12강. 캔버스 [3] : 드로잉 응용

1. 도형 합성 및 변환

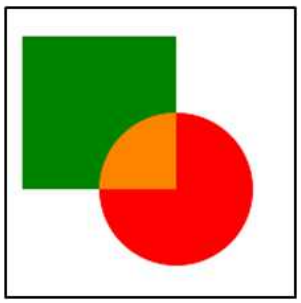
1.1 도형 합성

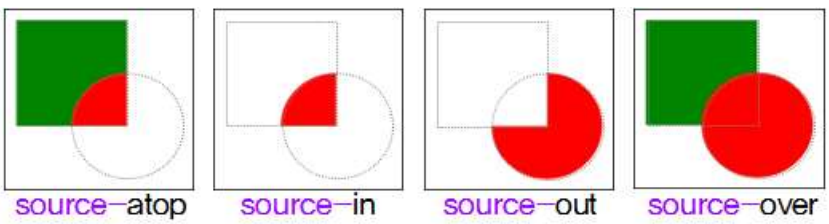
1.1.1 globalCompositeOperation 속성 값

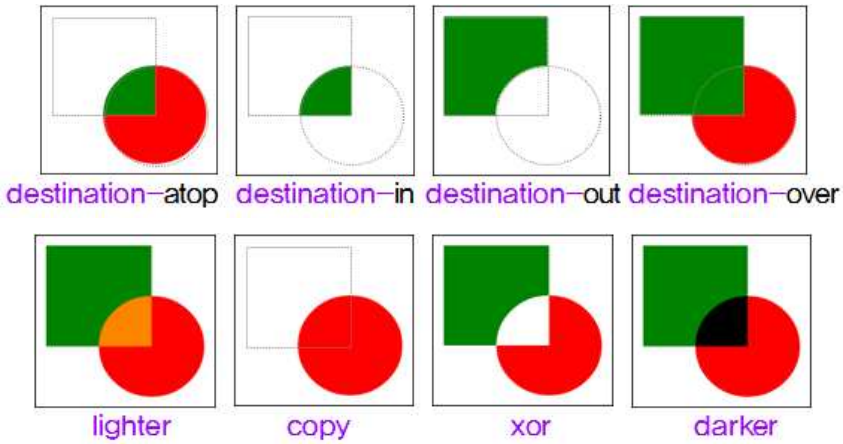
- globalCompositeOperation 속성을 설명하면 다양한 기본 합성 동작을 지정할 수 있다. 이 속성을 이용함으로써 도형이 그려진 순서와 상관없이 겹쳐지는 부분에 대한 처리가 가능하다.

예제

```
<!DOCTYPE html><html><head>
  <script type="text/javascript">
    function attr_values()
    {
      var canvas = document.getElementById('canvas');
      context = canvas.getContext('2d');
      context.beginPath();
      context.fillStyle = 'green';
      context.fillRect(10, 10, 100, 100);
      context.globalCompositeOperation = 'lighter';
      context.fillStyle = 'red';
      context.arc(110, 110, 50, 0, 360*Math.PI/180, true);
      context.fill();
    }
  </script>
</head>
<body onload="attr_values();">
  <canvas id="canvas" width="170" height="170" style="border:solid 1px #000000">
    canvas 사용하기  </canvas>
</body>
</html>
```







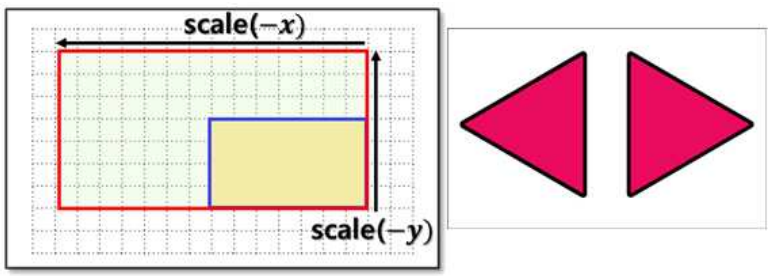
1.2 도형 변환

- 캔버스의 좌표계의 이동, 회전 및 확대/축소 기능을 통한 도형의 변환을 제공한다.
- translate() 메서드는 좌표 공간의 시작점을 (x,y)로 이동한다.
- rotate() 메서드는 좌표 공간을 시계 방향으로 회전한다.
- scale() 메서드는 좌표 공간을 수평/수직 방향으로 좌표 공간을 확대/축소 한다.

예제 도형변환

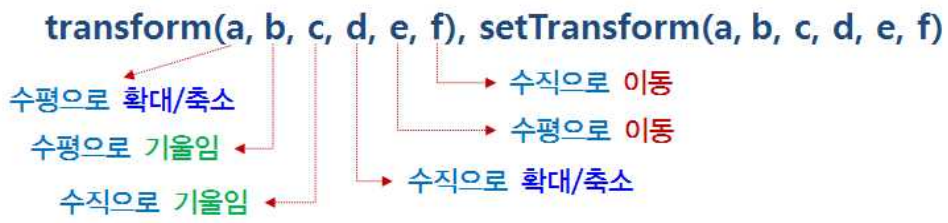
```
<!DOCTYPE html> <html><head>
<script>
function polygon(context, x, y, radius, sides, startAngle, anticlockwise) {
    if (sides < 3) return; //3각형 이하는 그리지 않도록 한다.
    var degree = (Math.PI * 2)/sides;
    degree = anticlockwise ? -degree : degree;
    context.save(); //드로잉 상태를 저정한다.
    context.translate(x,y);       //드로잉 좌표 공간을 다각형 중심좌표로 이동한다.
    context.rotate(startAngle); //시작 각도를 중심으로 그리도록 하기 위하여 회전한다.
    context.moveTo(radius,0);    //다각형의 시작 위치로 이동한다.
    for (var i = 1; i < sides; i++) { //면의 수 만큼 루프를 반복한다
        context.lineTo(radius*Math.cos(degree*i),radius*Math.sin(degree*i));
    }
    context.closePath(); //패스를 닫는다.
    context.restore(); //기존 드로잉 상태를 복구한다.
}
function ScaleMirror() { //그린 도형을 서로 마주보게 만든다.
    var canvas = document.getElementById("myCanvas");
    var context = canvas.getContext("2d");
    context.lineWidth = 10;
    context.lineJoin = "round"
    context.fillStyle = "rgba(227,11,93,1.0)";
```

```
    //polygon 메서드를 사용하여 삼각형을 그린다.
    context.beginPath();
    polygon(context,120, 120, 100, 3, (90 * Math.PI/2), false);
    context.stroke();
    context.fill();
    context.translate(canvas.width, 0); //캔버스의 중심으로 중앙으로 이동시킨다.
    context.scale(-1, 1);//그리는 방향을 수평 반대 방향으로 그리도록 한다.
    //polygon 메서드를 사용하여 동일한 조건으로 삼각형을 하나 더 그린다.
    polygon(context,120, 120, 100, 3, (90 * Math.PI/2), false);
    context.stroke();
    context.fill();
}
</script>
</head>
<body onload="ScaleMirror();">
<canvas id="myCanvas" width="400" height="250" style="border: 10px inset #aaa">
캔버스 연습</canvas>
</body></html>
```



1.2.1 사용자 정의 변환

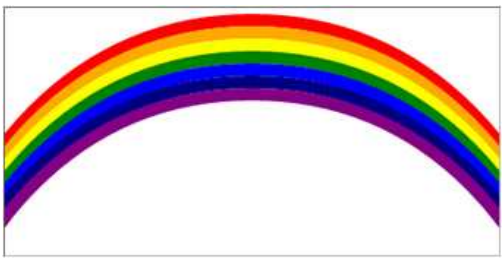
- 캔버스 컨텍스트에서는 변환 행렬을 직접 조작할 수 있도록 transform() 메서드와 setTransform() 메서드를 제공한다.



- transform() 메서드가 지정된 인수로 행렬로 현재 변환 행렬에 적용시키기 때문에 연속해서 이 메서드를 호출하면 변환 행렬 값이 누적된다. 반면에 setTransform() 메서드는 위에 언급한 바와 같이 현재 변환 행렬을 원래 값인 단위 행렬로 재 설정한 다음 transform() 메서드를 호출한다. 따라서 이 메서드를 호출할 때마다 매번 새로운 변환 행렬이 적용된다.

예제 사용자 정의변환

```
<!DOCTYPE html><html><head>
 <script>
function Transformation() {
var canvas = document.getElementById('myCanvas');
var context = canvas.getContext('2d');
var colors = new Array("red", "orange", "yellow", "green", "blue", "navy", "purple");
context.lineWidth = 10;
for (var i = 0; i <colors.length; i++) { //호를 반복해서 그리도록 한다.
 context.transform (1, 0, 0, 1, 0, 10); // 아래쪽으로 10픽셀만큼 이동하는 변환 행렬
 context.strokeStyle = colors[i]; //색상을 적용시킨다.
 context.beginPath();
 context.arc(200, 250, 250, 0, Math.PI, true);
 context.stroke();
 }
}
</script>
</head>
<body onload="Transformation();">
<canvas id="myCanvas" width="400" height="200" style="border: 10px inset #aaa">
캔버스 연습</canvas>
</body></html>
```



2. 텍스트 그리기

2.1 텍스트 테두리 및 채우기

- context.fillText(text, x, y [, maxWidth])는 지정된 위치에 색이 채워진 텍스트를 삽입한다. maxWidth가 지정된 경우, 텍스트는 maxWidth 폭 크기에 맞게 조정된 텍스트를 채운다.
- context.strokeText(text, x, y [, maxWidth])는 지정된 위치에 테두리만 있는 텍스트를 삽입한다.
- context.measureText(text)는 현재 글꼴에서 주어진 텍스트의 폭을 반환한다.

예제 – 텍스트 테두리 및 채우기

```
<!DOCTYPE html><html><head>
<script>
   function fillText() {
   var canvas = document.getElementById('myCanvas');
   var context = canvas.getContext('2d');
```

```
    gradient = context.createLinearGradient(0, 0, 600, 0);
    gradient.addColorStop("0", "magenta");
    gradient.addColorStop(".25", "blue");
    gradient.addColorStop(".50", "green");
    gradient.addColorStop(".75", "yellow");
    gradient.addColorStop("1.0", "red");
    context.fillStyle = gradient;
    context.font = "50pt 궁서체";
    context.strokeStyle = "red";
    context.lineWidth = 2;

    context.strokeText("컴퓨터과학 HTML5", 20, 65);
    context.fillText("컴퓨터과학 HTML5", 20, 140);
    context.fillStyle = 'blue';
    context.strokeText("컴퓨터과학 HTML5", 20, 210);
    context.fillText("컴퓨터과학 HTML5", 20, 210);
}
</script>
</head>
<body onload="fillText();">
   <canvas id="myCanvas" width="600" height="400" >캔버스 연습</canvas>
</body>
</html>
```

2.2 글꼴 설정 및 텍스트 배치

- context.font는 현재 글꼴의 설정 값을 반환한다. font 속성 값은 글자 스타일, 글자 크기, 글꼴 등으로 지정할 수 있다.
  - context.font = "italic 15pt 굴림체"

2.2.1 textAlign 속성

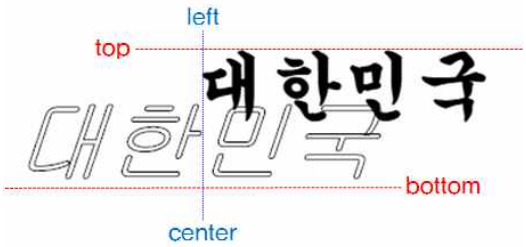
- 현재 텍스트의 수평에 대한 맞춤(정렬) 설정 값을 반환한다. 텍스트의 맞춤을 변경하기 위하여 설정될 수 있고, 기본 값은 start이다. 지정 할 수 있는 값은 left, right, center, start, end이다.

2.2.2 textBaseline 속성

- 현재 텍스트의 수직에 대한 기준선을 지정한다. 수직 위치의 기준 정렬을 변경하기 위하여 설정될 수 있고, 기본 값은 alphabetic이다. 지정 할 수 있는 값은 top, hanging, middle, alphabetic, ideographic, bottom이다.

예제 – 텍스트 그리기

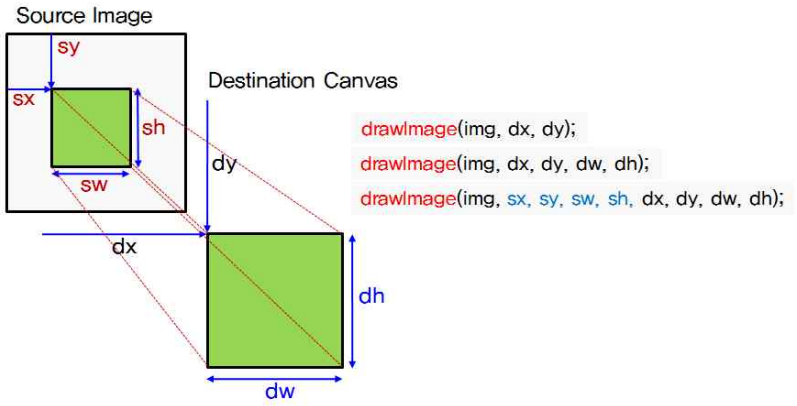
```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <script type="text/javascript">
    function arcTo()    {
     var canvas = document.getElementById("myCanvas");
     var context = canvas.getContext("2d");
    context.font = "bold 40pt 궁서체";
    context.textAlign = 'left';
    context.textBaseline = 'top';
    context.fillText("대한민국", 200, 100);
    context.font = "italic 50pt 굴림체";
    context.textAlign = 'center';
    context.textBaseline = 'bottom';
    context.strokeText("대한민국", 200, 200);
    }
  </script>
</head>
<body onload="arcTo();">
  <canvas id="myCanvas" width="500" height="400" >canvas 사용하기 </canvas>
</body>
</html>
```



3 이미지 처리

3.1 이미지 삽입 관련 메서드

- 이미지를 그리기 위해서는 drawImage() 메서드를 사용하고, 이미지의 일부나 전체, 또는 확대 및 축소하여 캔버스 어디든지 그릴 수 있다.



3.1.1 이미지 삽입

- context.drawImage(이미지, dx, dy)는 지정한 이미지를 원래 크기로 삽입한다.
- context.drawImage(이미지, dx, dy, dw, dh)는 이미지를 지정한 크기로 삽입한다.
- context.drawImage(이미지, sx, sy, sw, sh, dx, dy, dw, dh)는 이미지의 일부분을 잘라내어 삽입한다.

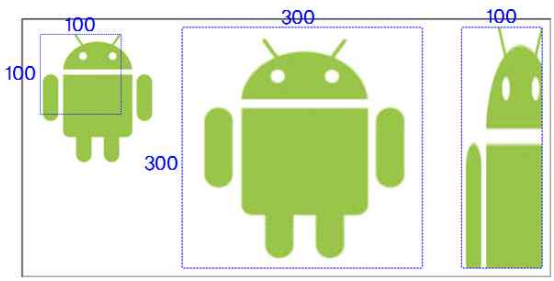
예제

```
<!DOCTYPE html><html><head>
<script>
function drawImage() {
        var canvas = document.getElementById('myCanvas');
        var context = canvas.getContext('2d');
        var img = new Image();
        img.src = 'image.jpg';
        img.onload = function(e) { context.drawImage(img, 10, 10, 380, 280); }
}
</script>
</head>
<body onload="drawImage();">
<canvas id="myCanvas" width="400" height="300" style="border: 10px inset #aaa">
캔버스 연습</canvas>
</body></html>
```

3.1.2 이미지 삽입

예제

```
<!DOCTYPE html><html>
<head>
<script>
function drawImage() {
var canvas = document.getElementById('myCanvas');
var context = canvas.getContext('2d');
var img = new Image();
img.src = 'android.jpg';
img.onload = function(e) {
        context.drawImage(img, 10, 10);
        context.drawImage(img, 200, 10, 300, 300);
        context.drawImage(img, 10, 10, 100, 100, 550, 10, 100, 300);
   }
        context.beginPath();
        context.setLineDash([3,1]);
        context.lineWidth =2;
        context.strokeStyle="blue";
        context.strokeRect(200, 10, 300, 300);
        context.strokeRect(550, 10, 100, 300);
}
    </script>
</head>
<body onload="drawImage();">
<canvas id="myCanvas" width="660" height="320" style="border: 2px inset #aaa">
캔버스 연습</canvas>
</body></html>
```



3.2 이미지 조작

- 이미지 데이터 메서드를 사용하면 이미지의 각 픽셀에 접근해서 다양한 조작을 할 수 있다. createImageData() 메서드를 사용하여 비어 있는 이미지에 객체를 생성할 수 있

다. getImageData() 메서드를 사용하여 이미지 픽셀에 접근하고 putImageData() 메서드를 사용하여 픽셀을 다시 이미지에 넣을 수 있다.

- imagedata= context.createImageData(sw, sh), imagedata = context.createImageData(이미지데이터)는 모든 픽셀이 투명한 검은색으로 초기화된 ImageData 객체를 생성한다.
- imagedata = context.getImageData(sx, sy, sw, sh)는 캔버스 영역에 지정된 사각형 영역에 대한 이미지를 포함한 ImageData 객체를 반환한다.
- context.putImageData(이미지데이터, dx, dy)는 캔버스에 지정된 ImageData 객체의 데이터를 그린다.

3.2.1 이미지 데이터 처리

- 다음은 ImageData 객체를 생성하는 방법을 나타낸 것이다.

```
imageCopy = context.createImageData( canvas.width, canvas.height )
```

- 다음은 캔버스 영역에 지정한 영역에서(0,0) 캔버스 크기만큼(canvas.width, canvas.height)의 이미지 데이터를 imageData 객체에 반환하는 방법을 나타낸 것으로, 캔버스에서 이미지 데이터를 가져옴으로써, 이미지 조작을 자유자재로 할 수 있다.

```
imageData = context.getImageData( 0, 0, canvas.width, canvas.height )
```

- 생성된 이미지 배열 데이터를 복사하거나 조작하기 위해서는 다음과 같이 반복문을 통해서 처리하게 된다.

```
for (var i=0; i < imageData.data.length; i++)
    imageCopy.data[i] = imageData.data[i];
```

- 가져온 이미지 데이터의 배열을 조작한다.

```
for (var i=3; i < imageData.data.length-4; i+=4)
    imageData.data[i] = imageData.data[i] / 2;

for (var i=0; i < imageData.data.length-4; i+=4) {
average = ( imageData.data[i]+imageData.data[i+1]+imageData.data[i+2] ) / 3;
  imageData.data[i] = imageData.data[i+1] = imageData.data[i+2] = average;
}
```

- 캔버스에서 가져온 이미지 데이터를 복사한 후 캔버스 영역에 출력하도록 한 방법을 나타낸 것이다.

```
context.putImageData ( imageCopy, 0, 0 )
```

3.2.2 이미지 데이터 처리

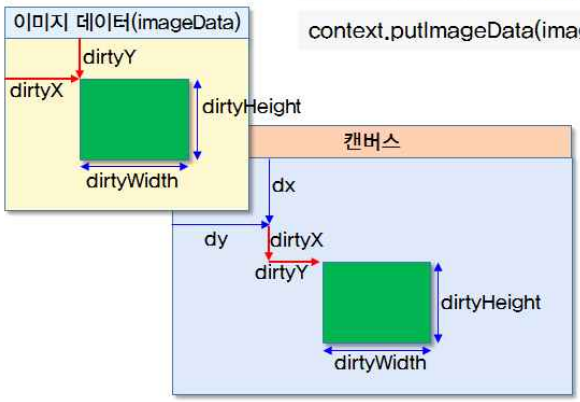
- putImageData() 메서드는 이미지 데이터의 원하는 부분만 캔버스에 출력할 수 있는 기능을 제공하고 있다. 추가적인 옵션 dirty*를 지정한다면 이미지 데이터의 원하는

부분만을 캔버스에 출력할 수 있다.

context.putImageData(이미지데이터, dx, dy [, dirtyX, dirtyY, dirtyWidth, dirtyHeight])

context.putImageData(imageData, 0, 0)

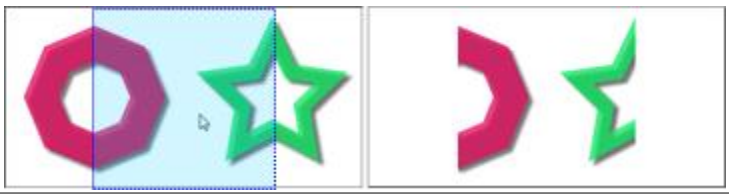
context.putImageData(imageData, 0, 0, 100, 50, 200, 200)



예제 이미지 데이터 처리

```
<!DOCTYPE html><html><head>
 <script type="text/javascript" src="Canvas_Lab.js"></script>
 <script>
function copyImage() {
var canvas = document.getElementById('myCanvas');
var context = canvas.getContext('2d');
//그리는 도형의 그림자 효과를 설정한다.
context.shadowBlur = 5; context.shadowOffsetX = 5;
context.shadowOffsetY = 5;  context.shadowColor = 'rgba(0,0,0,0.75)';
//다각형을 그린다.
context.beginPath();
context.strokeStyle = "black"
drawPolygon(context, 100, 100, 80, 8, Math.PI/2, false); //팔각형 외부를 그린다
drawPolygon(context, 100, 100, 40, 8, Math.PI/2, true); //팔각형 내부를 그린다
context.fillStyle="rgba(227,11,93,0.75)";
context.fill();
//별을 그린다.
context.beginPath();
drawStar(context, 300, 100, 90, false); //바깥쪽 별을 그린다.
drawStar(context, 300, 100, 50, true);  //안쪽 별을 그린다.
context.fillStyle="rgba(11,227,93,0.75)";
context.fill();
canvas.addEventListener("click", function (){ //캔버스의  마우스 클릭시
var canvas2 = document.getElementById("yourCanvas");
var context2 = canvas2.getContext("2d");
```

```
//캔버스에 그려진 이미지 데이터를 가져온다.
var imageBuffer = context.getImageData(0, 0, canvas.width, canvas.height);
//이미지 데이터를 원하는 영역만 다른 캔버스에 출력한다.
context2.putImageData(imageBuffer, 0, 0, 100, 0, 200, 200); }, false);
}
</script>
</head>
<body onload="copyImage();">
<canvas id="myCanvas" width="400" height="200" style="border: 10px inset #aaa">
캔버스 연습</canvas>
<canvas id="yourCanvas" width="400" height="200" style="border: 10px outset
#aaa">캔버스 연습</canvas>
<p>왼쪽 캔버스를 마우스로 클릭하면 오른쪽 캔버스에 이미지를 복사합니다.</p>
</body>
</html>
```



3.3 이미지 데이터 저장

- url = canvas.toDataURL(type, quality)은 캔버스에 그려진 내용을 URL 문자열로 반환해주는 기능을 제공한다. type은 이미지의 종류 (“image/jpeg”, “image/png”)를 나타내고 quality는 이미지의 품질을 나타내는 것으로 0.0 ~ 1.0(손실이 없는 가장 좋은 품질) 사이의 실수 값을 지정해야 한다.

3.4 클리핑 영역 지정

- 캔버스 영역에서 드로잉을 제한하도록 해주는 패스로 정의된 특수한 영역을 말한다. 컨텍스트가 초기화될 때, 클리핑 영역의 크기와 캔버스의 크기는 동일하게 된다. 클리핑 영역은 현재의 클리핑 영역과 현재 패스에서 그려진 영역이 겹치는 부분을 계산하여 새로운 클리핑 영역으로 생성한다.
- 클링핑 영역은 원하는 클리핑 모양의 패스를 지정한 다음, clip() 메서드를 호출하면 된다.
- 클리핑 영역을 설정한 후 clearRect() 메서드를 호출하면 지워지는 영역이 클리핑 영역으로 제한된다.

4. 애니메이션

4.1 기존 방법의 문제

- 애니메이션은 setTimeout() 메서드나 setInterval() 메서드를 사용한다. setTimeout() 메서드는 특정 시간에 처리할 함수를 한 번만 호출한다. setInterval() 메서드는 특정 시간마다 처리할 함수를 반복적으로 호출한다.
- 이러한 메서드들의 문제는 애니메이션을 업데이트하기 위한 최적의 주기가 무엇인지 모르기 때문에 개발자는 가장 간단한 방법으로 최소한의 시간으로 고정시킨다. 이러한 방법들은 과도한 그리기로 인한 프레임 손실, CPU주기가 낭비되는 등의 문제가 발생한다.
- window 객체의 requestAnimationFrame() 및 cancelAnimationFrame() 메서드 사용하도록 한다.

4.2 스크립트 기반 애니메이션용 타이밍 컨트롤

- 캔버스에서는 requestAnimationFrame() 및 cancelAnimationFrame() 메서드를 사용하여 애니메이션을 구현한다. 이 메서드는 브라우저가 페이지 디스플레이를 업데이트해야 할 때만 사용자 애플리케이션에 알리도록 할 수 있으므로 프레임 손실 문제를 해결할 수 있다.
- requestAnimationFrame() 메서드는 다음 애니메이션 프레임을 그릴 준비가 되면 브라우저에게 특정 콜백을 호출할 수 있도록 요청한다. 다음 애니메이션을 그릴 시간을 알 필요가 없는 장점이 있다.
  - requestId = window.requestAnimationFrame()
- cancelAnimationFrame() 메서드는 애니메이션 프레임을 업데이트하도록 이전에 예약되어 있는 콜백 요청을 취소하는데 사용된다.
  - window.cancelAnimationFrame(requestId)

예제 캔버스에서 애니메이션 구현

```
<!DOCTYPE html><html ><head>
<script>
    function drawImage() {
    var canvas = document.getElementById('myCanvas');
    var context = canvas.getContext('2d');
    var requestId = 0;
    var x =  25, y = 25, rectWidth = 50, speed = 5;
    context.fillStyle="rgba(11,227,93,0.75)";
    context.fillRect(x, y, rectWidth, rectWidth);
function draw() {
            context.clearRect(0, 0, canvas.width, canvas.height);
            context.fillRect(x, y, rectWidth, rectWidth);
            }
```

```
function animate() {
        requestId = window.requestAnimationFrame(animate);
        x += speed;
        if(x <= 0 || x >= (canvas.width - rectWidth)){ speed = -speed; }
        draw();
        }
canvas.addEventListener("click", function (){ //캔버스의 이벤트리스너에서 마우스 클릭
        if(requestId > 0) { //애니메이션이 동작중일 때
        window.cancelAnimationFrame(requestId); //애니메이션을 정지한다..
        requestId = 0;  //핸들 번호를 초기화 시킨다.
        } else {  //애니메이션이 정지 상태이면 다시 시작한다.
        requestId = window.requestAnimationFrame(animate);
        }
        }, false);
        }
    </script>
</head>
<body onload="drawImage();">
<canvas id="myCanvas" width="500" height="100" style="border: 10px inset #aaa">
캔버스 연습</canvas>
</body></html>
```