HTML5プロフェッショナル認定試験
レベル2ポイント解説無料セミナー

株式会社クリーク・アンド・リバー社 認定講師

高井 歩

本日の内容

- 試験概要
- Canvas (2D)
  - 概要
  - コンテキスト
  - 基本図形描画
  - テキスト描画
  - 変形
  - エフェクト
  - イメージデータ
- SVG
  - SVGの特徴
  - Canvasとの違い

HTML5プロフェッショナル認定資格とは

- 次世代のWebプロフェッショナルのスキルの向上に貢献するために、HTML5を活用したWebページやWebアプリケーションなどのデザイン、設計、構築に関する体系だった知識とスキルを備えたHTML5のプロフェッショナルを中立的な立場で公平かつ厳正に認定する資格制度です。

- Webデザイナー、Webプログラマー、スマートフォンアプリ開発者、サーバーサイドエンジニアなどの、Web開発プロジェクトやWebサービスに関わるあらゆるプロフェッショナルが対象です。

- 多くの企業が推進する次世代Web言語の認定資格として、HTML5のプロフェッショナルのスキルの向上に役立ちます。
また、企業内や研修機関での『技術力を担保する客観的基準』としても活用できます。

二つのレベル

HTML5 Level.1

マルチデバイスに対応したWebコンテンツを
HTML5を使ってデザイン・作成できる。

対象

- Webデザイナー
- グラフィックデザイナー
- フロントエンドプログラマー
- HTMLコーダー
- Webディレクター
- Webシステム開発者
- スマートフォンアプリ開発者
- サーバーサイドエンジニア

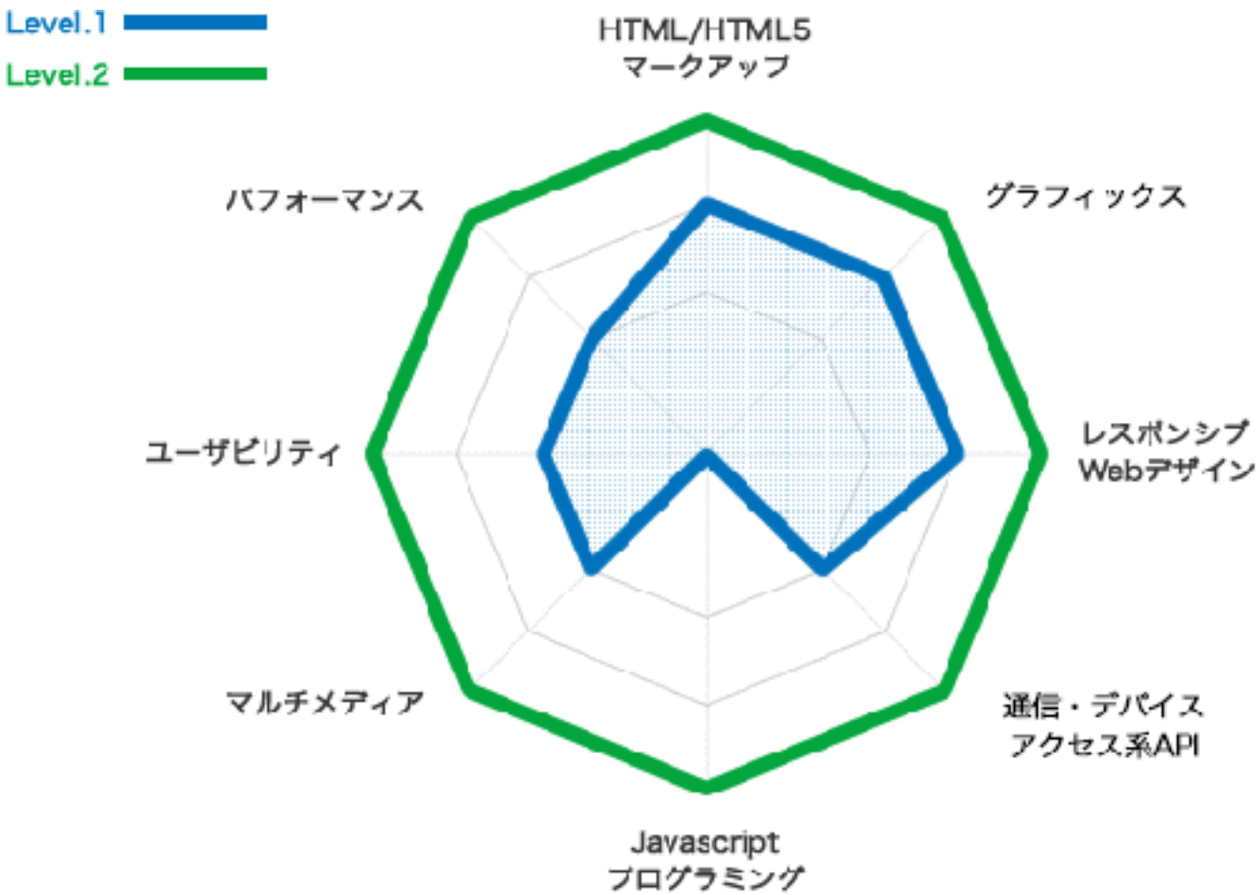
HTML5 Level.2

最新のAPIを駆使したWebアプリケーションを設計・開発できる。

対象

- Webデザイナー
- フロントエンドプログラマー
- HTMLコーダー
- Webディレクター
- Webシステム開発者
- スマートフォンアプリ開発者
- サーバーサイドエンジニア

レベル1とレベル2の資格体系



HTML/HTML5マークアップ

HTML5に関するタグの用途、構造の組み立て方に関する技術

グラフィックス

JavascriptやCSSなどを用いて、動的にグラフィックスを生成したりアニメーションを実現したりする技術

レスポンシブWebデザイン

一つのソースで、スマートフォンなどの様々なデバイスの画面サイズに対応させるための技術

通信・デバイスアクセス系API

JavaScriptからクラウドと通信をして情報の送受信を行ったり、センサーなどのデバイスにアクセスしたりする技術

Javascriptプログラミング

Javascriptを使って、動的なWebコンテンツを作成する技術

マルチメディア

３D・動画・音声ファイルなどのマルチメディアコンテンツの表示・再生に関する技術

ユーザビリティ

JavaScriptやCSSなどを用いて、デザイン仕様に沿った見やすい表示や操作しやすいコンテンツを作成するための技術

パフォーマンス

ストレージや並列処理を使ってコンテンツを効率良く高速に動作させたり、オフラインでも動作する仕組みを作るための技術

レベル1とレベル2の資格体系

ベーシックレベル
HTML5プロフェッショナル向け

所要時間：90分
試験問題数：約60問
受験料：￥15,000 （税別）
認定条件： HTML5 レベル1試験に合格すること
認定の有意性の期限：5年間

認定名： HTML5 Level1 (Markup Professional)
試験名： HTML5 Level1 Exam

この資格の認定者は、下記のスキルと知識を持つWebプロフェッショナルであることを証明できます。

- **HTML5を使ってWebコンテンツを作成することができる。**
- **ユーザー体験を考慮したWEBコンテンツを設計・作成することができる。**
- **スマートフォンや組み込み機器など、ブラウザが利用可能な様々なデバイスに対応したコンテンツを制作できる。**
- **HTML5で何ができるか、どういった技術を使うべきかの広範囲の基礎知識を有する。**

アドバンストレベル
HTML5プロフェッショナル向け

所要時間：90分
試験問題数：40〜45問
受験料：￥15,000(税別)
認定条件：HTML5 レベル2試験に合格し、かつ有意なHTML5レベル1認定を保有していること。
認定の有意性の期限：5年間

認定名： HTML5 Level2 (Application Development Professional)
試験名： HTML5 Level2 Exam

この資格の認定者は、下記のスキルと知識を持つWebプロフェッショナルであることを証明できます。

- **動的に動作させて高いユーザビリティを実現するリッチユーザインターフェイスアプリケーションを作成することができる。**
- **マルチデバイスに対応し高パフォーマンスで動作する動的コンテンツを作成することができる。**
- **システム間連携を行いリアルタイムな情報を提供するアプリケーションを作成することがでる。**
- **スマートフォンなどでネイティブアプリに近い機能を組み込んだ先端のWebアプリケーションに近い機能を組み込んだ先端のWebアプリケーションを開発することができる。**
- **APIのセキュリティモデルを理解したうえで開発することができる。**

レベル2の出題構成(1)

主題	項目	重要度
JavaScript	JavaScript文法	10
WebブラウザにおけるJavaScript API	イベント	8
	ドキュメントオブジェクト/DOM	6
	ウィンドウオブジェクト	8
	Selectors API	7
	History API	**7**
	テスト・デバッグ	6
グラフィックス・アニメーション	**Canvas(2D)**	**8**
	SVG	**2**
	Timing control for script-based animation	2
マルチメディア	メディア要素のAPI	5
ストレージ	Web Storage	7
	Indexed Database API	5
	File API	5
	バイナリーデータ	4

レベル2の出題構成(2)

主題	項目	重要度
通信	Web Socket	5
	XMLHttpRequest	5
	Server-Sent Event	1
デバイスアクセス	Geolocation API	5
	Device Orientation	1
パフォーマンスとオフライン	Web Workers	5
	High Resolution Time	2
	オフラインアプリケーションAPI	3
	Page Visibility	2
	Navigation Timing	1
セキュリティモデル	クロスオリジン制約とCORS	4
	セキュリティモデルとSSLの関係	4

学習方法

- **参考書**
- **サンプル問題**
- **出題範囲を確認**
  - 説明できない用語が無いようにする。
- **自分でサンプルを作って確かめる。**
  - 処理の順番などを確認する。(初期化->処理->後片付け)
  - Webブラウザ毎に動作が異なることがあるので注意。
  - Webサーバの有無で動作が異なることがあるので注意。

2.3.1Canvas(2D)

知識問題 コードリーティング問題

Canvas(2D)の概要

- **Canvas**は元々はAppleの独自実装だった機能をHTML5に取りこんだ

- それまではWebブラウザ上で画像を作成、修正するにはFlashなどが必要だった

- Canvasを**JavaScriptから操作**することで、柔軟な画像処理が出来るようになった

Canvasの特徴と提供機能

Canvas(2D)では、

- 直線、矩形、円弧、パスなどの図形の描画
- テキスト描画
- 変形
- エフェクト
- 画像表示
- ビットマップデータへの直接アクセス

などの機能を提供している

Canvasが利用可能な条件

- Internet Explorer 9以降、その他主なWebブラウザでは対応済み
- Internet Explorer 6～8は**JavaScriptのライブラリを読み込むことでエミュレート**できる
- ページに**canvas要素が存在する**

```
<canvas id="cv" width="300" height="300" />
```

コンテキスト

- Canvas要素をJavaScriptから操作して直接描画を行なうのではなく、描画専用のオブジェクトである**コンテキスト**を使用して描画する

- コンテキストは線の色や塗り潰しの色、フォントなど描画に必要な様々な情報を保持している

- コンテキストは2D,3Dの用途に合せて使い分ける

2Dコンテキストの概要

- Canvas要素から描画に必要なコンテキスト(CanvasRenderingContext2Dオブジェクト)を取得するには、**getContext()**メソッドを使用する

- canvas.**getContext("2d")**

- 現状では、"2d"以外に3Dの描画技術であるWebGLを使用する"webgl"を指定できる(出題範囲外)

- "webgl"ではWebGLRenderingContext オブジェクトが取得できる(出題範囲外)

描画状態の遷移

コンテキストには描画状態として、

- 変形につかう変換行列
- クリッピングパス
- 線の色,線の幅,線の端の形,線の接続点の形
- ドロップシャドウの設定
- 塗り潰しの色,透明度
- 合成方法
- フォント,文字寄せ,ベースライン

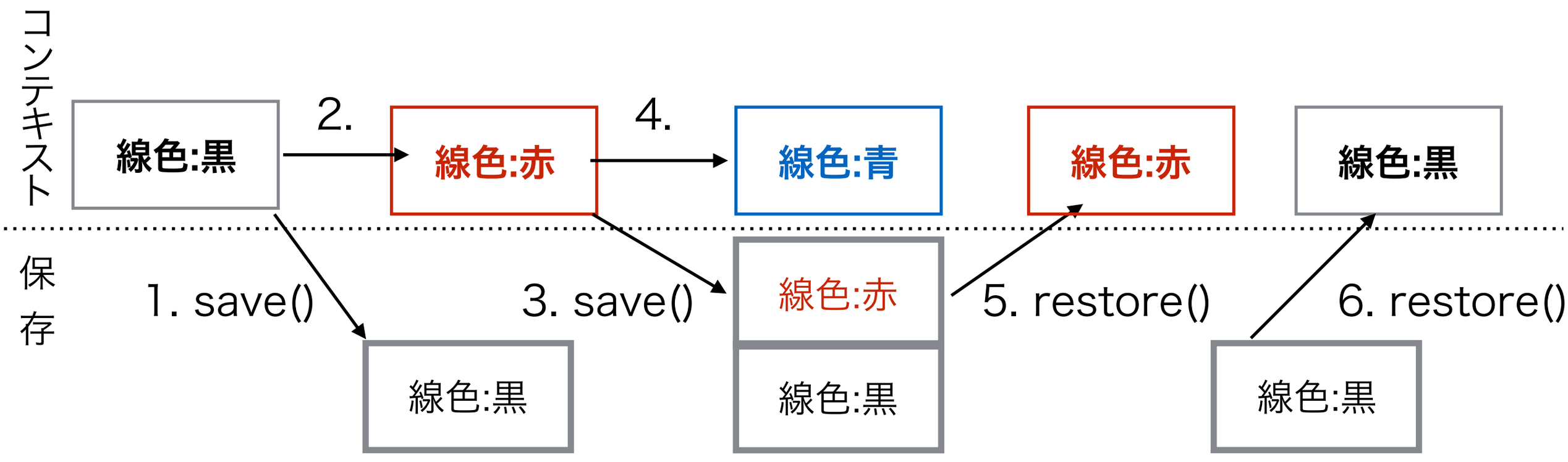
のような情報が保存されている

描画する内容に合わせてこれらの情報を書き換えていく

描画状態の保存と復元する方法

- 描画する色や線の太さ、変形などを一時的に変更したい場合などに、現在の状態をcontext.**save()**メソッドで保存し、context.**restore()** メソッドで元に戻せる

- 保存した描画状態は**後入れ先出し**の**スタック形式**で保存されるため、複数の状態を保持できる



基本図形描画

- 曲線や直線を繋いでパスを作り描画する機能や、使用頻度の高い矩形や円弧の描画については専用のメソッドを用意している

- パスを使う場合、**beginPath()**メソッドでパスを開始し、**moveTo()**,**lineTo()**,**bezierCurveTo()**などのメソッドでパスを構成する要素を追加する

- パスが完成したら、オープンパスであれば**stroke()**,クローズパスであれば**closePath()**のあと**stroke()**または**fill()**を呼び出して描画を行なう

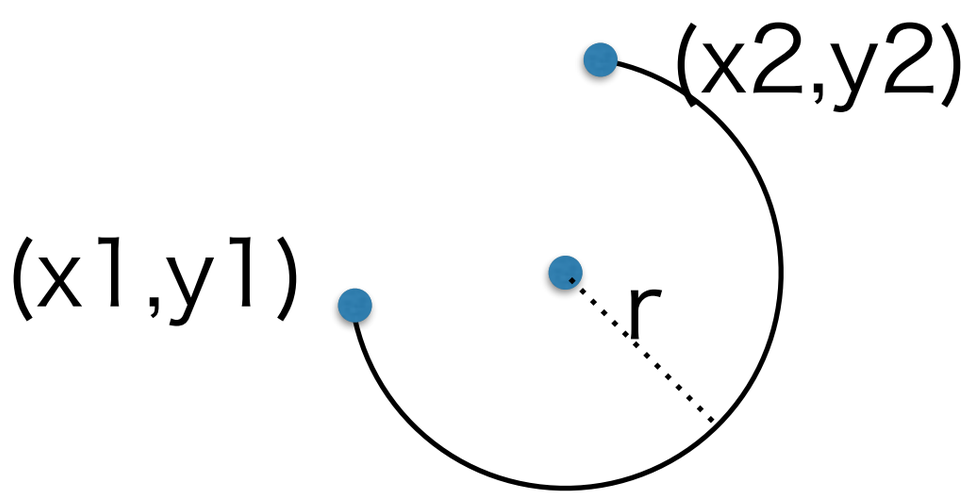
線

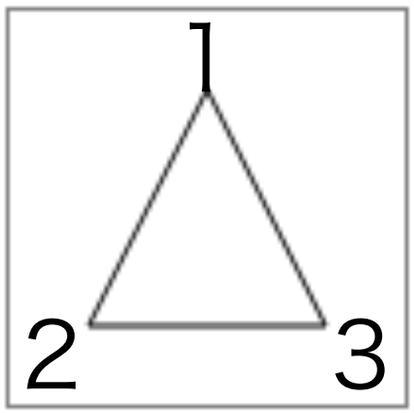
- 線をパスに追加するには、**moveTo()**メソッドで始点に移動し、**lineTo()**や**arcTo()**,**bezierCurveTo()**メソッドで次の点までのパスを追加していく

- **moveTo(x , y)**

- **lineTo(x , y)**

- **arcTo(x1 , y1 , x2 , y2 , r)**

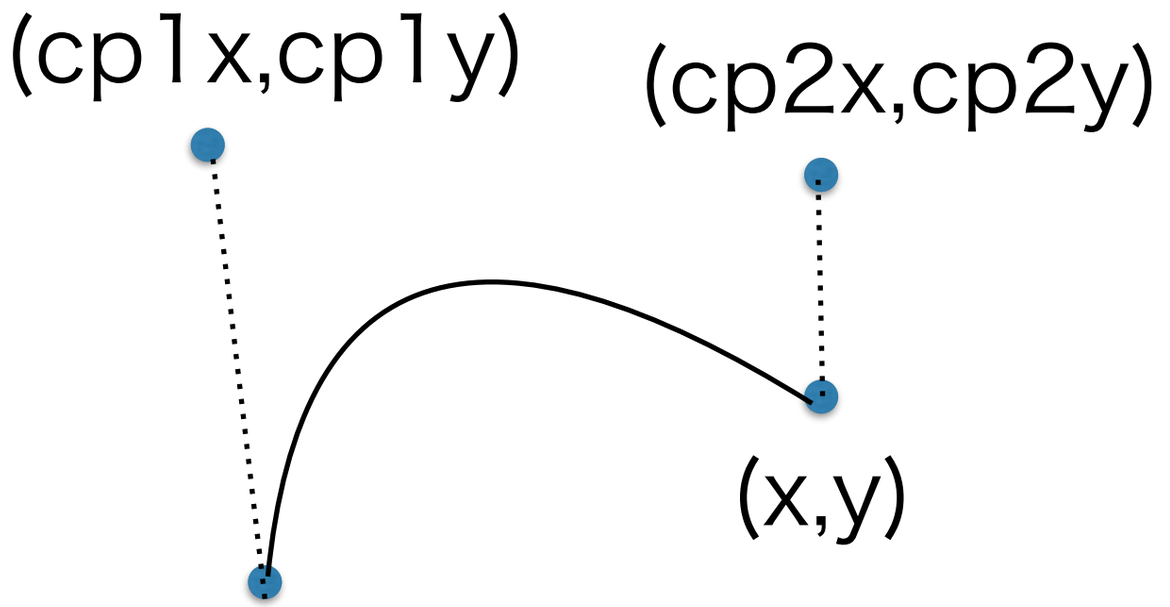


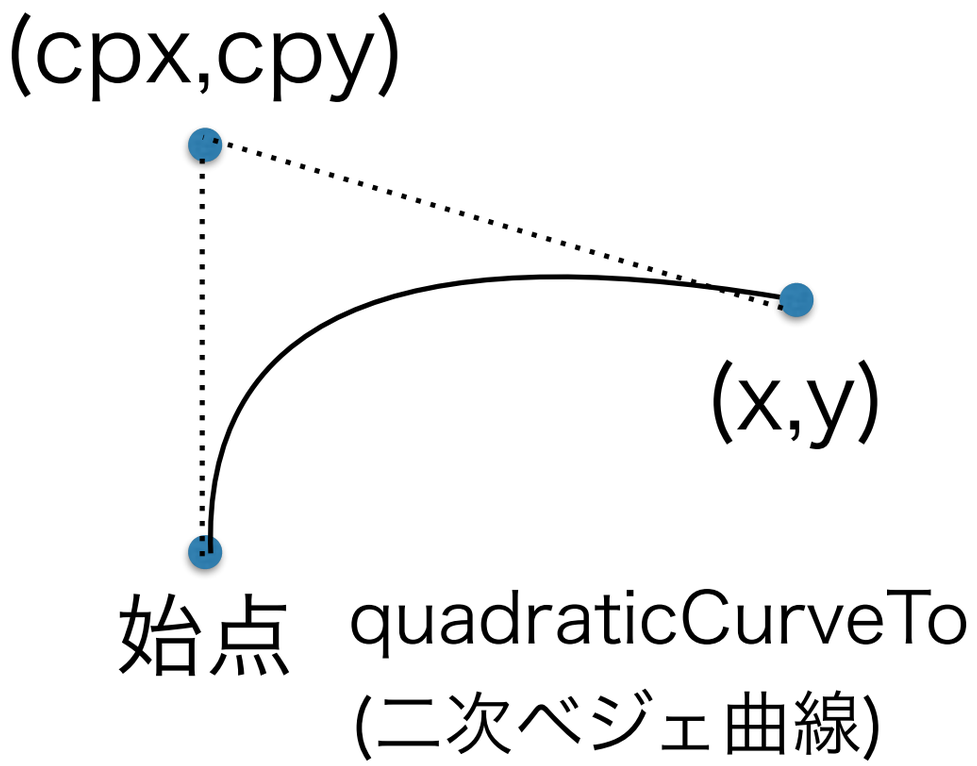



```
   context.beginPath();
1) context.moveTo(50,20);
2) context.lineTo(20,80);
3) context.lineTo(80,80);
   context.closePath();
   context.stroke();
```

曲線

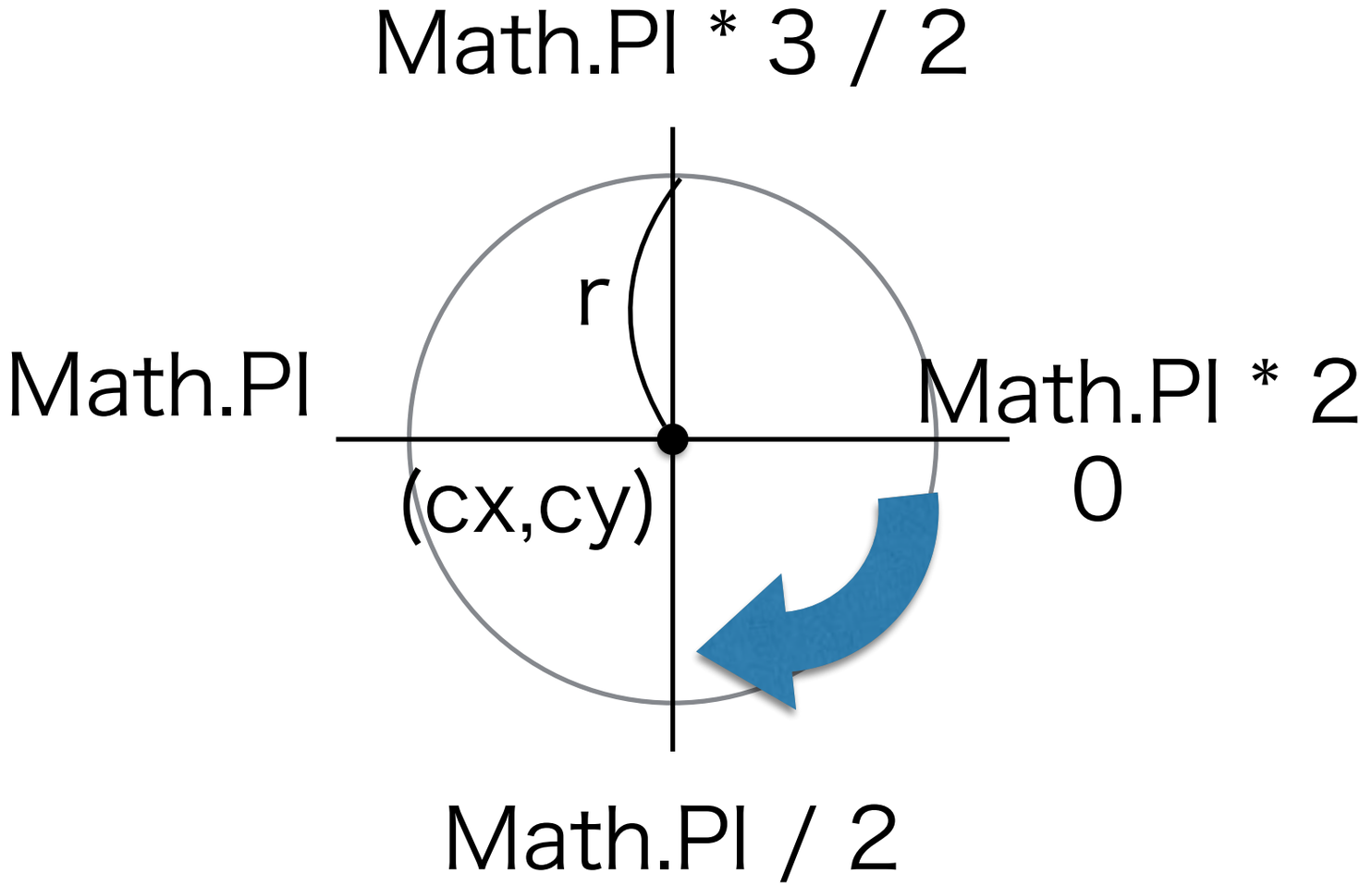
- ベジェ曲線
  - **bezierCurveTo(cp1x, cp1y, cp2x, cp2y, x, y)**
  - **quadraticCurveTo(cpx, cpy, x, y)**





円弧

- **arc(cx , cy , r , 開始角度 , 終了角度 , 反時計周りか)**
- 角度0は右
- 角度指定は**ラジアン**
  - **半径**に対する円周の比
  - 90°は$\pi/2$
  - 180°はπ
  - 360°は2π



- 角度は**時計周り**で数える
- 指定された角度まで時計回りに線を引くか、反時計周りに線を引くか

矩形

- パスを使って矩形を描くこともできるが、矩形描画用メソッドを使うと簡単に描画できる

- 矩形の**輪郭**を描画

  - **strokeRect(x , y , width , height)**

- 矩形の**塗り潰し**

  - **fillRect(x , y , width , height)**

- 矩形の**消去**(透明にする)

  - **clearRect(x , y , width , height)**

Canvasの塗りつぶし

- パスを記述したあと、**fill()**メソッドを呼び出すことで**fillStyle**に従って塗り潰しが描画される

塗り潰しの場合は、closePathしなくても暗黙的に閉じられる

```
context.beginPath();
context.moveTo(90,0);
context.lineTo(35,180);
context.lineTo(180,70);
context.lineTo(0,70);
context.lineTo(150,180);
context.fill();
```

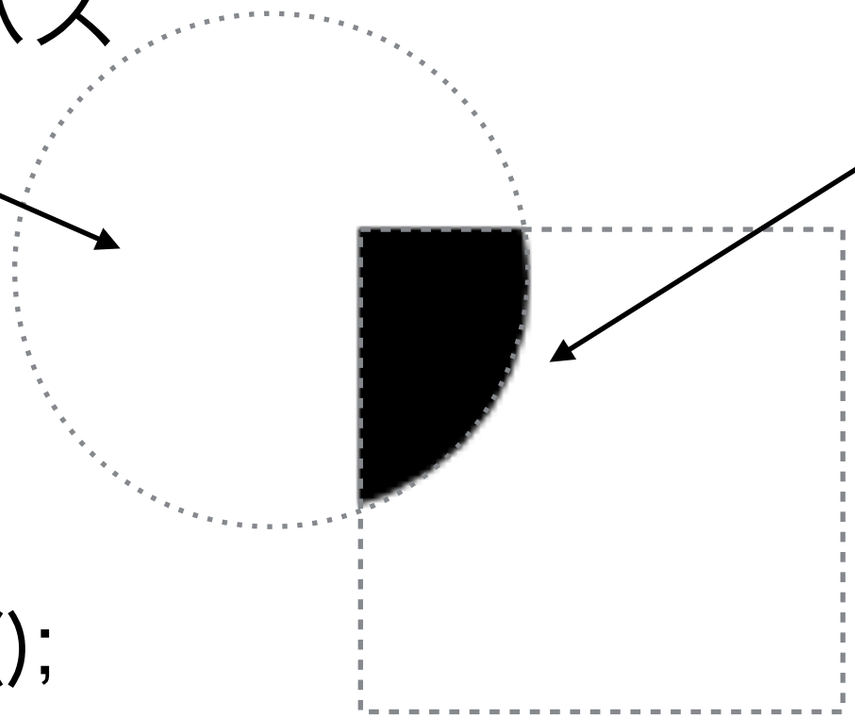
```
context.beginPath();
context.moveTo(10,10);
context.lineTo(10,110);
context.lineTo(110,110);
context.lineTo(110,10);
context.moveTo(50,20);
context.lineTo(160,20);
context.lineTo(160,60);
context.lineTo(50,60);
context.fill();
```

時計周りと反時計周りのパスが重なると塗り潰されないことがある

クリッピング領域

- **クリッピング領域(**クリッピングパス)を使うと描画したくない部分をマスクすることが出来る

クリッピングパス



fillRectと
クリッピングパス
が重なっている部分だけ
描画されている

```
context.beginPath();
context.arc(50, 50, 40, 0, Math.PI*2, false);
context.clip();
context.fillRect(60,40,100,100);
```

本来塗り潰される部分が
clip()メソッドで
クリッピング領域になる

問題1

次の描画結果になるスクリプトはどれか

A.
```
context.beginPath();
context.arc(50, 50, 40, 0, 135, false);
context.stroke();
```

B.
```
context2.beginPath();
context2.arc(50, 50, 40, 0, 135 * Math.PI / 180, false);
context2.stroke();
```

C.
```
context3.beginPath();
context3.arc(50, 50, 40, 0, 135 * Math.PI / 180, true);
context3.stroke();
```

テキスト描画

- Canvasに対して、指定した色、指定したフォント、文字の大きさでテキストを描画することができる
- 輪郭だけの袋文字や、ドロップシャドウなどの効果を付けることもできる
- CSSによるスタイル指定や、リンクの設定はできない

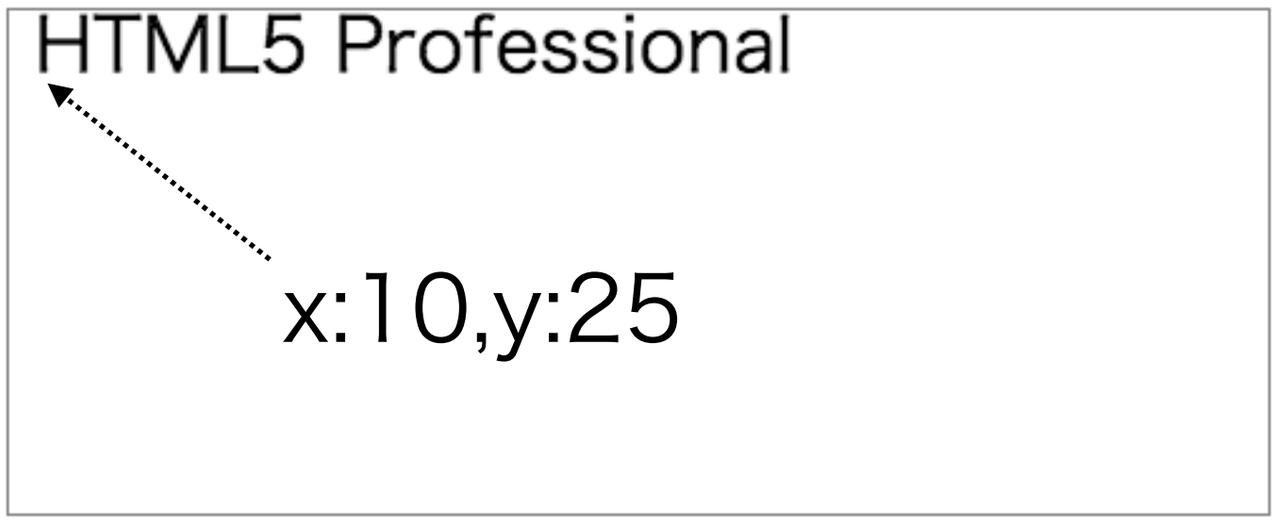
塗りつぶし描画

- 通常のテキスト描画をする場合には、
  context.fillText()メソッドを使用する

- context.**fillText(文字列,x,y)**

- x,yは通常はテキストの左下を指すので注意が必要

```
context.font = "30px 'sans-serif'";
context.fillText('HTML5 Professional', 10, 25);
```



輪郭描画

- 袋文字のような輪郭だけのテキスト描画をする場合には、context.**strokeText()**メソッドを使用する

- context.**strokeText(文字列,x,y)**

- x,yは通常はテキストの左下を指すので注意が必要

```
context.font = "30px 'sans-serif'";
context.strokeText('HTML5 Professional', 10, 25);
```

フォントの設定

- 文字サイズ、フォントの設定
  - fontプロパティ
- 文字色(他の図形描画と共通)
  - fillColor , strokeColorプロパティ
- ドロップシャドウ
  - shadowColor , shadowBlur プロパティ
  - shadowOffsetX , shadowOffsetY プロパティ
- 文字の配置
  - textAlign プロパティ

テキスト幅の算定

- テキストの折り返しや、テキストを枠線で囲む場合などに、テキストの幅を必要とする場合がある

- プロポーショナルフォントでは同じ文字数でもフォントや文字の並びによって全体の幅が異なるため、context.**measureText()**メソッドで実際のフォントと文字列で幅何ピクセルになるか計算する

```
context.font = "30px 'sans-serif' ";
var text = 'HTML5 Professional';
var metrics = context.measureText(text);
console.log( metrics.width + 'px' );
```

問題2

以下のスクリプトを実行した結果として正しいものはどれか

```
context.strokeStyle = '#FF0000';

context.fillText('HTML5',50,50);
```

図中の点はX=50,Y=50を表わしている

A.

B.

C.

D.

変形

- Canvasに対する描画を**拡大・縮小、回転、移動**させることができる

- 現在のCanvasの内容を変形させるのではなく、変形を**設定した後の描画に適用**されるので注意が必要

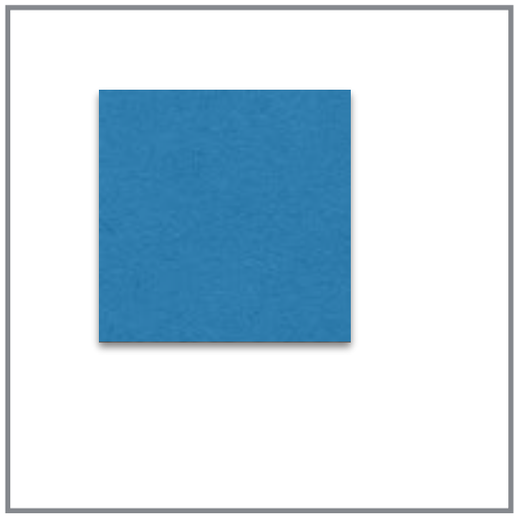
- 複数の変形を指定した場合、**指定の順番が異なると結果が変化**する。指定した逆の順番に適用されると考えやすい

拡大・縮小

・拡大縮小は、描画しようとする内容だけを拡大、縮小するのではなく、Canvas全体に適用される

context.**scale(1.5);**

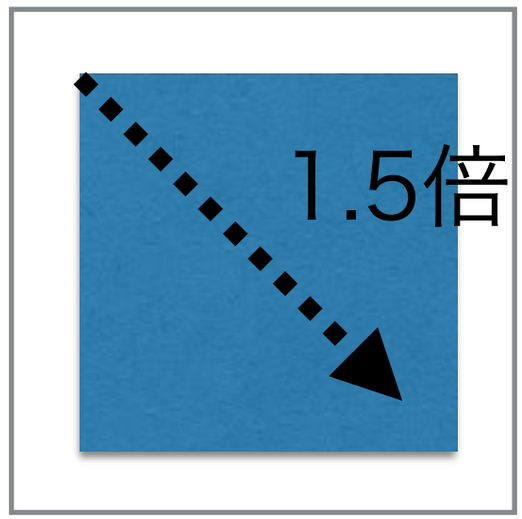
が設定されているCanvasに



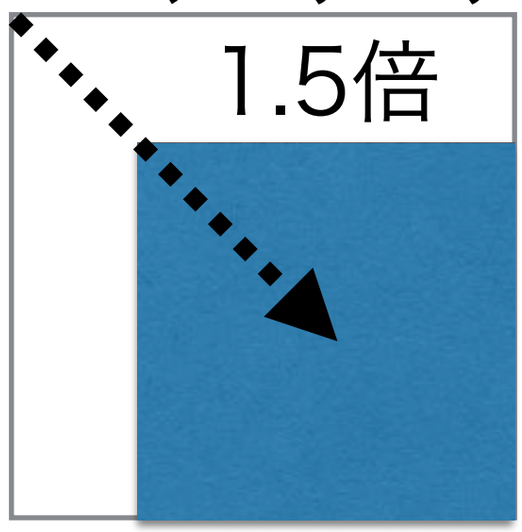
fillRect(10,10,20,20);

を実行すると

× fillRect(10,10,**30,30**)



○ fillRect(**15,15,30,30**)



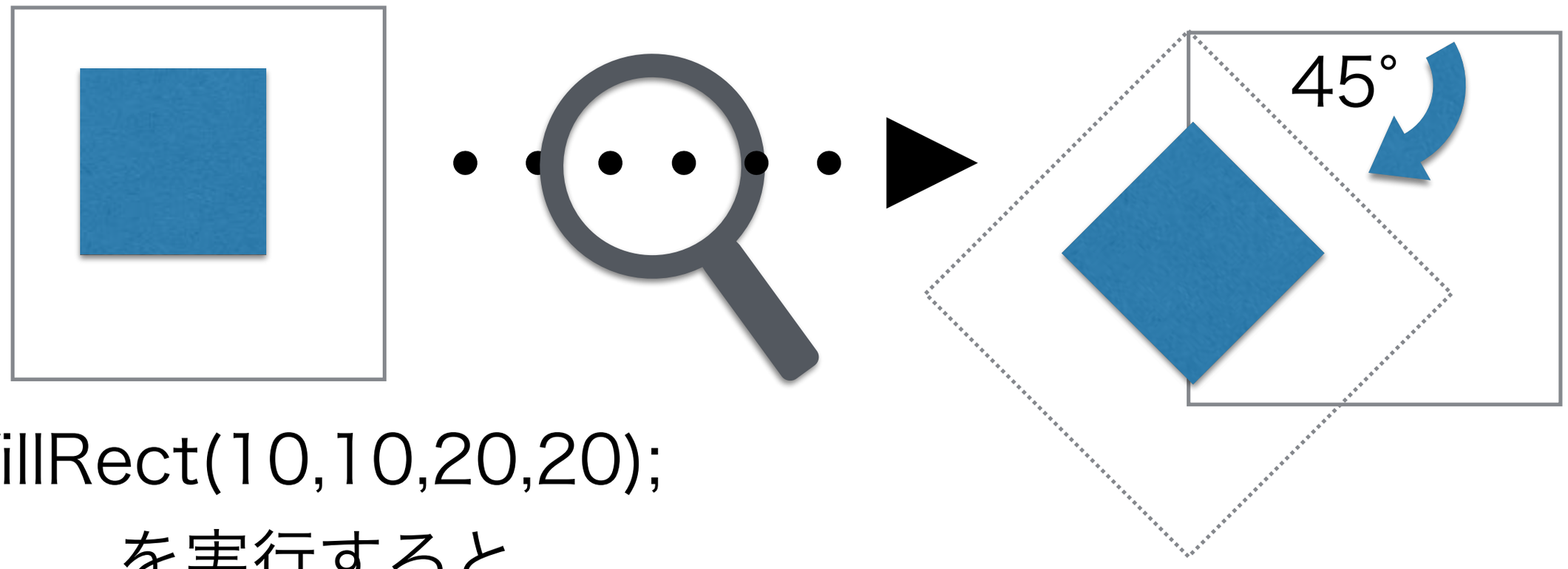
回転

- 回転はCanvas左上を原点に時計周りに回転する

- **Rotate()**メソッドの単位はラジアンなので、度で指定する場合には 度 * Math.PI / 180 で変換する

context.rotate**(45 * Math.PI / 180);**

が設定されているCanvasに



fillRect(10,10,20,20);

を実行すると

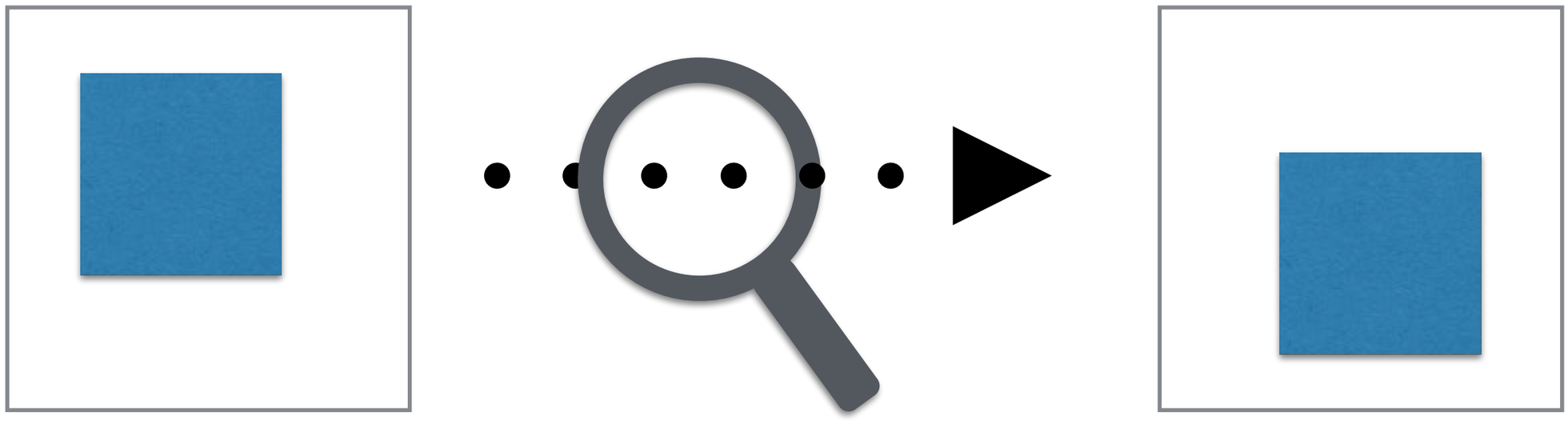
移動

- 移動はcontext.**translate()**メソッドで指定した量だけ、縦横にずれて描画される

- 複雑な変形の場合には原点で拡大縮小や回転をしたあと移動させるとわかりやすい

context.**translate(20,30);**

が設定されているCanvasに



fillRect(10,10,20,20);

を実行すると

fillRect(**30,40**,20,20);

と同じ結果に

変換マトリックス

- **scale(),rotate(),translate()**メソッドは、実行されると内部の変換マトリックスに影響を与える

- **setTransform()**メソッドは変換マトリックス(行列)を直接上書きする

- **setTransform(a , b , c , d , e , f)**

変換マトリックス

$$\begin{pmatrix} x & y & 1 \end{pmatrix} \begin{pmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{pmatrix}$$

例)translate(10,20)

$$\begin{pmatrix} x & y & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 10 \\ 0 & 1 & 20 \\ 0 & 0 & 1 \end{pmatrix}$$

x' = x*1+y*0+1*10

y' = x*0 + y*1+1*20

1 = x*0 + y*0 + 1*1

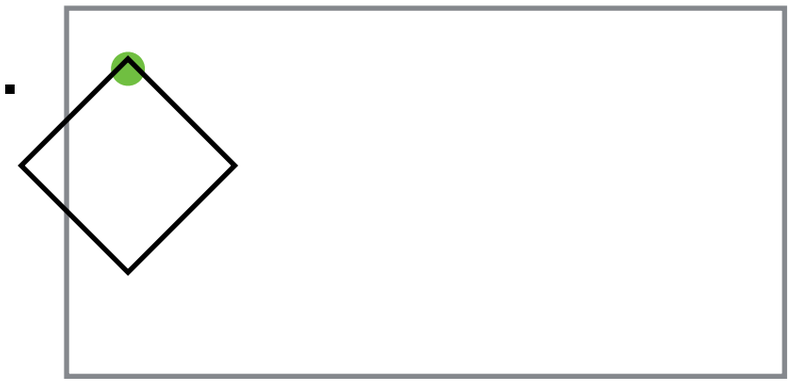
問題3

以下のスクリプトを実行した結果はどれか

```
context.translate(10,10);
context.rotate( Math.PI / 4);
context.strokeRect(10,10,30,30);
```

※ 図中の点はX=10,Y=10を表わしている

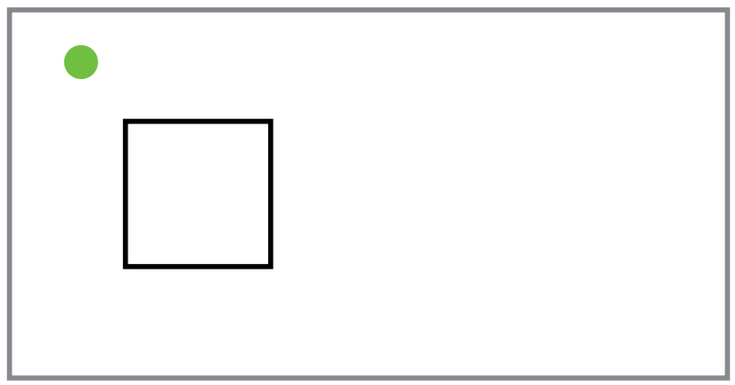
A.



B.

C.

D.



エフェクト

- Canvasでは、単に図形を描画するだけでは無く、画像に対してさまざまな表現を行なうことができる

- Canvas全体の**透明度を変化**させたり、描画や画像の表示の際にPhotoshopのレイヤーのように**合成方法を変えられる**

- 組込みのエフェクトで不足する場合は、自分で**ピクセル単位の編集**を加えて独自の効果を与えることもできる

透明度指定

- Canvasへの描画に対して透明度を設定できる
- context.**globalAlpha**に0.0(透明)～1.0(不透明)の値を代入することで、以後の描画に透明度が反映される

```
context1.fillStyle = "red";
context1.fillRect(0,0,50,50);
context1.fillStyle = "green";
context1.fillRect(20,20,50,50);
context1.fillStyle = "blue";
context1.globalAlpha = 0.5;
context1.fillRect(40,40,50,50);
```

合成方法の指定

- **現在のCanvasの内容と次の描画内容の合成方法**を指定できる

- context.**globalCompositeOperation**プロパティに文字列で合成方法を指定すると、乗算(multiply)やソフトライト(soft-light)など様々な合成方法を簡単に実現できる

- ただし、Photoshopなどのレイヤーのような仕組みは無いため、描画を行なう前に合成方法を指定する必要があり、描画の後で既に完了した描画の合成方法を変えることはできない

問題4

context.globalAlphaを0.8にした場合、画像の表示はどのようになるか、正しいものはどれか

A. 設定したCanvas全体がすこし透明になる

B. 設定したCanvas全体がほぼ透明になる

C. contextに次に行なう描画が少し透明になる

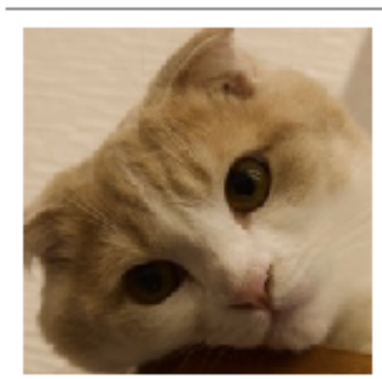
D. contextに次に行なう描画がほぼ透明になる

画像の描画

コンテキストに画像(**Imageオブジェクト**)を描画するには、context.**drawImage**メソッドを使用する。drawImageメソッドの描画結果には、クリッピング領域や合成方法の指定が影響する。

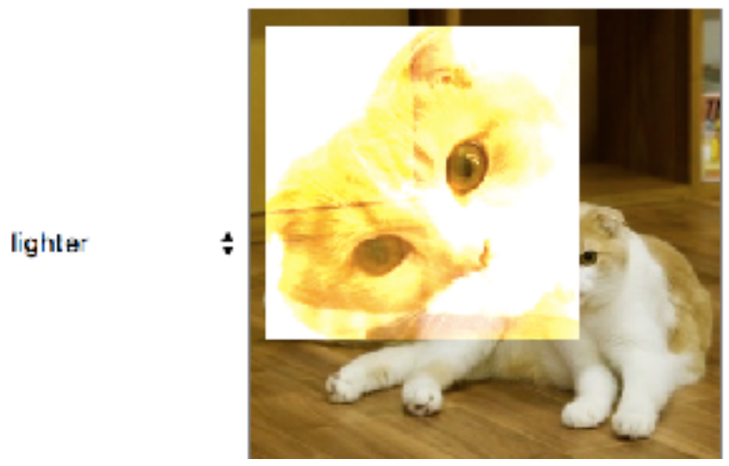
context.**drawImage(Imageオブジェクト , x , y)**

※ x,yは画像左上の表示位置を指定する



画像の描画

クリッピング



合成方法の指定

イメージデータ

標準の機能として実装されていない処理を行ないたい場合は、ピクセル単位で色の取得や書き込みの必要があり、そのために**ImageDataオブジェクト**を使用する

ImageDataオブジェクトでは、画像を**1ピクセルあたりR,G,B,Aの4バイトのデータを持つ一次元配列**として扱う

Canvasの内容を含むImageDataオブジェクトを作ることも、全く新規の画像データを作ることもできる

イメージデータ

ImageDataオブジェクトのdataプロパティの模式図

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	...
color	R0	G0	B0	A0	R1	G1	B1	A1	R2	G2	B2	A2	R3	G3	B3	A3	...
value	0	0	0	0	0	0	0	0	FF	0	0	0	0	0	0	0	...

(ピクセルのx座標+y座標*画像の幅) * 4 + c = 配列のインデックス

c = 0　R

c = 1　G

c = 2　B

c = 3　A

例) 幅300pxの画像の座標(10,40)のピクセルのBの値

(10 + 40 * 300) * 4 + 2 = 12012 バイト目

画像の生成

- context.**createImageData()**メソッドを使って、新規の**ImageData**オブジェクトを生成する

  context.**createImageData(width , height)**

- Canvasの内容を含むImageDataオブジェクトを取得するには、context.getImageData()メソッドを使用する

  context.getImageData(left , top , width , height)

- ImageDataオブジェクトは、context.putImageData()メソッドでCanvasに描画することができる

  context.putImageData(imageData , left , top)

問題5

画像に関するオブジェクトの生成メソッドとCanvasへの描画メソッドの組み合わせとして正しいものを2つ選べ。

A. Image / new Image() / drawImage()

B. ImageData / new Image() / drawImageData()

C. Image / createImage() / putImage()

D. ImageData / createImageData() / putImageData()

2.3.2 SVG

知識問題

SVGの特徴

Scalable Vector Graphics

- **ベクター形式の画像**であるため、高解像度なデバイスや、ページの拡大表示でも滑らかに表示できる。

- **XML**による**テキスト記述**のベクター形式で、内容によってはビットマップ形式よりもデータ量が少なくなる

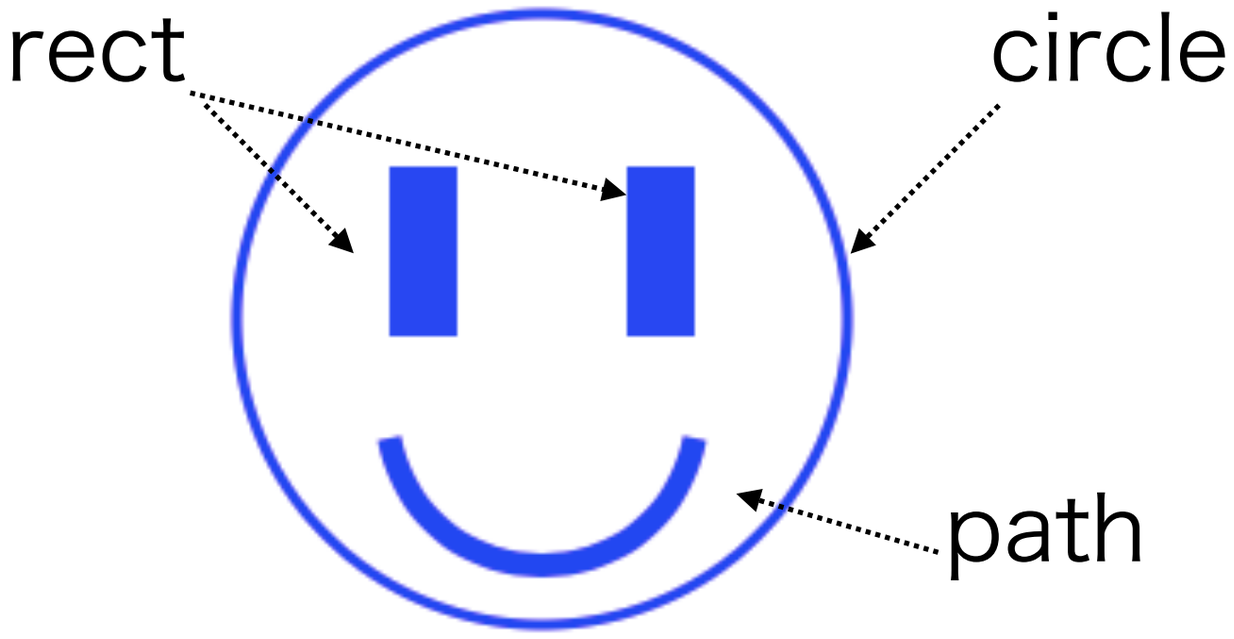
- **SVG要素**を使ってインラインで画像を記述できる

- **img要素のsrc属性**や、**object要素のdata属性**にSVG形式のファイルを指定すると表示できる

- CSSの**background-image**にSVG形式のファイルを指定して表示できる

インライン形式のSVGの例

```
<svg version="1.1" xmlns="http://www.w3.org/2000/svg" width="200" height="200" >
    <circle cx="95" cy="95" r="90" fill-opacity="0" stroke="#1017f1" stroke-width="3"/>
    <rect x="50" y="50" width="20" height="50" fill="#1d24f2"/>
    <rect x="120" y="50" width="20" height="50" fill="#1d24f2"/>
    <path d="M140 130C130 180 60 180 50 130" fill-opacity="0" stroke="#1017f1"
stroke-width="7" />
</svg>
```



Canvasとの違い

- CanvasはビットマップSVGはベクター形式
- Canvasの操作にはJavaScriptが必須、SVGはJavaScriptによる操作の他、画像として読み込んだり、タグを直接記述することもできる
- Canvasの編集は上書きのみ、SVGは要素やその属性を修正することで内容を変更できる
- 複雑な画像では画像の計算に時間が掛ることがある

問題6

SVGの特徴として正しくない説明はどれか

A. ベクター形式である

B. SVGタグではなくObjectタグを使用する

C. 後から描画要素を編集できる

D. 拡大しても滑らかに表示される

本日の内容

- 試験概要
- Canvas (2D)
  - 概要
  - コンテキスト
  - 基本図形描画
  - テキスト描画
  - 変形
  - エフェクト
  - イメージデータ
- SVG
  - SVGの特徴
  - Canvasとの違い

問題の解答

- 問題1: **C** / 角度がラジアンで指定されていて、反時計周りに線を引くCが正解。

- 問題2: **B** / strokeStyleはfill系のメソッドには影響を与えない。テキスト描画の座標は左下が基準。

- 問題3: **C** / 詳細はサンプルファイルを参照。

- 問題4: **C** / globalAlphaはCanvasへの描画の透明度。設定以降の描画全てに影響する。0.0で完全な透明、1.0で不透明。

- 問題5: **A,D** / オブジェクトの種類によって使用するメソッドが異なる。

- 問題6: **B** / Object要素でもSVGは表示できるが、SVG要素を使うこともできる。