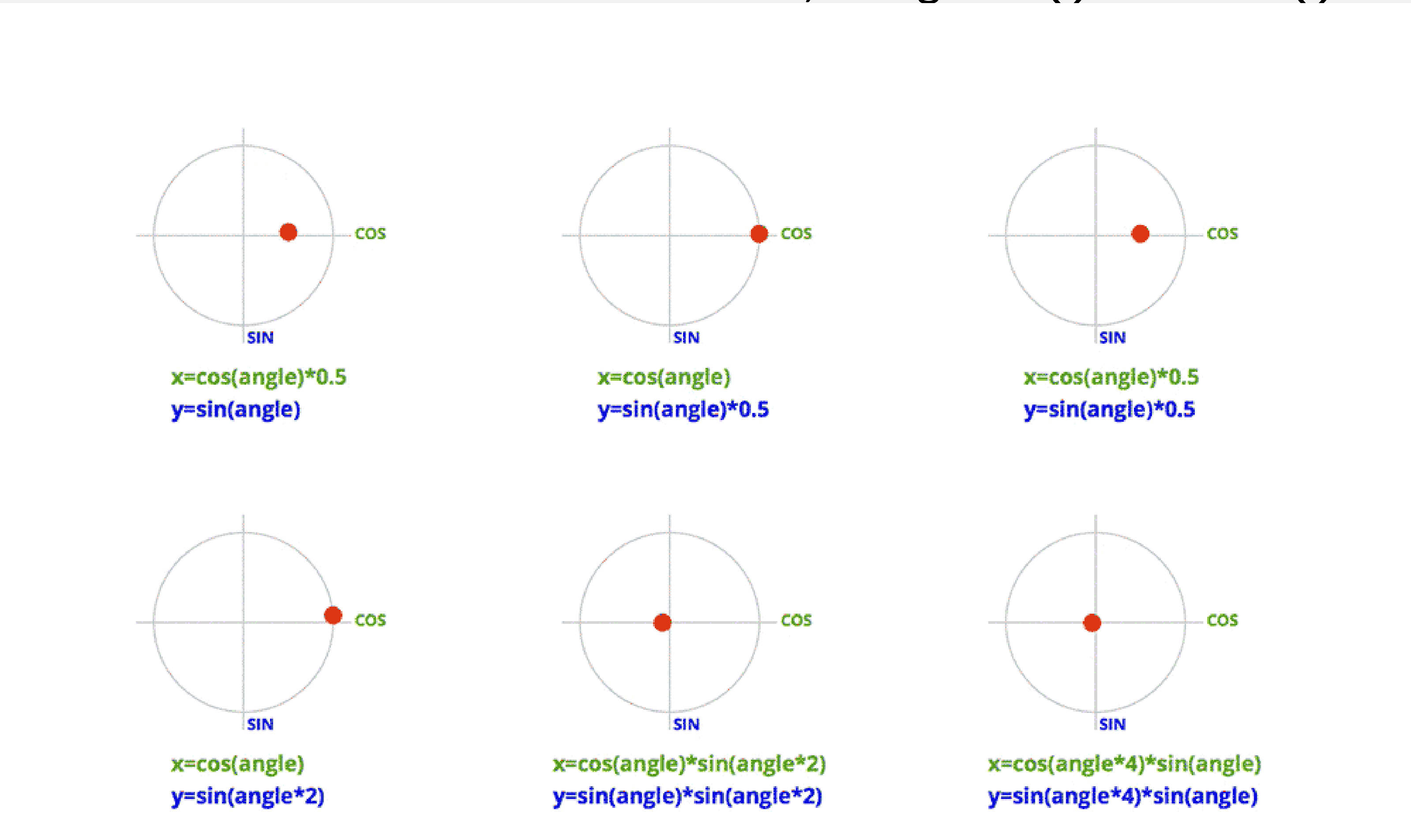
Maths, Physics and Animation

CIRCULAR MOVEMENT



Maths, Physics and Animation

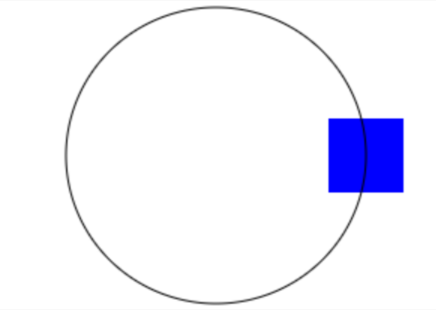
MORE SOPHISTICATED ANIMATIONS, using `sin()` and `cos()`



Example: circular motion

A square moves along a circular trajectory, with a constant velocity of 1 degree per frame. The circular trajectory is centered in the Canvas, with a radius of 100 pixels. It takes 2 s to perform a full revolution. After that, the animation stops.

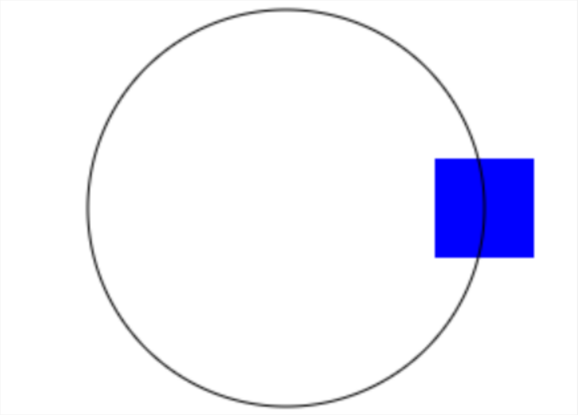
```
let square = {
      color: "blue",  d: 50,
      r: 100, // movement radius
      ang: 0, // movement initial angle (in degrees)

      draw() {
        ctx.fillStyle = this.color;
        ctx.beginPath();
        // center the square center in the imaginary circle
        let posX = W / 2 - this.d / 2 + this.r * Math.cos(Math.PI / 180 * this.ang)
        let posY = H / 2 - this.d / 2 + this.r * Math.sin(Math.PI / 180 * this.ang)
        ctx.fillRect(posX, posY, this.d, this.d);
      },

      update() {
        this.ang++; // increase 1 degree per frame
      }
};
```



Example: circular motion

A square moves along a circular trajectory, with a constant velocity of 1 degree per frame. The circular trajectory is centered in the Canvas, with a radius of 100 pixels. It takes 2 s to perform a full revolution. After that, the animation stops.

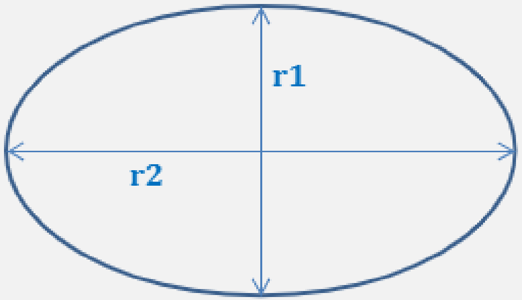
```
let timer = window.setInterval(render, 2000 / 360); // start animation (360º in 2s)

function render() {
      ctx.clearRect(0, 0, W, H); //ERASE

      square.draw(); //DRAW SQUARE

      //auxiliary circle
      ctx.beginPath();
      ctx.arc(W / 2, H / 2, square.r, 0, 2 * Math.PI);
      ctx.stroke();

      square.update(); //UPDATE SQUARE

      //stop animation, after a full revolution (360 degrees)
      if (square.ang > 360) window.clearInterval(timer);
}
```



Try yourself...

Alter the previous example, and move the square along an elliptical trajectory, knowing that the ellipse has two different radius (r1 = 100 pixels and r2 = 200 pixels) and that:

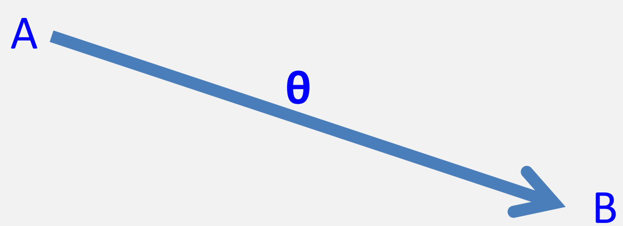
$$\begin{cases} x = c_x + \mathrm{r}2.\cos\theta \\ y = c_y + \mathrm{r}1.\sin\theta \end{cases}$$



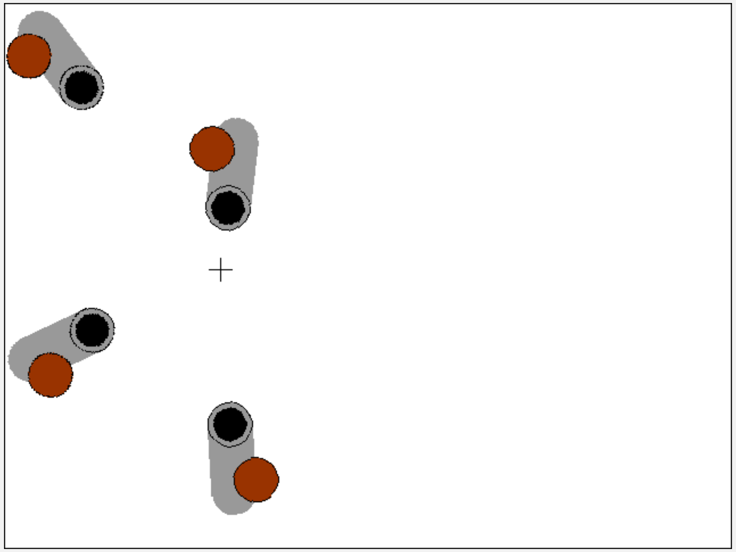
Maths, Physics and Animation

TRIGONOMETRY - ORIENTATION

- Given two points (A and B), the orientation $\overrightarrow{AB}$ is given by:
  $$\overrightarrow{AB} = \theta = atan2(B_y - A_y, B_x - A_x)$$

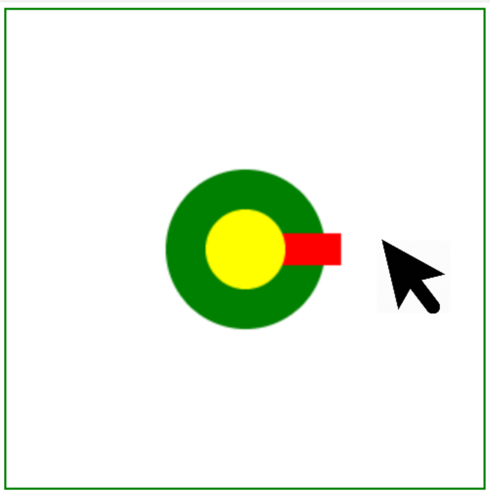


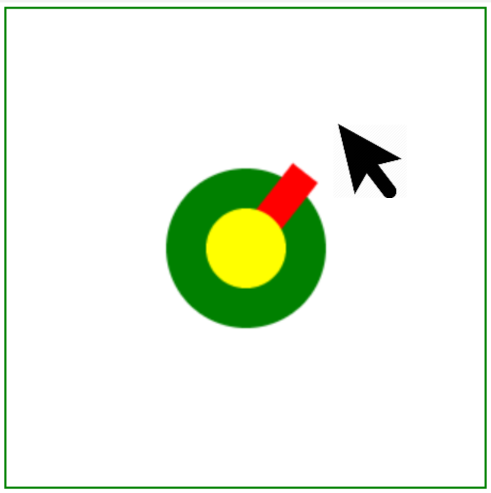
- With the orientation, **a point along this vector** is given by:
  $$\begin{cases} x = \mathrm{k}.\cos\theta \\ y = \mathrm{k}.\sin\theta \end{cases}$$

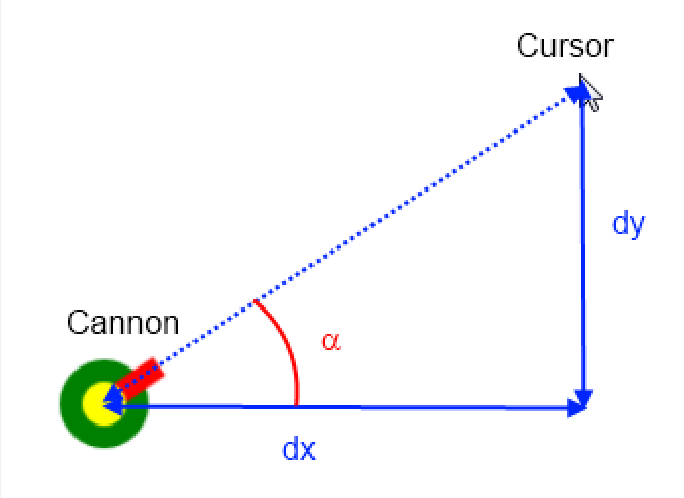


Orientation: example

Cannon: centered in Canvas, like the image below, where the gun (red rectangle) follow mouse movements





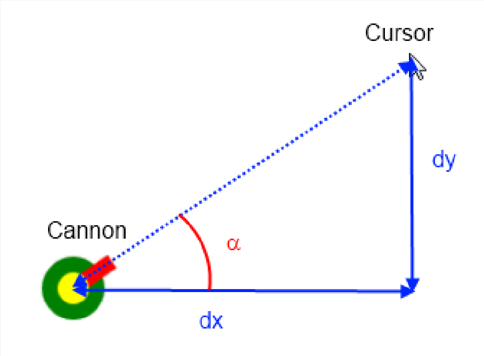


$$\overrightarrow{AB} = \theta = atan2(B_y - A_y, B_x - A_x)$$

Orientation: example

Cannon: centered in Canvas, like the image below, where the gun (red rectangle) follow mouse movements

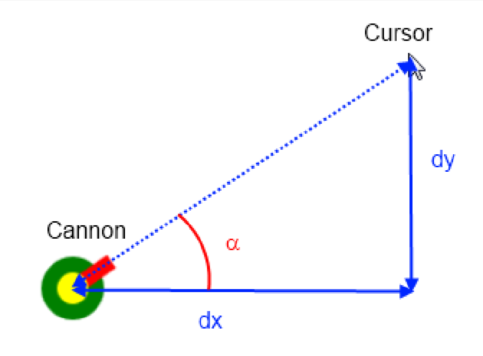
```
//CANNON gun settings
let angle = 0; orientation (directional) angle
ctx.strokeStyle = "red";
ctx.lineWidth = 20;

// Event listener for mousemove
canvas.addEventListener('mousemove', e => {
     let x = e.offsetX; let y = e.offsetY;

     let dx = x - W/2;
     let dy = y - H/2;
     angle = Math.atan2(dy, dx); // update cannon orientation angle
});
```



Orientation: example

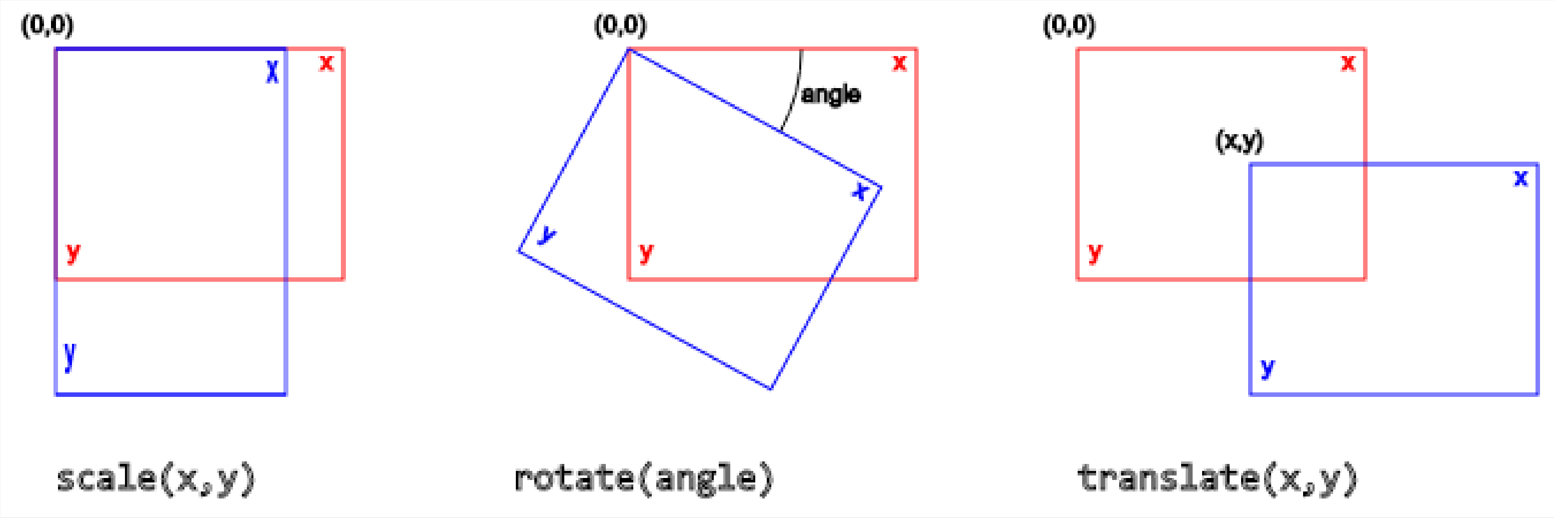
Cannon: centered in Canvas, like the image below, where the gun (red rectangle) follow mouse movements

```
function render() {
    ctx.clearRect(0, 0, W, H);
    //draw cannon
    ctx.fillStyle = "green";
    ctx.beginPath();
    ctx.arc(W / 2, H / 2, 50, 0, 2 * Math.PI);
    ctx.fill();
    ctx.beginPath();
    ctx.moveTo(W/2, H/2);
    ctx.lineTo(W/2 + 75*Math.cos(angle), H/2 + 75*Math.sin(angle));
    ctx.stroke();
    ctx.fillStyle = "yellow";
    ctx.beginPath();
    ctx.arc(W / 2, H / 2, 25, 0, 2 * Math.PI);
    ctx.fill();
    requestAnimationFrame(render);
}
```



Transformations

- In Canvas, a transformation changes its system coordinates (!**not the Canvas element**)
- A transformation alters all the objects that are drawn **afterwards**

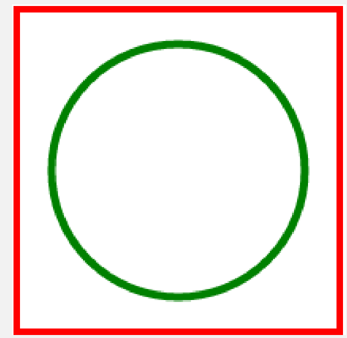


Transformations

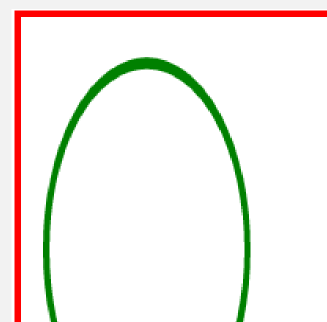
- **scale(s_x,s_y)**: applies a scale factor **s** to the object coordinates

$$\begin{cases} \acute{x} = sx * x \\ \acute{y} = sy * y \end{cases}$$

```
<canvas id="myCanvas" style="border: solid 3px red">

ctx.strokeStyle = 'green';

ctx.lineWidth = 5.0;
ctx.arc(100,100,80,0,2*Math.PI);
ctx.stroke();
```

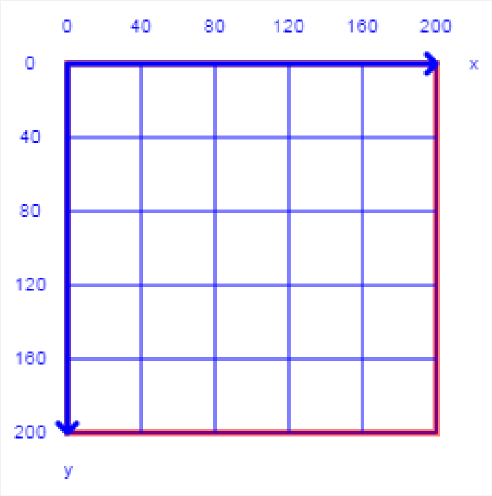


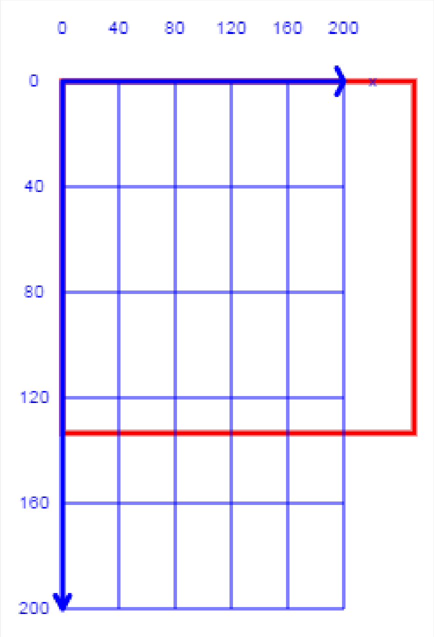
```
(…)
ctx.strokeStyle = 'green';

ctx.scale(0.8, 1.5);
ctx.lineWidth = 5.0;
ctx.arc(100,100,80,0,2*Math.PI);
ctx.stroke();
```

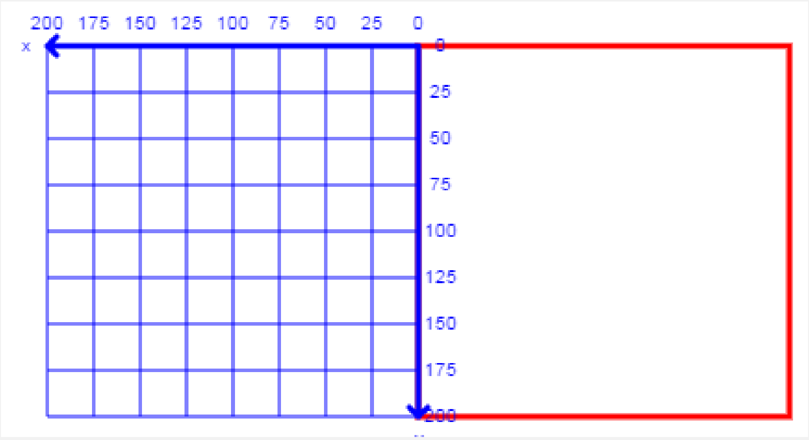


Transformations

- **`scale(`s_x, s_y`)`**: if **`s`** is negative, the scale adds a <u>mirror effect</u>
  - some object coordinates must be negative in order for the object to be drawn inside the Canvas element





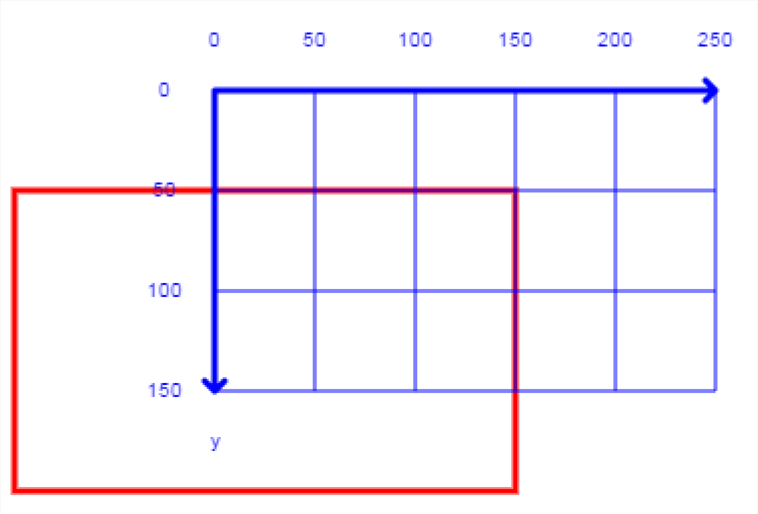


Transformations

- **translate(d_x,d_y)**: moves the origin of the system coordinates to point **(d_x,d_y)**

$$\begin{cases} \acute{x} = x + dx \\ \acute{y} = y + dy \end{cases}$$

No translation



`translate(100,-50)`

Transformations

- **rotate(θ)**: rotates the anti-clockwise system coordinate by **θ** radians, having the origin as center of rotation

$$\begin{cases} \acute{x} = x * \cos\theta - y * \sin\theta \\ \acute{y} = x * \sin\theta + y * \cos\theta \end{cases}$$

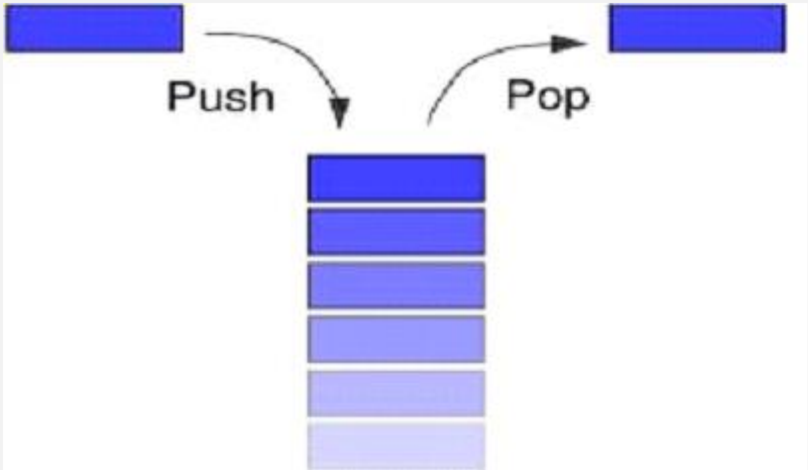
```
ctx.fillStyle = 'red';
ctx.strokeStyle = 'black';
ctx.font = '40px Arial';
// 1. desenha rectângulo com texto
ctx.fillRect (100, 5, 150, 40);
ctx.strokeText ('1. Hello', 105, 40);
// 2. primeira rotação de 30º
ctx.rotate (30 * Math.PI / 180);
ctx.fillRect (100, 5, 150, 40);
ctx.strokeText ('2. Hello', 105, 40);
// 3. segunda rotação de 30º
ctx.rotate (30 * Math.PI / 180);
ctx.fillRect (100, 5, 150, 40);
ctx.strokeText ('3. Hello', 105, 40);
```

States

- The Canvas 2D context allows saving several configurations in the so called **drawing states**
- A drawing state saves information about:
  - Transformations
  - Clipping regions
  - Style (strokeStyle and fillStyle)
  - Composition (globalAlpha, …)
  - Lines (lineWidth, lineCap, …)
  - Text (font, textAlign,…)
  - Shadows (shadowOffsetX, shadowBlur,…)
- A drawing state does not save: the actual path and bitmaps (images)

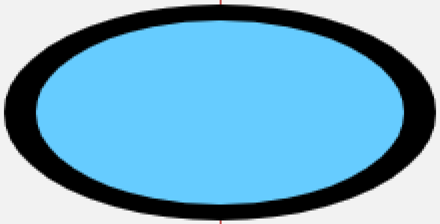
States

- States work like a stack (LIFO data structure: *Last In First Out*)
  - Push: puts a object into the stack
  - Pop: retrieves na object from the stack
- In Canvas:
  - `save()`: equivalent to a push
  - `restore()`: equivalent to a pop

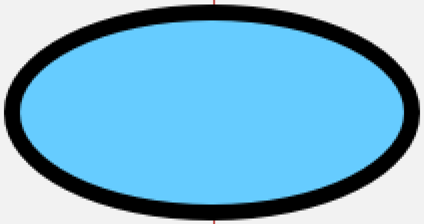


States

```
// Circle with squashed outline
ctx.scale(2, 1);
ctx.beginPath();
ctx.arc(50, 50, 50, 0, Math.PI*2);
ctx.fillStyle = "#6cf";
ctx.fill();
ctx.lineWidth = 8;
ctx.strokeStyle = "#000";
ctx.stroke();
```



```
// Circle with intact outline
ctx.save();
ctx.scale(2, 1);
ctx.beginPath();
ctx.arc(50, 50, 50, 0, Math.PI*2);
ctx.fillStyle = "#6cf";
ctx.fill();
ctx.restore();
ctx.lineWidth = 8;
ctx.strokeStyle = "#000";
ctx.stroke();
```



States

```
let angle = 0;
ctx.fillStyle = '#0095DD';

function render() {
   ctx.clearRect(0, 0, W, H)

   // draw a STATIC blue rectangle
   ctx.fillRect(W/2 - 50, H/2 - 50, 100, 100);

   // draw a ROTATING grey rectangle
   ctx.save();
   ctx.translate(W / 2, H / 2)
   ctx.rotate(angle * Math.PI / 180)
   ctx.fillStyle = '#4D4E53';
   ctx.fillRect(-50, -50, 100, 100); //draw considering (0,0) as the its center
   ctx.restore();

   angle++;
   window.requestAnimationFrame(render)
}
render();
```

Try yourself…

1. Write text “Hello!!!” centered in the Canvas element. Then apply the following transformations, making sure to adapt the text coordinates in order to be always visible

    a) Scale(-1,1)

    b) Scale(1,-1)

    c) Scale(-1,-1)

    d) All the above in the same Canvas

Try yourself...

2. Draw the following three squares: all are centered in the Canvas element. The second is rotated by 30 degrees and the third by 60 degrees, with all rotation axes at the Canvas center.

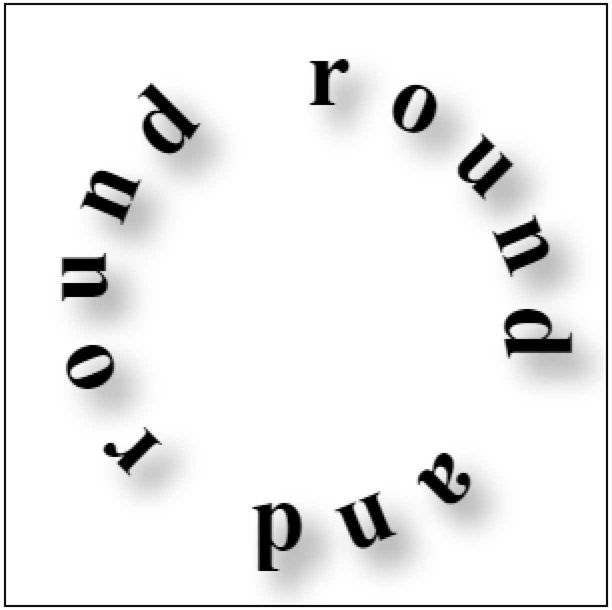
HINT: To rotate an object around its center, you must first translate the Canvas coordinate system to the center of the object and then rotate it

Try yourself...

3. Using states and transformations, write the following text “round and round ”, as if it was standing along a circle (radius 100 pixels), centered in the Canvas element.

HINT: Get the angle of rotation by dividing a full circle by the number of characters

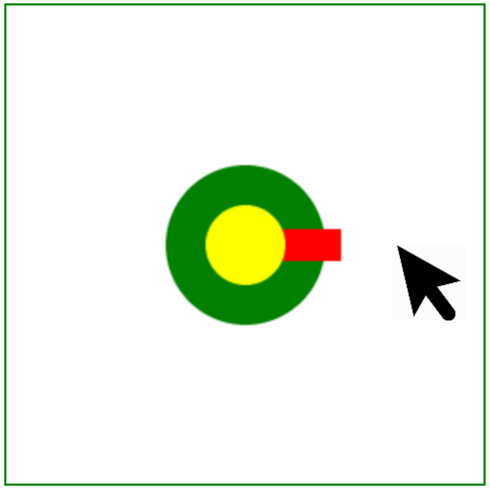
```
const text = "round and round ";
let nChars = text.length;
```

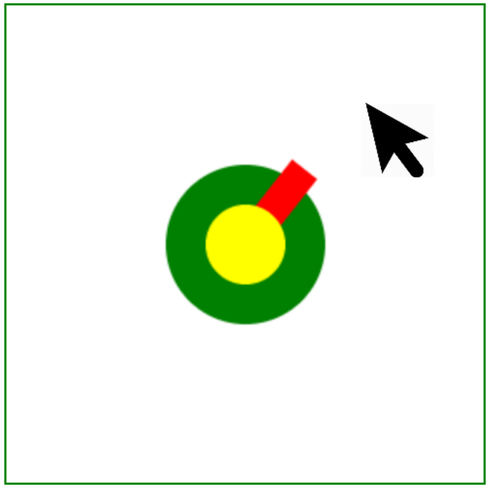


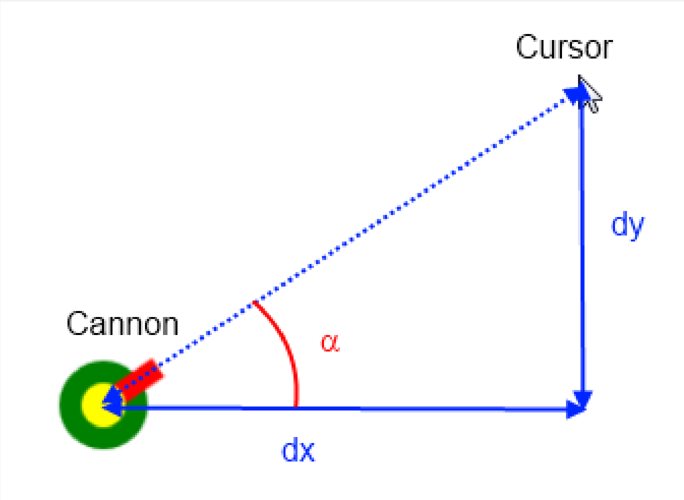
Try yourself...

Remember the Cannon following the mouse cursor?

Alter it so by drawing the red part using a rectangle shape, and by rotation it using transformations.





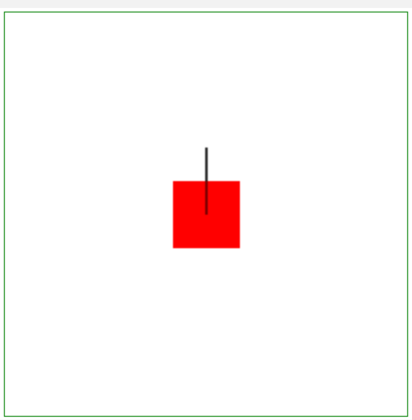


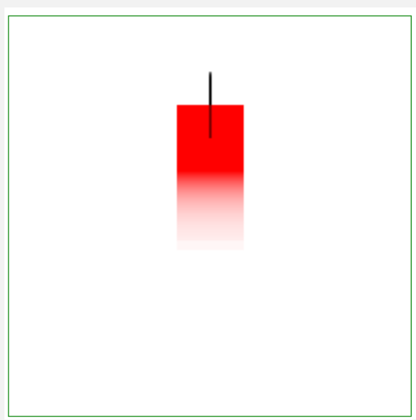
Try yourself...

Remember the circle controlled by the arrow keys?

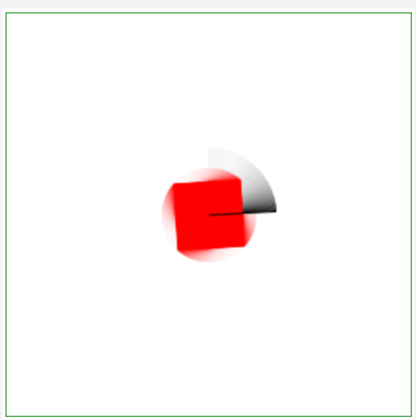
Alter it so that you may control a red square, that starts in the Canvas center. Use the UP key to move ir forward. Use the RIGHT and LEFT keys to rotate.

Combine the UP key with one of the others to make it circle around.

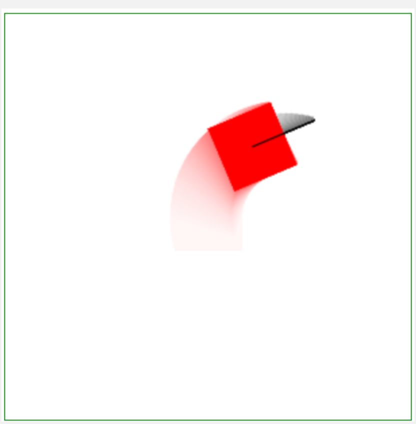




UP key



RIGHT key



UP + RIGHT keys