

 免費電子書

學習

# html5-canvas

Free unaffiliated eBook created from  
**Stack Overflow contributors.**

#html5-  
canvas

.....	1
<b>1: html5-canvas</b> .....	<b>2</b>
Examples.....	2
Html5 Canvas.....	2
.....	2
.....	3
.....	3
.....	4
Html5 Canvas.....	4
.....	4
.....	4
.....	5
.....	5
<b>2: “getImageData”“putImageData”</b> .....	<b>7</b>
Examples.....	7
“context.getImageData”.....	7
.....	8
<b>3: .....</b>	<b>10</b>
Examples.....	10
2DrequestAnimationFrame.....	10
1.....	10
.....	11
requestAnimationFrameNOT setInterval.....	12
“”.....	13
.....	14
Robert Penners.....	16
requestAnimationFrame.....	21
[x0y0][x1y1].....	21
<b>4: .....</b>	<b>23</b>
Examples.....	23
“”.....	23

“”	23
“destination-in”	24
“source-in”	25
“source-atop”	25
“”	26
“”	26
“”	27
FX“”	27
“globalAlpha”	27
<b>5:</b>	<b>29</b>
Examples	29
“context.drawImage”Canvas	29
“context.drawImage”Canvas	30
“context.drawImage”Canvas	30
“context.drawImage”Canvas	31
<b>6:</b>	<b>32</b>
Examples	32
“context.drawImage”Canvas	32
“context.drawImage”Canvas	34
“context.drawImage”Canvas	35
“context.drawImage”Canvas	35
<b>7:</b>	<b>38</b>
Examples	38
“context.drawImage”Canvas	38
“context.drawImage”Canvas	39
“context.drawImage”Canvas	39
“context.drawImage”Canvas	39

Canvas.....	40
“”.....	43
<b>8:</b> .....	<b>46</b>
Examples.....	46
.....	46
.....	46
.....	47
<b>9:</b> .....	<b>49</b>
.....	49
Examples.....	50
.....	50
svg.....	50
.....	51
.....	51
.....	51
.....	51
can playonload.....	52
.....	52
.....	52
.....	52
.....	52
webM.....	52
.....	52
<b>10:</b> .....	<b>54</b>
Examples.....	54
.....	54
.....	55
.....	55
.....	56
.....	58
.....	59
.....	59
.....	

..... 62

**CanvasRenderingContext2D.fillCircleTexttxtyradiusstart[end[forward]];..... 62**

**CanvasRenderingContext2D.strokeCircleTexttxtyradiusstart[end[forward]];..... 62**

**CanvasRenderingContext2D.measureCircleTexttxtradius;..... 62**

..... 68

..... 70

..... 71

..... 71

..... 71

..... 72

..... 74

..... 74

..... 77

..... 79

..... 79

..... 79

..... 82

..... 85

..... 85

**11: ..... 89**

Examples..... 89

..... 89

..... 90

..... 90

..... 91

..... 94

..... 94

..... 94

..... 95

..... 95

..... 95

.....	96
.....	96
.....	97
.....	98
.....	98
.....	98
<b>12:</b> .....	<b>100</b>
.....	100
.....	100
Examples.....	100
.....	100
.....	100
.....	100
.....	100
.....	101
<b>13:</b> .....	<b>102</b>
Examples.....	102
.....	102
2.....	102
.....	102
2.....	102
.....	105
.....	106
2.....	107
2 .....	108
XY.....	109
XY.....	109
XY.....	110
XY.....	110
<b>14:</b> .....	<b>112</b>
Examples.....	112
.....	112
.....	

<b>15:</b>	<b>115</b>
.....	115
Examples.....	115
.....	115
.....	<b>115</b>
lineTo.....	117
arc.....	117
quadraticCurveTo.....	118
bezierCurveTo.....	118
arcTo.....	119
rect.....	119
closePath.....	120
beginPath.....	121
lineCap.....	122
lineJoin.....	122
strokeStyle.....	123
fillStyle.....	124
lineWidth.....	124
shadowColorshadowBlurshadowOffsetXshadowOffsetY.....	125
createLinearGradient.....	126
createRadialGradient.....	127
.....	<b>130</b>
createPattern.....	130
stroke.....	132
.....	<b>132</b>
.....	135
clip.....	135
<b>16:</b>	<b>136</b>
Examples.....	136
.....	136
.....	

.....137

.....138

.....139

.....139

**17: .....143**

Examples.....143

.....143

.....144

- .....144

.....145

.....145

.....145

.....146

.....147

**18: .....149**

Examples.....149

.....149

.....149

.....150

Debounced resize.....150

.....151

.....152



---

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: [html5-canvas](#)

It is an unofficial and free html5-canvas ebook created for educational purposes. All the content is extracted from [Stack Overflow Documentation](#), which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official html5-canvas.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to [info@zzzprojects.com](mailto:info@zzzprojects.com)

# 1: html5-canvas

## Examples

### Html5 Canvas

#### Html5-Canvas .....

- Html5。
- Internet Explorer 9+。
- 
- 300px150px。
- JavaScriptCanvas。

#### Html5JavaScriptHtml5-Canvas

```
<!doctype html>
<html>
<head>
<style>
    body{ background-color:white; }
    #canvasHtml5{border:1px solid red; }
    #canvasJavascript{border:1px solid blue; }
</style>
<script>
window.onload=(function(){

    // add a canvas element using javascript
    var canvas=document.createElement('canvas');
    canvas.id='canvasJavascript'
    document.body.appendChild(canvas);

}); // end $(function()){};
</script>
</head>
<body>

    <!-- add a canvas element using html -->
    <canvas id='canvasHtml5'></canvas>

</body>
</html>
```

#### CSS。

```
canvas {
    width : 1000px;
    height : 1000px;
}
```

。 。 300 x 150。

CSSwidthheight300 x 150◦

```
canvas {  
  width : 1000px;  
  height : 1000px;  
}
```

◦ 12◦ ◦ ◦

◦

CSSwidthheight◦

```
canvas {  
  width : 1000px;  
  height : 1000px;  
}
```

◦ 2D◦ ◦

◦

```
function createCanvas(width, height){  
  var canvas = document.createElement("canvas"); // create a canvas element  
  canvas.width = width;  
  canvas.height = height;  
  return canvas;  
}  
  
var myCanvas = createCanvas(256,256); // create a small canvas 256 by 256 pixels  
var ctx = myCanvas.getContext("2d");  
ctx.fillStyle = "blue";  
ctx.fillRect(0,0,256,256);
```

◦ ◦

```
function createCanvas(width, height){  
  var canvas = document.createElement("canvas"); // create a canvas element  
  canvas.width = width;  
  canvas.height = height;  
  return canvas;  
}  
  
var myCanvas = createCanvas(256,256); // create a small canvas 256 by 256 pixels  
var ctx = myCanvas.getContext("2d");  
ctx.fillStyle = "blue";  
ctx.fillRect(0,0,256,256);
```

(0,0)HTML5 Canvas◦ e.clientXe.clientYleftrightXY

```
var canvas = document.getElementById("myCanvas");  
var ctx = canvas.getContext("2d");  
ctx.font = "16px Arial";
```

```

canvas.addEventListener("mousemove", function(e) {
    var cRect = canvas.getBoundingClientRect(); // Gets CSS pos, and width/height
    var canvasX = Math.round(e.clientX - cRect.left); // Subtract the 'left' of the canvas
    var canvasY = Math.round(e.clientY - cRect.top); // from the X/Y positions to make
    ctx.clearRect(0, 0, canvas.width, canvas.height); // (0,0) the top left of the canvas
    ctx.fillText("X: "+canvasX+", Y: "+canvasY, 10, 20);
});

```

Math.roundx,y°

## HTML

```

<canvas id="canvas" width=300 height=100 style="background-color:#808080;">
</canvas>

```

## Javascript

```

<canvas id="canvas" width=300 height=100 style="background-color:#808080;">
</canvas>

```



Hello World

## Html5 Canvas

### Canvas

- 
- 
- °
- 
- 
- 
- 
- 
- 
- 
- 
- 
- 
- 
- 

- /
-

- 
- Photoshop
- Flash。

CanvasAlpha。 Photoshop。

- HSL
- “”/
- -
- 
- 
- 
- 
- Sobel Edge Detection
- ◦ “photoshop”。 - “
- /
- .jpg | .png
- ◦
- ◦ Canvas。 GPU。
- ◦ ◦

2Drotate(r) r。

## HTML

```
<canvas id="canvas" width=240 height=240 style="background-color:#808080;">
</canvas>

<button type="button" onclick="rotate_ctx();">Rotate context</button>
```

## Javascript

```
<canvas id="canvas" width=240 height=240 style="background-color:#808080;">
</canvas>

<button type="button" onclick="rotate_ctx();">Rotate context</button>
```

## JSfiddle

canvas.toDataURL() *URI*。

canvas.toDataURL(type, encoderOptions) typeimage/png; encoderOptions010.92。

URI“Download to myImage.jpg”。

## HTML

```
<canvas id="canvas" width=240 height=240 style="background-color:#808080;">
</canvas>
```

```
<p></p>  
<a id="download" download="myImage.jpg" href="" onclick="download_img(this);">Download to  
myImage.jpg</a>
```

## Javascript

```
<canvas id="canvas" width=240 height=240 style="background-color:#808080;">  
</canvas>  
<p></p>  
<a id="download" download="myImage.jpg" href="" onclick="download_img(this);">Download to  
myImage.jpg</a>
```

JSfiddle ◦

html5-canvas <https://riptutorial.com/zh-TW/html5-canvas/topic/1892/html5-canvas>

## 2: “getImageData” “putImageData”

### Examples

“context.getImageData”

Html5 Canvas◦

#### Canvas

- ◦
- ◦
- HSL◦ CanvasBlend Compositing◦
- “”/
- ◦
- ◦
- getImageData◦
- getImageDataGPU◦ CanvasgetImageData◦
- .png getImageData.pngalpha◦

getImageData◦

getImageDataimageData

imageData.data◦

dataUint8ClampedArrayAlpha◦

```
// determine which pixels to fetch (this fetches all pixels on the canvas)
var x=0;
var y=0;
var width=canvas.width;
var height=canvas.height;

// Fetch the imageData object
var imageData = context.getImageData(x,y,width,height);

// Pull the pixel color data array from the imageData object
var pixelDataArray = imageData.data;
```

data[xy]

```
// determine which pixels to fetch (this fetches all pixels on the canvas)
var x=0;
var y=0;
var width=canvas.width;
var height=canvas.height;
```

```
// Fetch the imageData object
var imageData = context.getImageData(x,y,width,height);

// Pull the pixel color data array from the imageData object
var pixelDataArray = imageData.data;
```

## alpha

```
// determine which pixels to fetch (this fetches all pixels on the canvas)
var x=0;
var y=0;
var width=canvas.width;
var height=canvas.height;

// Fetch the imageData object
var imageData = context.getImageData(x,y,width,height);

// Pull the pixel color data array from the imageData object
var pixelDataArray = imageData.data;
```

---

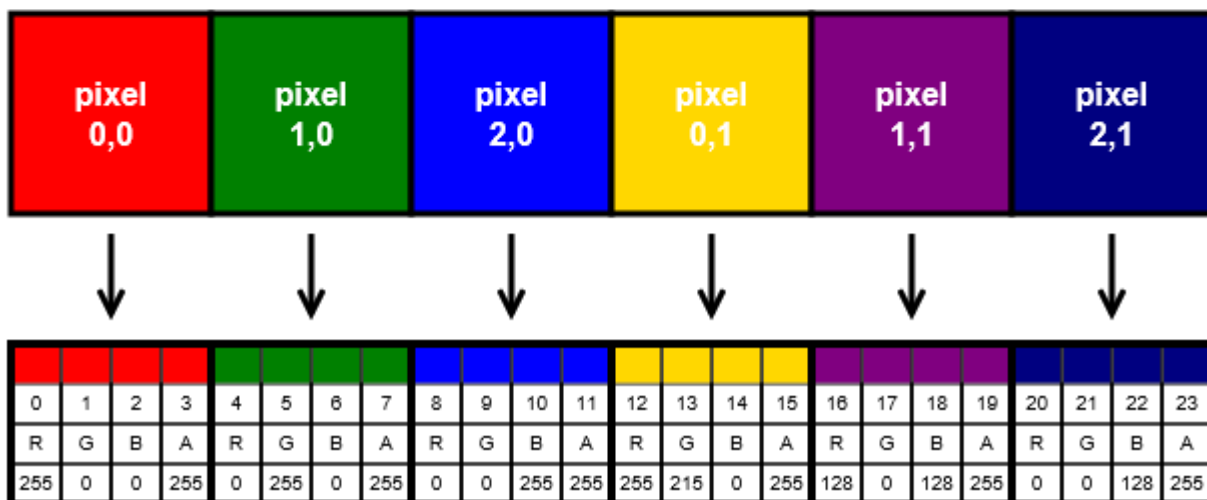
context.getImageData**2x3**



## 2x3 pixel canvas



Pixels are arranged sequentially by row  
Each pixel gets 4 array elements  
(Red, Blue, Green & Alpha)



“getImageData”“putImageData” <https://riptutorial.com/zh-TW/html5-canvas/topic/5573/-getImageData-putImageData->

# 3:

## Examples

### 2DrequestAnimationFrame

#### 2D。 DOM

```
// this example assumes ctx and canvas have been created
const textToDisplay = "This is an example that uses the canvas to animate some text.";
const textStyle     = "white";
const BGStyle       = "black"; // background style
const textSpeed     = 0.2;     // in pixels per millisecond
const textHorMargin = 8;       // have the text a little outside the canvas

ctx.font = Math.floor(canvas.height * 0.8) + "px arial"; // size the font to 80% of canvas height
var textWidth      = ctx.measureText(textToDisplay).width; // get the text width
var totalTextSize = (canvas.width + textHorMargin * 2 + textWidth);
ctx.textBaseline   = "middle"; // not put the text in the vertical center
ctx.textAlign      = "left";   // align to the left
var textX          = canvas.width + 8; // start with the text off screen to the right
var textOffset     = 0;         // how far the text has moved

var startTime;
// this function is call once a frame which is approx 16.66 ms (60fps)
function update(time){ // time is passed by requestAnimationFrame
  if(startTime === undefined){ // get a reference for the start time if this is the first frame
    startTime = time;
  }
  ctx.fillStyle = BGStyle;
  ctx.fillRect(0, 0, canvas.width, canvas.height); // clear the canvas by drawing over it
  textOffset = ((time - startTime) * textSpeed) % (totalTextSize); // move the text left
  ctx.fillStyle = textStyle; // set the text style
  ctx.fillText(textToDisplay, textX - textOffset, canvas.height / 2); // render the text

  requestAnimationFrame(update); // all done request the next frame
}
requestAnimationFrame(update); // to start request the first frame
```

#### jsfiddle

### 1

#### 1== 1

```
<!doctype html>
<html>
<head>
<style>
  body{ background-color:white; }
```

```

    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // canvas related variables
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");
    var cw=canvas.width;
    var ch=canvas.height;

    // animation interval variables
    var nextTime=0;        // the next animation begins at "nextTime"
    var duration=1000;     // run animation every 1000ms

    var x=20;              // the X where the next rect is drawn

    // start the animation
    requestAnimationFrame(animate);

    function animate(currentTime){

        // wait for nextTime to occur
        if(currentTime<nextTime){
            // request another loop of animation
            requestAnimationFrame(animate);
            // time hasn't elapsed so just return
            return;
        }
        // set nextTime
        nextTime=currentTime+duration;

        // add another rectangle every 1000ms
        ctx.fillStyle='#'+Math.floor(Math.random()*16777215).toString(16);
        ctx.fillRect(x,30,30,30);

        // update X position for next rectangle
        x+=30;

        // request another loop of animation
        requestAnimationFrame(animate);
    }

}); // end $(function(){});
</script>
</head>
<body>
    <canvas id="canvas" width=512 height=512></canvas>
</body>
</html>

```

```

<!doctype html>
<html>
<head>
<style>
    body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

```

```

// canvas related variables
var canvas=document.getElementById("canvas");
var ctx=canvas.getContext("2d");
var cw=canvas.width;
var ch=canvas.height;
// canvas styling for the clock
ctx.strokeStyle='lightgray';
ctx.fillStyle='skyblue';
ctx.lineWidth=5;

// cache often used values
var PI=Math.PI;
var fullCircle=PI*2;
var sa=-PI/2;    // == the 12 o'clock angle in context.arc

// start the animation
requestAnimationFrame(animate);

function animate(currentTime){

    // get the current seconds value from the system clock
    var date=new Date();
    var seconds=date.getSeconds();

    // clear the canvas
    ctx.clearRect(0,0,cw,ch);

    // draw a full circle (== the clock face);
    ctx.beginPath();
    ctx.moveTo(100,100);
    ctx.arc(100,100,75,0,fullCircle);
    ctx.stroke();
    // draw a wedge representing the current seconds value
    ctx.beginPath();
    ctx.moveTo(100,100);
    ctx.arc(100,100,75,sa,sa+fullCircle*seconds/60);
    ctx.fill();

    // request another loop of animation
    requestAnimationFrame(animate);
}

}); // end $(function(){});
</script>
</head>
<body>
    <canvas id="canvas" width=512 height=512></canvas>
</body>
</html>

```

## requestAnimationFrame NOT setInterval

requestAnimationFrame **setInterval**

- clear +. +. .
- . " . . .
-

◦ ◦

60requestAnimationFrame60""。 20-30requestAnimationFrame。

animateCirclerequestAnimationFrame。 'requestAnimatonFrameonly。

## `requestAnimationFrame

```
<!doctype html>
<html>
<head>
<style>
  body{ background-color:white; }
  #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

  // canvas related variables
  var canvas=document.getElementById("canvas");
  var ctx=canvas.getContext("2d");
  var cw=canvas.width;
  var ch=canvas.height;

  // start the animation
  requestAnimationFrame(animate);

  function animate(currentTime){

    // draw a full randomly circle
    var x=Math.random()*canvas.width;
    var y=Math.random()*canvas.height;
    var radius=10+Math.random()*15;
    ctx.beginPath();
    ctx.arc(x,y,radius,0,Math.PI*2);
    ctx.fillStyle='#'+Math.floor(Math.random()*16777215).toString(16);
    ctx.fill();

    // request another loop of animation
    requestAnimationFrame(animate);
  }

}); // end $(function(){});
</script>
</head>
<body>
  <canvas id="canvas" width=512 height=512></canvas>
</body>
</html>
```

requestAnimationFrame[stackoverflow](#)

“ ”

## Canvas

image.onload◦

```

<!doctype html>
<html>
<head>
<style>
    body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // canvas related variables
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");
    var cw=canvas.width;
    var ch=canvas.height;

    // animation related variables
    var minX=20;           // Keep the image animating
    var maxX=250;          // between minX & maxX
    var x=minX;            // The current X-coordinate
    var speedX=1;          // The image will move at 1px per loop
    var direction=1;       // The image direction: 1==righward, -1==leftward
    var y=20;              // The Y-coordinate

    // Load a new image
    // IMPORTANT!!! You must give the image time to load by using img.onload!
    var img=new Image();
    img.onload=start;
    img.src="https://dl.dropboxusercontent.com/u/139992952/stackoverflow/sun.png";
    function start(){
        // the image is fully loaded so start animating
        requestAnimationFrame(animate);
    }

    function animate(time){

        // clear the canvas
        ctx.clearRect(0,0,cw,ch);

        // draw
        ctx.drawImage(img,x,y);

        // update
        x += speedX * direction;
        // keep "x" inside min & max
        if(x<minX){ x=minX; direction*=-1; }
        if(x>maxX){ x=maxX; direction*=-1; }

        // request another loop of animation
        requestAnimationFrame(animate);
    }

}); // end $(function(){});
</script>
</head>
<body>
    <canvas id="canvas" width=512 height=512></canvas>
</body>
</html>

```

30。 30。 =。 =。

mousemove。

- “”。
- requestAnimationFrameCanvas。

Canvas。

requestAnimationFrame'requestAnimationFrame)setInterval'。

```
<!doctype html>
<html>
<head>
<style>
  body{ background-color: ivory; }
  #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

  function log(){console.log.apply(console,arguments);}

  // canvas variables
  var canvas=document.getElementById("canvas");
  var ctx=canvas.getContext("2d");
  var cw=canvas.width;
  var ch=canvas.height;
  // set canvas styling
  ctx.strokeStyle='skyblue';
  ctx.lineJoin='round';
  ctx.lineCap='round';
  ctx.lineWidth=6;

  // handle windows scrolling & resizing
  function reOffset(){
    var BB=canvas.getBoundingClientRect();
    offsetX=BB.left;
    offsetY=BB.top;
  }
  var offsetX,offsetY;
  reOffset();
  window.onscroll=function(e){ reOffset(); }
  window.onresize=function(e){ reOffset(); }

  // vars to save points created during mousemove handling
  var points=[];
  var lastLength=0;

  // start the animation loop
  requestAnimationFrame(draw);

  canvas.onmousemove=function(e){handleMouseMove(e);}

  function handleMouseMove(e){
    // tell the browser we're handling this event
```

```

    e.preventDefault();
    e.stopPropagation();

    // get the mouse position
    mouseX=parseInt(e.clientX-offsetX);
    mouseY=parseInt(e.clientY-offsetY);

    // save the mouse position in the points[] array
    // but don't draw anything
    points.push({x:mouseX,y:mouseY});
}

function draw(){
    // No additional points? Request another frame and return
    var length=points.length;
    if(length==lastLength){requestAnimationFrame(draw);return;}

    // draw the additional points
    var point=points[lastLength];
    ctx.beginPath();
    ctx.moveTo(point.x,point.y)
    for(var i=lastLength;i<length;i++){
        point=points[i];
        ctx.lineTo(point.x,point.y);
    }
    ctx.stroke();

    // request another animation loop
    requestAnimationFrame(draw);
}

}); // end window.onload
</script>
</head>
<body>
    <h4>Move mouse over Canvas to sketch</h4>
    <canvas id="canvas" width=512 height=512></canvas>
</body>
</html>

```

## Robert Penners

◦

“”

- X
- 
- 
- RGB◦
- ◦

“”

- 
- 
- 
-



“”

- -
- 
- /
- “”。

Robert Penner“”。

<https://github.com/danro/jquery-easing/blob/master/jquery.easing.js>

```
// t: elapsed time inside duration (currentTime-startTime),
// b: beginning value,
// c: total change from beginning value (endingValue-startingValue),
// d: total duration
var Easings={
  easeInQuad: function (t, b, c, d) {
    return c*(t/=d)*t + b;
  },
  easeOutQuad: function (t, b, c, d) {
    return -c *(t/=d)*(t-2) + b;
  },
  easeInOutQuad: function (t, b, c, d) {
    if ((t/=d/2) < 1) return c/2*t*t + b;
    return -c/2 * ((--t)*(t-2) - 1) + b;
  },
  easeInCubic: function (t, b, c, d) {
    return c*(t/=d)*t*t + b;
  },
  easeOutCubic: function (t, b, c, d) {
    return c*((t=t/d-1)*t*t + 1) + b;
  },
  easeInOutCubic: function (t, b, c, d) {
    if ((t/=d/2) < 1) return c/2*t*t*t + b;
    return c/2*((t-=2)*t*t + 2) + b;
  },
  easeInQuart: function (t, b, c, d) {
    return c*(t/=d)*t*t*t + b;
  },
  easeOutQuart: function (t, b, c, d) {
    return -c * ((t=t/d-1)*t*t*t - 1) + b;
  },
  easeInOutQuart: function (t, b, c, d) {
    if ((t/=d/2) < 1) return c/2*t*t*t*t + b;
    return -c/2 * ((t-=2)*t*t*t - 2) + b;
  },
  easeInQuint: function (t, b, c, d) {
    return c*(t/=d)*t*t*t*t + b;
  },
  easeOutQuint: function (t, b, c, d) {
    return c*((t=t/d-1)*t*t*t*t + 1) + b;
  },
  easeInOutQuint: function (t, b, c, d) {
    if ((t/=d/2) < 1) return c/2*t*t*t*t*t + b;
    return c/2*((t-=2)*t*t*t*t + 2) + b;
  },
  easeInSine: function (t, b, c, d) {
    return -c * Math.cos(t/d * (Math.PI/2)) + c + b;
  },
  easeOutSine: function (t, b, c, d) {
    return c * Math.sin(t/d * (Math.PI/2)) + b;
  },
  easeInOutSine: function (t, b, c, d) {
    return -c/2 * (Math.cos(Math.PI*t/d) - 1) + b + c/2;
  }
};
```

```

easeOutSine: function (t, b, c, d) {
    return c * Math.sin(t/d * (Math.PI/2)) + b;
},
easeInOutSine: function (t, b, c, d) {
    return -c/2 * (Math.cos(Math.PI*t/d) - 1) + b;
},
easeInExpo: function (t, b, c, d) {
    return (t==0) ? b : c * Math.pow(2, 10 * (t/d - 1)) + b;
},
easeOutExpo: function (t, b, c, d) {
    return (t==d) ? b+c : c * (-Math.pow(2, -10 * t/d) + 1) + b;
},
easeInOutExpo: function (t, b, c, d) {
    if (t==0) return b;
    if (t==d) return b+c;
    if ((t/=d/2) < 1) return c/2 * Math.pow(2, 10 * (t - 1)) + b;
    return c/2 * (-Math.pow(2, -10 * --t) + 2) + b;
},
easeInCirc: function (t, b, c, d) {
    return -c * (Math.sqrt(1 - (t/=d)*t) - 1) + b;
},
easeOutCirc: function (t, b, c, d) {
    return c * Math.sqrt(1 - (t=t/d-1)*t) + b;
},
easeInOutCirc: function (t, b, c, d) {
    if ((t/=d/2) < 1) return -c/2 * (Math.sqrt(1 - t*t) - 1) + b;
    return c/2 * (Math.sqrt(1 - (t-=2)*t) + 1) + b;
},
easeInElastic: function (t, b, c, d) {
    var s=1.70158;var p=0;var a=c;
    if (t==0) return b; if ((t/=d)==1) return b+c; if (!p) p=d*.3;
    if (a < Math.abs(c)) { a=c; var s=p/4; }
    else var s = p/(2*Math.PI) * Math.asin (c/a);
    return -(a*Math.pow(2,10*(t-=1)) * Math.sin( (t*d-s)*(2*Math.PI)/p )) + b;
},
easeOutElastic: function (t, b, c, d) {
    var s=1.70158;var p=0;var a=c;
    if (t==0) return b; if ((t/=d)==1) return b+c; if (!p) p=d*.3;
    if (a < Math.abs(c)) { a=c; var s=p/4; }
    else var s = p/(2*Math.PI) * Math.asin (c/a);
    return a*Math.pow(2,-10*t) * Math.sin( (t*d-s)*(2*Math.PI)/p ) + c + b;
},
easeInOutElastic: function (t, b, c, d) {
    var s=1.70158;var p=0;var a=c;
    if (t==0) return b; if ((t/=d/2)==2) return b+c; if (!p) p=d*(.3*1.5);
    if (a < Math.abs(c)) { a=c; var s=p/4; }
    else var s = p/(2*Math.PI) * Math.asin (c/a);
    if (t < 1) return -.5*(a*Math.pow(2,10*(t-=1)) * Math.sin( (t*d-s)*(2*Math.PI)/p )) + b;
    return a*Math.pow(2,-10*(t-=1)) * Math.sin( (t*d-s)*(2*Math.PI)/p )*.5 + c + b;
},
easeInBack: function (t, b, c, d, s) {
    if (s == undefined) s = 1.70158;
    return c*(t/=d)*t*((s+1)*t - s) + b;
},
easeOutBack: function (t, b, c, d, s) {
    if (s == undefined) s = 1.70158;
    return c*((t=t/d-1)*t*((s+1)*t + s) + 1) + b;
},
easeInOutBack: function (t, b, c, d, s) {
    if (s == undefined) s = 1.70158;
    if ((t/=d/2) < 1) return c/2*(t*t*((s*=(1.525))+1)*t - s)) + b;

```

```

        return c/2*((t-=2)*t*((s*=(1.525))+1)*t + s) + 2) + b;
    },
    easeInBounce: function (t, b, c, d) {
        return c - Easings.easeOutBounce (d-t, 0, c, d) + b;
    },
    easeOutBounce: function (t, b, c, d) {
        if ((t/=d) < (1/2.75)) {
            return c*(7.5625*t*t) + b;
        } else if (t < (2/2.75)) {
            return c*(7.5625*(t-=(1.5/2.75))*t + .75) + b;
        } else if (t < (2.5/2.75)) {
            return c*(7.5625*(t-=(2.25/2.75))*t + .9375) + b;
        } else {
            return c*(7.5625*(t-=(2.625/2.75))*t + .984375) + b;
        }
    },
    easeInOutBounce: function (t, b, c, d) {
        if (t < d/2) return Easings.easeInBounce (t*2, 0, c, d) * .5 + b;
        return Easings.easeOutBounce (t*2-d, 0, c, d) * .5 + c*.5 + b;
    },
};

```

```

// t: elapsed time inside duration (currentTime-startTime),
// b: beginning value,
// c: total change from beginning value (endingValue-startingValue),
// d: total duration
var Easings={
    easeInQuad: function (t, b, c, d) {
        return c*(t/=d)*t + b;
    },
    easeOutQuad: function (t, b, c, d) {
        return -c *(t/=d)*(t-2) + b;
    },
    easeInOutQuad: function (t, b, c, d) {
        if ((t/=d/2) < 1) return c/2*t*t + b;
        return -c/2 * ((--t)*(t-2) - 1) + b;
    },
    easeInCubic: function (t, b, c, d) {
        return c*(t/=d)*t*t + b;
    },
    easeOutCubic: function (t, b, c, d) {
        return c*((t=t/d-1)*t*t + 1) + b;
    },
    easeInOutCubic: function (t, b, c, d) {
        if ((t/=d/2) < 1) return c/2*t*t*t + b;
        return c/2*((t-=2)*t*t + 2) + b;
    },
    easeInQuart: function (t, b, c, d) {
        return c*(t/=d)*t*t*t + b;
    },
    easeOutQuart: function (t, b, c, d) {
        return -c * ((t=t/d-1)*t*t*t - 1) + b;
    },
    easeInOutQuart: function (t, b, c, d) {
        if ((t/=d/2) < 1) return c/2*t*t*t*t + b;
        return -c/2 * ((t-=2)*t*t*t - 2) + b;
    },
    easeInQuint: function (t, b, c, d) {
        return c*(t/=d)*t*t*t*t + b;
    },
};

```

```

easeOutQuint: function (t, b, c, d) {
    return c*((t=t/d-1)*t*t*t*t + 1) + b;
},
easeInOutQuint: function (t, b, c, d) {
    if ((t/=d/2) < 1) return c/2*t*t*t*t*t + b;
    return c/2*((t-=2)*t*t*t*t + 2) + b;
},
easeInSine: function (t, b, c, d) {
    return -c * Math.cos(t/d * (Math.PI/2)) + c + b;
},
easeOutSine: function (t, b, c, d) {
    return c * Math.sin(t/d * (Math.PI/2)) + b;
},
easeInOutSine: function (t, b, c, d) {
    return -c/2 * (Math.cos(Math.PI*t/d) - 1) + b;
},
easeInExpo: function (t, b, c, d) {
    return (t==0) ? b : c * Math.pow(2, 10 * (t/d - 1)) + b;
},
easeOutExpo: function (t, b, c, d) {
    return (t==d) ? b+c : c * (-Math.pow(2, -10 * t/d) + 1) + b;
},
easeInOutExpo: function (t, b, c, d) {
    if (t==0) return b;
    if (t==d) return b+c;
    if ((t/=d/2) < 1) return c/2 * Math.pow(2, 10 * (t - 1)) + b;
    return c/2 * (-Math.pow(2, -10 * --t) + 2) + b;
},
easeInCirc: function (t, b, c, d) {
    return -c * (Math.sqrt(1 - (t/=d)*t) - 1) + b;
},
easeOutCirc: function (t, b, c, d) {
    return c * Math.sqrt(1 - (t=t/d-1)*t) + b;
},
easeInOutCirc: function (t, b, c, d) {
    if ((t/=d/2) < 1) return -c/2 * (Math.sqrt(1 - t*t) - 1) + b;
    return c/2 * (Math.sqrt(1 - (t-=2)*t) + 1) + b;
},
easeInElastic: function (t, b, c, d) {
    var s=1.70158;var p=0;var a=c;
    if (t==0) return b; if ((t/=d)==1) return b+c; if (!p) p=d*.3;
    if (a < Math.abs(c)) { a=c; var s=p/4; }
    else var s = p/(2*Math.PI) * Math.asin (c/a);
    return -(a*Math.pow(2,10*(t-=1)) * Math.sin( (t*d-s)*(2*Math.PI)/p )) + b;
},
easeOutElastic: function (t, b, c, d) {
    var s=1.70158;var p=0;var a=c;
    if (t==0) return b; if ((t/=d)==1) return b+c; if (!p) p=d*.3;
    if (a < Math.abs(c)) { a=c; var s=p/4; }
    else var s = p/(2*Math.PI) * Math.asin (c/a);
    return a*Math.pow(2,-10*t) * Math.sin( (t*d-s)*(2*Math.PI)/p ) + c + b;
},
easeInOutElastic: function (t, b, c, d) {
    var s=1.70158;var p=0;var a=c;
    if (t==0) return b; if ((t/=d/2)==2) return b+c; if (!p) p=d*(.3*1.5);
    if (a < Math.abs(c)) { a=c; var s=p/4; }
    else var s = p/(2*Math.PI) * Math.asin (c/a);
    if (t < 1) return -.5*(a*Math.pow(2,10*(t-=1)) * Math.sin( (t*d-s)*(2*Math.PI)/p )) + b;
    return a*Math.pow(2,-10*(t-=1)) * Math.sin( (t*d-s)*(2*Math.PI)/p )*.5 + c + b;
},
easeInBack: function (t, b, c, d, s) {

```

```

    if (s == undefined) s = 1.70158;
    return c*(t/=d)*t*((s+1)*t - s) + b;
},
easeOutBack: function (t, b, c, d, s) {
    if (s == undefined) s = 1.70158;
    return c*((t=t/d-1)*t*((s+1)*t + s) + 1) + b;
},
easeInOutBack: function (t, b, c, d, s) {
    if (s == undefined) s = 1.70158;
    if ((t/=d/2) < 1) return c/2*(t*t*((s*=(1.525))+1)*t - s)) + b;
    return c/2*((t-=2)*t*((s*=(1.525))+1)*t + s) + 2) + b;
},
easeInBounce: function (t, b, c, d) {
    return c - Easings.easeOutBounce (d-t, 0, c, d) + b;
},
easeOutBounce: function (t, b, c, d) {
    if ((t/=d) < (1/2.75)) {
        return c*(7.5625*t*t) + b;
    } else if (t < (2/2.75)) {
        return c*(7.5625*(t-=(1.5/2.75))*t + .75) + b;
    } else if (t < (2.5/2.75)) {
        return c*(7.5625*(t-=(2.25/2.75))*t + .9375) + b;
    } else {
        return c*(7.5625*(t-=(2.625/2.75))*t + .984375) + b;
    }
},
easeInOutBounce: function (t, b, c, d) {
    if (t < d/2) return Easings.easeInBounce (t*2, 0, c, d) * .5 + b;
    return Easings.easeOutBounce (t*2-d, 0, c, d) * .5 + c*.5 + b;
},
};

```

## requestAnimationFrame

requestAnimationFrame60。 60fps。 120fps。

60(60 / FRAMES\_PER\_SECOND) % 1 === 0true。

```

const FRAMES_PER_SECOND = 30; // Valid values are 60,30,20,15,10...
// set the min time to render the next frame
const FRAME_MIN_TIME = (1000/60) * (60 / FRAMES_PER_SECOND) - (1000/60) * 0.5;
var lastFrameTime = 0; // the last frame time
function update(time){
    if(time-lastFrameTime < FRAME_MIN_TIME){ //skip the frame if the call is too early
        requestAnimationFrame(update);
        return; // return as there is nothing to do
    }
    lastFrameTime = time; // remember the time of the rendered frame
    // render the frame
    requestAnimationFrame(update); // get next frame
}
requestAnimationFrame(update); // start animation

```

[x0y0][x1y1]

[startXstartY][endXendY][xy]

```
// dx is the total distance to move in the X direction
var dx = endX - startX;

// dy is the total distance to move in the Y direction
var dy = endY - startY;

// use a pct (percentage) to travel the total distances
// start at 0% which == the starting point
// end at 100% which == then ending point
var pct=0;

// use dx & dy to calculate where the current [x,y] is at a given pct
var x = startX + dx * pct/100;
var y = startY + dx * pct/100;
```

```
// dx is the total distance to move in the X direction
var dx = endX - startX;

// dy is the total distance to move in the Y direction
var dy = endY - startY;

// use a pct (percentage) to travel the total distances
// start at 0% which == the starting point
// end at 100% which == then ending point
var pct=0;

// use dx & dy to calculate where the current [x,y] is at a given pct
var x = startX + dx * pct/100;
var y = startY + dx * pct/100;
```

<https://riptutorial.com/zh-TW/html5-canvas/topic/4822/>

## 4:

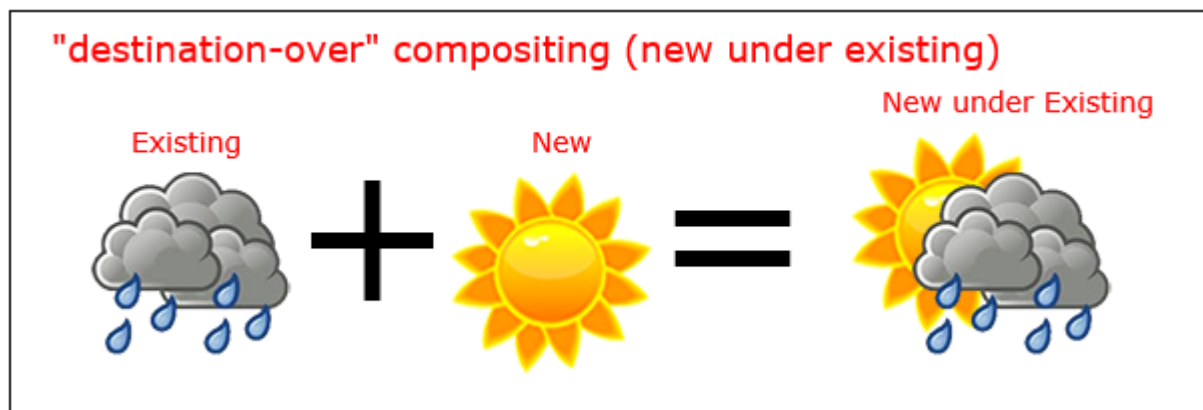
### Examples

“ ”

```
context.globalCompositeOperation = "destination-over"
```

“ ”。

```
context.globalCompositeOperation = "destination-over"
```



“ ”

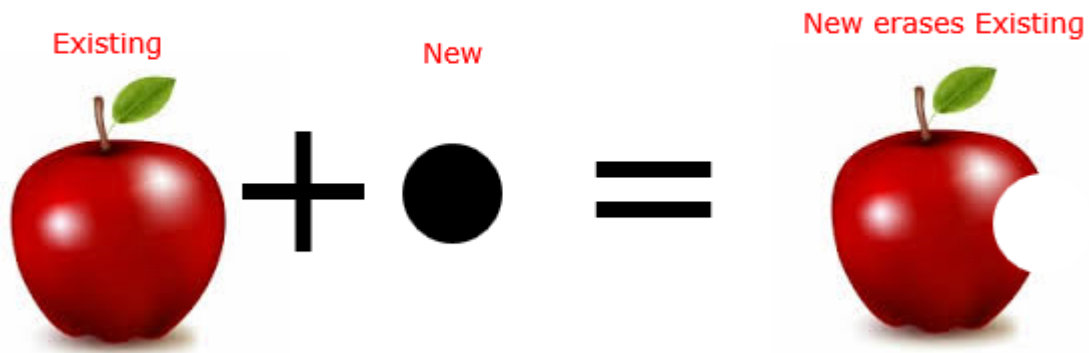
```
context.globalCompositeOperation = "destination-out"
```

“ ”。

- “ ”。

```
context.globalCompositeOperation = "destination-out"
```

## "destination-out" compositing (new erases existing)

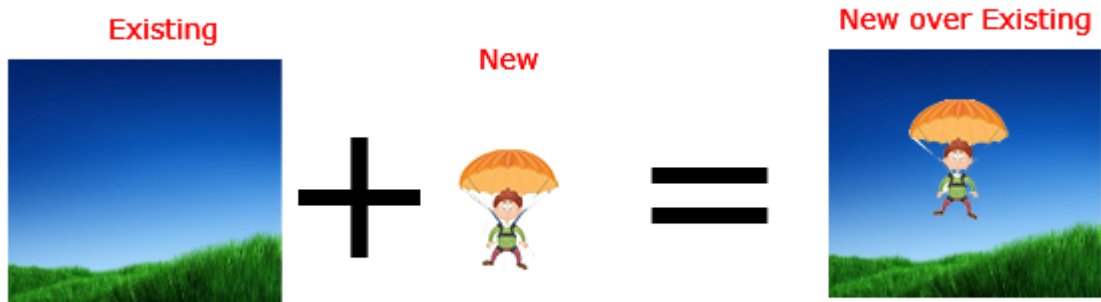


```
context.globalCompositeOperation = "source-over"
```

“source-over”[]。

```
context.globalCompositeOperation = "source-over"
```

## "source-over" compositing (new over existing)



“destination-in”

```
context.globalCompositeOperation = "destination-in"
```

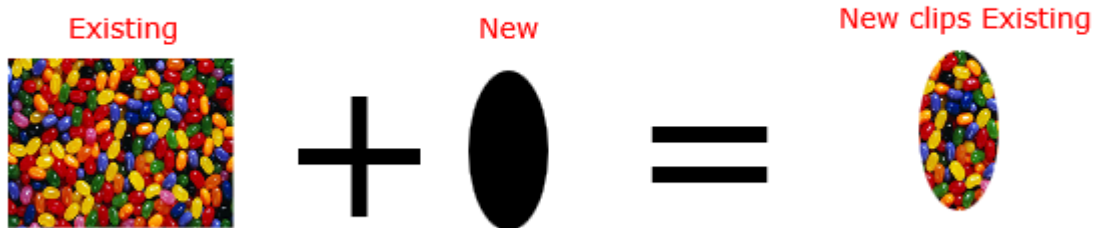
“destination-in”。

。

```
context.globalCompositeOperation = "destination-in"
```



## "destination-in" compositing (new clips existing)



## “source-in”

```
context.globalCompositeOperation = "source-in";
```

source-in°

°

```
context.globalCompositeOperation = "source-in";
```

## "source-in" compositing (existing clips new)



## “source-atop”

```
context.globalCompositeOperation = 'source-atop'
```

source-atop compositing°

```
context.globalCompositeOperation = 'source-atop'
```

## "source-atop" compositing (existing clips new)

Existing



New  
(Undesired outer shadow)



Existing clips New  
No outer shadow



“”

```
ctx.globalCompositeOperation = 'difference';
```

alpha

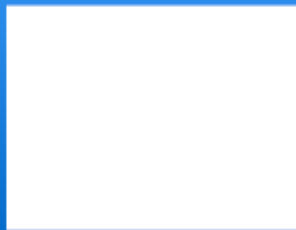
```
ctx.globalCompositeOperation = 'difference';
```

## ctx.globalCompositeOperation = "difference"

Image



FillRect "White"



“”

```
ctx.globalCompositeOperation = 'color';
```

alpha

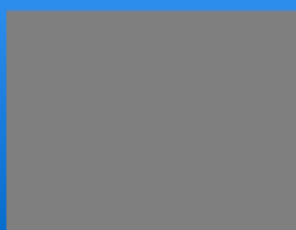
```
ctx.globalCompositeOperation = 'color';
```

## ctx.globalCompositeOperation = "color"

Image



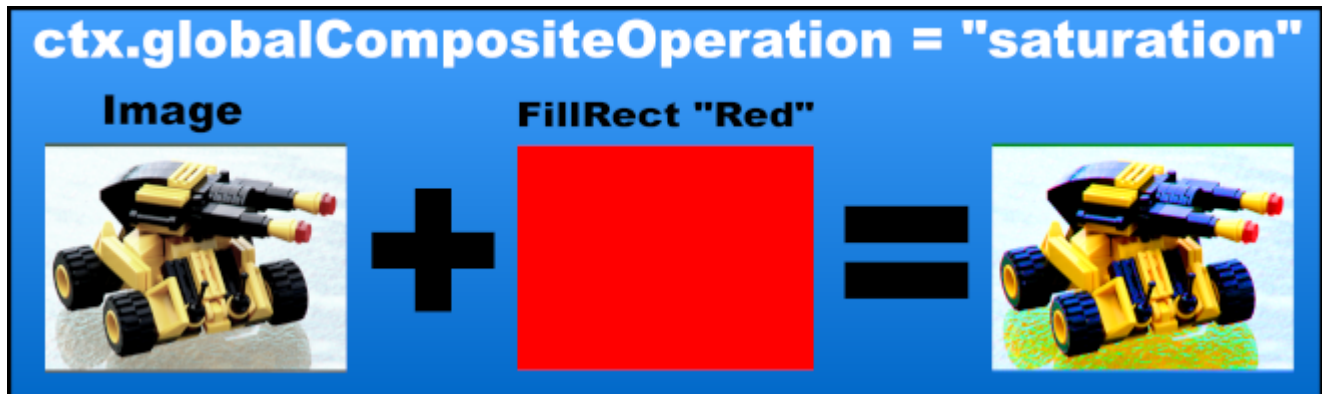
FillRect "#888"



“”

```
ctx.globalCompositeOperation = 'saturation';
```

```
ctx.globalCompositeOperation = 'saturation';
```



FX“”

FX

```
ctx.globalCompositeOperation = 'luminosity';
```

。

```
ctx.globalCompositeOperation = 'luminosity';
```



“globalAlpha”

```
context.globalAlpha=0.50
```

globalAlpha0.001.00。

globalAlpha1.00。

globalAlpha。

```
context.globalAlpha=0.50
```

<https://riptutorial.com/zh-TW/html5-canvas/topic/5547/>

## 5:

## Examples

◦

```
function cropImage(image, croppingCoords) {
    var cc = croppingCoords;
    var workCan = document.createElement("canvas"); // create a canvas
    workCan.width = Math.floor(cc.width); // set the canvas resolution to the cropped image
    size
    workCan.height = Math.floor(cc.height);
    var ctx = workCan.getContext("2d"); // get a 2D rendering interface
    ctx.drawImage(image, -Math.floor(cc.x), -Math.floor(cc.y)); // draw the image offset to
    place it correctly on the cropped region
    image.src = workCan.toDataURL(); // set the image source to the canvas as a data URL
    return image;
}
```

```
function cropImage(image, croppingCoords) {
    var cc = croppingCoords;
    var workCan = document.createElement("canvas"); // create a canvas
    workCan.width = Math.floor(cc.width); // set the canvas resolution to the cropped image
    size
    workCan.height = Math.floor(cc.height);
    var ctx = workCan.getContext("2d"); // get a 2D rendering interface
    ctx.drawImage(image, -Math.floor(cc.x), -Math.floor(cc.y)); // draw the image offset to
    place it correctly on the cropped region
    image.src = workCan.toDataURL(); // set the image source to the canvas as a data URL
    return image;
}
```

## Tained

◦ dataURL◦

```
vr image = new Image();
image.src = "file:///myLocalImage.png";
image.onload = function(){
    ctx.drawImage(this,0,0);
    ctx.getImageData(0,0,canvas.width,canvas.height); // throws a security error
}
```

◦

## “context.drawImage”Canvas

context.drawImage◦ ◦

JavaScript◦ JavaScriptJavaScriptimage.src◦ context.drawImage◦

## .drawImage

```
var img=new Image();
img.onload=start;
img.onerror=function(){alert(img.src+' failed');}
img.src="someImage.png";
function start(){
    // start() is called AFTER the image is fully loaded regardless
    // of start's position in the code
}
```

```
var img=new Image();
img.onload=start;
img.onerror=function(){alert(img.src+' failed');}
img.src="someImage.png";
function start(){
    // start() is called AFTER the image is fully loaded regardless
    // of start's position in the code
}
```

◦  
◦ ◦ ◦



◦ ◦ ◦ ◦



```
var image = new Image();
```

```

image.src = "imgURL";
image.onload = function(){
    scaleToFit(this);
}

function scaleToFit(img){
    // get the scale
    var scale = Math.min(canvas.width / img.width, canvas.height / img.height);
    // get the top left position of the image
    var x = (canvas.width / 2) - (img.width / 2) * scale;
    var y = (canvas.height / 2) - (img.height / 2) * scale;
    ctx.drawImage(img, x, y, img.width * scale, img.height * scale);
}

```

```

var image = new Image();
image.src = "imgURL";
image.onload = function(){
    scaleToFit(this);
}

function scaleToFit(img){
    // get the scale
    var scale = Math.min(canvas.width / img.width, canvas.height / img.height);
    // get the top left position of the image
    var x = (canvas.width / 2) - (img.width / 2) * scale;
    var y = (canvas.height / 2) - (img.height / 2) * scale;
    ctx.drawImage(img, x, y, img.width * scale, img.height * scale);
}

```

◦ ◦

<https://riptutorial.com/zh-TW/html5-canvas/topic/3210/>

## 6:

### Examples

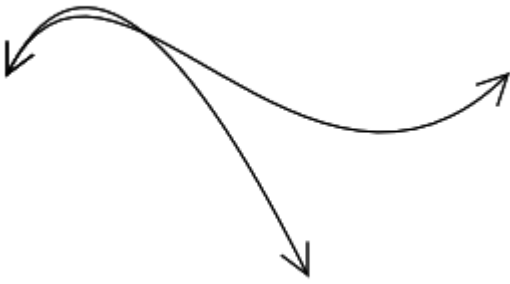


```
// Usage:
drawLineWithArrows(50,50,150,50,5,8,true,true);

// x0,y0: the line's starting point
// x1,y1: the line's ending point
// width: the distance the arrowhead perpendicularly extends away from the line
// height: the distance the arrowhead extends backward from the endpoint
// arrowStart: true/false directing to draw arrowhead at the line's starting point
// arrowEnd: true/false directing to draw arrowhead at the line's ending point

function drawLineWithArrows(x0,y0,x1,y1,aWidth,aLength,arrowStart,arrowEnd){
    var dx=x1-x0;
    var dy=y1-y0;
    var angle=Math.atan2(dy,dx);
    var length=Math.sqrt(dx*dx+dy*dy);
    //
    ctx.translate(x0,y0);
    ctx.rotate(angle);
    ctx.beginPath();
    ctx.moveTo(0,0);
    ctx.lineTo(length,0);
    if(arrowStart){
        ctx.moveTo(aLength,-aWidth);
        ctx.lineTo(0,0);
        ctx.lineTo(aLength,aWidth);
    }
    if(arrowEnd){
        ctx.moveTo(length-aLength,-aWidth);
        ctx.lineTo(length,0);
        ctx.lineTo(length-aLength,aWidth);
    }
    //
    ctx.stroke();
    ctx.setTransform(1,0,0,1,0,0);
}
```





```
// Usage:
var p0={x:50,y:100};
var p1={x:100,y:0};
var p2={x:200,y:200};
var p3={x:300,y:100};

cubicCurveArrowHeads(p0, p1, p2, p3, 15, true, true);

quadraticCurveArrowHeads(p0, p1, p2, 15, true, true);

// or use defaults true for both ends with arrow heads
cubicCurveArrowHeads(p0, p1, p2, p3, 15);

quadraticCurveArrowHeads(p0, p1, p2, 15);

// draws both cubic and quadratic bezier
function bezWithArrowheads(p0, p1, p2, p3, arrowLength, hasStartArrow, hasEndArrow) {
  var x, y, norm, ex, ey;
  function pointsToNormalisedVec(p,pp){
    var len;
    norm.y = pp.x - p.x;
    norm.x = -(pp.y - p.y);
    len = Math.sqrt(norm.x * norm.x + norm.y * norm.y);
    norm.x /= len;
    norm.y /= len;
    return norm;
  }

  var arrowWidth = arrowLength / 2;
  norm = {};
  // defaults to true for both arrows if arguments not included
  hasStartArrow = hasStartArrow === undefined || hasStartArrow === null ? true :
hasStartArrow;
  hasEndArrow = hasEndArrow === undefined || hasEndArrow === null ? true : hasEndArrow;
  ctx.beginPath();
  ctx.moveTo(p0.x, p0.y);
  if (p3 === undefined) {
    ctx.quadraticCurveTo(p1.x, p1.y, p2.x, p2.y);
    ex = p2.x; // get end point
    ey = p2.y;
    norm = pointsToNormalisedVec(p1,p2);
  } else {
    ctx.bezierCurveTo(p1.x, p1.y, p2.x, p2.y, p3.x, p3.y)
    ex = p3.x; // get end point
    ey = p3.y;
  }
}
```

```

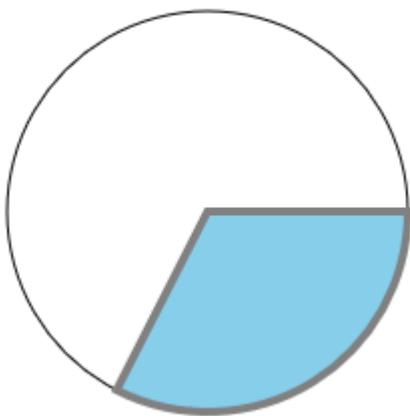
    norm = pointsToNormalisedVec(p2,p3);
  }
  if (hasEndArrow) {
    x = arrowWidth * norm.x + arrowLength * -norm.y;
    y = arrowWidth * norm.y + arrowLength * norm.x;
    ctx.moveTo(ex + x, ey + y);
    ctx.lineTo(ex, ey);
    x = arrowWidth * -norm.x + arrowLength * -norm.y;
    y = arrowWidth * -norm.y + arrowLength * norm.x;
    ctx.lineTo(ex + x, ey + y);
  }
  if (hasStartArrow) {
    norm = pointsToNormalisedVec(p0,p1);
    x = arrowWidth * norm.x - arrowLength * -norm.y;
    y = arrowWidth * norm.y - arrowLength * norm.x;
    ctx.moveTo(p0.x + x, p0.y + y);
    ctx.lineTo(p0.x, p0.y);
    x = arrowWidth * -norm.x - arrowLength * -norm.y;
    y = arrowWidth * -norm.y - arrowLength * norm.x;
    ctx.lineTo(p0.x + x, p0.y + y);
  }

  ctx.stroke();
}

function cubicCurveArrowheads(p0, p1, p2, p3, arrowLength, hasStartArrow, hasEndArrow) {
  bezWithArrowheads(p0, p1, p2, p3, arrowLength, hasStartArrow, hasEndArrow);
}
function quadraticCurveArrowheads(p0, p1, p2, arrowLength, hasStartArrow, hasEndArrow) {
  bezWithArrowheads(p0, p1, p2, undefined, arrowLength, hasStartArrow, hasEndArrow);
}

```

...°



```

// Usage
var wedge={
  cx:150, cy:150,
  radius:100,
  startAngle:0,
  endAngle:Math.PI*.65
}

drawWedge(wedge, 'skyblue', 'gray', 4);

```

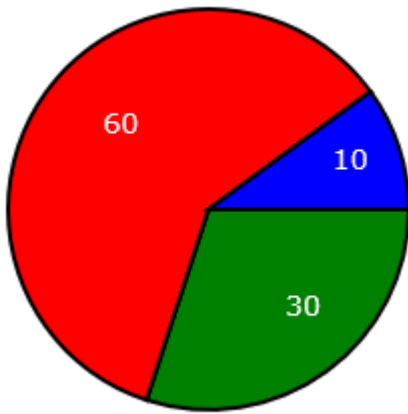
```
function drawWedge(w,fill,stroke,strokeWidth){
    ctx.beginPath();
    ctx.moveTo(w.cx, w.cy);
    ctx.arc(w.cx, w.cy, w.radius, w.startAngle, w.endAngle);
    ctx.closePath();
    ctx.fillStyle=fill;
    ctx.fill();
    ctx.strokeStyle=stroke;
    ctx.lineWidth=strokeWidth;
    ctx.stroke();
}
```



```
// Usage:
var arc={
    cx:150, cy:150,
    innerRadius:75, outerRadius:100,
    startAngle:-Math.PI/4, endAngle:Math.PI
}

drawArc(arc, 'skyblue', 'gray', 4);

function drawArc(a,fill,stroke,strokeWidth){
    ctx.beginPath();
    ctx.arc(a.cx,a.cy,a.innerRadius,a.startAngle,a.endAngle);
    ctx.arc(a.cx,a.cy,a.outerRadius,a.endAngle,a.startAngle,true);
    ctx.closePath();
    ctx.fillStyle=fill;
    ctx.strokeStyle=stroke;
    ctx.lineWidth=strokeWidth
    ctx.fill();
    ctx.stroke();
}
```



```
<!doctype html>
<html>
<head>
<style>
  body{ background-color:white; }
  #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

  var canvas = document.getElementById("canvas");
  var ctx = canvas.getContext("2d");
  ctx.lineWidth = 2;
  ctx.font = '14px verdana';

  var PI2 = Math.PI * 2;
  var myColor = ["Green", "Red", "Blue"];
  var myData = [30, 60, 10];
  var cx = 150;
  var cy = 150;
  var radius = 100;

  pieChart(myData, myColor);

  function pieChart(data, colors) {
    var total = 0;
    for (var i = 0; i < data.length; i++) {
      total += data[i];
    }

    var sweeps = []
    for (var i = 0; i < data.length; i++) {
      sweeps.push(data[i] / total * PI2);
    }

    var accumAngle = 0;
    for (var i = 0; i < sweeps.length; i++) {
      drawWedge(accumAngle, accumAngle + sweeps[i], colors[i], data[i]);
      accumAngle += sweeps[i];
    }
  }

  function drawWedge(startAngle, endAngle, fill, label) {
    // draw the wedge
```

```
ctx.beginPath();
ctx.moveTo(cx, cy);
ctx.arc(cx, cy, radius, startAngle, endAngle, false);
ctx.closePath();
ctx.fillStyle = fill;
ctx.strokeStyle = 'black';
ctx.fill();
ctx.stroke();

// draw the label
var midAngle = startAngle + (endAngle - startAngle) / 2;
var labelRadius = radius * .65;
var x = cx + (labelRadius) * Math.cos(midAngle);
var y = cy + (labelRadius) * Math.sin(midAngle);
ctx.fillStyle = 'white';
ctx.fillText(label, x, y);
}

}); // end $(function){}
</script>
</head>
<body>
  <canvas id="canvas" width=512 height=512></canvas>
</body>
</html>
```

<https://riptutorial.com/zh-TW/html5-canvas/topic/5492/>

# 7:

## Examples

“”

### Canvas

◦



◦ ◦

```
// this arc (==circle) is not draggable!!
context.beginPath();
context.arc(20, 30, 15, 0, Math.PI*2);
context.fillStyle='blue';
context.fill();
```

.....

- ...xy = [20,30]◦
- ...= 15◦
- .....◦ ◦

.....

### Canvas◦

xy == [20,30]◦

.....

### Canvas◦

- ◦
- ◦
- ◦
-

- 
- CanvasCanvas◦

## CanvasILLUSION

- ◦
- 
- 
- 
- .....

◦

```
// this arc (==circle) is not draggable!!
context.beginPath();
context.arc(20, 30, 15, 0, Math.PI*2);
context.fillStyle='blue';
context.fill();
```

“”

“”

## JavaScript“”◦

```
var myCircle = { x:30, y:20, radius:15 };
```

◦ ◦

## shape◦

```
var myCircle = { x:30, y:20, radius:15 };
```

## mousedown

◦ ◦ true / false isDraggingisDragging◦

## mousemove

mousemove◦ x,y◦

## mouseupmouseout

“isDragging”◦ ◦

◦

```
var myCircle = { x:30, y:20, radius:15 };
```

## Canvas

◦

◦ ◦ “”◦

◦ ◦ context.isPointInPath ◦ isPointInPath - 10◦

isPointInPathisPointInPath“”◦ “”isPointInPathstrokefill◦ /◦

◦ ◦

## Path

```
// canvas related vars
var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
var cw=canvas.width;
var ch=canvas.height;
document.body.appendChild(canvas);
canvas.style.border='1px solid red';

// used to calc canvas position relative to window
function reOffset(){
    var BB=canvas.getBoundingClientRect();
    offsetX=BB.left;
    offsetY=BB.top;
}
var offsetX,offsetY;
reOffset();
window.onscroll=function(e){ reOffset(); }
window.onresize=function(e){ reOffset(); }
canvas.onresize=function(e){ reOffset(); }

// save relevant information about shapes drawn on the canvas
var shapes=[];
// define one circle and save it in the shapes[] array
shapes.push( {x:20, y:20, radius:15, color:'blue'} );
// define one rectangle and save it in the shapes[] array
shapes.push( {x:100, y:-1, width:75, height:35, color:'red'} );
// define one triangle path and save it in the shapes[] array
shapes.push( {x:0, y:0, points:[{x:50,y:30},{x:75,y:60},{x:25,y:60}],color:'green'} );

// drag related vars
var isDragging=false;
var startX,startY;

// hold the index of the shape being dragged (if any)
var selectedShapeIndex;
```



```

// draw the shapes on the canvas
drawAll();

// listen for mouse events
canvas.onmousedown=handleMouseDown;
canvas.onmousemove=handleMouseMove;
canvas.onmouseup=handleMouseUp;
canvas.onmouseout=handleMouseOut;

// given mouse X & Y (mx & my) and shape object
// return true/false whether mouse is inside the shape
function isMouseInShape(mx,my,shape){
    if(shape.radius){
        // this is a circle
        var dx=mx-shape.x;
        var dy=my-shape.y;
        // math test to see if mouse is inside circle
        if(dx*dx+dy*dy<shape.radius*shape.radius){
            // yes, mouse is inside this circle
            return(true);
        }
    }else if(shape.width){
        // this is a rectangle
        var rLeft=shape.x;
        var rRight=shape.x+shape.width;
        var rTop=shape.y;
        var rBott=shape.y+shape.height;
        // math test to see if mouse is inside rectangle
        if( mx>rLeft && mx<rRight && my>rTop && my<rBott){
            return(true);
        }
    }else if(shape.points){
        // this is a polyline path
        // First redefine the path again (no need to stroke/fill!)
        defineIrregularPath(shape);
        // Then hit-test with isPointInPath
        if(ctx.isPointInPath(mx,my)){
            return(true);
        }
    }
    // the mouse isn't in any of the shapes
    return(false);
}

function handleMouseDown(e){
    // tell the browser we're handling this event
    e.preventDefault();
    e.stopPropagation();
    // calculate the current mouse position
    startX=parseInt(e.clientX-offsetX);
    startY=parseInt(e.clientY-offsetY);
    // test mouse position against all shapes
    // post result if mouse is in a shape
    for(var i=0;i<shapes.length;i++){
        if(isMouseInShape(startX,startY,shapes[i])){
            // the mouse is inside this shape
            // select this shape
            selectedShapeIndex=i;
            // set the isDragging flag
            isDragging=true;
            // and return (==stop looking for

```

```

        // further shapes under the mouse)
        return;
    }
}

function handleMouseUp(e) {
    // return if we're not dragging
    if(!isDragging){return;}
    // tell the browser we're handling this event
    e.preventDefault();
    e.stopPropagation();
    // the drag is over -- clear the isDragging flag
    isDragging=false;
}

function handleMouseOut(e) {
    // return if we're not dragging
    if(!isDragging){return;}
    // tell the browser we're handling this event
    e.preventDefault();
    e.stopPropagation();
    // the drag is over -- clear the isDragging flag
    isDragging=false;
}

function handleMouseMove(e) {
    // return if we're not dragging
    if(!isDragging){return;}
    // tell the browser we're handling this event
    e.preventDefault();
    e.stopPropagation();
    // calculate the current mouse position
    mouseX=parseInt(e.clientX-offsetX);
    mouseY=parseInt(e.clientY-offsetY);
    // how far has the mouse dragged from its previous mousemove position?
    var dx=mouseX-startX;
    var dy=mouseY-startY;
    // move the selected shape by the drag distance
    var selectedShape=shapes[selectedShapeIndex];
    selectedShape.x+=dx;
    selectedShape.y+=dy;
    // clear the canvas and redraw all shapes
    drawAll();
    // update the starting drag position (== the current mouse position)
    startX=mouseX;
    startY=mouseY;
}

// clear the canvas and
// redraw all shapes in their current positions
function drawAll(){
    ctx.clearRect(0,0,cw,ch);
    for(var i=0;i<shapes.length;i++){
        var shape=shapes[i];
        if(shape.radius){
            // it's a circle
            ctx.beginPath();
            ctx.arc(shape.x,shape.y,shape.radius,0,Math.PI*2);
            ctx.fillStyle=shape.color;
            ctx.fill();
        }
    }
}

```

```

    }else if(shape.width){
        // it's a rectangle
        ctx.fillStyle=shape.color;
        ctx.fillRect(shape.x,shape.y,shape.width,shape.height);
    }else if(shape.points){
        // its a polyline path
        defineIrregularPath(shape);
        ctx.fillStyle=shape.color;
        ctx.fill();
    }
}

function defineIrregularPath(shape){
    var points=shape.points;
    ctx.beginPath();
    ctx.moveTo(shape.x+points[0].x,shape.y+points[0].y);
    for(var i=1;i<points.length;i++){
        ctx.lineTo(shape.x+points[i].x,shape.y+points[i].y);
    }
    ctx.closePath();
}

```

“”

“”。

## Canvas

```

// canvas related vars
var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
canvas.width=378;
canvas.height=378;
var cw=canvas.width;
var ch=canvas.height;
document.body.appendChild(canvas);
canvas.style.border='1px solid red';

// used to calc canvas position relative to window
function reOffset(){
    var BB=canvas.getBoundingClientRect();
    offsetX=BB.left;
    offsetY=BB.top;
}
var offsetX,offsetY;
reOffset();
window.onscroll=function(e){ reOffset(); }
window.onresize=function(e){ reOffset(); }
canvas.onresize=function(e){ reOffset(); }

// save relevant information about shapes drawn on the canvas
var shapes=[];

// drag related vars
var isDragging=false;
var startX,startY;

// hold the index of the shape being dragged (if any)

```

```

var selectedShapeIndex;

// load the image
var card=new Image();
card.onload=function(){
    // define one image and save it in the shapes[] array
    shapes.push( {x:30, y:10, width:127, height:150, image:card} );
    // draw the shapes on the canvas
    drawAll();
    // listen for mouse events
    canvas.onmousedown=handleMouseDown;
    canvas.onmousemove=handleMouseMove;
    canvas.onmouseup=handleMouseUp;
    canvas.onmouseout=handleMouseOut;
};
// put your image src here!
card.src='https://dl.dropboxusercontent.com/u/139992952/stackoverflow/card.png';

// given mouse X & Y (mx & my) and shape object
// return true/false whether mouse is inside the shape
function isMouseInShape(mx,my,shape){
    // is this shape an image?
    if(shape.image){
        // this is a rectangle
        var rLeft=shape.x;
        var rRight=shape.x+shape.width;
        var rTop=shape.y;
        var rBott=shape.y+shape.height;
        // math test to see if mouse is inside image
        if( mx>rLeft && mx<rRight && my>rTop && my<rBott){
            return(true);
        }
    }
    // the mouse isn't in any of this shapes
    return(false);
}

function handleMouseDown(e){
    // tell the browser we're handling this event
    e.preventDefault();
    e.stopPropagation();
    // calculate the current mouse position
    startX=parseInt(e.clientX-offsetX);
    startY=parseInt(e.clientY-offsetY);
    // test mouse position against all shapes
    // post result if mouse is in a shape
    for(var i=0;i<shapes.length;i++){
        if(isMouseInShape(startX,startY,shapes[i])){
            // the mouse is inside this shape
            // select this shape
            selectedShapeIndex=i;
            // set the isDragging flag
            isDragging=true;
            // and return (==stop looking for
            // further shapes under the mouse)
            return;
        }
    }
}

```

```

function handleMouseUp(e){
    // return if we're not dragging
    if(!isDragging){return;}
    // tell the browser we're handling this event
    e.preventDefault();
    e.stopPropagation();
    // the drag is over -- clear the isDragging flag
    isDragging=false;
}

function handleMouseOut(e){
    // return if we're not dragging
    if(!isDragging){return;}
    // tell the browser we're handling this event
    e.preventDefault();
    e.stopPropagation();
    // the drag is over -- clear the isDragging flag
    isDragging=false;
}

function handleMouseMove(e){
    // return if we're not dragging
    if(!isDragging){return;}
    // tell the browser we're handling this event
    e.preventDefault();
    e.stopPropagation();
    // calculate the current mouse position
    mouseX=parseInt(e.clientX-offsetX);
    mouseY=parseInt(e.clientY-offsetY);
    // how far has the mouse dragged from its previous mousemove position?
    var dx=mouseX-startX;
    var dy=mouseY-startY;
    // move the selected shape by the drag distance
    var selectedShape=shapes[selectedShapeIndex];
    selectedShape.x+=dx;
    selectedShape.y+=dy;
    // clear the canvas and redraw all shapes
    drawAll();
    // update the starting drag position (== the current mouse position)
    startX=mouseX;
    startY=mouseY;
}

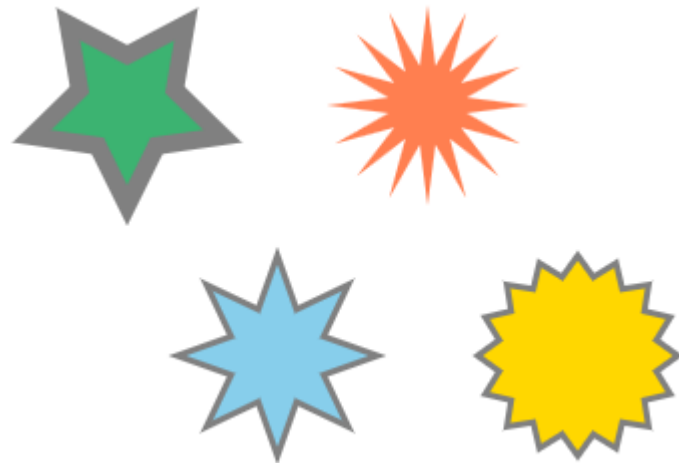
// clear the canvas and
// redraw all shapes in their current positions
function drawAll(){
    ctx.clearRect(0,0,cw,ch);
    for(var i=0;i<shapes.length;i++){
        var shape=shapes[i];
        if(shape.image){
            // it's an image
            ctx.drawImage(shape.image,shape.x,shape.y);
        }
    }
}

```

<https://riptutorial.com/zh-TW/html5-canvas/topic/5318/>

## 8:

### Examples



```
// Usage:
drawStar(75,75,5,50,25,'mediumseagreen','gray',9);
drawStar(150,200,8,50,25,'skyblue','gray',3);
drawStar(225,75,16,50,20,'coral','transparent',0);
drawStar(300,200,16,50,40,'gold','gray',3);

// centerX, centerY: the center point of the star
// points: the number of points on the exterior of the star
// inner: the radius of the inner points of the star
// outer: the radius of the outer points of the star
// fill, stroke: the fill and stroke colors to apply
// line: the linewidth of the stroke

function drawStar(centerX, centerY, points, outer, inner, fill, stroke, line) {
  // define the star
  ctx.beginPath();
  ctx.moveTo(centerX, centerY+outer);
  for (var i=0; i < 2*points+1; i++) {
    var r = (i%2 == 0)? outer : inner;
    var a = Math.PI * i/points;
    ctx.lineTo(centerX + r*Math.sin(a), centerY + r*Math.cos(a));
  };
  ctx.closePath();
  // draw
  ctx.fillStyle=fill;
  ctx.fill();
  ctx.strokeStyle=stroke;
  ctx.lineWidth=line;
  ctx.stroke()
}
```



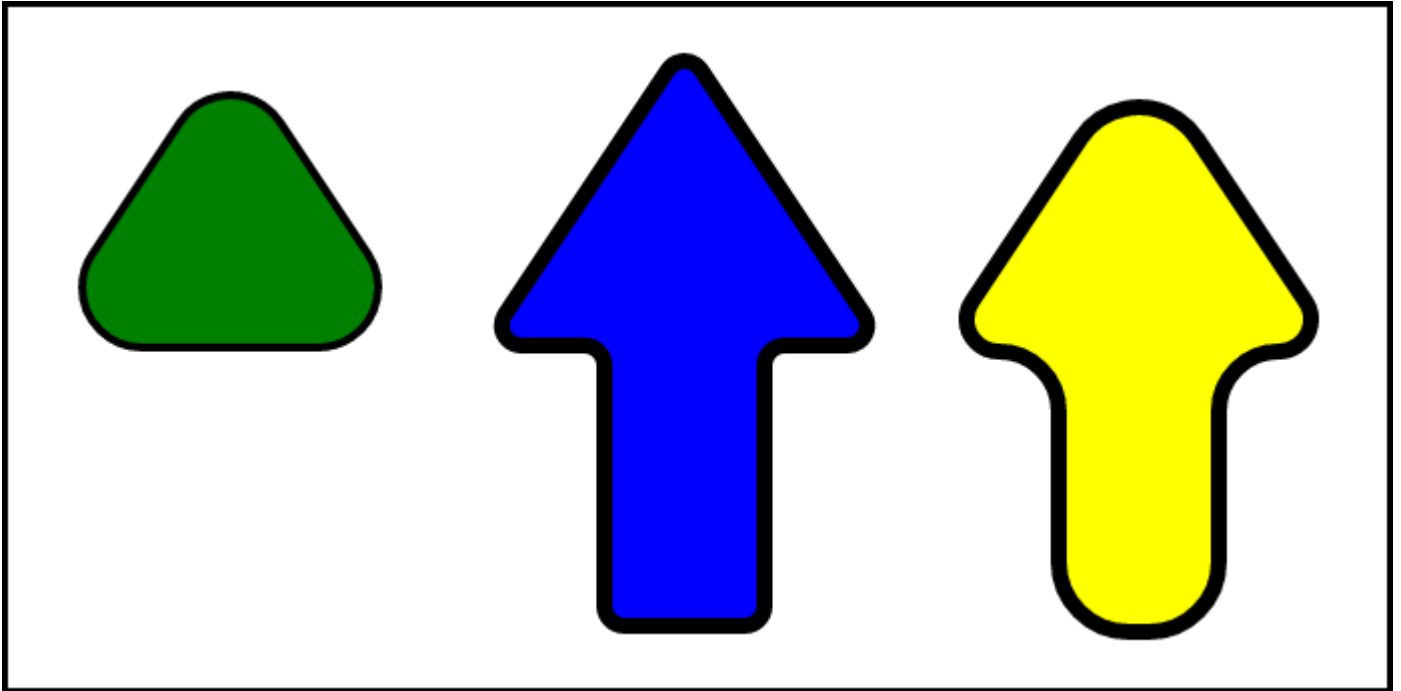
```
// Usage:
drawRegularPolygon(3,25,75,50,6,'gray','red',0);
drawRegularPolygon(5,25,150,50,6,'gray','gold',0);
drawRegularPolygon(6,25,225,50,6,'gray','lightblue',0);
drawRegularPolygon(10,25,300,50,6,'gray','lightgreen',0);

function
drawRegularPolygon(sideCount,radius,centerX,centerY,strokeWidth,strokeColor,fillColor,rotationRadians)

    var angles=Math.PI*2/sideCount;
    ctx.translate(centerX,centerY);
    ctx.rotate(rotationRadians);
    ctx.beginPath();
    ctx.moveTo(radius,0);
    for(var i=1;i<sideCount;i++){
        ctx.rotate(angles);
        ctx.lineTo(radius,0);
    }
    ctx.closePath();
    ctx.fillStyle=fillColor;
    ctx.strokeStyle = strokeColor;
    ctx.lineWidth = strokeWidth;
    ctx.stroke();
    ctx.fill();
    ctx.rotate(angles*-(sideCount-1));
    ctx.rotate(-rotationRadians);
    ctx.translate(-centerX,-centerY);
}
```

◦

[{x:?,y:?},{x:?,y:?},...,{x:?,y:?}]◦ ◦



```
var triangle = [
    { x: 200, y : 50 },
    { x: 300, y : 200 },
    { x: 100, y : 200 }
];
var cornerRadius = 30;
ctx.lineWidth = 4;
ctx.fillStyle = "Green";
ctx.strokeStyle = "black";
ctx.beginPath(); // start a new path
roundedPoly(triangle, cornerRadius);
ctx.fill();
ctx.stroke();
```

```
var triangle = [
    { x: 200, y : 50 },
    { x: 300, y : 200 },
    { x: 100, y : 200 }
];
var cornerRadius = 30;
ctx.lineWidth = 4;
ctx.fillStyle = "Green";
ctx.strokeStyle = "black";
ctx.beginPath(); // start a new path
roundedPoly(triangle, cornerRadius);
ctx.fill();
ctx.stroke();
```

<https://riptutorial.com/zh-TW/html5-canvas/topic/5493/>



9:

2D.

- 
- 
- 
- 
- 
- 
- 
- JPG / JPEG
- PNG
- GIF
- SVG
- M-JPEG
- WEBM
- WebP

Wiki

3“jpeg”“png”“gif”。

## JPEG

JPEG。 JPEG。

Canvas canvas.toDataURL canvas.toBlob JPEG。 JPEGJPG。 。

## JPEG

**PNG**

PNGAlpha。JPG。

alphaPNGui。 JPEG。 。

# PNGJavascript API。

```
32RGBAcanvas.toDataURLcanvas.toBlobPNG。 PNG。
```

PNG

## GIF

GIF。 GIF256。 gif。 Gif

PNGGIFJavascript API。 GIFAPI。

## Examples

```
var image = new Image(); // see note on creating an image
image.src = "imageURL";
image.onload = function(){
    ctx.drawImage(this,0,0);
}
```

- `new Image()`
- `document.createElement("img")`
- `<img src = 'imageUrl' id='myImage'>`HTML`document.getElementById('myImage')`

HTMLImageElement

### Image.src

srcURLdataURL。。

- `image.src = "http://my.domain.com/images/myImage.jpg"`
- `image.src = "data:image/gif;base64,R0lGODlhAQABAIAAAUEBAAACwAAAAAQABAAACakQBADS="` \*

\* dataURL1 x 1gif

src。 syncrionse/onload。

`document.getElementById("myImage")` src。 HTMLImageElement.complete HTMLImageElement.complete true  
。

- 
- 
- `src""`

。。

```
var image = new Image(); // see note on creating an image
image.src = "imageURL";
image.onload = function(){
    ctx.drawImage(this,0,0);
}
```

`image.onerror = myImgErrorHandler`。

### svg

### SVG

SVGHTMLImagedrawImage()。

```
var image = new Image();
image.onload = function(){
```

```

    ctx.drawImage(this, 0,0);
}
image.src = "someFile.SVG";

```

SVG。

SVG。

- **HTMLImageElement <img> SVG**

SVGImage <image/> xlink:href <use xlink:href="anImage.SVG#anElement"/> funcIRI url()

HTMLImageSVG。

SVG Image。

HTMLImageSVG。 dataURI。

- **<svg> widthheight。**

。 。 Blink。

- **SVG 。**

Internet Explorer<EdgeSVG<foreignObject>Safari 9。

。

。 /。

stackoverflow[HTML5 canvas](#)。

。 。 。

```

// It is assumed you know how to add a canvas and correctly size it.
var canvas = document.getElementById("myCanvas"); // get the canvas from the page
var ctx = canvas.getContext("2d");
var videoContainer; // object to hold video and associated info

```

```

// It is assumed you know how to add a canvas and correctly size it.
var canvas = document.getElementById("myCanvas"); // get the canvas from the page
var ctx = canvas.getContext("2d");
var videoContainer; // object to hold video and associated info

```

。 。

。 oncanplay。

```

// It is assumed you know how to add a canvas and correctly size it.
var canvas = document.getElementById("myCanvas"); // get the canvas from the page
var ctx = canvas.getContext("2d");
var videoContainer; // object to hold video and associated info

```

oncanplaythrough 。

```
// It is assumed you know how to add a canvas and correctly size it.
var canvas = document.getElementById("myCanvas"); // get the canvas from the page
var ctx = canvas.getContext("2d");
var videoContainer; // object to hold video and associated info
```

canPlay。

## can playonload

```
// It is assumed you know how to add a canvas and correctly size it.
var canvas = document.getElementById("myCanvas"); // get the canvas from the page
var ctx = canvas.getContext("2d");
var videoContainer; // object to hold video and associated info
```

。 。 60fps。 w。 。

FX。

```
// It is assumed you know how to add a canvas and correctly size it.
var canvas = document.getElementById("myCanvas"); // get the canvas from the page
var ctx = canvas.getContext("2d");
var videoContainer; // object to hold video and associated info
```

。 。 。 。

```
// It is assumed you know how to add a canvas and correctly size it.
var canvas = document.getElementById("myCanvas"); // get the canvas from the page
var ctx = canvas.getContext("2d");
var videoContainer; // object to hold video and associated info
```

```
// It is assumed you know how to add a canvas and correctly size it.
var canvas = document.getElementById("myCanvas"); // get the canvas from the page
var ctx = canvas.getContext("2d");
var videoContainer; // object to hold video and associated info
```

。 。 MDN HTMLMediaElement。

ctx.getImageData。 canvas.toDataURL。 。

。

。

## webM

WebM。

```

name = "CanvasCapture"; // Placed into the Mux and Write Application Name fields of the WebM
header
quality = 0.7; // good quality 1 Best < 0.7 ok to poor
fps = 30; // I have tried all sorts of frame rates and all seem to work
        // Do some test to workout what your machine can handle as there
        // is a lot of variation between machines.
var video = new Groover.Video(fps,quality,name)
function capture(){
    if(video.timecode < 5000){ // 5 seconds
        setTimeout(capture,video.frameDelay);
    }else{
        var videoElement = document.createElement("video");
        videoElement.src = URL.createObjectURL(video.toBlob());
        document.body.appendChild(videoElement);
        video = undefined; // DeReference as it is memory hungry.
        return;
    }
    // first frame sets the video size
    video.addFrame(canvas); // Add current canvas frame
}
capture(); // start capture

```

◦ ◦ 50fpsHD 1080◦

Wammy◦ 30◦

webP◦ ChromewebP◦ FirefoxEdgewebP◦Libwebp JavascriptJavascriptWebP◦ webp  
◦

webM◦WhammyJavascript WebM

```

name = "CanvasCapture"; // Placed into the Mux and Write Application Name fields of the WebM
header
quality = 0.7; // good quality 1 Best < 0.7 ok to poor
fps = 30; // I have tried all sorts of frame rates and all seem to work
        // Do some test to workout what your machine can handle as there
        // is a lot of variation between machines.
var video = new Groover.Video(fps,quality,name)
function capture(){
    if(video.timecode < 5000){ // 5 seconds
        setTimeout(capture,video.frameDelay);
    }else{
        var videoElement = document.createElement("video");
        videoElement.src = URL.createObjectURL(video.toBlob());
        document.body.appendChild(videoElement);
        video = undefined; // DeReference as it is memory hungry.
        return;
    }
    // first frame sets the video size
    video.addFrame(canvas); // Add current canvas frame
}
capture(); // start capture

```

<https://riptutorial.com/zh-TW/html5-canvas/topic/3689/>

# 10:

## Examples

◦ ◦

fillText◦

```
var canvas = document.getElementById('canvas');  
var ctx = canvas.getContext('2d');  
ctx.fillText("My text", 0, 0);
```

fillText

- 1.
2. x
3. y

◦ 200200px

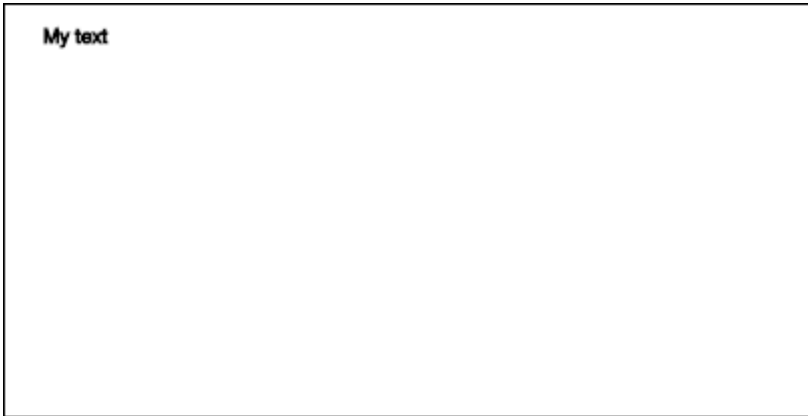
```
var canvas = document.getElementById('canvas');  
var ctx = canvas.getContext('2d');  
ctx.fillText("My text", 0, 0);
```



My text

strokeText

```
var canvas = document.getElementById('canvas');  
var ctx = canvas.getContext('2d');  
ctx.fillText("My text", 0, 0);
```



sans-serif 10px fillText strokeText ◦ ◦

fillText strokeText ◦ canvas API ◦

font

- 
- 
- 
- font-size / line-height
- 

```
ctx.font = "italic small-caps bold 40px Helvetica, Arial, sans-serif";  
ctx.fillText("My text", 20, 50);
```



textAlign

- 
- 
- 
- 
- 

```
ctx.font = "italic small-caps bold 40px Helvetica, Arial, sans-serif";  
ctx.fillText("My text", 20, 50);
```

Native Canvas API ◦ ◦

```
function wrapText(text, x, y, maxWidth, fontSize, fontFace){
  var firstY=y;
  var words = text.split(' ');
  var line = '';
  var lineHeight=fontSize*1.286; // a good approx for 10-18px sizes

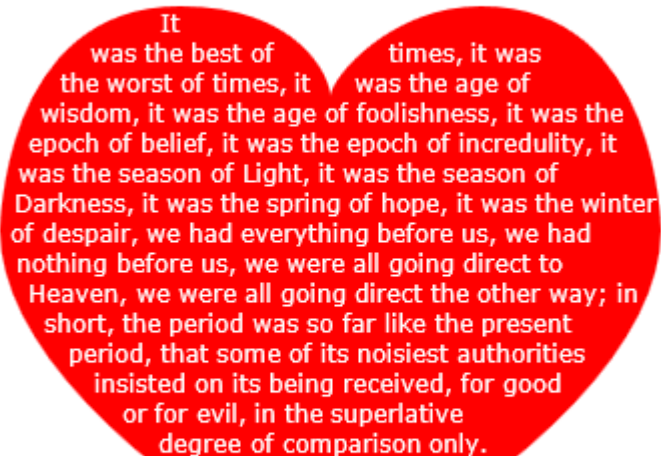
  ctx.font=fontSize+" "+fontFace;
  ctx.textBaseline='top';

  for(var n = 0; n < words.length; n++) {
    var testLine = line + words[n] + ' ';
    var metrics = ctx.measureText(testLine);
    var testWidth = metrics.width;
    if(testWidth > maxWidth) {
      ctx.fillText(line, x, y);
      if(n<words.length-1){
        line = words[n] + ' ';
        y += lineHeight;
      }
    }
    else {
      line = testLine;
    }
  }
  ctx.fillText(line, x, y);
}
```

◦

◦

/◦



It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness, it was the epoch of belief, it was the epoch of incredulity, it was the season of Light, it was the season of Darkness, it was the spring of hope, it was the winter of despair, we had everything before us, we had nothing before us, we were all going direct to Heaven, we were all going direct the other way; in short, the period was so far like the present period, that some of its noisiest authorities insisted on its being received, for good or for evil, in the superlative degree of comparison only.



```

<!doctype html>
<html>
<head>
<style>
    body{ background-color:white; padding:10px; }
    #canvas{border:1px solid red;}
</style>
<script>
window.onload=(function(){

    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");
    var cw=canvas.width;
    var ch=canvas.height;

    var fontsize=12;
    var fontface='verdana';
    var lineHeight=parseInt(fontsize*1.286);

    var text='It was the best of times, it was the worst of times, it was the age of wisdom,
it was the age of foolishness, it was the epoch of belief, it was the epoch of incredulity, it
was the season of Light, it was the season of Darkness, it was the spring of hope, it was the
winter of despair, we had everything before us, we had nothing before us, we were all going
direct to Heaven, we were all going direct the other way; in short, the period was so far like
the present period, that some of its noisiest authorities insisted on its being received, for
good or for evil, in the superlative degree of comparison only.';
    var words=text.split(' ');
    var wordWidths=[];
    ctx.font=fontsize+'px '+fontface;
    for(var i=0;i<words.length;i++){ wordWidths.push(ctx.measureText(words[i]).width); }
    var spaceWidth=ctx.measureText(' ').width;
    var wordIndex=0
    var data=[];

    // Demo: draw Heart
    // Note: the shape can be ANY opaque drawing -- even an image
    ctx.scale(3,3);
    ctx.beginPath();
    ctx.moveTo(75,40);
    ctx.bezierCurveTo(75,37,70,25,50,25);
    ctx.bezierCurveTo(20,25,20,62.5,20,62.5);
    ctx.bezierCurveTo(20,80,40,102,75,120);
    ctx.bezierCurveTo(110,102,130,80,130,62.5);
    ctx.bezierCurveTo(130,62.5,130,25,100,25);
    ctx.bezierCurveTo(85,25,75,37,75,40);
    ctx.fillStyle='red';
    ctx.fill();
    ctx.setTransform(1,0,0,1,0,0);

    // fill heart with text
    ctx.fillStyle='white';
    var imgDataData=ctx.getImageData(0,0,cw,ch).data;
    for(var i=0;i<imgDataData.length;i+=4){
        data.push(imgDataData[i+3]);
    }
    placeWords();

    // draw words sequentially into next available block of
    //     available opaque pixels
    function placeWords(){
        var sx=0;

```

```

var sy=0;
var y=0;
var wordIndex=0;
ctx.textBaseline='top';
while(y<ch && wordIndex<words.length){
  sx=0;
  sy=y;
  var startingIndex=wordIndex;
  while(sx<cw && wordIndex<words.length){
    var x=getRect(sx,sy,lineHeight);
    var available=x-sx;
    var spacer=spaceWidth; // spacer=0 to have no left margin
    var w=spacer+wordWidths[wordIndex];
    while(available>=w){
      ctx.fillText(words[wordIndex],spacer+sx,sy);
      sx+=w;
      available-=w;
      spacer=spaceWidth;
      wordIndex++;
      w=spacer+wordWidths[wordIndex];
    }
    sx=x+1;
  }
  y=(wordIndex>startingIndex)?y+lineHeight:y+1;
}
}

```

// find a rectangular block of opaque pixels

```

function getRect(sx,sy,height){
  var x=sx;
  var y=sy;
  var ok=true;
  while(ok){
    if(data[y*cw+x]<250){ok=false;}
    y++;
    if(y>=sy+height){
      y=sy;
      x++;
      if(x>=cw){ok=false;}
    }
  }
  return(x);
}

```

```

}); // end $(function(){});

```

```

</script>

```

```

</head>

```

```

<body>

```

```

  <h4>Note: the shape must be closed and alpha=250 inside</h4>

```

```

  <canvas id="canvas" width=400 height=400></canvas>

```

```

</body>

```

```

</html>

```

◦

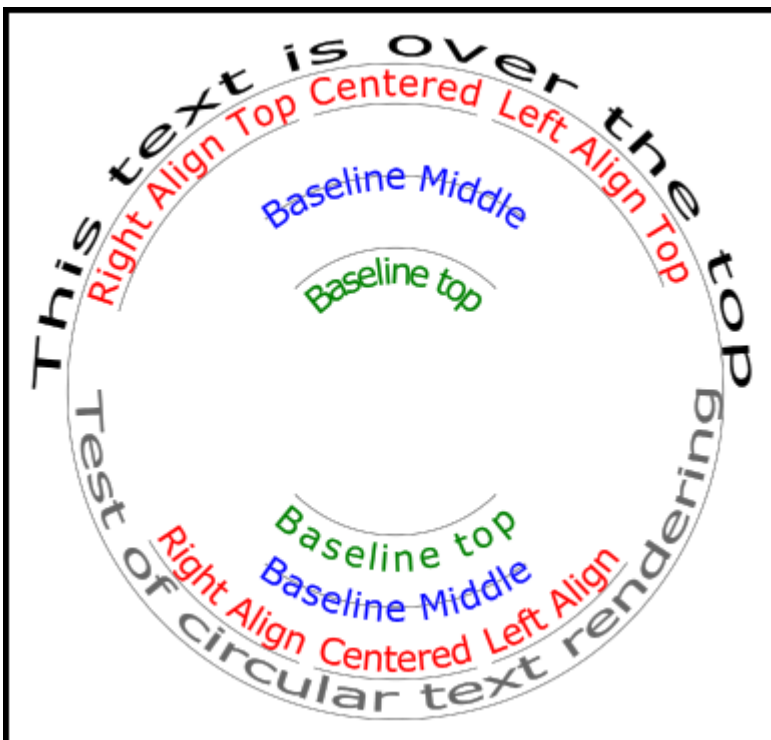
◦ image.onload◦

# Water!

```
function drawImageInsideText(canvas,x,y,img,text,font){
    var c=canvas.cloneNode();
    var ctx=c.getContext('2d');
    ctx.font=font;
    ctx.fillText(text,x,y);
    ctx.globalCompositeOperation='source-atop';
    ctx.drawImage(img,0,0);
    canvas.getContext('2d').drawImage(c,0,0);
}
```

- 
- CanvasRenderingContext2D◦

stackoverflow ◦



2D3◦

- **ctx.fillTextCircleTexttextxyradiusstartendforward;**
- **ctx.strokeCircleTexttextxyradiusstartendforward;**
- **ctx.measureCircleTexttextradius;**

```

(function(){
    const FILL = 0;          // const to indicate filltext render
    const STROKE = 1;
    var renderType = FILL; // used internal to set fill or stroke text
    const multiplyCurrentTransform = true; // if true Use current transform when rendering
                                         // if false use absolute coordinates which is a
little quicker
                                         // after render the currentTransform is restored to
default transform

    // measure circle text
    // ctx: canvas context
    // text: string of text to measure
    // r: radius in pixels
    //
    // returns the size metrics of the text
    //
    // width: Pixel width of text
    // angularWidth : angular width of text in radians
    // pixelAngularSize : angular width of a pixel in radians
    var measure = function(ctx, text, radius){
        var textWidth = ctx.measureText(text).width; // get the width of all the text
        return {
            width           : textWidth,
            angularWidth    : (1 / radius) * textWidth,
            pixelAngularSize : 1 / radius
        };
    }

    // displays text along a circle
    // ctx: canvas context
    // text: string of text to measure
    // x,y: position of circle center
    // r: radius of circle in pixels
    // start: angle in radians to start.
    // [end]: optional. If included text align is ignored and the text is
    //         scaled to fit between start and end;
    // [forward]: optional default true. if true text direction is forwards, if false
direction is backward
    var circleText = function (ctx, text, x, y, radius, start, end, forward) {
        var i, textWidth, pA, pAS, a, aw, wScale, aligned, dir, fontSize;
        if(text.trim() === "" || ctx.globalAlpha === 0){ // dont render empty string or
transparent
            return;
        }
        if(isNaN(x) || isNaN(y) || isNaN(radius) || isNaN(start) || (end !== undefined && end
!== null && isNaN(end))){ //
            throw TypeError("circle text arguments requires a number for x,y, radius, start,
and end.")
        }
        aligned = ctx.textAlign;          // save the current textAlign so that it can be
restored at end
        dir = forward ? 1 : forward === false ? -1 : 1; // set dir if not true or false set
forward as true
        pAS = 1 / radius;                // get the angular size of a pixel in radians
        textWidth = ctx.measureText(text).width; // get the width of all the text
        if (end !== undefined && end !== null) { // if end is supplied then fit text between
start and end
            pA = ((end - start) / textWidth) * dir;

```

```

        wScale = (pA / pAS) * dir;
    } else {
        // if no end is supplied correct start and end for alignment
        // if forward is not given then swap top of circle text to read the correct
direction
        if(forward === null || forward === undefined){
            if(((start % (Math.PI * 2)) + Math.PI * 2) % (Math.PI * 2) > Math.PI){
                dir = -1;
            }
        }
        pA = -pAS * dir ;
        wScale = -1 * dir;
        switch (aligned) {
            case "center": // if centered move around half width
                start -= (pA * textWidth) / 2;
                end = start + pA * textWidth;
                break;
            case "right": // intentionally falls through to case "end"
            case "end":
                end = start;
                start -= pA * textWidth;
                break;
            case "left": // intentionally falls through to case "start"
            case "start":
                end = start + pA * textWidth;
        }
    }

    ctx.textAlign = "center"; // align for rendering
    a = start; // set the start angle
    for (var i = 0; i < text.length; i += 1) { // for each character
        aw = ctx.measureText(text[i]).width * pA; // get the angular width of the text
        var xDx = Math.cos(a + aw / 2); // get the xAxes vector from the center
x,y out
        var xDy = Math.sin(a + aw / 2);
        if(multiplyCurrentTransform){ // transform multiplying current transform
            ctx.save();
            if (xDy < 0) { // is the text upside down. If it is flip it
                ctx.transform(-xDy * wScale, xDx * wScale, -xDx, -xDy, xDx * radius + x,
xDy * radius + y);
            } else {
                ctx.transform(-xDy * wScale, xDx * wScale, xDx, xDy, xDx * radius + x, xDy
* radius + y);
            }
        }else{
            if (xDy < 0) { // is the text upside down. If it is flip it
                ctx.setTransform(-xDy * wScale, xDx * wScale, -xDx, -xDy, xDx * radius +
x, xDy * radius + y);
            } else {
                ctx.setTransform(-xDy * wScale, xDx * wScale, xDx, xDy, xDx * radius + x,
xDy * radius + y);
            }
        }
        if(renderType === FILL){
            ctx.fillText(text[i], 0, 0); // render the character
        }else{
            ctx.strokeText(text[i], 0, 0); // render the character
        }
        if(multiplyCurrentTransform){ // restore current transform
            ctx.restore();
        }
        a += aw; // step to the next angle
    }
}

```

```

    }
    // all done clean up.
    if(!multiplyCurrentTransform){
        ctx.setTransform(1, 0, 0, 1, 0, 0); // restore the transform
    }
    ctx.textAlign = aligned; // restore the text alignment
}
// define fill text
var fillCircleText = function(text, x, y, radius, start, end, forward){
    renderType = FILL;
    circleText(this, text, x, y, radius, start, end, forward);
}
// define stroke text
var strokeCircleText = function(text, x, y, radius, start, end, forward){
    renderType = STROKE;
    circleText(this, text, x, y, radius, start, end, forward);
}
// define measure text
var measureCircleTextExt = function(text, radius){
    return measure(this, text, radius);
}
// set the prototypes
CanvasRenderingContext2D.prototype.fillCircleText = fillCircleText;
CanvasRenderingContext2D.prototype.strokeCircleText = strokeCircleText;
CanvasRenderingContext2D.prototype.measureCircleText = measureCircleTextExt;
})();

```

CanvasRenderingContext2D prototype<sup>3</sup>。 fillCircleText strokeCircleTextmeasureCircleText

## ***CanvasRenderingContext2D.fillCircleTexttext xyradiusstart[end[forward]];***

## ***CanvasRenderingContext2D.strokeCircleText textxyradiusstart[end[forward]];***

- **textString**。
- **x y**。
- **radius**
- **start**。
- **[]**。 ctx.textAlignstartend。
- **[forward]**'true'。 'false'。

textBaseline。 ctx.TextBaseline。

TypeErrorNaN。

textctx.globalAlpha = 0。

# CanvasRenderingContext2D.measureCircleText textradius;

```
(function(){
    const FILL = 0;          // const to indicate filltext render
    const STROKE = 1;
    var renderType = FILL; // used internal to set fill or stroke text
    const multiplyCurrentTransform = true; // if true Use current transform when rendering
                                         // if false use absolute coordinates which is a
little quicker
                                         // after render the currentTransform is restored to
default transform

    // measure circle text
    // ctx: canvas context
    // text: string of text to measure
    // r: radius in pixels
    //
    // returns the size metrics of the text
    //
    // width: Pixel width of text
    // angularWidth : angular width of text in radians
    // pixelAngularSize : angular width of a pixel in radians
    var measure = function(ctx, text, radius){
        var textWidth = ctx.measureText(text).width; // get the width of all the text
        return {
            width           : textWidth,
            angularWidth    : (1 / radius) * textWidth,
            pixelAngularSize : 1 / radius
        };
    }

    // displays text along a circle
    // ctx: canvas context
    // text: string of text to measure
    // x,y: position of circle center
    // r: radius of circle in pixels
    // start: angle in radians to start.
    // [end]: optional. If included text align is ignored and the text is
    //         scaled to fit between start and end;
    // [forward]: optional default true. if true text direction is forwards, if false
direction is backward
    var circleText = function (ctx, text, x, y, radius, start, end, forward) {
        var i, textWidth, pA, pAS, a, aw, wScale, aligned, dir, fontSize;
        if(text.trim() === "" || ctx.globalAlpha === 0){ // dont render empty string or
transparent
            return;
        }
        if(isNaN(x) || isNaN(y) || isNaN(radius) || isNaN(start) || (end !== undefined && end
!== null && isNaN(end))){ //
            throw TypeError("circle text arguments requires a number for x,y, radius, start,
and end.")
        }
        aligned = ctx.textAlign;          // save the current textAlign so that it can be
restored at end
```

```

    dir = forward ? 1 : forward === false ? -1 : 1; // set dir if not true or false set
forward as true
    pAS = 1 / radius; // get the angular size of a pixel in radians
    textWidth = ctx.measureText(text).width; // get the width of all the text
    if (end !== undefined && end !== null) { // if end is supplied then fit text between
start and end
        pA = ((end - start) / textWidth) * dir;
        wScale = (pA / pAS) * dir;
    } else { // if no end is supplied correct start and end for alignment
// if forward is not given then swap top of circle text to read the correct
direction
        if(forward === null || forward === undefined){
            if(((start % (Math.PI * 2)) + Math.PI * 2) % (Math.PI * 2) > Math.PI){
                dir = -1;
            }
        }
        pA = -pAS * dir ;
        wScale = -1 * dir;
        switch (aligned) {
        case "center": // if centered move around half width
            start -= (pA * textWidth) / 2;
            end = start + pA * textWidth;
            break;
        case "right": // intentionally falls through to case "end"
        case "end":
            end = start;
            start -= pA * textWidth;
            break;
        case "left": // intentionally falls through to case "start"
        case "start":
            end = start + pA * textWidth;
        }
    }

    ctx.textAlign = "center"; // align for rendering
    a = start; // set the start angle
    for (var i = 0; i < text.length; i += 1) { // for each character
        aw = ctx.measureText(text[i]).width * pA; // get the angular width of the text
        var xDx = Math.cos(a + aw / 2); // get the xAxes vector from the center
x,y out
        var xDy = Math.sin(a + aw / 2);
        if(multiplyCurrentTransform){ // transform multiplying current transform
            ctx.save();
            if (xDy < 0) { // is the text upside down. If it is flip it
                ctx.transform(-xDy * wScale, xDx * wScale, -xDx, -xDy, xDx * radius + x,
xDy * radius + y);
            } else {
                ctx.transform(-xDy * wScale, xDx * wScale, xDx, xDy, xDx * radius + x, xDy
* radius + y);
            }
        }else{
            if (xDy < 0) { // is the text upside down. If it is flip it
                ctx.setTransform(-xDy * wScale, xDx * wScale, -xDx, -xDy, xDx * radius +
x, xDy * radius + y);
            } else {
                ctx.setTransform(-xDy * wScale, xDx * wScale, xDx, xDy, xDx * radius + x,
xDy * radius + y);
            }
        }
        if(renderType === FILL){
            ctx.fillText(text[i], 0, 0); // render the character

```



```

    }else{
        ctx.strokeText(text[i], 0, 0); // render the character
    }
    if(multiplyCurrentTransform){ // restore current transform
        ctx.restore();
    }
    a += aw; // step to the next angle
}
// all done clean up.
if(!multiplyCurrentTransform){
    ctx.setTransform(1, 0, 0, 1, 0, 0); // restore the transform
}
ctx.textAlign = aligned; // restore the text alignment
}
// define fill text
var fillCircleText = function(text, x, y, radius, start, end, forward){
    renderType = FILL;
    circleText(this, text, x, y, radius, start, end, forward);
}
// define stroke text
var strokeCircleText = function(text, x, y, radius, start, end, forward){
    renderType = STROKE;
    circleText(this, text, x, y, radius, start, end, forward);
}
// define measure text
var measureCircleTextExt = function(text, radius){
    return measure(this, text, radius);
}
// set the prototypes
CanvasRenderingContext2D.prototype.fillCircleText = fillCircleText;
CanvasRenderingContext2D.prototype.strokeCircleText = strokeCircleText;
CanvasRenderingContext2D.prototype.measureCircleText = measureCircleTextExt;
})();

```

## Object

```

(function(){
    const FILL = 0; // const to indicate filltext render
    const STROKE = 1;
    var renderType = FILL; // used internal to set fill or stroke text
    const multiplyCurrentTransform = true; // if true Use current transform when rendering
    // if false use absolute coordinates which is a
    little quicker
    // after render the currentTransform is restored to
    default transform

    // measure circle text
    // ctx: canvas context
    // text: string of text to measure
    // r: radius in pixels
    //
    // returns the size metrics of the text
    //
    // width: Pixel width of text
    // angularWidth : angular width of text in radians
    // pixelAngularSize : angular width of a pixel in radians
    var measure = function(ctx, text, radius){
        var textWidth = ctx.measureText(text).width; // get the width of all the text
    }
})();

```

```

    return {
        width           : textWidth,
        angularWidth     : (1 / radius) * textWidth,
        pixelAngularSize : 1 / radius
    };
}

// displays text along a circle
// ctx: canvas context
// text: string of text to measure
// x,y: position of circle center
// r: radius of circle in pixels
// start: angle in radians to start.
// [end]: optional. If included text align is ignored and the text is
//        scaled to fit between start and end;
// [forward]: optional default true. if true text direction is forwards, if false
direction is backward
var circleText = function (ctx, text, x, y, radius, start, end, forward) {
    var i, textWidth, pA, pAS, a, aw, wScale, aligned, dir, fontSize;
    if(text.trim() === "" || ctx.globalAlpha === 0){ // dont render empty string or
transparent
        return;
    }
    if(isNaN(x) || isNaN(y) || isNaN(radius) || isNaN(start) || (end !== undefined && end
!== null && isNaN(end))){ //
        throw TypeError("circle text arguments requires a number for x,y, radius, start,
and end.")
    }
    aligned = ctx.textAlign;          // save the current textAlign so that it can be
restored at end
    dir = forward ? 1 : forward === false ? -1 : 1; // set dir if not true or false set
forward as true
    pAS = 1 / radius;                // get the angular size of a pixel in radians
    textWidth = ctx.measureText(text).width; // get the width of all the text
    if (end !== undefined && end !== null) { // if end is supplied then fit text between
start and end
        pA = ((end - start) / textWidth) * dir;
        wScale = (pA / pAS) * dir;
    } else {                          // if no end is supplied correct start and end for alignment
// if forward is not given then swap top of circle text to read the correct
direction
        if(forward === null || forward === undefined){
            if(((start % (Math.PI * 2)) + Math.PI * 2) % (Math.PI * 2) > Math.PI){
                dir = -1;
            }
        }
        pA = -pAS * dir ;
        wScale = -1 * dir;
        switch (aligned) {
            case "center":          // if centered move around half width
                start -= (pA * textWidth) / 2;
                end = start + pA * textWidth;
                break;
            case "right":// intentionally falls through to case "end"
            case "end":
                end = start;
                start -= pA * textWidth;
                break;
            case "left": // intentionally falls through to case "start"
            case "start":
                end = start + pA * textWidth;

```

```

    }
}

ctx.textAlign = "center"; // align for rendering
a = start; // set the start angle
for (var i = 0; i < text.length; i += 1) { // for each character
    aw = ctx.measureText(text[i]).width * pA; // get the angular width of the text
    var xDx = Math.cos(a + aw / 2); // get the xAxes vector from the center
x,y out
    var xDy = Math.sin(a + aw / 2);
    if(multiplyCurrentTransform){ // transform multiplying current transform
        ctx.save();
        if (xDy < 0) { // is the text upside down. If it is flip it
            ctx.transform(-xDy * wScale, xDx * wScale, -xDx, -xDy, xDx * radius + x,
xDy * radius + y);
        } else {
            ctx.transform(-xDy * wScale, xDx * wScale, xDx, xDy, xDx * radius + x, xDy
* radius + y);
        }
    }else{
        if (xDy < 0) { // is the text upside down. If it is flip it
            ctx.setTransform(-xDy * wScale, xDx * wScale, -xDx, -xDy, xDx * radius +
x, xDy * radius + y);
        } else {
            ctx.setTransform(-xDy * wScale, xDx * wScale, xDx, xDy, xDx * radius + x,
xDy * radius + y);
        }
    }
    if(renderType === FILL){
        ctx.fillText(text[i], 0, 0); // render the character
    }else{
        ctx.strokeText(text[i], 0, 0); // render the character
    }
    if(multiplyCurrentTransform){ // restore current transform
        ctx.restore();
    }
    a += aw; // step to the next angle
}
// all done clean up.
if(!multiplyCurrentTransform){
    ctx.setTransform(1, 0, 0, 1, 0, 0); // restore the transform
}
ctx.textAlign = aligned; // restore the text alignment
}
// define fill text
var fillCircleText = function(text, x, y, radius, start, end, forward){
    renderType = FILL;
    circleText(this, text, x, y, radius, start, end, forward);
}
// define stroke text
var strokeCircleText = function(text, x, y, radius, start, end, forward){
    renderType = STROKE;
    circleText(this, text, x, y, radius, start, end, forward);
}
// define measure text
var measureCircleTextExt = function(text, radius){
    return measure(this, text, radius);
}
// set the prototypes
CanvasRenderingContext2D.prototype.fillCircleText = fillCircleText;
CanvasRenderingContext2D.prototype.strokeCircleText = strokeCircleText;

```

```
CanvasRenderingContext2D.prototype.measureCircleText = measureCircleTextExt;
})();
```

```
(function(){
  const FILL = 0;          // const to indicate filltext render
  const STROKE = 1;
  var renderType = FILL; // used internal to set fill or stroke text
  const multiplyCurrentTransform = true; // if true Use current transform when rendering
                                     // if false use absolute coordinates which is a
little quicker
                                     // after render the currentTransform is restored to
default transform

  // measure circle text
  // ctx: canvas context
  // text: string of text to measure
  // r: radius in pixels
  //
  // returns the size metrics of the text
  //
  // width: Pixel width of text
  // angularWidth : angular width of text in radians
  // pixelAngularSize : angular width of a pixel in radians
  var measure = function(ctx, text, radius){
    var textWidth = ctx.measureText(text).width; // get the width of all the text
    return {
      width           : textWidth,
      angularWidth    : (1 / radius) * textWidth,
      pixelAngularSize : 1 / radius
    };
  }

  // displays text along a circle
  // ctx: canvas context
  // text: string of text to measure
  // x,y: position of circle center
  // r: radius of circle in pixels
  // start: angle in radians to start.
  // [end]: optional. If included text align is ignored and the text is
  //        scaled to fit between start and end;
  // [forward]: optional default true. if true text direction is forwards, if false
direction is backward
  var circleText = function (ctx, text, x, y, radius, start, end, forward) {
    var i, textWidth, pA, pAS, a, aw, wScale, aligned, dir, fontSize;
    if(text.trim() === "" || ctx.globalAlpha === 0){ // dont render empty string or
transparent
      return;
    }
    if(isNaN(x) || isNaN(y) || isNaN(radius) || isNaN(start) || (end !== undefined && end
!== null && isNaN(end))){ //
      throw TypeError("circle text arguments requires a number for x,y, radius, start,
and end.")
    }
    aligned = ctx.textAlign;          // save the current textAlign so that it can be
restored at end
    dir = forward ? 1 : forward === false ? -1 : 1; // set dir if not true or false set
forward as true
  }
```

```

    pAS = 1 / radius; // get the angular size of a pixel in radians
    textWidth = ctx.measureText(text).width; // get the width of all the text
    if (end !== undefined && end !== null) { // if end is supplied then fit text between
start and end
        pA = ((end - start) / textWidth) * dir;
        wScale = (pA / pAS) * dir;
    } else { // if no end is supplied correct start and end for alignment
        // if forward is not given then swap top of circle text to read the correct
direction
        if(forward === null || forward === undefined){
            if(((start % (Math.PI * 2)) + Math.PI * 2) % (Math.PI * 2) > Math.PI){
                dir = -1;
            }
        }
        pA = -pAS * dir ;
        wScale = -1 * dir;
        switch (aligned) {
        case "center": // if centered move around half width
            start -= (pA * textWidth) / 2;
            end = start + pA * textWidth;
            break;
        case "right": // intentionally falls through to case "end"
        case "end":
            end = start;
            start -= pA * textWidth;
            break;
        case "left": // intentionally falls through to case "start"
        case "start":
            end = start + pA * textWidth;
        }
    }

    ctx.textAlign = "center"; // align for rendering
    a = start; // set the start angle
    for (var i = 0; i < text.length; i += 1) { // for each character
        aw = ctx.measureText(text[i]).width * pA; // get the angular width of the text
        var xDx = Math.cos(a + aw / 2); // get the xAxes vector from the center
x,y out
        var xDy = Math.sin(a + aw / 2);
        if(multiplyCurrentTransform){ // transform multiplying current transform
            ctx.save();
            if (xDy < 0) { // is the text upside down. If it is flip it
                ctx.transform(-xDy * wScale, xDx * wScale, -xDx, -xDy, xDx * radius + x,
xDy * radius + y);
            } else {
                ctx.transform(-xDy * wScale, xDx * wScale, xDx, xDy, xDx * radius + x, xDy
* radius + y);
            }
        }else{
            if (xDy < 0) { // is the text upside down. If it is flip it
                ctx.setTransform(-xDy * wScale, xDx * wScale, -xDx, -xDy, xDx * radius +
x, xDy * radius + y);
            } else {
                ctx.setTransform(-xDy * wScale, xDx * wScale, xDx, xDy, xDx * radius + x,
xDy * radius + y);
            }
        }
        if(renderType === FILL){
            ctx.fillText(text[i], 0, 0); // render the character
        }else{
            ctx.strokeText(text[i], 0, 0); // render the character
        }
    }
}

```

```

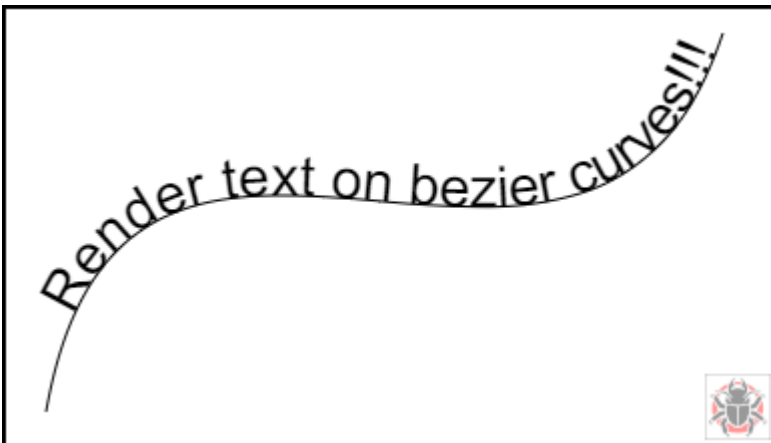
    }
    if(multiplyCurrentTransform){ // restore current transform
        ctx.restore();
    }
    a += aw; // step to the next angle
}
// all done clean up.
if(!multiplyCurrentTransform){
    ctx.setTransform(1, 0, 0, 1, 0, 0); // restore the transform
}
ctx.textAlign = aligned; // restore the text alignment
}
// define fill text
var fillCircleText = function(text, x, y, radius, start, end, forward){
    renderType = FILL;
    circleText(this, text, x, y, radius, start, end, forward);
}
// define stroke text
var strokeCircleText = function(text, x, y, radius, start, end, forward){
    renderType = STROKE;
    circleText(this, text, x, y, radius, start, end, forward);
}
// define measure text
var measureCircleTextExt = function(text, radius){
    return measure(this, text, radius);
}
// set the prototypes
CanvasRenderingContext2D.prototype.fillCircleText = fillCircleText;
CanvasRenderingContext2D.prototype.strokeCircleText = strokeCircleText;
CanvasRenderingContext2D.prototype.measureCircleText = measureCircleTextExt;
})();

```

。 |“H”。 。

const multiplyCurrentTransform = true;。 false。 。

multiplyCurrentTransform = true before fillCircleText strokeCircleText。 2D  
multiplyCurrentTransform = false



**textOnCurveX1Y1X2Y2X3Y3X4Y4**

。

- text
- offset  $\geq 0$
- x1,y1 - x3,y3
- x1,y1 - x4,y4

```
textOnCurve("Hello world!", 50, 100, 100, 200, 200, 300, 100); // draws text on quadratic curve
// 50 pixels from start of curve
```

```
textOnCurve("Hello world!", 50, 100, 100, 200, 200, 300, 100, 400, 200);
// draws text on cubic curve
// 50 pixels from start of curve
```

```
textOnCurve("Hello world!", 50, 100, 100, 200, 200, 300, 100); // draws text on quadratic curve
// 50 pixels from start of curve
```

```
textOnCurve("Hello world!", 50, 100, 100, 200, 200, 300, 100, 400, 200);
// draws text on cubic curve
// 50 pixels from start of curve
```

◦

```
textOnCurve("Hello world!", 50, 100, 100, 200, 200, 300, 100); // draws text on quadratic curve
// 50 pixels from start of curve
```

```
textOnCurve("Hello world!", 50, 100, 100, 200, 200, 300, 100, 400, 200);
// draws text on cubic curve
// 50 pixels from start of curve
```

◦ justifiedTextCanvasRenderingContext2D **A**◦

---

◦

## Justified text examples.

### < 'left' aligned

This text is aligned from the left of the canvas.  
This text is near the max spacing size  
This text is way too short.  
This text is too long for the space provided and will overflow#

### < 'end' aligned from x plus the width ----->

This text is aligned using 'end' and starts at x + width  
This text is near the max spacing size  
This text is way too short.  
#This text is too long for the space provided and will overflow

### 'center' aligned

This is aligned with 'center' and is placed from the center  
This text is near the max spacing size  
This text is way too short.  
This text is just too long for the space provided and will overflow

```
(function(){
  const FILL = 0;          // const to indicate filltext render
  const STROKE = 1;
  const MEASURE = 2;
  var renderType = FILL; // used internal to set fill or stroke text

  var maxSpaceSize = 3; // Multiplier for max space size. If greater then no justification
  applied
  var minSpaceSize = 0.5; // Multiplier for minimum space size
  var renderTextJustified = function(ctx,text,x,y,width){
    var words, wordsWidth, count, spaces, spaceWidth, adjSpace, renderer, i, textAlign,
    useSize, totalWidth;
    textAlign = ctx.textAlign; // get current align settings
    ctx.textAlign = "left";
    wordsWidth = 0;
    words = text.split(" ").map(word => {
      var w = ctx.measureText(word).width;
      wordsWidth += w;
      return {
        width : w,
        word : word,
      };
    });
    // count = num words, spaces = number spaces, spaceWidth normal space size
    // adjSpace new space size >= min size. useSize Resulting space size used to render
    count = words.length;
    spaces = count - 1;
    spaceWidth = ctx.measureText(" ").width;
    adjSpace = Math.max(spaceWidth * minSpaceSize, (width - wordsWidth) / spaces);
    useSize = adjSpace > spaceWidth * maxSpaceSize ? spaceWidth : adjSpace;
    totalWidth = wordsWidth + useSize * spaces
    if(renderType === MEASURE){ // if measuring return size
```



```

        ctx.textAlign = textAlign;
        return totalWidth;
    }
    renderer = renderType === FILL ? ctx.fillText.bind(ctx) : ctx.strokeText.bind(ctx); //
fill or stroke
    switch(textAlign){
        case "right":
            x -= totalWidth;
            break;
        case "end":
            x += width - totalWidth;
            break;
        case "center": // intentional fall through to default
            x -= totalWidth / 2;
        default:
    }
    if(useSize === spaceWidth){ // if space size unchanged
        renderer(text,x,y);
    } else {
        for(i = 0; i < count; i += 1){
            renderer(words[i].word,x,y);
            x += words[i].width;
            x += useSize;
        }
    }
    ctx.textAlign = textAlign;
}
// Parse vet and set settings object.
var justifiedTextSettings = function(settings){
    var min,max;
    var vetNumber = (num, defaultNum) => {
        num = num !== null && num !== null && !isNaN(num) ? num : defaultNum;
        if(num < 0){
            num = defaultNum;
        }
        return num;
    }
    if(settings === undefined || settings === null){
        return;
    }
    max = vetNumber(settings.maxSpaceSize, maxSpaceSize);
    min = vetNumber(settings.minSpaceSize, minSpaceSize);
    if(min > max){
        return;
    }
    minSpaceSize = min;
    maxSpaceSize = max;
}
// define fill text
var fillJustifyText = function(text, x, y, width, settings){
    justifiedTextSettings(settings);
    renderType = FILL;
    renderTextJustified(this, text, x, y, width);
}
// define stroke text
var strokeJustifyText = function(text, x, y, width, settings){
    justifiedTextSettings(settings);
    renderType = STROKE;
    renderTextJustified(this, text, x, y, width);
}
// define measure text

```

```

var measureJustifiedText = function(text, width, settings){
    justifiedTextSettings(settings);
    renderType = MEASURE;
    return renderTextJustified(this, text, 0, 0, width);
}
// code point A
// set the prototypes
CanvasRenderingContext2D.prototype.fillJustifyText = fillJustifyText;
CanvasRenderingContext2D.prototype.strokeJustifyText = strokeJustifyText;
CanvasRenderingContext2D.prototype.measureJustifiedText = measureJustifiedText;
// code point B

// optional code if you do not wish to extend the CanvasRenderingContext2D prototype
/* Uncomment from here to the closing comment
window.justifiedText = {
    fill : function(ctx, text, x, y, width, settings){
        justifiedTextSettings(settings);
        renderType = FILL;
        renderTextJustified(ctx, text, x, y, width);
    },
    stroke : function(ctx, text, x, y, width, settings){
        justifiedTextSettings(settings);
        renderType = STROKE;
        renderTextJustified(ctx, text, x, y, width);
    },
    measure : function(ctx, text, width, settings){
        justifiedTextSettings(settings);
        renderType = MEASURE;
        return renderTextJustified(ctx, text, 0, 0, width);
    }
}
to here*/
})();

```

**A**CanvasRenderingContext2D// code point A// code point B/\* Uncomment from here to the closing comment/\* Uncomment from here to the closing comment

CanvasRenderingContext2D<sup>2D</sup>.

- `ctx.fillJustifyTexttextxywidth[settings];`
- `ctx.strokeJustifyTexttextxywidth[settings];`
- `ctx.measureJustifiedTexttextwidth[settings];`

◦ `measureJustifiedText`◦ `width`◦

`[]`◦

- **text**◦
- **xy**◦
- **width**◦ /◦ `maxSpaceSize = 6`◦ `minSpaceSize = 0.5`
- ◦ ◦

settings◦

**minmax[space]◦** maxSpaceSize = 63 \* ctx.measureText("").width◦ minSpaceSize = 0.5 0.5\*  
ctx.measureText(" ").widthminSpaceSize \* ctx.measureText(" ").width◦

**minmax◦** ◦ minSpaceSizemaxSpaceSize**min max◦**

```
(function(){
  const FILL = 0;          // const to indicate filltext render
  const STROKE = 1;
  const MEASURE = 2;
  var renderType = FILL; // used internal to set fill or stroke text

  var maxSpaceSize = 3; // Multiplier for max space size. If greater then no justification
  applied
  var minSpaceSize = 0.5; // Multiplier for minimum space size
  var renderTextJustified = function(ctx,text,x,y,width){
    var words, wordsWidth, count, spaces, spaceWidth, adjSpace, renderer, i, textAlign,
    useSize, totalWidth;
    textAlign = ctx.textAlign; // get current align settings
    ctx.textAlign = "left";
    wordsWidth = 0;
    words = text.split(" ").map(word => {
      var w = ctx.measureText(word).width;
      wordsWidth += w;
      return {
        width : w,
        word : word,
      };
    });
    // count = num words, spaces = number spaces, spaceWidth normal space size
    // adjSpace new space size >= min size. useSize Resulting space size used to render
    count = words.length;
    spaces = count - 1;
    spaceWidth = ctx.measureText(" ").width;
    adjSpace = Math.max(spaceWidth * minSpaceSize, (width - wordsWidth) / spaces);
    useSize = adjSpace > spaceWidth * maxSpaceSize ? spaceWidth : adjSpace;
    totalWidth = wordsWidth + useSize * spaces
    if(renderType === MEASURE){ // if measuring return size
      ctx.textAlign = textAlign;
      return totalWidth;
    }
    renderer = renderType === FILL ? ctx.fillText.bind(ctx) : ctx.strokeText.bind(ctx); //
    fill or stroke
    switch(textAlign){
      case "right":
        x -= totalWidth;
        break;
      case "end":
        x += width - totalWidth;
        break;
      case "center": // intentional fall through to default
        x -= totalWidth / 2;
      default:
    }
    if(useSize === spaceWidth){ // if space size unchanged
      renderer(text,x,y);
    } else {
      for(i = 0; i < count; i += 1){
```

```

        renderer(words[i].word,x,y);
        x += words[i].width;
        x += useSize;
    }
}
ctx.textAlign = textAlign;
}
// Parse vet and set settings object.
var justifiedTextSettings = function(settings){
    var min,max;
    var vetNumber = (num, defaultNum) => {
        num = num !== null && num !== null && !isNaN(num) ? num : defaultNum;
        if(num < 0){
            num = defaultNum;
        }
        return num;
    }
    if(settings === undefined || settings === null){
        return;
    }
    max = vetNumber(settings.maxSpaceSize, maxSpaceSize);
    min = vetNumber(settings.minSpaceSize, minSpaceSize);
    if(min > max){
        return;
    }
    minSpaceSize = min;
    maxSpaceSize = max;
}
// define fill text
var fillJustifyText = function(text, x, y, width, settings){
    justifiedTextSettings(settings);
    renderType = FILL;
    renderTextJustified(this, text, x, y, width);
}
// define stroke text
var strokeJustifyText = function(text, x, y, width, settings){
    justifiedTextSettings(settings);
    renderType = STROKE;
    renderTextJustified(this, text, x, y, width);
}
// define measure text
var measureJustifiedText = function(text, width, settings){
    justifiedTextSettings(settings);
    renderType = MEASURE;
    return renderTextJustified(this, text, 0, 0, width);
}
// code point A
// set the prototypes
CanvasRenderingContext2D.prototype.fillJustifyText = fillJustifyText;
CanvasRenderingContext2D.prototype.strokeJustifyText = strokeJustifyText;
CanvasRenderingContext2D.prototype.measureJustifiedText = measureJustifiedText;
// code point B

// optional code if you do not wish to extend the CanvasRenderingContext2D prototype
/* Uncomment from here to the closing comment
window.justifiedText = {
    fill : function(ctx, text, x, y, width, settings){
        justifiedTextSettings(settings);
        renderType = FILL;
        renderTextJustified(ctx, text, x, y, width);
    },

```

```

stroke : function(ctx, text, x, y, width, settings){
  justifiedTextSettings(settings);
  renderType = STROKE;
  renderTextJustified(ctx, text, x, y, width);
},
measure : function(ctx, text, width, settings){
  justifiedTextSettings(settings);
  renderType = MEASURE;
  return renderTextJustified(ctx, text, 0, 0, width);
}
}
to here*/
})();

```

2D<sub>textAlign</sub>° 'left'right'center'start'end'xx + width

2D°

```

(function(){
  const FILL = 0;          // const to indicate filltext render
  const STROKE = 1;
  const MEASURE = 2;
  var renderType = FILL; // used internal to set fill or stroke text

  var maxSpaceSize = 3; // Multiplier for max space size. If greater then no justification
  applied
  var minSpaceSize = 0.5; // Multiplier for minimum space size
  var renderTextJustified = function(ctx,text,x,y,width){
    var words, wordsWidth, count, spaces, spaceWidth, adjSpace, renderer, i, textAlign,
    useSize, totalWidth;
    textAlign = ctx.textAlign; // get current align settings
    ctx.textAlign = "left";
    wordsWidth = 0;
    words = text.split(" ").map(word => {
      var w = ctx.measureText(word).width;
      wordsWidth += w;
      return {
        width : w,
        word : word,
      };
    });
    // count = num words, spaces = number spaces, spaceWidth normal space size
    // adjSpace new space size >= min size. useSize Resulting space size used to render
    count = words.length;
    spaces = count - 1;
    spaceWidth = ctx.measureText(" ").width;
    adjSpace = Math.max(spaceWidth * minSpaceSize, (width - wordsWidth) / spaces);
    useSize = adjSpace > spaceWidth * maxSpaceSize ? spaceWidth : adjSpace;
    totalWidth = wordsWidth + useSize * spaces
    if(renderType === MEASURE){ // if measuring return size
      ctx.textAlign = textAlign;
      return totalWidth;
    }
    renderer = renderType === FILL ? ctx.fillText.bind(ctx) : ctx.strokeText.bind(ctx); //
    fill or stroke
    switch(textAlign){
      case "right":
        x -= totalWidth;

```

```

        break;
    case "end":
        x += width - totalWidth;
        break;
    case "center": // intentional fall through to default
        x -= totalWidth / 2;
    default:
    }
    if(useSize === spaceWidth){ // if space size unchanged
        renderer(text,x,y);
    } else {
        for(i = 0; i < count; i += 1){
            renderer(words[i].word,x,y);
            x += words[i].width;
            x += useSize;
        }
    }
    ctx.textAlign = textAlign;
}
// Parse vet and set settings object.
var justifiedTextSettings = function(settings){
    var min,max;
    var vetNumber = (num, defaultNum) => {
        num = num !== null && num !== null && !isNaN(num) ? num : defaultNum;
        if(num < 0){
            num = defaultNum;
        }
        return num;
    }
    if(settings === undefined || settings === null){
        return;
    }
    max = vetNumber(settings.maxSpaceSize, maxSpaceSize);
    min = vetNumber(settings.minSpaceSize, minSpaceSize);
    if(min > max){
        return;
    }
    minSpaceSize = min;
    maxSpaceSize = max;
}
// define fill text
var fillJustifyText = function(text, x, y, width, settings){
    justifiedTextSettings(settings);
    renderType = FILL;
    renderTextJustified(this, text, x, y, width);
}
// define stroke text
var strokeJustifyText = function(text, x, y, width, settings){
    justifiedTextSettings(settings);
    renderType = STROKE;
    renderTextJustified(this, text, x, y, width);
}
// define measure text
var measureJustifiedText = function(text, width, settings){
    justifiedTextSettings(settings);
    renderType = MEASURE;
    return renderTextJustified(this, text, 0, 0, width);
}
// code point A
// set the prototypes
CanvasRenderingContext2D.prototype.fillJustifyText = fillJustifyText;

```

```

CanvasRenderingContext2D.prototype.strokeJustifyText = strokeJustifyText;
CanvasRenderingContext2D.prototype.measureJustifiedText = measureJustifiedText;
// code point B

// optional code if you do not wish to extend the CanvasRenderingContext2D prototype
/* Uncomment from here to the closing comment
window.justifiedText = {
    fill : function(ctx, text, x, y, width, settings){
        justifiedTextSettings(settings);
        renderType = FILL;
        renderTextJustified(ctx, text, x, y, width);
    },
    stroke : function(ctx, text, x, y, width, settings){
        justifiedTextSettings(settings);
        renderType = STROKE;
        renderTextJustified(ctx, text, x, y, width);
    },
    measure : function(ctx, text, width, settings){
        justifiedTextSettings(settings);
        renderType = MEASURE;
        return renderTextJustified(ctx, text, 0, 0, width);
    }
}
to here*/
})();

```

◦

## ◦ Justified

### Justified paragraph examples.

Typesetting is the composition of text by means of arranging physical types or the digital equivalents. Stored letters and other symbols are retrieved and ordered according to a language's orthography for visual display. Typesetting requires the prior process of designing a font. One significant effect of typesetting was that authorship of works could be spotted more easily; making it difficult for copiers who have not gained permission.

Typesetting is the composition of text by means of arranging physical types or the digital equivalents. Stored letters and other symbols are retrieved and ordered according to a language's orthography for visual display. Typesetting requires the prior process of designing a font. One significant effect of typesetting was that authorship of works could be spotted more easily; making it difficult for copiers who have not gained permission.

**setting.compact = true false 1.21.5** ◦ ◦

```

// Requires justified text extensions
(function(){

```

```

// code point A
if(typeof CanvasRenderingContext2D.prototype.fillJustifyText !== "function"){
    throw new ReferenceError("Justified Paragraph extension missing required
CanvasRenderingContext2D justified text extension");
}
var maxSpaceSize = 3; // Multiplier for max space size. If greater then no justification
applied
var minSpaceSize = 0.5; // Multiplier for minimum space size
var compact = true; // if true then try and fit as many words as possible. If false then
try to get the spacing as close as possible to normal
var lineSpacing = 1.5; // space between lines
const noJustifySetting = { // This setting forces justified text off. Used to render last
line of paragraph.
    minSpaceSize : 1,
    maxSpaceSize : 1,
}

// Parse vet and set settings object.
var justifiedTextSettings = function(settings){
    var min, max;
    var vetNumber = (num, defaultNum) => {
        num = num !== null && num !== null && !isNaN(num) ? num : defaultNum;
        return num < 0 ? defaultNum : num;
    }
    if(settings === undefined || settings === null){ return; }
    compact = settings.compact === true ? true : settings.compact === false ? false :
compact;
    max = vetNumber(settings.maxSpaceSize, maxSpaceSize);
    min = vetNumber(settings.minSpaceSize, minSpaceSize);
    lineSpacing = vetNumber(settings.lineSpacing, lineSpacing);
    if(min > max){ return; }
    minSpaceSize = min;
    maxSpaceSize = max;
}
var getFontSize = function(font){ // get the font size.
    var numFind = /[0-9]+/;
    var number = numFind.exec(font)[0];
    if(isNaN(number)){
        throw new ReferenceError("justifiedPar Cant find font size");
    }
    return Number(number);
}
function justifiedPar(ctx, text, x, y, width, settings, stroke){
    var spaceWidth, minS, maxS, words, count, lines, lineWidth, lastLineWidth, lastSize,
i, renderer, fontSize, adjSpace, spaces, word, lineWords, lineFound;
    spaceWidth = ctx.measureText(" ").width;
    minS = spaceWidth * minSpaceSize;
    maxS = spaceWidth * maxSpaceSize;
    words = text.split(" ").map(word => { // measure all words.
        var w = ctx.measureText(word).width;
        return {
            width : w,
            word : word,
        };
    });
    // count = num words, spaces = number spaces, spaceWidth normal space size
    // adjSpace new space size >= min size. useSize Resulting space size used to render
    count = 0;
    lines = [];
    // create lines by shifting words from the words array until the spacing is optimal.
    If compact

```



```

        // true then will true and fit as many words as possible. Else it will try and get the
spacing as
        // close as possible to the normal spacing
        while(words.length > 0){
            lastLineWidth = 0;
            lastSize = -1;
            lineFound = false;
            // each line must have at least one word.
            word = words.shift();
            lineWidth = word.width;
            lineWords = [word.word];
            count = 0;
            while(lineWidth < width && words.length > 0){ // Add words to line
                word = words.shift();
                lineWidth += word.width;
                lineWords.push(word.word);
                count += 1;
                spaces = count - 1;
                adjSpace = (width - lineWidth) / spaces;
                if(minS > adjSpace){ // if spacing less than min remove last word and finish
line
                    lineFound = true;
                    words.unshift(word);
                    lineWords.pop();
                }else{
                    if(!compact){ // if compact mode
                        if(adjSpace < spaceWidth){ // if less than normal space width
                            if(lastSize === -1){
                                lastSize = adjSpace;
                            }
                            // check if with last word on if its closer to space width
                            if(Math.abs(spaceWidth - adjSpace) < Math.abs(spaceWidth -
lastSize)){
                                lineFound = true; // yes keep it
                            }else{
                                words.unshift(word); // no better fit if last word removes
                                lineWords.pop();
                                lineFound = true;
                            }
                        }
                    }
                    lastSize = adjSpace; // remember spacing
                }
                lines.push(lineWords.join(" ")); // and the line
            }
            // lines have been worked out get font size, render, and render all the lines. last
            // line may need to be rendered as normal so it is outside the loop.
            fontSize = getFontSize(ctx.font);
            renderer = stroke === true ? ctx.strokeJustifyText.bind(ctx) :
ctx.fillTextJustifyText.bind(ctx);
            for(i = 0; i < lines.length - 1; i++){
                renderer(lines[i], x, y, width, settings);
                y += lineSpacing * fontSize;
            }
            if(lines.length > 0){ // last line if left or start aligned for no justify
                if(ctx.textAlign === "left" || ctx.textAlign === "start"){
                    renderer(lines[lines.length - 1], x, y, width, noJustifySetting);
                    ctx.measureJustifiedText("", width, settings);
                }else{
                    renderer(lines[lines.length - 1], x, y, width);
                }
            }
        }
    }
}

```

```

    }
  }
  // return details about the paragraph.
  y += lineSpacing * fontSize;
  return {
    nextLine : y,
    fontSize : fontSize,
    lineHeight : lineSpacing * fontSize,
  };
}
// define fill
var fillParagraphText = function(text, x, y, width, settings){
  justifiedTextSettings(settings);
  settings = {
    minSpaceSize : minSpaceSize,
    maxSpaceSize : maxSpaceSize,
  };
  return justifiedPar(this, text, x, y, width, settings);
}
// define stroke
var strokeParagraphText = function(text, x, y, width, settings){
  justifiedTextSettings(settings);
  settings = {
    minSpaceSize : minSpaceSize,
    maxSpaceSize : maxSpaceSize,
  };
  return justifiedPar(this, text, x, y, width, settings, true);
}
CanvasRenderingContext2D.prototype.fillParaText = fillParagraphText;
CanvasRenderingContext2D.prototype.strokeParaText = strokeParagraphText;
})();

```

CanvasRenderingContext2D. **Justified**.

CanvasRenderingContext2D.prototype.fillJustifyText **ReferenceError**

```

// Requires justified text extensions
(function(){
  // code point A
  if(typeof CanvasRenderingContext2D.prototype.fillJustifyText !== "function"){
    throw new ReferenceError("Justified Paragraph extension missing required
CanvasRenderingContext2D justified text extension");
  }
  var maxSpaceSize = 3; // Multiplier for max space size. If greater then no justification
applied
  var minSpaceSize = 0.5; // Multiplier for minimum space size
  var compact = true; // if true then try and fit as many words as possible. If false then
try to get the spacing as close as possible to normal
  var lineSpacing = 1.5; // space between lines
  const noJustifySetting = { // This setting forces justified text off. Used to render last
line of paragraph.
    minSpaceSize : 1,
    maxSpaceSize : 1,
  }

  // Parse vet and set settings object.
  var justifiedTextSettings = function(settings){
    var min, max;

```

```

var vetNumber = (num, defaultNum) => {
  num = num !== null && num !== null && !isNaN(num) ? num : defaultNum;
  return num < 0 ? defaultNum : num;
}
if(settings === undefined || settings === null){ return; }
compact = settings.compact === true ? true : settings.compact === false ? false :
compact;
max = vetNumber(settings.maxSpaceSize, maxSpaceSize);
min = vetNumber(settings.minSpaceSize, minSpaceSize);
lineSpacing = vetNumber(settings.lineSpacing, lineSpacing);
if(min > max){ return; }
minSpaceSize = min;
maxSpaceSize = max;
}
var getFontSize = function(font){ // get the font size.
  var numFind = /[0-9]+/;
  var number = numFind.exec(font)[0];
  if(isNaN(number)){
    throw new ReferenceError("justifiedPar Cant find font size");
  }
  return Number(number);
}
function justifiedPar(ctx, text, x, y, width, settings, stroke){
  var spaceWidth, minS, maxS, words, count, lines, lineWidth, lastLineWidth, lastSize,
i, renderer, fontSize, adjSpace, spaces, word, lineWords, lineFound;
  spaceWidth = ctx.measureText(" ").width;
  minS = spaceWidth * minSpaceSize;
  maxS = spaceWidth * maxSpaceSize;
  words = text.split(" ").map(word => { // measure all words.
    var w = ctx.measureText(word).width;
    return {
      width : w,
      word : word,
    };
  });
  // count = num words, spaces = number spaces, spaceWidth normal space size
  // adjSpace new space size >= min size. useSize Resulting space size used to render
  count = 0;
  lines = [];
  // create lines by shifting words from the words array until the spacing is optimal.
If compact
  // true then will true and fit as many words as possible. Else it will try and get the
spacing as
  // close as possible to the normal spacing
  while(words.length > 0){
    lastLineWidth = 0;
    lastSize = -1;
    lineFound = false;
    // each line must have at least one word.
    word = words.shift();
    lineWidth = word.width;
    lineWords = [word.word];
    count = 0;
    while(lineWidth < width && words.length > 0){ // Add words to line
      word = words.shift();
      lineWidth += word.width;
      lineWords.push(word.word);
      count += 1;
      spaces = count - 1;
      adjSpace = (width - lineWidth) / spaces;
      if(minS > adjSpace){ // if spacing less than min remove last word and finish

```

```

line
    lineFound = true;
    words.unshift(word);
    lineWords.pop();
}else{
    if(!compact){ // if compact mode
        if(adjSpace < spaceWidth){ // if less than normal space width
            if(lastSize === -1){
                lastSize = adjSpace;
            }
            // check if with last word on if its closer to space width
            if(Math.abs(spaceWidth - adjSpace) < Math.abs(spaceWidth -
lastSize)){
                lineFound = true; // yes keep it
            }else{
                words.unshift(word); // no better fit if last word removes
                lineWords.pop();
                lineFound = true;
            }
        }
    }
    lastSize = adjSpace; // remember spacing
}
lines.push(lineWords.join(" ")); // and the line
}
// lines have been worked out get font size, render, and render all the lines. last
// line may need to be rendered as normal so it is outside the loop.
fontSize = getFontSize(ctx.font);
render = stroke === true ? ctx.strokeJustifyText.bind(ctx) :
ctx.fillTextJustifyText.bind(ctx);
for(i = 0; i < lines.length - 1; i++){
    render(lines[i], x, y, width, settings);
    y += lineSpacing * fontSize;
}
if(lines.length > 0){ // last line if left or start aligned for no justify
    if(ctx.textAlign === "left" || ctx.textAlign === "start"){
        render(lines[lines.length - 1], x, y, width, noJustifySetting);
        ctx.measureJustifiedText("", width, settings);
    }else{
        render(lines[lines.length - 1], x, y, width);
    }
}
// return details about the paragraph.
y += lineSpacing * fontSize;
return {
    nextLine : y,
    fontSize : fontSize,
    lineHeight : lineSpacing * fontSize,
};
}
// define fill
var fillParagraphText = function(text, x, y, width, settings){
    justifiedTextSettings(settings);
    settings = {
        minSpaceSize : minSpaceSize,
        maxSpaceSize : maxSpaceSize,
    };
    return justifiedPar(this, text, x, y, width, settings);
}
// define stroke

```

```

var strokeParagraphText = function(text, x, y, width, settings){
  justifiedTextSettings(settings);
  settings = {
    minSpaceSize : minSpaceSize,
    maxSpaceSize : maxSpaceSize,
  };
  return justifiedPar(this, text, x, y, width, settings,true);
}
CanvasRenderingContext2D.prototype.fillParaText = fillParagraphText;
CanvasRenderingContext2D.prototype.strokeParaText = strokeParagraphText;
})();

```

◦ []◦

settings◦

- **compact** true◦ true◦ false◦
- **lineSpacing** 1.5◦ 1.5

◦ ◦ compact true false Truthy◦

◦ ◦

- **nextLine**◦
- **fontSize**◦ 14px arial
- **lineHeight**

◦ ◦ ◦ ◦

**Justified**◦ ◦ justifiedTextSettings◦ **Justified**◦ // Code point A◦ ◦

```

// Requires justified text extensions
(function(){
  // code point A
  if(typeof CanvasRenderingContext2D.prototype.fillJustifyText !== "function"){
    throw new ReferenceError("Justified Paragraph extension missing required
CanvasRenderingContext2D justified text extension");
  }
  var maxSpaceSize = 3; // Multiplier for max space size. If greater then no justification
  applied
  var minSpaceSize = 0.5; // Multiplier for minimum space size
  var compact = true; // if true then try and fit as many words as possible. If false then
  try to get the spacing as close as possible to normal
  var lineSpacing = 1.5; // space between lines
  const noJustifySetting = { // This setting forces justified text off. Used to render last
  line of paragraph.
    minSpaceSize : 1,
    maxSpaceSize : 1,
  }

  // Parse vet and set settings object.
  var justifiedTextSettings = function(settings){
    var min, max;
    var vetNumber = (num, defaultNum) => {

```

```

        num = num !== null && num !== null && !isNaN(num) ? num : defaultNum;
        return num < 0 ? defaultNum : num;
    }
    if(settings === undefined || settings === null){ return; }
    compact = settings.compact === true ? true : settings.compact === false ? false :
compact;
    max = vetNumber(settings.maxSpaceSize, maxSpaceSize);
    min = vetNumber(settings.minSpaceSize, minSpaceSize);
    lineSpacing = vetNumber(settings.lineSpacing, lineSpacing);
    if(min > max){ return; }
    minSpaceSize = min;
    maxSpaceSize = max;
}
var getFontSize = function(font){ // get the font size.
    var numFind = /[0-9]+/;
    var number = numFind.exec(font)[0];
    if(isNaN(number)){
        throw new ReferenceError("justifiedPar Cant find font size");
    }
    return Number(number);
}
function justifiedPar(ctx, text, x, y, width, settings, stroke){
    var spaceWidth, minS, maxS, words, count, lines, lineWidth, lastLineWidth, lastSize,
i, renderer, fontSize, adjSpace, spaces, word, lineWords, lineFound;
    spaceWidth = ctx.measureText(" ").width;
    minS = spaceWidth * minSpaceSize;
    maxS = spaceWidth * maxSpaceSize;
    words = text.split(" ").map(word => { // measure all words.
        var w = ctx.measureText(word).width;
        return {
            width : w,
            word : word,
        };
    });
    // count = num words, spaces = number spaces, spaceWidth normal space size
    // adjSpace new space size >= min size. useSize Resulting space size used to render
    count = 0;
    lines = [];
    // create lines by shifting words from the words array until the spacing is optimal.
If compact
    // true then will true and fit as many words as possible. Else it will try and get the
spacing as
    // close as possible to the normal spacing
    while(words.length > 0){
        lastLineWidth = 0;
        lastSize = -1;
        lineFound = false;
        // each line must have at least one word.
        word = words.shift();
        lineWidth = word.width;
        lineWords = [word.word];
        count = 0;
        while(lineWidth < width && words.length > 0){ // Add words to line
            word = words.shift();
            lineWidth += word.width;
            lineWords.push(word.word);
            count += 1;
            spaces = count - 1;
            adjSpace = (width - lineWidth) / spaces;
            if(minS > adjSpace){ // if spacing less than min remove last word and finish
line

```

```

        lineFound = true;
        words.unshift(word);
        lineWords.pop();
    }else{
        if(!compact){ // if compact mode
            if(adjSpace < spaceWidth){ // if less than normal space width
                if(lastSize === -1){
                    lastSize = adjSpace;
                }
                // check if with last word on if its closer to space width
                if(Math.abs(spaceWidth - adjSpace) < Math.abs(spaceWidth -
lastSize)){
                    lineFound = true; // yes keep it
                }else{
                    words.unshift(word); // no better fit if last word removes
                    lineWords.pop();
                    lineFound = true;
                }
            }
        }
        lastSize = adjSpace; // remember spacing
    }
    lines.push(lineWords.join(" ")); // and the line
}
// lines have been worked out get font size, render, and render all the lines. last
// line may need to be rendered as normal so it is outside the loop.
fontSize = getFontSize(ctx.font);
render = stroke === true ? ctx.strokeJustifyText.bind(ctx) :
ctx.fillTextJustifyText.bind(ctx);
for(i = 0; i < lines.length - 1; i++){
    render(lines[i], x, y, width, settings);
    y += lineSpacing * fontSize;
}
if(lines.length > 0){ // last line if left or start aligned for no justify
    if(ctx.textAlign === "left" || ctx.textAlign === "start"){
        render(lines[lines.length - 1], x, y, width, noJustifySetting);
        ctx.measureJustifiedText("", width, settings);
    }else{
        render(lines[lines.length - 1], x, y, width);
    }
}
// return details about the paragraph.
y += lineSpacing * fontSize;
return {
    nextLine : y,
    fontSize : fontSize,
    lineHeight : lineSpacing * fontSize,
};
}
// define fill
var fillParagraphText = function(text, x, y, width, settings){
    justifiedTextSettings(settings);
    settings = {
        minSpaceSize : minSpaceSize,
        maxSpaceSize : maxSpaceSize,
    };
    return justifiedPar(this, text, x, y, width, settings);
}
// define stroke
var strokeParagraphText = function(text, x, y, width, settings){

```

```

        justifiedTextSettings(settings);
        settings = {
            minSpaceSize : minSpaceSize,
            maxSpaceSize : maxSpaceSize,
        };
        return justifiedPar(this, text, x, y, width, settings,true);
    }
    CanvasRenderingContext2D.prototype.fillParaText = fillParagraphText;
    CanvasRenderingContext2D.prototype.strokeParaText = strokeParagraphText;
}) ();

```

leftstart° °

° ° ° ° **while+** words[0].width += ?°

<https://riptutorial.com/zh-TW/html5-canvas/topic/5235/>



# 11:

## Examples

◦

context.bezierCurveTo◦

```
// Return: an array of approximately evenly spaced points along a cubic Bezier curve
//
// Attribution: Stackoverflow's @Blindman67
// Cite: http://stackoverflow.com/questions/36637211/drawing-a-curved-line-in-css-or-canvas-
and-moving-circle-along-it/36827074#36827074
// As modified from the above citation
//
// ptCount: sample this many points at interval along the curve
// pxTolerance: approximate spacing allowed between points
// Ax,Ay,Bx,By,Cx,Cy,Dx,Dy: control points defining the curve
//
function plotCBez(ptCount,pxTolerance,Ax,Ay,Bx,By,Cx,Cy,Dx,Dy) {
    var deltaBAx=Bx-Ax;
    var deltaCBx=Cx-Bx;
    var deltaDCx=Dx-Cx;
    var deltaBAy=By-Ay;
    var deltaCBy=Cy-By;
    var deltaDCy=Dy-Cy;
    var ax,ay,bx,by;
    var lastX=-10000;
    var lastY=-10000;
    var pts=[{x:Ax,y:Ay}];
    for(var i=1;i<ptCount;i++){
        var t=i/ptCount;
        ax=Ax+deltaBAx*t;
        bx=Bx+deltaCBx*t;
        cx=Cx+deltaDCx*t;
        ax+=(bx-ax)*t;
        bx+=(cx-bx)*t;
        //
        ay=Ay+deltaBAy*t;
        by=By+deltaCBy*t;
        cy=Cy+deltaDCy*t;
        ay+=(by-ay)*t;
        by+=(cy-by)*t;
        var x=ax+(bx-ax)*t;
        var y=ay+(by-ay)*t;
        var dx=x-lastX;
        var dy=y-lastY;
        if(dx*dx+dy*dy>pxTolerance){
            pts.push({x:x,y:y});
            lastX=x;
            lastY=y;
        }
    }
    pts.push({x:Dx,y:Dy});
    return(pts);
}
```

◦

context.quadraticCurveTo◦

```
// Return: an array of approximately evenly spaced points along a Quadratic curve
//
// Attribution: Stackoverflow's @Blindman67
// Cite: http://stackoverflow.com/questions/36637211/drawing-a-curved-line-in-css-or-canvas-
and-moving-circle-along-it/36827074#36827074
// As modified from the above citation
//
// ptCount: sample this many points at interval along the curve
// pxTolerance: approximate spacing allowed between points
// Ax,Ay,Bx,By,Cx,Cy: control points defining the curve
//
function plotQBez(ptCount,pxTolerance,Ax,Ay,Bx,By,Cx,Cy){
    var deltaBAx=Bx-Ax;
    var deltaCBx=Cx-Bx;
    var deltaBAy=By-Ay;
    var deltaCBy=Cy-By;
    var ax,ay;
    var lastX=-10000;
    var lastY=-10000;
    var pts=[{x:Ax,y:Ay}];
    for(var i=1;i<ptCount;i++){
        var t=i/ptCount;
        ax=Ax+deltaBAx*t;
        ay=Ay+deltaBAy*t;
        var x=ax+((Bx+deltaCBx*t)-ax)*t;
        var y=ay+((By+deltaCBy*t)-ay)*t;
        var dx=x-lastX;
        var dy=y-lastY;
        if(dx*dx+dy*dy>pxTolerance){
            pts.push({x:x,y:y});
            lastX=x;
            lastY=y;
        }
    }
    pts.push({x:Cx,y:Cy});
    return(pts);
}
```

◦

context.lineToPath◦

```
// Return: an array of approximately evenly spaced points along a line
//
// pxTolerance: approximate spacing allowed between points
// Ax,Ay,Bx,By: end points defining the line
//
function plotLine(pxTolerance,Ax,Ay,Bx,By){
    var dx=Bx-Ax;
    var dy=By-Ay;
    var ptCount=parseInt(Math.sqrt(dx*dx+dy*dy))*3;
    var lastX=-10000;
    var lastY=-10000;
    var pts=[{x:Ax,y:Ay}];
```

```

    for(var i=1;i<=ptCount;i++){
        var t=i/ptCount;
        var x=Ax+dx*t;
        var y=Ay+dy*t;
        var dx1=x-lastX;
        var dy1=y-lastY;
        if(dx1*dx1+dy1*dy1>pxTolerance){
            pts.push({x:x,y:y});
            lastX=x;
            lastY=y;
        }
    }
    pts.push({x:Bx,y:By});
    return(pts);
}

```

◦

context.lineTo context.quadraticCurveTo/context.bezierCurveTo **PathPath**◦

```

// Path related variables
var A={x:50,y:100};
var B={x:125,y:25};
var BB={x:150,y:15};
var BB2={x:150,y:185};
var C={x:175,y:200};
var D={x:300,y:150};
var n=1000;
var tolerance=1.5;
var pts;

// canvas related variables
var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
document.body.appendChild(canvas);
canvas.width=378;
canvas.height=256;

// Tell the Context to plot waypoint in addition to
// drawing the path
plotPathCommands(ctx,n,tolerance);

// Path drawing commands
ctx.beginPath();
ctx.moveTo(A.x,A.y);
ctx.bezierCurveTo(B.x,B.y,C.x,C.y,D.x,D.y);
ctx.quadraticCurveTo(BB.x,BB.y,A.x,A.y);
ctx.lineTo(D.x,D.y);
ctx.strokeStyle='gray';
ctx.stroke();

// Tell the Context to stop plotting waypoints
ctx.stopPlottingPathCommands();

// Demo: Incrementally draw the path using the plotted points
ptsToRects(ctx.getPathPoints());
function ptsToRects(pts){
    ctx.fillStyle='red';
    var i=0;

```

```

requestAnimationFrame(animate);
function animate(){
    ctx.fillRect(pts[i].x-0.50,pts[i].y-0.50,tolerance,tolerance);
    i++;
    if(i<pts.length){ requestAnimationFrame(animate); }
}
}

```

## Canvas Context

- beginPath
- 
- lineTo
- quadraticCurveTo
- bezierCurveTo

## ContextstopPlottingPathCommandsContext

“” -

getPathPointspoints-array

points-array

getPathPointsclearPathPoints

```

// Path related variables
var A={x:50,y:100};
var B={x:125,y:25};
var BB={x:150,y:15};
var BB2={x:150,y:185};
var C={x:175,y:200};
var D={x:300,y:150};
var n=1000;
var tolerance=1.5;
var pts;

// canvas related variables
var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
document.body.appendChild(canvas);
canvas.width=378;
canvas.height=256;

// Tell the Context to plot waypoint in addition to
// drawing the path
plotPathCommands(ctx,n,tolerance);

// Path drawing commands
ctx.beginPath();
ctx.moveTo(A.x,A.y);
ctx.bezierCurveTo(B.x,B.y,C.x,C.y,D.x,D.y);
ctx.quadraticCurveTo(BB.x,BB.y,A.x,A.y);
ctx.lineTo(D.x,D.y);
ctx.strokeStyle='gray';
ctx.stroke();

```

```

// Tell the Context to stop plotting waypoints
ctx.stopPlottingPathCommands();

// Demo: Incrementally draw the path using the plotted points
ptsToRects(ctx.getPathPoints());
function ptsToRects(pts){
    ctx.fillStyle='red';
    var i=0;
    requestAnimationFrame(animate);
    function animate(){
        ctx.fillRect(pts[i].x-0.50,pts[i].y-0.50,tolerance,tolerance);
        i++;
        if(i<pts.length){ requestAnimationFrame(animate); }
    }
}

```

```

// Path related variables
var A={x:50,y:100};
var B={x:125,y:25};
var BB={x:150,y:15};
var BB2={x:150,y:185};
var C={x:175,y:200};
var D={x:300,y:150};
var n=1000;
var tolerance=1.5;
var pts;

// canvas related variables
var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
document.body.appendChild(canvas);
canvas.width=378;
canvas.height=256;

// Tell the Context to plot waypoint in addition to
// drawing the path
plotPathCommands(ctx,n,tolerance);

// Path drawing commands
ctx.beginPath();
ctx.moveTo(A.x,A.y);
ctx.bezierCurveTo(B.x,B.y,C.x,C.y,D.x,D.y);
ctx.quadraticCurveTo(BB.x,BB.y,A.x,A.y);
ctx.lineTo(D.x,D.y);
ctx.strokeStyle='gray';
ctx.stroke();

// Tell the Context to stop plotting waypoints
ctx.stopPlottingPathCommands();

// Demo: Incrementally draw the path using the plotted points
ptsToRects(ctx.getPathPoints());
function ptsToRects(pts){
    ctx.fillStyle='red';
    var i=0;
    requestAnimationFrame(animate);
    function animate(){
        ctx.fillRect(pts[i].x-0.50,pts[i].y-0.50,tolerance,tolerance);
        i++;
    }
}

```

```

        if(i<pts.length){ requestAnimationFrame(animate); }
    }
}

```

3.

```

function quadraticBezierLength(x1,y1,x2,y2,x3,y3)
    var a, e, c, d, u, a1, e1, c1, d1, u1, v1x, v1y;

    v1x = x2 * 2;
    v1y = y2 * 2;
    d = x1 - v1x + x3;
    d1 = y1 - v1y + y3;
    e = v1x - 2 * x1;
    e1 = v1y - 2 * y1;
    c1 = (a = 4 * (d * d + d1 * d1));
    c1 += (b = 4 * (d * e + d1 * e1));
    c1 += (c = e * e + e1 * e1);
    c1 = 2 * Math.sqrt(c1);
    a1 = 2 * a * (u = Math.sqrt(a));
    u1 = b / u;
    a = 4 * c * a - b * b;
    c = 2 * Math.sqrt(c);
    return (a1 * c1 + u * b * (c1 - c) + a * Math.log((2 * u + u1 + c1) / (u1 + c))) / (4 *
a1);
}

```

$$F_t = a * 1 - t^2 + 2 * b * 1 - t * t + c * t^2$$

。

splitCurveAtposition 0.0 = 0.5 =1 =。 。 Xx4。 undefinednull

```

var p1 = {x : 10 , y : 100};
var p2 = {x : 100, y : 200};
var p3 = {x : 200, y : 0};
var newCurves = splitCurveAt(0.5, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y)

var i = 0;
var p = newCurves
// Draw the 2 new curves
// Assumes ctx is canvas 2d context
ctx.lineWidth = 1;
ctx.strokeStyle = "black";
ctx.beginPath();
ctx.moveTo(p[i++],p[i++]);
ctx.quadraticCurveTo(p[i++], p[i++], p[i++], p[i++]);
ctx.quadraticCurveTo(p[i++], p[i++], p[i++], p[i++]);
ctx.stroke();

```

```

var p1 = {x : 10 , y : 100};
var p2 = {x : 100, y : 200};
var p3 = {x : 200, y : 0};
var newCurves = splitCurveAt(0.5, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y)

var i = 0;

```

```

var p = newCurves
// Draw the 2 new curves
// Assumes ctx is canvas 2d context
ctx.lineWidth = 1;
ctx.strokeStyle = "black";
ctx.beginPath();
ctx.moveTo(p[i++],p[i++]);
ctx.quadraticCurveTo(p[i++], p[i++], p[i++], p[i++]);
ctx.quadraticCurveTo(p[i++], p[i++], p[i++], p[i++]);
ctx.stroke();

```

## splitCurveAt = function(position, x1, y1, x2, y2, x3, y3, [x4, y4])

[x4, y4]。

/\* \*/ position >= 0 position >= 1 。

```

var p1 = {x : 10 , y : 100};
var p2 = {x : 100, y : 200};
var p3 = {x : 200, y : 0};
var newCurves = splitCurveAt(0.5, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y)

var i = 0;
var p = newCurves
// Draw the 2 new curves
// Assumes ctx is canvas 2d context
ctx.lineWidth = 1;
ctx.strokeStyle = "black";
ctx.beginPath();
ctx.moveTo(p[i++],p[i++]);
ctx.quadraticCurveTo(p[i++], p[i++], p[i++], p[i++]);
ctx.quadraticCurveTo(p[i++], p[i++], p[i++], p[i++]);
ctx.stroke();

```

。

。

trimBezier fromPostoPos 。

fromPostoPos01。

Xx4。

undefinednull

。 68。

。

```

var p1 = {x : 10 , y : 100};
var p2 = {x : 100, y : 200};
var p3 = {x : 200, y : 0};
var newCurve = splitCurveAt(0.25, 0.75, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y)

var i = 0;
var p = newCurve
// Draw the trimmed curve
// Assumes ctx is canvas 2d context

```

```
ctx.lineWidth = 1;
ctx.strokeStyle = "black";
ctx.beginPath();
ctx.moveTo(p[i++],p[i++]);
ctx.quadraticCurveTo(p[i++], p[i++], p[i++], p[i++]);
ctx.stroke();
```

◦

```
var p1 = {x : 10 , y : 100};
var p2 = {x : 100, y : 200};
var p3 = {x : 200, y : 0};
var newCurve = splitCurveAt(0.25, 0.75, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y)

var i = 0;
var p = newCurve
// Draw the trimmed curve
// Assumes ctx is canvas 2d context
ctx.lineWidth = 1;
ctx.strokeStyle = "black";
ctx.beginPath();
ctx.moveTo(p[i++],p[i++]);
ctx.quadraticCurveTo(p[i++], p[i++], p[i++], p[i++]);
ctx.stroke();
```

---

**trimBezier = functionfromPostoPosx1y1x2y2x3y3[x4y4]**

**[x4y4]**◦

**Split Bezier Curves At**

```
var p1 = {x : 10 , y : 100};
var p2 = {x : 100, y : 200};
var p3 = {x : 200, y : 0};
var newCurve = splitCurveAt(0.25, 0.75, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y)

var i = 0;
var p = newCurve
// Draw the trimmed curve
// Assumes ctx is canvas 2d context
ctx.lineWidth = 1;
ctx.strokeStyle = "black";
ctx.beginPath();
ctx.moveTo(p[i++],p[i++]);
ctx.quadraticCurveTo(p[i++], p[i++], p[i++], p[i++]);
ctx.stroke();
```

**4**◦

---

◦ “”◦

**4099 +**◦



```

// Return: Close approximation of the length of a Cubic Bezier curve
//
// Ax,Ay,Bx,By,Cx,Cy,Dx,Dy: the 4 control points of the curve
// sampleCount [optional, default=40]: how many intervals to calculate
// Requires: cubicQxy (included below)
//
function cubicBezierLength(Ax,Ay,Bx,By,Cx,Cy,Dx,Dy,sampleCount){
    var ptCount=sampleCount||40;
    var totDist=0;
    var lastX=Ax;
    var lastY=Ay;
    var dx,dy;
    for(var i=1;i<ptCount;i++){
        var pt=cubicQxy(i/ptCount,Ax,Ay,Bx,By,Cx,Cy,Dx,Dy);
        dx=pt.x-lastX;
        dy=pt.y-lastY;
        totDist+=Math.sqrt(dx*dx+dy*dy);
        lastX=pt.x;
        lastY=pt.y;
    }
    dx=Dx-lastX;
    dy=Dy-lastY;
    totDist+=Math.sqrt(dx*dx+dy*dy);
    return(parseInt(totDist));
}

// Return: an [x,y] point along a cubic Bezier curve at interval T
//
// Attribution: Stackoverflow's @Blindman67
// Cite: http://stackoverflow.com/questions/36637211/drawing-a-curved-line-in-css-or-canvas-
// and-moving-circle-along-it/36827074#36827074
// As modified from the above citation
//
// t: an interval along the curve (0<=t<=1)
// ax,ay,bx,by,cx,cy,dx,dy: control points defining the curve
//
function cubicQxy(t,ax,ay,bx,by,cx,cy,dx,dy) {
    ax += (bx - ax) * t;
    bx += (cx - bx) * t;
    cx += (dx - cx) * t;
    ax += (bx - ax) * t;
    bx += (cx - bx) * t;
    ay += (by - ay) * t;
    by += (cy - by) * t;
    cy += (dx - cy) * t;
    ay += (by - ay) * t;
    by += (cy - by) * t;
    return({
        x:ax +(bx - ax) * t,
        y:ay +(by - ay) * t
    });
}

```

positionbeziercubicpositionposition0 <= position <= 1.<0> 10,1。

68。

。。

```

var p1 = {x : 10 , y : 100};
var p2 = {x : 100, y : 200};
var p3 = {x : 200, y : 0};
var p4 = {x : 300, y : 100};
var point = {x : null, y : null};

// for cubic beziers
point = getPointOnCurve(0.5, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y, p4.x, p4.y, point);
// or No need to set point as it is a reference and will be set
getPointOnCurve(0.5, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y, p4.x, p4.y, point);
// or to create a new point
var point1 = getPointOnCurve(0.5, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y, p4.x, p4.y);

// for quadratic beziers
point = getPointOnCurve(0.5, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y, null, null, point);
// or No need to set point as it is a reference and will be set
getPointOnCurve(0.5, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y, null, null, point);
// or to create a new point
var point1 = getPointOnCurve(0.5, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y);

```

## getPointOnCurve = function positionx1y1x2y2x3y3[x4y4][vec]

[x4y4].

x4 y4nullundefined° vec° °

```

var p1 = {x : 10 , y : 100};
var p2 = {x : 100, y : 200};
var p3 = {x : 200, y : 0};
var p4 = {x : 300, y : 100};
var point = {x : null, y : null};

// for cubic beziers
point = getPointOnCurve(0.5, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y, p4.x, p4.y, point);
// or No need to set point as it is a reference and will be set
getPointOnCurve(0.5, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y, p4.x, p4.y, point);
// or to create a new point
var point1 = getPointOnCurve(0.5, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y, p4.x, p4.y);

// for quadratic beziers
point = getPointOnCurve(0.5, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y, null, null, point);
// or No need to set point as it is a reference and will be set
getPointOnCurve(0.5, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y, null, null, point);
// or to create a new point
var point1 = getPointOnCurve(0.5, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y);

```

°

```

// This method was discovered by Blindman67 and solves by first normalising the control point
thereby reducing the algorithm complexity
// x1,y1, x2,y2, x3,y3 Start, Control, and End coords of bezier
// [extent] is optional and if provided the extent will be added to it allowing you to use the
function
// to get the extent of many beziers.
// returns extent object (if not supplied a new extent is created)
// Extent object properties

```

```

// top, left, right, bottom, width, height
function getQuadraticCurveExtent(x1, y1, x2, y2, x3, y3, extent) {
    var brx, bx, x, bry, by, y, px, py;

    // solve quadratic for bounds by BM67 normalizing equation
    brx = x3 - x1; // get x range
    bx = x2 - x1; // get x control point offset
    x = bx / brx; // normalise control point which is used to check if maxima is in range

    // do the same for the y points
    bry = y3 - y1;
    by = y2 - y1;
    y = by / bry;

    px = x1; // set defaults in case maxima outside range
    py = y1;

    // find top/left, top/right, bottom/left, or bottom/right
    if (x < 0 || x > 1) { // check if x maxima is on the curve
        px = bx * bx / (2 * bx - brx) + x1; // get the x maxima
    }
    if (y < 0 || y > 1) { // same as x
        py = by * by / (2 * by - bry) + y1;
    }

    // create extent object and add extent
    if (extent === undefined) {
        extent = {};
        extent.left = Math.min(x1, x3, px);
        extent.top = Math.min(y1, y3, py);
        extent.right = Math.max(x1, x3, px);
        extent.bottom = Math.max(y1, y3, py);
    } else { // use supplied extent and extend it to fit this curve
        extent.left = Math.min(x1, x3, px, extent.left);
        extent.top = Math.min(y1, y3, py, extent.top);
        extent.right = Math.max(x1, x3, px, extent.right);
        extent.bottom = Math.max(y1, y3, py, extent.bottom);
    }

    extent.width = extent.right - extent.left;
    extent.height = extent.bottom - extent.top;
    return extent;
}

```

。

<https://riptutorial.com/zh-TW/html5-canvas/topic/5281/>

## 12:

- void clearRectxywidthheight
- ImageData createImageDatawidthheight

alpha: false°

## Examples

clearRect°

```
// Clear the entire canvas
ctx.clearRect(0, 0, canvas.width, canvas.height);
```

clearRect°

°

```
// Clear the entire canvas
ctx.clearRect(0, 0, canvas.width, canvas.height);
```

2Dctx.savectx.restore ° °

putImageData° °

```
var imageData = ctx.createImageData(canvas.width, canvas.height);
ctx.putImageData(imageData, 0, 0);
```

putImageData° °

globalCompositeOperation°

```
// All pixels being drawn will be transparent
ctx.globalCompositeOperation = 'destination-out';

// Clear a triangular section
ctx.globalAlpha = 1;    // ensure alpha is 1
ctx.fillStyle = '#000'; // ensure the current fillStyle does not have any transparency
ctx.beginPath();
ctx.moveTo(10, 0);
ctx.lineTo(0, 10);
ctx.lineTo(20, 10);
ctx.fill();

// Begin drawing normally again
ctx.globalCompositeOperation = 'source-over';
```

°

clearRect ◦

```
// create the background gradient once
var bgGrad = ctx.createLinearGradient(0,0,0,canvas.height);
bgGrad.addColorStop(0,"#0FF");
bgGrad.addColorStop(1,"#08F");

// Every time you need to clear the canvas
ctx.fillStyle = bgGrad;
ctx.fillRect(0,0,canvas.width,canvas.height);
```

**clearRect** 0.004ms0.008ms **400**◦ ◦

◦ clearRect()◦

```
ctx.globalCompositeOperation = 'copy';
```

<https://riptutorial.com/zh-TW/html5-canvas/topic/5245/>

# 13:

## Examples

```
// circle objects: { x:, y:, radius: }
// return true if the 2 circles are colliding
// c1 and c2 are circles as defined above

function CirclesColliding(c1,c2){
    var dx=c2.x-c1.x;
    var dy=c2.y-c1.y;
    var rSum=c1.radius+c2.radius;
    return (dx*dx+dy*dy<=rSum*rSum);
}
```

## 2

```
// rectangle objects { x:, y:, width:, height: }
// return true if the 2 rectangles are colliding
// r1 and r2 are rectangles as defined above

function RectsColliding(r1,r2){
    return !(
        r1.x>r2.x+r2.width ||
        r1.x+r1.width<r2.x ||
        r1.y>r2.y+r2.height ||
        r1.y+r1.height<r2.y
    );
}
```

```
// rectangle object: { x:, y:, width:, height: }
// circle object: { x:, y:, radius: }
// return true if the rectangle and circle are colliding

function RectCircleColliding(rect,circle){
    var dx=Math.abs(circle.x-(rect.x+rect.width/2));
    var dy=Math.abs(circle.y-(rect.y+rect.height/2));

    if( dx > circle.radius+rect.width/2 ){ return(false); }
    if( dy > circle.radius+rect.height/2 ){ return(false); }

    if( dx <= rect.width ){ return(true); }
    if( dy <= rect.height ){ return(true); }

    var dx=dx-rect.width;
    var dy=dy-rect.height
    return(dx*dx+dy*dy<=circle.radius*circle.radius);
}
```

## 2

truefalse °

```

// point object: {x:, y:}
// p0 & p1 form one segment, p2 & p3 form the second segment
// Returns true if lines segments are intercepting
var lineSegmentsIntercept = (function(){ // function as singleton so that closure can be
used

    var v1, v2, v3, cross, u1, u2; // working variable are closed over so they do not
need creation

    // each time the function is called. This gives a
significant performance boost.
    v1 = {x : null, y : null}; // line p0, p1 as vector
    v2 = {x : null, y : null}; // line p2, p3 as vector
    v3 = {x : null, y : null}; // the line from p0 to p2 as vector

    function lineSegmentsIntercept (p0, p1, p2, p3) {
        v1.x = p1.x - p0.x; // line p0, p1 as vector
        v1.y = p1.y - p0.y;
        v2.x = p3.x - p2.x; // line p2, p3 as vector
        v2.y = p3.y - p2.y;
        if((cross = v1.x * v2.y - v1.y * v2.x) === 0){ // cross prod 0 if lines parallel
            return false; // no intercept
        }
        v3 = {x : p0.x - p2.x, y : p0.y - p2.y}; // the line from p0 to p2 as vector
        u2 = (v1.x * v3.y - v1.y * v3.x) / cross; // get unit distance along line p2 p3
        // code point B
        if (u2 >= 0 && u2 <= 1){ // is intercept on line p2, p3
            u1 = (v2.x * v3.y - v2.y * v3.x) / cross; // get unit distance on line p0, p1;
            // code point A
            return (u1 >= 0 && u1 <= 1); // return true if on line else false.
            // code point A end
        }
        return false; // no intercept;
        // code point B end
    }
    return lineSegmentsIntercept; // return function with closure for optimisation.
})();

```

```

// point object: {x:, y:}
// p0 & p1 form one segment, p2 & p3 form the second segment
// Returns true if lines segments are intercepting
var lineSegmentsIntercept = (function(){ // function as singleton so that closure can be
used

    var v1, v2, v3, cross, u1, u2; // working variable are closed over so they do not
need creation

    // each time the function is called. This gives a
significant performance boost.
    v1 = {x : null, y : null}; // line p0, p1 as vector
    v2 = {x : null, y : null}; // line p2, p3 as vector
    v3 = {x : null, y : null}; // the line from p0 to p2 as vector

    function lineSegmentsIntercept (p0, p1, p2, p3) {
        v1.x = p1.x - p0.x; // line p0, p1 as vector
        v1.y = p1.y - p0.y;
        v2.x = p3.x - p2.x; // line p2, p3 as vector
        v2.y = p3.y - p2.y;
        if((cross = v1.x * v2.y - v1.y * v2.x) === 0){ // cross prod 0 if lines parallel
            return false; // no intercept
        }
        v3 = {x : p0.x - p2.x, y : p0.y - p2.y}; // the line from p0 to p2 as vector

```

```

    u2 = (v1.x * v3.y - v1.y * v3.x) / cross; // get unit distance along line p2 p3
    // code point B
    if (u2 >= 0 && u2 <= 1){ // is intercept on line p2, p3
        u1 = (v2.x * v3.y - v2.y * v3.x) / cross; // get unit distance on line p0, p1;
        // code point A
        return (u1 >= 0 && u1 <= 1); // return true if on line else false.
        // code point A end
    }
    return false; // no intercept;
    // code point B end
}
return lineSegmentsIntercept; // return function with closure for optimisation.
})();

```

◦ code point AA endcode point A

```

// point object: {x:, y:}
// p0 & p1 form one segment, p2 & p3 form the second segment
// Returns true if lines segments are intercepting
var lineSegmentsIntercept = (function(){ // function as singleton so that closure can be
used

    var v1, v2, v3, cross, u1, u2; // working variable are closed over so they do not
need creation

    // each time the function is called. This gives a
significant performance boost.
    v1 = {x : null, y : null}; // line p0, p1 as vector
    v2 = {x : null, y : null}; // line p2, p3 as vector
    v3 = {x : null, y : null}; // the line from p0 to p2 as vector

    function lineSegmentsIntercept (p0, p1, p2, p3) {
        v1.x = p1.x - p0.x; // line p0, p1 as vector
        v1.y = p1.y - p0.y;
        v2.x = p3.x - p2.x; // line p2, p3 as vector
        v2.y = p3.y - p2.y;
        if((cross = v1.x * v2.y - v1.y * v2.x) === 0){ // cross prod 0 if lines parallel
            return false; // no intercept
        }
        v3 = {x : p0.x - p2.x, y : p0.y - p2.y}; // the line from p0 to p2 as vector
        u2 = (v1.x * v3.y - v1.y * v3.x) / cross; // get unit distance along line p2 p3
        // code point B
        if (u2 >= 0 && u2 <= 1){ // is intercept on line p2, p3
            u1 = (v2.x * v3.y - v2.y * v3.x) / cross; // get unit distance on line p0, p1;
            // code point A
            return (u1 >= 0 && u1 <= 1); // return true if on line else false.
            // code point A end
        }
        return false; // no intercept;
        // code point B end
    }
    return lineSegmentsIntercept; // return function with closure for optimisation.
})();

```

code point BB endcode point B B end

```

// point object: {x:, y:}
// p0 & p1 form one segment, p2 & p3 form the second segment
// Returns true if lines segments are intercepting
var lineSegmentsIntercept = (function(){ // function as singleton so that closure can be

```



used

```
var v1, v2, v3, cross, u1, u2; // working variable are closed over so they do not
need creation

// each time the function is called. This gives a
significant performance boost.
v1 = {x : null, y : null}; // line p0, p1 as vector
v2 = {x : null, y : null}; // line p2, p3 as vector
v3 = {x : null, y : null}; // the line from p0 to p2 as vector

function lineSegmentsIntercept (p0, p1, p2, p3) {
  v1.x = p1.x - p0.x; // line p0, p1 as vector
  v1.y = p1.y - p0.y;
  v2.x = p3.x - p2.x; // line p2, p3 as vector
  v2.y = p3.y - p2.y;
  if((cross = v1.x * v2.y - v1.y * v2.x) === 0){ // cross prod 0 if lines parallel
    return false; // no intercept
  }
  v3 = {x : p0.x - p2.x, y : p0.y - p2.y}; // the line from p0 to p2 as vector
  u2 = (v1.x * v3.y - v1.y * v3.x) / cross; // get unit distance along line p2 p3
  // code point B
  if (u2 >= 0 && u2 <= 1){ // is intercept on line p2, p3
    u1 = (v2.x * v3.y - v2.y * v3.x) / cross; // get unit distance on line p0, p1;
    // code point A
    return (u1 >= 0 && u1 <= 1); // return true if on line else false.
    // code point A end
  }
  return false; // no intercept;
  // code point B end
}
return lineSegmentsIntercept; // return function with closure for optimisation.
})();
```

false{x : xCoord, y : yCoord}

```
// [x0,y0] to [x1,y1] define a line segment
// [cx,cy] is circle centerpoint, cr is circle radius
function isCircleSegmentColliding(x0,y0,x1,y1,cx,cy,cr){

  // calc delta distance: source point to line start
  var dx=cx-x0;
  var dy=cy-y0;

  // calc delta distance: line start to end
  var dxx=x1-x0;
  var dyy=y1-y0;

  // Calc position on line normalized between 0.00 & 1.00
  // == dot product divided by delta line distances squared
  var t=(dx*dxx+dy*dyy)/(dxx*dxx+dyy*dyy);

  // calc nearest pt on line
  var x=x0+dxx*t;
  var y=y0+dyy*t;

  // clamp results to being on the segment
  if(t<0){x=x0;y=y0;}
  if(t>1){x=x1;y=y1;}

  return( (cx-x)*(cx-x)+(cy-y)*(cy-y) < cr*cr );
```

```

}

// var rect={x:,y:,width:,height:};
// var line={x1:,y1:,x2:,y2:};
// Get intersetting point of line segment & rectangle (if any)
function lineRectCollide(line,rect){

    // p=line startpoint, p2=line endpoint
    var p={x:line.x1,y:line.y1};
    var p2={x:line.x2,y:line.y2};

    // top rect line
    var q={x:rect.x,y:rect.y};
    var q2={x:rect.x+rect.width,y:rect.y};
    if(lineSegmentsCollide(p,p2,q,q2)){ return true; }
    // right rect line
    var q=q2;
    var q2={x:rect.x+rect.width,y:rect.y+rect.height};
    if(lineSegmentsCollide(p,p2,q,q2)){ return true; }
    // bottom rect line
    var q=q2;
    var q2={x:rect.x,y:rect.y+rect.height};
    if(lineSegmentsCollide(p,p2,q,q2)){ return true; }
    // left rect line
    var q=q2;
    var q2={x:rect.x,y:rect.y};
    if(lineSegmentsCollide(p,p2,q,q2)){ return true; }

    // not intersecting with any of the 4 rect sides
    return(false);
}

// point object: {x:, y:}
// p0 & p1 form one segment, p2 & p3 form the second segment
// Get intersetting point of 2 line segments (if any)
// Attribution: http://paulbourke.net/geometry/pointlineplane/
function lineSegmentsCollide(p0,p1,p2,p3) {

    var unknownA = (p3.x-p2.x) * (p0.y-p2.y) - (p3.y-p2.y) * (p0.x-p2.x);
    var unknownB = (p1.x-p0.x) * (p0.y-p2.y) - (p1.y-p0.y) * (p0.x-p2.x);
    var denominator = (p3.y-p2.y) * (p1.x-p0.x) - (p3.x-p2.x) * (p1.y-p0.y);

    // Test if Coincident
    // If the denominator and numerator for the ua and ub are 0
    // then the two lines are coincident.
    if(unknownA==0 && unknownB==0 && denominator==0){return(null);}

    // Test if Parallel
    // If the denominator for the equations for ua and ub is 0
    // then the two lines are parallel.
    if (denominator == 0) return null;

    // test if line segments are colliding
    unknownA /= denominator;
    unknownB /= denominator;
    var isIntersecting=(unknownA>=0 && unknownA<=1 && unknownB>=0 && unknownB<=1)

    return(isIntersecting);
}

```

Markus Jarderot @ 2

```
// polygon objects are an array of vertices forming the polygon
//     var polygon1=[{x:100,y:100},{x:150,y:150},{x:50,y:150},...];
// THE POLYGONS MUST BE CONVEX
// return true if the 2 polygons are colliding

function convexPolygonsCollide(a, b){
    var polygons = [a, b];
    var minA, maxA, projected, i, i1, j, minB, maxB;

    for (i = 0; i < polygons.length; i++) {

        // for each polygon, look at each edge of the polygon, and determine if it separates
        // the two shapes
        var polygon = polygons[i];
        for (i1 = 0; i1 < polygon.length; i1++) {

            // grab 2 vertices to create an edge
            var i2 = (i1 + 1) % polygon.length;
            var p1 = polygon[i1];
            var p2 = polygon[i2];

            // find the line perpendicular to this edge
            var normal = { x: p2.y - p1.y, y: p1.x - p2.x };

            minA = maxA = undefined;
            // for each vertex in the first shape, project it onto the line perpendicular to
            the edge
            // and keep track of the min and max of these values
            for (j = 0; j < a.length; j++) {
                projected = normal.x * a[j].x + normal.y * a[j].y;
                if (minA==undefined || projected < minA) {
                    minA = projected;
                }
                if (maxA==undefined || projected > maxA) {
                    maxA = projected;
                }
            }

            // for each vertex in the second shape, project it onto the line perpendicular to
            the edge
            // and keep track of the min and max of these values
            minB = maxB = undefined;
            for (j = 0; j < b.length; j++) {
                projected = normal.x * b[j].x + normal.y * b[j].y;
                if (minB==undefined || projected < minB) {
                    minB = projected;
                }
                if (maxB==undefined || projected > maxB) {
                    maxB = projected;
                }
            }

            // if there is no overlap between the projects, the edge we are looking at
```

```

separates the two
    // polygons, and we know there is no overlap
    if (maxA < minB || maxB < minA) {
        return false;
    }
}
}
return true;
};

```

## 2

### 2.

```

// polygon objects are an array of vertices forming the polygon
//   var polygon1=[{x:100,y:100},{x:150,y:150},{x:50,y:150},...];
// The polygons can be both concave and convex
// return true if the 2 polygons are colliding

function polygonsCollide(p1,p2){
    // turn vertices into line points
    var lines1=verticesToLinePoints(p1);
    var lines2=verticesToLinePoints(p2);
    // test each poly1 side vs each poly2 side for intersections
    for(i=0; i<lines1.length; i++){
        for(j=0; j<lines2.length; j++){
            // test if sides intersect
            var p0=lines1[i][0];
            var p1=lines1[i][1];
            var p2=lines2[j][0];
            var p3=lines2[j][1];
            // found an intersection -- polys do collide
            if(lineSegmentsCollide(p0,p1,p2,p3)){return(true);}
        }
    }
    // none of the sides intersect
    return(false);
}

// helper: turn vertices into line points
function verticesToLinePoints(p){
    // make sure polys are self-closing
    if(!(p[0].x==p[p.length-1].x && p[0].y==p[p.length-1].y)){
        p.push({x:p[0].x,y:p[0].y});
    }
    var lines=[];
    for(var i=1;i<p.length;i++){
        var p1=p[i-1];
        var p2=p[i];
        lines.push([
            {x:p1.x, y:p1.y},
            {x:p2.x, y:p2.y}
        ]);
    }
    return(lines);
}

// helper: test line intersections
// point object: {x:, y:}
// p0 & p1 form one segment, p2 & p3 form the second segment
// Get intersecting point of 2 line segments (if any)
// Attribution: http://paulbourke.net/geometry/pointlineplane/

```

```

function lineSegmentsCollide(p0,p1,p2,p3) {
    var unknownA = (p3.x-p2.x) * (p0.y-p2.y) - (p3.y-p2.y) * (p0.x-p2.x);
    var unknownB = (p1.x-p0.x) * (p0.y-p2.y) - (p1.y-p0.y) * (p0.x-p2.x);
    var denominator = (p3.y-p2.y) * (p1.x-p0.x) - (p3.x-p2.x) * (p1.y-p0.y);

    // Test if Coincident
    // If the denominator and numerator for the ua and ub are 0
    // then the two lines are coincident.
    if(unknownA==0 && unknownB==0 && denominator==0){return(null);}

    // Test if Parallel
    // If the denominator for the equations for ua and ub is 0
    // then the two lines are parallel.
    if (denominator == 0) return null;

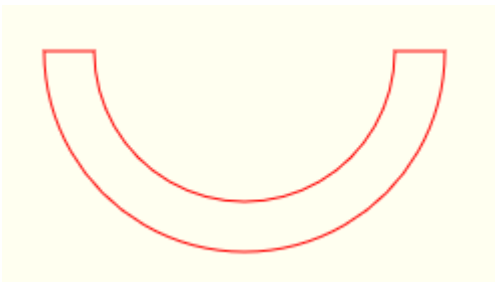
    // test if line segments are colliding
    unknownA /= denominator;
    unknownB /= denominator;
    var isIntersecting=(unknownA>=0 && unknownA<=1 && unknownB>=0 && unknownB<=1)

    return(isIntersecting);
}

```

XY

[xy]°



```

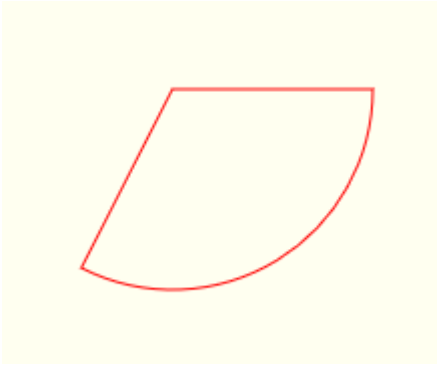
var arc={
    cx:150, cy:150,
    innerRadius:75, outerRadius:100,
    startAngle:0, endAngle:Math.PI
}

function isPointInArc(x,y,arc){
    var dx=x-arc.cx;
    var dy=y-arc.cy;
    var dxy=dx*dx+dy*dy;
    var rrOuter=arc.outerRadius*arc.outerRadius;
    var rrInner=arc.innerRadius*arc.innerRadius;
    if(dxy<rrInner || dxy>rrOuter){return(false);}
    var angle=(Math.atan2(dy,dx)+PI2)%PI2;
    return(angle>=arc.startAngle && angle<=arc.endAngle);
}

```

XY

[xy]°



```
// wedge objects: {cx:,cy:,radius:,startAngle:,endAngle:}
// var wedge={
//     cx:150, cy:150, // centerpoint
//     radius:100,
//     startAngle:0, endAngle:Math.PI
// }
// Return true if the x,y point is inside the closed wedge

function isPointInWedge(x,y,wedge){
    var PI2=Math.PI*2;
    var dx=x-wedge.cx;
    var dy=y-wedge.cy;
    var rr=wedge.radius*wedge.radius;
    if(dx*dx+dy*dy>rr){return(false);}
    var angle=(Math.atan2(dy,dx)+PI2)%PI2;
    return(angle>=wedge.startAngle && angle<=wedge.endAngle);
}
```

**XY**

**[xy]**<sup>o</sup>

```
// circle objects: {cx:,cy:,radius:,startAngle:,endAngle:}
// var circle={
//     cx:150, cy:150, // centerpoint
//     radius:100,
// }
// Return true if the x,y point is inside the circle

function isPointInCircle(x,y,circle){
    var dx=x-circle.cx;
    var dy=y-circle.cy;
    return(dx*dx+dy*dy<circle.radius*circle.radius);
}
```

**XY**

**[xy]**<sup>o</sup>

```
// rectangle objects: {x:, y:, width:, height: }
// var rect={x:10, y:15, width:25, height:20}
// Return true if the x,y point is inside the rectangle

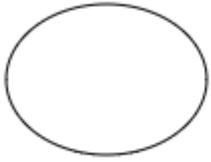
function isPointInRectangle(x,y,rect){
```


```
return(x>rect.x && x<rect.x+rect.width && y>rect.y && y<rect.y+rect.height);  
}
```

<https://riptutorial.com/zh-TW/html5-canvas/topic/5017/>

## 14:

### Examples



`context.ellipse`  

```
// draws an ellipse based on x,y being top-left coordinate
function drawEllipse(x,y,width,height){
    var PI2=Math.PI*2;
    var ratio=height/width;
    var radius=Math.max(width,height)/2;
    var increment = 1 / radius;
    var cx=x+width/2;
    var cy=y+height/2;

    ctx.beginPath();
    var x = cx + radius * Math.cos(0);
    var y = cy - ratio * radius * Math.sin(0);
    ctx.lineTo(x,y);

    for(var radians=increment; radians<PI2; radians+=increment){
        var x = cx + radius * Math.cos(radians);
        var y = cy - ratio * radius * Math.sin(radians);
        ctx.lineTo(x,y);
    }

    ctx.closePath();
    ctx.stroke();
}
```

```
// draws an ellipse based on x,y being top-left coordinate
function drawEllipse(x,y,width,height){
    var PI2=Math.PI*2;
    var ratio=height/width;
    var radius=Math.max(width,height)/2;
    var increment = 1 / radius;
    var cx=x+width/2;
    var cy=y+height/2;

    ctx.beginPath();
    var x = cx + radius * Math.cos(0);
    var y = cy - ratio * radius * Math.sin(0);
    ctx.lineTo(x,y);

    for(var radians=increment; radians<PI2; radians+=increment){
        var x = cx + radius * Math.cos(radians);
        var y = cy - ratio * radius * Math.sin(radians);
        ctx.lineTo(x,y);
    }
}
```



```

    }

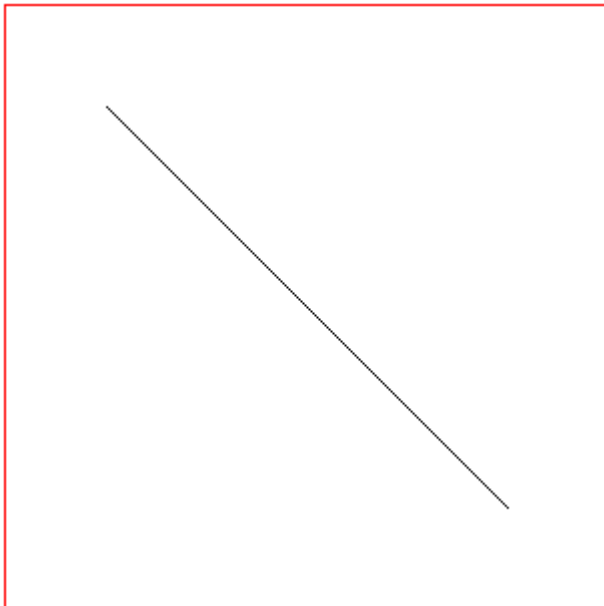
    ctx.closePath();
    ctx.stroke();
}

```

Canvas""。

Bresenham's\_line2。

*context.lineTo*



```

// Usage:
bresenhamLine(50,50,250,250);

// x,y line start
// xx,yy line end
// the pixel at line start and line end are drawn
function bresenhamLine(x, y, xx, yy){
    var oldFill = ctx.fillStyle; // save old fill style
    ctx.fillStyle = ctx.strokeStyle; // move stroke style to fill
    xx = Math.floor(xx);
    yy = Math.floor(yy);
    x = Math.floor(x);
    y = Math.floor(y);
    // BRESENHAM
    var dx = Math.abs(xx-x);
    var sx = x < xx ? 1 : -1;
    var dy = -Math.abs(yy-y);
    var sy = y < yy ? 1 : -1;
    var err = dx+dy;
    var errC; // error value
    var end = false;
    var x1 = x;
    var y1 = y;

    while(!end){
        ctx.fillRect(x1, y1, 1, 1); // draw each pixel as a rect

```

```
    if (x1 === xx && y1 === yy) {
        end = true;
    }else{
        errC = 2*err;
        if (errC >= dy) {
            err += dy;
            x1 += sx;
        }
        if (errC <= dx) {
            err += dx;
            y1 += sy;
        }
    }
}
ctx.fillStyle = oldFill; // restore old fill style
}
```

<https://riptutorial.com/zh-TW/html5-canvas/topic/5133/>

---

## 15:

- context.beginPath
- context.moveToSTARTXstartY
- context.lineToendX
- context.arccenterXcenterYradiusstartingRadianAngleendingRadianAngle
- context.quadraticCurveTocontrolXcontrolYendX
- context.bezierCurveTocontrolX1controlY1controlX2controlY2endX
- context.arcTopointX1pointY1pointX2pointY2radius
- context.rectleftXtopYwidthheight;
- context.closePath

## Examples

=====

TODO。 “draft”。

TODO“action”strokefillclip

=====

“”。

“”。

Canvas

- xy。
- 。
- == - CSS。
- beginPath
- 
- lineTo
- 
- quadraticCurveTo
- bezierCurveTo
- arcTo
- 
- closePath

---

### beginPath

```
context.beginPath()
```

。

◦ ◦

“”== coordinate [0,0]◦

```
context.beginPath()
```

[startXstartY]◦

◦ ◦ ◦ context.moveTo “”◦

## lineTo

```
context.beginPath()
```

[endXendY]

.lineTo◦ 3◦

```
context.beginPath()
```

◦ ◦ radians = degrees \* Math.PI / 180;◦

0◦ endsAngle = startingAngle + 360360== Math.PI 2`context.arc10,10,20,0Math.PI 2;

[true | false] context.arc(10,10,20,0,Math.PI\*2,true)

## quadraticCurveTo

```
context.beginPath()
```

◦ ◦

## bezierCurveTo

```
context.beginPath()
```

◦ ◦

## arcTo

```
context.beginPath()
```

◦ Point1Point2◦

◦

```
context.beginPath()
```

◦

`context.rect`◦ ◦

## closePath

```
context.beginPath()
```

◦

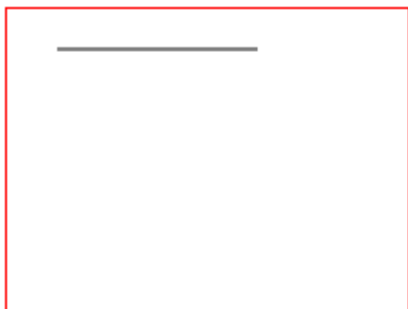
`2closePath`“”◦

◦ `context.closePath``context.beginPath`◦ `closePath` - “” `beginPath`◦

## lineTo

```
context.lineTo(endX, endY)
```

`[endXendY]`



```
context.lineTo(endX, endY)
```

`.lineTo`◦ 3◦



```
context.lineTo(endX, endY)
```

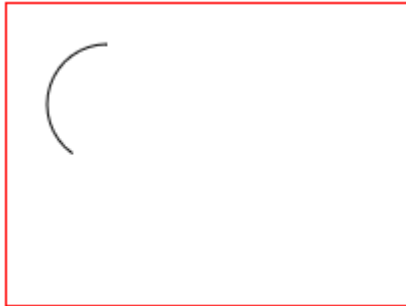
## arc

```
context.arc(centerX, centerY, radius, startingRadianAngle, endingRadianAngle)
```

◦ ◦ radians = degrees \* Math.PI / 180; ◦

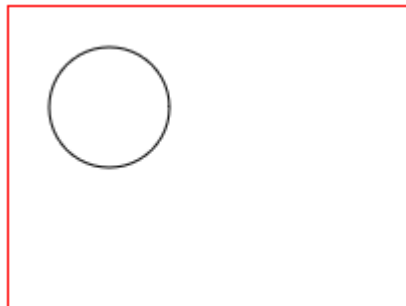
0◦

[true | false] context.arc(10,10,20,0,Math.PI\*2,true)



```
context.arc(centerX, centerY, radius, startingRadianAngle, endingRadianAngle)
```

endsAngle = startingAngle + 360360== Math.PI2◦

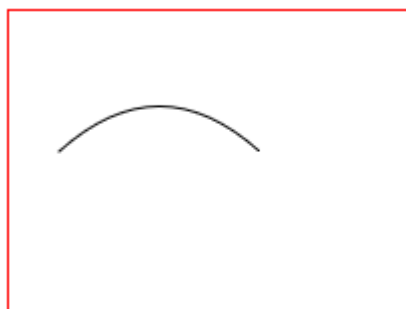


```
context.arc(centerX, centerY, radius, startingRadianAngle, endingRadianAngle)
```

## quadraticCurveTo

```
context.quadraticCurveTo(controlX, controlY, endingX, endingY)
```

◦ ◦

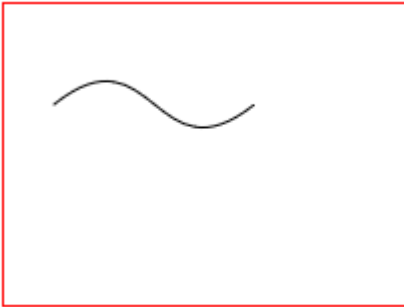


```
context.quadraticCurveTo(controlX, controlY, endingX, endingY)
```

## bezierCurveTo

```
context.bezierCurveTo(control1X, control1Y, control2X, control2Y, endingX, endingY)
```

◦ ◦



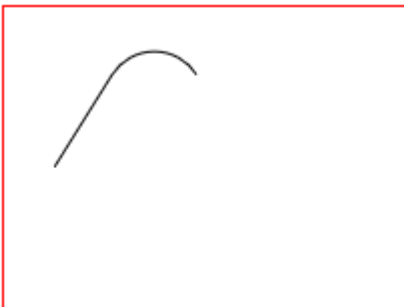
```
context.bezierCurveTo(control1X, control1Y, control2X, control2Y, endingX, endingY)
```

## arcTo

```
context.arcTo(pointX1, pointY1, pointX2, pointY2, radius);
```

◦ Point1Point2◦

◦



```
context.arcTo(pointX1, pointY1, pointX2, pointY2, radius);
```

## rect

```
context.rect(leftX, topY, width, height)
```

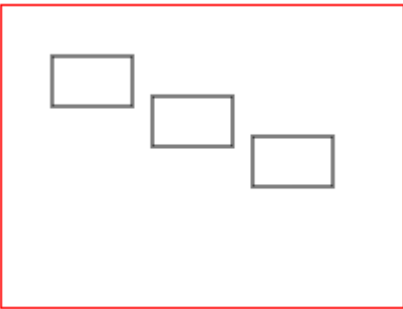
◦



```
context.rect(leftX, topY, width, height)
```

context.rect°

°



```
context.rect(leftX, topY, width, height)
```

## closePath

```
context.closePath()
```

°

2closePath“”°

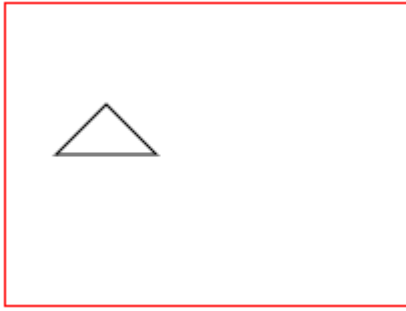
°

context.closePathcontext.beginPath °

closePath - “”beginPath°

closePath°   closePath°





```
context.closePath()
```

## beginPath

```
context.beginPath()
```

◦

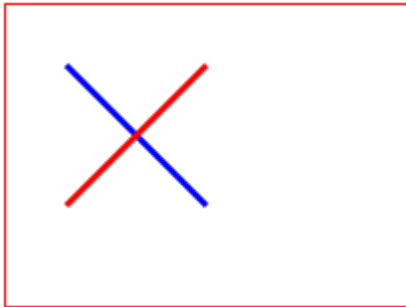
“”== coordinate [0,0]◦

### beginPath

◦ beginPath◦

“X”◦

beginPath◦ “X”◦



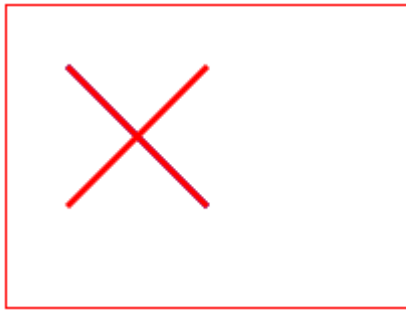
```
context.beginPath()
```

beginPath◦ “X”◦

stroke()◦

beginPath stroke()◦

stroke()◦



```
context.beginPath()
```

## lineCap

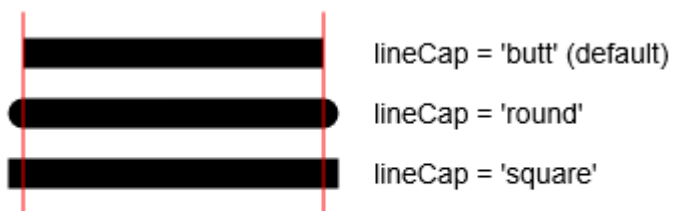
```
context.lineCap=capStyle // butt (default), round, square
```

◦

- **butt** lineCap◦

- ◦

- ◦



butt (default) stays inside line start & end  
round & square extend beyond line start & end

```
context.lineCap=capStyle // butt (default), round, square
```

## lineJoin

```
context.lineJoin=joinStyle // miter (default), round, bevel
```

◦

- ◦

- ◦

- ◦



lineJoin = 'miter' (default)

lineJoin = 'round'

lineJoin = 'bevel'

```
context.lineJoin=joinStyle // miter (default), round, bevel
```

## strokeStyle

```
context.strokeStyle=color
```

◦

color

- **colorCSS** context.strokeStyle='red'
- context.strokeStyle='#FF0000'
- **RGB** context.strokeStyle='rgb(red,green,blue)' ' 0-255◦
- **HSL** context.strokeStyle='hsl(hue,saturation,lightness)' ' 0-3600-100◦
- **HSLA** context.strokeStyle='hsl(hue,saturation,lightness,alpha)' ' 0-3600-100alpha0.00-1.00◦
- 
- context.createLinearGradient
- context.createRadialGradient
- context.createPattern



CSS color



Linear Gradient color



Radial Gradient color



Pattern color

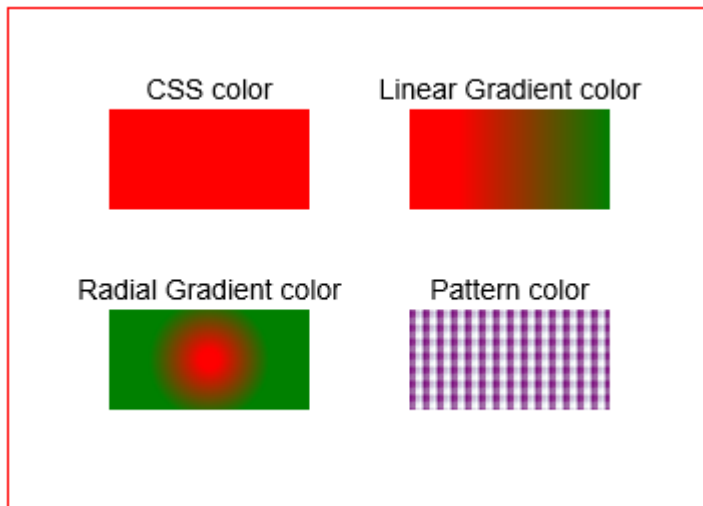
```
context.strokeStyle=color
```

## fillStyle

```
context.fillStyle=color
```

。

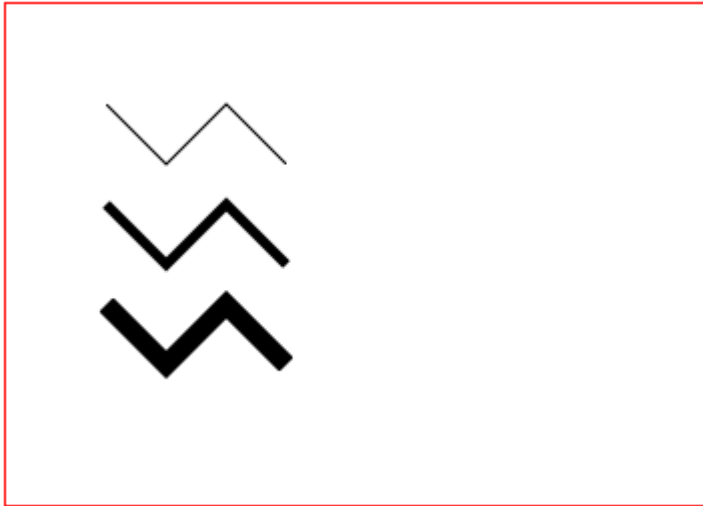
- **colorCSS** `context.fillStyle='red'`
- `context.fillStyle='#FF0000'`
- **RGB** `context.fillStyle='rgb(red,green,blue)'` 0-255。
- **HSL** `context.fillStyle='hsl(hue,saturation,lightness)'` 0-3600-100。
- **HSLA** `context.fillStyle='hsl(hue,saturation,lightness,alpha)'` 0-3600-100alpha0.00-1.00。
- `context.createLinearGradient`
- `context.createRadialGradient`
- `context.createPattern`



```
context.fillStyle=color
```

## lineWidth

```
context.lineWidth=lineWidth
```



```
context.lineWidth=lineWidth
```

## shadowColorshadowBlurshadowOffsetXshadowOffsetY

```
shadowColor = color          // CSS color
shadowBlur = width           // integer blur width
shadowOffsetX = distance     // shadow is moved horizontally by this offset
shadowOffsetY = distance     // shadow is moved vertically by this offset
```

◦  
◦  
◦

- **shadowColor**◦
- **shadowBlur**◦
- **shadowOffsetX**◦ ◦
- **shadowOffsetY**◦ ◦

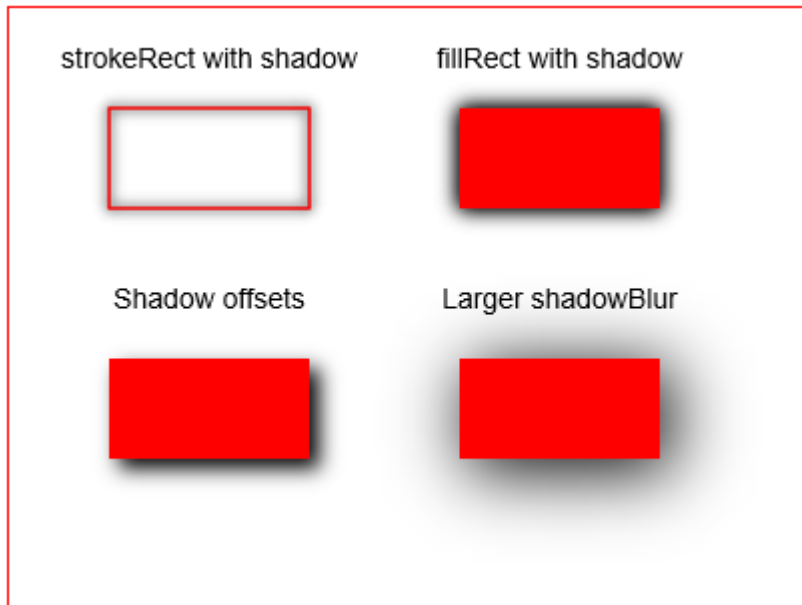
## shadowOffsetXshadowOffsetY

◦ ◦  
◦ ◦  
◦ shadowColor◦

```
shadowColor = color          // CSS color
shadowBlur = width           // integer blur width
shadowOffsetX = distance     // shadow is moved horizontally by this offset
shadowOffsetY = distance     // shadow is moved vertically by this offset
```

◦

◦ “”◦ document.createElement - DOM◦ ◦ ◦ ◦



```
shadowColor = color           // CSS color
shadowBlur = width            // integer blur width
shadowOffsetX = distance      // shadow is moved horizontally by this offset
shadowOffsetY = distance      // shadow is moved vertically by this offset
```

## createLinearGradient

```
var gradient = createLinearGradient( startX, startY, endX, endY )
gradient.addColorStop(gradientPercentPosition, CssColor)
gradient.addColorStop(gradientPercentPosition, CssColor)
[optionally add more color stops to add to the variety of the gradient]
```

◦

strokeStyle/fillStyle◦

strokefillPath◦

1.◦ ◦ var gradient = context.createLinearGradient◦

2. 2◦ gradient.addColorStop◦

- **startXstartY**◦ gradientPercentPosition◦

- **endXendY**◦ gradientPercentPosition◦

- **gradientPercentPosition**0.001.00◦ ◦

- 0.00[startXstartY]◦

- 1.00[endXendY]◦

- “”0.001.000100◦

- **CssColorCSS**。

。

CanvasContext。 JavaScript。 CanvasPath

。 ==。 。

## Canvas

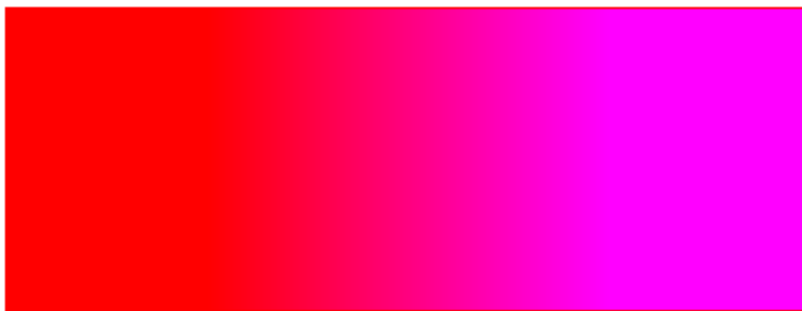
“”。

stroke() fill()。

1. 

```
var gradient = createLinearGradient( startX, startY, endX, endY )
gradient.addColorStop(gradientPercentPosition, CssColor)
gradient.addColorStop(gradientPercentPosition, CssColor)
[optionally add more color stops to add to the variety of the gradient]
```

### 2. Canvas“”



3. stroke() fill() “”。

4. “”“”...。



```
var gradient = createLinearGradient( startX, startY, endX, endY )
gradient.addColorStop(gradientPercentPosition, CssColor)
gradient.addColorStop(gradientPercentPosition, CssColor)
[optionally add more color stops to add to the variety of the gradient]
```

## createRadialGradient

```
var gradient = createRadialGradient(
    centerX1, centerY1, radius1,    // this is the "display" circle
    centerX2, centerY2, radius2    // this is the "light casting" circle
)
gradient.addColorStop(gradientPercentPosition, CssColor)
gradient.addColorStop(gradientPercentPosition, CssColor)
[optionally add more color stops to add to the variety of the gradient]
```

◦ ◦

...

Canvas ◦

Canvas“”◦

- 2“”“”◦
- ◦
- ◦
- ◦

1. ◦ ◦ var gradient = context.radialLinearGradient◦

2. 2◦ gradient.addColorStop◦

- **centerX1centerY1radius1**◦
- **centerX2centerY2radius2**◦
- **gradientPercentPosition0.001.00**◦ ◦
  - 0.00◦
  - 1.00◦
  - “”0.001.000100◦
- **CssColorCSS**◦

CanvasContext◦ JavaScript◦ CanvasPath

◦ ==◦ ◦

**Canvas**

“”◦

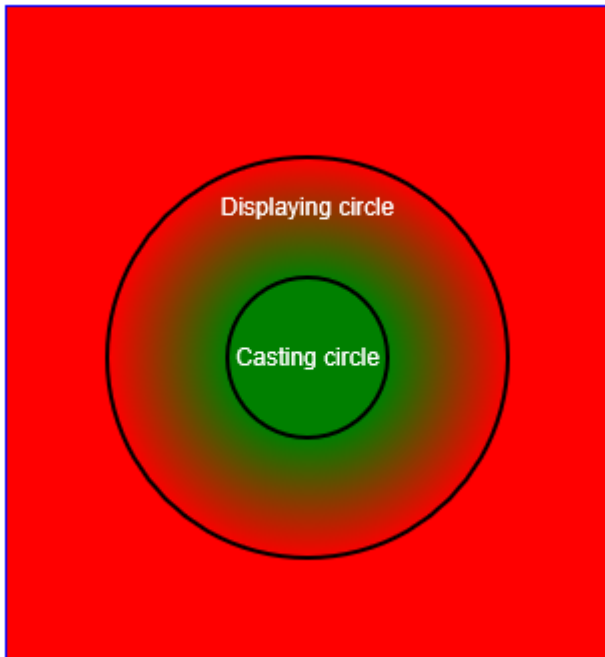
stroke() fill()◦

1. var gradient = createRadialGradient(
 centerX1, centerY1, radius1, // this is the "display" circle
 centerX2, centerY2, radius2 // this is the "light casting" circle
 )



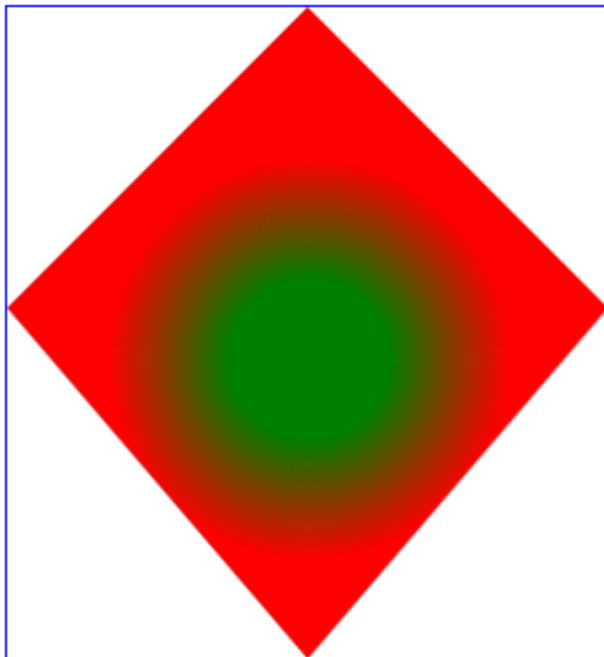
```
gradient.addColorStop(gradientPercentPosition, CssColor)
gradient.addColorStop(gradientPercentPosition, CssColor)
[optionally add more color stops to add to the variety of the gradient]
```

## 2. Canvas“



3. `stroke()` `fill()` “”。

4. “”“” ...。



```
var gradient = createRadialGradient(
    centerX1, centerY1, radius1,    // this is the "display" circle
    centerX2, centerY2, radius2    // this is the "light casting" circle
)
gradient.addColorStop(gradientPercentPosition, CssColor)
```

```
gradient.addColorStop(gradientPercentPosition, CssColor)
[optionally add more color stops to add to the variety of the gradient]
```

## createRadialGradient

W3C HTML5 Canvas.

createRadialGradient W3C createRadialGradient

createRadialGradient

```
createRadialGradient ...0.01.0.
```

```
createRadialGradient(x0, y0, r0, x1, y1, r1)x0y0r0x1 y1r1. . r0r1IndexSizeError.
CanvasGradient.
```

1.  $x_0 = x_1, y_0 = y_1, r_0 = r_1$  . .
2.  $x_\omega = x_1 - x_0\omega + x_0; y_\omega = y_1 - y_0\omega + y_0; r_\omega = r_1 - r_0\omega + r_0$  .
3.  $\omega r_\omega > 0, \omega r_\omega \leq x_\omega y_\omega$  .

## createPattern

```
var pattern = createPattern(imageObject, repeat)
```

.

strokeStyle/fillStyle .

strokefillPath.

- **imageObject**.
  - HTMLImageElement ---imgImage
  - HTMLCanvasElement ---canvas
  - HTMLVideoElement ---
  - ImageBitmap
  - .
- **repeat**imageObjectCSS.
  - " "---
  - "repeat-x"---1
  - "repeat-y"---1
  - " "---

.

## Canvas

“”repeat。

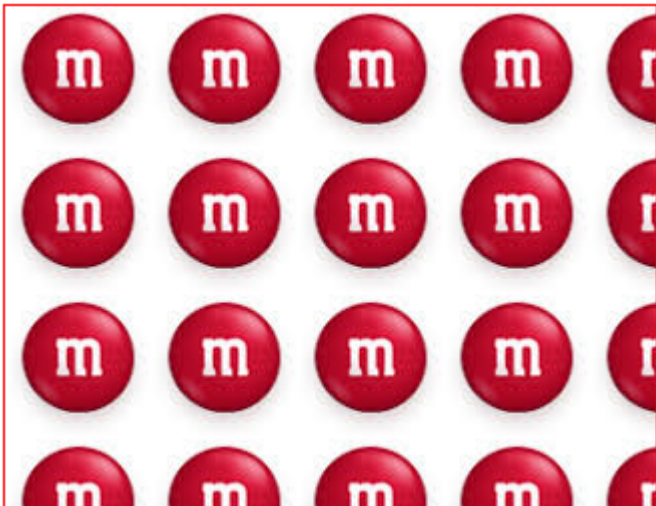
stroke()fill()。

- 1.。 patternimage.onload。



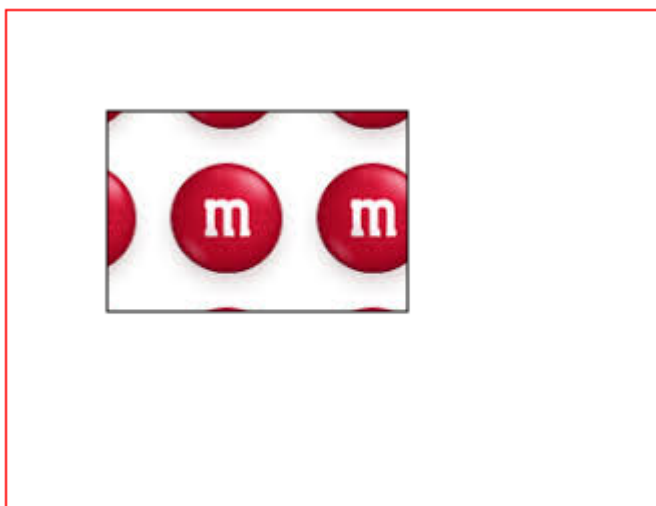
2. `var pattern = createPattern(imageObject,repeat)`

3. Canvas“”



4. stroke()fill() “”。

5. “”“”...。



```
var pattern = createPattern(imageObject, repeat)
```

## stroke

```
context.stroke()
```

context.strokeStyle **Path**

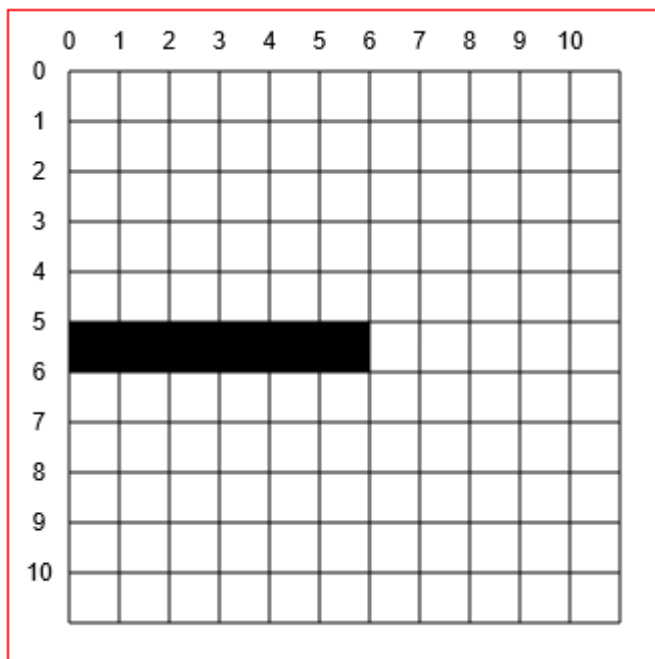
context.stroke context.fill **Path**



[0, 5] [5, 5] **1**

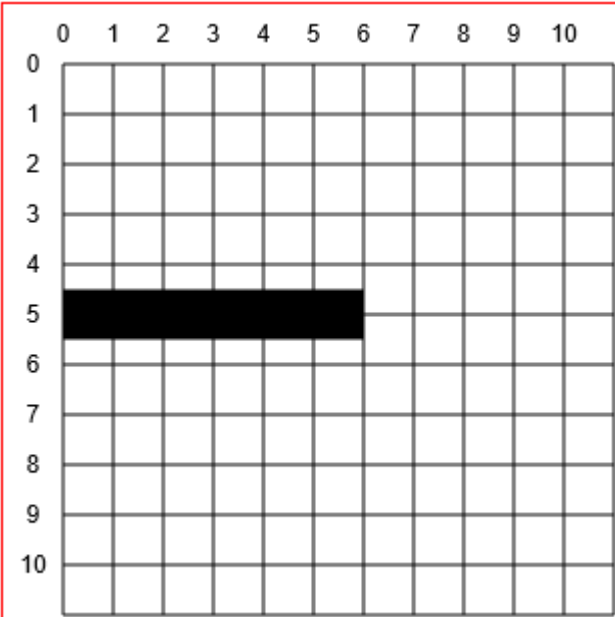
```
context.stroke()
```

$y = 56$

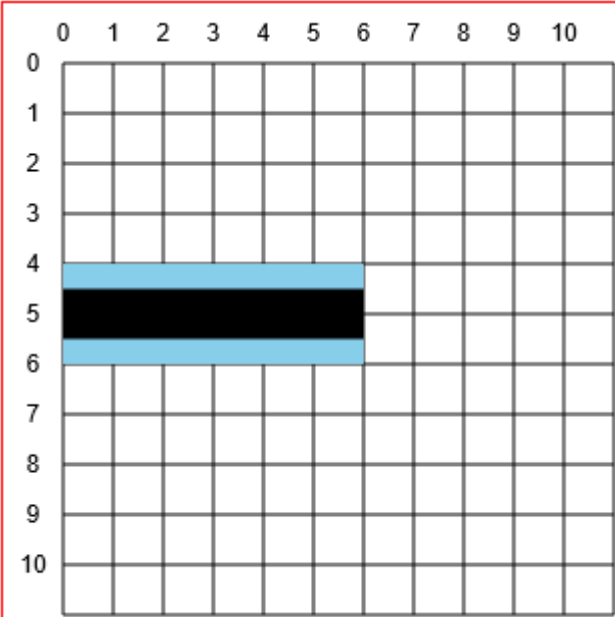


..... Canvas

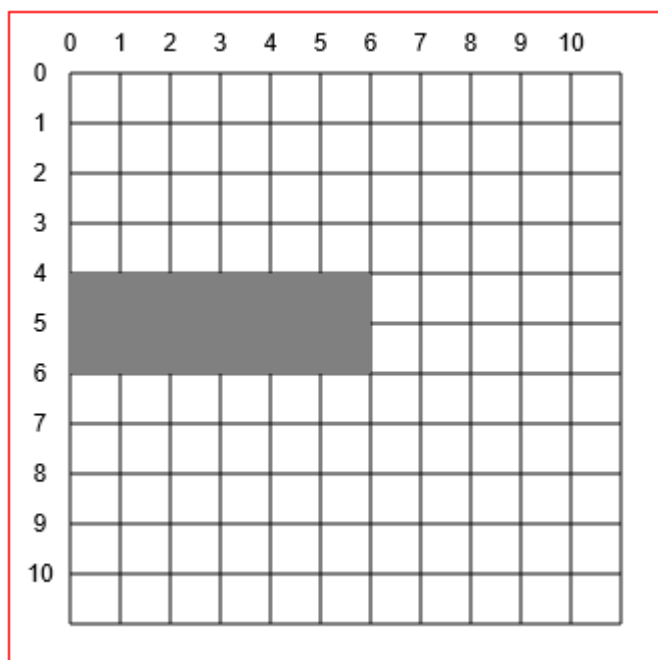
$y == 5.0$  Canvas  $y == 4.5$   $y == 5.5$



.....

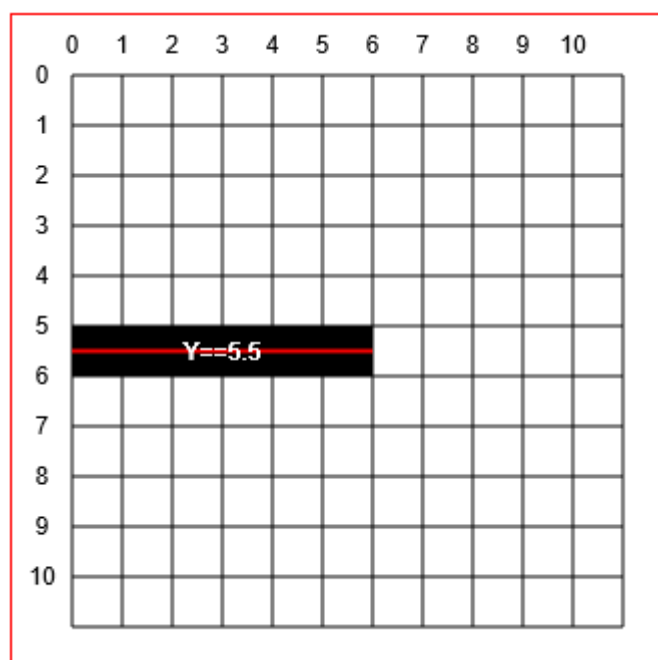


Canvas4.06.02◦ blackblack◦ “”Canvas◦

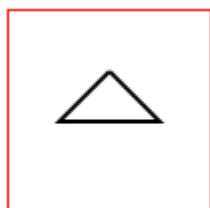


1

```
context.stroke()
```



```
context.stroke()
```



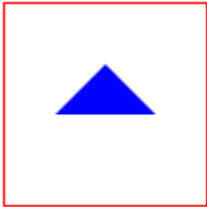
```
context.stroke()
```

```
context.fill()
```

```
context.fillStylePathPath。
```

```
context.fill context.stroke Path。
```

```
context.fill()
```



```
context.fill()
```

## clip

```
context.clip
```

。



```
context.clip
```

<https://riptutorial.com/zh-TW/html5-canvas/topic/3241/-->

# 16:

## Examples

◦ ◦ 2D◦ 2D601000◦

```
// assumes that the canvas context is in ctx and in scope
function drawImageRST(image, x, y, scale, rotation){
    ctx.setTransform(scale, 0, 0, scale, x, y); // set the scale and translation
    ctx.rotate(rotation);                       // add the rotation
    ctx.drawImage(image, -image.width / 2, -image.height / 2); // draw the image offset by
half its width and height
}
```

◦

```
// assumes that the canvas context is in ctx and in scope
function drawImageRST(image, x, y, scale, rotation){
    ctx.setTransform(scale, 0, 0, scale, x, y); // set the scale and translation
    ctx.rotate(rotation);                       // add the rotation
    ctx.drawImage(image, -image.width / 2, -image.height / 2); // draw the image offset by
half its width and height
}
```

◦ ◦

```
// assumes that the canvas context is in ctx and in scope
function drawImageRST(image, x, y, scale, rotation){
    ctx.setTransform(scale, 0, 0, scale, x, y); // set the scale and translation
    ctx.rotate(rotation);                       // add the rotation
    ctx.drawImage(image, -image.width / 2, -image.height / 2); // draw the image offset by
half its width and height
}
```

## alpha

```
// assumes that the canvas context is in ctx and in scope
function drawImageRST(image, x, y, scale, rotation){
    ctx.setTransform(scale, 0, 0, scale, x, y); // set the scale and translation
    ctx.rotate(rotation);                       // add the rotation
    ctx.drawImage(image, -image.width / 2, -image.height / 2); // draw the image offset by
half its width and height
}
```

```
// assumes that the canvas context is in ctx and in scope
function drawImageRST(image, x, y, scale, rotation){
    ctx.setTransform(scale, 0, 0, scale, x, y); // set the scale and translation
    ctx.rotate(rotation);                       // add the rotation
    ctx.drawImage(image, -image.width / 2, -image.height / 2); // draw the image offset by
half its width and height
}
```





1-5/。

1. [0,0]

```
context.translate( shapeCenterX, shapeCenterY );
```

2. `context.translate( shapeCenterX, shapeCenterY );`

3. `context.translate( shapeCenterX, shapeCenterY );`

4. 。

```
context.translate( shapeCenterX, shapeCenterY );
```

5. 1

- 51

```
context.translate( shapeCenterX, shapeCenterY );
```

- 5211-3

```
context.translate( shapeCenterX, shapeCenterY );
```

```
context.translate( shapeCenterX, shapeCenterY );
```

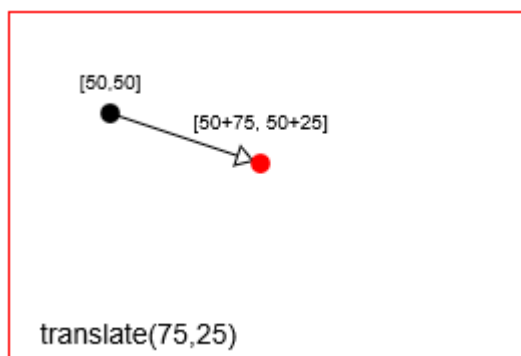
。

- distanceXdistanceY Y。
- radian angle 。
- scalingFactorXscalingFactorY 。

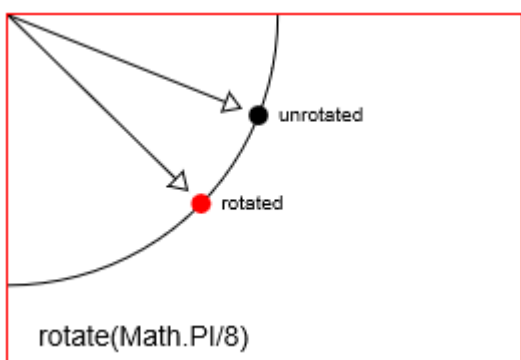
`context.transform`

◦

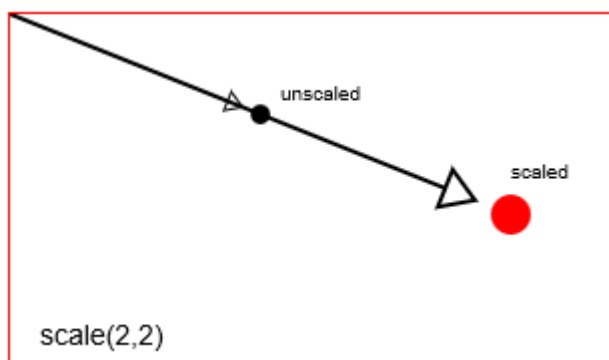
```
context.translate(75,25)==context.translate(75,25)
```



```
context.rotate(Math.PI/8)
```



```
context.scale(2,2)
```



Canvas◦

- context.translate canvas [0,0]◦
- context.rotate◦
- context.scale◦ xy every  $x*=scaleX$  every  $y*=scaleY$ ◦

Canvas◦ ==◦

```
// reset context transformations to the default (untransformed) state
context.setTransform(1,0,0,1,0,0);
```

Canvascontext.translate context.rotatecontext.scale。

Canvas。

- context.transformCanvas
- translate, rotate & scaleCanvas
- context.setTransformCanvas
- *Canvas* - 。

————

- 
- ◦
- context.isPointInPath◦
  - ◦
- ◦
  - .translate, .rotate, .scale◦
- ◦

————

“”

context.translate context.rotate context.scale◦ ◦

- translate rotate scale◦ ◦
- setContextTransform◦ ◦
- resetContextTransform==◦
- getTransformedPoint◦
- getScreenPoint◦
- getMatrix◦

```
var TransformationMatrix=( function(){
  // private
  var self;
  var m=[1,0,0,1,0,0];
  var reset=function(){ var m=[1,0,0,1,0,0]; }
  var multiply=function(mat){
    var m0=m[0]*mat[0]+m[2]*mat[1];
    var m1=m[1]*mat[0]+m[3]*mat[1];
    var m2=m[0]*mat[2]+m[2]*mat[3];
```

```

        var m3=m[1]*mat[2]+m[3]*mat[3];
        var m4=m[0]*mat[4]+m[2]*mat[5]+m[4];
        var m5=m[1]*mat[4]+m[3]*mat[5]+m[5];
        m=[m0,m1,m2,m3,m4,m5];
    }
    var screenPoint=function(transformedX,transformedY){
        // invert
        var d =1/(m[0]*m[3]-m[1]*m[2]);
        im=[ m[3]*d, -m[1]*d, -m[2]*d, m[0]*d, d*(m[2]*m[5]-m[3]*m[4]), d*(m[1]*m[4]-
m[0]*m[5]) ];
        // point
        return({
            x:transformedX*im[0]+transformedY*im[2]+im[4],
            y:transformedX*im[1]+transformedY*im[3]+im[5]
        });
    }
    var transformedPoint=function(screenX,screenY){
        return({
            x:screenX*m[0] + screenY*m[2] + m[4],
            y:screenX*m[1] + screenY*m[3] + m[5]
        });
    }
    // public
    function TransformationMatrix(){
        self=this;
    }
    // shared methods
    TransformationMatrix.prototype.translate=function(x,y){
        var mat=[ 1, 0, 0, 1, x, y ];
        multiply(mat);
    };
    TransformationMatrix.prototype.rotate=function(rAngle){
        var c = Math.cos(rAngle);
        var s = Math.sin(rAngle);
        var mat=[ c, s, -s, c, 0, 0 ];
        multiply(mat);
    };
    TransformationMatrix.prototype.scale=function(x,y){
        var mat=[ x, 0, 0, y, 0, 0 ];
        multiply(mat);
    };
    TransformationMatrix.prototype.skew=function(radianX,radianY){
        var mat=[ 1, Math.tan(radianY), Math.tan(radianX), 1, 0, 0 ];
        multiply(mat);
    };
    TransformationMatrix.prototype.reset=function(){
        reset();
    }
    TransformationMatrix.prototype.setContextTransform=function(ctx){
        ctx.setTransform(m[0],m[1],m[2],m[3],m[4],m[5]);
    }
    TransformationMatrix.prototype.resetContextTransform=function(ctx){
        ctx.setTransform(1,0,0,1,0,0);
    }
    TransformationMatrix.prototype.getTransformedPoint=function(screenX,screenY){
        return(transformedPoint(screenX,screenY));
    }
    TransformationMatrix.prototype.getScreenPoint=function(transformedX,transformedY){
        return(screenPoint(transformedX,transformedY));
    }
    TransformationMatrix.prototype.getMatrix=function(){

```

```

        var clone=[m[0],m[1],m[2],m[3],m[4],m[5]];
        return(clone);
    }
    // return public
    return(TransformationMatrix);
}) ();

```

## Transformation Matrix“Class”

- ==。
- 。
- 。

```

var TransformationMatrix=( function(){
    // private
    var self;
    var m=[1,0,0,1,0,0];
    var reset=function(){ var m=[1,0,0,1,0,0]; }
    var multiply=function(mat) {
        var m0=m[0]*mat[0]+m[2]*mat[1];
        var m1=m[1]*mat[0]+m[3]*mat[1];
        var m2=m[0]*mat[2]+m[2]*mat[3];
        var m3=m[1]*mat[2]+m[3]*mat[3];
        var m4=m[0]*mat[4]+m[2]*mat[5]+m[4];
        var m5=m[1]*mat[4]+m[3]*mat[5]+m[5];
        m=[m0,m1,m2,m3,m4,m5];
    }
    var screenPoint=function(transformedX,transformedY) {
        // invert
        var d =1/(m[0]*m[3]-m[1]*m[2]);
        im=[ m[3]*d, -m[1]*d, -m[2]*d, m[0]*d, d*(m[2]*m[5]-m[3]*m[4]), d*(m[1]*m[4]-
m[0]*m[5]) ];
        // point
        return({
            x:transformedX*im[0]+transformedY*im[2]+im[4],
            y:transformedX*im[1]+transformedY*im[3]+im[5]
        });
    }
    var transformedPoint=function(screenX,screenY) {
        return({
            x:screenX*m[0] + screenY*m[2] + m[4],
            y:screenX*m[1] + screenY*m[3] + m[5]
        });
    }
    // public
    function TransformationMatrix(){
        self=this;
    }
    // shared methods
    TransformationMatrix.prototype.translate=function(x,y) {
        var mat=[ 1, 0, 0, 1, x, y ];
        multiply(mat);
    };
    TransformationMatrix.prototype.rotate=function(rAngle) {
        var c = Math.cos(rAngle);
        var s = Math.sin(rAngle);
        var mat=[ c, s, -s, c, 0, 0 ];

```

```

        multiply(mat);
    };
    TransformationMatrix.prototype.scale=function(x,y){
        var mat=[ x, 0, 0, y, 0, 0 ];
        multiply(mat);
    };
    TransformationMatrix.prototype.skew=function(radianX,radianY){
        var mat=[ 1, Math.tan(radianY), Math.tan(radianX), 1, 0, 0 ];
        multiply(mat);
    };
    TransformationMatrix.prototype.reset=function(){
        reset();
    }
    TransformationMatrix.prototype.setContextTransform=function(ctx){
        ctx.setTransform(m[0],m[1],m[2],m[3],m[4],m[5]);
    }
    TransformationMatrix.prototype.resetContextTransform=function(ctx){
        ctx.setTransform(1,0,0,1,0,0);
    }
    TransformationMatrix.prototype.getTransformedPoint=function(screenX,screenY){
        return(transformedPoint(screenX,screenY));
    }
    TransformationMatrix.prototype.getScreenPoint=function(transformedX,transformedY){
        return(screenPoint(transformedX,transformedY));
    }
    TransformationMatrix.prototype.getMatrix=function(){
        var clone=[m[0],m[1],m[2],m[3],m[4],m[5]];
        return(clone);
    }
    // return public
    return(TransformationMatrix);
}) ();

```

<https://riptutorial.com/zh-TW/html5-canvas/topic/5494/>

# 17:

## Examples

“”。

- ImageObjectCanvas。
- Canvas。
- 。



```
var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
document.body.appendChild(canvas);
canvas.style.background='navy';
canvas.style.border='1px solid red;';

// Always(!) wait for your images to fully load before trying to drawImage them!
var img=new Image();
img.onload=start;
// put your img.src here...
img.src='http://i.stack.imgur.com/bXaB6.png';
function start(){
    ctx.drawImage(img,20,20);
    var sticker=stickerEffect(img,5);
    ctx.drawImage(sticker, 150,20);
}

function stickerEffect(img,grow){
    var canvas1=document.createElement("canvas");
    var ctx1=canvas1.getContext("2d");
    var canvas2=document.createElement("canvas");
    var ctx2=canvas2.getContext("2d");
    canvas1.width=canvas2.width=img.width+grow*2;
    canvas1.height=canvas2.height=img.height+grow*2;
    ctx1.drawImage(img,grow,grow);
    ctx2.shadowColor='white';
    ctx2.shadowBlur=2;
    for(var i=0;i<grow;i++){
        ctx2.drawImage(canvas1,0,0);
        ctx1.drawImage(canvas2,0,0);
    }
    ctx2.shadowColor='rgba(0,0,0,0)';
    ctx2.drawImage(img,grow,grow);
    return(canvas2);
}
```

◦

context.shadowColor◦

```
// start shadowing
context.shadowColor='black';

... render some shadowed drawings ...

// turn off shadowing.
context.shadowColor='rgba(0,0,0,0)';
```

-

◦

- **Canvas** var memoryCanvas = document.createElement('canvas') ...
- context.drawImage(memoryCanvas,x,y)



```
var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
var cw=canvas.width;
var ch=canvas.height;
canvas.style.border='1px solid red;';
document.body.appendChild(canvas);

// Always(!) use "img.onload" to give your image time to
//      fully load before you try drawing it to the Canvas!
var img=new Image();
img.onload=start;
// Put your own img.src here
img.src="http://i.stack.imgur.com/hYFNe.png";
function start(){
    ctx.drawImage(img,0,20);
    var cached=cacheShadowedImage(img,'black',5,3,3);
    for(var i=0;i<5;i++){
        ctx.drawImage(cached,i*(img.width+10),80);
    }
}

function cacheShadowedImage(img,shadowcolor,blur){
    var c=document.createElement('canvas');
    var cctx=c.getContext('2d');
    c.width=img.width+blur*2+2;
    c.height=img.height+blur*2+2;
```



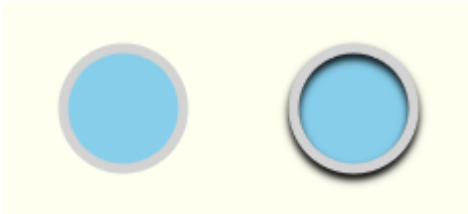
```

cctx.shadowColor=shadowcolor;
cctx.shadowBlur=blur;
cctx.drawImage(img,blur+1,blur+1);
return(c);
}

```

## 3D。

“”



```

var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
document.body.appendChild(canvas);

ctx.fillStyle='skyblue';
ctx.strokeStyle='lightgray';
ctx.lineWidth=5;

// without shadow
ctx.beginPath();
ctx.arc(60,60,30,0,Math.PI*2);
ctx.closePath();
ctx.fill();
ctx.stroke();

// with shadow
ctx.shadowColor='black';
ctx.shadowBlur=4;
ctx.shadowOffsetY=3;
ctx.beginPath();
ctx.arc(175,60,30,0,Math.PI*2);
ctx.closePath();
ctx.fill();
ctx.stroke();
// stop the shadowing
ctx.shadowColor='rgba(0,0,0,0)';

```

## CanvasCSSinner-shadow。

- 。
- 。

。

\_\_\_\_\_



destination-in° °

1. ° ° - °

2. **destination-in** °

3. ° ° ° *context.lineWidth*°

```
var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
document.body.appendChild(canvas);

// draw an opaque shape -- here we use a rounded rectangle
defineRoundedRect(30,30,100,75,10);

// set shadowing
ctx.shadowColor='black';
ctx.shadowBlur=10;

// stroke the shadowed rounded rectangle
ctx.lineWidth=4;
ctx.stroke();

// set compositing to erase everything outside the stroke
ctx.globalCompositeOperation='destination-in';
ctx.fill();

// always clean up -- set compositing back to default
ctx.globalCompositeOperation='source-over';

function defineRoundedRect(x,y,width,height,radius) {
  ctx.beginPath();
  ctx.moveTo(x + radius, y);
  ctx.lineTo(x + width - radius, y);
  ctx.quadraticCurveTo(x + width, y, x + width, y + radius);
  ctx.lineTo(x + width, y + height - radius);
  ctx.quadraticCurveTo(x + width, y + height, x + width - radius, y + height);
  ctx.lineTo(x + radius, y + height);
  ctx.quadraticCurveTo(x, y + height, x, y + height - radius);
  ctx.lineTo(x, y + radius);
  ctx.quadraticCurveTo(x, y, x + radius, y);
  ctx.closePath();
}
```



1-3destination-over。

4. destination-over。

5. context.shadowColor。

6. 。。

```
var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
document.body.appendChild(canvas);

// draw an opaque shape -- here we use a rounded rectangle
defineRoundedRect(30,30,100,75,10);

// set shadowing
ctx.shadowColor='black';
ctx.shadowBlur=10;

// stroke the shadowed rounded rectangle
ctx.lineWidth=4;
ctx.stroke();

// set compositing to erase everything outside the stroke
ctx.globalCompositeOperation='destination-in';
ctx.fill();

// always clean up -- set compositing back to default
ctx.globalCompositeOperation='source-over';

function defineRoundedRect(x,y,width,height,radius) {
    ctx.beginPath();
    ctx.moveTo(x + radius, y);
    ctx.lineTo(x + width - radius, y);
    ctx.quadraticCurveTo(x + width, y, x + width, y + radius);
    ctx.lineTo(x + width, y + height - radius);
    ctx.quadraticCurveTo(x + width, y + height, x + width - radius, y + height);
    ctx.lineTo(x + radius, y + height);
    ctx.quadraticCurveTo(x, y + height, x, y + height - radius);
    ctx.lineTo(x, y + radius);
    ctx.quadraticCurveTo(x, y, x + radius, y);
    ctx.closePath();
}
```



shadowOffsetX。

```
var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
document.body.appendChild(canvas);
```

```

// draw an opaque shape -- here we use a rounded rectangle
defineRoundedRect(30,30,100,75,10);

// set shadowing
ctx.shadowColor='black';
ctx.shadowBlur=10;

// stroke the shadowed rounded rectangle
ctx.lineWidth=4;
ctx.stroke();

// set compositing to erase everything outside the stroke
ctx.globalCompositeOperation='destination-in';
ctx.fill();

// always clean up -- set compositing back to default
ctx.globalCompositeOperation='source-over';

function defineRoundedRect(x,y,width,height,radius) {
    ctx.beginPath();
    ctx.moveTo(x + radius, y);
    ctx.lineTo(x + width - radius, y);
    ctx.quadraticCurveTo(x + width, y, x + width, y + radius);
    ctx.lineTo(x + width, y + height - radius);
    ctx.quadraticCurveTo(x + width, y + height, x + width - radius, y + height);
    ctx.lineTo(x + radius, y + height);
    ctx.quadraticCurveTo(x, y + height, x, y + height - radius);
    ctx.lineTo(x, y + radius);
    ctx.quadraticCurveTo(x, y, x + radius, y);
    ctx.closePath();
}

```

<https://riptutorial.com/zh-TW/html5-canvas/topic/5322/>

# 18:

## Examples

javascript

```
var canvas;    // Global canvas reference
var ctx;       // Global 2D context reference
// Creates a canvas
function createCanvas () {
    const canvas = document.createElement("canvas");
    canvas.style.position = "absolute"; // Set the style
    canvas.style.left     = "0px";      // Position in top left
    canvas.style.top      = "0px";
    canvas.style.zIndex   = 1;
    document.body.appendChild(canvas); // Add to document
    return canvas;
}
// Resizes canvas. Will create a canvas if it does not exist
function sizeCanvas () {
    if (canvas === undefined) {           // Check for global canvas reference
        canvas = createCanvas();         // Create a new canvas element
        ctx = canvas.getContext("2d");    // Get the 2D context
    }
    canvas.width  = innerWidth;           // Set the canvas resolution to fill the page
    canvas.height = innerHeight;
}
// Removes the canvas
function removeCanvas () {
    if (canvas !== undefined) {           // Make sure there is something to remove
        removeEventListener("resize", sizeCanvas); // Remove resize event
        document.body.removeChild(canvas);         // Remove the canvas from the DOM
        ctx = undefined;                          // Dereference the context
        canvas = undefined;                        // Dereference the canvas
    }
}

// Add the resize listener
addEventListener("resize", sizeCanvas);
// Call sizeCanvas to create and set the canvas resolution
sizeCanvas();
// ctx and canvas are now available for use.
```

removeCanvas()

jsfiddle

Canvas. - .

```
// variables holding the current canvas offset position
//     relative to the window
var offsetX,offsetY;

// a function to recalculate the canvas offsets
function reOffset(){
```

```

    var BB=canvas.getBoundingClientRect();
    offsetX=BB.left;
    offsetY=BB.top;
}

// listen for window resizing (and scrolling) events
//      and then recalculate the canvas offsets
window.onscroll=function(e){ reOffset(); }
window.onresize=function(e){ reOffset(); }

// example usage of the offsets in a mouse handler
function handleMouseUp(e){
    // use offsetX & offsetY to get the correct mouse position
    mouseX=parseInt(e.clientX-offsetX);
    mouseY=parseInt(e.clientY-offsetY);
    // ...
}

```

- 
- ◦ requestAnimationFrame◦

60fps◦ DOM◦ GC◦

---

## Debounced resize

- 

```

// Assume canvas is in scope
addEventListener("resize", debouncedResize );

// debounce timeout handle
var debounceTimeoutHandle;

// The debounce time in ms (1/1000th second)
const DEBOUNCE_TIME = 100;

// Resize function
function debouncedResize () {
    clearTimeout(debounceTimeoutHandle); // Clears any pending debounce events

    // Schedule a canvas resize
    debounceTimeoutHandle = setTimeout(resizeCanvas, DEBOUNCE_TIME);
}

// canvas resize function
function resizeCanvas () { ... resize and redraw ... }

```

100ms◦ ◦ ◦

- ◦ ◦
  -
-

◦

## KISSK EEP imple tupid ◦

◦ ◦ **resize**◦

◦ ◦

```
// Assume canvas is in scope
addEventListener("resize", debouncedResize );

// debounce timeout handle
var debounceTimeoutHandle;

// The debounce time in ms (1/1000th second)
const DEBOUNCE_TIME = 100;

// Resize function
function debouncedResize () {
    clearTimeout(debounceTimeoutHandle); // Clears any pending debounce events

    // Schedule a canvas resize
    debounceTimeoutHandle = setTimeout(resizeCanvas, DEBOUNCE_TIME);
}

// canvas resize function
function resizeCanvas () { ... resize and redraw ... }
```

<https://riptutorial.com/zh-TW/html5-canvas/topic/5495/>

S. No		Contributors
1	html5-canvas	<a href="#">almcd</a> , <a href="#">Blindman67</a> , <a href="#">Community</a> , <a href="#">Daniel Dees</a> , <a href="#">Kaiido</a> , <a href="#">markE</a> , <a href="#">ndugger</a> , <a href="#">Spencer Wieczorek</a> , <a href="#">Stephen Leppik</a> , <a href="#">user2314737</a>
2	“getImageData” “putImageData”	<a href="#">markE</a>
3		<a href="#">Blindman67</a> , <a href="#">markE</a>
4		<a href="#">Blindman67</a> , <a href="#">markE</a>
5		<a href="#">Blindman67</a> , <a href="#">Kaiido</a> , <a href="#">markE</a>
6		<a href="#">Blindman67</a> , <a href="#">markE</a>
7		<a href="#">markE</a>
8		<a href="#">Blindman67</a> , <a href="#">markE</a>
9		<a href="#">Blindman67</a> , <a href="#">Bobby</a> , <a href="#">Kaiido</a>
10		<a href="#">almcd</a> , <a href="#">Blindman67</a> , <a href="#">markE</a> , <a href="#">RamenChef</a>
11		<a href="#">Blindman67</a> , <a href="#">markE</a>
12		<a href="#">bjanes</a> , <a href="#">Blindman67</a> , <a href="#">Kaiido</a> , <a href="#">markE</a> , <a href="#">Mike C</a> , <a href="#">Ronen Ness</a>
13		<a href="#">Blindman67</a> , <a href="#">markE</a>
14		<a href="#">Blindman67</a> , <a href="#">markE</a>
15		<a href="#">AgataB</a> , <a href="#">markE</a>
16		<a href="#">Blindman67</a> , <a href="#">markE</a>
17		<a href="#">markE</a>
18		<a href="#">Blindman67</a> , <a href="#">markE</a> , <a href="#">mnoronha</a>