JavaScript 超入門

河西　朝雄著

KASAI.SOFTWARELAB

はじめに

本書はプログラム経験のない人が簡単に JavaScript でプログラムを作成できるように、要点のみを解説した超入門書で、1 章と 2 章で構成します。

1 章では HTML5 から使用できるようになった<canvas>要素（タグ）へのグラフィックス処理を例にして JavaScript の基本的な文法を説明します。if 文,for 文や for の二重ループといった流れ制御文の使い方を説明します。データを効率よく管理するための配列として 1 次元配列と 2 次元配列について説明します。ある処理単位をひとかたまりにして名前を付け、その名前で呼び出しが行えるようにした関数について説明します。初心者にとってグラフィックス処理で得られる結果は視覚的に興味がわく題材となります。

2 章では HTML 要素（タグ）をオブジェクトとして取得し JavaScript から操作する方法や、タグを用いずにコンストラクタから直接オブジェクトを生成し document に追加する方法を説明します。HTML 要素のクリック（タッチ）で発生する onClick イベントや選択ボックスの項目の選択で発生する onChange イベントなどの処理方法を説明します。JavaScript の標準オブジェクトである Math オブジェクトや String オブジェクトの使い方を説明します。

目次

1 章　グラフィックスを用いた JavaScript 入門　4

1-1　グラフィックス処理の概要　5
1-2　for 文　9
1-3　if else 文　13
1-4　二重ループ　17
1-5　1 次元配列　20
1-6　2 次元配列　23
1-7　関数　27

2 章　オブジェクトの操作　32

2-1　オブジェクトの取得　33
2-2　連想配列と for in 文　36
2-3　イベント処理　38
2-4　選択ボックスの処理　41
2-5　オブジェクトの生成と追加　44
2-6　Math オブジェクト　48
2-7　String オブジェクト　52

練習問題解答　55

1 章　グラフィックスを用いた JavaScript 入門

HTML5 から使用できるようになった<canvas>要素（タグ）へのグラフィックス処理を例にして JavaScript の基本的な文法を説明します。初心者にとってグラフィックス処理で得られる結果は視覚的に興味がわく題材となります。

プログラムを作る上で基本となることは、プログラムの流れを制御する流れ制御文、データをまとめて管理する配列、一連の処理内容をひとかたまりにまとめて記述しそれを呼び出して使う関数などです。このようなプログラムを作る際の決まりを定めたものを言語仕様といいます。この章では以下のような基本的な言語仕様について説明します。

・for 文
・if else 文
・二重ループ
・1 次元配列
・2 次元配列
・関数

1-1　グラフィックス処理の概要

HTML5 の<canvas>タグを利用すれば、JavaScript から lineTo や strokeRect などのメソッドを使って、直線、矩形（四角）、円などの 2 次元グラフィックス描画を行うことができます。

1.　<canvas>タグ

HTML5 ではグラフィックス描画を行うための領域を<canvas>タグで指定することができます。width と height 属性にキャンバスの幅と高さをピクセル単位で指定します。

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

「注」ピクセル単位を明示するには"400px"とします。

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

2.　<canvas>領域への描画手順

<canvas>タグで指定した領域にグラフィックス描画を行うにはキャンバスオブジェクトを取得し、さらにそのキャンバスオブジェクトから実際にグラフィックス描画を行うためのコンテキストオブジェクトを取得します。getContext に指定できる引数は現在「2d:2 次元グラフィックス」だけです。このコンテキストオブジェクトに対し strokeRect メソッドや fillRect メソッドを使って図形を描画します。

```
var canvas = document.getElementById("canvas");
if(canvas.getContext){
    var context = canvas.getContext("2d");
    // context に対し描画メソッドを適用する
}
```

3.　描画色

描画を行う際の描画色は strokeStyle プロパティに設定します。指定する色は"blue"のような色名、"#0000ff"のような 16 進数の RGB 値、"rgb(0,0,255)"のような rgb 関数が指定できます。

```
context.strokeStyle = "blue"; //  描画色
```

矩形や円の内部を塗りつぶす色は fillStyle プロパティに設定します。

```
context.fillStyle = "blue"; //  塗りつぶす色
```

4.　矩形の描画

矩形の描画は strokeRect メソッドを使います。

context.strokeRect(x,y,w,h)

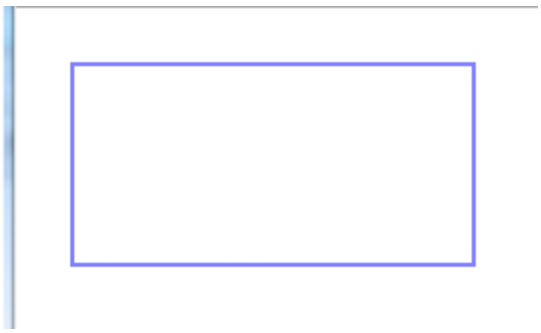
・・・左上隅座標を(x,y)、幅を w、高さを h とする矩形を描く。

矩形の内部を塗りつぶすには fillRect メソッドを使います。

・・・左上隅座標を(x,y)、幅を w、高さを h とする矩形内部を塗りつぶす。ただし外枠は描画されない。

「例題 1-1」左上隅座標を（20,20)、幅 200、高さ 100 の矩形を青で描きます。

```
<!DOCTYPE html>
<html>
<body>
<canvas id="canvas" width="400" height="400"></canvas>
<script type="text/javascript">
    var canvas = document.getElementById("canvas");
    if(canvas.getContext){
        var context = canvas.getContext("2d");
        context.strokeStyle = "blue"; //  描画色
        context.strokeRect(20, 20, 200, 100);
    }
</script>
</body>
</html>
```



5. 直線の描画

矩形を描く strokeRect メソッドは、このメソッドでキャンバスに直接描画を行いました。他の図形（直線や円など）はパスに対する描画メソッドを使って一旦パスに対し描画を行い、その後 stroke メソッドを使ってパス情報をキャンバスに描画します。パスとは直線、円（円弧)、ベジェ曲線などの図形の各点の情報を繋いだ経路です。

パスへの描画を行うにはまず、beginPath メソッドを使って現在のパスをリセット（クリ

ア）してから行います。

パスに対し直線を描くためのメソッドとして以下があります。

moveTo(x,y)

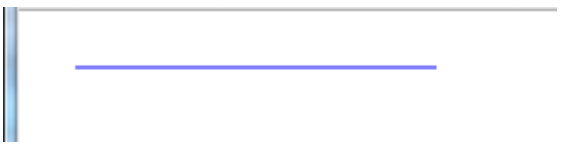
・・・描画現在位置を(x,y)に移動。

lineTo(x,y)

・・・描画現在位置から(x,y)に直線を描く。

「例題 1-2」(20,20)－(200,20)間に青色の直線を描きます。

```
<!DOCTYPE html>
<html>
<body>
<canvas id="canvas" width="400" height="400"></canvas>
<script type="text/javascript">
	var canvas = document.getElementById("canvas");
	if(canvas.getContext){
		var context = canvas.getContext("2d");
		context.strokeStyle = "blue"; //  描画色
		context.beginPath();		//  パスの開始
		context.moveTo(20, 20);
		context.lineTo(200, 20);
		context.stroke();		//  パスの描画
	}
</script>
</body>
</html>
```



「注」JavaScript の書き方

JavaScript は HTML ファイル内に次のように<script>・・・</script>タグで囲んだスクリプトブロック内に記述します。type 属性に"text/javascript"を指定することで、このスクリプトブロックは JavaScript で記述されていることを示します。"text/javascript"は省略可能です。JavaScript の文の終わりにはセミコロン（；）を置きます。

<!DOCTYPE html>

```
<html>
<head>
<title>プログラム例</title>
</head>
<body>
<script type="text/javascript">
    // JavaScript のプログラム
    ・
    ・
</script>
</body>
</html>
```

「注」オブジェクト、メソッド、プロパティ

オブジェクトとは処理を行う対象物を表し、オブジェクトを操作するための命令としてメソッドとプロパティがあります。使用できるメソッドとプロパティはオブジェクトごとに決められています。上の例では context というオブジェクトに対し strokeStyle プロパティや strokeRect メソッドを使用しています。

メソッド,プロパティの一般的な書式は以下のようになります。オブジェクトとメソッドまたはプロパティの間は「.」で区切ります。

オブジェクト.メソッド（引数 1,引数 2,・・・）;
オブジェクト.プロパティ[＝値] ;

メソッドは与えられた複数の引数（ない場合もある）をオブジェクトに渡し、そこである処理をした後、戻り値があればそれを返します。

プロパティはオブジェクトの属性を示す変数のようなもので、１つの値を取得あるいは設定します。「オブジェクト.プロパティ＝値」なら値の設定となり、「オブジェクト.プロパティ」なら値の取得になります。

1-2　for 文

プログラムの流れを制御する文を流れ制御文と呼び、for 文、if else 文などがあります。

1.　for 文の書式

for 文は決められた回数の繰り返しに使います。たとえば、

```
    ↓初期値
for (i=1;i<=10;i++){
    ・    ↑   ↑増減式
    ・    終了条件式
}
```

は、変数 i を 1 から始め、10 以下の間、i を+1 しながら、{と}で囲まれた範囲（ブロック）を繰り返します。{　}内に書く文が 1 つのときは{　}を省略できます。繰り返しのことをループと呼びます。for 文の繰り返しを制御している変数のことをループ変数と呼びます。

☆文例

```
for (i=5;i<=10;i++)
```

・・・i を 5 から始め i を+1 しながら 10 まで繰り返す。6 回繰り返すことになる。

```
for (i=10;i>0;i--)
```

・・・i を 10 から始め、i を-1 しながら 0 より大きい間繰り返す。10 回繰り返すことになる。

```
for (x=-2.0;x<=2.0;x+=0.5)
```

・・・x を-2.0 から始め,+0.5 しながら 2.0 になるまで繰り返す。

2.　演算子

データを加減乗除するものを演算子と呼びます。i++は変数 i の内容を+1 します。i—は変数 i の内容を-1 します。++をインクリメント演算子、--をデクリメント演算子と呼びます。i++は i=i+1 と同じ意味、i—は i=i-1 と同じ意味です。i+1 の＋は加算演算子、i-1 の-は減算演算子です。

x+=0.5 は x の内容を+0.5 します。x=x+0.5 と同じ意味です。+=を複合代入演算子と呼びます。

3.　変数

プログラムを書く上で、変数の役割をしっかり押さえておく必要があります。変数はプログラムの進行によってその値が変わります。この意味から「変わる数＝変数」と呼びます。これに対し 10 や”hello”などの変化しないものを定数と呼びます。定数は数値定数と文字列定数に分かれます。

〇変数の宣言

変数は使用する前に、予約語の var を使って宣言します。var は variable を意味します。

var i;

は変数名が「i」の変数を宣言しています。

複数の変数を宣言するには変数をコンマ（,）で区切ります。

var i,j;

それぞれ単独に

var i;

var j;

と宣言しても良いです。

JavaScript では変数は宣言しなくても使用できますが、プログラムの見やすさ、安全性を考えると宣言すべきです。

○変数名の付け方

変数に付ける名前を変数名と呼びます。変数名には名前付け規則がありますが本書では以下のような規則で命名します。

・英字で始まる英数字。a1 や sum など。

・英大文字と英小文字を区別する。SUM,Sum,sum はいずれも異なる変数名となる。

・予約語を使用してはいけないが、それを含むことはできる。たとえば、for は認められないが force は認められる。

変数名はユーザが決めれば良いわけですが、ある程度の共通ルールに従った方がプログラムを読みやすくします。

・for などのループ変数には i,j,k などを使う。座標を示すものは x,y などを使う。

・変数の役割を連想できる変数名を使う。合計なら sum など。

「例題 1-2」始点の x 座標を 20、終点の x 座標を 200 とする直線を y 座標を 20～200 まで 20 きざみに変化させて描きます。

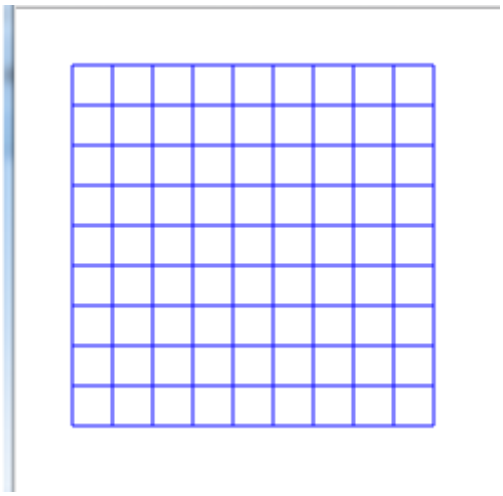
```
<!DOCTYPE html>
<html>
<body>
<canvas id="canvas" width="400" height="400"></canvas>
<script type="text/javascript">
    var canvas = document.getElementById("canvas");
    if(canvas.getContext){
        var context = canvas.getContext("2d");
        context.strokeStyle = "blue"; //  描画色
```

```
        for (var y=20;y<=200;y+=20){
            context.beginPath();          // パスの開始
            context.moveTo(20, y);
            context.lineTo(200, y);
            context.stroke();             // パスの描画
        }
    }
</script>
</body>
</html>
```

「練習問題 1-2」始点の y 座標を 20、終点の y 座標を 200 とする直線を x 座標を 20～200 まで 20 きざみに変化させて描く処理を追加しなさい。

```
<!DOCTYPE html>
<html>
<body>
<canvas id="canvas" width="400" height="400"></canvas>
<script type="text/javascript">
    var canvas = document.getElementById("canvas");
    if(canvas.getContext){
        var context = canvas.getContext("2d");
        context.strokeStyle = "blue";
        for (var x=20;x<=200;x+=20){ // 縦線
            context.beginPath();
```

```
                    ①
                    ②
                context.stroke();
            }
            for (var y=20;y<=200;y+=20){ // 横線
                context.beginPath();
                context.moveTo(20, y);
                context.lineTo(200, y);
                context.stroke();
            }
        }
</script>
</body>
</html>
```



1-3　if else 文

1.　if else 文の書式

条件を判定し、その判定に応じて実行する処理を変えるには if else 文を使います。

```
if (条件式) {
    文 1　←条件を満たしたときに実行される文
}
else {
    文 2　←条件を満たさないときに実行される文
}
```

else 部に書くものがなければ else 部全体を省略します。{　}内に書く文が 1 つの時は{　}を省略することができます。

条件式としては、次のようなものを書きます。条件式を満たしたときを真、満たさなかったときを偽と呼びます。

a>1	変数 a が 1 より大きいとき真
a==1	変数 a が 1 のとき真
a==1 && b==1	変数 a が 1 でかつ変数 b が 1 のとき真
a==1 || b==1	変数 a が 1 または変数 b が 1 のとき真

2.　比較演算子

条件式において、大きい、小さい、等しいなどの大小比較を行う演算子を比較演算子と呼び次の 6 つがあります。等しいは==と=を 2 つ書くことに注意してください。

比較演算子	意味
>	左辺は右辺より大きい。
>=	左辺は右辺より大きいか等しい。
<	左辺は右辺より小さい。
<=	左辺は右辺より小さいか等しい。
==	左辺と右辺は等しい。
!=	左辺と右辺は等しくない。

3.　論理演算子

1 つの条件式の真と偽を否定（反転）する!演算子、2 つの条件式を組み合わせて真・偽を判定する&&演算子、||演算子を論理演算子と呼びます。

論理演算子	意味
!	否定（NOT）。真なら偽、偽なら真。
&&	かつ（AND）。2 つの条件式の両方が真のとき真。

||	または(OR)。2 つの条件式のどちらかが真のとき真。

4. ブロックとインデント

for 文や if 文では制御対象となる文が複数になります。このような論理的にひとまとまりの複数の文をブロック（複文）と呼び｛と｝で囲みます。ブロックの中にブロックが入る構造をネスト（入れ子）と呼び、ネストが深くなるたびに、ブロックの書き出す位置を右にずらします。これをインデントと呼びます。インデントの幅は何文字でも良いのですが、通常 4 文字が多いです。以下は<script>ブロックの中に for ブロック、for ブロックの中に if ブロックがネストしています。ネストするたびにインデントが深く（右に書き出し位置が長く）なります。

```
<script type="text/javascript">
    for (・・・){
        if (・・・){

        }
    }
</script>
```

「例題 1-3」直線を赤と青の交互に描きます。y 座標を 20～200 まで 20 きざみで変化させるとき、y が 40,80,120,160,200 といった 40 の倍数のきは青、そうでない場合は赤とします。y が 40 の倍数か調べるには余りを求める「%」演算子を使って「y%40」が「0」なら 40 の倍数と判定します。

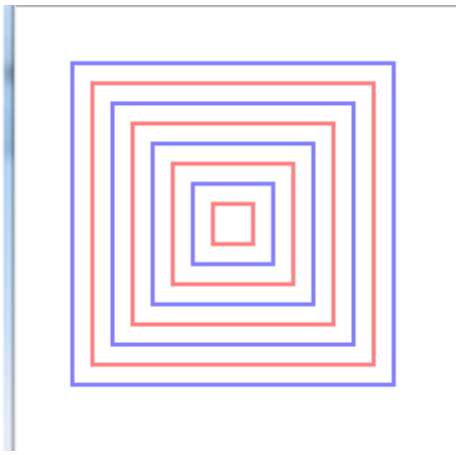
```
<!DOCTYPE html>
<html>
<body>
<canvas id="canvas" width="400" height="400"></canvas>
<script type="text/javascript">
    var canvas = document.getElementById("canvas");
    if(canvas.getContext){
        var context = canvas.getContext("2d");
        for (var y=20;y<=200;y+=20){
            if (y%40==0)
                context.strokeStyle = "blue"; //  青
            else
                context.strokeStyle = "red"; //  赤
```

```
                context.beginPath();
                context.moveTo(20, y);
                context.lineTo(200, y);
                context.stroke();
            }
        }
</script>
</body>
</html>
```

「練習問題 1-3」(20,20)を左上隅座標、幅と高さを 200 の四角から始め、左上隅座標を(10,10)移動しながら、幅と高さを 20 ずつ減らした矩形を青と赤で交互に描きなさい。

```
<!DOCTYPE html>
<html>
<body>
<canvas id="canvas" width="400" height="400"></canvas>
<script type="text/javascript">
    var canvas = document.getElementById("canvas");
    if(canvas.getContext){
        var context = canvas.getContext("2d");
        for (var x=20;x<=100;x+=10){
            if (_____①_____)
                context.strokeStyle = "blue"; //  青
            else
```

```
                context.strokeStyle = "red"; //  赤
                context.strokeRect(x, x, 200-2*x, 200-2*x);
            }
        }
</script>
</body>
</html>
```



1-4　二重ループ

ループの中にループが何重にも入る構造を多重ループと呼びます。for 文の二重ループは以下のような構造です。

```
var i,j;
for (i=1;i<=2;i++){
    for (j=1;j<=3;j++){
        A
    }
}
```

ループ変数 i が外ループ、ループ変数 j が内ループを管理します。この二重ループは次のように動作します。

外側のループを開始し、i が 1 のまま内側のループの j を 1～3 まで繰り返します。内側ループの繰り返しが終了すると外側のループ変数 i が 2 となり、再び内側のループを繰り返します。

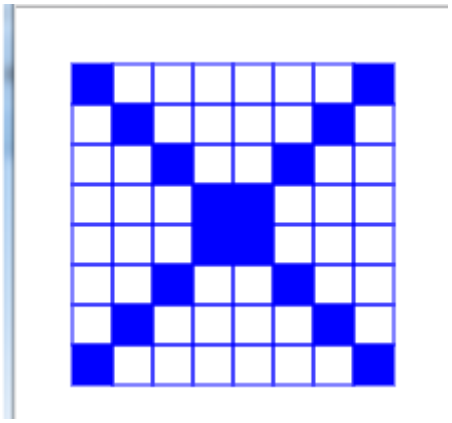
上の二重ループにおいて、A 部におけるループ変数 i と j の値をトレースすると次のようになります。

i	j
1	1
	2
	3
2	1
	2
	3

「例題 1-4」縦 8×横 8 の矩形を青で描きます。縦をループ変数 i、横をループ変数 j で管理したとき、i==j（対角線の要素）のときと i+j==9（逆対角線の要素）のときに矩形内部を青で塗ります。結果として「X」の文字が描けます。

```
<!DOCTYPE html>
<html>
<body>
<canvas id="canvas" width="400" height="400"></canvas>
<script type="text/javascript">
    var canvas = document.getElementById("canvas");
```

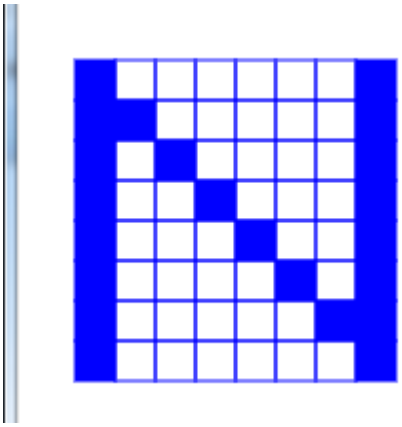
```
    if(canvas.getContext){
        var context = canvas.getContext("2d");
        context.strokeStyle = "blue"; //  輪郭の色
        context.fillStyle = "blue"; //  塗る色
        for (var i=1;i<=8;i++){
            for (var j=1;j<=8;j++){
                context.strokeRect(20*j, 20*i, 20, 20);
                if (i==j || i+j==9)
                    context.fillRect(20*j, 20*i, 20, 20);
            }
        }
    }
</script>
</body>
</html>
```



「練習問題 1-4」N の文字を描きなさい。

```
<!DOCTYPE html>
<html>
<body>
<canvas id="canvas" width="400" height="400"></canvas>
<script type="text/javascript">
    var canvas = document.getElementById("canvas");
    if(canvas.getContext){
        var context = canvas.getContext("2d");
        context.strokeStyle = "blue"; //  輪郭の色
```

```
        context.fillStyle = "blue"; //  塗る色
        for (var i=1;i<=8;i++){
            for (var j=1;j<=8;j++){
                context.strokeRect(20*j, 20*i, 20, 20);
                if (_____①_____)
                    context.fillRect(20*j, 20*i, 20, 20);
            }
        }
    }
</script>
</body>
</html>
```



1-5　1 次元配列

配列は、配列名と添字（そえじ）を用いて多数のデータを管理するためのデータ構造です。

```
a[i]
  ↑添字。配列要素の番号
↑配列名
```

配列は Array オブジェクトと new 演算子を用いて次のように宣言します。

```
var a=new Array(5);
      ↑          ↑配列のサイズ（要素数）
      配列名
```

これで a[0]~a[4]という 5 個の要素が確保されます。この配列に対し

```
for (i=0;i<5;i++)
    a[i]=i;
```

とすれば次のように a[0]～a[4]の要素にデータが入ります。配列の基底（先頭）要素の添字は 0 スタートです。

配列の宣言時に初期化データを指定することもできます。

```
var girl=new Array("結衣","彩香","理沙");
```

「例題 1-5」M を示す図形の各点の x,y 座標が配列 x[]と y[]に格納されています。各点の座標値を 10 倍した座標位置で各点を直線で結びます。始点位置（i==0 のとき）だけは moveTo で移動します。

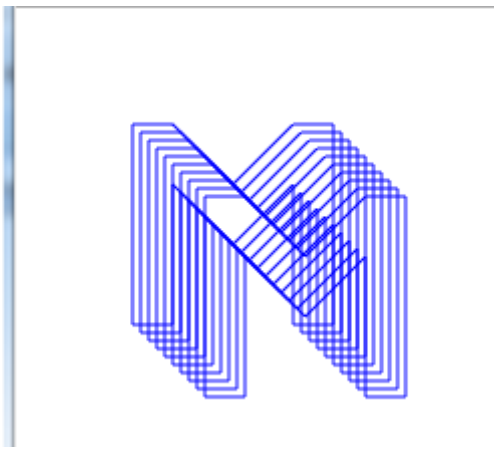
```
<!DOCTYPE html>
<html>
<body>
<canvas id="canvas" width="400" height="400"></canvas>
<script type="text/javascript">
    var canvas = document.getElementById("canvas");
    if(canvas.getContext){
        var context = canvas.getContext("2d");
        var x=new Array(0,2,2,5,8,8,10,10,8,5,2,0,0);
        var y=new Array(0,0,7,4,7,0,0,10,10,7,10,10,0);
        context.strokeStyle = "blue"; //  青
        context.beginPath();
        for (var i=0;i<x.length;i++){
```

```
            var px=50+10*x[i];
            var py=150-10*y[i];
            if (i==0)
                context.moveTo(px,py);
            else
                context.lineTo(px,py);
        }
        context.stroke();
    }
</script>
</body>
</html>
```

「練習問題 1-5」描画の開始ベースを（50,150）位置から(4,4)ずつ、ずらしながら M を 10 回描きなさい。

```
<!DOCTYPE html>
<html>
<body>
<canvas id="canvas" width="400" height="400"></canvas>
<script type="text/javascript">
    var canvas = document.getElementById("canvas");
    if(canvas.getContext){
        var context = canvas.getContext("2d");
        var x=new Array(0,2,2,5,8,8,10,10,8,5,2,0,0);
        var y=new Array(0,0,7,4,7,0,0,10,10,7,10,10,0);
        context.strokeStyle = "blue"; //  青
        for (n=0;n<10;n++){
```

```
                context.beginPath();
                for (var i=0;i<x.length;i++){
                    var px=_____①_____
                    var py=_____②_____
                    if (i==0)
                        context.moveTo(px,py);
                    else
                        context.lineTo(px,py);
                }
                context.stroke();
            }
        }
</script>
</body>
</html>
```



1-6　2 次元配列

JavaScript での 2 次元配列の宣言は C や Java などと異なる特殊な形になります。3 行 4 列の 2 次元配列を宣言するには次のようにします。a[i]という行要素にさらに配列オブジェクトを生成して仮想的に 2 次元配列として扱います。

```
var i;
var a=new Array(3);　←3 行の配列を宣言
for (i=0;i<3;i++){
    a[i]=new Array(4);　← i 行にさらに 4 列の配列を宣言
}
```

これで次のような 2 次元配列が宣言されました。

	0	1	2	3
0				
1				
2				

i 行 j 列の要素は a[i][j]で参照できます。2 次元配列の宣言の仕方は C,Java などと異なりますが、参照の仕方は同じです。

行要素を i、列要素を j で管理して 2 次元配列の全ての要素に 0 を格納するには次のようにします。

```
for (i=0;i<3;i++){
    for (j=0;j<4;j++){
        a[i][j]=0;
    }
}
```

宣言時に 2 次元配列に初期化データを与えるには次のようにします。

```
var a=new Array(3);
a[0]=new Array(0,0,0,0);
a[1]=new Array(0,0,0,0);
a[2]=new Array(0,0,0,0);
```

「注」行列

配列を扱うとき行要素を i、列要素を j で表します。変数名は i と j でなければいけないということではありませんが、数学で行列を扱うときに a_{ij} のような表現を使うため、これにあ

わせてあります。ただ初心者には i と j が見わけにくいので注意して入力してください。

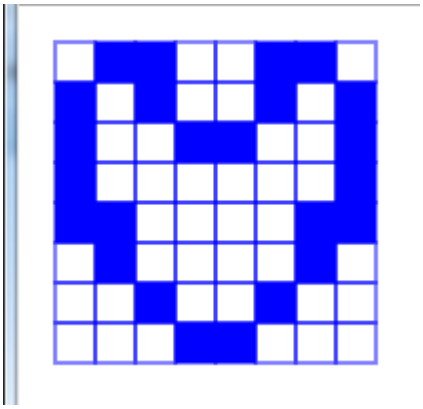
「例題 1-6」8 行 8 列の 2 次元配列 m[][]にハートを示すデータが格納されています。8 行 8 列で矩形を描く際に、配列要素が「1」なら矩形の中を塗り、「0」なら塗らないことにします。

```
<!DOCTYPE html>
<html>
<body>
<canvas id="canvas" width="400" height="400"></canvas>
<script type="text/javascript">
    var canvas = document.getElementById("canvas");
    if(canvas.getContext){
        var context = canvas.getContext("2d");

        var m=new Array(8);
        m[0]=new Array(0,1,1,0,0,1,1,0);
        m[1]=new Array(1,0,1,0,0,1,0,1);
        m[2]=new Array(1,0,0,1,1,0,0,1);
        m[3]=new Array(1,0,0,0,0,0,0,1);
        m[4]=new Array(1,1,0,0,0,0,1,1);
        m[5]=new Array(0,1,0,0,0,0,1,0);
        m[6]=new Array(0,0,1,0,0,1,0,0);
        m[7]=new Array(0,0,0,1,1,0,0,0);

        context.strokeStyle = "blue"; // 輪郭の色
        context.fillStyle = "blue"; // 塗る色
        for (var i=0;i<8;i++){
            for (var j=0;j<8;j++){
                context.strokeRect(20*j+10, 20*i+10, 20, 20);
                if (m[i][j]==1)
                    context.fillRect(20*j+10, 20*i+10, 20, 20);
            }
        }
    }
</script>
```

```
</body>
</html>
```



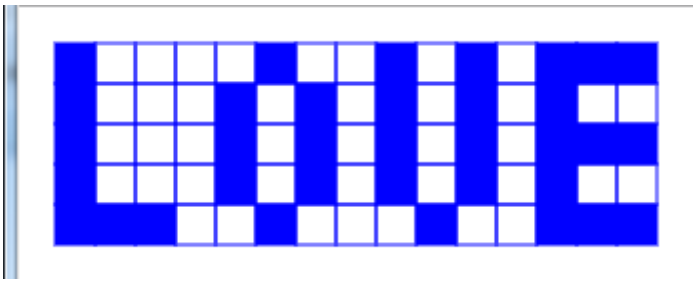
「練習問題 1-6」LOVE を示すデータを 5 行 15 列の 2 次元配列 m[][]に格納しなさい。

```
<!DOCTYPE html>
<html>
<body>
<canvas id="canvas" width="400" height="400"></canvas>
<script type="text/javascript">
    var canvas = document.getElementById("canvas");
    if(canvas.getContext){
        var context = canvas.getContext("2d");

        var m=new Array(5);
        m[0]=new Array(1,0,0,0,0,1,0,0,1,0,1,0,1,1,1);
        m[1]=new Array(1,0,0,0,1,0,1,0,1,0,1,0,1,0,0);
        m[2]=new Array(1,0,0,0,1,0,1,0,1,0,1,0,1,1,1);
        m[3]=new Array(_____①_____);
        m[4]=new Array(_____②_____);

        context.strokeStyle = "blue"; //  輪郭の色
        context.fillStyle = "blue"; //  塗る色
        for (var i=0;i<m.length;i++){
            for (var j=0;j<m[0].length;j++){
```

```
                    context.strokeRect(20*j+10, 20*i+10, 20, 20);
                    if (m[i][j]==1)
                        context.fillRect(20*j+10, 20*i+10, 20, 20);
                }
            }
        }
</script>
</body>
</html>
```



1-7　関数

ある処理単位をひとかたまりにして名前を付け、その名前で呼び出しが行えるようにしたものを関数と呼びます。関数は function を用いて定義します。関数の定義は通常は HTML の<head>部で行いますが、<body>部で行うこともできます。関数名の付け方は変数名と同じです。

```
          ↓関数名
function spc(n)　←関数の定義
{              ↑仮引数。データを受け取る変数
      ・
      ・
      return s;　←呼び出し側に変数 s の値が返される。これを戻り値と呼ぶ。
}
 ・
 ・
 spc(i);　←関数の呼び出し
     ↑実引数。関数に渡すデータ
```

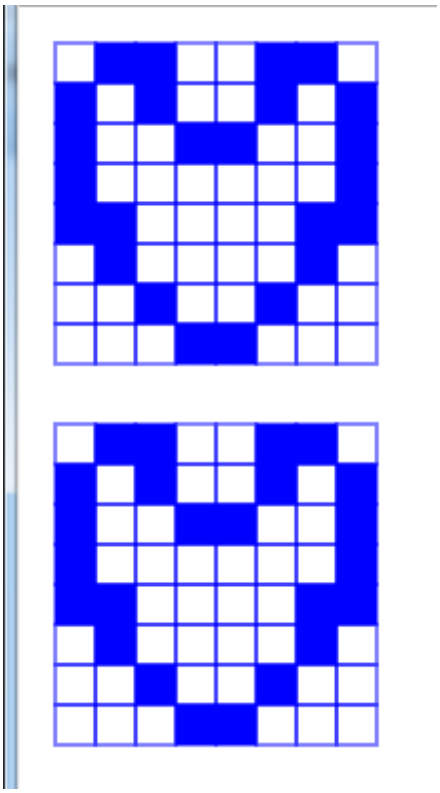
呼び出し側と関数側でデータを授受するものを引数（ひきすう）と呼びます。関数呼び出し側に書く引数を実引数と呼び、変数、定数、式（a+b など）などを指定します。関数の定義側に書く引数を仮引数と呼び、変数を指定します。その際「var」は指定しません。

関数側から値を呼び出し元に返すには return 文を使います。値を返さない場合は return 文は置きません。

「例題 1-7」配列 m[][]に格納されているデータに基づいて矩形を描く関数 disp を作ります。disp の引数は描画コンテキスト context と描画ベース位置の x,y 座標とします。

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript">
    var m=new Array(8);
    m[0]=new Array(0,1,1,0,0,1,1,0);
    m[1]=new Array(1,0,1,0,0,1,0,1);
    m[2]=new Array(1,0,0,1,1,0,0,1);
    m[3]=new Array(1,0,0,0,0,0,0,1);
```

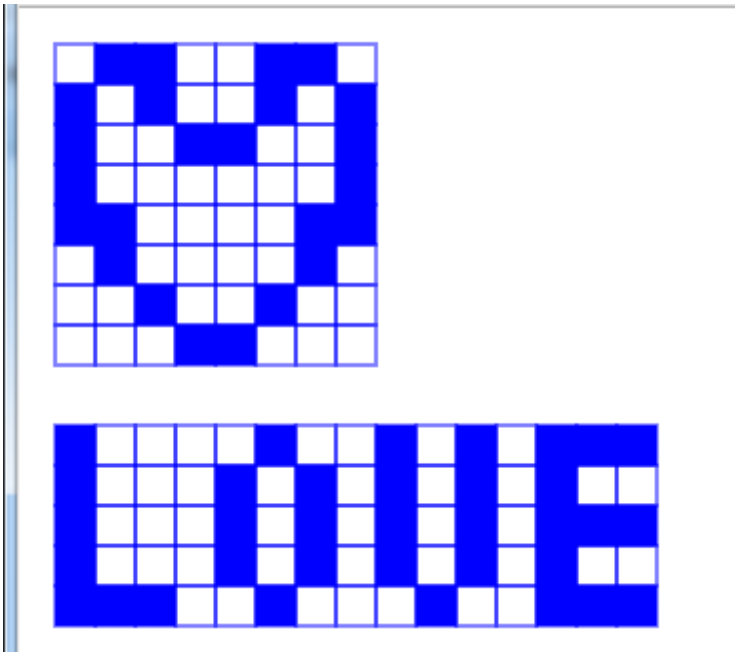
```
    m[4]=new Array(1,1,0,0,0,0,1,1);
    m[5]=new Array(0,1,0,0,0,0,1,0);
    m[6]=new Array(0,0,1,0,0,1,0,0);
    m[7]=new Array(0,0,0,1,1,0,0,0);
    function disp(context,x,y)
    {
        for (var i=0;i<m.length;i++){
            for (var j=0;j<m[0].length;j++){
                context.strokeRect(20*j+x, 20*i+y, 20, 20);
                if (m[i][j]==1)
                    context.fillRect(20*j+x, 20*i+y, 20, 20);
            }
        }
    }
</script>
<body>
<canvas id="canvas" width="400" height="400"></canvas>
<script type="text/javascript">
    var canvas = document.getElementById("canvas");
    if(canvas.getContext){
        context = canvas.getContext("2d");
        context.strokeStyle = "blue"; //  輪郭の色
        context.fillStyle = "blue"; //  塗る色
        disp(context,10,10);
        disp(context,10,200);
    }
</script>
</body>
</html>
```



「練習問題 1-7」異なる 2 次元配列 m1[][]と m2[][]のデータを受け取ってそのデータに基づいて矩形を描くように関数 disp を修正しなさい。

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript">
	var m1=new Array(8);
	m1[0]=new Array(0,1,1,0,0,1,1,0);
	m1[1]=new Array(1,0,1,0,0,1,0,1);
	m1[2]=new Array(1,0,0,1,1,0,0,1);
	m1[3]=new Array(1,0,0,0,0,0,0,1);
	m1[4]=new Array(1,1,0,0,0,0,1,1);
	m1[5]=new Array(0,1,0,0,0,0,1,0);
	m1[6]=new Array(0,0,1,0,0,1,0,0);
	m1[7]=new Array(0,0,0,1,1,0,0,0);
	var m2=new Array(5);
	m2[0]=new Array(1,0,0,0,0,1,0,0,1,0,1,0,1,1,1);
```

```
    m2[1]=new Array(1,0,0,0,1,0,1,0,1,0,1,0,1,0,0);
    m2[2]=new Array(1,0,0,0,1,0,1,0,1,0,1,0,1,1,1);
    m2[3]=new Array(1,0,0,0,1,0,1,0,1,0,1,0,1,0,0);
    m2[4]=new Array(1,1,1,0,0,1,0,0,0,1,0,0,1,1,1);
    function disp(　　①　　)
    {
        for (var i=0;i<m.length;i++){
            for (var j=0;j<m[0].length;j++){
                context.strokeRect(20*j+x, 20*i+y, 20, 20);
                if (m[i][j]==1)
                    context.fillRect(20*j+x, 20*i+y, 20, 20);
            }
        }
    }
</script>
<body>
<canvas id="canvas" width="400" height="400"></canvas>
<script type="text/javascript">
    var canvas = document.getElementById("canvas");
    if(canvas.getContext){
        context = canvas.getContext("2d");
        context.strokeStyle = "blue"; //  輪郭の色
        context.fillStyle = "blue"; //  塗る色
        disp(context,m1,10,10);
        disp(　　②　　);
    }
</script>
</body>
</html>
```



「注」変数のスコープ

変数の使用できる範囲をスコープと言います。関数の中で宣言された変数はその関数の中だけで使用できます。このような変数の使用範囲をローカルスコープと言い、ローカルスコープを持つ変数をローカル変数と言います。

関数の外で宣言された変数はすべての関数で共通に使用できます。このような変数の使用範囲をグローバルスコープと言い、グローバルスコープを持つ変数をグローバル変数と言います。

JavaScript の変数のスコープはローカルスコープとグローバルスコープの 2 種類だけです。C,Java にあるブロックスコープはありません。

```
var a=1; //  グローバルスコープ
function f1()
{
    var i=1; //  ローカルスコープ
}
```

2章　オブジェクトの操作

HTML 要素（タグ）をオブジェクトとして取得し JavaScript から操作する方法や、タグを用いずにコンストラクタから直接オブジェクトを生成し document に追加する方法を説明します。HTML 要素のクリック（タッチ）で発生する onClick イベントや選択ボックスの項目の選択で発生する onChange イベントなどの処理方法を説明します。JavaScript の標準オブジェクトである Math オブジェクトや String オブジェクトの使い方を説明します。

この章ではこれらの内容を以下の各節で説明します。

・オブジェクトの取得
・連想配列と for in 文
・イベント処理
・選択ボックスの処理
・オブジェクトの生成と追加
・Math オブジェクト
・String オブジェクト

2-1　オブジェクトの取得

<canvas>、 、<textarea>、<img>などの HTML 要素（タグ）を JavaScript から操作する方法を説明します。

1.　getElementById メソッド

JavaScript から HTML 要素（タグ）を操作するためには、getElementById メソッドを使って HTML 要素をオブジェクトとして取得します。既に 1 章で説明したように<canvas>要素をオブジェクトとして取得するには、<canvas>要素に id を付け、その id を getElementById メソッドの引数にします。

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

var canvas = document.getElementById("canvas");
```

2.　innerHTML プロパティ

innerHTML プロパティは親要素の HTML テキストを参照します。たとえば 要素の内容を変えるには以下のようにします。

```
<div id="div1">                    ←親要素
ここの内容が変わります             ←HTML テキスト
</div>

var div1=document.getElementById("div1");
div1.innerHTML="<h3>見出し</h3>";
```

「例題 2-1-1」 要素内に「カナダ」の文字とイメージを表示します。

```
<!DOCTYPE html>
<html>
<body>
<div id="div1"></div>
<script type="text/javascript">
    var div1=document.getElementById("div1");
    div1.innerHTML="カナダ<br />";
    div1.innerHTML+="<img src='canada.gif' />";
</script>
</body>
```

</html>

3. value プロパティ

value プロパティは<textarea>タグや<input>タグのテキストを参照します。<textarea>内でのテキストの改行は「¥n」を使います。

「例題 2-1-2」<textarea>に文字を出力します。

```
<!DOCTYPE html>
<html>
<body>
<textarea id="text1" rows="5" cols="40"></textarea><br>
<script type="text/javascript">
    var text1=document.getElementById("text1");
    text1.value="Saki¥nAya¥nRiho¥n";
</script>
</body>
</html>
```

「補足」innerHTML プロパティや value プロパティを使って HTML テキストや単なるテキストを出力する方法以外に以下のような方法もあります。

4. document.write メソッド

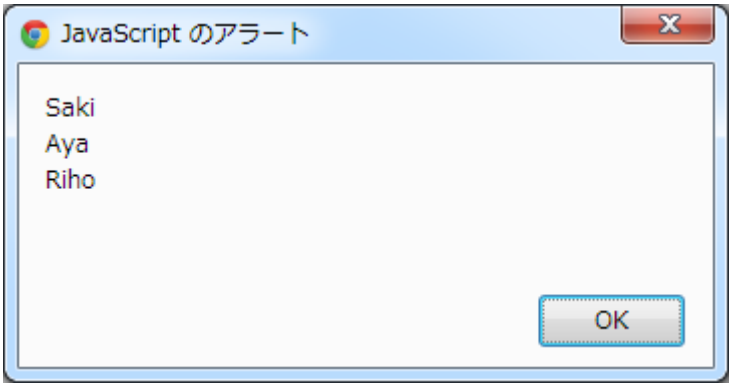
document は Web ページを示すオブジェクト、write は document オブジェクトに対して出力を行うメソッドです。write メソッドの()内には""で囲んだ HTML テキストを指定します。

```
document.write("カナダ<br />");
document.write("<img src='canada.gif' />");
```

5. alert メソッド

window オブジェクトのメソッド alert は、メッセージボックスを開いてテキストを表示します。テキストの改行は「¥n」を使います。window オブジェクトのメソッドは「window.」を省略することができます。

```
alert("Saki¥nAya¥nRiho¥n");
```



2-2　連想配列と for in 文

１．連想配列

連想配列とは、キー（プロパティ）を指定して値をセット出来る配列です。以下は apple,orange,strawberry をキーとする連想配列です。

var fruits = {apple:50,orange:20,strawberry:60};

連想配列の要素はキーを配列の添字として使う fruits["apple"]でも、キーをプロパティとして使う「fruits.apple」でも参照できます。配列の添字として使う場合はキーを「"」で囲みます。

2.　オブジェクトリテラル

オブジェクトリテラルは次のような「キー名：値」をカンマで区切り、全体を｛｝で囲んだものです。

｛キー1：値 1, キー2：値 2,・・・｝

3.　for in 文

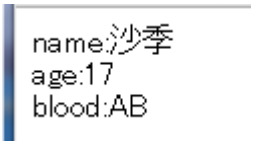
連想配列の全要素を取得するには for in 文を用いて以下のようにします。fruit にキー（プロパティ）が取得できます。

```
for (var fruit in fruits){
    alert(fruit+":"+fruits[fruit]);
}
```

「例題 2-2」name,age,blood をキーとする連想配列を作り、キーと値を表示します。

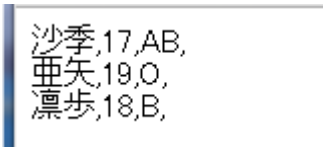
```
<!DOCTYPE html>
<html>
<body>
<div id="result"></div>
<script type="text/javascript">
    var result=document.getElementById("result");
    var girls = {
        name:"沙季",   // 名前
        age:17,        // 年齢
        blood:"AB"     // 血液型
    };
    for (var girl in girls){
        result.innerHTML+=girl+":"+girls[girl]+"<br>";
```

```
    }
</script>
</body>
</html>
```



「練習問題 2-2」name,age,blood をキーとする 3 人分の 2 次元連想配列を作り、キーと値を表示しなさい。2 次元連想配列の作り方は「1-6　2 次元配列」を参照。

```
<!DOCTYPE html>
<html>
<body>
<div id="result"></div>
<script type="text/javascript">
    var result=document.getElementById("result");
    var girls=new Array(3);
    girls[0]={name:"沙季",age:17,blood:"AB"};
    girls[1]={name:"亜矢",age:19,blood:"O"};
    girls[2]={name:"凛歩",age:18,blood:"B"};
    for (var i=0;i<girls.length;i++){
        for (var girl in  ___①___){
            result.innerHTML+= _____②_____ +",";
        }
        result.innerHTML+="<br />";
    }
</script>
</body>
</html>
```



2-3　イベント処理

イベント処理を行うには以下のいずれかの方法でイベントの処理をする関数を指定します。この関数をイベントハンドラとかイベントリスナーと呼びます。

① タグに指定する方法（この節で説明）
② addEventListener による方法（2-5 で説明）
③ プロパティに指定する方法（本書では説明しません）

1.　onClick イベント

要素（タグ）をマウスでクリック（あるいは指でタッチ）したときに onClick イベントが発生します。onClick イベントはパソコンでもタブレット端末でも使用できます。たとえば<img>タグのクリックで clickEvent 関数を呼び出すには以下のようにします。属性名の英大小は区別されませんので onclick としても良いです。指定する関数の引数には「event」を指定します。これは Event オブジェクトを意味します。Event オブジェクトを渡す必要がなければ指定しません。

```
function clickEvent(event)
{
    // 処理内容
}

<img ・・・ onClick="clickEvent(event)" />
```

2.　Event オブジェクト

イベントハンドラの引数 event には Event オブジェクトが渡されます。Event オブジェクトのプロパティを使って各種イベント情報を取得できます。

プロパティ	意味
clientX	マウスのクライアント上の x 座標。
clientY	マウスのクライアント上の y 座標。
target	イベントの発生元のオブジェクト。古い IE では srcElement。

「例題 2-3」3 つの green.png イメージを配置し、クリック（タッチ）したイメージを white.png に変えます。

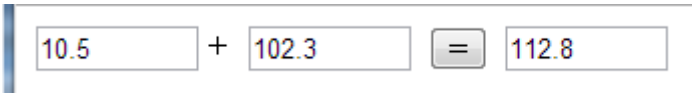
```
<!DOCTYPE html>
<html>
```

```
<head>
<script type="text/javascript">
	function clickEvent(event)
	{
		var obj=event.target; //  または event.srcElement
		obj.src="white.png";
	}
</script>
</head>
<body>
<img src="green.png" onClick="clickEvent(event)" />
<img src="green.png" onClick="clickEvent(event)" />
<img src="green.png" onClick="clickEvent(event)" />
</body>
</html>
```

「練習問題 2-3」ボタンのクリック（タッチ）で 2 つの入力ボックスの内容を加算した値を 3 番目の入力ボックスに表示しなさい。eval 関数は数字文字列を数値に変換します。

```
<!DOCTYPE html>
<head>
<script type="text/javascript">
	function calc()
	{
		var t1=document.getElementById("t1");
		var t2=document.getElementById("t2");
		var t3=document.getElementById("t3");
		  ①  =eval(t1.value)+eval(t2.value);
	}
</script>
</head>
<body>
<input id="t1" type="text" size="8">＋
```

```
<input id="t2" type="text" size="8">
<input type="button" value="＝" ＿＿＿②＿＿＿>
<input id="t3" type="text" size="8">
</body>
</html>
```



2-4　選択ボックスの処理

1.　<select>タグ

選択ボックスは全体を<select>・・</select>で囲み、その中に各項目を<option>で指定します。<option>タグの value アトリビュートに項目の文字列を指定します。選択ボタン（▼)のクリックで選択項目が前の状態から変化すると onChange イベントが発生します。

```
<select onChange="check(this)">
  <option value="img1.gif">イメージ 1

</select>
```

2.　options[]コレクション

選択ボックスの各項目は options[]コレクションで取得できます。同種のオブジェクトを集めたものをコレクションと呼びます。実体は配列です。選択ボックスの選択されている項目の番号（0 スタート）は selectedIndex プロパティで取得できます。選択ボックスの項目の選択で、選択項目を取得するには次のようにします。

```
function check(box)
{              ↓選択ボックスの各項目を示すコレクション
    box.options[box.selectedIndex].value
}                          ↑          ↑値を示すプロパティ
                           選択されている項目の番号
```

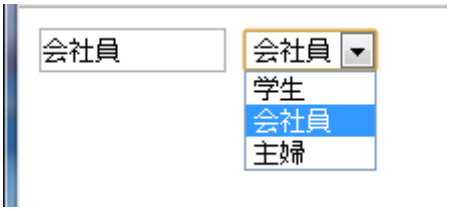
3.　this

this はカレントオブジェクトへの参照を表す予約語です。あるイベントが発生したときに this をイベント処理関数の引数として渡すことにより関数側で、イベントが発生したオブジェクトを参照することができます。

「例題 2-4」選択ボックスで選択した職業を<input>に表示します。

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript">
    function check(box)
    {
        var text=document.getElementById("text");
```

```
        text.value=box.options[box.selectedIndex].value;
    }
</script>
</head>
<body>
<input type="text" id="text" size="10" value="学生" />
<select onChange="check(this)">
<option value="学生">学生</option>
<option value="会社員">会社員</option>
<option value="主婦">主婦</option>
</select>
</body>
</html>
```



「練習問題 2-4」選択ボックスで選択した国の国旗のイメージを表示しなさい。

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript">
    function check(box)
    {
        var img=document.getElementById("img");
        ____①____=box.options[box.selectedIndex].value;
    }
</script>
</head>
<body>
<img id="img" src="canada.gif" />
<select onChange="check(this)">
<option value="canada.gif">カナダ</option>
```

<option value="korea.gif">韓国</option>
<option value="australia.gif">オーストラリア</option>
</select>
</body>
</html>

2-5　オブジェクトの生成と追加

タグを用いずにオブジェクトを生成し document に追加する方法を説明します。

1.　コンストラクタ

コンストラクタはオブジェクト名と同じ名前の特別なメソッドで、new 演算子を使ってオブジェクトのインスタンスを生成します。たとえば Image オブジェクトのインスタンスを生成するには以下のようにします。

```
var img = new Image();
img.src="green.png";
```

2.　appendChild メソッド

上で生成した img を document.body に追加するには appendChild メソッドを使って以下のようにします。

```
document.body.appendChild(img);
```

3.　配置位置

オブジェクトの配置位置を（x,y）位置に設定するには style プロパティを使って以下のようにします。

```
img.style.position="absolute";
img.style.left=x+"px";
img.style.top=y+"px";
```

4.　addEventListener メソッド

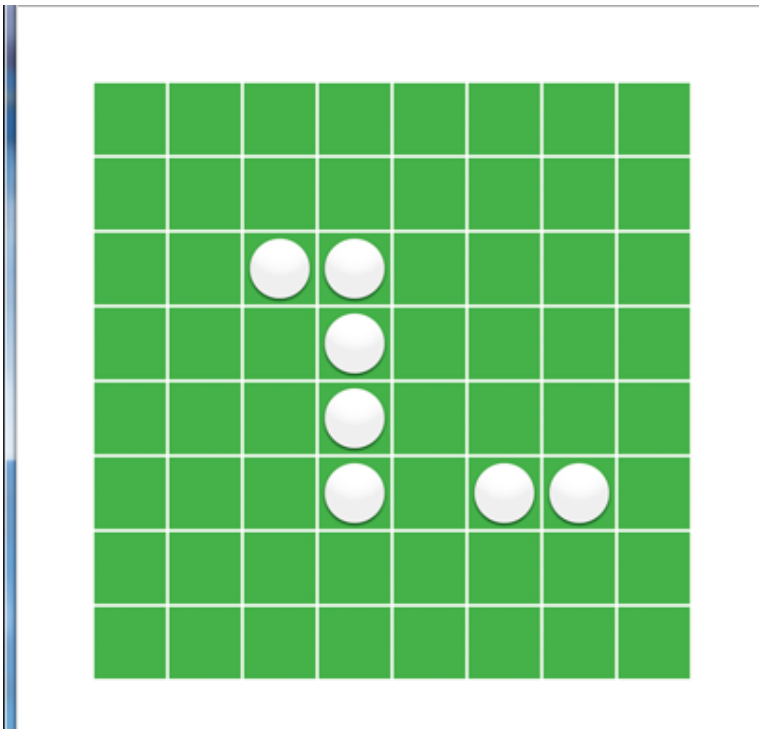
タグのイベント属性にイベントハンドラを指定する代わりに addEventListener メソッドを使ってイベントハンドラを登録することができます。たとえば img に onClick イベントハンドラを指定するには以下のようにします。指定するイベント名は「on」を抜いた「click」です。英大小は区別されません。

```
img.addEventListener("click",clickEvent);
```

「例題 2-5」8×8 の green.png オブジェクトを生成し、document に追加します。それぞれ同じ onClick イベントハンドラを指定します。クリック（タッチ）したイメージの内容を white.png に変更します。

```
<!DOCTYPE html>
<html>
<body>
```

```
<script type="text/javascript">
    function clickEvent(event)
    {
        var obj=event.srcElement;
        obj.src="white.png";
    }
    for (var i=0;i<8;i++){
        for (var j=0;j<8;j++){
            var img=new Image();
            img.src="green.png";
            img.id="img"+(i+j+1);
            img.style.position="absolute";
            img.style.left=j*40+40+"px";
            img.style.top=i*40+40+"px";
            img.addEventListener("click",clickEvent);
            document.body.appendChild(img);
        }
    }
</script>
</body>
</html>
```



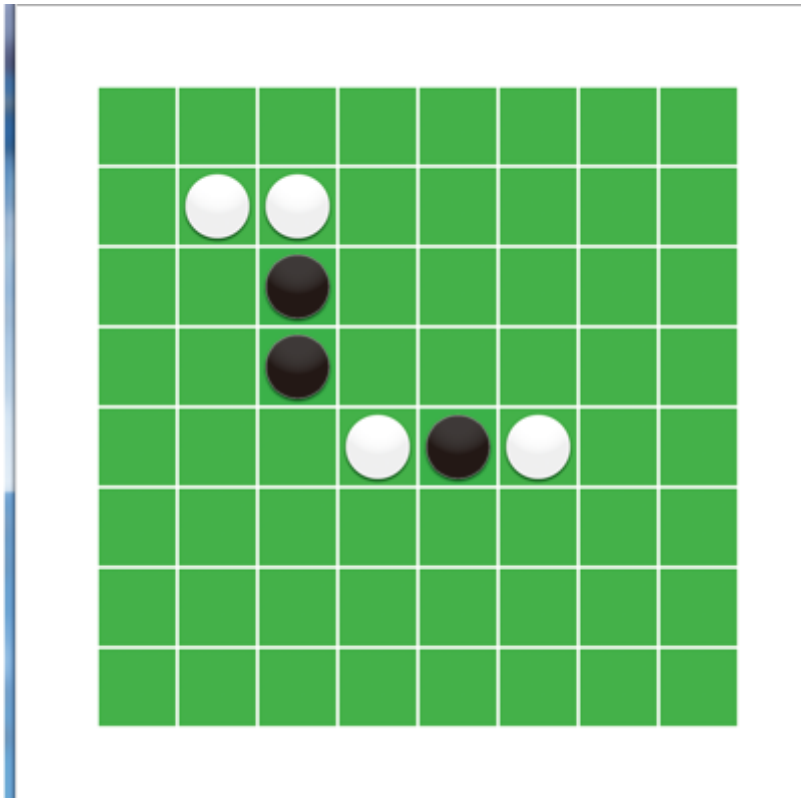
「注」appendChild で追加するのではなく document.write で書きだす方式にする場合は以下のようになります。

```
    for (var i=0;i<8;i++){
        for (var j=0;j<8;j++){
            document.write("<img src='green.png'"+" id='img"+(i+j+1)+
"' style='position:absolute;left:"+(j*40+40)+"px;top:"+(i*40+40)+
"px' onclick='clickEvent(event);' />");
        }
    }
```

「練習問題 2-5」クリック（タッチ）したイメージの内容が「green.png」または「black.png」の場合は「white.png」に置き換え、そうでない場合は「black.png」に置き換えなさい。match メソッドは文字列内に引数で指定した文字列を含むか調べます。

```
<!DOCTYPE html>
<html>
<body>
<script type="text/javascript">
    function clickEvent(event)
    {
        var obj=event.srcElement;
        if (obj.src.match("green.png") || obj.src.match("black.png"))
            ①
        else
            ②
    }
    for (var i=0;i<8;i++){
        for (var j=0;j<8;j++){
            var img=new Image();
            img.src="green.png";
            img.id="img"+(i+j+1);
            img.style.position="absolute";
            img.style.left=j*40+40+"px";
            img.style.top=i*40+40+"px";
            img.addEventListener("click",clickEvent);
            document.body.appendChild(img);
        }
```

```
    }
</script>
</body>
</html>
```



2-6　Math オブジェクト

JavaScript には Array,Date,Math,Number,String などの標準オブジェクトがあります。

1.　Math オブジェクト

Math オブジェクトは sin、cos、sqrt などの数値計算を行なうためのオブジェクトです。Math オブジェクトのメソッドは静的メソッドとして定義されていますので、ユーザは Math オブジェクトを生成せずに予約オブジェクト名の Math を用いて次のようにメソッドを使用します。これで、√10 の結果が得られます。

Math.sqrt(10);

2.　三角関数

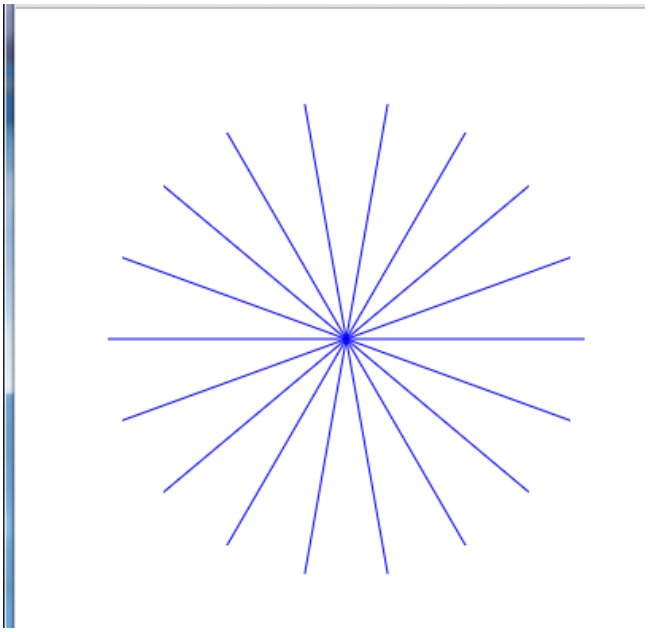
sin,cos などの三角関数の引数の単位は度でなくラジアンです。ラジアンは 1 単位円の角度に対する円弧の長さです。1 単位円の半周の円弧の長さはπでこれが 180 度に相当しますので、x 度をラジアンに変換するには以下のようにします。PI は Math オブジェクトのプロパティでπの値を示します。

x*Math.PI/180

「例題 2-6」中心(200,200),半径 150 の円周上の点を 20° ごとに中心と直線で結びます。

```
<!DOCTYPE html>
<html>
<body>
<canvas id="canvas" width="400" height="400"></canvas>
<script type="text/javascript">
    var canvas = document.getElementById("canvas");
    if(canvas.getContext){
        var context = canvas.getContext("2d");
        context.strokeStyle = "blue"; //  青
        for (var i=0;i<360;i+=20){
            var x=200+150*Math.cos(i*Math.PI/180);
            var y=200+150*Math.sin(i*Math.PI/180);
            context.beginPath();
            context.moveTo(200,200);
            context.lineTo(x,y);
            context.stroke();
        }
```

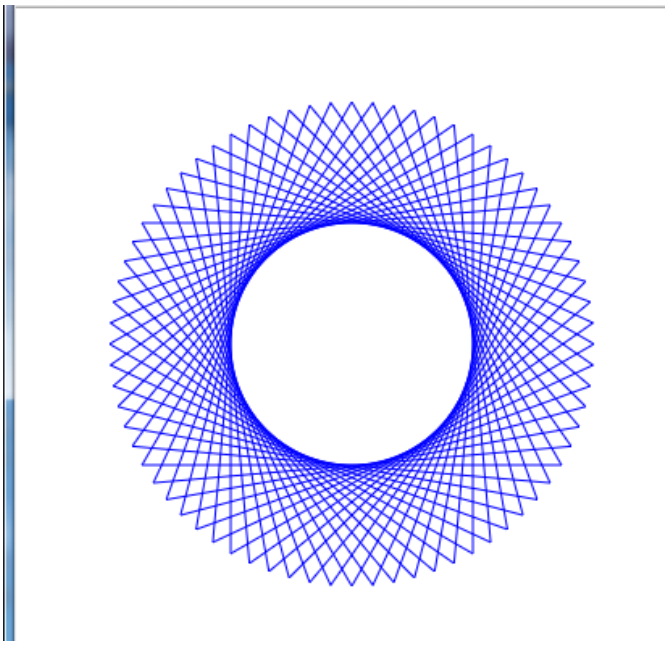
```
    }
</script>
</body>
</html>
```



「練習問題 2-6-1」中心(200,200),半径 150 の円周上の 120° 離れた 2 点を 5° 置きに結びなさい。

```
<!DOCTYPE html>
<html>
<body>
<canvas id="canvas" width="400" height="400"></canvas>
<script type="text/javascript">
    var canvas = document.getElementById("canvas");
    if(canvas.getContext){
        var context = canvas.getContext("2d");
        context.strokeStyle = "blue"; //  青
        for (var i=0;i<360;i+=5){
            var x1=200+150*Math.cos(i*Math.PI/180);
            var y1=200+150*Math.sin(i*Math.PI/180);
            var x2=200+150*_____①_____*Math.PI/180);
            var y2=200+150*_____②_____*Math.PI/180);
            context.beginPath();
```

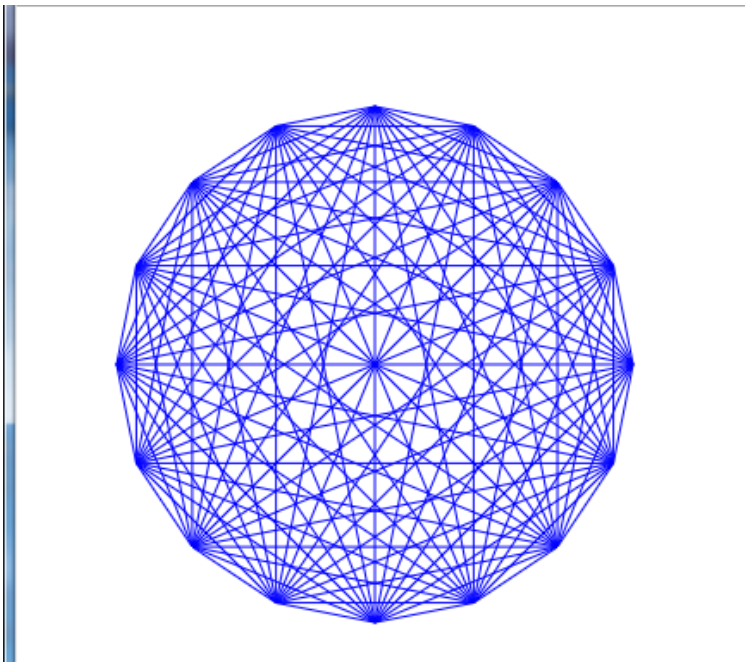
```
                context.moveTo(x1,y1);
                context.lineTo(x2,y2);
                context.stroke();
            }
        }
</script>
</body>
</html>
```



「練習問題 2-6-2」 中心(200,200),半径 150 の円周上を n 分割した各点を総当たりで結びなさい。

```
<!DOCTYPE html>
<html>
<body>
<canvas id="canvas" width="400" height="400"></canvas>
<script type="text/javascript">
    var canvas = document.getElementById("canvas");
    if(canvas.getContext){
        var context = canvas.getContext("2d");
        context.strokeStyle = "blue"; //  青
        var n=16; //  分割数
        for (var i=0;i<n-1;i++){
```

```
                var x1=200+150*Math.cos(i*2*Math.PI/n);
                var y1=200-150*Math.sin(i*2*Math.PI/n);
                for (var j=i+1;j<n;j++){
                    var x2=200+150*Math.cos(j*2*Math.PI/n);
                    var y2=200-150*Math.sin(j*2*Math.PI/n);
                    context.beginPath();
                    _____①_____
                    _____②_____
                    context.stroke();
                }
            }
        }
</script>
</body>
</html>
```



2-7　String オブジェクト

1.　String オブジェクト

String オブジェクトは、文字列の変数への代入あるいは String コンストラクタによる明示的な宣言のどちらかで生成できます。

```
var s="hello";
```

または

```
var s=new String("hello");
```

この文字列オブジェクトに対し、toUpperCase メソッドを使って

```
p=s.toUpperCase();
```

とすると、p には"HELLO"が得られます。

また length プロパティを使えば文字列の長さを調べることができます。

```
n=s.length;
```

とすると、n には 5 が得られます。英数文字も日本語も 1 文字として扱います。

2.　文字列の連結

文字列を含む式の中の＋演算子は文字列連結演算子として機能します。

“Hello”+”JavaScript”

は”HelloJavaScript”となります。文字列と数値を＋演算子でつないだ場合、数値は文字列に変換されて連結されます。

“5”+5

は数値の 5 が文字列の”5”に変換されて連結されるので、”55”となります。ただし、5+5+"5" のような場合、5+5 が先に 10 と計算され、それが文字列に変換されて連結されるので、"105" となります。数値演算を明示するには(5+5)+"5"のようにかっこで囲みます。

変数名と変数の内容を表す文字列を作るには次のようにします。

```
var sum=100;
“sum=”+sum
```

は、”sum=”という文字列に変数 sum の内容の 100 が文字列に変換されて連結されるので、”sum=100”という文字列になります。

3.　文字列の比較

文字列の比較は「==」演算子を用いて行うことができます。

```
var name="Riho";
if (name=="Riho")
    alert(name);
```

4. 部分文字列の取り出し

文字列中から指定した文字列を取り出すメソッドとしてcharAt,charCodeAt,substring,slice があります。substring と slice の違いは引数に負数を与えた場合の解釈だけです。

var s="JavaScript";
s.charAt(1);・・・"a"。１番目の文字
s.charCodeAt(1);・・・97。１番目の文字の文字コード（ASCII コード）
s.substring(4,6);・・・"Sc"。4 番目～5 番目の文字列
s.substring(4);・・・"Script"。4 番目～後端までの文字列
s.slice(4,6);・・・"Sc"。4 番目～5 番目の文字列
s.slice(1,-1);・・・"avaScrip"。１番目の文字～後ろから１番目の文字列

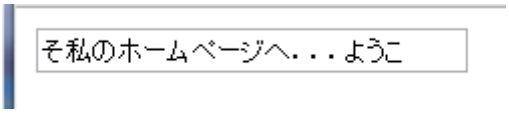
5. 一定時間の処理

指定間隔である処理を行うには setInterval メソッドを使います。

```
setInterval(function(){
    // ある処理
},指定ミリ秒);
```

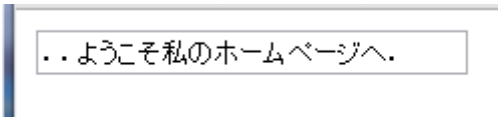
「例題 2-7」msg の文字列を左スクロールして表示します。msg の 1 文字目とそれ以後を入れ替える処理を 500 ミリ秒間隔で行います。

```
<!DOCTYPE html>
<html>
<body>
<input type "text" id="result" size="30" />
<script type="text/javascript">
    var result=document.getElementById("result");
    var msg="ようこそ私のホームページへ. . .  ";
    setInterval(function() {
        result.value=msg;
        msg=msg.substring(1,msg.length)+msg.substring(0,1);
    },500);
</script>
</body>
</html>
```



「練習問題 2-7」msg の文字列を右スクロールして表示しなさい。msg の最後の文字目とそれ以前を入れ替える処理を 500 ミリ秒間隔で行います。

```
<!DOCTYPE html>
<html>
<body>
<input type "text" id="result" size="30" />
<script type="text/javascript">
    var result=document.getElementById("result");
    var msg="ようこそ私のホームページへ. . .  ";
    setInterval(function() {
        result.value=msg;
        var n=      ①      
        msg=msg.substring(n-1)+          ②          
    },500);
</script>
</body>
</html>
```



練習問題解答

「練習問題 1-2」
①context.moveTo(x, 20);
②context.lineTo(x, 200);

「練習問題 1-3」
①x%20==0

「練習問題 1-4」
①j==1 || i==j || j==8

「練習問題 1-5」
①50+n*4+10*x[i];
②150+n*4-10*y[i];

「練習問題 1-6」
①1,0,0,0,1,0,1,0,1,0,1,0,1,0,0
②1,1,1,0,0,1,0,0,0,1,0,0,1,1,1

「練習問題 1-7」
①context,m,x,y
②context,m2,10,200

「練習問題 2-2」
①girls[i]
②girls[i][girl]

「練習問題 2-3」
①t3.value
②onClick="calc()"

「練習問題 2-4」
①img.src

「練習問題 2-5」

①obj.src="white.png";

②obj.src="black.png";

「練習問題 2-6-1」

①Math.cos((i+120)

②Math.sin((i+120)

「練習問題 2-6-2」

①context.moveTo(x1,y1);

②context.lineTo(x2,y2);

「練習問題 2-7」

①msg.length;

②msg.substring(0,n-1);

JavaScript 超入門
2013 年 9 月 1 日　初版　第 1 刷発行
著者＝河西 朝雄
発行者＝河西 朝雄
発行所＝カサイ．ソフトウエアラボ
長野県茅野市ちの 813　TEL.0266-72-4778