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。

・変数と演算子

・for文

・if else文

・二重ループ

・1次元配列

・2次元配列

・関数

・その他の制御文

while文、else if文、switch case文、連想配列とfor in文

1-1　JavaScriptの概要

1.　JavaScriptとは

JavaScriptはHTML内に記述し、Webページを動的に制御するためのスクリプト言語です。JavaScriptの特徴は以下の通りです。

・JavaScriptでプログラムを作る上で必用なソフトはメモ帳などのエディタとInternet ExplorerやChromeなどのブラウザだけで良い。

・HTMLの<script>ブロック内にJavaScriptプログラムを記述する。

・プログラムの書式や言語仕様はC言語と極めて近い。

・Javaという名称がついているが、オブジェクトの扱いはJavaとは全く異なる。

・JavaScriptで扱えるオブジェクトを大別すると、JavaScriptで規定している標準オブジェクト（コアオブジェクト）、W3Cで規定しているHTMLやXMLで記述された各要素を取り扱うためのDOMオブジェクト、明確な規定はないが多くのブラウザで採用されているブラウザ関連のBrowserオブジェクトの3種類となる。

・HTMLタグ中のonXXX属性にJavaScript関数を指定してマウス操作、フォーム要素のチェックなどの各種イベント処理を行なうことができる。addEventListenerメソッドを使ってイベントリスナーを追加することもできる。

・Styleオブジェクトを使ってオブジェクトの位置やサイズを制御できる。

・キャンバスオブジェクトに対しグラフィックス処理ができる。

・ビデオ、オーディオ、マップなどのオブジェクトを操作できる。

・マウスのドラッグ＆ドロップ処理ができる。

・スマートフォンやタブレットでのタッチ処理やマルチタッチ処理ができる。

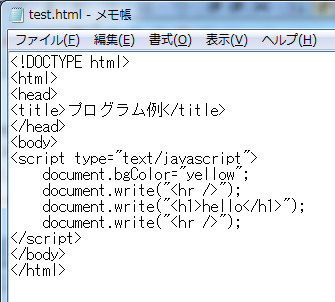
・スマートフォンやタブレットに内蔵されているGeolocation(ジオロケーション)やセンサーを操作できる。

「注」スクリプト言語

スクリプト言語とはメインになる言語の中に埋め込んで、機能を補完したり拡張したりするための言語です。

2.　JavaScriptの書式

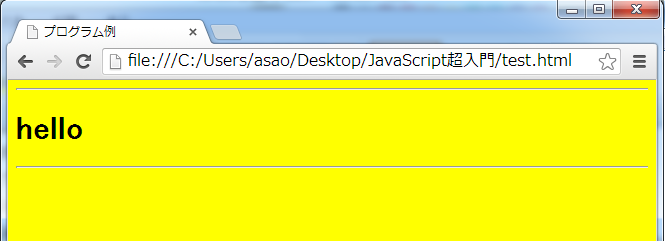
　JavaScriptはHTMLファイル内に<script>・・・</script>タグで囲んだスクリプトブロック内に記述します。JavaScriptで水平線と「hello」という文字をブラウザに表示するプログラムを「test.html」としてメモ帳で編集した例を以下に示します。



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。スクリプトブロックは<head>部または<body>部に置くことができます。

このJavaScriptプログラムにおいてdocumentはこのスクリプトブロックが置かれているHTMLファイルを表すオブジェクトです。bgColorはdocumentオブジェクトの背景色を示すプロパティです。writeは（）内の内容をdocumentオブジェクトに出力するメソッドです。（）内にはダブルクオーテーションで囲んだ"<h1>hello</h1>"のようなHTML文を書きます。JavaScriptの文の終わりにはセミコロン（；）を置きます。

このプログラム「test.html」をブラウザで見ると次のようになります。



3.　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。先の例ではdocumentというオブジェクトに対しwriteメソッドやbgColorプロパティを使用しています。

　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　グラフィックス処理の概要

HTML5の<canvas>要素（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>要素（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。矩形の内部を塗りつぶすにはfillRectメソッドを使います。

|  |  |
| --- | --- |
| 矩形の描画メソッド | 機能 |
| 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。 |
| context.fillRect(x,y,w,h) | 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。 |
| context.clearRect(x,y,w,h) | 左上隅座標を(x,y)、幅をw、高さをhとする矩形内部を背景色（デフォルトで白）でクリア。 |

「注」領域のクリア

「context.clearRect(x,y,w,h)」を行うと指定した四角の領域が背景色（デフォルトで白）でクリアされます。ブラウザによってはbeginPathでパスに設定しなかった図形（イメージなど）はclearRectで消去できない場合があります。この場合は白でfillRectします。

context.fillStyle="white";

context.fillRect(x,y,w,h);

「例題1-2-1」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

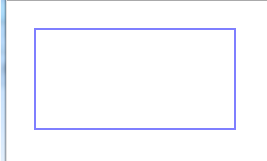
context.strokeRect(20, 20, 200, 100);

}

</script>

</body>

</html>



「練習問題1-2-1」赤の四角を追加しなさい。

<!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"; // 青

context.strokeRect(20, 20, 200, 100);

context.strokeStyle = ① ; // 赤

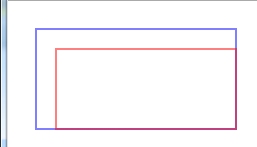
② ;

}

</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。

|  |  |
| --- | --- |
| パスに対し直線を描くメソッド | 機能 |
| context.moveTo(x,y) | 描画現在位置を(x,y)に移動。 |
| context.lineTo(x,y) | 描画現在位置から(x,y)に直線を描く。 |

「例題1-2-2」(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);

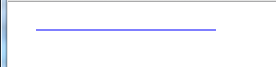
context.stroke(); // パスの描画

}

</script>

</body>

</html>



「練習問題1-2-2」三角形を描きなさい。

<!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"; // 描画色

context.beginPath(); // パスの開始

context.moveTo(0, 200);

context.lineTo(200, 200);

① ;

② ;

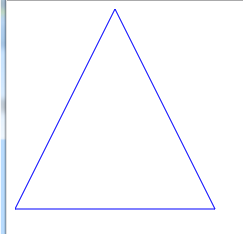
context.stroke(); // パスの描画

}

</script>

</body>

</html>



6.　円と円弧

円と円弧は直線の場合と同様にarcメソッドで一度パスに描画してから、strokeあるいはfillメソッドでキャンバスに描画します。

|  |  |
| --- | --- |
| 円と円弧の描画メソッド | 機能 |
| context.arc(x,y,r,start,end,clockwise) | 中心(x,y)、半径rの円（円弧）を角度start～end（ラジアン）まで描きます。clockwiseにtrueを指定すると反時計回り、falseを指定すると時計回りで円弧を描きます。 |

「例題1-2-3」円、円弧、円弧の内部を塗った図形を描きます。

<!DOCTYPE html>

<html>

<body>

<canvas id="canvas" width="600" 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 = "violet"; // 塗る色

context.beginPath();

context.arc(100, 100, 60, 0, Math.PI\*2, false);

context.stroke();

context.beginPath();

context.arc(250, 100, 60, 10 \* Math.PI / 180, 80 \* Math.PI / 180, true);

context.stroke();

context.beginPath();

context.arc(400, 100, 60, 10 \* Math.PI / 180, 80 \* Math.PI / 180, true);

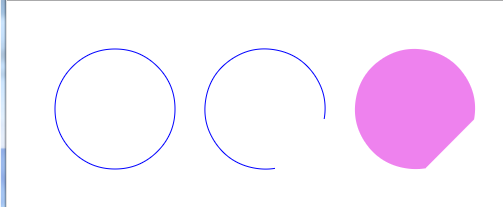
context.fill();

}

</script>

</body>

</html>



「練習問題1-2-3」半径100の円を1つ描き、その中に半径50の半円を2つ描きなさい。

<!DOCTYPE html>

<html>

<body>

<canvas id="canvas" width="600" 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.beginPath();

context.arc(150, 150, 100, 0, Math.PI\*2, false);

context.stroke();

context.beginPath();

① ;

context.stroke();

context.beginPath();

② ;

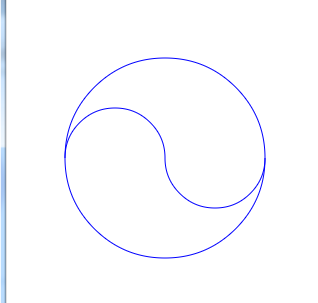
context.stroke();

}

</script>

</body>

</html>



7.　テキストの描画

テキストの描画はstrokeTextメソッドまたはfillTextメソッドで行います。

|  |  |
| --- | --- |
| テキストの描画メソッド | 機能 |
| context.strokeText(テキスト,x, y) | 輪郭で描画します。 |
| context.fillText(テキスト,x, y) | 塗りつぶしで描画します。 |

描画位置の(x,y)はテキストの左下隅の座標です。fontプロパティにフォントサイズ、書体（italic/bold）、フォント名などを空白で区切って指定します。フォント名に空白が含まれる場合はフォント名を(')で囲みます。

context.font = "36px 'ＭＳ Ｐゴシック'";

「例題1-2-4」(10,40)位置に「ストロークテキスト」というテキストを青色でフォントサイズ32px、フォント名「ＭＳ Ｐゴシック」のフォントで描きます。

<!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";

context.font = "32px 'ＭＳ Ｐゴシック'";

context.strokeText("ストロークテキスト", 10, 40);

}

</script>

</body>

</html>



「練習問題1-2-4」3人の武将の名前を描きなさい。

<!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";

context.font = "32px 'ＭＳ Ｐゴシック'";

context.strokeText("織田信長", 10, 40);

① ;

context.strokeText("徳川家康", 10, 120);

}

</script>

</body>

</html>



「補足」fillTextでテキストを描く場合、フォントサイズの大きいものはstrokeTextとfillTextを併用した方がきれいに描けますが、フォントサイズが小さいものはfillTextのみの方がきれいに描けます。

<!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";

context.fillStyle = "blue";

context.font = "32px 'ＭＳ Ｐゴシック'";

context.strokeText("フィルテキスト1", 10, 40);

context.fillText("フィルテキスト1", 10, 40);

context.font = "16px 'ＭＳ Ｐゴシック'";

context.strokeText("フィルテキスト2", 10, 80);

context.fillText("フィルテキスト2", 10, 80);

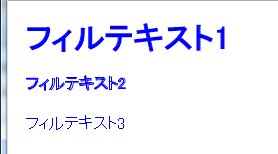
context.fillText("フィルテキスト3", 10, 120);

}

</script>

</body>

</html>



1-3　変数と演算子

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。

1.　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。以下は変数名が「i」の変数を宣言しています。

var i;

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。

2.　変数の初期化

　変数を宣言したままだと、その変数の内容は不定で、0になっている保証はありません。そこで、変数は初期化を行う必要があります。

　変数sの宣言時に0で初期化するには次のようにします。

var s=0;

　変数の宣言と代入を分けて、次のようにすることもできます。

var s;

s=0;

　プログラム中で一度だけ初期化を行えば良いのであれば前者を、複数の場所でそのつど初期化が必要な場合には後者を使います。

複数の変数を宣言時に初期化するには次のようにします。

var sum=0,ave=0;

3.　変数名の付け方

　変数に付ける名前を変数名と呼びます。変数名の名前付けのおおよその規則は以下のようなものです。

・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

・変数の役割を連想できる変数名を使う。合計ならsumなど。

4.　変数への代入と更新

変数へのデータの代入を行う場合、左辺には１つの変数しか指定できませんが、右辺には、定数、変数、これらを演算子で結んだ式を指定することができます。

var a,b,c,x=10,y=20;

a=1;

b=x;

c=x+y+30;

これで、aに1、bに10、cに60が入ります。

プログラムでは次のような表現を良く使います。

sum=sum+10;

これを数学の等式と考えるとおかしなことになりますが、プログラムでは、右辺のsum+10を行なった結果の値を左辺の変数sumに代入するという意味になります。

従って、以下を行なえば、sumの値は最初1ですが、sum+10で11となり、その値がsumに代入されるので、sumの値は11になります。

sum=1;

sum=sum+10;

「〇＝〇＋□」のように左辺と右辺に同じ変数〇がある場合は変数の内容に□を足した値が新しい変数の値になります。このように変数の内容に値を足すなどして新しい値にすることを、変数の更新といいます。

sum=sum+10;

は

sum+=10;

と書くことができ、+=を複合代入演算子といいます。本書では後者の使い方をします。+=の他に-=、\*=、/=などが使えます。

5.　算術演算子

変数sumにいくつかのデータの合計を求めるには、「sum=103+1020+・・・+5020;」のように書いておけば、あとはコンピュータが計算結果を求めて、変数sumに代入してくれます。ここで、足し算をしている「+」を演算子と呼びます。計算の基本は、「足す」、「引く」、「掛ける」、「割る」の４種類で、これを行なう演算子を算術演算子（加減乗除演算子）と呼びます。この他に余りを求める演算子と負符号を示す演算子があります。これらの演算子は次の記号を使います。

|  |  |
| --- | --- |
| 演算子 | 意味 |
| + | 加算 |
| - | 減算 |
| \* | 乗算 |
| / | 除算 |
| % | 余り(剰余) |
| - | 負符号 |

数学の記号では加算の「＋」と減算の「－」は同じですが、乗算の「×」が「\*」、除算の「÷」が「/」となります。

演算処理を行なう際には演算子の優先順位があります。

sum=10+20\*30;

という計算式があったとき、「\*」の優先順位は「+」より高いと決めているので、「20\*30」が先に計算され、次にその結果と10が加算され、最後に「=」演算子でsumに代入されます。「=」演算子は一番優先順位が低いのです。これはうまいルールなのです。もし「=」の優先順位が他の演算子より高かったら「sum=10」が先に行なわれてしまいます。こうしたルールを作っておくことで、人間の常識とある程度一致した記述ができるのです。

ところで、どうしても優先順位を変えたければ、（）を使って

sum=(10+20)\*30;

とします。これで10+20が先に計算されます。以下のような同じ優先順位なら、左から演算されます。

a+b+c

「例題1-3」変数aとbの加算、減算、乗算、除算、剰余の結果を表示します。

<!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");

context.strokeStyle = "blue";

context.font = "32px 'ＭＳ Ｐゴシック'";

var a=100,b=30,tasu,hiku,kakeru,waru,amari;

tasu=a+b;

hiku=a-b;

kakeru=a\*b;

waru=a/b;

amari=a%b;

context.strokeText("a+b="+tasu, 10, 40);

context.strokeText("a-b="+hiku, 10, 80);

context.strokeText("a\*b="+kakeru, 10, 120);

context.strokeText("a/b="+waru, 10, 160);

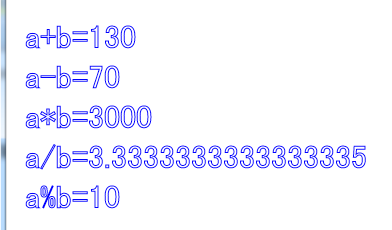
context.strokeText("a%b="+amari, 10, 200);

}

</script>

</body>

</html>



「練習問題1-3」変数のtasu,hiku,kakeru,waru,amariを使わずにstrokeTextメソッドの引数に直接、演算式を指定しなさい。

<!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";

context.font = "32px 'ＭＳ Ｐゴシック'";

var a=100,b=30;

context.strokeText("a+b="+(a+b), 10, 40);

context.strokeText("a-b="+(a-b), 10, 80);

context.strokeText("a\*b="+(a\*b), 10, 120);

context.strokeText("a/b="+ ① , 10, 160);

context.strokeText("a%b="+ ② , 10, 200);

}

</script>

</body>

</html>

1-4　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文や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。

ループ変数は、

var i;

for (i=0;i<=10;i++)

のように通常の変数のように宣言することもできますが、for文の中で宣言することもできます。

for (var i=0;i<=10;i++)

□for文の例

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.　増分演算子、減分演算子

　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。++を増分演算子（インクリメント演算子）、--を減分演算子（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。

「例題1-4」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

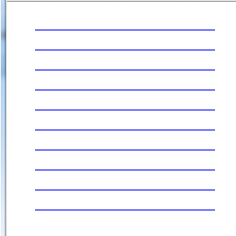
}

}

</script>

</body>

</html>



「練習問題1-4」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();

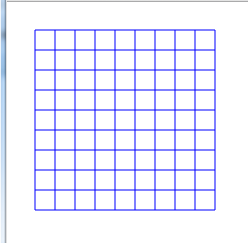
}

}

</script>

</body>

</html>



1-5　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節に書くものがなければelse節全体を省略します。{ }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.　真偽の判定

　真偽値は真をtrue、偽をfalseという定数を使って表します。真偽値を扱う変数flagは次のように宣言します。

var flag=false;

　変数flagの真偽値を反転するには以下のようにします。

flag=!flag;

変数flagの値がtrueかを判定するには、

if (flag==true)

としますが、これは一般に以下のように簡略した記述ができます。

if (flag)

5.　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1-5」A点を中心に半径50、角度120°の円弧を下側に描き、A点より半径分（50）

離れたB点を中心に角度120°の円弧を上側に描きます。この操作を繰り返すことでしめ縄模様が描けます。描く円弧の色を変数flagを使って赤と青の交互に描きます。

「図1-1　しめ縄模様」

<!DOCTYPE html>

<html>

<body>

<canvas id="canvas" width="600" height="400"></canvas>

<script type="text/javascript">

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

if(canvas.getContext){

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

context.strokeStyle = "blue";

var flag=false;

for (var x=50;x<=600;x+=50){

context.beginPath();

context.arc(x, 100, 50, 0,120\*Math.PI/180, false);

context.stroke();

if (flag)

context.strokeStyle = "blue"; // 青

else

context.strokeStyle = "red"; // 赤

context.beginPath();

context.arc(x+50, 100, 50, 180\*Math.PI/180,300\*Math.PI/180, false);

context.stroke();

flag=!flag;

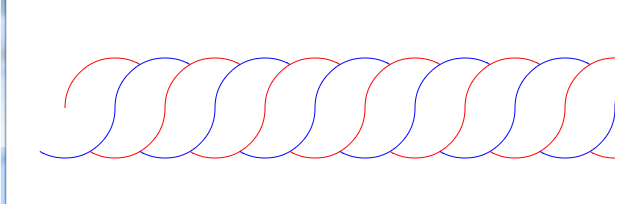
}

}

</script>

</body>

</html>



「練習問題1-5」円と四角を交互に描きなさい。

<!DOCTYPE html>

<html>

<body>

<canvas id="canvas" width="600" height="400"></canvas>

<script type="text/javascript">

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

if(canvas.getContext){

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

context.strokeStyle = "blue";

var flag=false;

for (var x=0;x<=400;x+=80){

if (flag){

① ;

}

else {

context.beginPath();

② ;

context.stroke();

}

flag=!flag;

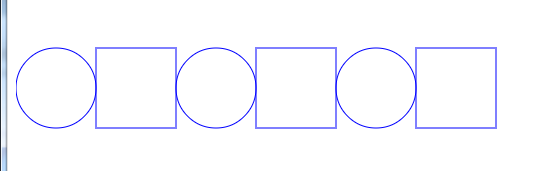
}

}

</script>

</body>

</html>



1-6　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。

for (var i=1;i<=2;i++){

for (var 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1-6」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);

}

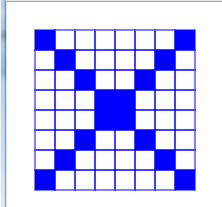
}

}

</script>

</body>

</html>



「練習問題1-6」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);

}

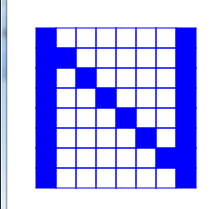
}

}

</script>

</body>

</html>



1-7　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。

1.　1次元配列の宣言

　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。配列の基底（先頭）要素の添字は0スタートです。

配列は配列名と添字を使って管理します。

　　a[i]

↑添字。配列要素の番号

　　↑配列名

a[0]～a[4]の要素に0を入れるにはfor文を使って以下のようにします。

for (i=0;i<5;i++)

a[i]=0;

2.　lengthプロパティ

配列の長さ（要素数）はlengthプロパティで取得できます。「for (i=0;i<5;i++)」のような特定の値を使うと配列のサイズが変わった場合は「5」という値を変更しなければなりませんが、以下のように記述しておけば、このよな変更は必要ありません。

for (i=0;i<a.length;i++)

a[i]=0;

3.　初期化

　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");

「注」変数名にnameは使えますが、配列名にnameを使うとおかしな結果となります。以下のようにした場合、name[0]は"結衣"でなく"結"の1文字となってしまいます。

var name=new Array("結衣","彩香","理沙");

「例題1-7」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1-7」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();

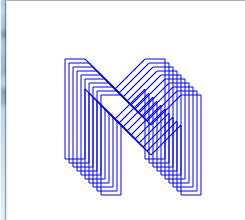
}

}

</script>

</body>

</html>



1-8　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。

1.　2次元配列の宣言

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

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

2.　2次元配列の参照

　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;

}

}

3.　2次元配列の行要素数と列要素数

　2次元配列a[ ][ ]の行要素数はa.length、列要素数はa[0].lengthで取得できます。従って3,4などの定数を使わずに以下のように記述できます。

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

for (j=0;j<a[0].length;j++){

a[i][j]=0;

}

}

4.　初期化

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うことではありませんが、数学で行列を扱うときにａij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1-8」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<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);

}

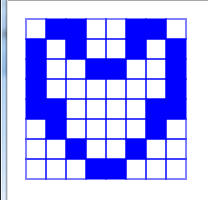
}

}

</script>

</body>

</html>



「練習問題1-8」LOVEを示すデータを5行15列の2次元配列m[ ][ ]に格納し、このデータを元に文字を描きなさい。

<!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");

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);

}

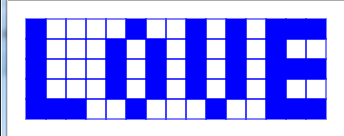
}

}

</script>

</body>

</html>



1-9　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。

1.　関数の定義と呼び出し

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文を置きません。

2.　配列の引数

　関数に配列全体を渡すには実引数、仮引数に配列名のみ指定します。

function abc(x)

{

x[0]～x[4]で呼び出し側の配列a[0]～a[4]が参照できる

x.lengthで配列のサイズを取得できる

}

・

・

var a=new Array(10,20,30,40,50);

abc(a);

これで関数側ではx[0]～x[4]で呼び出し側の配列a[0]～a[4]が参照できます。関数側で受け取った配列のサイズを取得するにはx.lengthとします。この値はこの例では5となります。以下のように呼び出したときのx.lengthは3です。

var b=new Array(10,20,30);

abc(b);

2次元配列の場合はx.lengthで縦のサイズ、x[0].lengthで横のサイズが得られます。

「例題1-9」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の引数は描画ベース位置のx,y座標とします。

<!DOCTYPE html>

<html>

<head>

<script type="text/javascript">

var context;

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(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(10,10);

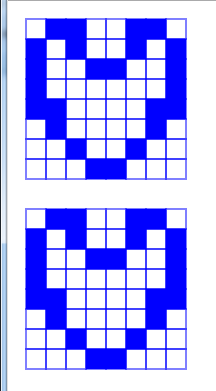
disp(10,200);

}

</script>

</body>

</html>



「練習問題1-9」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 context;

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(m,x,y)

{

for (var i=0;i< ① ;i++){

for (var j=0;j< ② ;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(m1,10,10);

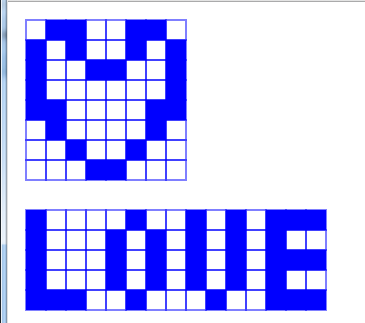
disp(m2,10,200);

}

</script>

</body>

</html>



3.　変数のスコープ

　変数の使用できる範囲をスコープと呼びます。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呼び、ローカルスコープを持つ変数をローカル変数と呼びます。

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呼び、グローバルスコープを持つ変数をグローバル変数と呼びます。

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

}

例題1-9におけるの変数contextや配列m[ ][ ]はグローバル変数です。変数i,jはローカル変数です。

1-10　その他の制御文

　既にfor文、if else文について説明しましたが、これら以外の制御文を説明します。

1.　while文

for文は繰り返す回数が予め決まっていましたが、while文は繰り返す回数が決まっていない不定回反復です。（）内の条件式が真の間｛ ｝内を繰り返します。

while (条件式) {

文

}

「例題1-10-1」1日目に1円預金する、2日目は倍の2円、3日目はさらに倍の4円、4日目はさらに倍の8円と預金していくとき、何日目で預金総額は100万円を越えるか調べるプログラムを作ります。日をday、その日の預金額をmoney、預金総額をsumで管理します。プログラムの①～⑤の意味は以下のとおりです。

①預金合計が1000000以下の間繰り返す。

②預金合計に今日の預金額を加える。

③日と預金合計を表示。

④預金額を2倍する。

⑤日を1日進める。

<!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.fillStyle = "blue";

context.font = "16px 'ＭＳ Ｐゴシック'";

var day=1,sum=0,money=1,msg="";

while (sum<=1000000) { // ①

sum+=money; // ②

msg=day+"日："+sum+"円";

context.fillText(msg, 10, 16\*day); // ③

money\*=2; // ④

day++; // ⑤

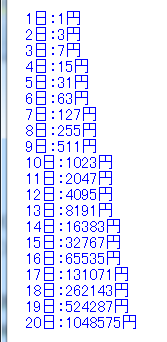
}

}

</script>

</body>

</html>



2.　else if文

if else文は2方向分岐でしたが、else節にさらにifが入ったelse if文は複数方向選択を行います。たとえば変数aに点数が入っていたときその点数が80以上、80未満70以上、70未満60以上、60未満の４種類に判定するには次のようになります。a>=70の判定に入るときは、aが80未満であることは明白なので、else if(80>a && a>=70)という条件式にしなくてよいことに注意してください。

if (a>=80){

80以上のときの処理

}

else if (p>=70){　←(80>a && a>=70)という条件式にしなくてよい

80未満70以上のときの処理

}

else if(p>=60){

70未満60以上のときの処理

}

else {

60未満のときの処理

}

「例題1-10-2」配列girl[ ]に名前、point[ ]に得点が格納されているとき、「優」、「良」、「可」、「不可」を判定して表示します。

<!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");

context.fillStyle = "blue";

context.font = "16px 'ＭＳ Ｐゴシック'";

var girl=new Array("金沢沙希","北川弥生","佐藤結衣","山口景子");

var point=new Array(78,62,59,90);

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

var msg=girl[i]+":"+point[i]+":";

if (point[i]>=80)

msg+="優";

else if (point[i]>=70)

msg+="良";

else if (point[i]>=60)

msg+="可";

else

msg+="不可";

context.fillText(msg, 10, 20\*(i+1));

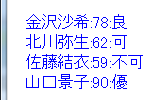
}

}

</script>

</body>

</html>



3.　switch case文

else if文の他にswitch case文でも複数方向選択を行えます。switch case文は構造がスマートですが大小判断はできません。たとえばnが1,2,3の場合で処理を分けるには次のようにします。breakでswitch case文を抜けるので、breakがないと下のcase部に進んでしまいます。default部は書く内容がなければ省略できます。

switch (n) {

case 1:1のときの処理

break;

case 2:2のときの処理

break;

case 3:3のときの処理

break;

default:それ以外のときの処理

break;

}

「例題1-10-3」変数nの値が0なら「凶」、1なら「小吉」、2なら「大吉」と表示します。nの初期値を変えて実行し直してみてください。

<!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";

context.font = "32px 'ＭＳ Ｐゴシック'";

var n=1,msg="";

switch (n){

case 0:msg="凶";break;

case 1:msg="小吉";break;

case 2:msg="大吉";break;

}

context.strokeText("あなたの運勢は"+msg, 10, 40);

}

</script>

</body>

</html>



4.　連想配列とfor in文

連想配列とは、キー（プロパティ）を指定して値をセットできる配列です。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。{と}で囲んだデータをオブジェクトリテラルと呼びます。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,・・・}

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1-10-4」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>

<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";

context.font = "36px 'ＭＳ Ｐゴシック'";

var result=document.getElementById("result");

var girls = {

name:"沙季", // 名前

age:17, // 年齢

blood:"AB" // 血液型

};

var n=1;

for (var girl in girls){

context.strokeText(girl+":"+girls[girl], 10, n\*40);

n++;

}

}

</script>

</body>

</html>

