最終更新　2024.2.8

# Canvasの使い方

## Canvasの使い方

　図形を書きたいときはCanvasを使います。例を示します。

<!DOCTYPE html>

<html lang="ja">

<head>

<meta charset="utf-8">

<title>canvas の使い方</title>

<script>

"use strict";

window.onload = init;

let canvas, d, canvas\_location;

let b1, p1;

function init(){

 canvas = document.getElementById("canvas-1");

 d = canvas.getContext('2d');

 canvas\_location = canvas.getBoundingClientRect();

 canvas.style.backgroundColor = "rgb(255,225,200)";

 p1 = document.getElementById("paragraph-1");

 b1 = document.getElementById("button-1");

 b1.addEventListener("click", draw);

 p1.innerHTML = "Canvas の left top : " +

 String(canvas\_location.left) + " " + String(canvas\_location.top) +

 "　　　width height : " + String(canvas.width) + " " +

 String(canvas.height);

}

// 図形を描く

function draw(){

// 折れ線

 let x1, y1, x2, y2, x3, y3;

 x1 = 20; y1 = 20; x2 = 40; y2 = 100; x3 = 70; y3 = 40;

 d.beginPath(); // これまでのパスをクリア

 d.lineWidth = 1;

 d.strokeStyle = "rgb(0,0,0)";

 d.moveTo(x1,y1); // パスを構築　ペンアップ

 d.lineTo(x2,y2); // パスを構築　ペンダウン

 d.lineTo(x3,y3); // パスを構築　ペンダウン

 d.stroke(); // パスを描く

// 長方形

 let x, y, xlen, ylen;

 x = 180; y = 200; xlen = 70; ylen = 50;

 d.beginPath(); // これまでのパスをクリア

 // これを怠ると d.fill() したときに、

 // これまでに描いた図形も fill() する

 d.rect(x, y, xlen, ylen); // 長方形のパスを構築する

 d.fillStyle = "rgb(200,255,200)"; // 塗りつぶし色

 d.fill(); // 塗りつぶす

 d.lineWidth = 3; // 線幅

 d.strokeStyle = "rgb(0,255,0)"; // 線色

 d.stroke() // 外枠を書く

}

</script>

</head>

<body>

<p>この下にキャンバスを描きます</p>

<p>

<canvas id="canvas-1" width="300px" height="300px"></canvas>

</p>

<p>

<input type="button" id="button-1" value="図形描画">

</p>

<p id="paragraph-1"></p>

</body>

</html>

　図形を描くには canvas という要素を使用します。

　<canvas id="canvas-1" width="300px" height="300px"></canvas> のように幅と高さを指定します。こうすると、300x300 ピクセルの図形描画用領域を確保できます。

　canvasに図形を描くときは、独特の作法があります。デバイスコンテキストと呼ばれるものを取得し、それに対して命令を送ります。デバイスコンテキストは以下のように取得します。

canvas = document.getElementById("canvas-1");

d = canvas.getContext("2d");

　変数canvas, d, b1, p1などは関数の外側で宣言しているので、グローバル変数です。

　canvasのウィンドウ上での位置の取得方法は以下の通りです。この例では使いませんが、マウスの相対位置を計算するときに必要です。

canvas\_location = canvas.getBoundingClientRect();

　背景色を以下のように設定します。

canvas.style.backgroundColor = "rgb(255,225,200)";

　canvasに線を書くのは、独特の作法があり、少し難解です。パス（折れ線の経路）というものを構築し、そのパスに沿って線幅、線色、などを指定して線を書きます。例を示します。

d.beginPath(); // これまでのパスをクリア

d.lineWidth = 1; // 線幅

d.strokeStyle = "rgb(0,0,0)"; // 線色

d.moveTo(x1,y1); // パスを構築　ペンアップ

d.lineTo(x2,y2); // パスを構築　ペンダウン

d.lineTo(x3,y3); // パスを構築　ペンダウン

d.stroke(); // パスを描く

　まず、これまでに構築してきたパスをクリアします。次に線幅と線色を指定しています。この指定はstroke() の手前ならどこに入れても構いません。

　moveTo(x,y) はペンを上げて (x,y) まで移動します。movetoとすると、エラーは出ず、何も起こらないので注意してください。Tは大文字です。lineTo(x,y) はペンを下げて (x,y) まで移動します。

　なぜこんな命令を使うのか不思議に思う人も多いと思います。line(x1, y1, x2, y2) のような命令が自然なように見えます。かつて（1990年以前のWindowsが普及するまでの時代）大型コンピュータで図形を描くときは、プロッタと呼ばれる装置を使っていました。プロッタの命令は「ペンを○番に交換」「ペンを上げて (x,y) へ行く」「ペンを下げて (x,y) へ行く」の3つだけです。moveTo, lineToはプロッタの時代の命令体系です。余談ですが、印刷業界標準のプリンタ制御言語はPostScriptです。このPostScriptもmoveto, linetoという命令で線を描きます。

　長方形は以下のように書きます。

let x, y, xlen, ylen;

x = 180; y = 200; xlen = 70; ylen = 50;

d.beginPath(); // これまでのパスをクリア

 // これを怠ると d.fill() したときに、

 // これまでに描いた図形も fill() する

d.rect(x, y, xlen, ylen); // 長方形のパスを構築する

d.fillStyle = "rgb(200,255,200)"; // 塗りつぶし色

d.fill(); // 塗りつぶす

d.lineWidth = 3; // 線幅

d.strokeStyle = "rgb(0,255,0)"; // 線色

d.stroke() // 外枠を書く

　輪郭を持つ図形は「輪郭の内側の塗りつぶし色」「輪郭の太さ」「輪郭の線色」の3つの要素を持ちます。内側を塗りつぶさないときはd.fillStyleとd.fill() は不要です。輪郭を描かないときは、d.lineWidthとd.strokeStyleとd.stroke() は不要です。

　上で示した例では、折れ線と長方形を書きました。それ以外にも以下のような図形があります。

* 円
* 楕円
* 文字

　それぞれの方法を示します。変数宣言は省略します。

円

x = 70; y = 210; radius = 50;

start\_angle = 0; end\_angle = Math.PI\*2;

d.beginPath();

d.arc(x,y,radius,start\_angle,end\_angle);

// 塗りつぶし

d.fillStyle = "rgb(255, 200, 200)";

d.fill();

// 外枠

d.lineWidth = 10

d.strokeStyle = "rgb(255, 0, 0)";

d.stroke();

楕円

x = 200; y = 70; xr = 50, yr = 100;

rotate = 45 \* Math.PI/180;

start\_angle = 0;

end\_angle = Math.PI\*2;

d.beginPath();

d.ellipse(x,y,xr, yr, rotate, start\_angle,end\_angle);

// 塗りつぶし

d.fillStyle = "rgb(220,220,255)";

d.fill();

// 外枠

d.lineWidth = 10

d.strokeStyle = "rgb(150,150,255)";

d.stroke();

文字

x = 20, y = 150;

d.font = "14pt Arial"; // フォント

d.fillStyle = "rgb(0,0,0)"; // 文字色

d.fillText("abc", x, y);

　JavaScriptでプログラムを組むには、通常のプログラミングのテクニックに加えて、htmlの知識、イベントの概念の理解が必要であり、プログラミング初心者には敷居が高いです。加えて、一つのことをするのに多くの方法があります。ここでは紹介しませんでしたが、無名関数など難解な書き方が色々とあります。

　その一方で、ブラウザさえあれば、スマホ、タブレット、PCのどこでも実行できるので、非常に実用性が高い言語でもあります。JavaScriptをプログラミングの学習に使うのが適切か否か、筆者は答えを出すことができません。

　高校の情報Ⅰの教科書に使われるプログラミング言語は、pythonかJavaScriptの二択のようです。センター試験の情報の試作問題はpython風の記述になっています。

　JavaScriptを使ったプログラムはPCのローカルファイルシステムにアクセスしたり（ダイアログを経由することが必要）、Webカメラなどのデバイスを制御する（ユーザーの許可が必要）ことができます。ブラウザさえあれば、だれにでも使ってもらえるアプリを作ることができる（PC, スマホ, タブレットなどデバイスを問わない）ので、作ったプログラムの汎用性は非常に高いです。一方で、movetoのように、綴りミスにエラーが出ないため、デバッグがしづらい言語でもあります。

　JavaScriptの学習は優れたテキストがWeb上に多数あるようです。基本的な枠組みを理解したなら、あとはWebサイトのテキストで自習できるでしょう。

　2023年に登場したChatGPTはプログラムの組み方を根本的に変えました。サンプルプログラムを組んでもらうことができますし、デバッグを依頼することもできます。ChatGPTに聞きながらプログラミング学習をすすめるのもよいでしょう。ChatGPTはプログラミングの強力なツールですが、できたプログラムが正しいか否かの判断は人間がする必要があります。ChatGPTが存在しても、プログラミングの技能の重要性が下がることはないと予想します。