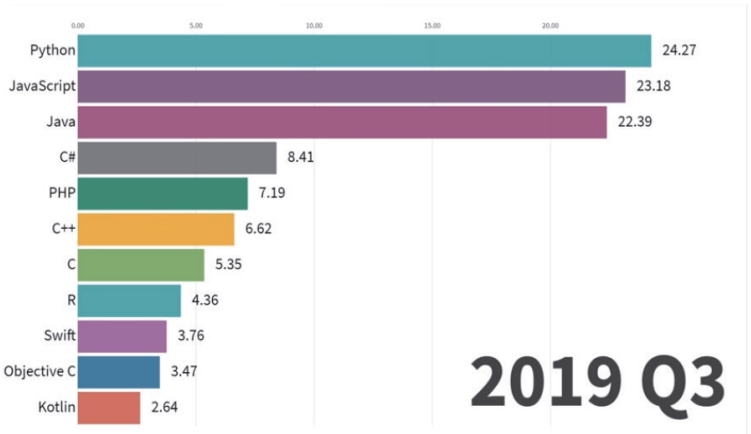
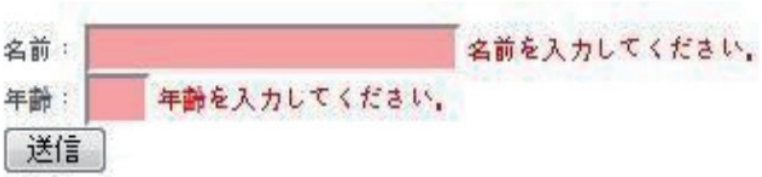
箕輪進修高等学校

クリエイト工学科 選択教科「プログラミング技術」

JavaScript とは…

JavaScript が登場したのは 1995 年。当初は「Web ページに動きを付けるもの」という程度にしか見られていませんでした。むしろ JavaScript を使った Web ページは、当時の非力なパソコンでは処理の負荷が高く、忌み嫌われることすらありました。　そんな JavaScript が脚光を浴びるようになったのは、JavaScript で高機能な Web ページを実現する「Ajax」（Asynchronous JavaScript+XML）という技術が登場したのがきっかけです。米 Google が Ajax を使って実現した「Gmail」や「Google マップ」といったサービスが、JavaScript の実力を世に知らしめました。JavaScript 自体に大きな変化があったわけではなく、潜在的に持っていた本来の能力が認められるようになったのです。　そして 2014 年、JavaScript の力を 100%発揮した Web アプリを作れるようになる HTML5 がついに登場します。HTML5 には、位置情報を利用するための「Geolocation API」、Web ページに図形を描画するための「Canvas 2D Context」、サーバーとのリアルタイム通信を実現する「WebSocket API」といった技術が含まれます。これらはいずれも、JavaScript で利用することを前提とした API です。JavaScript はもはや、HTML5 で Web アプリを開発する人にとって「知っていて当然の言語」であり、まさに「現代プログラマの必須知識」なのです。またパソコン、タブレット、スマートホンなどプラットフォームは違っても共通のブラウザさえあれば、同じアプリケーションを動作させることができ、開発の利便性からも注目されています。

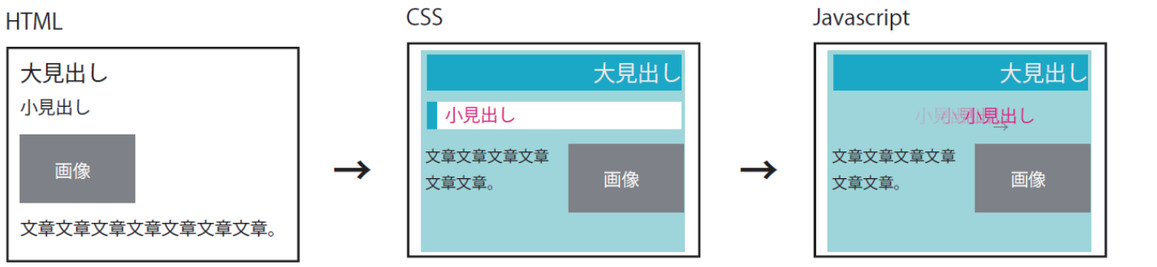




↑入力値チェックなどにも利用

ＨＴＭＬ

一般的にインターネット網を表すWWWはWorld Wide Web（ワールドワイドウェブ）の略称であり、Ｗｅｂページは HTML (Hyper Text Markup Language) で記述されている。またＷｅｂとは直訳すれば「蜘蛛の巣」のことで、世界中のネットワークが繋がる様子からこう呼ばれている。このＷｅｂページを閲覧するためのアプリケーションをブラウザと呼びクロームやインターネットエクスプローラなどが有名である。最近ではＨＴＭＬから見栄え（装飾）の部分を独立させ記述する　ＣＳＳ（カスケードスタイルシート）が主流となってきている。

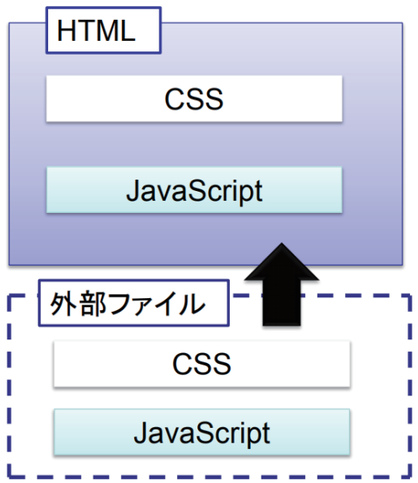


・見出しがある。
・画像がある。
・文章がある。
など、文章の構造を示す。

・背景色、文字色、線をつける。
・文字や余白の大きさを変える。
・文章や画像の位置を変える。
など、見た目を制御する。

・文字が流れる。
・画像をクリックして拡大表示。
・入力フォームからメールを送る。
など、動きを制御する。

また、ＣＳＳと JavaScript ファイルは簡単なものはＨＴＭＬ内に記述することもできるが、大規模なシステムの場合には外部ファイルとして読み込むことで汎用性が高まる。



拡張子

ファイルの種類を表すもので、jpg、.gif、.pdf など様々な種類がある。HTML ファイルの場合「.html」、もしくは「.htm」です。HTML の作成はメモ帳などのテキストエディタで行えますが、「.txt」などのテキストファイルで保存しても、コンピュータは、HTML として扱ってくれません。HTML ファイルで保存すれば、ブラウザで開いた時に、文章構造を解釈してくれます。

タグ

HTML は、タグというものを使うことによって、文章に意味を与えています。
ほとんどのタグは、開始タグと、終了タグがセットであります。
タグの書かれている部分全てをソースと呼びます。このように、タグは何種類もありますが、1 つ1 つに意味があり、正しい意味のタグを使う必要があります。

必須のタグ　<html><head><title><body>

HTML のタグはブラウザには表示されません。<h1> サンプル </h1> と書いた場合、「サンプル」のみ表示されます。また、HTML で記す情報は、Web ページに表示されるものだけではありません。ページ上表示されないけれど、サイトの情報として必要なタグもあります。

書き方例：

```
<html>                                  HTML を書き始めます
<head>                                  ヘッダ情報を書き始めます
    <title> サンプルサイト </title>      サイトタイトルを示す
</head>                                 ヘッダ情報を書き終えます
<body>                                  文章の本体（表示される部分）を書き始めます
</body>                                 文章の本体（表示される部分）を書き終えます
</html>                                 HTML を書き終えます
```

整形とコメントアウト

HTML ファイル内では、いくら改行や、（タグの外での）スペースを入れても、ブラウザの表示に変化はありません。ですので、適度に見やすいように改行をしたり、Tab（字下げ）を入れて、整形を行えます。
また、タグで囲まれた部分以外には、テキストを書いてはいけませんが、コメントアウト［<!-- -->］を使うことで、制作上のメモなどを入れられます。

Web ページを作る方法

Web ページを作る方法は大きく分けて３種類あります。
①テキストエデイタを用いて直接 HTML ＋ CSS を打つ
メモ帳や TeraPad などのテキストエディタを用いて直接 HTML ＋ CSS でプログラムファイルを作成します。
② Web オーサリングツール（Web デザインソフト）
視覚的な操作でテキストやイメージを配置しながら Web ページをデザインしていき、結果として HTML+CSS コードが自動生成されます。HTML や CSS の知識がなくても Web ページが作成できます。Web オーサリングツールとして IBM のホームページ・ビルダーやアドビシステムズの Adobe Dreamweaver、マイクロソフトの Microsoft Expression Web などがあります。
③ CMS（Contents Management System）
CMS は Web ページの作成から Web サイトの構築まで総合的に管理する Web ページ管理システムです。プロバイダの中には CMS を活用した「簡単ホームページ作成」といったサービスが提供されています。Web ページのデザインをテンプレートとして予め用意しておき、ユーザはイメージやテキストをそのテンプレートに配置するだけで Web ページの作成と登録が行えます。ブログなども CMS を利用して Web ページの作成、登録をしている例です。最近ではＷｏｒｄＰｒｅｓｓなども有名です。

ＨＴＭＬ入門

準備

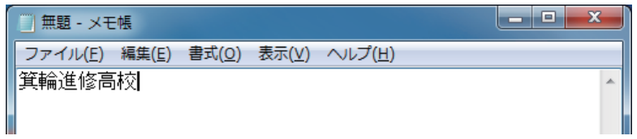
○自身のネットワーク上に「00 プログラミング技術」フォルダを作成。

課題１　次からの指示に従ってＨＴＭＬファイルを作成せよ。

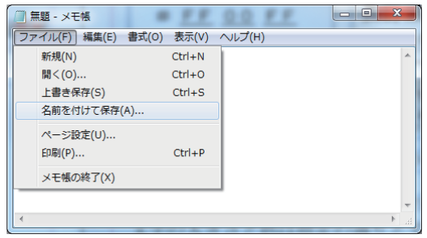
①自身のネットワーク上の「プログラミング技術」内に 「ＨＴＭＬ」フォルダを作成する。

②「アクセサリ」⇒「メモ帳」を起動。

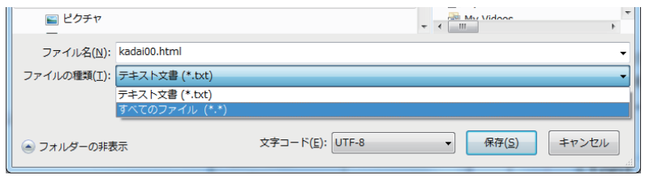
③メモ帳に文字「箕輪進修高校」と入力



④「ファイル」メニューから「名前を付けて保存」を選択

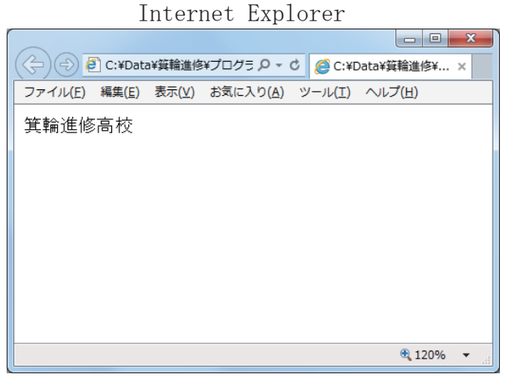


⑤ファイルの種類を「全てのファイル」
　文字コードを「ＵＴＦ－８」
　ファイル名を「kadai00.html」として
　作成した「HTML」フォルダへ保存する。

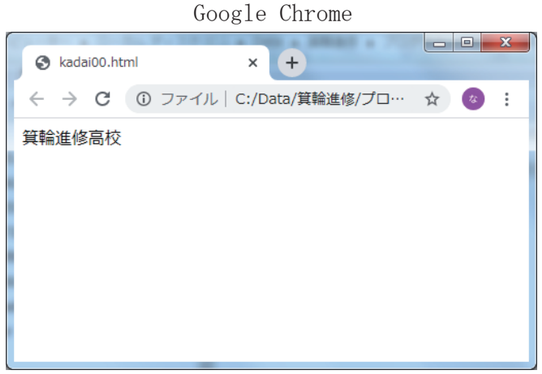


⑥ＰＣかエクスプローラで保存した「kadai00.html」をダブルクリックしてブラウザで開く。

Internet Explorer

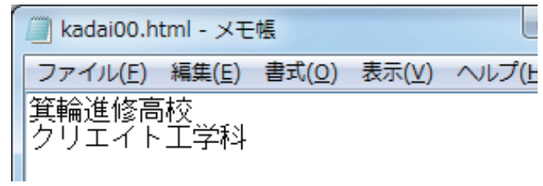


Google Chrome

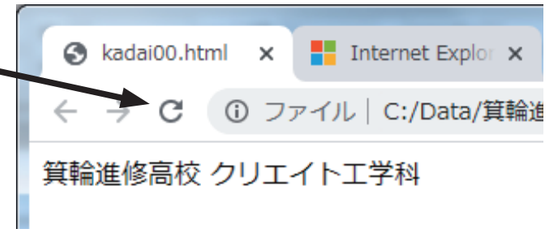


⑦メモ帳で例のように、次行に文字を追加する。

⑧メモ帳で上書き保存

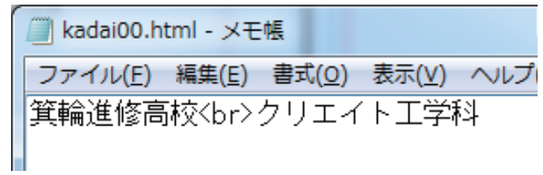


⑨ブラウザで再読み込み（リロード）

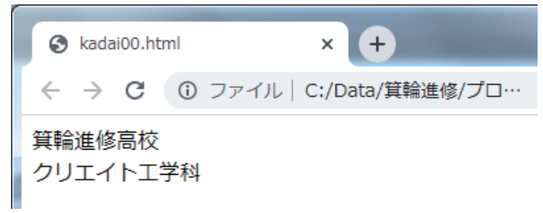


⑩改行タグ〈br〉使用し、例のように修正する。
　タグ〈br〉は「半角英数記号」で入力する！

　上書き保存　⇒　ブラウザでリロード

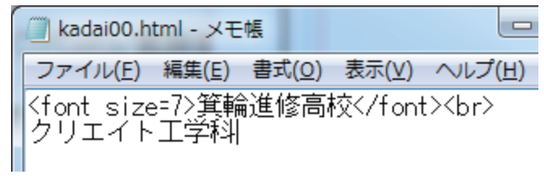


　整形タグ〈p〉に置き換えて試してみる。



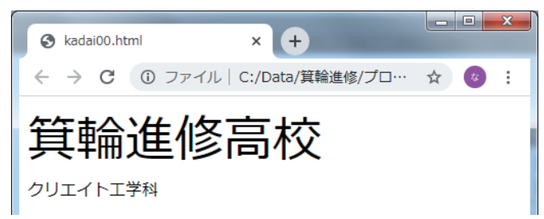
⑪文字指定タグを使用して例のように修正する。

 ○○

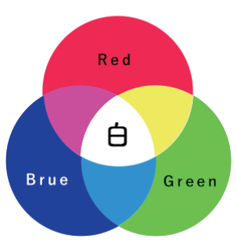


文字サイズ　１～７			
色の指定　＃	ＦＦ	００	ＦＦ
	R	G	B

①三原色を１６進数２ケタで指定
②BLUE, RED など色名で指定



光の三原色



代表的な色名とカラーコード

black	#	white	#
red	#	green	#
blue	#	yellow	#
fuchsia	#	aqua	#

⑫クラス共通フォルダから「kadai01.html」を自分のフォルダへコピーする。

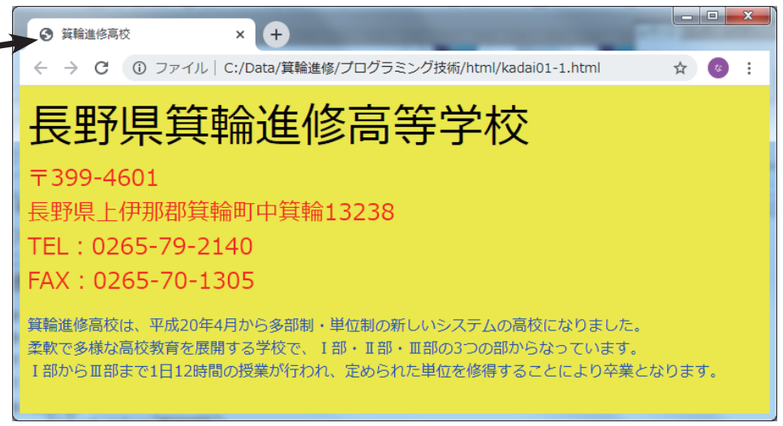
⑬メモ帳で開いて、右のように表示するようにタグを追加せよ。

⑭省略していたタグを追加し、見やすいようにインデントも行う。また、タイトルを「箕輪進修高校」と設定し、body タグで背景色を黄色に設定する。

kadai01.html

```
<html>
    <head>
        <title> 箕輪進修高校 </title>
    </head>
    <body bgcolor="#_____">

        ・・・本文・・・

    </body>
</html>
```

タイトル →



タグの基本

<タグ名> タグの中身 </タグ名>

↓

はさまれるタグの種類によって役割が変わる

機能	タグ
タイトル	<title> ～ </title>
改行（小）	
段　落	<p>～<p>
見出し文字	～ 　1が大、6が小。自動的に改行
文字サイズ	～（1が小、7が大）
文字色	～
太字	～
斜体	～

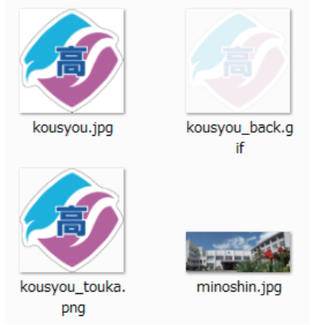
課題２

①クラス共通フォルダ内の「HTML」フォルダにある「image」というフォルダごと自身のＨＴＭＬフォルダへコピーする。（決して移動しない！）

②「kadai02.html」を自身のフォルダへコピーする。

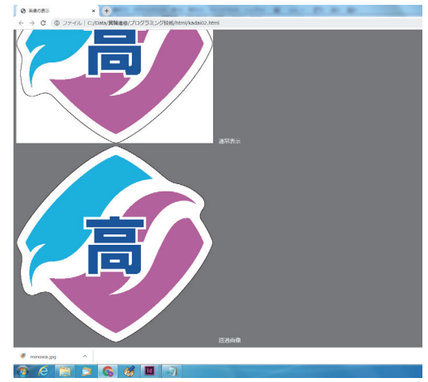
③「kadai02.html」を開いて次のようにに追加修正を行う。

ブラウザで読み込む。



kadai02.html

```
<html>
<head>
    <title>画像の表示</title>
</head>
<body bgcolor="#777777" text="white">

    <img src="image/kousyou.jpg">通常表示<br>
    <img src="image/kousyou_touka.png">透過画像<br>

</body>
```



画像タグ <img src="　　">

画像を使用する時のタグです。必須の属性がいくつかあります。終了タグを持ちません。

画像のある場所を URL で指定します。絶対パスか、相対パスで指定できます。

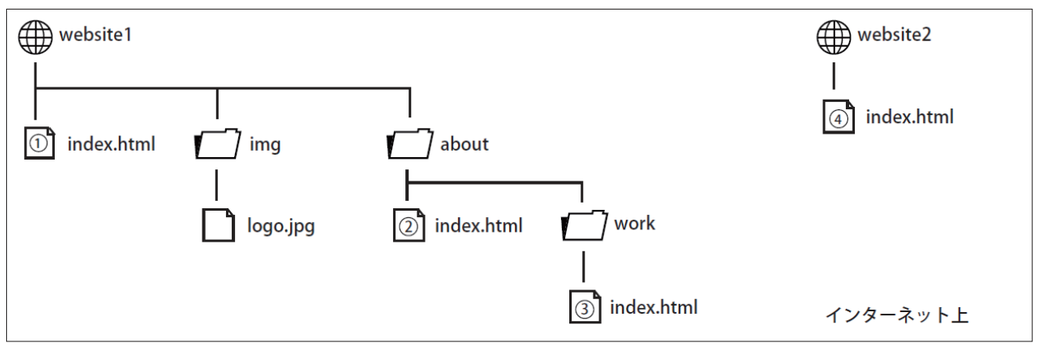
絶対パス、相対パス［http:、../］

書き方例：<img src=" URL" width="100" height="200">

画像の横幅（width）、縦幅（height）を指定します。画像サイズの単位は px ですが、px を記述する必要はありません。

○ 絶対パス、相対パス [http:、../]

パスの書き方は、絶対パスと相対パスの 2 種類あります。絶対パスは「http:」から始まるインターネット上の URL、相対パスは現在のファイルから相対的に見たファイルの位置を表すもので「../（一階層出る）」「about/index.html（about フォルダの中の index.html）」といった書き方を使います。



相対パス（同じサイト内でよく使う）

① から見た ③ → about に入って、work に入ったところの index.html → about/work/index.html

③ から見た logo.jpg → 一階層出て、一階層出て、img に入ったところの logo.jpg → ../../img/index.html

絶対パス（他のサイトを指定する場合よく使う）

④（どのファイルからでも） → インターネット上の website2 というサイトの index.html → http://www.website2/index.html

④画像「kousyou. jpg」を横 100 ピクセルの大きさに設定する。
ブラウザで再読み込みする。

⑤画像「kousyou_touka. png」を横 200 ピクセル、縦 100 ピクセルの大きさに設定し、ブラウザで再読み込みする。

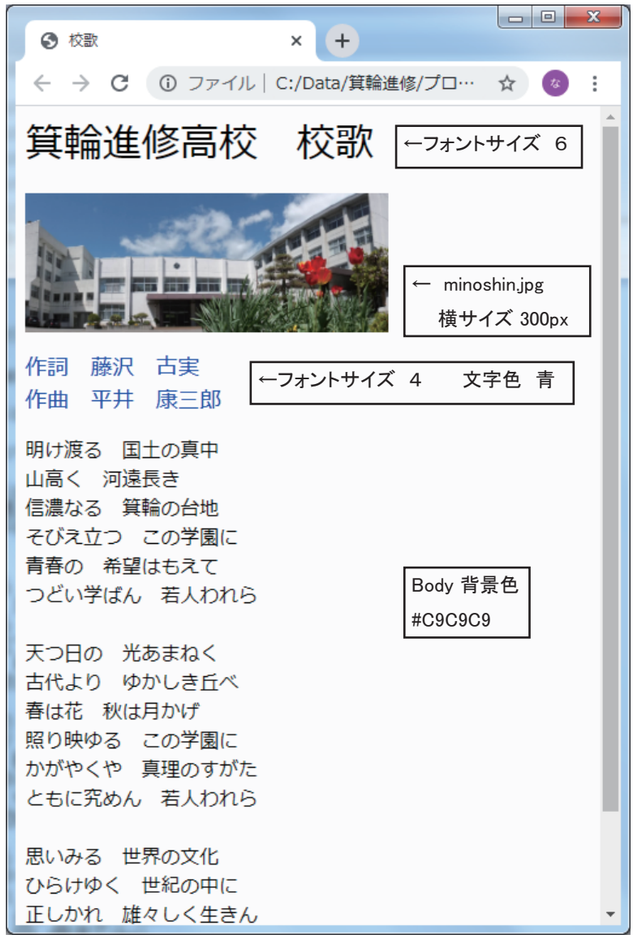
⑥背景に画像「kousyou_bak. gif」を設定し、ブラウザで再読み込みする。

jpg	デジタルカメラで撮影したデータなどに使われており、画像規格の中でも、もっともポピュラーな規格です。記録するときに圧縮して保存しているため、圧縮率を高くしてしまうと画質が劣化してしまいます。
png	インターネットのWebページの画像としてよく使われています。透明色の利用もでき、GIFよりも扱える色数が多いですが、品質劣化のないデータ圧縮が可能なのが特長です
gif	インターネットのWebページの画像としてよく使われている規格です。扱える色数が256色のため、色数が少ないイラスト等に適しています。また、透明色を設定できるため背景の色を反映することができます。年賀状などの切り抜き写真に便利です。

課題３

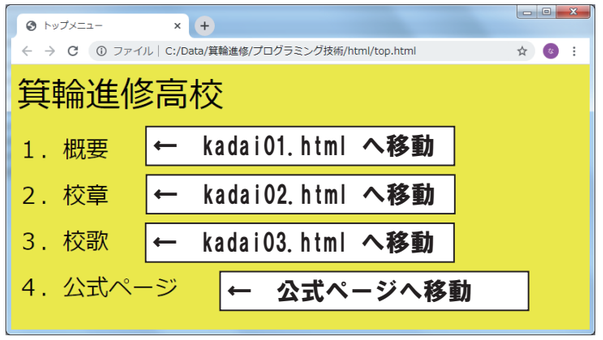
①クラス共通フォルダ内の「HTML」フォルダにある「kadai03. html」ファイルをへ自身のフォルダへコピーする。（決して移動しない！）

②「kadai03. html」を開いて、右の表示例になるように追加修正する。その際、少しづつ追加、修正を行い、そのつどブラウザで読み込み確認しながら進めること。



課題４

①メモ帳で右のファイルを新規作成し、「top. html」と名前を付けて保存、ブラウザで読み込んむ。



top.html

```
<html>
<head>
    <title>トップメニュー</title>
</head>
<body bgcolor="#FFFF00">
<font size=7>箕輪進修高校</font><p>
<font size=6>
1．概要<p>
2．校章<p>
3．校歌<p>
4．公式ページ<p>
</font>
</body>
</html
```

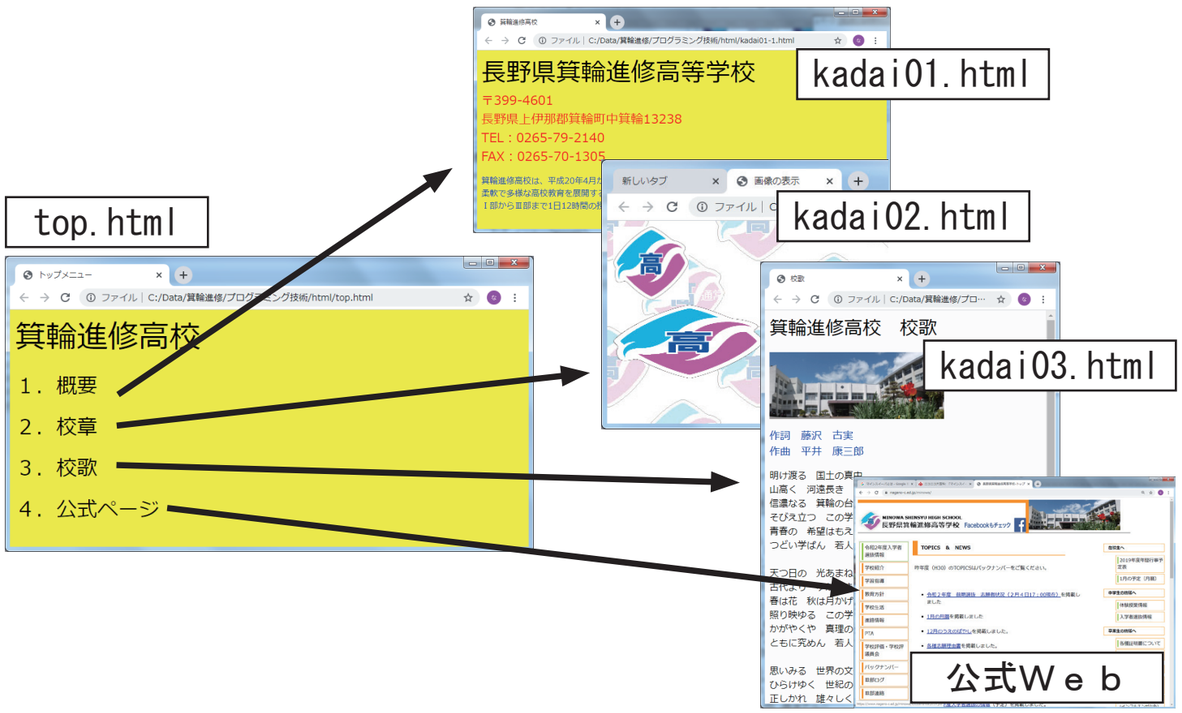
②「top. html」にそれぞれのページへジャンプするタグを付け加え、ブラウザで動作を確認する。各ページからトップへは「戻る」ボタンで戻る。

リンク例（アンカータグ）

```
<a href="kadai01.html"> 概要 </a> <br>
```

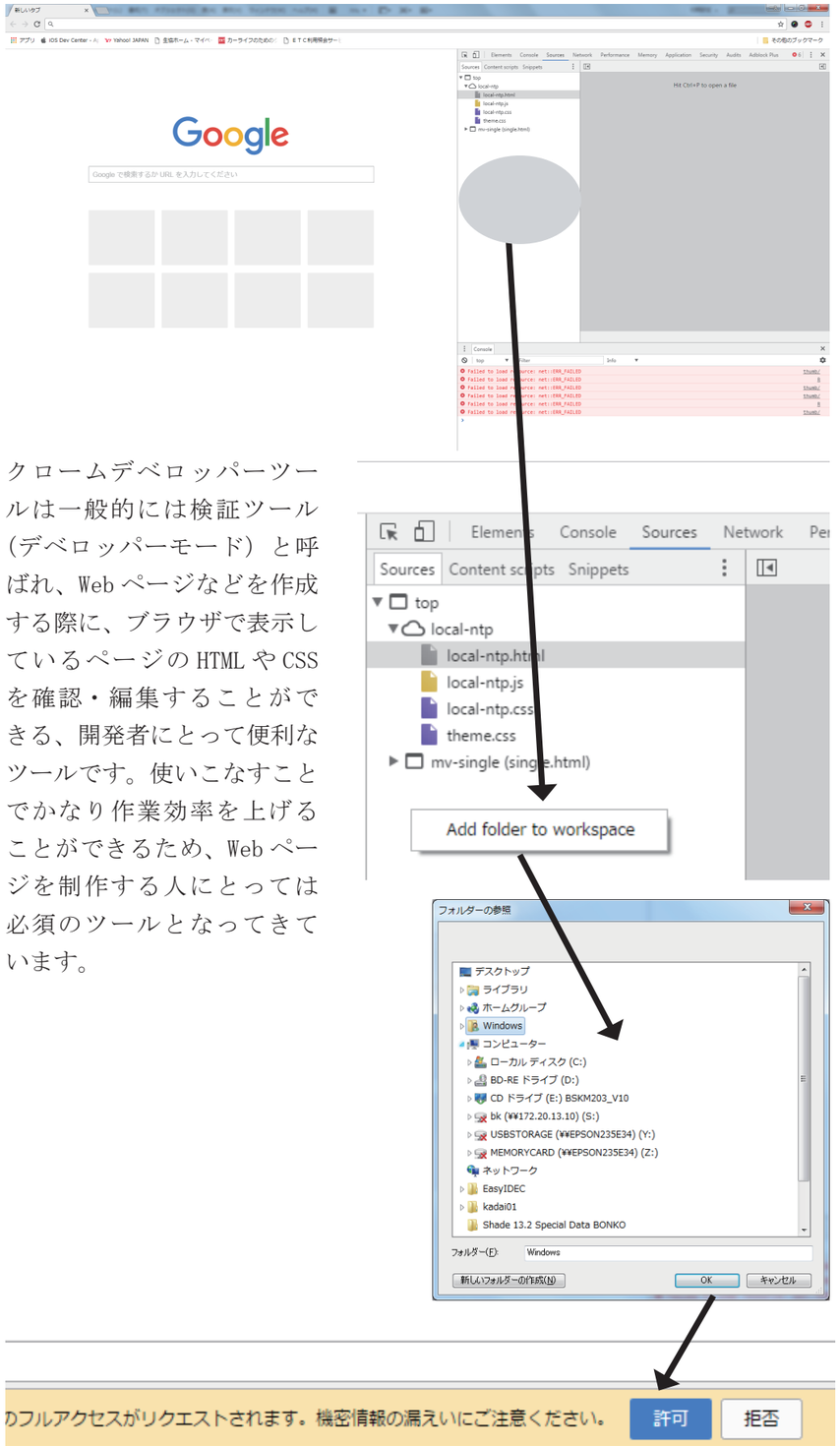
別タブで開く

```
<a href="http://www.nagano-c.ed.jp/minowa/" target="_blank">公式ページ</a>
```



Chrome Developer Tools

クロームデベロッパーツールは一般的には検証ツール（デベロッパーモード）と呼ばれ、Web ページなどを作成する際に、ブラウザで表示しているページの HTML や CSS を確認・編集することができる、開発者にとって便利なツールです。使いこなすことでかなり作業効率を上げることができるため、Web ページを制作する人にとっては必須のツールとなってきています。



① Chrome を起動し
　　Ｆ１２　を押します。

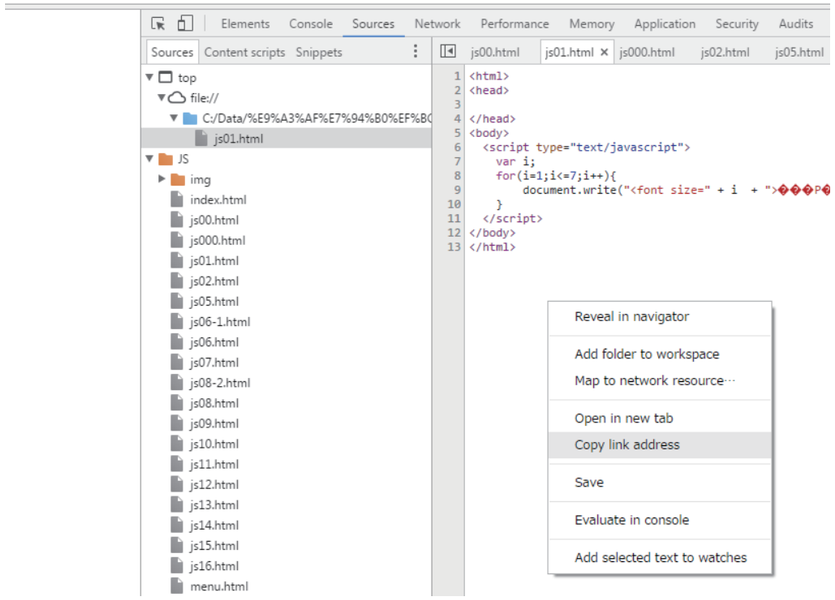
②図のあたりで右クリック

「Add folder to workspace」

を左クリック

③ワークスペース（修正可能）に追加する作業用のフォルダを指定します。

④ 「許可」をクリック

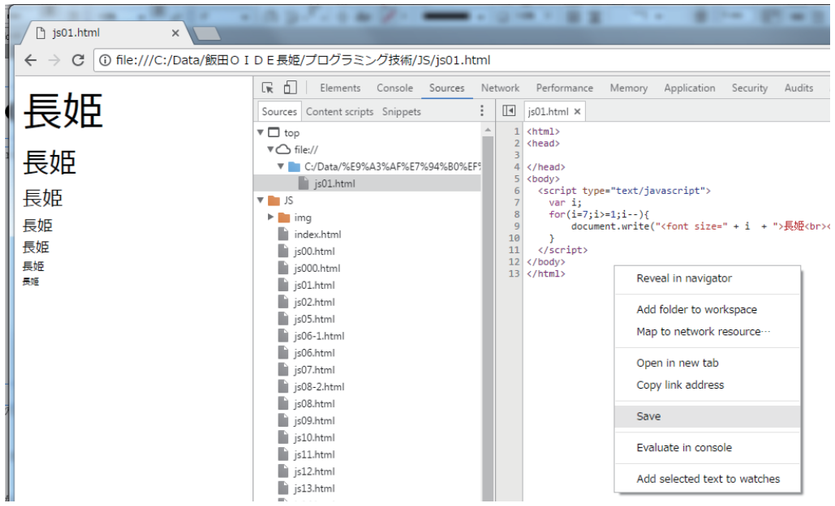


⑤選択したファイルのソースが表示され、編集が可能になります。

文字サイズの変更は
ＣＴＲＬキーと「＋」「－」

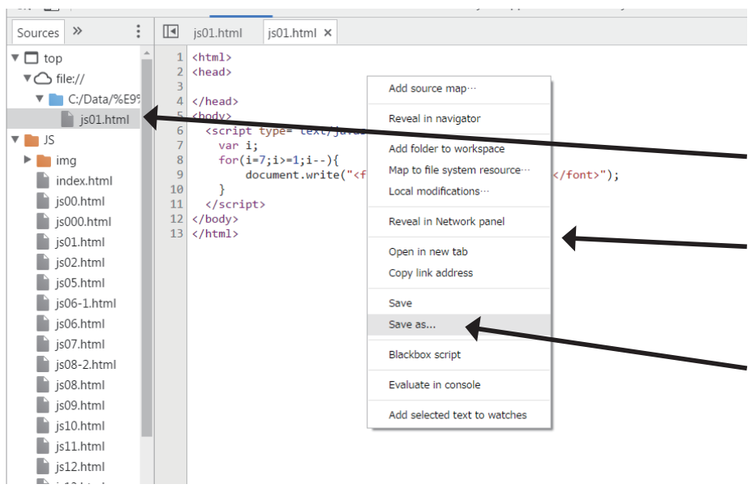
⑥ソースで右クリックし
「Copy to address」
を選択

⑦左上のブラウザのアドレスバーにペーストします。



日本語が文字化けしている場合は、デベロッパーツール上で修正します。

⑧以降、追加修正
「Save」**CTRL+S** して
ブラウザのリロードボタンで再読み込みします。
または **CTRL+R**



現在編集中のファイルを別の名前で保存するには

1）ここのファイル名をクリック

2）開いたソースタブで右クリックしてメニューを表示

3）「Save as...」を選択します

JavaScript 課題

準備

①自身のネットワーク上の「プログラミング技術」内に「JavaScript」フォルダを作成する。

②クラス共通フォルダから「js00.html」を自身のフォルダへコピーする。

③ブラウザ（Chrome）の表示を確認する。

④ブラウザ（Chrome）のデベロッパーツールを起動し、表示部分を変更し、上書き保存する。

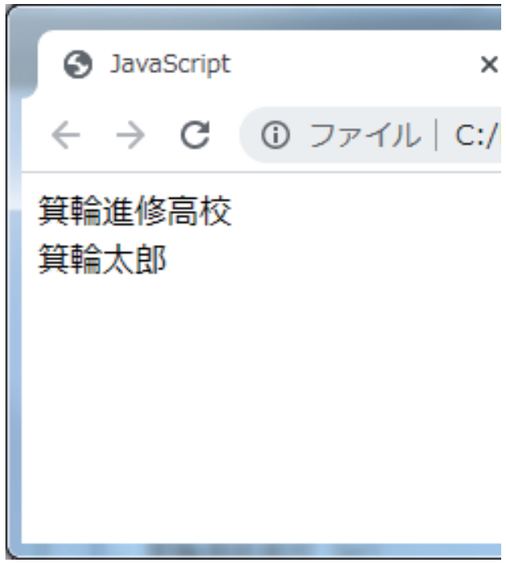
⑤リロードして表示を確認する。

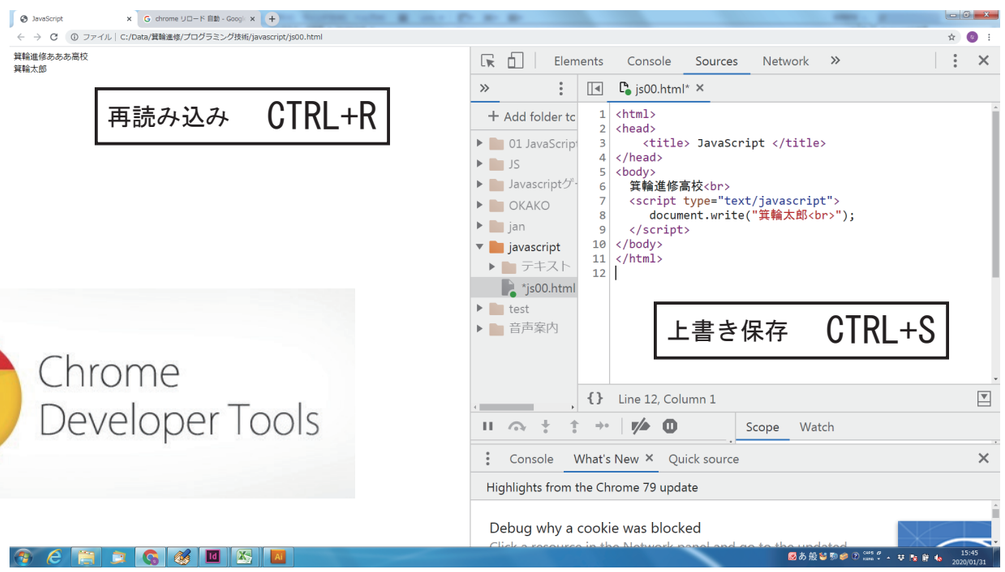
⑥何度か修正　→　リロードの手順を確認し、デベロッパーツールの使い方に慣れる。

js00.html

```
<html>
<head>
    <title> JavaScript </title>
</head>
<body>
  箕輪進修高校 <br>
  <script type="text/javascript">
     document.write(" 箕輪太郎 <br>");
  </script>
</body>
</html>
```

自分の名前に変更する





Chrome Developer Tools

JavaScript の基礎

◆ script タグ

<script></script> タグの中にスクリプトを記述することで、Web ブラウザに JavaScript のコードであると判断させ、解釈実行させる。<script> タグの type アトリビュートでスクリプト言語の種類を指定する。javascript の場合「text/javascript」を指定。

```
<script language="JavaScript " type = "text/javascript">
    JavaScript のコードを記述
    ・・・・
</script>
```

◆ document.write

渡された文字列を現在の HTML ドキュメントに出力する。

```
document.write("<hr>")
document.write("<h1>javascript</h1>")
document.write("</hr>")
```

＝＝

```
<hr>
<h1>javascript</h1>
</hr>
```

とする入力することで HTML 内に、直接記述したのと同じ動作をする。

・「document.write」で＜ｉｍｇ＞タグなどでイメージを表示させたい場合
<img src="new.gif">　のようなタグを　document.write("<img src="new.gif">")
のようには記述できない。
この場合次のように内部の「"」ダブルクォーテーションを「'」シングルクォーテーションに変更する

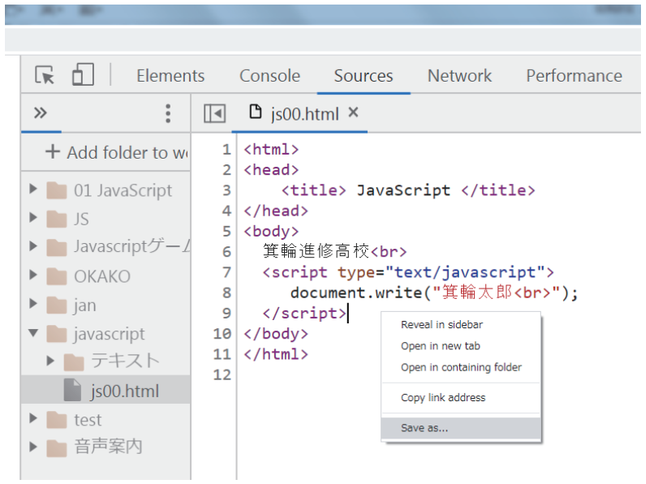
```
document.write("<img src='new.gif'>")
```

◆スクリプト記述方法の基本

- 大文字と小文字は区別される
- ステートメントの終わりはセミコロン。
- 「//」または「/*」「*/」でコメントを記述
- head エレメントや body エレメントなど、任意の場所に記述できる。
- Web ブラウザの設定によっては、エラーがあっても何も表示されない。
- 演算子など基本的な文法はＣ言語と同じ。

準備②

現在編集中のファイルを別名で保存します。



①編集領域で右クリック　→　「Save as…」

②ファイル名を「js01.html」として保存

　ファイルの種類、保存場所を確認すること。

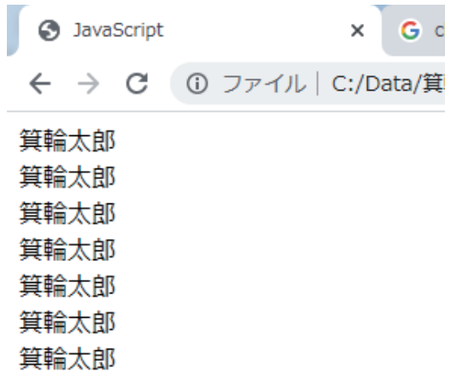
③上部のファイル名のタブに注意する。元の

　ファイルは閉じた方が良い。**自動的に保存したファイルには切り替わらないので注意！**

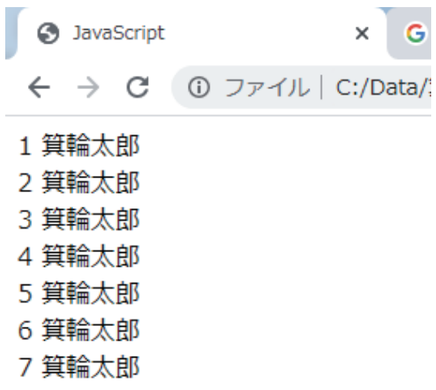
JavaScript 01-01　次のスクリプトを自分の名前を表示するように変更して実行する。

js01.html

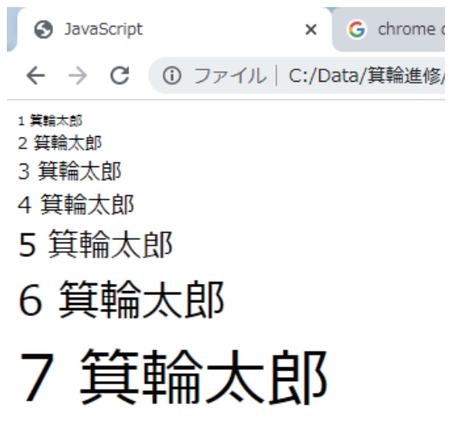
```
<html>
<head>

</head>
<body>
  <script type="text/javascript">
    var i;
      for(i=1;i<=7;i++){
            document.write(" 箕輪太郎 <br>");
      }
  </script>
</body>
</html>
```



JavaScript 01-02　①のスクリプトを番号をつけて表示するように変更する。



JavaScript 01-03　②のスクリプトをフォントサイズを１～７まで変化させて表示するように変更する。



◆オブジェクトとメソッド

　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。前の例では document というオブジェクトに対し write 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。

◆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;     var i,j;
```

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。

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。

〇変数名の付け方

変数に付ける名前を変数名と呼びます。変数名には使用可能な文字など規則がががあります。

・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。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。

〇変数のスコープ

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。

JavaScript 準備

クラス共通フォルダ内にある「img」フォルダごと自身のドライブの「JavaScript」フォルダ内へコピーする。

JavaScript 02-01

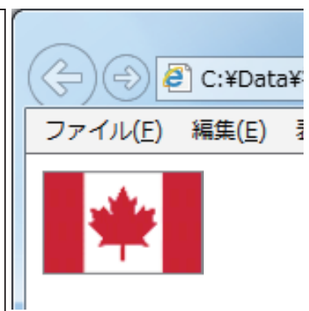
img フォルダ内にある国旗の画像を表示する。国旗はほかのイメージに変更してもよい。

※イメージファイルは「img」フォルダ内にあることを注意せよ。

※　”　” と　’　’ の使い分けに注意すること。

js02.html

```
<html>
<head>

</head>
<body>
  <script type="text/javascript">
      document.write("<img src='img/canada.gif'");
  </script>
</body>
</html>
```



JavaScript 02-02

img フォルダの中にある画像ファイルを横１０個分表示させるスクリプトを作れ。
画像はどれを使用してもよいが実行結果が収まるように大きさを調整すること。。

js02.html

JavaScript 02-03

実行例のように拡大しながら４つ表示するようにせよ。
イメージの横サイズを 50px, 100px, 150px, 200px と変化させながら表示する。

js02.html

◆繰返し（for）

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

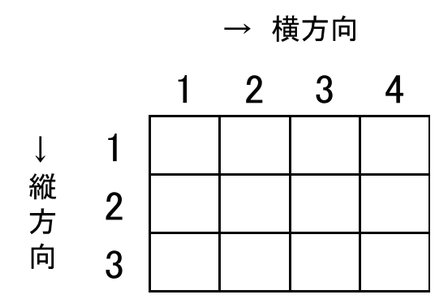
for 文は決められた回数の繰り返しに使います。たとえば、右の例は、変数 i を 1 から始め、10 以下の間、i を +1 しながら、{ と } で囲まれた範囲（ブロック）を繰り返します。{ } 内に書く文が 1 つのときは { } を省略できます。繰り返しのことをループと呼びます。for 文の繰り返しを制御している変数のことをループ変数と呼びます。

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

◆二重ループ（多重ループ）

ループの中にループが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<=4;i++){
      for (j=1;j<=3;j++){
            ○○○
      }
}
```

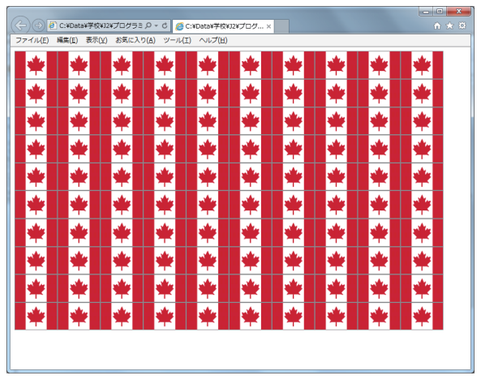


表の様なデータを扱う時、一般的には内側ループで横方向の繰り返し、外側ループで縦方向の繰り返しが行われます。理論的にが何重ループでも可能ですが、実際のプログラミングにおいては三重ループが理解できれば十分です。

JavaScript 02-04

img フォルダの中にある画像ファイルを横１０×縦１０個分表示させるスクリプトを作れ。画像はどれを使用してもよいが実行結果が収まるように大きさを調整すること。

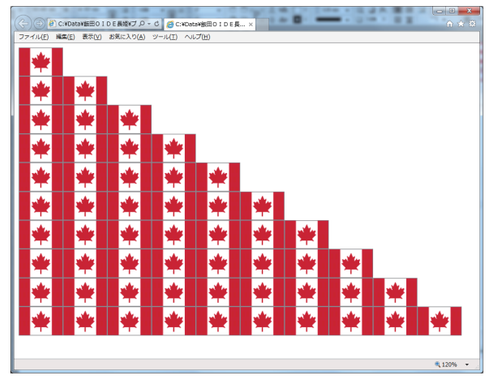
js02-04.html



JavaScript 02-05

img フォルダの中にある画像ファイルを横１０×縦１０個分の**三角形**に表示させるスクリプトを作れ。画像はどれを使用してもよいが実行結果が収まるように大きさを調整すること。

js02-05.html



◆条件判断（if ～ else ～）

```
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
}
```

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。
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。

○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

○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

◆ Date 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。
Date オブジェクトは、日付や時間などを扱うためのメソッドが定義されており、これらを使用することで、時間の計算や日付の換算等を簡単に行うことができます。

```
var today=new Date();     現在時刻の取得
var h=today.getHours();   時間
var m=today.getMinutes(); 分
var s=today.getSeconds(); 秒
```

関数	説明
getDate()	日を取得する
getDay()	曜日を表す番号を取得（0～6の7つ、0が日曜日）
getFullYear()	西暦を取得する
getHours()	何時かを取得する
getMilliseconds()	ミリ秒を取得する
getMinutes()	分を取得する
getMonth()	月を取得する
getSeconds()	秒を取得する
getTime()	1970年1月1日午前0時からの秒数を取得する
getYear()	年を取得する
setFullYear()	西暦年を設定する
setHours()	何時かを設定する
setMilliseconds()	ミリ秒設定する
setMinutes()	分を設定する
setMonth()	月を設定する
setSeconds()	秒を設定する
setTime()	1970年1月1日午前0時からの経過秒数を設定する
setYear()	年を設定する
setDate()	日を設定する

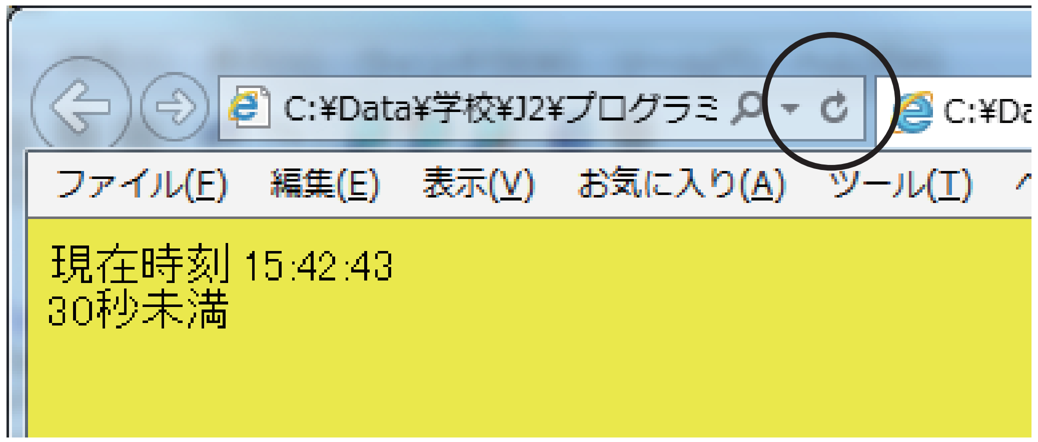
JavaScript 03 - 01 次のスクリプトを入力し実行せよ

・現在時刻（ページ読み込み時）を表示
・もし秒が３０秒未満なら背景色を水色（aqua）
　　　　３０秒以上なら　〃　　黄色（Yellow）に設定

js03.html

```
<html>
<body>
<script type="text/javascript">
    var today=new Date();            // Date オブジェクト
    var h=today.getHours();
    var m=today.getMinutes();
    var s=today.getSeconds();

    document.write(
    if (          ){

        document.write("30 秒以上 <br>");
    }
    else {

        document.write("30 秒未満 <br>");
    }
</script>
</body>
</html>
```

「リロード」ボタンで
何回か再読み込みしてみる



◆多方向分岐（switch ～ case ）

if - else の２分岐だけではなく多方向に分岐させる制御文もあります。

○ switch ～ case

まず整数式の値を評価、その値と case ラベルの定数式の値とを上から順番に比較し、一致したラベル以降の文を実行します。ブロックから抜けるためには break 文を使用します。

```
switch( 整数式 ) {
    case ○○ ： 文１ ←条件を満たしたときに実行される文
    case □□ ： 文２ ←条件を満たしたときに実行される文
    case △△ ： 文３←条件を満たしたときに実行される文
    default ： 文４ ←該当する条件がないとき実行される文
}
```

◆ 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。

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。

関数	説明
PI	円周率
abs()	絶対値
acos()	アークコサイン
asin()	アークサイン
atan()	アークタンジェント
ceil()	小数点切り上げ
cos()	コサイン
sin()	サイン
sqrt()	平方根
tan()	タンジェント
floor()	小数点以下を切り捨て
max()	2つの数値のうち大きい方を返す
min()	2つの数値のうち小さい方を返す
pow()	べき乗
random()	0から1の間でランダムな値を取得
round()	小数点以下を四捨五入

Math.sqrt(10)； ← これで、√10 の結果が得られます。

Math.random() は０～１未満（1 は入らない）までの小数による乱数を生成。

Math.floor() は小数点以下を切り捨てて整数化します。

Math.floor(Math.random()*10)； ← ０～９までの整数乱数を取得

◆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。

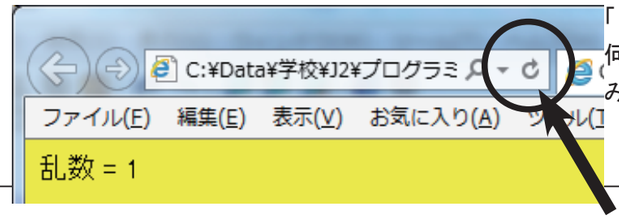
“5”+5

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。

JavaScript 04-01 「乱数」によって 背景色変えるスクリプトを作成する。
制御構造は switch - case を使用すること。

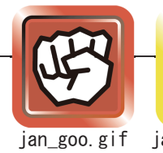
js04-01.html

```
<html>
<body>
<script type="text/javascript">
    var r = Math.floor(Math.random()[          ]);     /* 0 ～ 2 の乱数  */
    document.write(" 乱数 = " + r +"<br>");
    switch([    ]){
        case 0 : [                              ][    ] /* 水色 */
        case 1 : [                              ][    ] /* 黄色 */
        case 2 : [                              ][    ] /* 赤色 */
     }
</script>
</body>
</html>
```



「リロード」ボタンで何回か再読み込みしてみる

JavaScript 04-02 img フォルダの中に次の３つのイメージファイルがある。０～２の乱数を発生させ対応するイメージファイルを表示するスクリプトを作成せよ。制御構造は switch - case を使用すること。


jan_goo.gif 0


jan_cyoki.gif 1


jan_paa.gif 2

js04-02.html

```
<html>
<body>
<script type="text/javascript">
    var r =[                                    ]  /* 0 ～ 2 の乱数  */
    document.write(" 乱数 = " + r +"<br>");
    switch([    ]){
        case 0 : [                              ][    ] /* グー */
        case 1 : [                              ][    ] /* チョキ */
        case 2 : [                              ][    ] /* パー */
     }
</script>
</body>
</html>
```

「リロード」ボタンで何回か再読み込みしてみる

◆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]
   配列名↑  ↑添字。配列要素の番号
```

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);
配列名 ↑            ↑配列のサイズ（要素数）
```

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 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");
```

◆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
}
```

	0	1	2	3
0				
1				
2				

これで右のような 2 次元配列が宣言されました。
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;
    }
}
```

	0	1	2	3
0	0	0	0	0
1	0	0	0	0
2	0	0	0	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(1,0,1,0);
a[2] = new Array(1,2,3,4);
```

	0	1	2	3
0	0	0	0	0
1	1	0	1	0
2	1	2	3	4

JavaScript 05-01 配列に１００までの整数をいくつか設定し、そのデータと合計を求めて表示するスクリプトを作れ。

js05-01.html

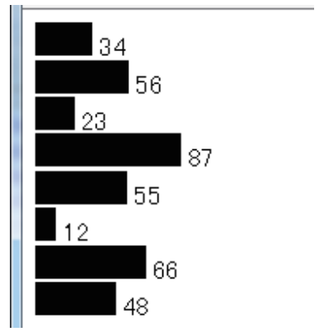
```
<html>
<body>
<script type="text/javascript">
    var a=new Array(34,56,23,87,55,12,66,48);
    var i,sum = 0;
    for ( i=0; i<a.length; i++ ){
        document.write(      + "<br>" );
        sum =
    }
    document.write("合計 = " + sum );
</script>
</body>
</html>
```

```
34
56
23
87
55
12
66
48
合計= 381
```

JavaScript 05-02 配列に１００までの整数をいくつか設定し、そのデータを横棒グラフにして表示するスクリプトを作れ。
グラフのために使用する画像は「img」フォルダ中の「black.png」または「red.png」とする。

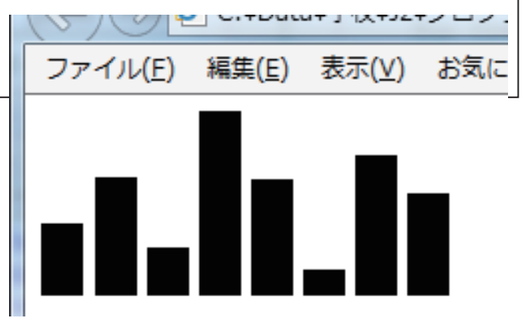
js05-02.html

```
<html>
<head>
</head>
<body>
<script type="text/javascript">
    var a=new Array(34,56,23,87,55,12,66,48);
    var i;
    for (i=0;i<a.length;i++){
        document.write(
        document.write(
    }
</script>
</body>
</html>
```



JavaScript 05-03 05-02 を変更して縦棒グラフを表示するようにせよ。

js05-03.html



JavaScript 06-01 配列に整数をいくつか設定し、そのデータと合計を求めて表示するスクリプトを、要素番号を１個ずつ取り出す繰返し「ｆｏｒ」を使って作れ。

js06-01.html

```
<html>
<body>
<script type="text/javascript">
    var a = [34,56,23,87,55,12,66,48];  // 配列初期設定の方法②
    var i,sum=0;
    for (          ){          // 配列から要素番号を１個ずつ取り出す繰返し
        document.write( a[i] + "<br>" );
        sum = sum +      
    }
    document.write(" 合計 = " + sum );
</script>
</body>
</html>
```

```
34
56
23
87
55
12
66
48
合計= 381
```

◆連想配列と for in 文

○ 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。

○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,・・・}
```

○ 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]);
}
```


JavaScript 06-02 **連想配列（添字を番号でなく名前を付けて管理）**

テストの点数を番号でなく名前で管理する。合計と最高点を表示する。

js06-02.html

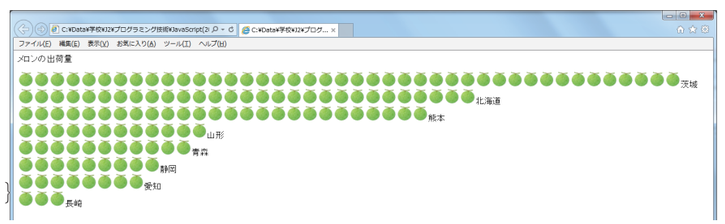
```
<html>
<body>
<script type="text/javascript">
    var a = { "Aoki":67,"Iida":56,"Ueno":87,"Eguchi":55 };  // 連想配列
    var i , sum = 0 ;            // 連想配列への代入例  a['Aoki'] = 67
    var max = "Aoki" ;
    for                                
        document.write( i + " : " + a[i] + "<br>" );
        sum = sum + a[i];
        if( a[i] > a[max]) max = i;
    }
    document.write(" 合計 = " + sum +"<br>");
    document.write(                                    );
</script>
</body>
</html>
```

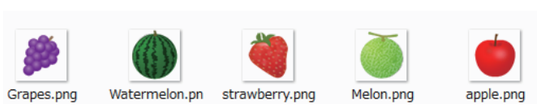
```
Aoki : 67
Iida : 56
Ueno : 87
Eguchi : 55
合計 = 265
最大 = Ueno:87
```

JavaScript 06-03 次表は都道府県別の果物の出荷量ランキングである。どれかを選択し横棒グラフにして表示するスクリプトを作れ。

	りんご		メロン		いちご		ぶどう		すいか	
順位	都道府県	出荷量(千t)	都道府県	出荷量(千t)	都道府県	出荷量(千t)	都道府県	出荷量(千t)	都道府県	出荷量(千t)
1	青森	340	茨城	42	栃木	26	山梨	31	熊本	56
2	長野	100	北海道	29	福岡	17	長野	22	千葉	46
3	山形	25	熊本	26	熊本	12	山形	12	山形	32
4	岩手	15	山形	12	長崎	11	岡山	6	鳥取	18
5	秋田	14	青森	11	静岡	10	福岡	4	長野	17
6	福島	10	静岡	9	愛知	10	北海道	4	茨城	16
7	北海道	3	愛知	8	佐賀	10	島根	2	北海道	16
8	富山	1	長崎	3	茨城	8	広島	2	新潟	15

```
<html>
<body>
<script type="text/javascript">
    var melon = {  " 茨城 "  :42,        // 連想配列 melon[' 茨城 '] = 42
                   " 北海道 ":29,
                   " 熊本 "  :26,
                   " 山形 "  :12,
                   " 青森 "  :11,
                   " 静岡 "  :9,
                   " 愛知 "  :8,
                   " 長崎 "  :3      };
    var i,j;
    document.write(" メロンの出荷量 <p>");
    for (sanchi in melon){
        for(j=0;                    j++){
            document.write("<img src='img/melon.png' width=30>");
        }
        document.write(                    );
    }
</script>
</body>
</html>
```





js06-03.html

◆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。
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。

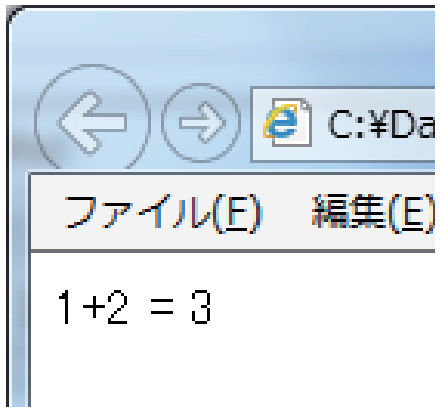
```
↓関数名
function spc(n)    ←関数の定義
{             ↑仮引数。データを受け取る変数
  ・・・
  ・・・
  return s; ←呼び出し側に変数 s の値が
               返される。これを戻り値と呼ぶ。
}
・・・
・・・
spc(i); ←関数の呼び出し
     ↑実引数。関数に渡すデータ
```

JavaScript 07-01

次のスクリプトを入力し、実行を確認せよ。

js07-01.html

```
<html>
<head>
  <script type="text/javascript">
    function add(a,b)
    {
        var c;
        c = a+b;
        document.write(a + "+" + b + " = " + c + "<br>");
    }
  </script>
</head>

<body>
  <script type="text/javascript">
    add(1,2);
  </script>
</body>
```

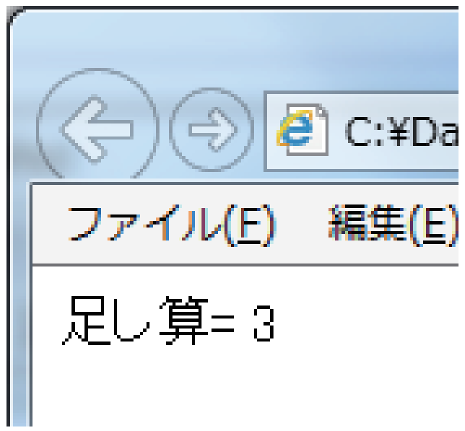


JavaScript 07-02

07-01 のスクリプトの関数を「値を戻す」ものに修正せよ。

js07-02.html

```
<html>
<head>
  <script type="text/javascript">
    function add(a,b)
    {
        var c;
        c = a+b;
        [          ]
    }
  </script>
</head>

<body>
  <script type="text/javascript">
    document.write(" 足し算 = " + add(1,2) );
  </script>
</body>
</html>
```



JavaScript 07-03

指定した画像ファイルを指定した回数、表示する関数を作成せよ。

js07-03.html

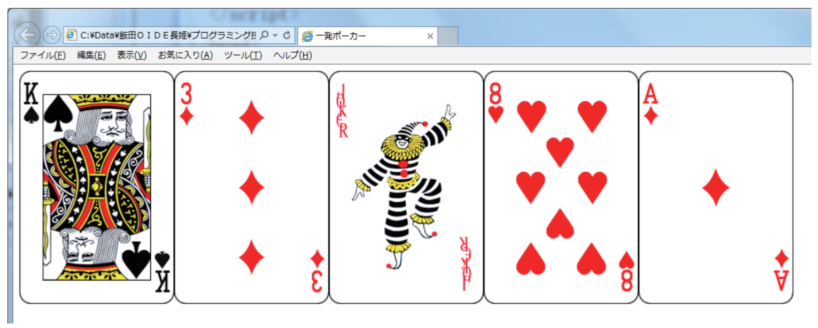
```
<html>
<head>
  <script type="text/javascript">
    function disp( img , n )
    {
        var i;
        for( i=0; i<n; i++){
            document.write(                              )
        }
        document.write("<br>");
    }
  </script>
</head>

<body>
  <script type="text/javascript">
    disp( "img/yui.gif" , 3 );
    disp( "img/boy1.gif" , 5 );
  </script>
</body>
</html>
```

JavaScript 07-04

m～n までの整数乱数を発生する関数「rnd」を作成する。
その関数を５回呼び出し「img/trump」フォルダ中の「1.png」～「53.png」を５枚分表示する。

js07-04.html

```
<html>
<head>
  <title> 一発ポーカー </title>
  <script type="text/javascript">
    function rnd(m,n)
    {
        return [                    ]   /* n ～ m の乱数 */
    }
  </script>
</head>

<body>
  <script type="text/javascript">
    var i,r;
    for(i=0;i<5;i++){
         r = rnd(1,53);  /* 1 - 53 の乱数 */
        document.write("<img src='img/trump/" + r +".png'>");
    }
  </script>
</body>
</html>
```



「リロード」 ボタンで何回か再読み込みしてみる

◆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
② addEventListener による方法
③ プロパティに指定する方法

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

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

○ 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右のようにします。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。

イベントの例

onClick	要素やリンクをクリックした時に発生
onDblClick	要素をダブルクリックした時に発生
onKeyUp	押していたキーをあげた時に発生
onKeyDown	キーを押した時に発生
onKeyPress	キーを押してる時に発生
onMouseOut	マウスが離れたした時に発生
onMouseOver	マウスが乗った時に発生
onMouseUp	クリックしたマウスを上げた時に発生
onMouseDown	マウスでクリックした時に発生

○ 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。

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。

○ 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。

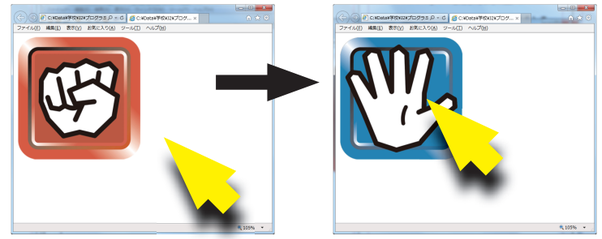
```
img.addEventListener("click",clickEvent);
```


JavaScript 08-01

ページに配置された画像「jan_goo.gif」の上にマウスカーソルを移動させると画像が「jan_paa.gif」に変わるスクリプト。

js08-01.html

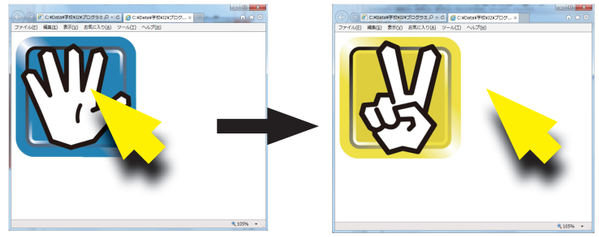
```
<html>
<head>
  <script type="text/javascript">
  </script>
</head>

<body>
    <img src="img/jan_goo.gif"                                    >
</body>
</html>
```



JavaScript 08-02

08-01 でマウスカーソルが画像から離れたら「jan_cyoki.gif」に変化するスクリプトを追加せよ

js08-02.html



JavaScript 08-03

08-02 を関数を使ったものに変更する。

js08-03.html

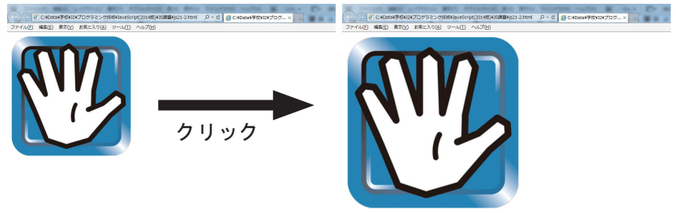
```
<html>
<head>
  <script type="text/javascript">
    function over(        )
     {
                                = "img/jan_paa.gif";
     }
    function out(        )
    {
                                = "img/jan_cyoki.gif"
    }

  </script>
</head>
<body>
    <img src="img/jan_goo.gif"                                                    >
</body>
</html>
```

JavaScript 08-04

08-03 に画像をクリックしたら大きさを「１．２倍」に拡大するスクリプトを追加せよ。

js08-04.html

```
<html>
<head>
  <script type="text/javascript">
    function over(　　)
      {
                                = "img/jan_paa.gif";
      }

    function out(　　)
    {
                                = "img/jan_cyoki.gif"
    }

    function clk(　　)
    {
                                *= 1.1;
    }
  </script>
</head>

<body>
    <img src="img/jan_goo.gif" onMouseOver="over(this)" onMouseOut="out(this)" onClick="clk(this)">
</body>
</html>
```

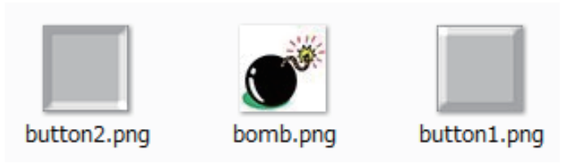


JavaScript 08-05

「img」ファルダ内の「button1.png」を縦横８×８個表示し、
クリックされた画像を「bomb.png」に変更するスクリプト。

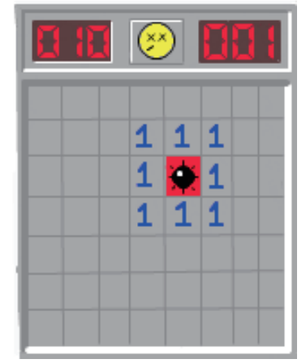
js08-05.html

```
<html>
<head>
  <script type="text/javascript">
    function change(obj)
      {
                                  /* 画像チェンジ */
      }
  </script>
</head>

<body>
    <script type="text/javascript">
        var i,j;
        for(i=0;i<8;i++){
            for(j=0;j<8;j++){
                document.write
            }
            document.write("<br>");
        }
    </script>
</body>
</html>
```



JavaScript 08-06

08-05 をベースに「マインスイーパー」もどきを作る。
５×５マスのどこかに爆弾を一つ乱数で設定する。
爆弾のマスをクリックしてしまったら背景色を黒色にする。



マインスイーパは、地雷が隠れているマスを開かずに、できるだけ短い時間で地雷の無いマス全てを開く（＝全ての地雷を見つける）ゲームである。地雷のある所を開いてしまうと失敗になる。名前の由来は、マイン＝地雷。スイーパー＝除去、掃除。Windows 3.1 の時代から Windows に標準搭載されており、パソコンに触った事の有る人であれば一度はプレイした事が有るであろう有名なゲームである。不朽の名作として紹介される事も多く、Windows 標準搭載ゲームの中で最も人気のあるゲームと言われている。

js08-06.html

```
<html>
<head>
  <script type="text/javascript">
    var bom;                          /* 爆弾の番号 グローバル変数 */
    function change(obj)
    {
        var id = obj.id;
        if(　　　　　　　　){
            obj.src = "img/bomb.png";
            document.bgColor = "black";
        }
        else
            obj.src="img/button2.png";
    }
  </script>
</head>
<body>
    <script type="text/javascript">
        var i,j,id;
        bom = Math.floor(Math.random()*(5*5));
        for(i=0;i<5;i++){
            for(j=0;j<5;j++){
                id =　　　　　　　　　　/* ＩＤ　0 ～ 24　*/
                document.write("<img id='" + id + "' src='img/button1.png' onClick ='change(this)'>");
            }
            document.write("<br>");
        }
    </script>
</body>
</html>
```

配置したオブジェクトＩＤ

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

◆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。

○ 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。

```
<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>";
```

○ 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右のようにします。

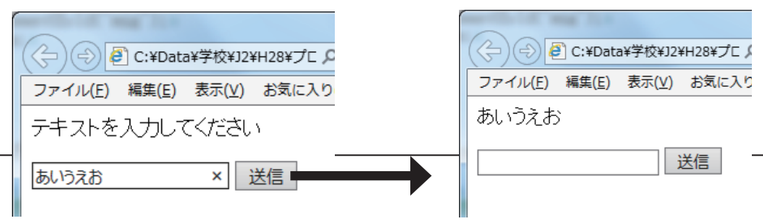
○ value プロパティ

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。

JavaScript 09-01　Ｗｅｂページに配置したテキストボックスの入力内容を表示する。

js09-01.html

```
<html>
<head>
     <script type="text/javascript">
          function bclick(){
               var t1 = document.getElementById(
               var m  = document.getElementById(
               m.innerHTML = t1.value;
               t1.value = "";
          }
     </script>
</head>
 <body>
     <p id="msg"> テキストを入力してください </p>
     <form>
          <input type="text" id="text1">
          <input type="button" value=" 送信 " onClick="bclick()">
     </form>
</body>
</html>
```



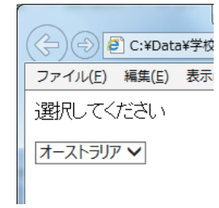
JavaScript 09-02　リスト、プルダウンリストの選択内容を表示する。

js09-02.html

```
<html>
<head>
     <script type="text/javascript">
     var str = "";
     function bclick(){
          var select1 = document.getElementById('select1').value;
          msg.innerHTML =
     }
     </script>
</head>
 <body>
     <p id="msg"> 選択してください </p>
     <form name="form1">
          <select id="select1" size=6 onChange="bclick();">
               <option value="australia.gif"> オーストラリア </option>
               <option value="canada.gif"> カナダ </option>
               <option value="japan.gif"> 日本 </option>
               <option value="korea.gif"> 韓国 </option>
          </select>
     </form>
</body>
</html>
```

size パラメータの省略→

size パラメーターを変更して実行してみる事



JavaScript 09-03　マウスの座標位置をテキストボックスに表示する

js09-03.html

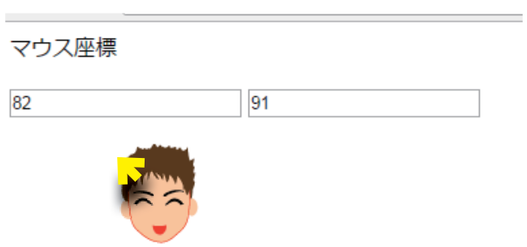
```
<html>
<head>
     <script type="text/javascript">
         function move(){
                var t1 = document.getElementById('text1');
                var t2 = document.getElementById('text2');
                                = event.clientX;
                                = event.clientY;;
         }
     </script>
</head>
 <body onMouseMove="move(event)">
     <p id="msg"> マウス座標 </p>
     <form>
          <input type="text" id="text1">
          <input type="text" id="text2">
     </form>
</body>
</html>
```

JavaScript 10-01

09-03 にマウスの移動に画像が追従するスクリプトを付け加えよ。

js10-01.html

```
<html>
<head>
     <script type="text/javascript">
     function move(){
               var t1 = document.getElementById('text1');
               var t2 = document.getElementById('text2');
               t1.value = event.clientX;
               t2.value = event.clientY;;
               var obj =
               obj.style.left = event.clientX;
               obj.style.top  = event.clientY;
     }
     </script>
</head>
<body onMouseMove="move(event)">
     <p id="msg">マウス座標</p>
     <form>
          <input type="text" id="text1">
          <input type="text" id="text2">
     </form>
     <img id="boy" src="img/boy1.gif" style="position:absolute">
</body>
</html>
```



画像は好きなものを使用

JavaScript 10-02

10-01 をマウスの移動で画像の大きさを変更するように修正せよ。
画像は左上を原点 (0,0) に固定し右下位置をマウス位置とする。
ただし、テキストボックスは取り除き下記のようなソースとする。

js10-02.html

```
<html>
<head>
<script type="text/javascript">
    function move(){
        var obj=document.getElementById("boy");
        obj.style.top    = 0;
        obj.style.left   =
        obj.style.width  = event.clientX;
        obj.style.height  =
    }
</script>
</head>
<body onMouseMove="move(event)">
     <img id="boy" src="img/boy1.gif" style="position:absolute">
</body>
</html>
```

JavaScript 10-03

10-02 を画像をクリッピングするものに変更する。

js10-03.html

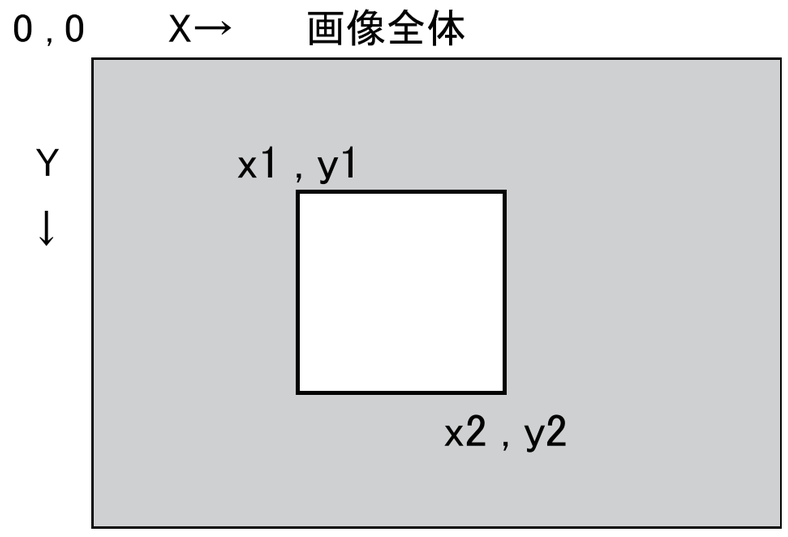
```
<html>
<head>
<script type="text/javascript">
    function clip(event)
    {
        var obj=document.getElementById("photo");
        var x = Math.floor(event.clientX);
        var y = Math.floor(event.clientY);
        obj.style.clip="rect(" + y + "," + (x+100) + "," + (y+100) + ", " + x + ")";
    }
</script>
</head>
<body onMouseMove="clip(event)">
    <img id="photo" src="img/oide1.jpg" style="position:absolute;left:0;top:0">
</body>
</html>
```

IE では動作しない場合は，Chrome で実行

クリップ領域を設定することで、任意領域の画像だけを表示することが出来ます。

```
obj.style.clip = rect( y1 , x2 , y2 , x1 ) ;
```

引数の順番に注意して設定してください。

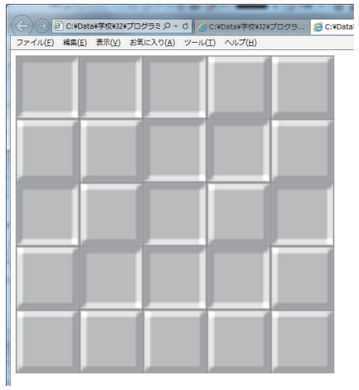


JavaScript 10-04

０８－０６を発展させてパズルゲーム「ライツアウト」もどきを作る。
「ライツアウト」はクリックされたマスの上下左右を反転させ、最終的にマスの画像を全て揃えるパズル。
ON と OFF の**２つの画像を変更して完成**とする。

js10-05.html

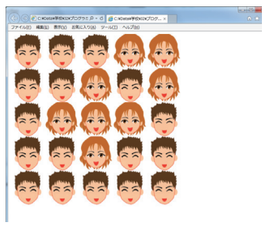
```
<html>
<head>
  <script type="text/javascript">
    var img1_path;                              // イメージ１ のフルパス
    function check(obj) {
        var x,y,no,id;
        no = Number(obj.id);                    // マスの番号
        hanten(no);                             // 画像を反転
        if(no-5 >=  0) hanten(no-5);            // 上
        if(                                     // 下
        if(                                     // 左
        if(                                     // 右
    }
    function hanten(no)   // 番号のピースを反転
    {
        var obj = document.getElementById(no)
        if(obj.src == img1_path)
            obj.src = "img/button2.png";
        else
            obj.src = "img/button1.png";
    }
  </script>
</head>
<body>
    <script type="text/javascript">
        var i,j,id;
        for(i=0;i<5;i++){
            for(j=0;j<5;j++){
                id =
                document.write("<img id='" + id + "' src='img/button1.png' width=100 onClick ='check(this)'>");
            }
            document.write("<br>");
        }
        img1_path = (document.getElementById("0" )).src   // 画像のフルパス
    </script>
</body>
</html>
```



※画像を変更して実行すること！！

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

配置したオブジェクトＩＤ



/* ＩＤ　０ ～ ２４　*/

※使用する画像は正方形

JavaScript 10-06

10-06 のライツアウトを１枚の画像を区切った形（画像のクリッピングを使用）のピースで作る。画像を変更して完成とする。

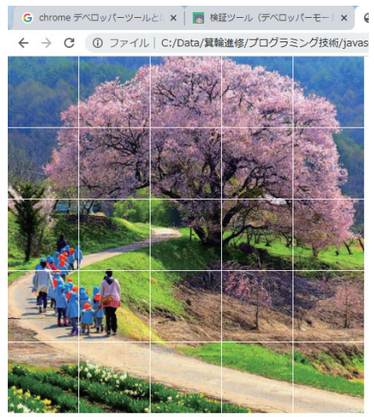
js10-06.html

※使用する画像は正方形

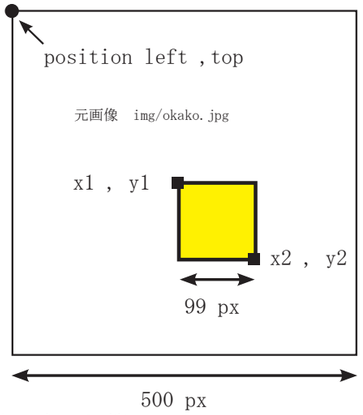
```
<html>
<head>
  <script type="text/javascript">
    var img1_path;        // イメージ１ のフルパス
    function check(obj) {
        ・・・省略
    }
    function hanten(no)   // 番号のピースを反転
    {
        var obj = document.getElementById(no)
        if(obj.src == img1_path)
            obj.src = "img/white.png";
        else
            obj.src = "img/sakura.jpg";
    }

  </script>
</head>
<body>
    <script type="text/javascript">
        var i,j,id,x1,y1,x2,y2;
        for(i=0;i<5;i++){
            for(j=0;j<5;j++){
                id = i*5+j;
                x1 = j*100;     y1 = i*100;        // クリップ領域の計算
                x2 = j*100+99;  y2 = i*100+99;
                document.write("<img id='" + id +
                "'  src='img/sakura.jpg'  width=500 style='" +
                "position: absolute; left:0 ;top:0; clip: rect(" +
                 y1  + "px " + x2 + "px " + y2 + "px " + x1 +
                 "px);'  onClick ='check(this)'>");        // クリッッピング
            }
        }
        img1_path = (document.getElementById("0" )).src
        //mondai( 3 );    // 出題関数呼び出し
    </script>
</body>
</html>
```

本来１行で記述すべきところをわかりやすさを考えて、複数行に分けて記述している。
￥マークで次行と続いている。

※領域の大きさは区切りの白枠を入れ、コマをわかりやすくするため９９ｐｘとする。



画像をクリッピングする



※後で追加してみる

```
function mondai( n )    // ｎ回クリック
{
    var i,no,obj;
    for(i=0;i<n;i++){
        no = Math.floor(Math.random()*25);
        obj = document.getElementById(no);
        check(obj);
    }
}
```

最初にランダムに何カ所かを自動でクリックする出題関数

◆タイマー処理

指定した時間が経過したときに何か処理を実行したい場合、タイマー処理という処理で実装することができます。タイマー処理には setInterval と setImeout の 2 種類があります。両者はとても似た関数ですが、少し挙動が異なります。

○ 2 種類の タイマー処理

setInterval…一定時間ごとに特定の処理を繰り返す

setTimeout…一定時間後に特定の処理をおこなう（繰り返さずに一度だけ）タイムアウト関数を使用します。

○ setTimeout メソッド

getTimeout メソッドを使えば、指定した関数を指定した時間経過後に呼び出すことができます。

次の例では func 関数は 1000 ミリ秒後に呼び出されれます。この関数をタイムアウト関数と呼びます

```
setTimeout( "func()" , 1000 );
```

タイムアウト関数の中で再度 setTimeout メソッドを実行すると、一定時間ごとに処理を行えます。

次の例で disp() 関数は 1000 ミリ秒（一秒）ごとに実行されます。

```
function disp(){
        setTimeout( "disp()" , 1000 );
        . . .
        . . . .
```

○ setInterval メソッド

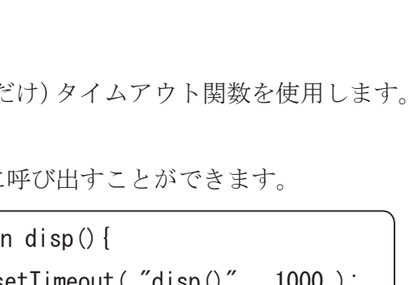
setInterval メソッドは一度実行するだけで、その後一定時間ごとに指定した関数を呼び出します。

実行を停止するためには clearInterval を使用します。

```
var id = setInterval("func()", 1000);
        . . .
        clearInterval(id);
```

JavaScript 11-01 指定時間ごとに画像を自動的に移動する。
（１００ミリ秒ごとに１０ｐｘ移動する）

```
<html>
<head>
<script type="text/javascript">
    var x=50;
    function move()
    {
        setTimeout("move()",100);
        var obj = document.getElementById("face");
        x = [          ]
        obj.style.left = x;
    }
</script>
</head>
<body onLoad="move()">
    <img id="face" src="img/boy1.gif" style="position:absolute;left:50;top:100">
</body>
</html>
```



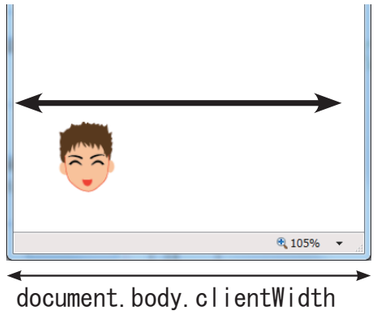
js11-01.html

右から左へ移動（←）させるためにはどうすればいいだろう？

JavaScript 11-02 画像を左右へ自動的に移動する。（画面の左右で跳ね返る）

```
<html>
<head>
<script type="text/javascript">
    var x=200,dx=10;
    function move()
    {
        setTimeout("move()",100);
        var obj = document.getElementById("face");
        x += [     ]
        obj.style.left = x;
        if (x<=0 || x+74 >= document.body.clientWidth )
            [                  ]
    }
</script>
</head>
<body onLoad="move()">
    <img id="face" src="img/boy1.gif" style="position:absolute;left:200;top:100">
</body>
</html>
```

js11-02.html

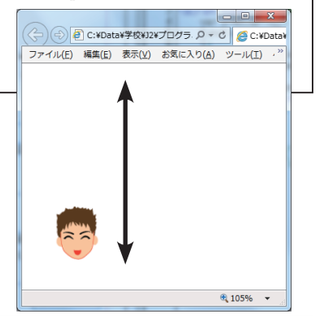


document.body.clientWidth

ブラウザ画面の横幅は document.body.clientWidth で取得

JavaScript 11-03 11-02 のスクリプトを画像を上下に移動するように変更せ

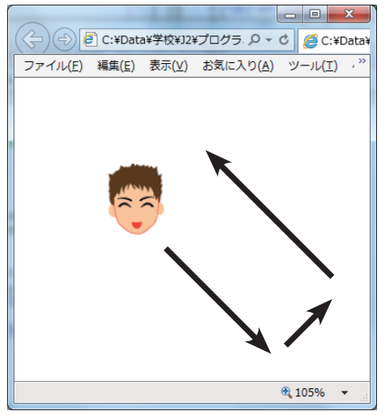
ブラウザ画面の高さは document.body.clientHeight



JavaScript 11-04 イメージがブラウザ内を斜めに跳ね返りながら移動するスクリプトを作れ。

```
<html>
<head>
<script type="text/javascript">
    var x=200,y=100;
    var dx=10,dy=10;
    function move()
    {
        setTimeout("move()",100);
        var obj = document.getElementById("face");
        x + =dx;
        y += dy;
        obj.style.left = x;
        obj.style.top = y;
        if (x<=0 || x+74>=document.body.clientWidth)
            [                  ]
        if (y<=0 || y+74>=document.body.clientHeight)
            [                  ]
    }
</script>
</head>
<body onLoad="move()">
    <img id="face" src="img/boy1.gif" style="position:absolute;left:200;top:100">
</body>
</html>
```

js11-04.html

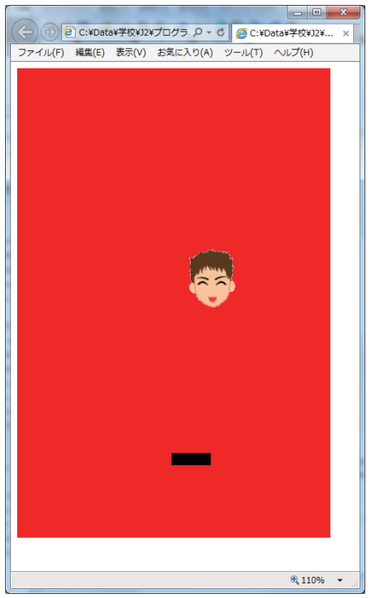


JavaScript 11-05

11-03 にラケット(画像 black.png)をマウスに追従しての移動させるスクリプトを追加せよ。(跳ね返す判定はなし) 更にゲーム領域として「red.png」を横 400px, 縦 600px で表示しこの範囲で動き回るように修正せよ。(一部の動作は IE でのみ可能)

js11-05.html

```
<html>
<head>
<script type="text/javascript">
    var x=200,y=100;
    var dx=10,dy=10;
    var rx,ry;
    function move()
    {
        setTimeout("move()",50);
        var obj1=document.getElementById("face");
        x+=dx;
        y+=dy;
        obj1.style.left=x;
        obj1.style.top=y;
        if (x<=0 || x+70>=400) dx=-dx;
        if (y<=0 || y+70>=600) dy=-dy;
    }

    function racket(event)
    {
        var obj = document.getElementById("bar");
            rx = event.clientX;
            ry = 500;
            obj.style.left = rx;
            obj.style.top
    }
</script>
</head>
<body onLoad="move()" onMouseMove="racket(event)">
    <img src="img/red.png" width=400 height=600>
    <img id="face" src="img/boy1.gif" style="position:absolute;left:200;top:100">
    <img id="bar" src="img/black.png" width=50 height=15 style="position:absolute">
</body>
</html>
```



JavaScript 11-06

11-04 にボールとラケットの当たり判定を追加し、簡単なラケットゲームを完成せよ。ボールの画像は変更すること。ボールを打ち損ねたら背景を黒色にし、ボールを止める。

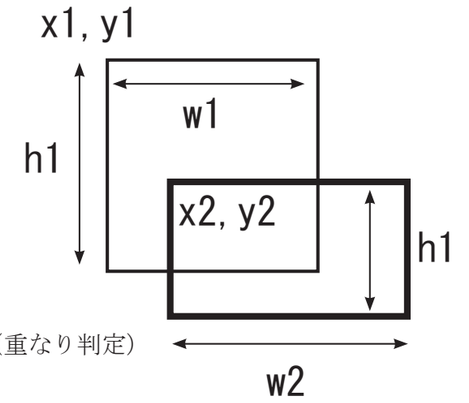
js11-06.html

```
<html>
<head>
<script type="text/javascript">
    var x=200,y=100;
    var dx=10,dy=10;
    var rx,ry;
    function move()
    {
        setTimeout("move()",50);
        var obj1=document.getElementById("face");
```

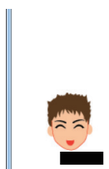
次ページへ続く

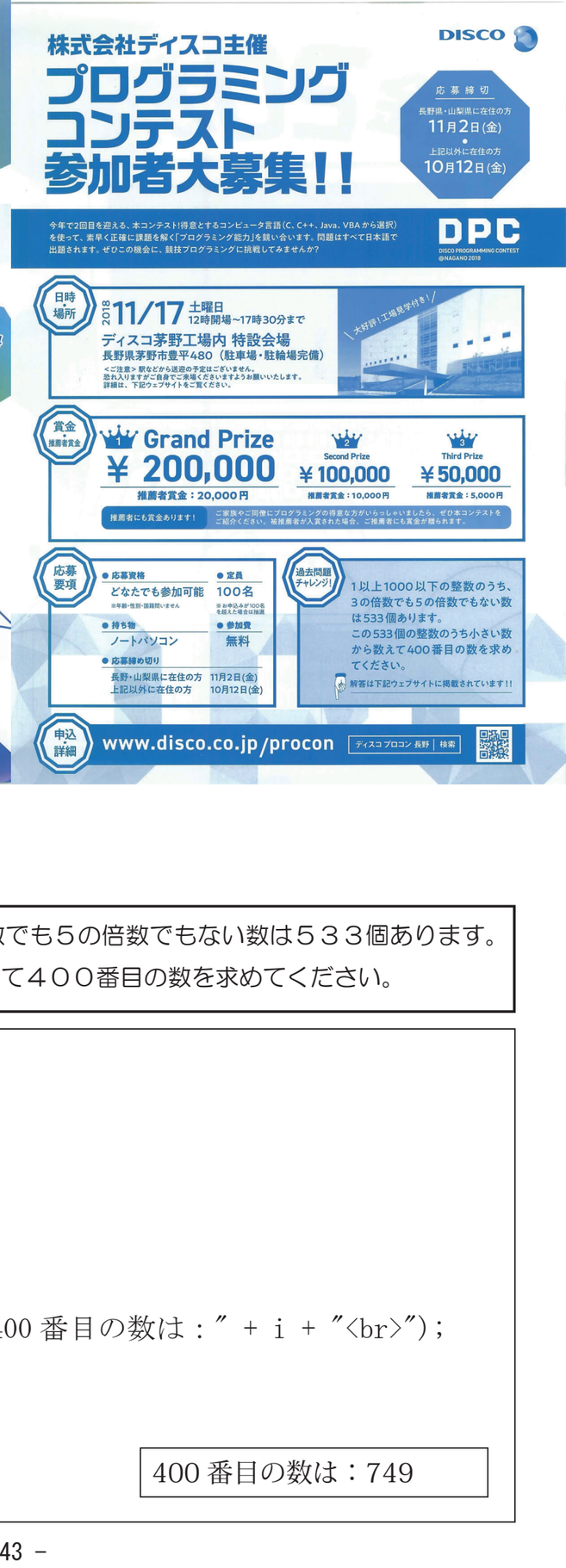
js11-06.html
前ページより

```
        var obj2 = document.getElementById("bar");
        x+=dx;
        y+=dy;
        obj1.style.left=x;
        obj1.style.top=y;
        if (x<=0 || x+70>=400) dx=-dx;
        if (y<=0)              dy=-dy;
        if(y+70>=600 ) {              // ボールが領域から出たら
            document.bgColor="black"; // 背景を黒色
            dx=0; dy=0;               // ボールの移動量０
         }
        if( overlap( obj1 , obj2) == 1 ) dy = -dy;

    }

    function racket(event)
    {
        var obj = document.getElementById("bar");
            rx = event.clientX;
            ry = 500;
            obj.style.left = rx;
            obj.style.top  = ry;
    }
```




```
    function overlap( obj1 , obj2 )  // 衝突判定関数（重なり判定）
    {
        var x1,y1,w1,h1,x2,y2,w2,h2;
        x1 = obj1.offsetLeft;   x2 = obj2.offsetLeft;
        y1 = obj1.offsetTop;    y2 = obj2.offsetTop;
        w1 = obj1.width;        w2 = obj2.width;
        h1 = obj1.height;       h2 = obj2.height;
        if(( x2 < x1+w1  &&  x2+w2 > x1 ) && (  y2 < y1+h1  &&  y1 < y2+h2 ) )
            return 1;  // ２つのオブジェクトが重なっていれば 1 を返す
        else
            return 0;
    }
/script>
</head>
<body onLoad="move()" onMouseMove="racket(event)">
    <img src="img/red.png" width=400 height=600>
    <img id="face" src="img/boy1.gif" style="position:absolute;left:200;top:100">
    <img id="bar" src="img/black.png" width=50 height=15 style="position:absolute">
</body>
</html>
```





チラシの過去問題

１以上１０００以下の整数のうち、３の倍数でも５の倍数でもない数は５３３個あります。
この５３３個の整数のうち小さい数から数えて４００番目の数を求めてください。

```
<body>
<script type="text/javascript">
    var i,cnt=0;
    for( i=1; i<=1000; i++){
        if( i%3!=0 && i%5!=0 ){
            cnt++;
            if( cnt == 400 )
                document.write("400 番目の数は：" + i + "<br>");
        }
    }
</script>
</body>
```

400 番目の数は：749

チャレンジ！練習問題（実際に出題された問題ではありません。）

Q1

きつね蕎麦 650 円、山菜蕎麦 780 円、鴨南蛮蕎麦 980 円。
賞金 20 万円で、お釣りなく、最も多く食べるには？

```
<body>
    <script type="text/javascript">
        var kitu,san,kamo,kei,hai,n=0;
        for(kitu=307; kitu>=1; kitu--){
            for(san=1;san<256;san++){
                for(kamo=1;kamo<204;kamo++){
                    kei = kitu*650 + san*780 + kamo*980 ;
                    hai = kitu + san + kamo;
                    if(kei == 200000){
                       n++;
                       document.write(n + "きつね (" + kitu + ") 山菜 (" + san + ") 鴨南蛮 (" + kamo + ")  合計 " + hai + "杯<br>");
                    }
        }}}
    </script>
</body>
```

Q2

１～ 1000 までの間で３と５の両方で割り切れる数の
20 番目と 40 番目と 60 番目の合計は？

```
<html>
<body>
    <script type="text/javascript">
        var n,cnt=0,sum=0;
        for(n=1; n<=1000; n++){
            if( n%3==0  &&  n%5==0 ){
            cnt++;
                document.write(cnt + "  " + n +"<br>");
             if(cnt==20 || cnt==40 || cnt==60)
                sum = sum + n;
          }
        }
        document.write("<br>答え  " + sum );
    </script>
</body>
</html>
```


Q3

次の 300 個の名字の文字列の中で、3 番目に多いアルファベットとその数は？

```
<html>
<head>
  <script type="text/javascript">
  </script>
</head>

<body>
    <script type="text/javascript">
//      以下の文字列を使用して問題を解いてください。
//      データ型は任意
//      main のローカル変数として定義してください。

var d=[
"SATO","SUZUKI","TAKAHASHI","TANAKA","WA
・・・省略
KAMOTO","HORIGUCHI","KITAJIMA","TOKUDA","KA
];

            var cnt=[];
            for(i=0;i<26;i++)       // カウント用配列クリア
            cnt[i]=0;

            for( var n=0; n<300; n++ ){                    // ３００人分繰り返す
                  for( var m=0; m<d[n].length; m++ ){
                var c = d[n].substr(m,1);       // １文字取り出し
                cnt[ c.charCodeAt(0)-65 ] ++;   // 文字カウント
            }
            }

            for(i=0; i<26; i++){                           // 順位付け、表示
                  document.write( String.fromCharCode(i+65) + "(" + cnt[i] +")");
                  var jyuni=1;
                  for(n=0;n<26;n++)
                        if( cnt[i] < cnt[n] ) jyuni++;
                  document.write( jyuni + " 位<br>");
        }

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

```
//      以下の文字列を使用して問題を解いてください。
//      データ型は任意
//      main のローカル変数として定義してください。
{
"SATO","SUZUKI","TAKAHASHI","TANAKA","WATANABE","ITO","YAMAMOTO","NAKA
MURA","KOBAYASHI","KATO",
"YOSHIDA","KOYANAGI","SASAKI","YAMAGUCHI","MATSUMOTO","INOUE","SAITO","KI
MURA","HAYASHI","SHIMIZU",
"IKEDA","ABE","MORI","HASHIMOTO","YAMASHITA","ISHIKAWA","NAKAJIMA","MAEDA
","FUJITA","OGAWA",
"OKADA","GOTO","HASEGAWA","MURAKAMI","KONDO","ISHI","SAKAMOTO","ENDO","
AOKI","NISHIMURA",
"FUKUDA","MIURA","FUJIWARA","MATSUDA","OKAMOTO","NAKAGAWA","HARADA","
TAKEUCHI","TAMURA","KANEKO",
"NAKAYAMA","ISHIDA","UEDA","NAKANO","MORITA","SHIBATA","YOKOYAMA","KUDO
","MIYAZAKI","TAKADA",
"ANDO","SAKAI","TAKAGI","KAWAMOTO","KOJIMA","MURATA","TAKEDA","UENO","SU
GIYAMA","MASUDA",
"SUGAWARA","HIRANO","KOYAMA","KUBO","CHIBA","MATSUI","IWASAKI","NOGUCHI","
MATSUO","KINOSHITA",
"KIKUCHI","NOMURA","SANO","WATABE","ARAI","SUGIMOTO","SAKURAI","FURUKAW
A","ICHIKAWA","SHIMADA",
"KOMATSU","TAKANO","MIZUNO","YOSHIKAWA","YAMAUCHI","NISHIDA","KIKUCHI","K
ITAMURA","HAMADA","IGARASHI",

"YASUDA","KAWAGUCHI","HIRATA","KAWASAKI","AZUMA","HONDA","KUBOTA","YOS
HIMURA","TSUJI","NAKANISHI",
"FUKUSHIMA","IWATA","HATORI","HIGUCHI","MATSUOKA","YAMANAKA","NAGAI","TA
GUCHI","AKIYAMA","TSUCHIYA",
"ISHIHARA","MATSUSHITA","BABA","YOSHIOKA","KOIKE","ASANO","NODA","KAWAMU
RA","HIROSE","HOSHINO",
"KURODA","OZAKI","TANABE","NAGATA","HORI","SUGANO","NISHIYAMA","KATAYAM
A","HIRAI","SAWADA",
"HONMA","HAYAKAWA","YOKOTA","ARAI","OKAZAKI","KAMATA","NARITA","MIYATA","
ODA","SUDO",
"SHINOHARA","KURIHARA","MIYAKE","OZAWA","FUKUI","UMEMOTO","MINAMI","OKUM
URA","KATAOKA","UCHIYAMA",
"KUWAHARA","OKA","TOMITA","SEKIGUCHI","MATSUNAGA","OKUDA","KITAGAWA","
MITANI","KOGA","UEHARA",
"YAGI","YOSHINO","SHIRAISHI","IMAMURA","KAMIMURA","KOIZUMI","NAKAO","NANBA
","AOYAMA","HIRAYAMA",
"MAKINO","TERADA","SHIBUYA","SAKAGUCHI","KAWAI","ADACHI","YAMADA","AMAN
O","NISHI","OGURA",
"SUGIURA","KAKUTA","MIZUTANI","NEMOTO","TSUJIMOTO","SEKINE","MORISHITA","S
AKUMA","TSUKAMOTO","UEDA",

"HORIUCHI","INAGAKI","NAKATANI","TANI","MATSUZAKI","HOSOKAWA","ENOMOTO","
OKABE","NISHIO","HATAKEYAMA",
"TASHIRO","NAKAHARA","TSUDA","KUROKI","NAGAO","KANAI","SAEKI","NISHIOKA","
HOSHI","YONEDA",
"TSUTSUMI","NOZAKI","OCHIAI","IZUMI","MACHIDA","YAMAZAKI","SUGITA","TOKUNAG
A","KASAHARA","TAKIZAWA",
"MIKI","KAJIWARA","SUDA","HIDAKA","MURAI","HIROTA","HORIE","FUJISAWA","KAWAB
ATA","IGUCHI",
"MUKAI","TAKENAKA","SAKAKIHARA","YASUI","TSUKADA","YANAGISAWA","OGATA"
,"KURITA","UNO","KUBOTA",
"KAMEI","MOGI","MIWA","TAKAI","YAMAMURA","FURUTANI","NAGASHIMA","YOSHIZA
WA","AOYAGI","SHINOZAKI",
"DEGUCHI","OGINO","KOMORI","ODAIRA","TAKASE","INADA","SONODA","YOKOI","TOMI
NAGA","TAKAMATSU",
"KANESHIRO","KOCHI","YANAGIDA","HORIKAWA","TANIMOTO","NINOMIYA","IKEGAMI","
HIGA","HANADA","USUI",
"KATAGIRI","SUGIHARA","ARAYA","KAWARA","SENDA","YASHIMA","FUKUHARA","AIZ
AWA","KURATA","TABATA",
"MATSUSE","IWASE","TAGAMI","MORIOKA","NAKAMOTO","HORIGUCHI","KITAJIMA","T
OKUDA","KASHIWAGI","AKASE",
};
```

チラシの過去問題

１以上１０００以下の整数のうち、３の倍数でも５の倍数でもない数は５３３個あります。
この５３３個の整数のうち小さい数から数えて４００番目の数を求めてください。

```
<body>
<script type="text/javascript">
    var i,cnt=0;

    for( i=1; i<=1000; i++){
        if( i%3!=0 && i%5!=0 ){
            cnt++;
            if( cnt == 400 )
                document.write("400 番目の数は：" + i + "<br>");
        }
    }
</script>
</body>
```

400 番目の数は：749

Q1　２０１８（Ｈ３０） チャレンジ！練習問題

５～ 100 の間の数値を、５で割って、余りが２もしくは４になる数値の合計を解答してください。

```
<html>
<head>
</head>
<body>
<script type="text/javascript">
    var i,sum=0;

    for( i=5; i<=100; i++){
        if( i%5==2 || i%5==4 )
            sum += i;
    }
    document.write("合計は：" + sum + "<br>");
</script>
</body>
</html>
```

合計は：2014

Q2

長野 "NAGANO" は何個ありますか？
テンプレートを使用して解答してください。

テンプレート　"NAGANOKANAGAWATOKYONAGANOKANAZAWAAOMORINAGANOTOKYO"

ヒント　JavaScript の文字列操作

```
str = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
文字列の長さ　　…　str.length　→ 26
文字列切り出し　…　str.substr( 開始位置 , 文字数 )　str.substr(2,3) → BCD
　　s = str.substr(3,5);　　s → CDEFG
文字列の比較　　…　if ( str == "ABC" ){ ・・・
```

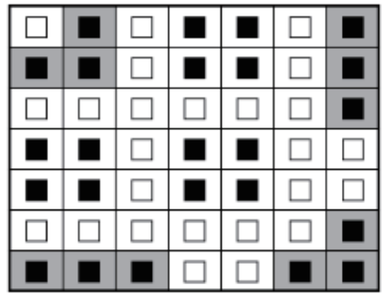
```
<body>
<script type="text/javascript">
    var str="NAGANOKANAGAWATOKYONAGANOKANAZAWAAOMORINAGANOTOKYO";
    var i,cnt=0;
    for(i=1;i<=str.length;i++){
        var s=str.substr(i,6);
        document.write(s);
        if( s == "NAGANO"){
            cnt++;
            document.write(" " + cnt + " 個目 ");
        }
        document.write("<br>");
    }
</script>
</body>
```

```
AGANOK
GANOKA
ANOKAN
NOKANA
OKANAG
KANAGA
ANAGAW
NAGAWA
AGAWAT
GAWATO
AWATOK
WATOKY
ATOKYO
TOKYON
OKYONA
KYONAG
YONAGA
ONAGAN
NAGANO 1 個目
AGANOK
GANOKA
ANOKAN
NOKANA
OKANAZ
KANAZA
ANAZAW
NAZAWA
AZAWAA
ZAWAAO
AWAAOM
WAAOMO
AAOMOR
AOMORI
OMORIN
MORINA
ORINAG
RINAGA
INAGAN
NAGANO 2 個目
AGANOT
GANOTO
ANOTOK
NOTOKY
OTOKYO
TOKYO
OKYO
KYO
YO
O
```

Q3

(1) ■が縦または横につながっている一番小さい塊は、何塊ありますか？
テンプレートを使用して解答してください。

例



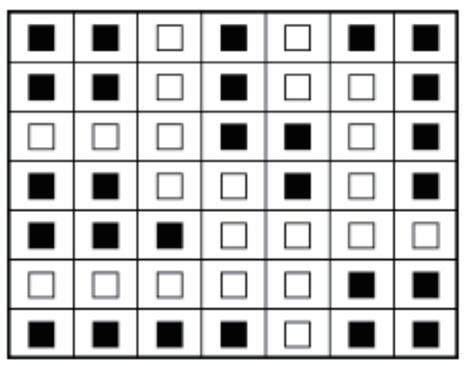
■(1)/□(0)

■(1)の一番小さい塊は、3個の塊です。その塊が4個あります

※斜めに接するものは塊としてカウントしません

解答

4



■(1)/□(0)

テンプレート

```
{1, 1, 0, 1, 0, 1, 1},
{1, 1, 0, 1, 0, 0, 1},
{0, 0, 0, 1, 1, 0, 1},
{1, 1, 0, 0, 1, 0, 1},
{1, 1, 1, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 1, 1},
{1, 1, 1, 1, 0, 1, 1},
```



```
<html>
<head>
  <script type="text/javascript">
    var d =[ [1,1,0,1,0,1,1],                    // テンプレート
             [1,1,0,1,0,0,1],
             [0,0,0,1,1,0,1],
             [1,1,0,0,1,0,1],
             [1,1,1,0,0,0,0],
             [0,0,0,0,0,1,1],
             [1,1,1,1,0,1,1] ] ;
    var b = ["□","■","■"];                       // パターン表示用
    var cnt = [0,0,0,0,0,0,0,0,0,0,0,0];         // カウント用配列
    function chk(x,y)
    {
      var n=0;
            if(x<0 || x>6 || y<0 || y>6)         // 枠外なら 0 を返す
                return 0;
            if( d[y][x]==0 || d[y][x]>=2)        // 0 か調査済なら 0 を返す
                return 0;
            n++;                                 // 連なりをカウント
            d[y][x]=2;                           // 調査済は 2 に
            n = n + chk( x   , y-1 );            // 上を調べる　再帰呼び出し
            n = n + chk( x   , y+1 );            // 下　　〃
            n = n + chk( x-1 , y   );            // 左　　〃
            n = n + chk( x+1 , y   );            // 右　　〃
            return n;                            // つながっている個数を返す
    }
  </script>
</head>
<body>
  <tt><font size=6>
  <script type="text/javascript">
    var x,y,a;
    for( y=0; y<7; y++){
            for( x=0; x<7; x++){
                  document.write( b[ d[y][x] ] );      // 配列の内容を表示
                  a = chk(x,y);                        // １マスごとチェック関数コール
                  cnt[ a ]++;                          // 連なっていた個数をカウント
            }
            document.write("<br>");
      }
      for(i=1;i<10;i++){                               // 小さい方から一つ目を表示
            if(cnt[i]!=0)
              document.write(i + " 個の塊が " + cnt[i] + " つあります<br>");
      }
  </script>
  </font></tt>
</body>
</html>
```

```
■■□■□■■
■■□■□□■
□□□■■□■
■■□□■□■
■■■□□□□
□□□□□■■
■■■■□■■
4 個の塊が 3 つあります
5 個の塊が 3 つあります
```

Q1　チャレンジ！練習問題

５～100の間の数値を、５で割って、余りが２もしくは４になる数値の合計を解答してください。

Q2　チャレンジ！練習問題

次のような「虫食い算」があります。
ｆｏｒの二重ループを使って、この答えを求めるプログラムを作りなさい。

虫食い算

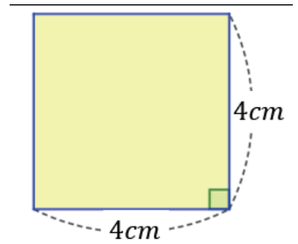
```
   1□2
×   5□
-------
  7392
```

上の数字をａとします。
ａは１０２～１９２のいずれかです。

下の数字をｂとします。
ｂは５０から５９のいずれかです。

Q3　チャレンジ！練習問題

１辺の長さが「４」の四角形は
面積と外周が同じ「１６」になります。
これと同じ性質を持つ四角形を、
１辺の長さが１０までについて探しなさい。

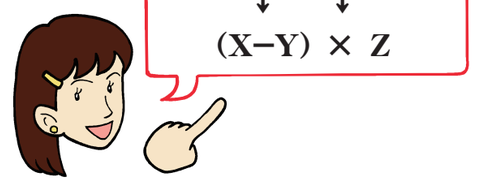


Q4　チャレンジ！練習問題

次の式は「＋」を「－」に「÷」を「×」に変えても計算結果は同じである。０以外の１ケタの数Ｘ，Ｙ，Ｚを見つけよ。答えは２通りある。

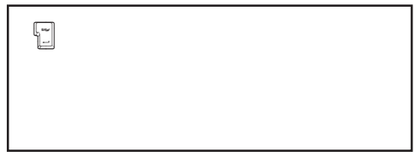
$$(X+Y) \div Z = (X-Y) \times Z$$

ヒント：for の３重ループを使う。



Q5　チャレンジ！練習問題

「空きビン３本で新品を１本」くれるという
キャンペーン中の牛乳があります。
次のとき、最終的に何本の牛乳が飲めますか？
　　①最初に、５本買ったとき。
　　②最初に、１７本買ったとき。
　　③最初に、６７本買ったとき。



```
main()
{
    int a,b=0,c=0;
    scanf("%d",&a);              最初の本数を入力
    while( __________){          牛乳がある間繰り返す
        __________                 １本飲む
        ________                   飲んだ本数をカウント
        ________                   空きビンが１本増える
        if (__________){           もし、空きビンが３本になったら
            __________                   １本もらえる
            ________                     空きビンは引き取られ０にな
る
        }
    }
    printf(" 全部で %d 本飲める ",c);
```

使用している変数

a・・・中身の入っている本数
b・・・空きビンの本数
c・・・飲んだ本数

Q5　チャレンジ！練習問題

「空きビン３本で新品を１本」くれるという
キャンペーン中の牛乳があります。
次のとき、最終的に何本の牛乳が飲めますか？
　　①最初に、５本買ったとき。
　　②最初に、１７本買ったとき。
　　③最初に、６７本買ったとき。

https://www.disco.co.jp/procon/backnumber/hiroshima2019/

ＣＡＮＶＡＳを使用したグラフィック処理

Canvas とは、HTML5 から新しく追加された図形を描くための技術仕様で、HTML の <canvas> 要素と JavaScript を組み合わせて図形を描画します。 JavaScript 以外のスクリプトでも描画可能ですが、ほとんどの場合 JavaScript が標準的に利用されています。
四角形や円をはじめとする様々な図形の描画、それらの変形、グラデーションを含む色や背景などのスタイル、画像やテキストとの合成などの表現が可能となります。

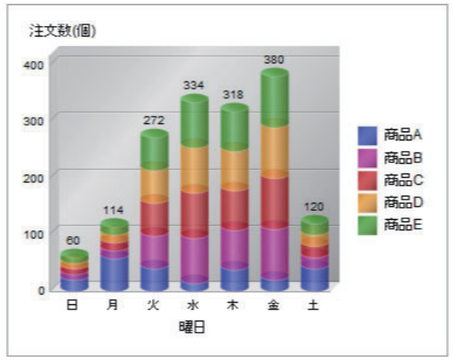
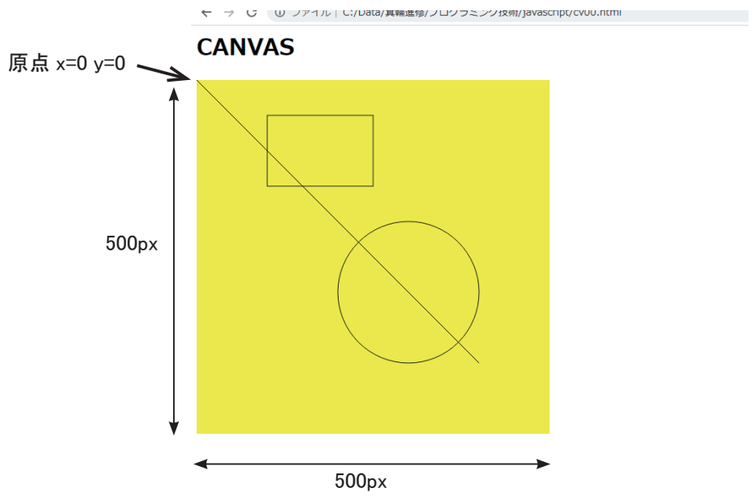


図1：Canvasを使ったグラフ描画の例

JavaScript cv01 CANVAS を使用して四角形、円、直線を描くための例

cv01.html

```
<html>
<head>
    <title> CANVAS </title>
</head>
 <body>
    <h1>CANVAS</h1>
    <canvas id="cvs1" width=500 height=500 style="background-color:yellow;"">
    </canvas>
      <script type="text/javascript">
        var sum,x,y,n,i;
        var canvas = document.getElementById("cvs1"); // canvas オブジェクト取得
        var ctx = canvas.getContext("2d");            // 2D 描画コンテキストの取得

        ctx.strokeRect(100,50,150,100);  // 左上座標 100,50 幅 100 高さ 50 の四角形を描く

        ctx.arc(300,300,100,0,Math.PI*2,true);     // 中心 300,300 半径１00px の円

        ctx.moveTo(0,0);                           // 直線引き始め
        ctx.lineTo(400,400);                       // 直線引き終わり
        ctx.stroke();                              // 直線を一括で描画

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



おもなＣＡＮＶＡＳ描画メソッドとプロパティ

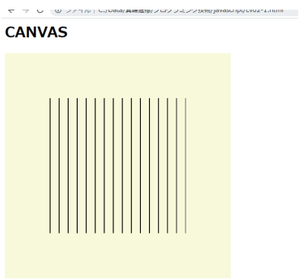
メソッド	
パス	
beginPath()	パスを開始
closePath()	パスを閉じる
線	
moveTo(x,y)x	，yに移動
lineTo(x,y)x	，yまで線を引く
描画	
stroke()	描画
fill()	塗りつぶし描画
四角形	
fillRect(x, y, width, height)	塗りつぶし四角形
strokeRect(x, y, width, height)	塗りつぶしでない四角形
clearRect(x, y, width, height)	指定範囲の描画をクリア
テキスト	
strokeText(str, x, y)	テキストを描く
fillText(str, x, y)	テキストを描く（塗りつぶし）
円	
arc(x, y, r, s, e, c)円	
arc(x1, y1, x2, y2, r)	点1と点２を半径rの円弧でつなぐ
clip()	クリッピング(パスで切り抜き)
グラデーション	
createLinearGradient(x1, y1, x2, y2)	1から2にグラデーション
createRadialGradient(x1, y1, r1, x2, y2, r2)	1から2にグラデーション
画像	
drawImage(img, x, y)	画像を描画
drawImage(img, x, y, width, height)	画像を描画（widthとheightを指定）
createPattern(img, repeat)	パターン作成
スケール	
scale(x, y)	変形(拡大縮小)
プロパティ	
線	
strokestyle	線の色
lineWidth	線の太さ
lineCap	線の端(butt, round, square)
塗りつぶし	
fillStyle	塗りつぶしの色
フォント	
font	フォント
影	
shadowBlur	影
shadowColor	影の色
shadowOffsetX	影の幅
shadowOffsetY	影の高さ

JavaScript cv02-1

扱いやすいように、引き始め、引き終わりの２点座標を与えて直線を引く関数を作り、
実行例のように直線を連続して描くプログラムを作る。

cv02-1.html

```
<html>
<head>
    <title> 連続した縦線 </title>
    <script type="text/javascript">
        function line(X1,Y1,X2,Y2)     // line 関数（変数名は大文字）
        {
            ctx.moveTo(X1,Y1);     // 直線引き始め
            ctx.lineTo(X2,Y2);     // 直線引き終わり
            ctx.stroke();          // 一括で描画
        }
    </script>
</head>
 <body>
    <h1>CANVAS</h1>
    <canvas id="cvs1" width=500 height=500 style="background-color:lightyellow;"">
    </canvas>                                                    // 背景色　ライトイエロー
      <script type="text/javascript">

        var x;
        var canvas = document.getElementById("cvs1");     // canvas オブジェクト取得
        var ctx = canvas.getContext("2d");                // 2D 描画コンテキストの取得

        for( x=100; x<=400;x+=20){          //  x 座標を移動する
            line(x,100,x,400);              // line 関数 呼び出し
        }
      </script>
</body>
</html>
```

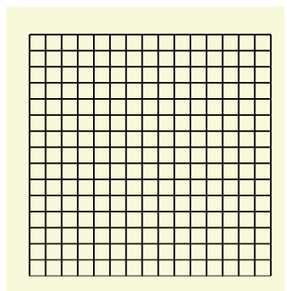


JavaScript cv02-2

実行例のように直線を縦横連続して格子を描くプログラムを作る。

cv02-2.html

```
      ・・・省略
</head>
 <body>
    <h1>CANVAS</h1>
    <canvas id="cvs1" width=500 height=500 style="background-color:lightyellow;"">
    </canvas>
      <script type="text/javascript">

        var x,y;
        var canvas = document.getElementById("cvs1");     // canvas オブジェクト取得
        var ctx = canvas.getContext("2d");                // 2D 描画コンテキストの取得

        for( x=100; x<=400;x+=20){
            line(x,100,x,400);
        }
        for( y=100; y<=400;y+=20){
            line(100,y,400,y);
        }

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

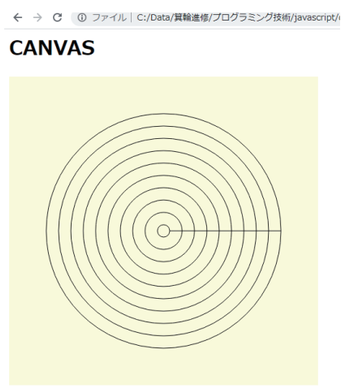


JavaScript cv02-3

実行例のように半径を 10px ～ 200px まで変化させながら円を描く（同心円）

cv02-3.html

```
・・・省略
</head>
 <body>
    <h1>CANVAS</h1>
    <canvas id="cvs1" width=500 height=500 style="background-color:lightyellow;"">
    </canvas>
      <script type="text/javascript">

        var x,y,r;

        x=250; y=250;

        var canvas = document.getElementById("cvs1");
        var ctx = canvas.getContext("2d");

        for( r=10; r<=200; r+=20){                  // 半径を変化
            ctx.arc(x,y,r,0,Math.PI*2,true);        // 円を描く
        }
        ctx.stroke();                              // まとめて描画
      </script>
</body>
</html>
```

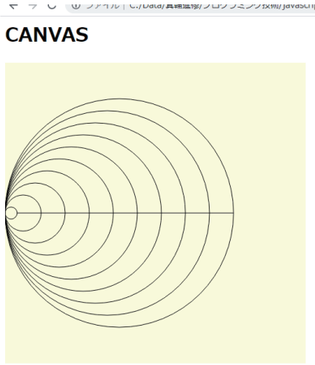


JavaScript cv02-4

実行例のように半径と中心座標を変化させながら円を描く

cv02-4.html

```
・・・省略
</head>
 <body>
    <h1>CANVAS</h1>
    <canvas id="cvs1" width=500 height=500 style="background-color:lightyellow;"">
    </canvas>
      <script type="text/javascript">

        var x,y,r;

        x=250; y=250;

        var canvas = document.getElementById("cvs1");
        var ctx = canvas.getContext("2d");

        for( r=10; r<=200; r+=20){              // 半径を変化
            ctx.arc(r,y,r,0,Math.PI*2,true);    // 中心座標も変化
        }
        ctx.stroke();                          // まとめて描く
      </script>
</body>
</html>
```



JavaScript cv03-1

実行例のようにサインカーブを描くプログラムを作成せよ。
数学組み込み関数の「Math.sin()」を使用する。

cv03-1.html

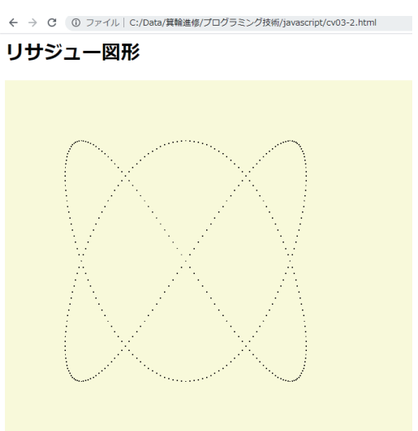
```
・・・省略
 <body>
    <h1> サインカーブ </h1>
    <canvas id="cvs1" width=800 height=800 style="background-color:lightyellow;"">
    </canvas>      // CANVAS サイズを少し大きくする
      <script type="text/javascript">

        var x,y,x1,y1,k,r;
        var rd=3.141592/180.0;  // RAD 変換のための定数

        y=300; r=100;              // 中心Ｙ座標、半径

        var canvas = document.getElementById("cvs1");
        var ctx = canvas.getContext("2d");

        for( k=0; k<=720; k++){          // 角度を変化
             y1 = Math.sin(k*rd)*r+y;    // Ｙ座標を計算
            line(k,y1,k,y1+1);           // 点を描画
        }

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

JavaScript cv03-2

実行例のように「リサジュー図形」を描くプログラムを作れ。
縦Ｙ横Ｘの比率を変えて、何度か実行して見よ。

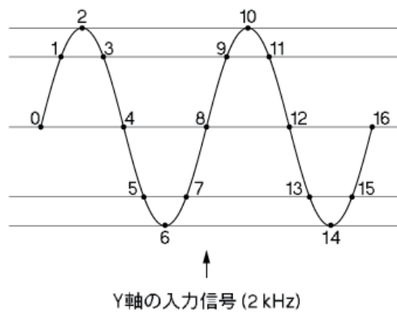
cv03-2.html

```
・・・省略
 <body>
    <h1> リサジュー図形 </h1>
    <canvas id="cvs1" width=800 height=800 style="background-color:lightyellow;"">
    </canvas>
      <script type="text/javascript">

        var x,y,k,x1,y1,r;
        var rd=3.141592/180.0;     // RAD 変換のための定数

        x=300; y=300; r=200;       // 中心座標、半径

        var canvas = document.getElementById("cvs1");
        var ctx = canvas.getContext("2d");

        for( k=0; k<=360; k++){            // 角度を変化
            x1 = Math.sin(k*rd*2)*r+x;     //  x座標計算
             y1 = Math.sin(k*rd*3)*r+y;    //  y座標計算
            line(x1,y1,x1,y1+1);           // 点を描画
        }

      </script>
</body>
```

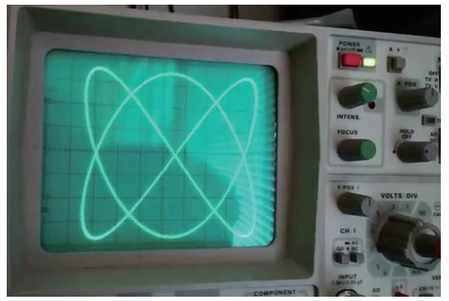


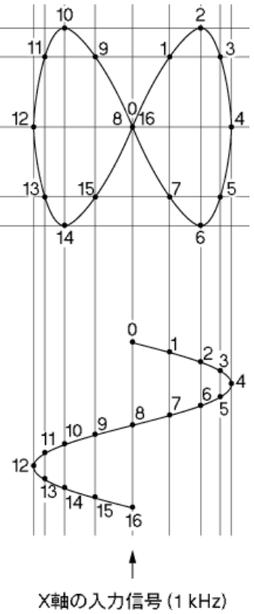
リサジュー図形とは

オシロスコープの垂直軸と水平軸に同時に交流信号（正弦波）を入力することにより数多くの図形パターンがスクリーンに描かれますが、これをリサジュー図形といいます。



両方の波形の周波数や位相の変化でシンプルなパターンだったり非常に複雑なパターンを描いたり、また回転しているように見えたり、静止したりと変化を繰り返します。



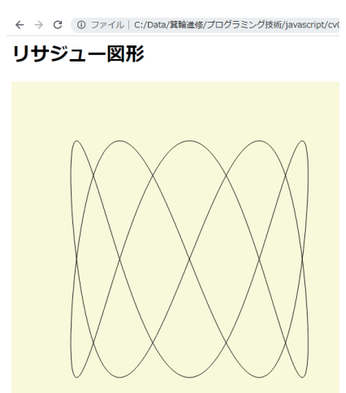


JavaScript cv03-3

実行例のように「リサジュー図形」を描くプログラムの点と点を線で結んで描くプログラムを作れ。関数 line() は使用しない。

cv03-3.html

```
・・・省略
 <body>
    <h1> リサジュー図形 </h1>
    <canvas id="cvs1" width=800 height=800 style="background-color:lightyellow;"">
    </canvas>
      <script type="text/javascript">

        var x,y,k,x1,y1,r;
        var rd=3.141592/180.0;     // RAD 変換のための定数

        x=300; y=300; r=200;       // 中心座標、半径

        var canvas = document.getElementById("cvs1");     // canvas オブジェクト取得
        var ctx = canvas.getContext("2d");                // 2D 描画コンテキストの取得

        for( k=0; k<=360; k++){            // 角度を変化
            x1 = Math.sin(k*rd*2)*r+x;     //  x座標計算
             y1 = Math.sin(k*rd*5)*r+y;    //  y座標計算
             if( k==0 )                    // もし角度が０なら
                 ctx.moveTo(x1,y1);        //   描き始めへ移動
             else                          // そうでなければ
                 ctx.lineTo(x1,y1);        //   前の点から線を引く
        }
        ctx.stroke();                      // まとめて描画
      </script>
</body>
</html></html>
```



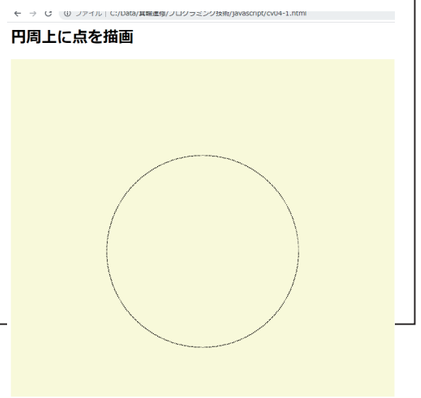
ジオメトリック・グラフィック（幾何学模様）

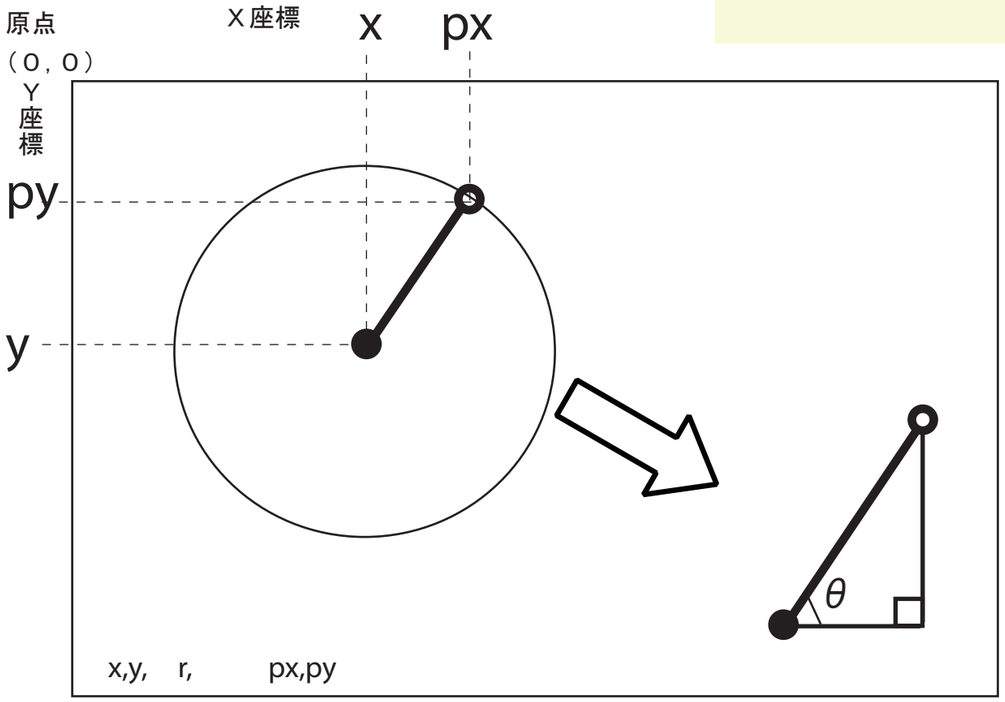
JavaScript cv04-1

中心座標と半径を与えて、円周上の点のｘｙ座標値を求め、そこに点を描画するプログラムを作れ。

cv04-1.html

```
・・・省略
 <body>
   <h1>円周上に点を描画</h1>
   <canvas id="cvs1" width=800 height=800 style="background-color:lightyellow;"">
   </canvas>
     <script type="text/javascript">

       var x,y,r,k,px,py;
       var rd=3.141592/180.0;     // RAD変換のための定数

       x=400; y=400; r=200;       // 中心座標、半径

       var canvas = document.getElementById("cvs1");     // canvasオブジェクト取得
       var ctx = canvas.getContext("2d");                // 2D描画コンテキストの取得

       for( k=0; k<=360; k++){          // 角度を０～360°
           px = Math.cos(k*rd)*r+x;     // 円周上のＸ座標
           py = Math.sin(k*rd)*r+y;     //   〃     Ｙ座標
           line(px,py,px,py+1);         // 点を描画
       }

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



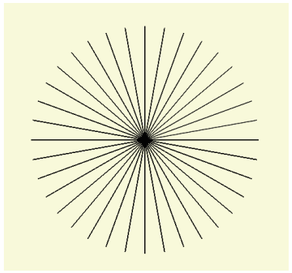


JavaScript cv04-2

中心座標と半径を与えて、円周上の点のｘｙ座標値を求め、そこに中心から線を引くプログラムを作れ。

cv04-2.html

```
・・・省略
 <body>
   <h1>円周上へ線を引く</h1>
   <canvas id="cvs1" width=800 height=800 style="background-color:lightyellow;"">
   </canvas>
     <script type="text/javascript">
       var x,y,r,k,px,py;
       var rd=3.141592/180.0;     // RAD変換のための定数

       x=400; y=400; r=200;       // 中心座標、半径

       var canvas = document.getElementById("cvs1");     // canvasオブジェクト取得
       var ctx = canvas.getContext("2d");                // 2D描画コンテキストの取得

       for( k=0; k<=360; k+=10){        // 角度 2～360°  10°おき
           px = Math.cos(k*rd)*r+x;     // 円周上のＸ座標
           py = Math.sin(k*rd)*r+y;     // 円周上のＹ座標
           line(x,y,px,py);             // 中心から円周上へ線を引く
       }
     </script>
</body>
</html>
```

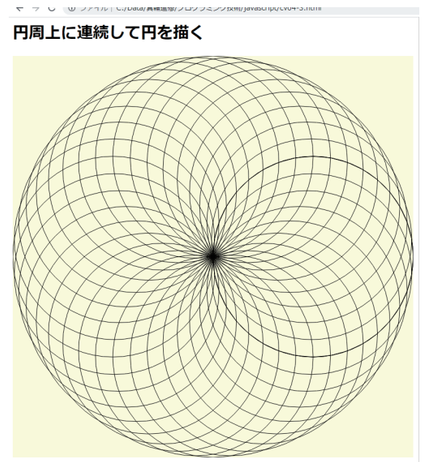


JavaScript cv04-3

中心座標と半径を与えて、円周上の点のｘｙ座標値を求め、そこに中心から線を引くプログラムを作れ。

cv04-3.html

```
・・・省略
 <body>
   <h1>円周上に連続して円を描く</h1>
   <canvas id="cvs1" width=800 height=800 style="background-color:lightyellow;"">
   </canvas>
     <script type="text/javascript">

       var x,y,r,k,px,py;
       var rd=3.141592/180.0;     // RAD変換のための定数

       x=400; y=400; r=200;       // 中心座標、半径

       var canvas = document.getElementById("cvs1");
       var ctx = canvas.getContext("2d");

       for( k=0; k<=360; k+=10){        // 角度 2～360°
           px = Math.cos(k*rd)*r+x;     // 円周上のＸ座標
           py = Math.sin(k*rd)*r+y;     // 円周上のＹ座標
           ctx.arc(px,py,r,0,Math.PI*2,true);
       }              // 円周上の点ををを中心として円を描く
       ctx.stroke();
     </script>
</body>
</html>
```

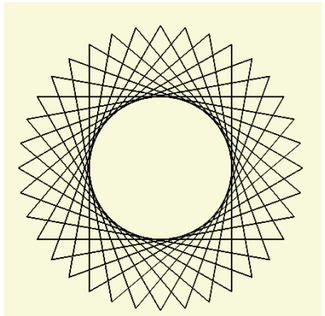


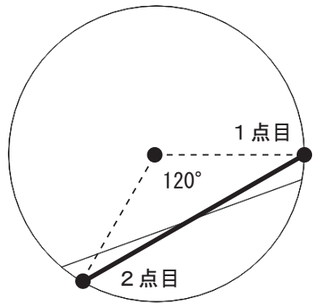
JavaScript cv04-4

円周上にある２点の間に引いた直線をぐるりと一回りさせるような、実行例の様な図形を描くプログラムを作りなさい。実行例では２点間の角度は１２０度としている。

cv04-4.html

```
・・・省略
 <body>
    <h1>直線の回転</h1>
    <canvas id="cvs1" width=800 height=800 style="background-color:lightyellow;"">
    </canvas>
      <script type="text/javascript">

        var x,y,r,k,x1,y1,x2,y2;
        var rd=3.141592/180.0;     // RAD変換のための定数

        x=400; y=400; r=200;       // 中心座標、半径

        var canvas = document.getElementById("cvs1");
        var ctx = canvas.getContext("2d");

        for( k=0; k<=360; k+=10){             // 角度 2 ～ 360°
            x1 = Math.cos(k*rd)*r+x;          //  １点目のＸ座標
            y1 = Math.sin(k*rd)*r+y;          //    〃     Ｙ座標
            x2 = Math.cos((k+120)*rd)*r+x;    //  ２点目のＸ座標
            y2 = Math.sin((k+120)*rd)*r+y;    //    〃     Ｙ座標
            line(x1,y1,x2,y2);    //  １点目から２点目へ線を引く
        }
        ctx.stroke();
      </script>
</body>
</html>
```



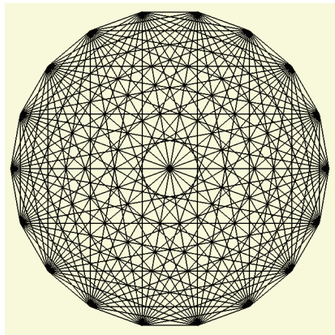


JavaScript cv04-5

実行例のような「ダイアモンドリング」と呼ばれる図形を描きなさい。

cv04-5.html

```
・・・省略
 <body>
    <h1>ダイアモンドリング</h1>
    <canvas id="cvs1" width=800 height=800 style="background-color:lightyellow;"">
    </canvas>
      <script type="text/javascript">
        var x,y,r,k,k2,x1,y1,x2,y2;
        var rd=3.141592/180.0;     // RAD変換のための定数
        x=400; y=400; r=300;       // 中心座標、半径
        var canvas = document.getElementById("cvs1");     // canvasオブジェクト取得
        var ctx = canvas.getContext("2d");                // 2D描画コンテキストの取得
        for( k=0; k<=360; k+=20){           // 角度ｋ 2 ～ 360°
            x1 = Math.cos(k*rd)*r+x;        //  １点目のＸ座標
            y1 = Math.sin(k*rd)*r+y;        //    〃     Ｙ座標
            for( k2=0; k2<=k; k2+=20){      // 角度ｋ 2 ～ 360°
                x2 = Math.cos(k2*rd)*r+x;   //  ２点目のＸ座標
                y2 = Math.sin(k2*rd)*r+y;   //  ２点目のＹ座標
                line(x1,y1,x2,y2); //  １点目から２点目に線を引く
            }
        }
      </script>
</body>
</html>
```



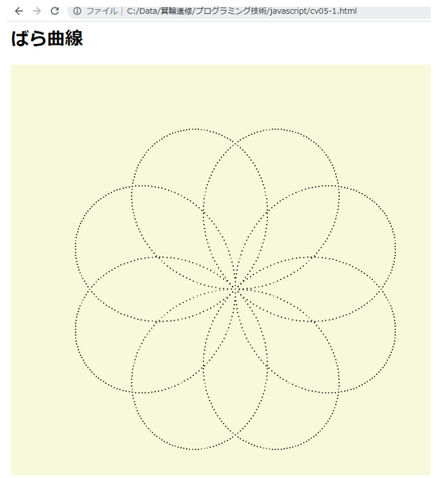
JavaScript cv05-1

円の半径をそのつど変化させて描画する「ばら曲線」という図形を描くプログラムを作る。
求める半径の式は

ｒ＝ａ×ｓｉｎ（ｂ×Θ）　ａは全体の半径、ｂは任意の値（例では４／３）

cv05-1.html

```
・・・省略
 <body>
    <h1>ばら曲線</h1>
    <canvas id="cvs1" width=800 height=800 style="background-color:lightyellow;"">
    </canvas>
      <script type="text/javascript">
        var x,y,r,px,py,a,b,k;
        var rd=3.141592/180.0;    // RAD変換のための定数
        x=400; y=400;             // 中止座標
        a=300.0; b=4.0/3.0;       // 元の半径、任意の数値
        var canvas = document.getElementById("cvs1");
        var ctx = canvas.getContext("2d");

        for( k=0; k<=1080; k++){        // 角度を変化
            r = a*Math.sin(b*k*rd);     // 描く半径を
            px = Math.cos(k*rd)*r+x;    // Ｘ座標
            py = Math.sin(k*rd)*r+y;    // Ｙ座標
            line(px,py,px,py+1);        // 点を描画
        }
      </script>
</body>
</html>
```

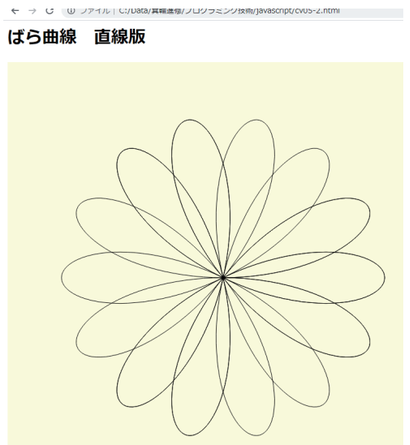


JavaScript cv05-2

「ばら曲線」の各点を直線で結ぶプログラムを作る。

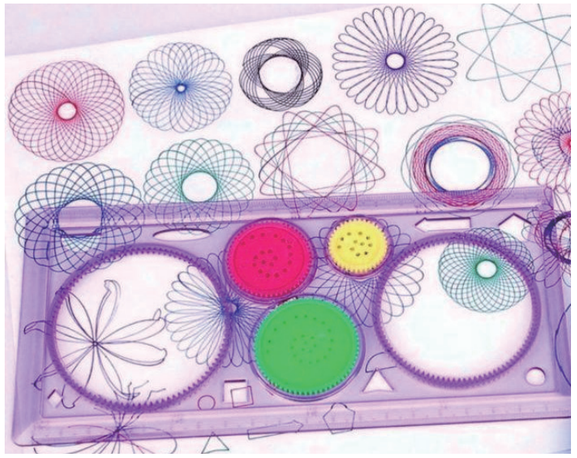
cv05-2.html

```
・・・省略
 <body>
    <h1>ばら曲線　直線版</h1>
    <canvas id="cvs1" width=800 height=800 style="background-color:lightyellow;"">
    </canvas>
      <script type="text/javascript">
        var x,y,r,px,py,a,b,k;
        var rd=3.141592/180.0;    // RAD変換のための定数
        x=400; y=400;             // 中止座標
        a=300.0; b=7.0/2.0;       // 元の半径、任意の数値
        var canvas = document.getElementById("cvs1");
        var ctx = canvas.getContext("2d");
        for( k=0; k<=1080; k++){        // 角度を変化
            r = a*Math.sin(b*k*rd);     // 描く半径を求める
            px = Math.cos(k*rd)*r+x;    // Ｘ座標
            py = Math.sin(k*rd)*r+y;    // Ｙ座標
            if(k==0)                    // もし最初なら
                ctx.moveTo(px,py);      //   ポイントへ移動
            else                        // そうでなければ
                ctx.lineTo(px,py);      //   線を引く
        }
        ctx.stroke();                   // まとめて描画
      </script>
</body>
</html>
```

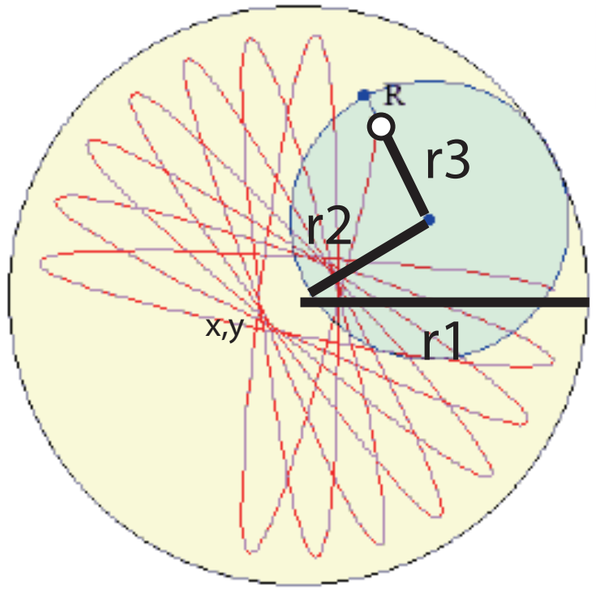


スピログラフ

歯車のついた円盤にペンを差し込んでぐるぐる回すと・・・あぁら不思議！きれいな模様が描けちゃう。あの昔懐かしいおもちゃ『スピログラフ』をプログラムで再現してみましょう。



この図形は『定円の内側を転がる円の内点（定点）の軌跡』で、数学的には『内トコロイド曲線』というのだそうです。



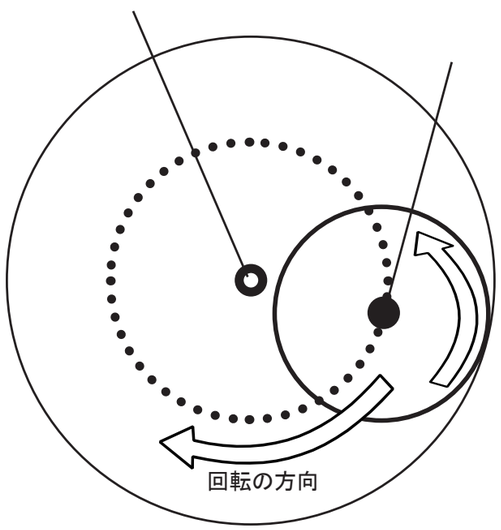
プログラム中での変数名の意味

- ○外側の円の半径　…　ｒ１
- ○内側の円の半径　…　ｒ２
- ○穴の位置
  （内側の円の中心からの半径）…　ｒ３
- ○外側の円を回転する回数　…　ｎ

まず第１段階として、回転する内側の円の中心座標の動きをプロットしてみる。

外側の円の中心座標

回転する円の中心



次にその点を中心として計算される、回転する円内の任意の点を求める。
このとき必要なのは中心座標（px,py）と半径（ｒ３）と角度であるが、角度は次のように考える。

外側の円に沿って、内側の円を１周させると、内側の円は何回転するだろうか？

外側の円の円周　＝　２　×　ｒ１　×　π
内側の　　〃　　＝　２　×　ｒ２　×　π

従って　　ｒ１／ｒ２　回転

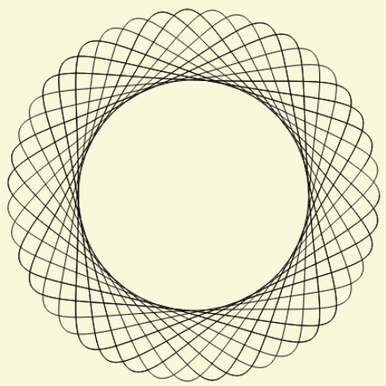
ということは１周３６０度する間に内側の円は３６０×ｒ１／ｒ２　度変化する。
つまり外側を１度動く毎に、内側の円はｒ１／ｒ２ 度　変化するということである。
この数値を「ｓ」としてプログラムにあらわすと・・・

JavaScript cv05-3

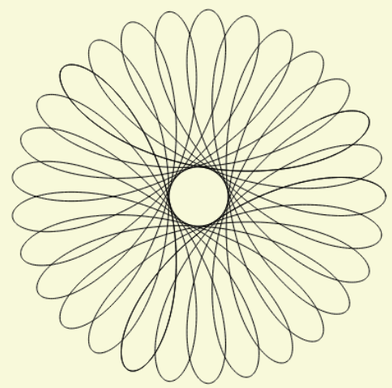
「スピログラフ」（内トコロイド曲線）を描くプログラムを作りなさい。
それぞれのパラメータを変えて、何回か実行してみよう！

cv05-3.html

```
<html>
<head>
    <title> スピログラフ 直線版 </title>
</head>
 <body>
    <h1> スピログラフ </h1>
    <canvas id="cvs1" width=800 height=800 style="background-color:lightyellow;"">
    </canvas>
      <script type="text/javascript">

        var x,y,px,py,xx,yy,r1,r2,r3,n,k,s;
        var rd=3.141592/180.0;        // RAD 変換のための定数

        x=400; y=400;                 // 中心座標
        r1=200; r2=90; r3=80; n=10;   // それぞれの半径、回転数
        s=r1/r2;                      // 内側の円の変化量

        var canvas = document.getElementById("cvs1");     // canvas オブジェクト取得
        var ctx = canvas.getContext("2d");                // 2D 描画コンテキストの取得

        for( k=0; k<=360*n; k++){              // 角度を変化
            px = Math.cos(k*rd)*(r1-r2)+x;     // 回転する円の中心座標Ｘ
             py = Math.sin(k*rd)*(r1-r2)+y;    //          〃          Ｙ
             xx = px+Math.cos(k*s*rd)*r3;      // プロットするＸ座標
             yy = py-Math.sin(k*s*rd)*r3;      //     〃      Ｙ座標
             if(k==0)                          // もし最初なら
                    ctx.moveTo(xx,yy)          //   描き始めへ移動
             else                              // そうでなければ
                  ctx.lineTo(xx,yy);           //   直線を引く
        }
        ctx.stroke();                          // まとめて描画
      </script>
</body>
</html>
```



r1=300; r2=130; r3=40; n=20;

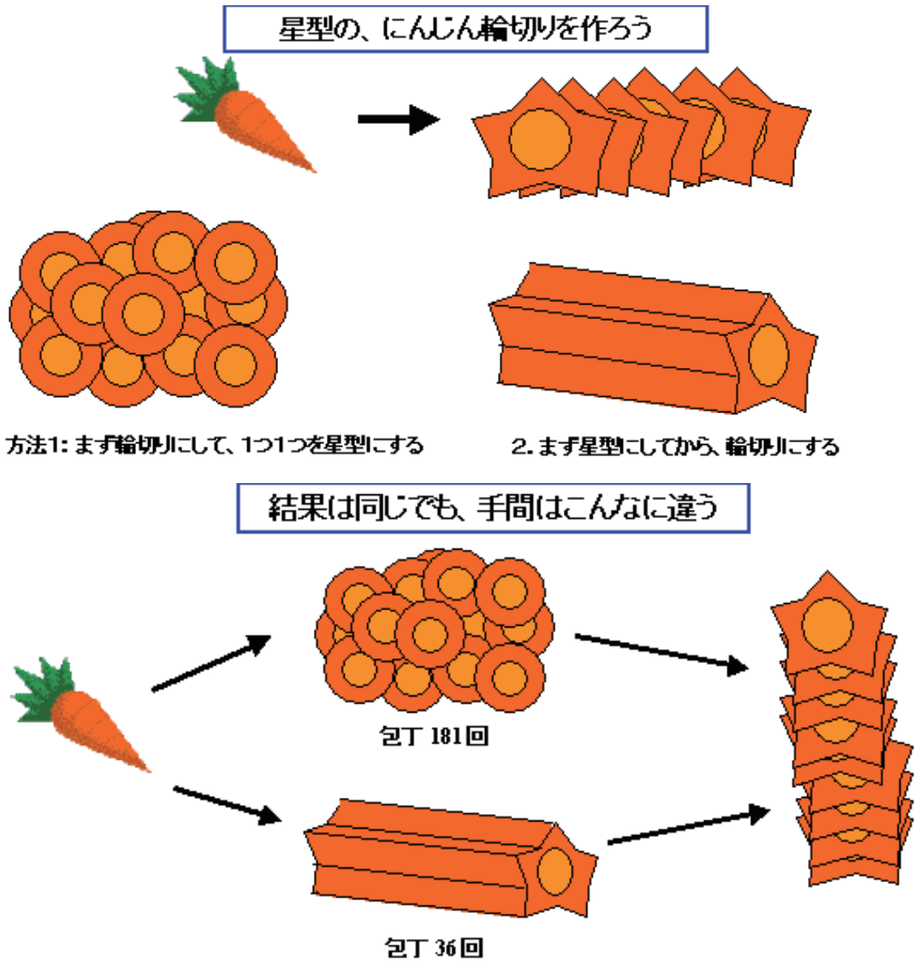


r1=200; r2=90; r3=80; n=10;

JavaScriptによるアルゴリズム入門1

私たちは、日々さまざまな問題を解決しながら生活をしている。これらの問題を解決するためには「処理手順」が必要になる。この処理手順がアルゴリズムである。

アルゴリズムの例



アルゴリズムは日本語では「算法」と訳されている。つまり問題を解決するための手順のことを意味している。ＪＩＳではアルゴリズムを『問題を解くためのものであって、明確に定義され、順序付けられた有限個の規則からなる集合』と定義している。

アルゴリズムの例題

１００個のテニスボールがある。１個の重さは１００ｇだが、１つだけ９０ｇのものが含まれている。この９０ｇのボールを見つける手順を考えよ。

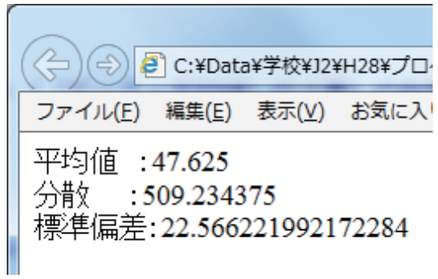
また、重さを量るのに１回で１０秒かかるとすれば、その方法では最大でどれくらいの時間がかかるのかも計算せよ？（時間が短い ⇒ 効率の良いアルゴリズム）

数値処理のアルゴリズム

JavaScript a1-01 平均・分散・標準偏差

jsa1-1.html

```
<html>
<head>
<meta charset="UTF-8">
<script type="text/javascript">
function heikin(data)          // 平均
{
    var sum = 0;
    for (i=0; i<data.length; i++) {
      sum = s[          ]
    }
    return [          ]
}

function bunsan(data)  // 分散＝（(データー平均値）の２乗）の総和 ÷ 個数
{
    var hei = heikin(data);    // 平均

    var bs = 0;
    for ([                    ]
        bs = bs + [                    ]
    }
    return [          ]
}

function h_hensa(data)    // 標準偏差
{
    var bs = bunsan(data);        // 分散
    return [          ] // 標準偏差
}
</script>
</head>
<body>
<script type="text/javascript">
    var d = [34,56,23,87,55,12,66,48];
    document.write("平均値   :" + heikin(d) + "<br>");
    document.write("分散     :" + bunsan(d) + "<br>");
    document.write("標準偏差 :" + h_hensa(d)+ "<p>" );
</script>
</body>
```



Mathオブジェクト

数学的な計算を行うには、Mathオブジェクトを使います。

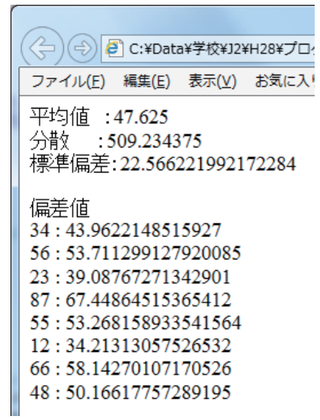
Mathオブジェクトの主なプロパティとメソッドには、次のようなものがあります。

メソッド・プロパティ	働き	使い方	計算結果
PI	円周率	a = math.PI;	約3.141592653....
abs( )	絶対値	a = Math.abs(−100);	100
ceil( )	切り上げ	a = Math.ceil(100.4);	101
floor( )	切り捨て	a= Math.floor(100.4);	100
max( )	最大値を返す	a = Math.max(1,8,3);	8
min( )	最小値を返す	a = Math.min(1,8,3);	1
pow( )	べき乗	a = Math.pow(4,2);	16
random( )	乱数を発生させる	Math.random( );	0から0.999..までの乱数
round( )	小数点以下を丸める	a = Math.round(100.4);	100
sqrt( )	平方根	a = Math.sqrt(25);	5

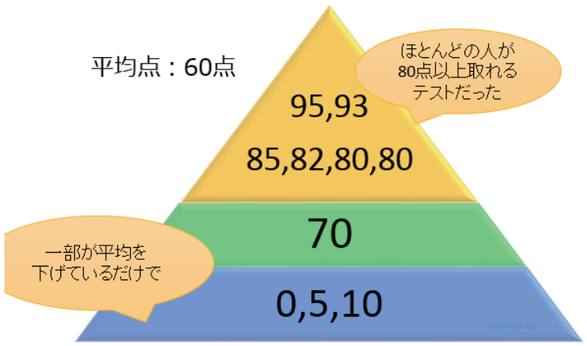
JavaScript a1-02 偏差値

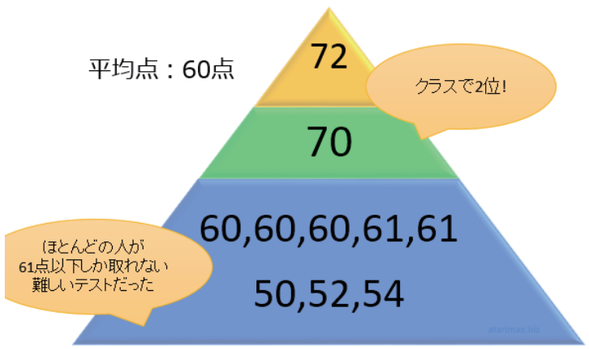
jsa1-2.html

```
<html>
  ・・・
 省略
  ・・・
function hensa( ary , dat ){
    var hei = heikin(ary);
    var h_hen = h_hensa(ary);
    var hen = [                    ]
    return hen;
}

</script>
</head>
<body>
<script type="text/javascript">
    var d = [34,56,23,87,55,12,66,48];
    document.write("平均値   :" + heikin(d) + "<br>");
    document.write("分散     :" + bunsan(d) + "<br>");
    document.write("標準偏差 :" + h_hensa(d)+ "<p>" );
    document.write("偏差値<br>");
    for(var n=0; n<d.length; n++){
        document.write(d[n] + " : " + hensa( d , d[n] ) + "<br>");
    }
</script>
</body>
</html>
```



偏差値とは…





「平均点が 60 点のテストで 70 点取ったよ！」と言ったら、それがどのくらいスゴイ事なのか分かりますか？おそらく、多くの方が「平均を超えているならそこそこ凄いんだろうなー」といった感想を持つはずです。

しかし、もしそのテストの点数分布が「0 点、5 点、10 点、70 点、80 点、80 点、82 点、85 点、93 点、95 点」（平均点 60 点）だとしたらどうでしょう？「ごく一部の生徒が平均を下げただけで、普通に勉強したら 80 点以上取れるテストだったんだな」と思いますよね。
このようなテストでの 70 点はむしろ勉強不足。少なくともスゴイ事とは言えません。

では逆に、もしそのテストの点数分布が「50 点、52 点、54 点、60 点、60 点、60 点、61 点、61 点、70 点、72 点」（平均点 60 点）だとしたらどうでしょう？
クラスで 2 位の成績ですし、点数分布から「多くの生徒が間違えた超難問のうちの 1 つを正解した」と推測できます。
これは間違いなくスゴイ事ですね！
このように、平均という数字は情報量が少なく、それだけでは意外と役に立たない数字なのです。

そこで役に立つのが「ばらつきの大きさを表す数値」である標準偏差。テストを平均点と標準偏差という 2 つの視点からみることで、「70 点を取ったこと」がどのくらいスゴイ事なのかが一気に分かりやすくなるんです。では、どうすれば「ばらつきの大きさ」を数値化できるのでしょうか？

例えば、平均点 50 点のテストで 90 点以上を取った人が何人もいたら「ばらつきの大きなテストだったんだろうな」と予想できますよね。この「各データの値と平均値の差」のことを「偏差」と言います。
たとえば平均点が 60 点なら、10 点の偏差は－ 50、80 点の偏差は＋ 20 となります

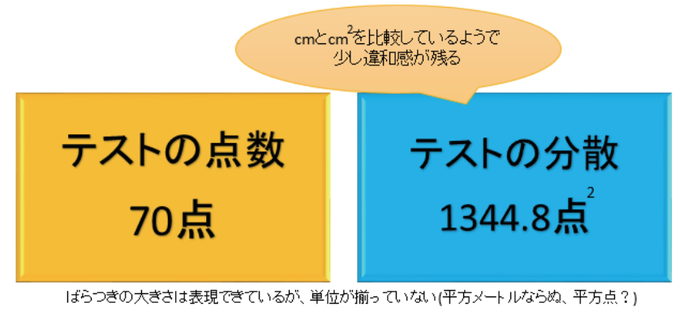
	平均 60 点なら…
（ 0 － 60）^2	
+（ 5 － 60）^2	
+（10 － 60）^2	
+（70 － 60）^2	
+（80 － 60）^2	
+（80 － 60）^2	
+（82 － 60）^2	
+（85 － 60）^2	分散
+（93 － 60）^2	↓
+（95 － 60）^2	÷ 10 = 1344.8

このように、ばらつきの大きさは「**各データの値と平均値の差がどれくらい大きいのか**」で判断できます。

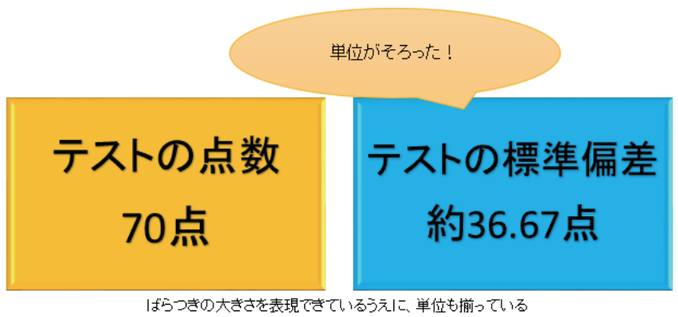
「平均との差がそこそこの値が 2 つあるよりも、平均との差がかなり大きい値が 1 つある方がばらつきが大きい」ことを表現するために、「**平均との差の 2 乗**」を利用してみましょう。

〔{（個々の値－平均）＾ 2} の総和〕/ 個数

これは **分散** と呼ばれ、標準偏差とともに「**データのばらつきの大きさ**」を表すのに利用されています。



ばらつきの大きさは表現できているが、単位が揃っていない(平方メートルならぬ、平方点？)



ばらつきの大きさを表現できているうえに、単位も揃っている

分散は、ばらつきの大きさを表すのに便利な数値ではあるのですが、「2 乗したせいで元のデータの数値と単位がそろわない」という欠点もあります。

そこで、分散の平方根（=√）を利用して、元のデータの数値と単位をそろえてみましょう。　この分散の平方根に当たる値が、**標準偏差**です。前ページの例だと…

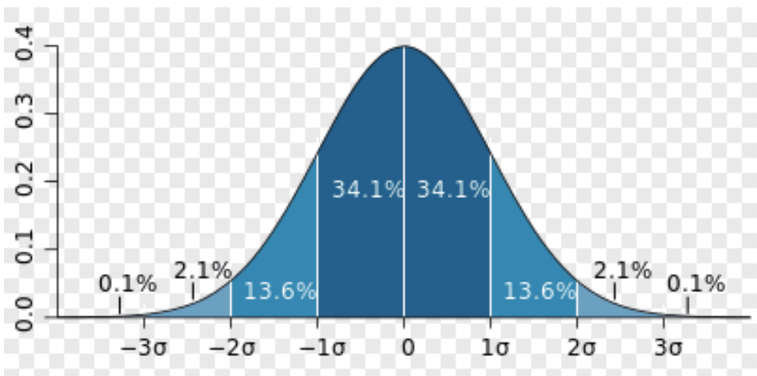
√ 1344.8 ＝約 36.67 点

標準偏差　＝ √分散

偏差値は、平均点 50 点・標準偏差が 10 点になるように調整した時のあなたのテストの点数を表しています。

偏差値 ＝｛10 ×（個々の値－平均）÷標準偏差｝＋ 50

「平均 60 点のテストで 70 点取ったよ！」と言われてもどのくらいスゴイのかは分かりませんが、「偏差値 60 取ったよ！」ならスゴさが分かります！



データの確率分布が正規分布と呼ばれる図のような形をしていた場合

「平均－標準偏差」～「平均＋標準偏差」内にあるデータが含まれる確率が約 68%
「平均－ 2 ×標準偏差」～「平均＋ 2 ×標準偏差」内に、あるデータが含まれる確率が約 95%
ということが分かっています。

あるテストの点数分布が正規分布に近似できて、平均点 50 点・標準偏差 10 点だったのなら、

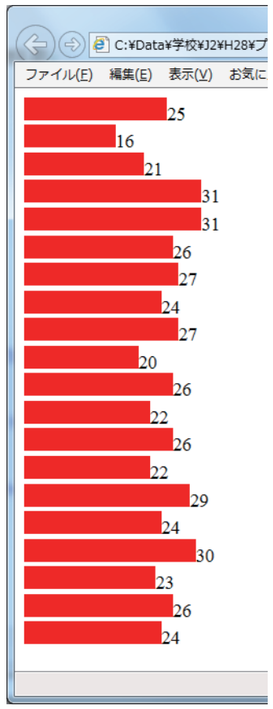
40 点から 60 点の間に受験者の約 68% が存在して、
30 点から 70 点の間に受験者の約 95% が存在しているということです。
逆に言えば、40 点以下に約 16% 存在し、60 点以上にも約 16% 存在するということでもあります。
成績が正規分布であると仮定すると，理論的には偏差値がわかれば順位を計算することができます。

数値処理のアルゴリズム

JavaScript a2-01 一様乱数（疑似乱数）

jsa2-1.html

```
<html>
<head>
<meta charset="UTF-8">
<script type="text/javascript">

    function ransu( max )   // 0 → max-1 までの一様乱数
    {
        return [          ]
    }


</script>
</head>
<body>
<script type="text/javascript">
    var data = [];
    var i;
    var cnt = [];

    for(i=0;i<=20;i++)
         cnt[i]=0;

    for(i=0;i<500;i++)
        data[i] = ransu(20)+1;       //  1 ～ 20 までの乱数を配列に格納

    for(i=0;i<data.length;i++){
        //document.write(data[i] + " ");
        cnt[ data[i] ]++;            // 乱数をカウント
    }

    for(i=1;i<=20;i++){
        document.write("<img src='img/red.png' width=" + cnt[i]*5 + " height=20>" + cnt[i] + "<br>");
    }
</script>
</body>
</html>
```



リロードして何回か実行してみること

JavaScript a2-02 正規乱数（ボックス＝ミューラー法）

今まで使用していた乱数は特定の範囲の乱数がばらばらに発生する「一様乱数」と呼ばれるものです。これに対してたとえばテストの得点などある範囲に偏りがある乱数が必要になる場合があります。このような乱数を発生させるアルゴリズムを考えます。

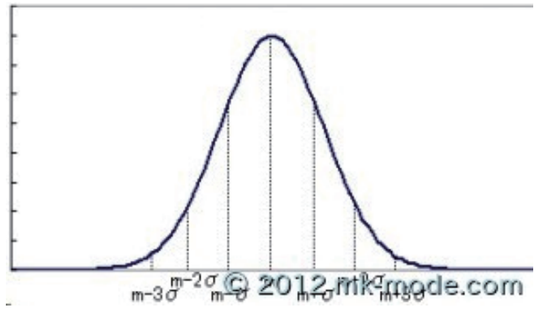
「正規乱数」とは「正規分布」を持つ「乱数」のことです。

「正規分布」は右のようなグラフになり、

以下のような式で定義されます。

$$N(m,\sigma^2)=\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{(x-m)^2}{2\sigma^2}}$$

m：平均　　σ：標準偏差



ボックス＝ミューラー法 (Box-Muller transform)

$$x=\sigma\sqrt{-2\log r_1}\cos 2\pi r_2+m$$
$$y=\sigma\sqrt{-2\log r_1}\sin 2\pi r_2+m$$

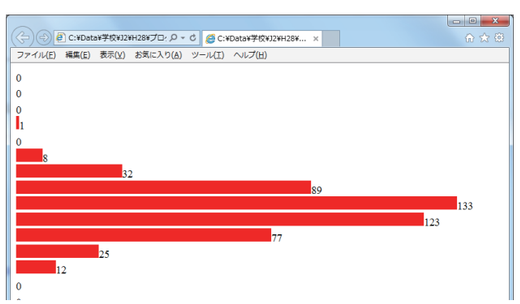
ｒ１、ｒ２：一様乱数（0.0 ～ 1.0）

(0.0 ～ 1.0）の２個１組の一様乱数 (r1, r2) で２個の正規乱数 (x, y) を生成できます。

この「正規乱数」の生成方法として、今回は「ボックス＝ミューラー法 (Box-Muller transform」を使用します。「ボックス＝ミューラー法」は、0 より大きく 1 以下、すなわち (0,1] の一様乱数を正規乱数（正規分布を持つ乱数）に変換する方法で、計算式は左のよ

jsa2-2.html

```
・・・
<script type="text/javascript">

    function ransu(max)          // 0 → max-1 までの一様乱数
    {
        return  Math.floor(Math.random()*max);
    }

    function seiki_ransu( mu , sigma ){  // 正規乱数  mu: 平均  sigma: 標準偏差
        var r1 = Math.random();
        var r2 = Math.random();
        var z  = Math.sqrt(-2 * Math.log(r1) ) * Math.cos(2*Math.PI*r2);
        return  Math.floor(mu + sigma*z);       //sin,cos どちらかひとつで良い
    }


</script>
・・・
    for(i=0;i<500;i++)
        data[i] = seiki_ransu(10 , 2);
・・・          // 平均が 10 標準偏差 2 の正規乱数
```

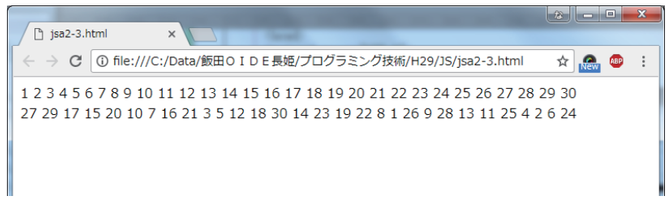


JavaScript a2-03 ランダムな順列

ビンゴゲームのように決められた範囲の数値をばらばらに出力するプログラムを考えてみる。同じ数値が２度出現しないようにする工夫が必要になる。

jsa2-3.html

```
<html>
<head>
<meta charset="UTF-8">
<script type="text/javascript">

    function ransu(max)       // 0 → max-1 までの一様乱数
    {
        return  Math.floor(Math.random()*max);
    }

    function swap(ary , a , b) // データ交換関数
    {
        [          ]


    }

    function ary_disp(ary)  // 配列内容表示
    {
        for(i=1;i<ary.length;i++){
            document.write(ary[i] + " ");
        }
        document.write("<br>");
    }
</script>
</head>
<body>
<script type="text/javascript">
    var data = [];
    var MAX=30;
      // 配列に順列をセット
    for(i=0;i<=MAX;i++)
         data[i] = i;

    ary_disp(data);

      // ランダムな順列を生成
    for([          ]
        [          ]
        swap( data , i , r );
    }
    ary_disp(data);
</script>
</body>
</html>
```

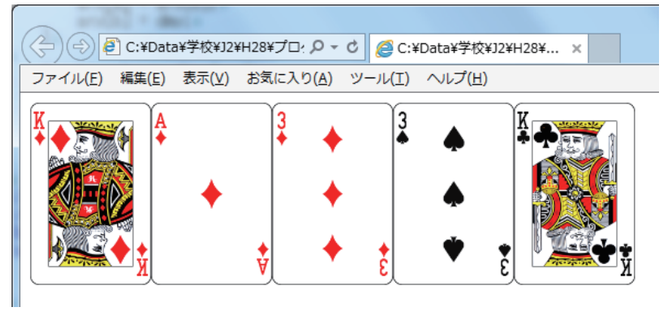


JavaScript a2-04 トランプをシャッフルして５枚取り出す

トランプ 53 枚（ジョーカー含む）の画像がある。(img/trump/1.png ～ 53.png)
これをシャッフルして最初の５枚を表示するスクリプトを作る。

jsa2-4.html

```
<html>
<head>
<meta charset="UTF-8">
<script type="text/javascript">
    function ransu(max)       // 0 → max-1 までの一様乱数
    {
        return  Math.floor(Math.random()*max);
    }

    function swap(ary , a , b)
    {
        [          ]


    }

    function trump_disp(ary)
    {
        for(i=1;i<=5;i++){
            document.write([                    ]
        }
    }

</script>
</head>
<body>
<script type="text/javascript">
    var data = [];
    var MAX=53;

    for(i=0;i<=MAX;i++)     // 配列に順列をセット
         data[i] = i;

    for(i=MAX;i>0;i--){     // トランプをシャッフル
        var r = ransu(i)+1;
        swap( data , i , r );
    }

    trump_disp(data);

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



トランプの画像データ

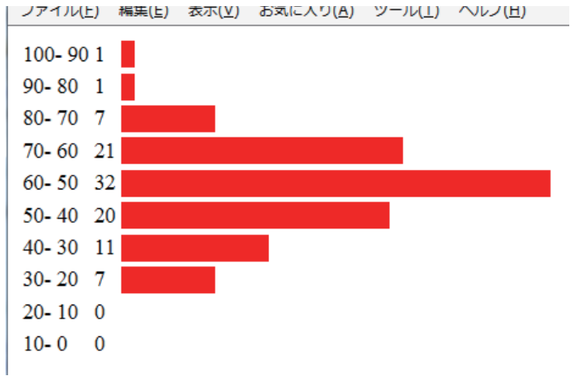
数値処理のアルゴリズム

JavaScript a3-01 度数分布

正規乱数（平均 60 点、標準偏差 10 点）によって 100 人分のテストの得点を疑似発生させ、10 点刻みの度数分布グラフを表示するスクリプトを作成せよ。

jsa3-1.html

```
<html>
<head>
<meta charset="UTF-8">
<script type="text/javascript">
    function seiki_ransu( mu , sigma ){  // 正規乱数  mu: 平均  sigma: 標準偏差
        var r1 = Math.random();
        var r2 = Math.random();
        var z  = Math.sqrt(-2 * Math.log(r1) ) * Math.cos(2*Math.PI*r2);
        return  Math.floor(mu + sigma*z);
    }
</script>
</head>

<body>

  <script type="text/javascript">
    var data = [];
    var i,j;
    var dosuu = [];
    for(i=0;i<=10;i++)
         dosuu[i]=0;
    for(i=0;i<100;i++)
        data[i] = seiki_ransu(60,15);     // 平均 60 標準偏差 15 の正規乱数
    for(i=0;i<data.length;i++){
        //document.write(data[i] + " ");
        dosuu[                                  ]++;
    }
    document.write("<table>");
    for(i=10;i>0;i--){
        document.write("<tr><td>" + i*10 + "- " + (i-1)*10 + "</td><td>" + dosuu[i] + "</td>");
        document.write("<td><img src='img/red.png' width=" + dosuu[i]*10 + " height=20></td></tr>");
    }
    document.write("</table>");

  </script>

</body>
</html>
```



HTML TABLE タグ

```
<table border=1>
  <tr>
    <td> 1 1 1 1</td>
    <td> 2 2 2 2</td>
    <td> 3 3 3 3 3 3</td>
  </tr>
  <tr>
    <td> 4 4</td>
    <td> 5 5 5 5 5</td>
    <td> 6 6 6 6</td>
  </tr>
</table>
```

1111	2222	333333
44	55555	6666

<table> …表の始まりであることを示す

<tr> …行の始まりであることを示す

<td> ～ </td> …セルであることを示す

</tr> …行の終わりであることを示す

</table> …表の終わりであることを示す

JavaScript a3-02 順位付け 1

配列に格納されているデータに大きい順に順位を付ける。

jsa3-2.html

```
<html>
<head>
<meta charset="UTF-8">
<script type="text/javascript">
</script>
</head>
<body>
<script type="text/javascript">
    var d = [15,8,10,13,18,9,6,20,3,16];
    var jyuni;
     for(var i=0;i<d.length;i++){
        jyuni=          
        for(var n=0;n<d.length;n++){
 ①……    if(                                    )
        }
        document.write(d[i] + " : " + jyuni + " 位<br>");
    }
</script>
</body>
</html>
```

```
15 : 4 位
8 : 8 位
10 : 6 位
13 : 5 位
18 : 2 位
9 : 7 位
6 : 9 位
20 : 1 位
3 : 10 位
16 : 3 位
```

このアルゴリズムで①の行が実行される回数は＿＿＿＿＿＿回となる。

データの個数をｎ個とすると　繰り返し回数＝＿＿＿＿＿＿＿＿となる。

```
15   4 位
 8   8 位
10   6 位
13   5 位
18   2 位
 9   7 位
 6   9 位
20   1 位
 3  10 位
16   3 位
```

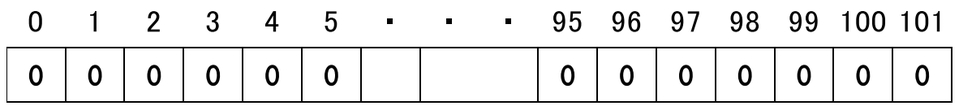
JavaScript a3-03 TABLE タグと CSS で見やすくレイアウト →

順位づけの改良

データが増える繰り返し回数が増大…

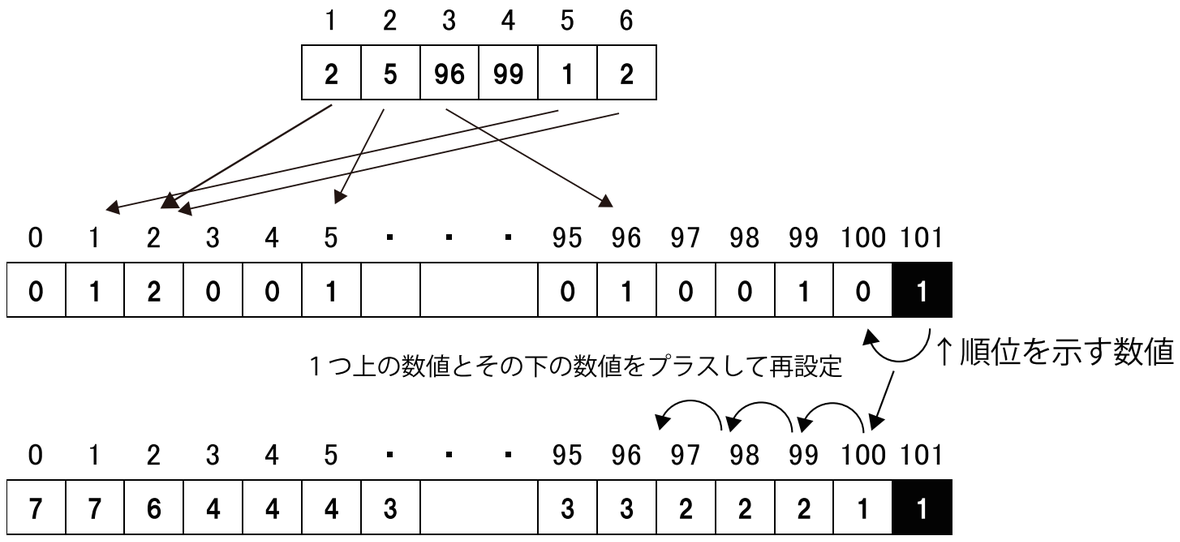
繰り返し回数を減らすアルゴリズムを考える・・・

例えばテストの得点のようにデータの範囲が限定されている（０点～１００点）場合０～１００と余分な配列１０１を用意し、内容をゼロクリアしておく。



↑１つ余分な配列

次のようなデータがあった場合、各データの
対応する配列のデータをカウントする。



以上の処理で「該当の得点」の順位が「得点＋１」の配列に求められる。
例えば９９点は配列の [100] にある「１」が順位となる。

このアルゴリズムで①②の行が実行される回数の合計は＿＿＿＿＿＿回となる。データの個数をｎ個、データの範囲をｍとすると
繰り返し回数＝＿＿＿＿＿＿＿＿となる。

JavaScript a3-04

順位付け２

配列に格納されているデータに大きい順に順位を付ける。

jsa3-4.html

```
<html>
<head>
<meta charset="UTF-8">
<script type="text/javascript">
</script>

<style type="text/css">
    td {
        padding:3px;
        text-align:right;
    }
</style>
</head>

<body>

<script type="text/javascript">

    var d = [15,8,10,13,18,9,6,20,3,16];  // データ範囲 0- 20
    var jyuni=[],i;
    for(i=0;i<=21;i++)
        [                                                    ]

    for(i=0;i<d.length;i++)
①…… [                                                    ]

    jyuni[21] = 1;

    for( i=20; i>=0; i-- )
②…… jyuni[i] = [                                          ]

    document.write("<table border=1><tr>");
    for(i=0;i<=21;i++)
        document.write("<td>" + i + "</td>");
    document.write("</tr><tr>");
    for(i=0;i<=21;i++)
        document.write("<td>" + jyuni[i] + "</td>");
    document.write("</tr></table>");

    document.write("<table>");
    for(i=0;i<d.length;i++)
        document.write("<tr><td>" +  d[i] + "</td><td>" + jyuni[ d[i]+1 ] + " 位</td></tr>");
    document.write("</table>");

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

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
11	11	11	11	10	10	10	9	9	8	7	6	6	6	5	5	4	3	3	2	2	1

15 4位
8 8位
10 6位
13 5位
18 2位
9 7位
6 9位
20 1位
3 10位
16 3位

数値処理のアルゴリズム

JavaScript a4-01 最大公約数１

例えばコンピュータを使って最大公約数を求めるとき、前述のような「しらみつぶし」のアルゴリズムを使えば以下のようなプログラムも考えられる。

jsa4-1.html

```
<html>
<head>
<meta charset="utf-8">
<script type="text/javascript">
     function bclick(){
          var t1 = document.getElementById('text1');
          var t2 = document.getElementById('text2');
          var m  = document.getElementById('msg');
          var n1 = Number(t1.value);
          var n2 = Number(t2.value);
          var gcd;
          for(var i=1;i<n1/2;i++){     // n 1 >n2 として
              if(                              )
                   gcd = i;
          }
          m.innerHTML = "最大公約数＝ " + gcd;
     }
</script>
</head>
 <body>
     数値１<input type="text" id="text1"><br>
     数値２<input type="text" id="text2">
     <input type="button" value="最大公約数" onClick="bclick()">
     <p id="msg">最大公約数</p>
</body>
</html>
```

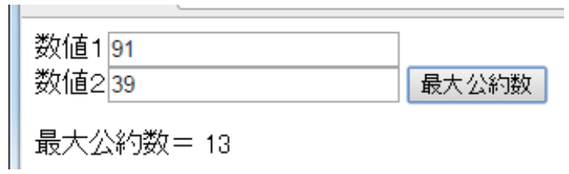
JavaScript a4-02 最大公約数２　ユークリッドの互除法

ユークリッドの互除法（Wikipedia より）

ユークリッドの互除法（ユークリッドのごじょほう）は、2 つの自然数または整式の最大公約数を求める手法の一つである。2 つの自然数（または整式）a, b (a ≧ b) について、a の b による剰余を r とすると、a と b との最大公約数は b と r との最大公約数に等しいという性質が成り立つ。この性質を利用して、b を r で割った剰余、除数 r をその剰余で割った剰余、と剰余を求める計算を逐次繰り返すと、剰余が 0 になった時の除数が a と b との最大公約数となる。明示的に記述された最古のアルゴリズムとしても知られ、紀元前 300 年頃に記されたユークリッドの『原論』第 7 巻、命題 1 から 3 がそれである。

ユークリッドの互除法の具体的なアルゴリズム

0）入力を x, y とする。(ただし　x ≧ y)
1）y = 0 なら、x を出力してアルゴリズムを終了する。
2）x を y で割った余りを新たに y とし、更に 元の y を新たに x とし (1.) に戻る。




```
<html>
<head>
     <meta charset="utf-8">
     <script type="text/javascript">
     function bclick(){
               var t1 = document.getElementById('text1');
               var t2 = document.getElementById('text2');
               var m  = document.getElementById('msg');
          var x = Number(t1.value);
          var y = Number(t2.value);
          var r;
          while(                    ){     // n 1 >n2 として
               r =
               x =
               y =
          }
               m.innerHTML = "最大公約数＝ " +
     }
     </script>
</head>
 <body>
     数値１<input type="text" id="text1"><br>
     数値２<input type="text" id="text2">
     <input type="button" value="最大公約数" onClick="bclick()">
     <p id="msg">最大公約数</p>
</body>
</html>
```

jsa4-2.html

JavaScript a4-03

素数１

素数とは、１と自分自身以外に約数をもたない整数のことで　２，３，５，７，１１、・・・などがある。

１～１０００まで整数の素数をすべて求めるプログラムを作る場合の簡単なアルゴリズムは・・・

１）調べる整数をｎとする。ｎの初期値は２。

２）ｎがｎ／２から２まで の数で割り切れるかどうかを、順次判定していく。（ループ）

３）もし割り切れる数があった場合は素数ではないので、ループを抜ける。

４）もし２までループしても割り切れる数がなかったら素数なので表示する。

４）ｎの値を１プラスして再び判定する。これを１０００まで繰り返す

jsa4-3.html

```
<html>
<head>
     <meta charset="utf-8">
     <script type="text/javascript">
     </script>
</head>
 <body>
     <script type="text/javascript">
        var n,i,flag;
        for( n=2; n<=1000; n++ ){
            for( i = Math.floor(n/2); i >= 2 ; i-- ){
       ①……    if( n%i == 0 ) break;
            }
            if( i==1 ) document.write( n + " ");
        }
     </script>
</body>
</html>
```

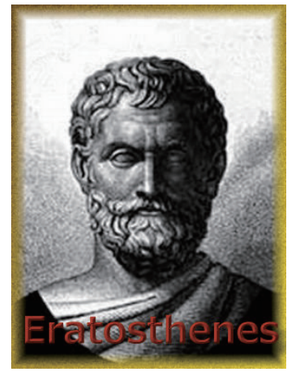
```
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101 103 107
109 113 127 131 137 139 149 151 157 163 167 173 179 181 191 193 197 199 211 223
227 229 233 239 241 251 257 263 269 271 277 281 283 293 307 311 313 317 331 337
347 349 353 359 367 373 379 383 389 397 401 409 419 421 431 433 439 443 449 457
461 463 467 479 487 491 499 503 509 521 523 541 547 557 563 569 571 577 587 593
599 601 607 613 617 619 631 641 643 647 653 659 661 673 677 683 691 701 709 719
727 733 739 743 751 757 761 769 773 787 797 809 811 821 823 827 829 839 853 857
859 863 877 881 883 887 907 911 919 929 937 941 947 953 967 971 977 983 991 997
```

このアルゴリズムで①の行が実行される回数は最大で　＿＿＿＿＿回となる。

JavaScript a4-04

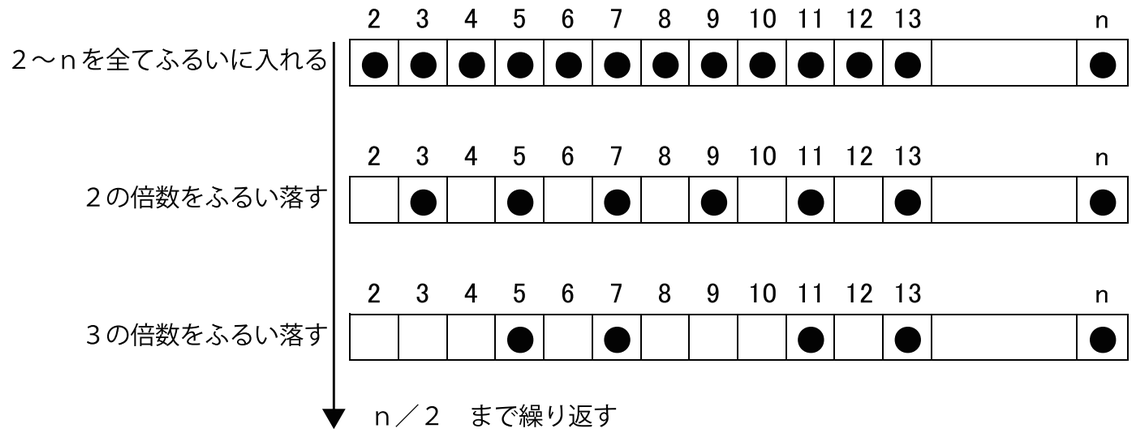
素数２　エラトステネスの篩（ふるい）

jsa4-4.html




```
<html>
<head>
     <meta charset="utf-8">
     <script type="text/javascript">
     </script>
</head>
 <body>
     <script type="text/javascript">
        var n,i,f=[];
        for(i=0;i<=1000;i++)
            f[i] = 1;        // ふるい全てに１を入れる

        for( n=2; n<=1000/2; n++ ){
            if( f[n] == 1 ){
                for(                                    ){
   ①……             f[i] = 0;              // ふるいから外す
                }
            }
        }

        for( n=2; n<=1000; n++ ){
            if( f[n] == 1 )                 // ふるいに残っていたら表示
                document.write( n + " ");
        }
     </script>
</body>
</html>
```

２～ｎまでの素数を求める場合…



このアルゴリズムで①の行が実行される回数は最大で＿＿＿＿＿回となり、最後に結果を表示するために配列を走査する回数＿＿＿＿＿回を加えると＿＿＿＿＿回となる。

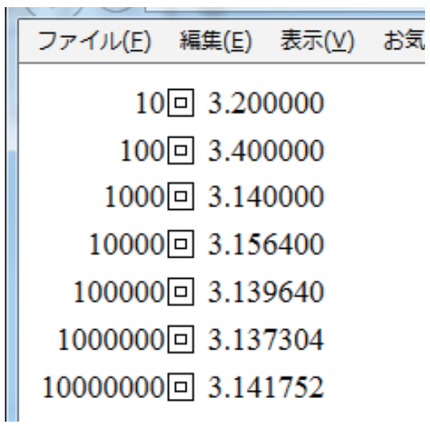
数値処理のアルゴリズム

JavaScript a4-05 モンテカルロ法　モンテカルロ法を用いて円周率（π）を求める

jsa4-5.html

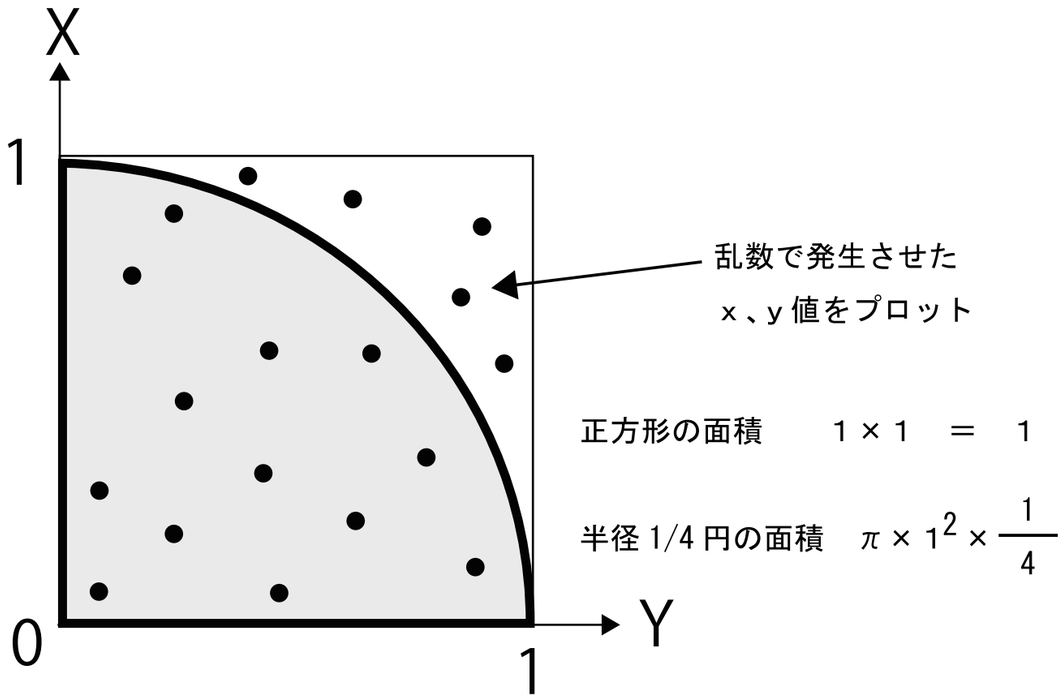
```
<html>
<head>
     <meta charset="utf-8">
     <script type="text/javascript">
     </script>
</head>
 <body>
     <script type="text/javascript">
        var sum,x,y,n,i;

        for(n=10;n<=10000000;n*=10){          // 10 回～ 1 千万回まで検証
            sum = 0;
            for( i=0;i<n;i++){
                x = Math.random();
                y = Math.random();
                if ( [                    ] ) sum++;
            }
            document.write( n + "回：" + (4*sum/n).toFixed(6) + "<br>");
        }                                     // 小数点以下 6 桁の文字列を生成
     </script>
</body>
</html>
```

TABLE タグとＣＳＳを用いて表示を揃えること



モンテカルロ法（Monte Carlo method, MC）

シミュレーションや数値計算を乱数を用いて行う手法の総称。ランダム法とも呼ばれ、その手法がギャンブルに似ていることからカジノで有名な国家モナコ公国の４つの地区の１つであるモンテカルロから名付けられた。



ばらまくまかれた乱数の総数　…　ｎ
1/4 円内にばらまかれた乱数　…　ａ
円外にばらまかれた乱数　…　ｂ

乱数の比率　＝　面積の比率

$$a : a + b = \frac{\pi}{4} : 1$$

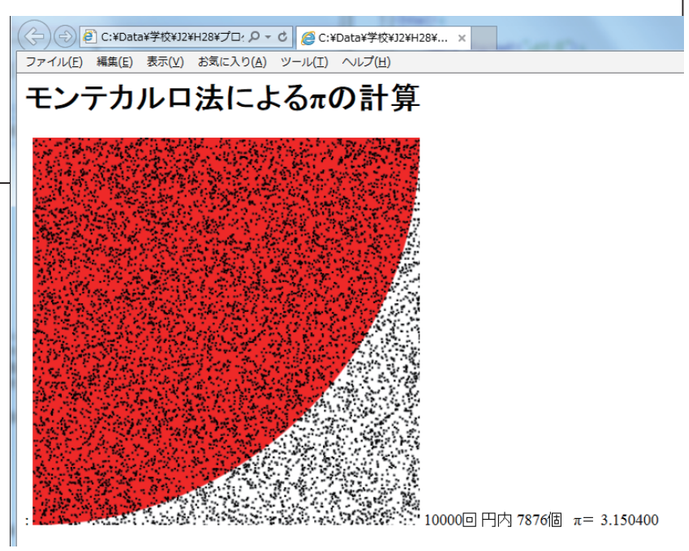
$$\pi = \frac{4a}{a+b} = \frac{4a}{n}$$

０～１の一様乱数を２つ発生させ、それらをｘ、ｙとして該当する座標にプロットする。この点が均一にばらまかれていると仮定すると、その面積比はばらまかれた乱数の個数と比例するはずである。
多くのプロットをする方がより正確な値に近づくと考えられる。

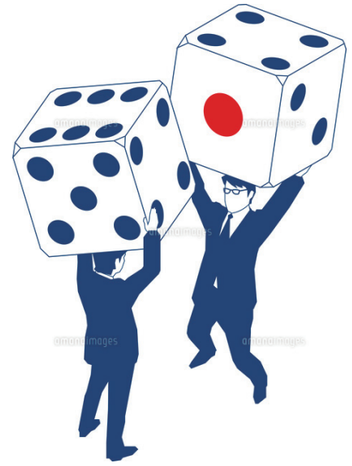
JavaScript a4-06 モンテカルロ法２ CANVAS を使用して実際に描画する

jsa4-6.html

```
<html>
<head>
     <meta charset="utf-8">
</head>
 <body>
    <h1> モンテカルロ法によるπの計算 </h1> :
    <canvas id="cvs1" width=400 height=400 >
    </canvas>
     <script type="text/javascript">
        var sum,x,y,n,i;
        var canvas = document.getElementById("cvs1");    // canvas オブジェクト取得
        var ctx = canvas.getContext("2d");               // 2D 描画コンテキストの取得
        ctx.fillStyle = 'rgb(255,00,00)';                // 塗りつぶしの色は赤
        ctx.arc(0,0,400,0,Math.PI*2,true);               // 半径 400px の円
        ctx.fill();                                      // 塗りつぶす
        ctx.fillStyle = 'rgb(0,00,00)';                  //塗りつぶしの色は黒

        sum = 0;
        n = 10000;                          // 乱数発生回数　変えてみる
        for( i=0;i<n;i++){
            x = Math.random();
            y = Math.random();
            ctx.fillRect(x*400,  y*400,  2,2);           // 2px の点を描画
            if (  [          ]  ) sum++;
        }
        document.write( n + "回　円内 " + sum + "個　π＝ " + (4*sum/n));
     </script>
</body>
</html>
```



JavaScript a4-07 モンテカルロ法３



あるパーティの余興として次のようなものを考えた。
このゲームにかかる時間をシミュレートするプログラムを考えよ。

１）安価な同じ賞品を多数用意する。
２）参加者がサイコロを２つ振る。
３）ゾロ目が出たら、その数だけ賞品をもらえる。
４）用意した賞品が無くなるまで続ける。

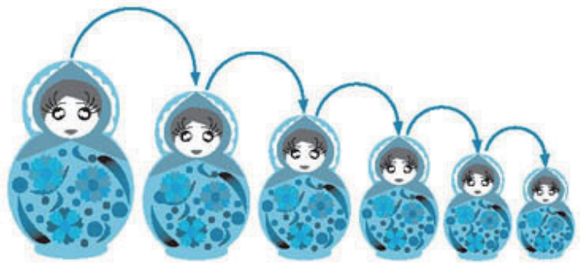
用意する賞品の個数、１人にかかる時間を設定してシミュレーションせよ。

jsa4-7.html

```
<html>
<head>
    <title> モンテカルロ法 </title>
    <script type="text/javascript">
    function rnd(m,n)
    {
        return  Math.floor(Math.random()*(n-m+1))+m;    /* n ～ m 乱数 */
    }
  </script>
</head>
<body>
      <script type="text/javascript">
        var i,n,p,r1,r2,t1,t,m,s;

        for(i=1;i<=10;i++){   // １０回
          n=0;              // 参加者
          p=100;            // 賞品の個数
          t1=20;            // １回にかかる時間（秒）
          t=0;              // 総時間
          while( p>0 ){     // 賞品が残っている間
            r1 = rnd(1,6);     // １つ目のサイコロ
            r2 = rnd(1,6);     // ２つ目のサイコロ
            if [          ] // ゾロ目なら賞品を減らす
            t=t+t1;                          // 総時間
            n++;                             // 参加者カウント
          }
          m = Math.floor(t/60);       // 分
          s = t%60;                   // 秒
          document.write("参加者 :" + n + " 人　所要時間 : " + m + " 分 " + s + " 秒<br>");
        }
      </script>
</body>
</html>
```

```
参加者 :179 人　所要時間 : 59 分 40 秒
参加者 :180 人　所要時間 : 60 分 0 秒
参加者 :208 人　所要時間 : 69 分 20 秒
参加者 :139 人　所要時間 : 46 分 20 秒
参加者 :198 人　所要時間 : 66 分 0 秒
参加者 :203 人　所要時間 : 67 分 40 秒
参加者 :118 人　所要時間 : 39 分 20 秒
参加者 :173 人　所要時間 : 57 分 40 秒
参加者 :239 人　所要時間 : 79 分 40 秒
参加者 :167 人　所要時間 : 55 分 40 秒
```

関数のマジック　再帰（リカーシブルコール）

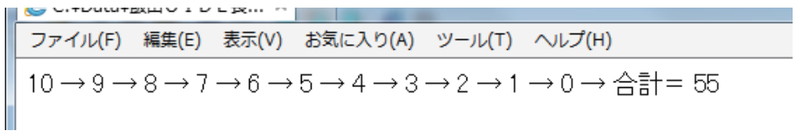


再帰呼び出しとは、プログラミングにおける技術の一つで、プログラムのある関数の中から自分自身の関数を呼び出すことである。
再帰呼び出しは、ある計算において、計算結果に対して同じ処理を連続的に行う場合に使うことが多い。また、ソースコードでは、ループ処理を使って記述するよりもきれいに書くことができる。
ただし、再帰を行うごとにスタックフレームを占有していくことから、パフォーマンスはあまり期待できないと言われている。また、再帰から抜ける条件が成り立たずに、無限ループに陥ることも少なくないため、扱いには注意が必要とされる。

JavaScript saiki01　１～ｘまでの合計を求める

saiki01.html

```
<html>
<head>
  <script type="text/javascript">
    function kei( n )
    {
        document.write(n + " → ");
        if(n==0)
            return 0;
        else
            return [          ]
    }
  </script>
</head>

<body>
  <script type="text/javascript">
    document.write( "合計＝ " + kei(10) );
  </script>
</body>
</html>
```



JavaScript saiki02　ユークリッドの互除法（最大公約数）

以前、学習した、２つの正数値の最大公約数を求めるアルゴリズム「ユークリッドの互除法」も再帰を使ってプログラミングする例題として示されることが多い。

ユークリッドの互除法（Wikipedia より）

ユークリッドの互除法（ユークリッドのごじょほう）は、2 つの自然数または整式の最大公約数を求める手法の一つである。2 つの自然数（または整式）a, b (a ≧ b) について、a の b による剰余を r とすると、a と b との最大公約数は b と r との最大公約数に等しいという性質が成り立つ。この性質を利用して、b を r で割った剰余、除数 r をその剰余で割った剰余、と剰余を求める計算を逐次繰り返すと、剰余が 0 になった時の除数が a と b との最大公約数となる。明示的に記述された最古のアルゴリズムとしても知られ、紀元前 300 年頃に記されたユークリッドの『原論』第 7 巻、命題 1 から 3 がそれである。

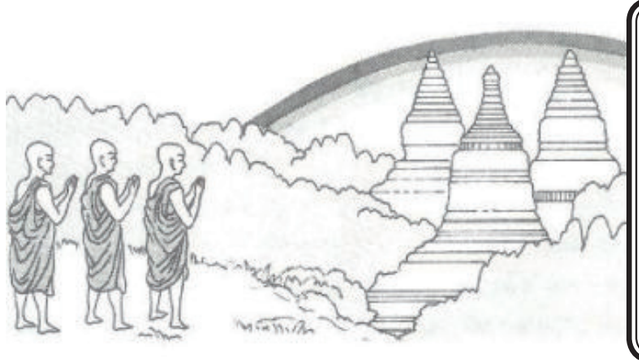
ユークリッドの互除法の具体的なアルゴリズム

0）入力を x, y とする。(ただし　x ≧ y)
1）y = 0 なら、x を出力してアルゴリズムを終了する。
2）x を y で割った余りを新たに y とし、更に 元の y を新たに x とし (1.) に戻る。

saiki02.html

```
<html>
<head>
  <script type="text/javascript">
    function gcd( x, y )
    {
        if(y==0)
            return x;
        else
            return [          ]
    }
  </script>
</head>

<body>
  <script type="text/javascript">
    document.write( gcd(252,105) );
  </script>
</body>
</html>
```

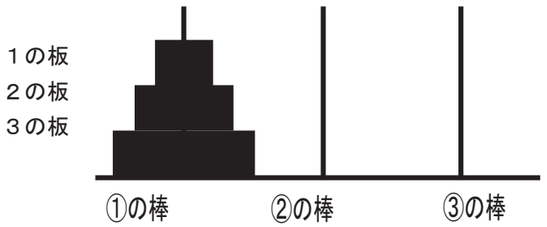
ハノイの塔



インドのベナレスにある大寺院に、ダイヤモンドの針が3本立った板があり、神はその一本に64枚の純金の円盤をはめた。昼夜の別なく、バラモン僧たちはそこにやってきて、それを別の針に移し替える作業に専念している。そして移し替えが完了したとき、寺院もバラモン僧たちも崩壊し、この世が終わるのである。

『ハノイの塔』は１８８３年にフランスのパズル研究家E．リュカが考えたゲームです。
台の上に３本の棒が固定されており、そのうちの一本に何枚かの円盤がはまっています。円盤は下へいくほど半径が大きくなっています。話しを簡単にするために、一番左の棒をA、真ん中の棒をB、一番右の棒をCとし、最初にAに何枚かの円盤がはまっているとしましょう。棒Bを利用して全ての円盤をAからCに移してください。
ハノイの塔のルールは次の通りです。

- 一回に一枚の円盤しか動かしてはいけない。
- 移動の途中で円盤の大小を逆に積んではいけない。
  常に大きい方の円盤が下になるようにする。
- 棒以外のところに円盤を置いてはいけない。



JavaScript saiki03 ハノイの塔

```
<html>
<head>
  <script type="text/javascript">
    function hanoi( n , a , b , c)
    {
          if( n>0 ){
            hanoi( n-1 , a , c , b );
            document.write(n + "の板 " + a + " ⇒ " + b + "<br>");
            hanoi( n-1 , c , b , a );
        }
     }

  </script>
</head>

<body>
    <script type="text/javascript">
        hanoi(3,1,2,3);
    </script>
</body>
</html>
```

```
1の板 1 ⇒ 2
2の板 1 ⇒ 3
1の板 2 ⇒ 3
3の板 1 ⇒ 2
1の板 3 ⇒ 1
2の板 3 ⇒ 2
1の板 1 ⇒ 2
```

saiki03.html

JavaScript saiki04 ハノイの塔　簡易ビジュアル版

saiki04.html

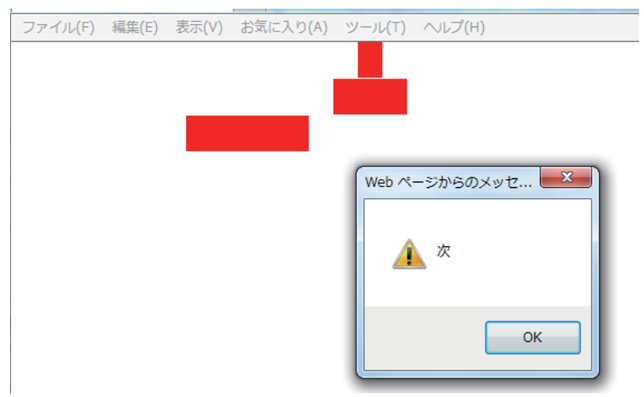
```
<html>
<head>
  <script type="text/javascript">

    var ita=[];

    function hanoi( n , a , b , c)
    {
          if( n>0 ){
            hanoi( n-1 , a , c , b );
            //document.write(n + "の板 " + a + " ⇒ " + b + "<br>");
            alert("次");resizeTo(800,600-n);
            ita[ n ].style.left = b*100-n*20;

            hanoi( n-1 , c , b , a );
        }
     }

  </script>
</head>

<body>
    <img id="1" src="img/red.png" width= 20 height=29 style="position:absolute;left:80;top: 0">
    <img id="2" src="img/red.png" width= 60 height=29 style="position:absolute;left:60;top:30">
    <img id="3" src="img/red.png" width=100 height=29 style="position:absolute;left:40;top:60">
    <script type="text/javascript">
        ita[1] = document.getElementById("1");
        ita[2] = document.getElementById("2");
        ita[3] = document.getElementById("3");
        hanoi(3,1,2,3);
    </script>
</body>
</html>
```

画面の再描画

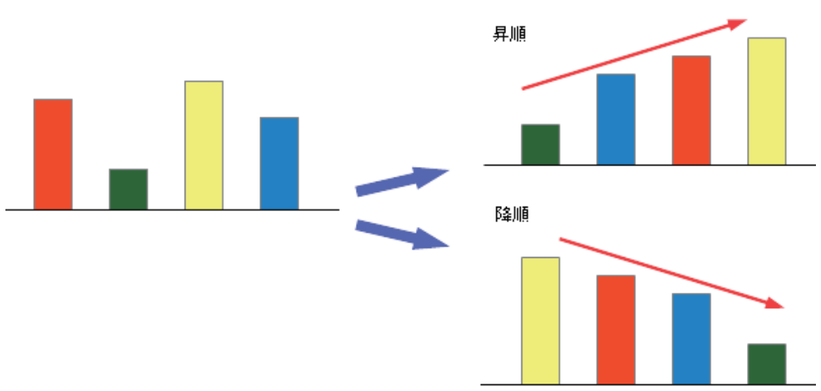
ＩＥのみ動作可能

このコードをここに追加すれば上下位置も変化

```
y=60;
switch(n){
    case 1 : if(ita[ 2 ].style.left == ( b*100-2*20 )+"px") y-=30;
    case 2 : if(ita[ 3 ].style.left == ( b*100-3*20 )+"px") y-=30;
}
ita[ n ].style.top = y;
```

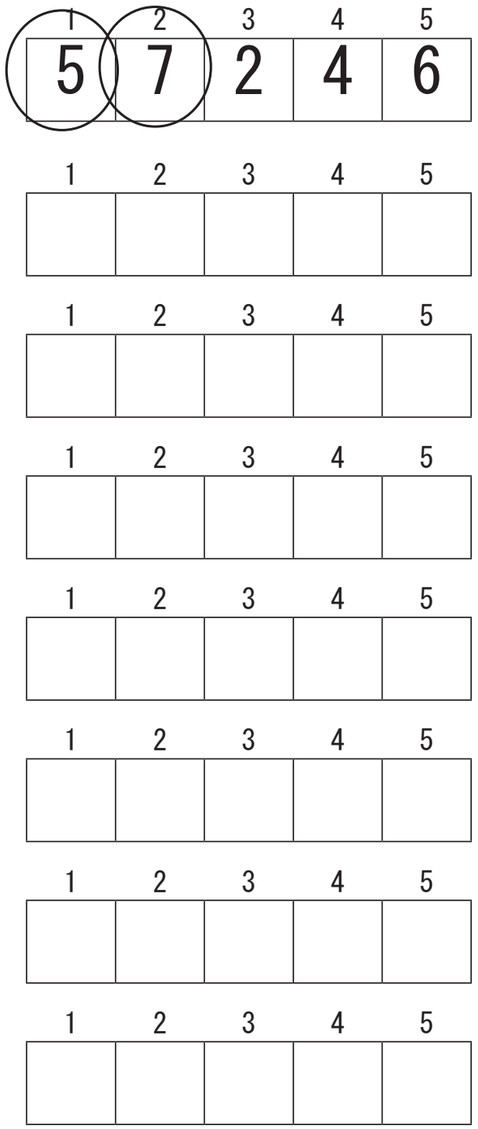


ソーティングのアルゴリズム



ソートとは、「並べ替え」の意味です。数値データを、大きい順、もしくは小さい順に並べ替える処理のことを言います。なお、データを小さい順に並べ替えることを、昇順（しょうじゅん）、大きい順に並べ替えることを、降順（こうじゅん）と言います。

交換法のアルゴリズム



まずは隣り合った「１」と「２」のデータを比べ、

もし､「１」のほうが「２」よりも大きかったら「１」と「２」の内容を入れ替える。

続いて「２」と「３」を比べ

もし､「２」のほうが「３」よりも大きかったら「２」と「３」の内容を入れ替える。

この操作を最終の「４」と「５」まで繰り返すシミュレーションをしてみよう。

「５」にはこの配列中の最大のものが入るはずだ。

続いて「１」から「４」についても同様の操作を繰り返す。「４」の場所には「１」から「４」の中で最大のものが入る。

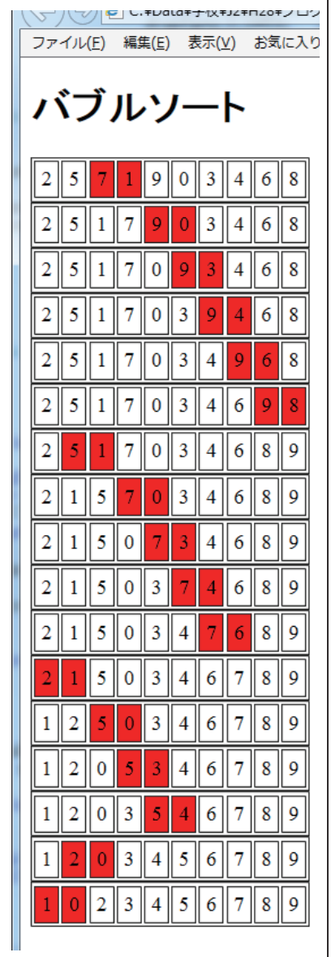
この時点で右側の２つ「４」「５」については右側から大きい順に並べ替えが完了している

このようにあたかも水中の＿＿＿のようにデータ浮かび上がっていくよう見えるため、この並び替えの方法を＿＿＿＿＿＿＿＿＿＿＿＿と呼ぶ。

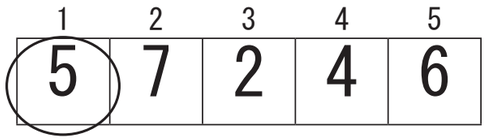
JavaScript a5-01 交換法（バブルソート）

jsa5-1.html

```
<html>
<head>
<title>Canvas Sample1</title>
<meta content="text/html; charset=utf-8" http-equiv="Content-Type">
<script type="text/javascript">
    function ransu(max)
      {
        return  Math.floor(Math.random()*max);
    }
    function swap(ary , a , b)     // 配列データ交換
    {
        var dmy = ary[a];
        ary[a] = ary[b];
        ary[b] = dmy;
    }
    function disp( ary ,a,b)  // 配列データ表示
    {
        document.write("<table border=1 cellpadding=5><tr>");
        for(var i=0;i<=MAX; i++){
            if( i==a || i==b)
                document.write("<td bgcolor=#ff0000>");  // 交換するデータは赤色
            else
                document.write("<td bgcolor=#ffffff>");
            document.write( ary[i] + "</td>");
        }
        document.write("</tr></table>");
    }
    function ary_init( ary )
    {
        var i;
        for(i=0;i<=MAX;i++)     // 配列に順列をセット
            ary[i] = i;
        for(i=MAX;i>0;i--){     // 配列をばらばらにする
            r = ransu(i);
            swap( ary , i , r );
        }
    }
    function bubble_sort( ary )   // バブルソート（交換法）
    {
          var i,j,n;
          for( i=MAX; i>0; i-- ){
                for( j=0; j<i; j++ ){
                      if(                                    )
                            disp( ary , j ,j+1 );// データ表示
                            swap( ary , j ,j+1 );// 〃    交換
                      }
                }
          }
    }
</script>
</head>
<body>
<h1>バブルソート</h1>
<script type="text/javascript">
    var MAX = 9;
    var data = [],i,r;
    ary_init( data );
    bubble_sort( data );   // バブルソート
</script>
</body>
</html>
```

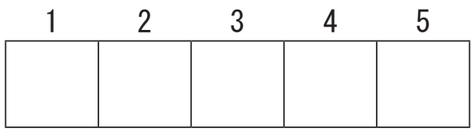


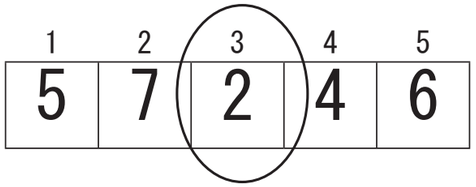
選択法のアルゴリズム



まずは「１」の場所に入るべきデータを見つける

「１」の場所には＿＿＿から＿＿＿のまでのデータのうちで１番小さいものが入るべきである。



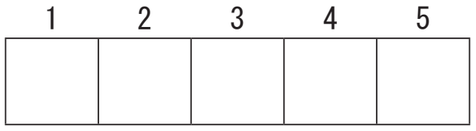


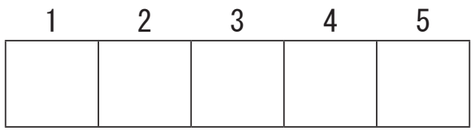
１～５までのデータで　最小値は＿＿＿＿

その番号は＿＿＿＿

そのデータと１番のデータの内容を入れ替える

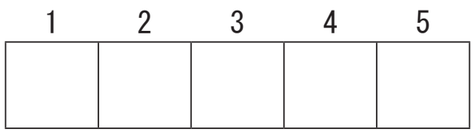
この時点で「１」の場所には１～５のデータの最小値がセットされる。

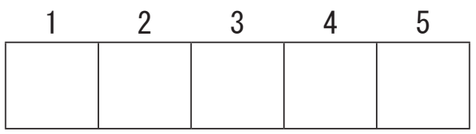




続いて「２」の場所に入るべきデータを見つける

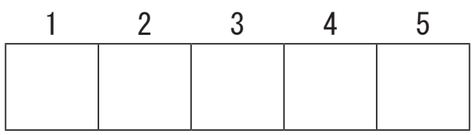
「２」の場所には＿＿＿から＿＿＿のまでのデータのうちで１番小さいものが入るべきである。

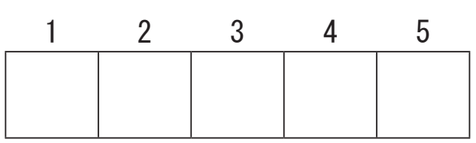




そのデータと２番のデータの内容を入れ替える

この時点で「２」の場所には２～５のデータの最小値がセットされる。



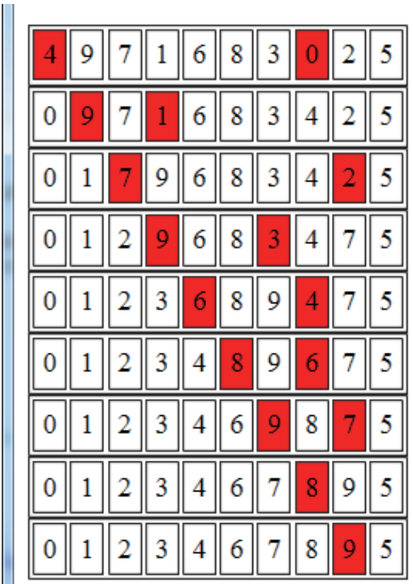


JavaScript a5-02 基本選択法（セレクションソート）

jsa5-2.html

```
<html>
<head>
<title>Canvas Sample1</title>
<meta content="text/html; charset=utf-8" http-equiv="Content-Type">
<script type="text/javascript">
     function ransu(max)
      {
         return  Math.floor(Math.random()*max);
    }
    function swap(ary , a , b)     // 配列データ交換
    {
        ・・・省略
    }
    function disp( ary ,a,b)  // 配列データ表示
    {
        ・・・省略
    }
    function ary_init( ary )
    {
        ・・・省略
    }
    function bubble_sort( ary )   // バブルソート（交換法）
    {
     ・・・  省略
    }
     function selection_sort( ary )
     {
          var i,j,n;
          for( i=0; i<MAX; i++ ){          // 入るべき位置
               n = i;
               for( j=i; j<=MAX; j++ ){    // 範囲から最小値を選択
                    [                    ]
               }
               disp( ary , i , n );   // 表示
               swap( ary , i , n  );  // 交換
          }
    }
</script>
 </head>
<body>
<h1>選択法</h1>
<script type="text/javascript">
    var MAX = 9;
    var data = [],i,r;
    ary_init( data );
   selection_sort( data );  // 選択法
</script>
</body>
```

前の課題の関数は残して追加する



挿入法のシミュレーション

「１」のデータを基準

「２」のデータを退避させ「２」の場所を空ける

続いてひとつ前のデータ「１」と退避させたデータを比較する。

退避させたデータが「１」よりも小さかったら

「１」のデータを「２」の場所へシフトする。

退避データを空いた「１」の場所へ入れる

退避データのほうが大きい場合はそのまま戻す

この操作を最終の「５」まで繰り返す

シミュレーションをしてみよう。

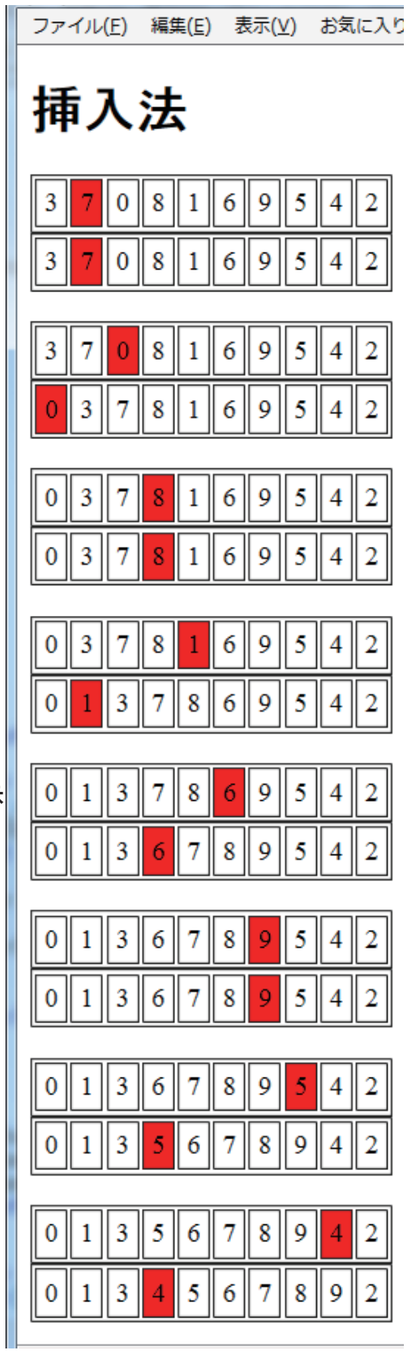
JavaScript a5-03 挿入法（インサートソート）

jsa5-3.html

```
<html>
<head>
<title>Canvas Sample1</title>
<meta content="text/html; charset=utf-8" http-equiv="Content-Type">
<script type="text/javascript">
    function ransu(max)
      {
        return  Math.floor(Math.random()*max);
    }
    function swap(ary , a , b)     // 配列データ交換
    {
        ・・・省略
    }
    function disp( ary ,a,b)  // 配列データ表示
    {
        ・・・省略
    }
    function ary_init( ary )
    {
        ・・・省略
    }
```

前の課題の関数は残して追加する

```
     function insert_sort( ary )  // 挿入法
     {
          var i,j,chk;
          for( i=1; i<=MAX; i++ ){
              disp( ary ,i );        // 退避データ表示
              chk = ary[i];
              j = i-1;
              while( chk < ary[j] ){
                  [                              ]
                  j--;
                  if(j<0) break;
              }
              [                                  ]
              disp( ary ,j+1 );      // 挿入データ表示
              document.write("<br>");
          }
      }
</script>
</head>
<body>
<h1>選択法</h1>
<script type="text/javascript">
    var MAX = 9;
    var data = [],i,r;
    ary_init( data );
    insert_sort( data );     // 挿入法
</script>
</body>
</html>
</script>
</body>
</html>
```

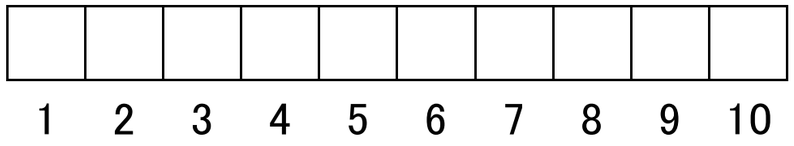


カウンティングソート（分布数え上げ法、バケットソート）

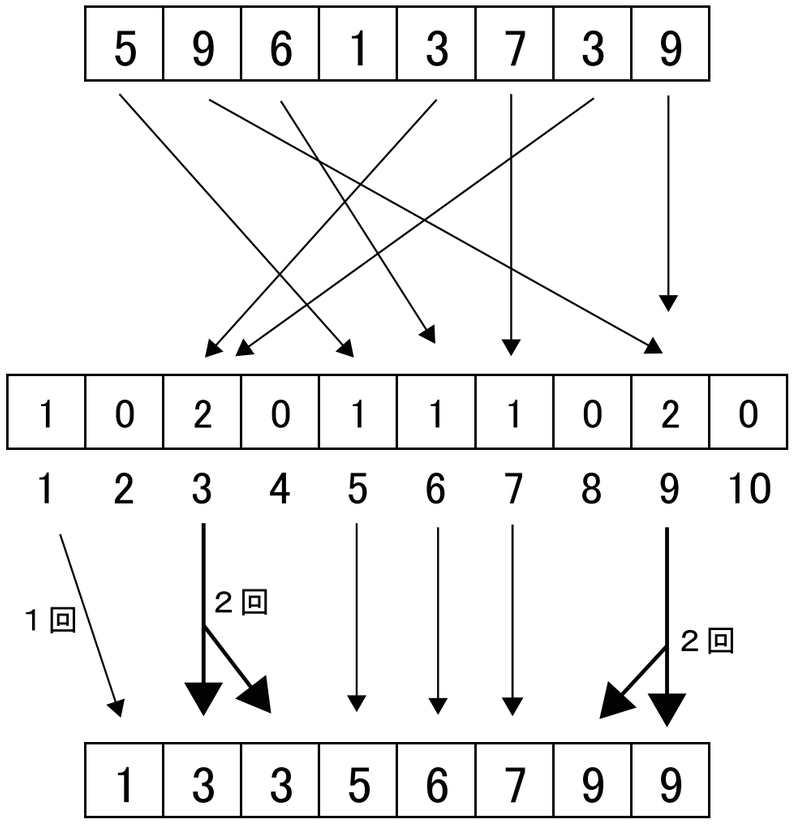
分布数え上げソートはソート対象のデータをキーにして、キーの出現回数とその累積度数分布を計算して利用することで整列を行うアルゴリズムです。
キーとなるデータがとりうる値の範囲をあらかじめ知っている必要があります。

データ範囲「１～１０」の場合

①データ範囲（1-10）に対応する配列を準備



②データの出現回数をカウント



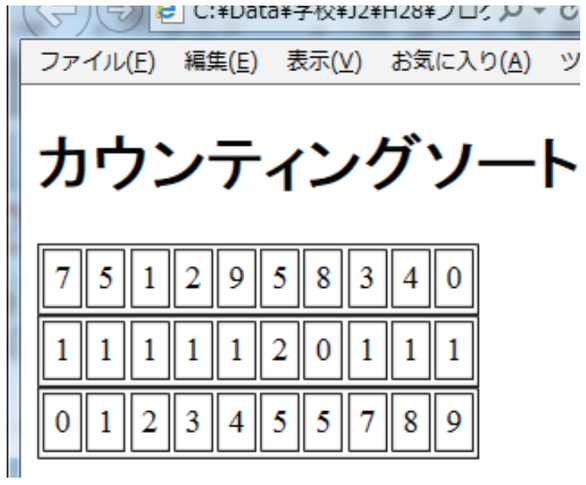
③出現回数分だけ元の配列に取り出す

JavaScript a5-04 カウンティングソート

jsa5-4.html

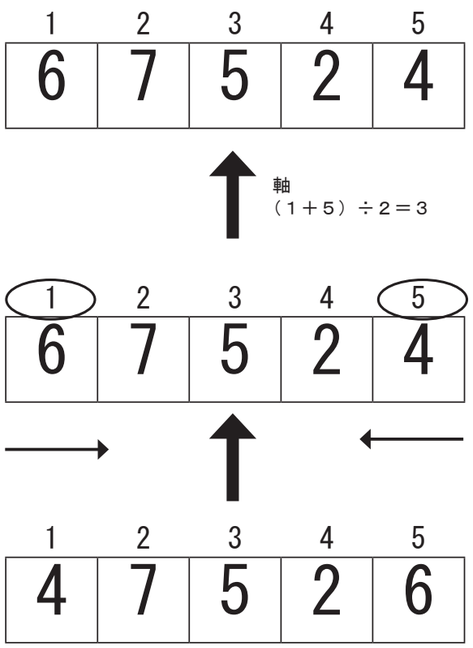
```
<html>
<head>
<title>Canvas Sample1</title>
<meta content="text/html; charset=utf-8" http-equiv="Content-Type">
<script type="text/javascript">
    function ransu(max)
      {
        return  Math.floor(Math.random()*max);
    }
    function swap(ary , a , b)    // 配列データ交換
    {
        ・・・省略
    }
    function disp( ary ,a,b)  // 配列データ表示
    {
        ・・・省略
    }
    function ary_init( ary )
    {
        ・・・省略
    }

function counting_sort( ary )
{
    var cnt = [],i,n=0;
    for(i=0;i<=MAX;i++)
        cnt[i] = 0;
    disp( ary );    // データ配列
    for(i=0;i<=MAX;i++)
        [          ]
    disp( cnt );    // カウント配列
    for(i=0;i<=MAX;i++){
        for(var  j=1;j<=cnt[i];j++){
            [          ]
            n++;
        }
    }
    disp(ary);      // ソート後
}
</script>
</head>
<body>
<h1>カウンティングソート</h1>
<script type="text/javascript">
    var MAX = 9;
    var data = [],i,r;
    ary_init( data );
    data[5]=5;                  // わざと重複データを作る
    counting_sort(data);        // カウンティングソート
</script>
</script>
</body>
</html>
```

前の課題の関数は残して追加する



ソーティングのアルゴリズム

クイックソート法のシミュレーション



軸（基準となるデータ）を決めて、それよりも大きなデータの集まりと小さなデータの集まりに分ける。これをそれぞれの集団に再帰的に実行することにより整列する方法。データの状態にもよるが、最速のソーティングアルゴリズムといわれている。

まず配列の中心位置を求めここを仮の軸とする。仮の中心の求め方は

配列の添字の最小値と最大値を加えて２で割る

続いて、軸の値を基準にして、軸の左右でデータの入れ替えを行う。

軸の左側から軸データより大きいものを探し、

軸の右側から軸データより小さいものを探す

その２つを入れ替える。これを繰り返す。

軸データ自身も交換の対象となる場合もある。

この操作が終了すると、仮の軸の左側には軸データよりも小さなデータが（グループ１）

軸の右側には大きなデータが（グループ２）

振り分けられる。

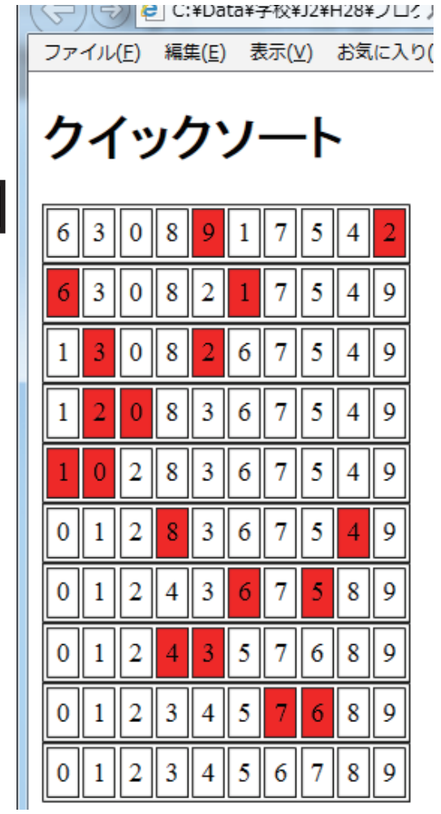
以後、それぞれのグループで再び軸を設定し、振り分けを行っていく。グループ分けできなくなった時点で整列の終了となる。

実際のプログラミングには再帰手続きを使用している。

JavaScript a5-05 クイックソート

jsa5-5.html

```
<html>
<head>
<title>Canvas Sample1</title>
<meta content="text/html; charset=utf-8" http-equiv="Content-Type">
<script type="text/javascript">
    function ransu(max)
      {
        return  Math.floor(Math.random()*max);
    }
    function swap(ary , a , b)    // 配列データ交換
    {
        ・・・省略
    }
    function disp( ary ,a,b)  // 配列データ表示
    {
        ・・・省略
    }
    function ary_init( ary )
    {
        ・・・省略  前の課題の関数は残して追加する
    }

var q_ary = [];  // クイックソート用配列
function quick_sort( low ,  high )
{
      var ct,ctv,L,R;
      L = low;  R = high;

      ct = Math.floor( (low+high)/2 );
      ctv = q_ary[ct];
      while(1){
            while( q_ary[L] < ctv ) L++;
            while( q_ary[R] > ctv ) R--;
            if( L>= R ) break;
            disp( q_ary , L , R );
            swap( q_ary , L , R );
            L++; R--;
      }
      if( (L-low) >= 2)
      if( (high-R) >= 2)
}
</script>
</head>
<body>
<h1>クイックソート</h1>
<script type="text/javascript">
    var MAX = 9;
    var data = [],i,r;
    ary_init( data );
     q_ary = data;              // 配列コピー
     quick_sort( 0 ,  MAX );    // クイックソート
     disp( data );
</script>
```



JavaScript a5-06

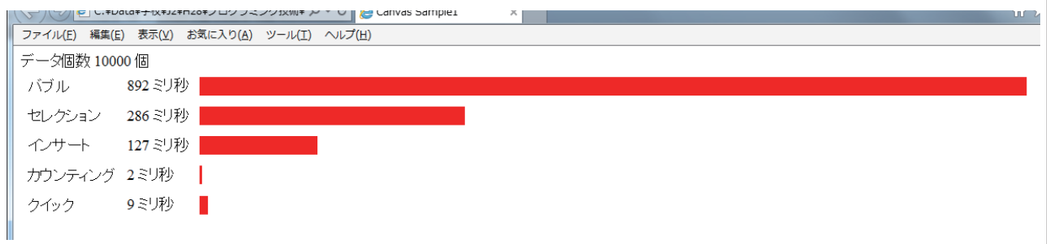
ソーティングアルゴリズムの評価

jsa5-6.html

```
<html>
<head>
<title>Canvas Sample1</title>
<meta content="text/html; charset=utf-8" http-equiv="Content-Type">

<script type="text/javascript">
    function ransu(max)
       {
        return  Math.floor(Math.random()*max);
    }
    function swap(ary , a , b)    // 配列データ交換
    {
        var dmy = ary[a];
        ary[a] = ary[b];
        ary[b] = dmy;
    }
    function ary_init( ary )   // 配列の初期設定
    {
        var i;
        for(i=0;i<=MAX;i++)    // 配列に順列をセット
            ary[i] = i;

        for(i=MAX;i>0;i--){    // 配列をばらばらにする
            r = ransu(i);
            swap( ary , i , r );
        }
    }

    function bubble_sort( ary )       // バブルソート（交換法）
    {
          var i,j,n;
          for( i=MAX; i>0; i-- ){
                for( j=0; j<i; j++ ){
                      if( ary[j] > ary[j+1] ){
                            swap( ary , j ,j+1 );  //   交換
                      }
                }
          }
    }
    function selection_sort( ary )    // 選択法
    {
          var i,j,n;
          for( i=0; i<MAX; i++ ){
                n=i;
                for( j=i; j<MAX; j++ ){
                      if( ary[n] > ary[j] ) n = j ;
                }
                swap( ary , i , n  );
          }
    }
    function insert_sort( ary )  // 挿入法
    {
          var i,j,chk;
          for( i=1; i<=MAX; i++ ){
                chk = ary[i];
                j = i-1;
                while( chk < ary[j] ){
                      ary[j+1] = ary[j];
                      j--;
                      if(j<0) break;
                }
                ary[j+1] = chk;
          }
    }
```



```
    function counting_sort( ary )  // カウンティングソート
    {
        var cnt = [],i,j,n=0;
        for( i=0; i<=MAX; i++ )
            cnt[i] = 0;
        for( i=0; i<=MAX; i++ )
            cnt[ ary[i] ]++;
        for( i=0; i<=MAX; i++ ){
            for( j=1; j<=cnt[i]; j++ ){
                ary[n] = i;
                n++;
            }
        }
    }
    var q_ary = [];
    function quick_sort( low ,  high )  // クイックソート
    {
          var ct,ctv,L,R;
          L = low;  R = high;

          ct = Math.floor( (low+high)/2 );
          ctv = q_ary[ct];
          while(1){
                while( q_ary[L] < ctv ) L++;
                while( q_ary[R] > ctv ) R--;
                if( L>= R ) break;
                swap( q_ary , L , R );
                L++; R--;
          }
          if( (L-low) >= 2) {quick_sort( low , L-1);}
          if( (high-R) >= 2) {quick_sort( R+1 , high);}
    }
</script>
</head>
<body>
<script type="text/javascript">
    var MAX = 10000;                                    // データ個数(変更して実行してみる)
    var data=[],i,r;
    var func = { "バブル"         : "bubble_sort(data)"     ,  // 連想配列
                 "セレクション"   : "selection_sort(data)"  ,
                 "インサート"     : "insert_sort(data)"     ,
                 "カウンティング" : "counting_sort(data)"   ,
                 "クイック"       : "quick_sort( 0 ,  MAX )"
               };
    var atart,end,time;

    document.write("データ個数 " + MAX + " 個<br><table cellpadding=5>");
    for( i in func ){
        ary_init(data);
        start = new Date().getTime();
        if(i==4) q_ary=data;              // クイックソートなら配列コピー
        eval( func[i] );                  // 文字列を JavaScript として実行
        end = new Date().getTime();
        time = end - start;               // ソーティング時間
        document.write("<tr><td> " + i + "</td><td>" + time + " ミリ秒</td>");
        document.write("<td><img src='img/red.png' width=" + time + " height=20></td></tr>");
    }
    document.write("</table>");
</script>
</body>
</html>
```



探索のアルゴリズム

○探索とは

配列の様なデータの集まりから、目的とするデータを見つけ出す処理を「探索」と呼びます。代表的なアルゴリズムを以下に示します。

○直接探索

配列に格納したデータを添字の値で直接探索する手法。例えば名簿番号と添字の値が一致している場合などである。

○線形探索

配列のデータを先頭から一つ一つ順番に目的のデータと比較しながら探索する手法。

○二分探索

あらかじめ整列されている配列のデータに対して、探索範囲の中間に存在するデータと目的のデータを比較し、探索範囲を二分しながら探索する手法。

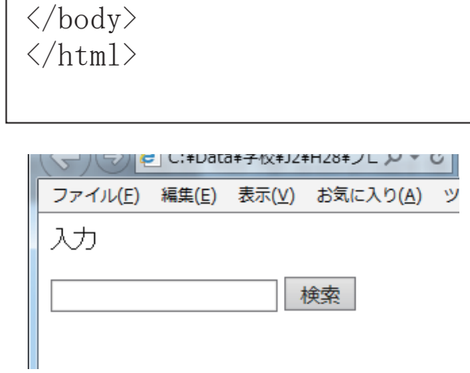
○ハッシュ探索

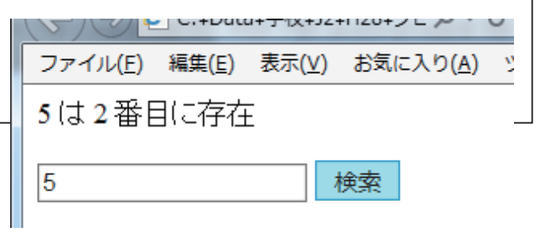
ハッシュ関数を使用して、配列のデータの中から目的のデータが格納されている添字を計算で求める手法。

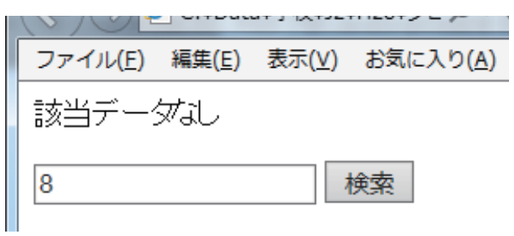
JavaScript a6-01 線形探索

jsa6-1.html

```
<html>
<head>
<title>1</title>
<meta content="text/html; charset=utf-8" http-equiv="Content-Type">

<script type="text/javascript">
    var a = [1,3,5,7,9,11,13,15,17,19];

    function bclick()
    {
        var i;
        var t1 = document.getElementById('text1').value;
        var m = document.getElementById('msg');
        for( i=0; i<a.length; i++ ){
            [                                    ]
        }
        if ([                    ]
            m.innerHTML = "該当データなし";
        else
            m.innerHTML = a[i] + "は" + i + "番目に存在";
    }
</script>

</head>
<body>
    <p id="msg">入力</p>
    <form>
        <input type="text" id="text1">
        <input type="button" value="検索" onClick="bclick();">
    </form>
</body>
</html>
```







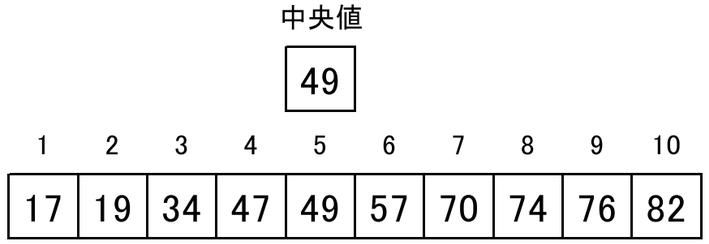
２分探索のアルゴリズム

79 を検索する場合

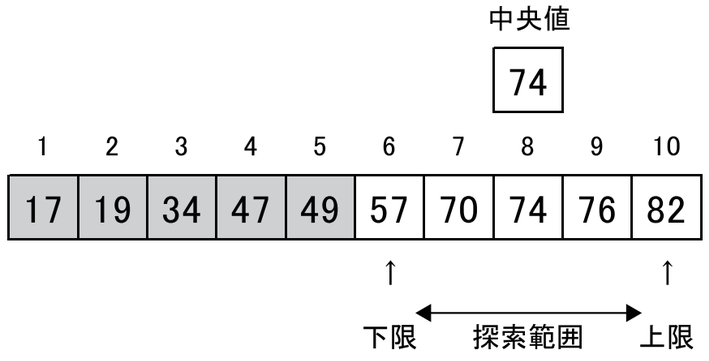
二分探索方は、探索するデータと整列された配列データの中央値を比較しながら、探索範囲を半分づつに狭めていく手法であり、人間が辞書を引く感覚に似ている。

$$\text{中央値} = \frac{(\text{下限}+\text{上限})}{2} \quad (\text{小数点以下切り捨て})$$

まず探索範囲の下限と上限から中央値（添字）を算出する。



中央値のデータと探索データを比較し、一致しない場合には探索範囲を半分にする。
この例では探索データは中央値よりも大きいので、配列の後半部分にあることがわかる。

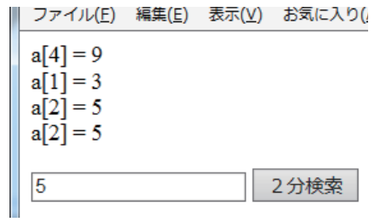


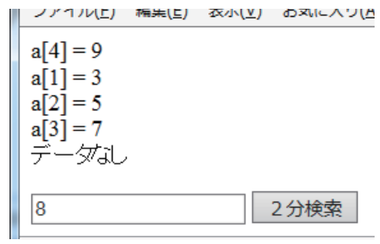
下限をずらし、新たな中央値を算出し、探索データと一致するか、探索範囲がなくなるまで繰り返す。

JavaScript a6-02 ２分探索

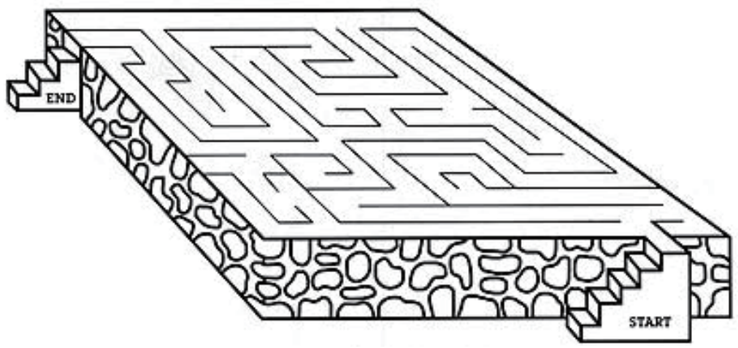
jsa6-2.html

```
<html>
<head>
<title>1</title>
<meta content="text/html; charset=utf-8" http-equiv="Content-Type">
<script type="text/javascript">
    var a = [1,3,5,7,9,11,13,15,17,19];
    function bclick()
    {
        var k,j,c,str;
         var t1 = document.getElementById('text1').value;
         var m = document.getElementById('msg');
        str="";
         k = 0;
        j = a.length - 1;;
         c = Math.floor( (k+j)/2 );
         while(  k <= j  ){
              str += "a[" + c + "] = " + a[c] + "<br>";
              if( t1 == a[c] ) break;
              if( t1 < a[c] )
                   j = c - 1 ;
              else
                   k = c + 1 ;
              c =                                    ;
         }
         if (  k > j )
              str += " データなし ";
         else
              str += "a[" + c +"] = " + t1 ;
        m.innerHTML = str;
    }
</script>
</head>
<body>
     <p id="msg"> 入力 </p>
     <form>
          <input type="text" id="text1">
          <input type="button" value=" ２分検索 " onClick="bclick();">
     </form>
</body>
</html>
```





迷路生成のアルゴリズム



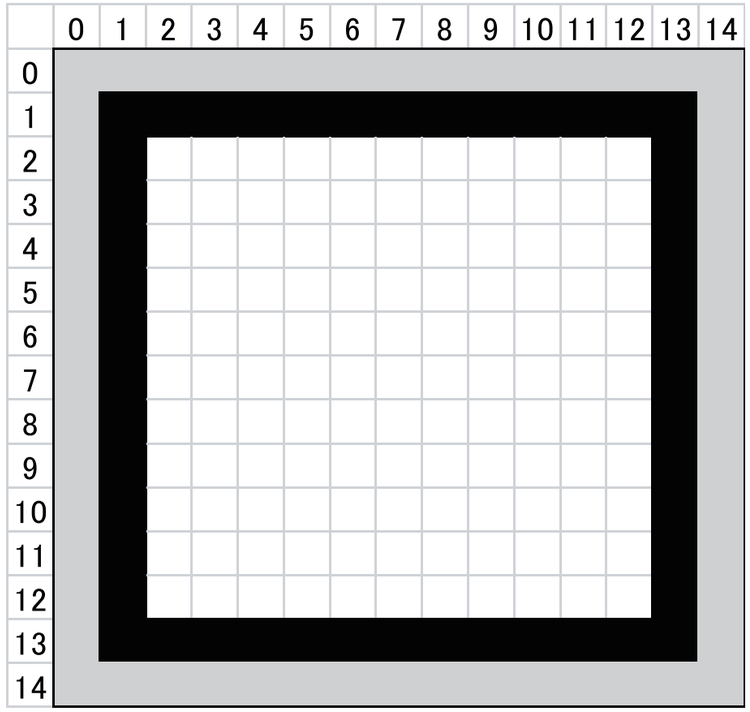
ゲームなどで使用される迷路の生成には、様々なアルゴリズムがある。そのうちのいくつかをプログラミングしてみよう。

迷路データは２次元配列とし、０が道、１を壁として考える。
１５×１５の配列の場合には図のようになります。（０～１４）

壁	壁	壁	壁	壁
壁	道	道	道	壁
壁	道	壁	道	壁
壁	道	壁	道	道
壁	道	壁	壁	壁

↓

1	1	1	1	1
1	0	0	0	1
1	0	1	0	1
1	0	1	0	0
1	0	1	1	1



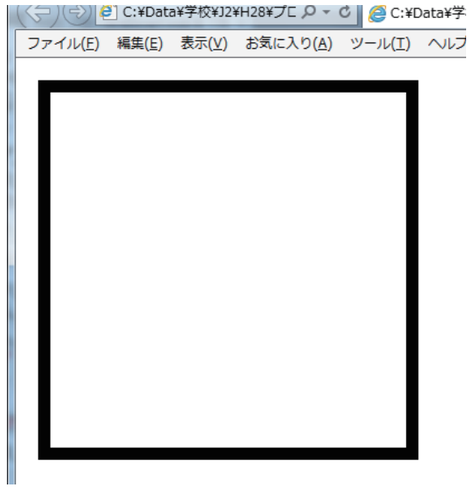
まずはこの配列を用意、外壁を設定し、これを表示させてみます。

JavaScript a7-01

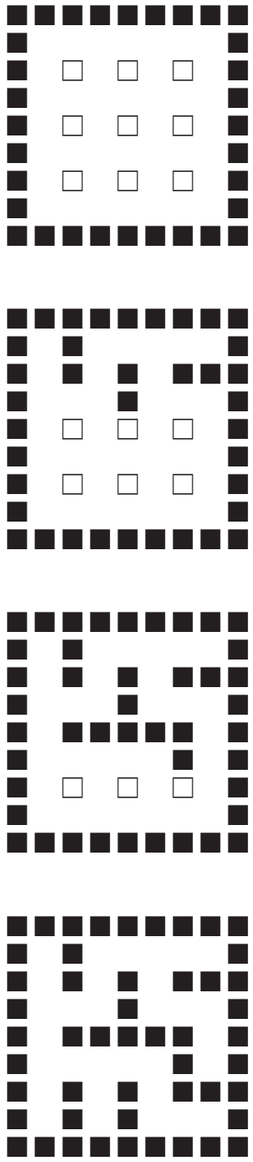
初期設定（迷路生成）

jsa7-1.html

```
<html>
<head>
<meta charset="UTF-8">
<script type="text/javascript">
    var W=32,H=32;              // 迷路の大きさ  W:横 H:縦  (偶数)
    var maze = [];              // 迷路用配列

    function ransu(max)         // 0 → max-1 までの一様乱数
    {
        return  Math.floor(Math.random()*max);
    }

    function maze_disp()        // 迷路表示関数
    {
      for(var y=0;y<=H;y++){
        for(var x=0; x<=W;x++){
            switch( maze[y][x] ){
                case 0 : document.write("<img src='img/white.png'>"); break;
                case 1 : document.write("<img src='img/black.png'>"); break;
                case 2 : document.write("<img src='img/red.png'>");   break;
            }
        }
        document.write("<br>");
      }
    }
</script>
</head>
<body>
<script type="text/javascript">
    var x,y;

    for(y=0;y<=H;y++){      // 全てを０クリア
        maze[y] = [];
        for(x=0; x<=W;x++){
            maze[y][x] = 0;
        }
    }
    for(var y=1; y<=H-1; y++ ){  // 外枠を設定
        for(var x=1; x<=W-1; x++ ){
            if(x==1 || x==W-1 || y==1 || y==H-1)
                maze[y][x] = 1;
        }
    }
    maze_disp();
</script>
</body>
</html>
```



１．棒倒し法



棒倒し法とは、迷路の盤上の基点から上下左右のどこかに壁を伸ばす事により、迷路を生成するアルゴリズムです。
基点から棒を倒していくような動作をするので、棒倒し法と呼ばれます。

棒倒し法では次のように迷路を生成します。

１．まず、外周の壁と１つ飛ばしに棒を倒す対象となる基点の壁があります。
（「□」が基点の壁になります）

２．左上の基点から右の基点に向けて上下左右にランダムに棒（壁）を倒します。

３．続けて、２行目以降も棒を倒しますが、**２行目以降については上方向以外にランダムに棒を倒します。（発展課題の部分）**

４．３行目も手順３と同様に棒を倒します。

５.一番下の行まで棒を倒し終わったら迷路の完成です。

発展課題

②のスクリプトはどんな位置でも上下左右いずれかに棒を倒します。
２行目以降については上方向には倒さないように修正しなさい。

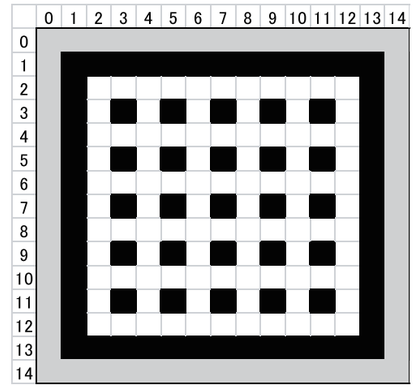
jsa7-2.html

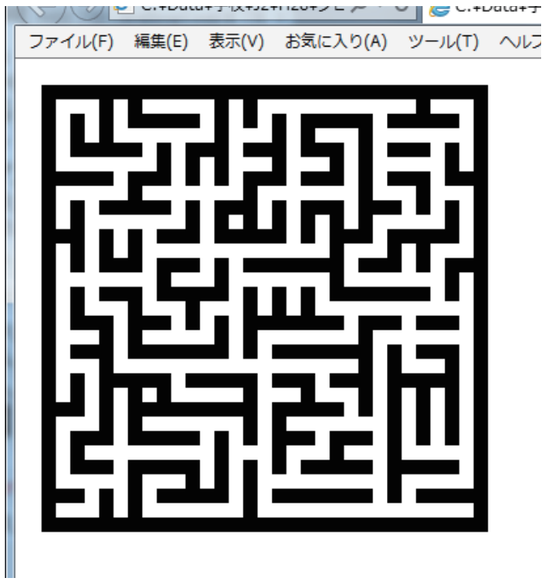
JavaScript a7-02

棒倒し法（迷路生成）

jsa7-2.html

```
<html>
<head>
<meta charset="UTF-8">
<script type="text/javascript">
    var W=32,H=32;              // 迷路の大きさ  W:横 H:縦  (偶数)
    var maze = [];              // 迷路用配列

    function ransu(max)         // 0 → max-1 までの一様乱数
    {
        return  Math.floor(Math.random()*max);
    }

    function maze_disp()        // 迷路表示関数
    {
          ・・・省略
    }

    function boutaosi()         // 棒倒し法
    {
        var r;
        for(var y=3;y<=H-3;y+=2){
            for(var x=3; x<=W-3;x+=2){
                maze[y][x] = 1;    // 起点の壁
                r = ransu(4) ;     // 0～3 の乱数
                switch( r ){
                    case 0 : maze[y-1][x  ] = 1 ; break;  // 上に倒す
                    case 1 :                      break;  // 下  〃
                    case 2 :                      break;  // 左  〃
                    case 3 :                      break;  // 右  〃
                }
            }
        }
    }
</script>
</head>
<body>
<script type="text/javascript">
    var x,y;

    ・・・初期設定（外枠）部分省略

    boutaosi();  // 棒倒し法
    maze_disp();
</script>
</body>
</html>
```





２．穴掘り法（道延ばし法）

穴掘り法とは、盤面の初期状態を全て壁の状態とし、ある地点から穴を掘るかのように道を伸ばして行くことで迷路を生成するアルゴリズムです。

1. 初期状態として以下の９×９の盤面があるとします。

2. 一番外側の外周は壁とし内側の７×７の領域の中で穴掘りの開始点を決めます。　ここでは、○を開始点とします。

3. 上下左右の２マス先を見て、それが壁である場合は穴を掘ります。 ここでは、上に進み、○の地点が最奥とします。

4. 最奥の地点から穴を掘るという「3.」の処理を穴を掘れる場所が無くなるまで繰り返します。
　例えば、左のように○の位置から上下左右２マス先が全て壁ではないため、これ以上は掘ることができません。

5. 「4.」のように掘ることができなくなった場合、既に掘った道から枝を生やすように、次の道を掘り進めます。

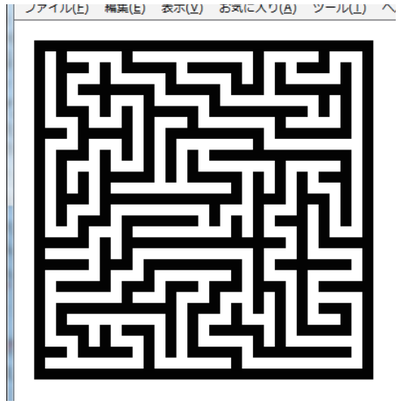
6. このように「3.」「4.」の処理である、掘れなくなるまで掘り進め、
　突き当たりに来たら、道上から新たな道を伸ばすという処理を繰り返して迷路が完成します。

JavaScript a7-03

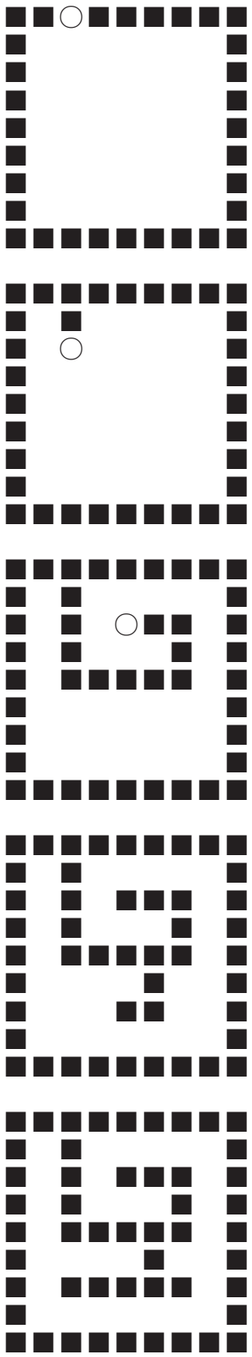
穴掘り法（道延ばし法）

jsa7-3.html

```
<html>
<head>
<meta charset="UTF-8">
<script type="text/javascript">
    var W=32,H=32;              // 迷路の大きさ  W:横 H:縦  (偶数)
    var maze = [];              // 迷路用配列

    function ransu(max)         // 0 → max-1 までの一様乱数
    {
        return  Math.floor(Math.random()*max);
    }

    function maze_disp()        // 迷路表示関数
    {
            ・・・省略
    }
     function anahori()
     {
         for(y=1;y<=H-1;y++){          // 内側を全て壁で埋める
             for(x=1; x<=W-1;x++){
                 maze[y][x] = 1;
             }
         }
         a1(2,2);                      // 穴掘り関数コール
     }
     function a1( x,y )
     {
         maze[y][x] = 0;
         var flag=[0,0,0,0];           // ４方向チェック用のフラグ
         var sum = 0,r;
         while(sum != 4){              // ４方向のいずれかに行けるうちは繰り返す
             r = ransu(4);             // 掘る方向（乱数）
             switch( r ){                          // 再帰呼び出し
                 case 0 : if(maze[y-2][x  ] == 1) {                    } break;
                 case 1 : if(maze[y+2][x  ] == 1) {                    } break;
                 case 2 :                         { maze[y  ][x-1]=0; a1(x-2,y  );} break;
                 case 3 :                         { maze[y  ][x+1]=0; a1(x+2,y  );} break;
             }
             flag[r]=1;                                // 掘った方向にフラグを立てる
             sum = flag[0]+flag[1]+flag[2]+flag[3];
         }
     }
</script>
</head>
<body>
<script type="text/javascript">
    var x,y;
    ・・・初期設定（外枠）部分省略
     anahori();  // 穴掘り法
    maze_disp();
</script>
</body>
```



3．壁延ばし法



壁伸ばし法とは、盤面の初期状態を全て壁の状態とし、ある壁を伸ばしていく形で迷路を生成するアルゴリズムです。
全てが壁の状態から道を伸ばしていく穴掘り法と考え方は同じであり、道ではなく壁を伸ばすという違いになります。
壁の伸ばし方に関しては色々と工夫が考えられますが、ここでは既存の壁から壁を伸ばしていく方法で迷路を作成したいと思います。

1. 初期状態として以下の９×９の盤面があるとします。

2. 一番外側の外周は壁とし、既存の壁である外周の壁の内ランダムに開始点を決めます。ここでは、〇を開始点とします。

3. 上下左右の２マス先を見て、それが道である場合には壁を伸ばします。

4. 端の地点から壁を伸ばすという「3.」の処理を壁が伸ばせなくなるまで続けます。
　例えば､左図のような場合はもう伸ばすことができません。

5. 「4.」のように伸ばせなくなった場合、既に存在する壁から枝を生やすように、 次の壁を掘り進めます。
　ここでは、伸ばせる位置まで戻り処理を続けます。

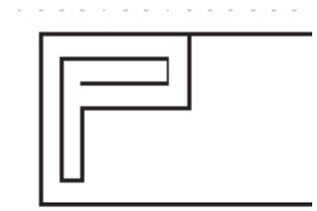
6. このように「3.」「4.」の処理を繰り返しながら壁を伸ばし続け、 突き当たりに来たら、壁上から新たな道を伸ばすという処理を繰り返して迷路が完成します。

壁延ばし法の
手動シミュレーション

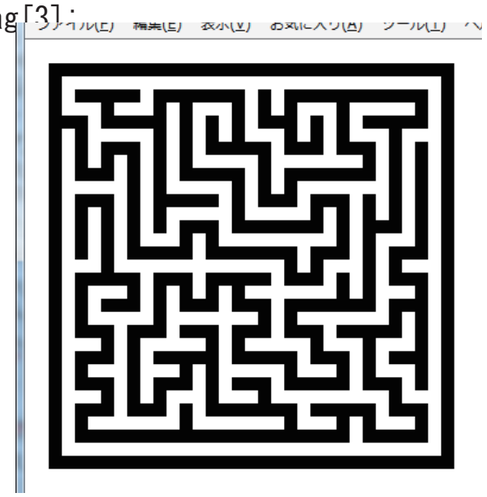
JavaScript a7-04 壁延ばし法

jsa7-4.html

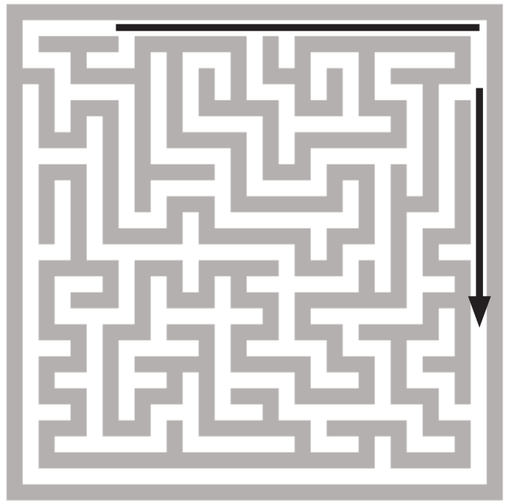
```
<html>
<head>
<meta charset="UTF-8">
<script type="text/javascript">
    var W=32,H=32;              // 迷路の大きさ  W:横 H:縦  (偶数)
    var maze = [];              // 迷路用配列

    function ransu(max)         // 0 → max-1 までの一様乱数
    {
        return  Math.floor(Math.random()*max);
    }

    function maze_disp()        // 迷路表示関数
    {
            ・・・省略
    }

     function kabenobasi( x,y )
     {

         maze[y][x] = 1;
         var flag=[0,0,0,0];     // ４方向チェック用のフラグ
         var sum = 0,r;
         while(sum != 4){        // ４方向のいずれかに行けるうちは繰り返す
             r = ransu(4);       // 延ばす方向（乱数）
             switch( r ){                                  // 再帰呼び出し
                 case 0 : if(                ) { maze[y-1][x  ]=1; kabenobasi(x  ,y-2);} break;
                 case 1 : if(                ) { maze[y+1][x  ]=1; kabenobasi(x  ,y+2);} break;
                 case 2 : if(maze[y  ][x-2] == 0) {                  } break;
                 case 3 : if(maze[y  ][x+2] == 0) {                  } break;
             }
             flag[r]=1;              // 延ばした方向にフラグを立てる
             sum = flag[0]+flag[1]+flag[2]+flag[3];
         }
     }

</script>
</head>
<body>
<script type="text/javascript">
    var x,y;
    ・・・初期設定（外枠）部分省略

    kabenobasi(1,5);  // 壁延ばし法
    maze_disp();
</script>
</body>
</html>
```



上下左右どこへも延ばせなくなったら
延ばせるポイントまで戻る
（再帰呼び出しをリータン）



迷路生成のアルゴリズム③

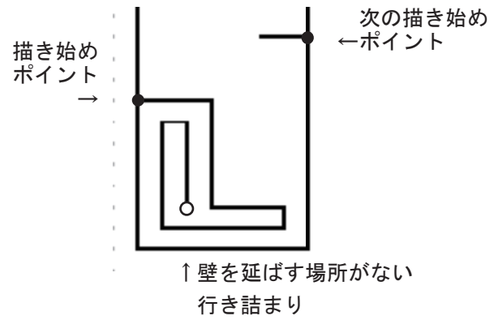


アルゴリズムの工夫

前ページのプログラムのままだと右図のように常に一番外側にルートができてしまいます。もう少し迷路らしい迷路にするにはアルゴリズムの工夫が必要です。

手書きで迷路を作る時に、より迷路らしくするためにどんな手順で描きましたか？
具体的にどんなふうなアルゴリズムに変更すればうまくいきそうですか？
手書きでシミュレーションしながら考えましょう！

◆ひとつのアイデア



左側の壁から描き始め
ランダムに壁を延ばしていくが、上下左右どこへも延ばせない状態になったら、今までは延ばせる場所まで戻っていたが・・・

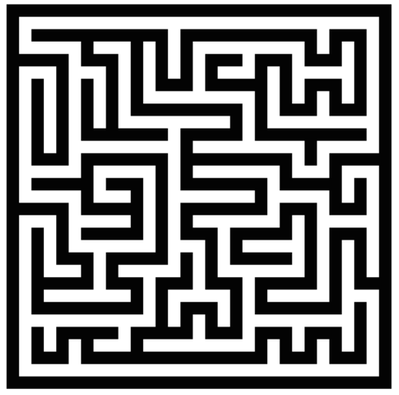
左側と右側の壁の交互に壁に新しいポイントを設定し、そこから壁を延ばしてみる。

行き詰るたびに左右の壁から延ばしていくと、関数を無限に呼び出してしまうため、左右の壁から始めるのは、ある程度回数を決めておく。

JavaScript a7-05

壁延ばし法の改良

jsa7-5.html



```
・・・省略
    var CNT = 0 , S = 1;
    function kabenobasi( x,y )
    {
        maze[y][x] = 1;
        var flag=[0,0,0,0];
        var sum = 0,r;
        while(sum != 4){
            r = ransu(4);
            switch( r ){
                case 0 : if(maze[y-2][x  ] == 0) { maze[y-1][x  ]=1; kabenobasi2(x  ,y-2);} break;
                case 1 : if(maze[y+2][x  ] == 0) { maze[y+1][x  ]=1; kabenobasi2(x  ,y+2);} break;
                case 2 : if(maze[y  ][x-2] == 0) { maze[y  ][x-1]=1; kabenobasi2(x-2,y  );} break;
                case 3 : if(maze[y  ][x+2] == 0) { maze[y  ][x+1]=1; kabenobasi2(x+2,y  );} break;
            }
            flag[r]=1;
            sum = flag[0]+flag[1]+flag[2]+flag[3];
        }
        if( CNT < 5 ){                  // 左右の壁からスタートするのは５回だけ
            CNT++;
            S = [                    ] // 左右の壁のいずれからか交互に延ばし始める
            var r = ransu((H/2))*2+1 ;                    ※　条件演算子を使用
            kabenobasi( S , r );
        }
    }
</script>
</head>
```

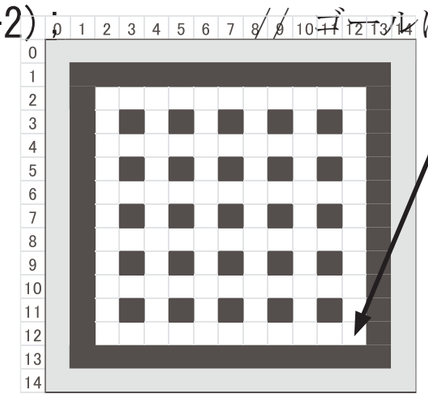
時々エラーで終息しない事あり!

JavaScript a7-06

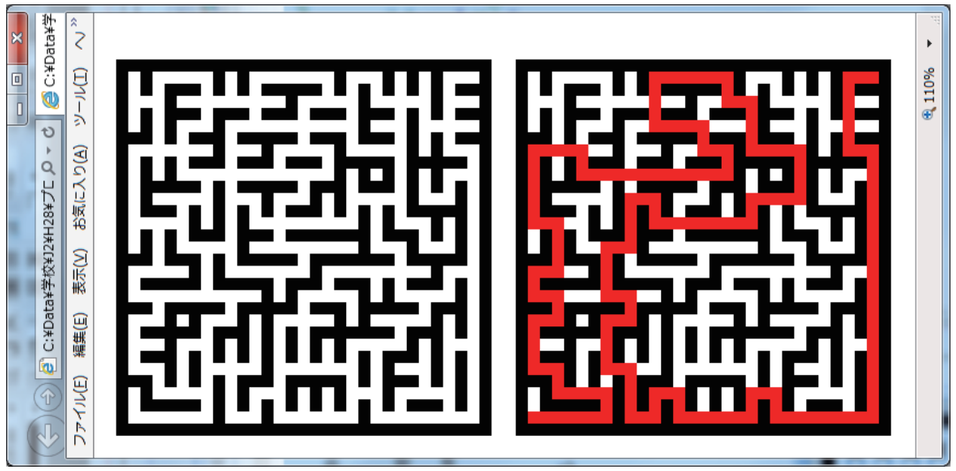
迷路探索のアルゴリズム例

jsa7-6.html

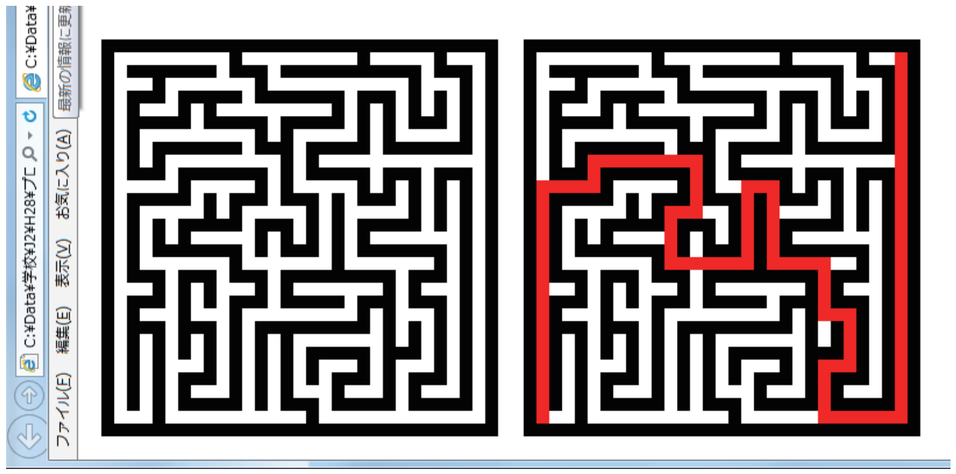
```
・・・省略

    function search(x,y,ex,ey)     // 現在位置とゴール位置
    {
        if( maze[y][x] != 0 ) return; // 壁なら戻る
        maze[y][x]=2;                 // 訪れたコマは赤色に
        if(x==ex && y==ey){           // ゴールなら
            maze_disp();              // 表示
            exit;                     // 終了（エラーを強制発行）
        }
                          // 全ての道を行き止まりになるまでたどる（再帰呼び出し）
         search(x  ,y-1,ex,ey);   // 上
         search(x  ,y+1,ex,ey);   // 下
         search(x-1,y  ,ex,ey);   // 左
         search(x+1,y  ,ex,ey);   // 右
         maze[y][x]=0;            // 行き止まりから戻る（白色）
    }
</script>
</head>
<body>
<script type="text/javascript">
     ・・・初期設定部分省略

kabenobasi(1,5);                  // 迷路生成関数
maze_disp();                      // 迷路を表示
search(2,2,  W-2,H-2);            // ゴールは右下
    // スタート      ゴール
</script>
</body>
</html>
```



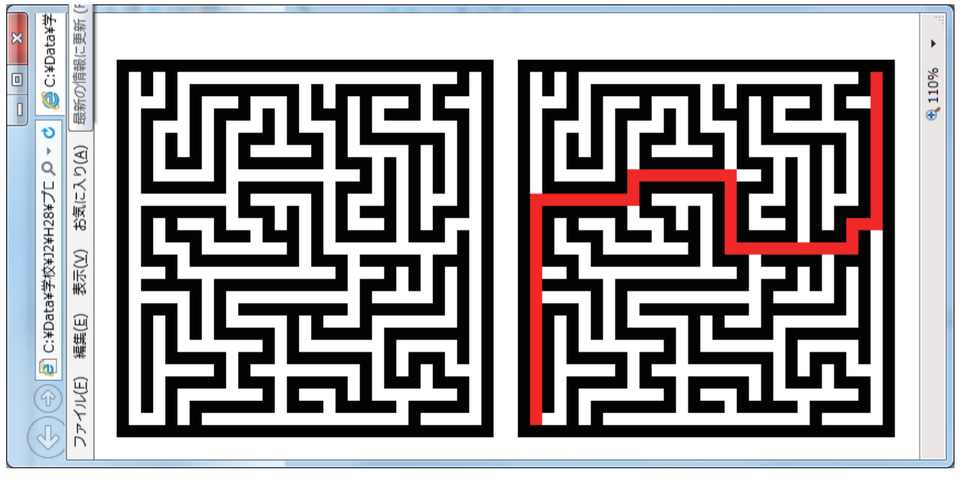
壁倒し法



穴掘り法



壁延ばし法



MINOSHIN Jyanken CUP

日　時：
場　所：電算室
参加者：プログラミング技術　受講者

じゃんけん…3 種類の指の出し方（グー・チョキ・パー）で三すくみを構成し、勝敗を決める手段。日本で拳遊びを基に考案され、現代では世界的に普及が進んでいる。単純だがその奥深さは計り知れない。古来より日本人はその必勝法を探求し続けているが、いまだにそれは発見されていない…。じゃんけんに「最強のアルゴリズム」は存在するのか?

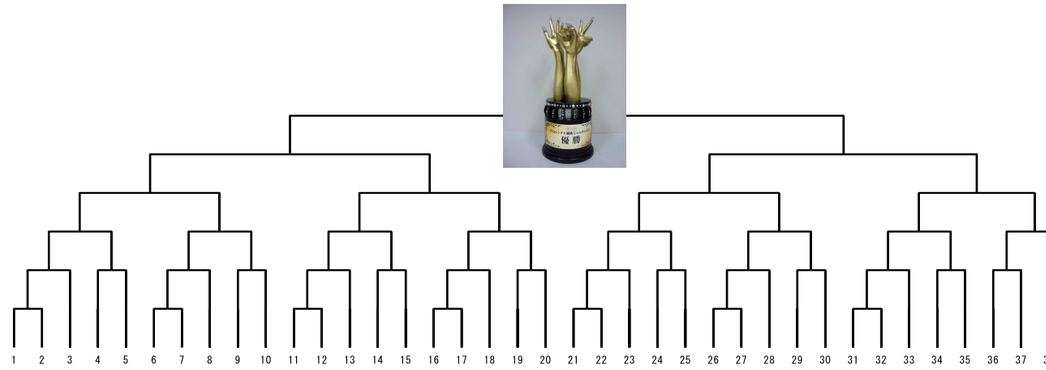
OIDE生の挑戦が今、幕を開ける!!

ルールと競技方法

- 各自、じゃんけんの手のアルゴリズムを考案し、プログラム化する。
- 乱数のみのアルゴリズムはNG。必ず何か工夫を加えること。
- ベースのプログラムに関数化したプログラムをインクルードし実行。
- 自動的に300回×3回、勝負させ、勝ち数が多い方を勝者とする。
- 勝ち数が同じ場合は勝敗がつくまで（どちらかが2勝）再戦を行う。
- 相手の手を先読み（後出し）するようなプログラムは反則。
- トーナメント方式で1位～3位までを決定。
- 最初の対戦時に自身のアルゴリズムをプレゼンする。(2分間)
- 審査員が優秀と認めたアルゴリズムには特別賞を授与する。

スケジュール

１回目	_____	競技説明・導入（ベースとなるプログラム確認）
２回目	_____	サンプルプログラム入力・アルゴリズム考
３回目	_____	アルゴリズム考案　→　提出・調整
４回目	_____	プログラム・プレゼン資料作成
５回目	_____	〃
６回目	_____	プログラム、プレゼン資料提出、組合せ決定
７回目	_____	じゃんけん大会
予備日	_____	



ベースとなるメインプログラム
(jyanken.html)Ver 1.0


```
<!DOCTYPE html>
<html>

<head>
<script type="text/javascript" src="j2700-40.js"></script>
<script type="text/javascript" src="seedrandom.min.js" ></script>
<meta content="text/html; charset=utf-8" http-equiv="Content-Type">


<title>じゃんけん対戦</title>
<script type="text/javascript">
    var p1_no,p2_no, p1_func,p2_func,kekka,msg;
    var DAT=[],AITE=[],H=[] ,CNT,JKN = [' グ',' チ',' パ'];
    var TE = [ [0,0,0],[0,0,0] ];

    var KAISU = 300;
    var TAISEN_KAISU = 3,T_CNT=0;
    var str;

    function init()
    {

        for(var i=0;i<=41;i++){
            DAT[i] = [];
            for(j=0;j<=KAISU;j++)
                DAT[i][j] = 0;
        }

        var player1 = document.getElementById("text1").value;
        var player2 = document.getElementById("text2").value;
        msg = document.getElementById("msg");
        var s1 = document.getElementById("source1");
        var s2 = document.getElementById("source2");
        kekka = document.getElementById("t_kekka");
        var p1img = document.getElementById("p1img");
        var p2img = document.getElementById("p2img");

        p1_func = "h27c" + player1 ;
        p2_func = "h27c" + player2 ;

        p1_no = Number(player1);
        p2_no = Number(player2);

        AITE[ p1_no ] = p2_no;
        AITE[ p2_no ] = p1_no;

        p1img.src = "img/" + p1_func + ".png";
        p2img.src = "img/" + p2_func + ".png";
                p1name.innerHTML = NAME[p1_no];
                p2name.innerHTML = NAME[p2_no];

        s1.value = window[ p1_func ];
        s2.value = window[ p2_func ];

}
function start()
{

    CNT=1; H=[0,0,0]; TE = [ [0,0,0],[0,0,0] ];
     T_CNT++;

    msg.innerHTML = "";
    kekka.value = "";

    jan_loop();

    //setTimeout("start()",5000);
}
function jan_loop()
{
    if( CNT > KAISU ) {
        var p1img,p2img;
```

フォルダの構成

jyanken.html　メインプログラム

m18a00-40.js　インクルードファイル

m18a00.js　　対戦相手サンプル

m18a___.js　　自分のプログラム

（名簿番号）

--- img フォルダ（イメージ）

　　m18a___.png　自分の画像

　　（名簿番号）

　　m18a00.png　サンプル画像

具体的な手順

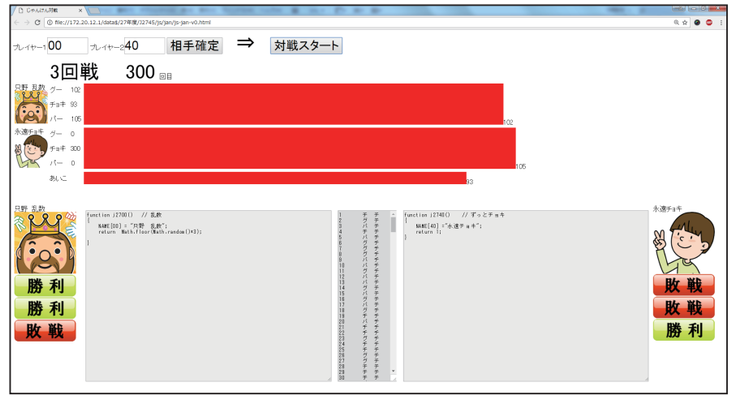
①自身のフォルダへ新フォルダ作成

②クラス共通フォルダの
「じゃんけん大会」の内容をコピー

③自分のアルゴリズムを、指定した
ファイル名、関数名でプログラミングする。（デバッグ）
対戦相手用のファイルも準備。
イメージファイルも準備する

④完成したら指定フォルダへコピー
（クラス共通フォルダ内、後日指定）


```
                var str1 = "point1" + T_CNT;
                var str2 = "point2" + T_CNT;

                p1img = document.getElementById(str1);
                p2img = document.getElementById(str2);

    if( H[1] > H[2] ){
            p1img.src = "img/win.png";
            p2img.src = "img/lost.png";
    }
            if( H[1] < H[2] ){
            p1img.src = "img/lost.png";
            p2img.src = "img/win.png";
            }
            if( H[1] == H[2] ){
            p1img.src = "img/draw.png";
            p2img.src = "img/draw.png";
            }

        return;
    }
    var p1 = eval( p1_func + "();" );
    var p2 = eval( p2_func + "();" );
    DAT[p1_no][CNT] = p1;  TE[0][p1]++;
    DAT[p2_no][CNT] = p2;  TE[1][p2]++;
    var hantei = ( p2 - p1 + 3 ) % 3;
    H[ hantei ]++;
    str="";
    str+= "<table cellpadding=2 >"
  str += "<tr><td></td><td colspan=2><font size=7>" + T_CNT + "回戦　　" + CNT + "</font>　回目</td></tr>" ;
  str += "<tr><td rowspan=3>" + NAME[p1_no] + "<br><img src='img/" + p1_func + ".png' width =80>" + "</td>" ;
  str += "<td>グー　　" + TE[0][0] + "</td>";
  str += "<td rowspan=3><img src='img/red.png' width=" + H[1]*10 + " height=100>" + H[1] + "</td></tr>";
  str+= "<tr><td>チョキ　" + TE[0][1] + "</td></tr><tr><td>パー　　" + TE[0][2] + "</td></tr>";
  str += "<tr><td rowspan=3>" + NAME[p2_no] + "<br><img src='img/" + p2_func + ".png' width =80>" + "</td>";
  str += "<td>グー　　" + TE[1][0] + "</td>";
  str += "<td rowspan=3><img src='img/red.png' width=" + H[2]*10 + " height=100>" + H[2] + "</td></tr>";
  str+= "<tr><td>チョキ　" + TE[1][1] + "</td></tr><tr><td>パー　　" + TE[1][2] + "</td></tr>";
 str += "<tr><td></td><td>あいこ</td><td><img src='img/red.png' width=" + H[0]*10 + " height=30>" + H[0] + "</td></tr>";
    str+="</table>";

        kekka.value += CNT + "¥t";;
        kekka.value += JKN[ DAT[p1_no][CNT] ] + "　";
        kekka.value += JKN[ DAT[p2_no][CNT] ] + "¥n";

        msg.innerHTML = str;
        CNT++;
        setTimeout("jan_loop()",10);
    }
</script>

</head>
<body >
プレイヤー１<input type ="text" id="text1" size=2 style="font-size:30px;">
プレイヤー２<input type ="text" id="text2" size=2 style="font-size:30px;">
<input type ="button" id="button1" value=" 相手確定 " onClick="init()" style="font-size:30px;"><font size=7>　⇒　</font>
<input type ="button" id="button2" value=" 対戦スタート " onClick="start()" style="font-size:30px;"><p>

<div id="msg" style="height:350px;">じゃんけん対戦</div>
<table border=0 width=100%><tr><td valign=top>
<div id="p1name">プレイヤー１</div>
<img id="p1img" src="img/man.png" width=150>
<img id="point11" src="img/toumei.png" width=150 height=53><br>
<img id="point12" src="img/toumei.png" width=150 height=53><br>
<img id="point13" src="img/toumei.png" width=150 height=53><br>
<img id="point14" src="img/toumei.png" width=150 height=53><br>
<img id="point15" src="img/toumei.png" width=150 height=53>
</td><td width=80%>
<textarea id="source1" rows="30" style="width:43%; font-size:14px; background: #eeeeee;">プレイヤー１のソースリスト</
textarea>
<textarea id="t_kekka" rows="30" style="width:10%; font-size:14px; background: #dddddd;">対戦結果</textarea>
<textarea id="source2" rows="30" style="width:43%; font-size:14px; background: #eeeeee;">プレイヤー２のソースリスト</
textarea>
</td><td valign="top">
<div id="p2name">プレイヤー２</div>
<img id="p2img" src="img/man.png" width=150 align=top>
<img id="point21" src="img/toumei.png" width=150 height=53><br>
<img id="point22" src="img/toumei.png" width=150 height=53><br>
<img id="point23" src="img/toumei.png" width=150 height=53><br>
<img id="point24" src="img/toumei.png" width=150 height=53><br>
<img id="point25" src="img/toumei.png" width=150 height=53>
</td></tr></table>
</body>
</html>
```



じゃんけん（勝敗判定）のアルゴリズム

グー（0） チョキ（1） パー（2）とする

○勝敗判定のアルゴリズムの例

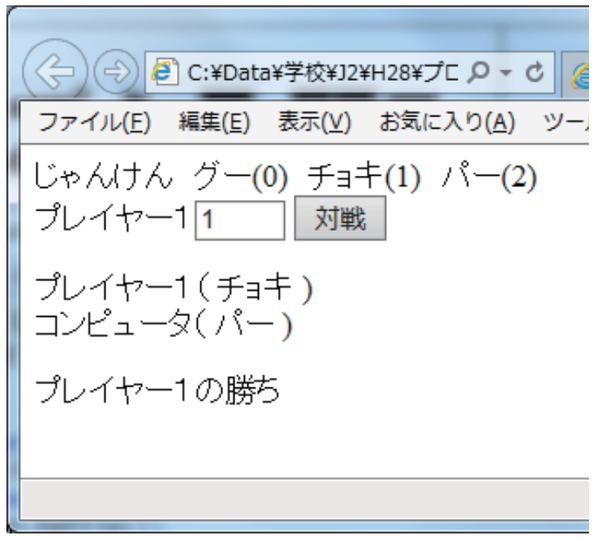
プレイヤー１を基準として … あいこ 0　勝ち 1　負け 2

```
if(  p1 == 0   ){   // プレイヤー１がグー
      if( p2 == 0 )  hantei = 0 ;      // プレイヤー２ グー   →  あいこ
      if( p2 == 1 )  hantei = 1 ;      // プレイヤー２ チョキ →  プ１ 勝ち
      if( p2 == 2 )  hantei = 2 ;      // プレイヤー２ パー   →  プ２ 負け（プ２ 勝ち）
}
if(  p1 == 1  ){    // プレイヤー１がチョキ
      if( p2 == 0 )  hantei = 0 ;      // プレイヤー２ グー   →  プ２ 勝ち
      if( p2 == 1 )  hantei = 1 ;      // プレイヤー２ チョキ →  あいこ
・・・・
・・・・
if ( hantei == 0 ) printf("あいこ  です¥n");
if ( hantei == 1 ) printf("プ１の勝ち¥n");
if ( hantei == 2 ) printf("プ２の勝ち¥n");
・・・・
・・・・
```

様々な問題を解決する場合、その法則を見つけ数式化することでプログラムコードを劇的に少なくすることができる場合がある。ただし、単純な判定文などで記述したほうが分かりやすいコードになる場合もあるので、何でも数式化するわけではなく使い分けも大切である。

勝敗の法則性を見つける（数式化）

プレイヤー1からみた勝敗		プレイヤー2		
		グー	チョキ	パー
プレイヤー1		0	1	2
グー	0			
チョキ	1			
パー	2			



JavaScript a8-01

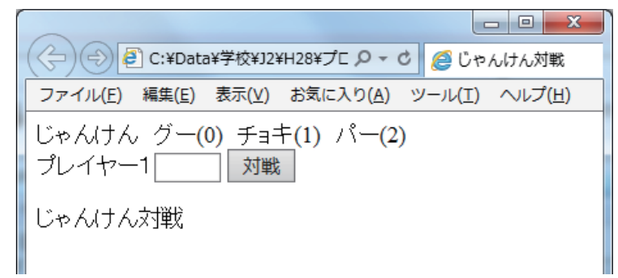
簡単なじゃんけんゲーム１

jsa8-1.html

```
<html>
<head>
<meta content="text/html; charset=utf-8" http-equiv="Content-Type">
<title> じゃんけん対戦 </title>
<script type="text/javascript">
    function battle()
    {
        var jan = ["グー","チョキ","パー"],str="";
        var t1 = document.getElementById("text1");
        var msg = document.getElementById("msg");

        var p1 = t1.value;                          // プレイヤー１は textbox 入力
        var p2 = [                                  ] // プレイヤー２はコンピュータ

        str += "プレイヤー１ ( " + jan[ p1 ] +" )<br>";
        str += "コンピュータ ( " + jan[ p2 ] +" )<p>";

        var hantei = [                                  ]// 勝敗判定
        switch ( hantei ){
            case 0 : str += "あいこ";                break;
            case 1 : str += "プレイヤー１の勝ち"; break;
            case 2 : str += "コンピュータの勝ち"; break;
        }
        msg.innerHTML = str;
    }
</script>

</head>
<body >
じゃんけん　グー (0)　チョキ (1)　パー (2)<br>
プレイヤー１<input type ="text" id="text1" size=2>
<input type ="button" id="button1" value="対戦" onClick="battle()" ><p>
<div id="msg"> じゃんけん対戦 </div>
</body>
</html>
```

グー	0
チョキ	1
パー	2

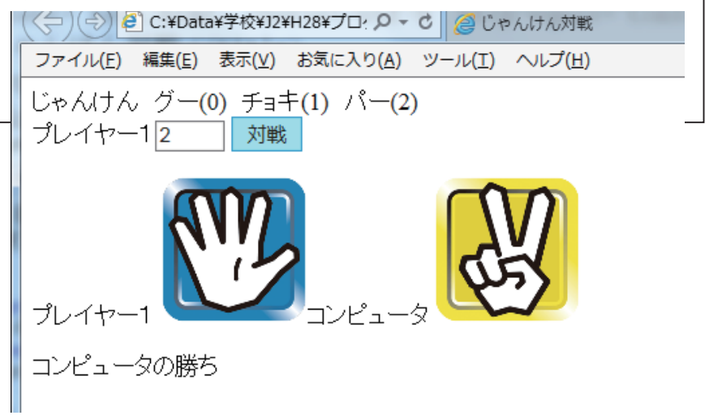
プレイヤー１から見て	
あいこ	0
勝ち	1
負け	2



JavaScript a8-02 結果をイメージ表示

jsa8-2.html

```
<html>
<head>
<meta content="text/html; charset=utf-8" http-equiv="Content-Type">
<title> じゃんけん対戦 </title>
<script type="text/javascript">
    function battle()
    {
        var jan = ["グー","チョキ","パー"],str="";
        var jan_img = ["jan_goo.gif","jan_cyoki.gif","jan_paa.gif"];
        var t1 = document.getElementById("text1");
        var msg = document.getElementById("msg");
        var p1 = t1.value;
        var p2 = Math.floor(Math.random()*3);    // プレイヤー２はコンピュータ
        str += "プレイヤー１ [          ]
        str += "コンピュータ [          ]
        var hantei = [          ]
        switch ( hantei ){
            case 0 : str += "あいこ";              break;
            case 1 : str += "プレイヤー１の勝ち"; break;
            case 2 : str += "コンピュータの勝ち"; break;
        }
        msg.innerHTML = str;
    }
</script>
</head>
<body >
じゃんけん　グー(0)　チョキ(1)　パー(2)<br>
プレイヤー１<input type ="text" id="text1" size=2>
<input type ="button" value="対戦" onClick="battle()" ><p>
<div id="msg"> じゃんけん対戦 </div>
</body>
</html>
```



JavaScript a8-03 自分の手は、画像をマウスで選択

jsa8-3.html

```
<html>
<head>
<meta content="text/html; charset=utf-8" http-equiv="Content-Type">
<title> じゃんけん対戦 </title>
<script type="text/javascript">

    function over(obj)
    {
        [                                  ]
    }

    function out(obj)
    {
        [                                  ]
    }

    function battle(obj)
    {
        var jan = ["グー","チョキ","パー"],str="";
        var jan_img = ["jan_goo.gif","jan_cyoki.gif","jan_paa.gif"];

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

        var p1 = [                         ] //プレイヤー１はマウスで選択した画像ＩＤ
        var p2 = [                         ] // プレイヤー２はコンピュータ

        str += "プレイヤー１ [                                  ]
        str += "コンピュータ [                                  ]

        var hantei = [                                  ]    //  勝敗判定
        switch ( hantei ){
            case 0 : str += "あいこ";              break;
            case 1 : str += "プレイヤー１の勝ち"; break;
            case 2 : str += "コンピュータの勝ち"; break;
        }
        msg.innerHTML = str;
    }

</script>
</head>
<body >
<img id="0" src="img/jan_goo.gif"   width=100 onMouseOver ="over(this)"
                                    onMouseOut="out(this)" onClick="battle(this)">
<img id="1" src="img/jan_cyoki.gif" width=100 onMouseOver ="over(this)"
                                    onMouseOut="out(this)" onClick="battle(this)">
<img id="2" src="img/jan_paa.gif"   width=100 onMouseOver ="over(this)"
                                    onMouseOut="out(this)" onClick="battle(this)">
<div id="msg"> じゃんけん対戦 </div>
</body>
</html>
```

インクルードファイル (m18a00-40.js)

```
var NAME = [    ];              // 名前用の配列

                        // それぞれのじゃんけん関数をインクルード

document.write("<script type='text/javascript' src='h28c00.js'></script>");
document.write("<script type='text/javascript' src='h28c01.js'></script>");
document.write("<script type='text/javascript' src='j2802.js'></script>");
・・・
document.write("<script type='text/javascript' src='h28c38.js'></script>");
document.write("<script type='text/javascript' src='h28c39.js'></script>");
document.write("<script type='text/javascript' src='h28c40.js'></script>");
```

イメージファイル (m18a____.png)

対戦中に表示する自分の写真、または好きなキャラクター
などを次の形式で作成、保存する。

サイズ　：　１５０ｐｘ　×　１５０ｐｘ（正方形）

ファイル形式　：　ＰＮＧ

ファイル名は　：　ｍ１８ａ＿＿＿＿．ＰＮＧ

名簿番号２ケタ

保存先　：　img フォルダ内

フォルダの構成

jyanken.html　メインプログラム
m18a00-40.js　インクルードファイル
m18a00.js　対戦相手サンプル
m18a___.js　自分のプログラム
（名簿番号）
--- img フォルダ（イメージ）
　　m18a___.png　自分の画像
　　（名簿番号）
　　m18a00.png　サンプル画像

じゃんけん大会　提出課題

①自分のプログラムリストと
　実行結果（対戦相手は自由）を印刷
②じゃんけん大会の感想

} これで Ａ４サイズ１枚

③プレゼン資料をＡ４用紙１枚に収まるようにして印刷

自分のプログラム と 対戦相手のプログラム

（自分で入力し、ベースプログラムと同じフォルダへ保存する）

ファイル名は「ｍ１８ａ＿＿＿＿．ｊｓ」とする。

名簿番号２ケタ

作成する関数は　グー (0)　チョキ (1)　パー (2) の
いずれかの整数値を返すものとする。

自分の例 (m18a41.js)　４１番の場合

```
NAME[41] =" 永遠チョキ ";
function m18a41()     // ずっとチョキ
{
    var jan;
    jan = 1;
    return jan;
}
```

関数の戻り値

グー	0
チョキ	1
パー	2

対戦相手の例 (m18a00.js)

```
NAME[00] = " 只野　乱数 ";
function m18a00()    // 乱数
{
    var jan;
    jan = Math.floor(Math.random()*3);
    return jan;
}
```

じゃんけんのアルゴリズム

３－１　　番　氏名

アルゴリズムのタイトル

※必ず、2つ以上考えて、調整後に他の人と同じにならないアイデアを採用する。

アルゴリズムのタイトル

アルゴリズムのタイトル

じゃんけんアルゴリズム

プログラム例

・関数名は「ｈ２８ｃ」＋自分の名簿番号２ケタ　h28c01 h28c20 h28c32

・何番と何番を対戦させるのかはテキストボックスに入力

・２人が出した手は配列 DAT[名簿番号][回数] に記録されていく

・対戦相手の番号は　AITE[自分の名簿番号]　で取得する

ベースプログラムでグローバル宣言されている変数（大文字）

（自分のプログラム関数の中で参照できる）

NAME []	**名前**	NAME [名簿番号]
DAT [] []	**手の履歴**	DAT [名簿番号] [回目]　２次元配列
AITE []	**対戦相手**	AITE [名簿番号]

例１　偶数回はグー、それ以外は乱数

```
NAME[50]="偶数はグー";    // 名簿番号の配列に名前の文字を設定

function m18a50()      // 関数名 h27c ＋自分の名簿番号２ケタ
{
    var jan;
    if(                    )      // もし回数が偶数回なら
        jan = 0 ;                 //    グー　を出す
    else                          // そうでなければ
        jan = Math.floor(Math.random()*3);  //      乱数で決定
    return jan;                   // 0 グー　1 チョキ　2 パー
}
```


例２　最初の１０回は乱数、次の１０回はグー、それ以外はチョキ

```
NAME[51]="長姫太郎";
function m18a51()
{
    var jan;
    if( CNT <= 10 )              //  CNT（回目）が１０以下なら
        jan = Math.floor(Math.random()*3);       //    乱数
    else if(                    ){ //  CNT が 11 - 20 なら
        jan = 0;                  //    グー
    }
    else {                        //  それ以外は
        jan = 1 ;                 //      チョキ
    }
    return jan;
}
```

例３　それぞれの手を次の確率で出す　グー（５０％）　チョキ（３０％）　パー（２０％）

```
NAME[52]="グー大好き";
function m18a52()
{
    var jan;
    var r =                                    // 0 ～ 9 の乱数
    if( r <= 4 )              //  0 - 4 なら
        jan = 0;              //    グー
    else if( r <= 7 ){        //  5 - 7 なら
        jan = 1;              //    チョキ
    }
    else {                    //  それ以外 (8 - 9) は
        jan = 2 ;             //      チョキ
    }
    return jan;
}
```

例４　相手の前回の手を調べ、それに勝つ手を出す

```
NAME[ 47 ] =″直前に勝利″;
function m18a47()      //相手の一手前に出した手に勝つ手を出す
{
    var jan,aite_mae;
    aite_mae = [          ]// 相手の前回の手

    if( aite_mae == 0) jan = 2;            // グーならパー
    if( aite_mae == 1) jan = 0;
    if( aite_mae == 2) jan = 1;
    return jan;
}
```

例５　前回の勝敗を調べて、次に出す手を決める

```
NAME[ 48 ] =″前回の勝敗″;
function m18a48()          //前回の勝敗を調べて出す手を決める
{
    var jan;
    var aite_mae  = [          ]//  前回の相手
    var jibun_mae = [          ]//  前回の自分
    var hantei =( aite_mae - jibun_mae + 3) % 3;   //勝敗

    switch( hantei ){
        case 0 : jan = 0; break;  // あいこなら      グー
        case 1 : jan = 1; break;  // 自分の勝ちなら チョキ
        case 2 : jan = 2; break;  // 自分の負けなら パー
    }
    return jan;
}
```


例６　１回～前回、相手の一番少ない手を調べる（乱数対策？）

```
NAME[ 49 ] = ″乱数に勝？″;
function m18a49() //相手が今まで一番出していない手に勝つ手を出す
{
    var jan;

    var aite_te = [0,0,0];          // 相手の手カウント用配列
    for(var i=1 ; i <= CNT; i++ )
       aite_te[          ] ++;  // 手をカウント

    var min = 0;                  // 仮に最少をグー(0)とする
    if( aite_te[ 1 ] < aite_te[ min ] ) min = 1; //最小はチョキ
    if([          ]) min = 2; //最小はパー

    if( min == 0) jan = 2;     // 最小がグー(0)ならパー(2)
    if( min == 1) jan = 0;
    if( min == 2) jan = 1;
    return jan;
}
```

変数の例　２番と３８番の対戦の場合

DAT[番号][回]　　対戦回 CNT　⇒　　**AITE[自分の番号]**

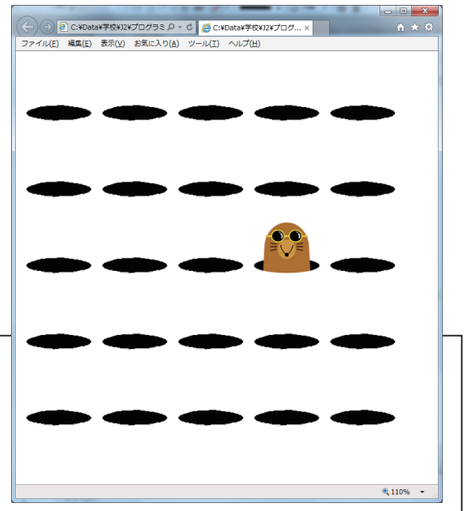
番号	1	2	3	4	5	6	7	8	9	197	198	199	200	AITE
0														
1														
2	0	2	2	1	0	0	1	2	1	1	1	0	2	38
3														
38	1	2	1	0	0	0	2	2	1	2	1	0	0	2
39														
40														

JavaScript による　ミニゲームベース①　もぐらたたき

JavaScript g1-01 もぐらたたきゲームもどきを作成する。最初に５×５の穴を表示させ、もぐらの画像をランダムに移動させる。

※必要な画像ファイルは共通フォルダからコピーすること

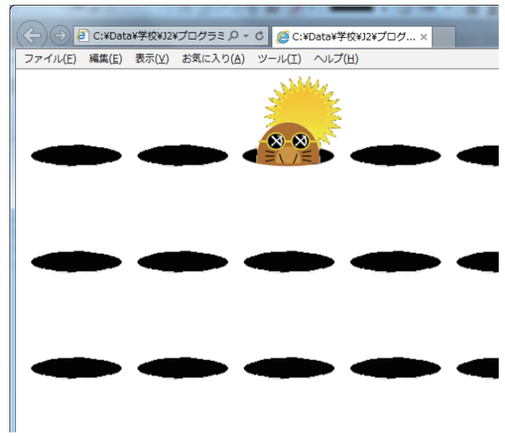
jsg1-1.html



```
<html>
<head>
<script type="text/JavaScript">
    var n=0;
    function mogura()
    {
        setTimeout("mogura()",1000);
        var mg = document.getElementById( n );      // 前回の出現位置
        mg.src = "img/mogura0.png";                 // 穴の画像
        n = Math.floor(Math.random()*25);           // 今回の出現位置（乱数）
        var mg = [                                ]
        mg.src = "img/mogura1.png";                 // もぐらの画像
    }
</script>
</head>
<body onLoad="mogura()">
    <script type="text/JavaScript">
        for(var i=0;i<25;i++){
            document.write("<img id='" + i + "' src='img/mogura0.png'>");
            if( i%5 == 4 ) document.write("<br>");
        }
    </script>
</body>
</html>
```


JavaScript g1-02 もぐらの画像をクリックしたら、叩かれた画像に変更させる。

jsg1-2.html



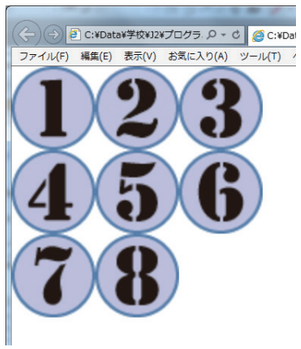
```
<html>
<head>
<script type="text/JavaScript">
     var n=0,img0_path;

     function mogura()
     {
        ・・・省略
    }

    function tataku(obj)
    {
        if(    obj.src != img0_path    )
            obj.src = "img/mogura2.png";
    }
                [                    ]
</script>
</head>
<body onLoad="mogura()">
    <script type="text/JavaScript">
        for(var i=0;i<25;i++){
            document.write("<img id='" + i + "' src='img/mogura0.png' onClick='tataku(this)'>");
            if( i%5 == 4 ) document.write("<br>");
        }
        img0_path = (document.getElementById("0" )).src;  // 画像フルパス
    </script>
</body>
</html>
```

JavaScriptによる　ミニゲームベース②　スライドパズル

JavaScript g2-01　スライドパズルを作成する。（画像８枚版）

jsg2-1.html

```
<html>
<head>
  <script type="text/javascript">
    var aki;    // 空のピース
    function check(obj) {// クリックされたピースを移動
        var x,y,no;
        x= obj.offsetLeft/100;   // オブジェクトの位置
        y= obj.offsetTop/100;
        no = y*3+x;                  // オブジェクトの位置番号
                        //      上   下   左   右 のいずれかが空きなら移動
        if(                                                    ){
            obj.style.left = (aki%3)*100;
            obj.style.top  = Math.floor(aki/3)*100;
                                  // 移動元が空ピースになる
        }
    }
  </script>
</head>
<body>
<script type="text/javascript">
    var i,j,id;
    var img = new Array("num1.gif","num2.gif", ・・・,"num8.gif");
    for(i=0;i<3;i++){          // ピースに使う８個の画像
      for(j=0;j<3;j++){
         id = i*3+j;
         if(id != 8){
           document.write("<img id='" + id + "' src='img/" + img[id] + "' width=100 " +
               "style='" +  "position: absolute; left:" + j*100 +" ;top:" + i*100 + ";' ¥
                                onClick ='check(this)'>");
         }
      }
    }
    aki = 8;  // 最初の空きピースは８番
 </script>
</body>
</html>
```



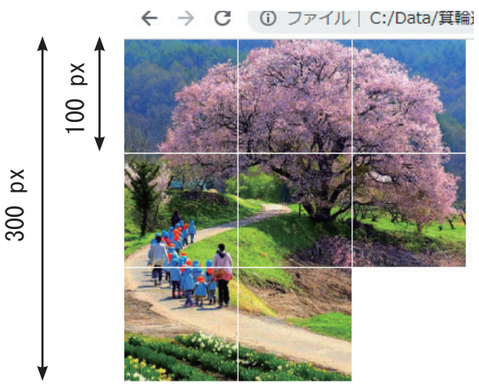
0	1	2
3	4	5
6	7	8

IDと位置の関係

JavaScript g2-02　スライドパズルを作成する。（１枚の画像をクリッピング Ver）

jsg2-2.html

```
<html>
<head>
  <script type="text/javascript">
     var aki;                  // 空のピース
    function check(obj) {   // クリックされた位置のピースを動かす
        var x,y,no,id;
        id = obj.id;
        x = obj.offsetLeft+(id%3)*100;        // ID から座標を計算
        y = obj.offsetTop+ Math.floor(id/3)*100;
        x = x/100; y = y/100;
        no = y*3+x;    //     オブジェクトの位置番号
                       //      上   下   左   右 のいずれかが空きなら移動
        if(                                                    ){
            obj.style.left = (aki%3)*100 - (id%3)*100;
            obj.style.top  = Math.floor(aki/3)*100 - Math.floor(id/3)*100;
                                  // 移動元が空ピースになる
        }
    }
  </script>
</head>
<body>
<script type="text/javascript">
    var i,j,id,x1,y1,x2,y2;
    for(i=0;i<3;i++){
      for(j=0;j<3;j++){
        id = i*3+j;
        x1=j*100;      y1=i*100;
        x2=j*100+99; y2=i*100+99;  // クリップ領域の計算
        if(id != 8){
          document.write("<img id='" + id + "' src='img/sakura.jpg' ¥
             width=300  style='" +  "position: absolute; left:0 ;top:0; ¥
             clip: rect(" + y1  + "px " + x2 + "px " + y2 + "px " + x1 + "px);' ¥
                                        onClick ='check(this)'>");
        }
      }
    }
    aki = 8;    // 最初の空きピースは８番
</script>
</body>
</html>
```

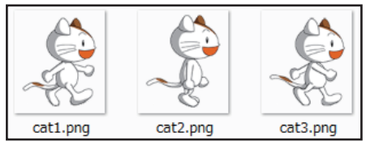


JavaScriptによる　ミニゲームベース③　横スクロールゲーム

JavaScript g3-01

キャラクタの画像（アニメーションＧＩＦ）を表示させ
背景画像を横スクロールさせるスクリプトを作成せよ。

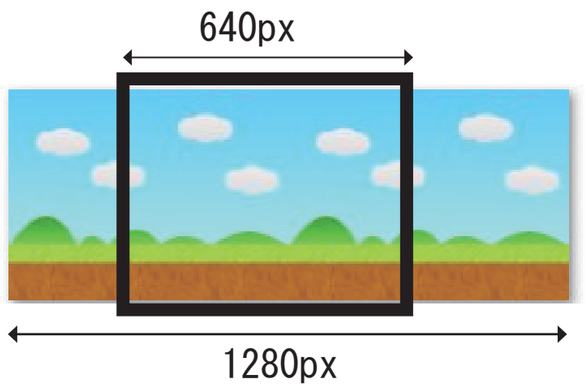
※必要な画像ファイルは共通フォルダからコピーすること



アニメーションＧＩＦファイル
cat_anime.gif

jsg3-3.html

```
<html>
<head>
<script type="text/JavaScript">
    var n=1,x= 0,y= 0,dx=2,dy=0;

    function move()
    {
        setTimeout("move()",20);

        var back = document.getElementById("back");
        if(  x < -640 )
            x = [          ]
        else
            x = x - dx;
        back.style.left = x;
        back.style.top  = y;
    }
</script>
</head>
<body>
    <img id="back" src="img/back1.png" style="position:absolute;">
    <img id="cat" src="img/cat_anime.gif" style="position:absolute;left:150;top:260;"">
    <script type="text/JavaScript">
        resizeTo(660,600);      // ウィンドウサイズ指定
        move();                 // 背景画像移動
    </script>
</body>
</html>
```

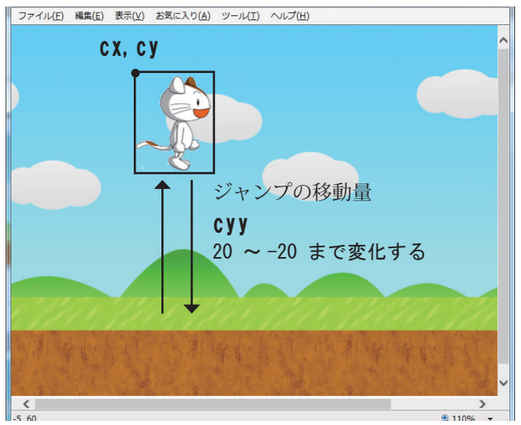


JavaScript g3-02

キー入力でキャラクタがジャンプするスクリプトを追加せよ。

jsg3-4.html

```
<html>
<head>
<script type="text/JavaScript">
    var n=1,x= 0,y= 0,dx=2,dy=0,cx=150,cy=260,cyy=20;

    function move()
    {
        ・・・省略

    function keydown(event)
    {
        var key = event.keyCode;
        if(  [          ]  )  [          ]
    }

    function jump()
    {
        var cat = document.getElementById("cat");
        cy = cy - cyy;
        cat.style.top  = cy;
        cyy--;
        if( cy < 260 ) {
            setTimeout("jump()",20);
        }
        else{
            cyy = 20;
        }
    }

</script>
</head>
<body onLoad="move()" onKeydown="keydown(event)">
    <img id="back" src="img/back1.png" style="position:absolute;">
    <img id="cat" src="img/cat_anime.gif" style="position:absolute;left:150;top:260;"">
    <script type="text/JavaScript">
        resizeTo(660,600);                  // ウィンドウサイズ指定
    </script>
</body>
</html>
```



キー入力イベント

押したキーのキーコードを取得するには、イベント引数である event.keyCode プロパティで取得できます。取得できるイベントは keyup、keydown、keypress です。

onkeydown　…　キーが押し下げられた時
onkeyup　…　キーを押した後、離された時
onkeypress …　キーを押している最中

例えばキー「A」が押されたときに 関数 func_a()
を呼び出す場合は例のようになります。

```
function keydown(event)
{
    var key = event.keyCode;
    if(   key == 65   )
        func_a();
}

・・・
・・

<body onLoad="move()" onKeydown="keydown(event)">

     ・・・
     ・・・

</body>
```


主なキーコード（１０進数）

キー	keyCode
0	48
1	49
2	50
3	51
4	52
5	53
6	54
7	55
8	56
9	57
A	65
B	66
C	67
D	68
E	69
F	70
G	71
H	72
I	73
J	74
K	75
L	76
M	77
N	78
O	79
P	80
Q	81
R	82
S	83
T	84
U	85
V	86
W	87
X	88
Y	89
Z	90

キー	keyCode
a	97
b	98
c	99
d	100
e	101
f	102
g	103
h	104
i	105
j	106
k	107
l	108
m	109
n	110
o	111
p	112
q	113
r	114
s	115
t	116
u	117
v	118
w	119
x	120
y	121
z	122

キー	keyCode
BackSpace	8
Tab	9
Clear	12
Enter	13
Command	15
Shift	16
Ctrl	17
Alt	18
CapsLock	20
Esc	27
スペースバー	32
PageUp	33
PageDown	34
End	35
Home	36
←（左矢印）	37
↑（上矢印）	38
→（右矢印）	39
↓（下矢印）	40
Insert	45
Delete	46
NumLock	144
, <	188
. >	190
/ ?	191
[{	219
¥ |	220
] }	221

※ブラウザにより
異なる場合が
あります。

JavaScriptによる　ミニゲームベース③　シューティング

JavaScript g4-01

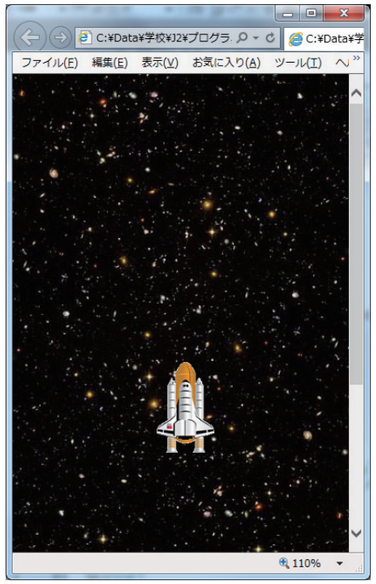
プリントを参考に、画像が無限に縦にスクロールするスクリプトを作成せよ。

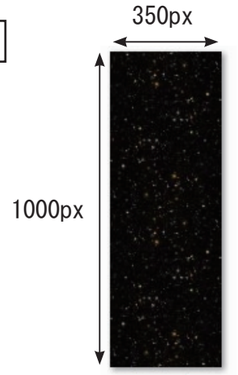
※必要な画像ファイルは共通フォルダからコピーすること

syatoru1.png

jsg4-1.html

```
<html>
<head>
<script type="text/javascript">
    var x = 0,y = -500;   // 背景画像の位置
    var dy=2;             // 背景画像の動き幅
    function move()
    {
        setTimeout("move()",20);
        var back = document.getElementById("back");
        if(          )
            y =          
        else
            y = y +  dy;
        back.style.left = x;
        back.style.top  = y;
    }
</script>
</head>
<body  onLoad="move()">
    <img id="back" src="img/space.jpg" style="position:absolute;">
    <img id="ship" src="img/syatoru1.png" width=60 style="position:absolute;left:150;top:300;">
    <script type="text/javascript">
        resizeTo(380,600);                  // ブラウザのウィンドウサイズ指定
        document.body.style.overflow = "hidden";// スクロールバー非表示
    </script>
</body>
</html>
```



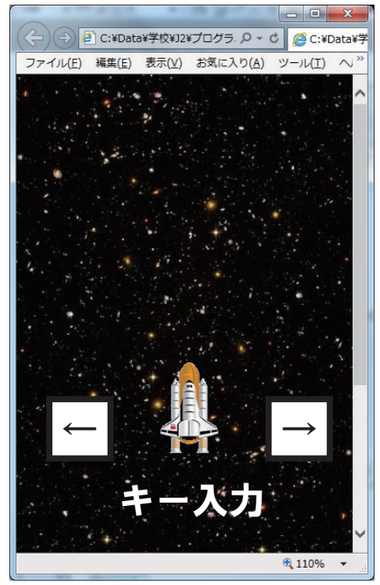


JavaScript g4-02

キー入力で画像（宇宙船）を左右に移動するスクリプトを追加せよ。キーコードについては別プリント参照。

jsg4-2.html

```
<html>
<head>
<script type="text/javascript">
    var x = 0,y = -500;
    var dx=10,dy=2;
    var sx = 150 ,  sy = 300;    // 宇宙船画像の位置

    function move()
    {
        ・・・省略
    }

    function keydown(event)
    {
        var ship = document.getElementById("ship");
        var key =                    
        if(  key ==          ) sx =          
        if(  key ==          ) sx =          
        ship.style.left = sx;
        ship.style.top  = sy;
    }
</script>
</head>
<body  onLoad="move()" onKeydown="keydown(event)" >     キーイベント処理
    <img id="back" src="img/space.jpg" style="position:absolute;">
    <img id="ship" src="img/syatoru1.png" width=60 style="position:absolute;left:150;top:300;">
    <script type="text/javascript">
        resizeTo(380,600);                          // ウィンドウサイズ指定
        document.body.style.overflow = "hidden";// スクロールバー非表示
    </script>
</body>
</html>
```



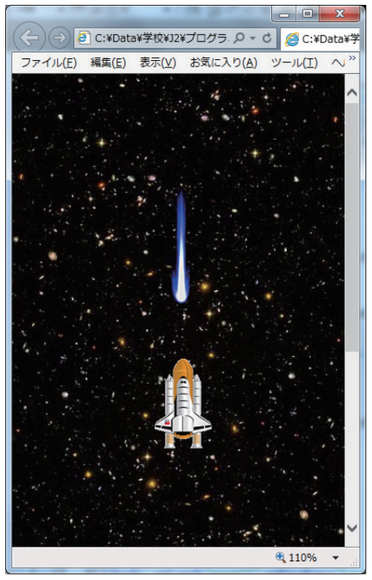
JavaScript g4-03

上方から流星の画像が飛行してくるスクリプトを追加する。
ウインド下方まで移動したら再び上方から出現するようにせよ。
更に流星の出現する横位置を乱数で変化させよ。

jsg4-3.html

```
<html>
<head>
<script type="text/javascript">
    var x=0, y= -500, dx=10, dy=2, sx=150, sy=300, bx=150, by= -100;
                                          // 流星 (beem.png) の位置
    function move()
    {
        setTimeout("move()",20);
        ・・・省略 (背景を動かす部分)

        var beem = document.getElementById("beem");
        by = [          ]
        if( by > 600 ) {
            by = [          ]
            bx = [                    ]
            beem.src ="img/beem.png";
        }
        beem.style.left = bx;
        beem.style.top  = by;
    }
    function keydown(event)
    {
      ・・・省略 (キー入力処理)
    }
</script>
</head>
<body  onLoad="move()" onKeydown="keydown(event)" >
    <img id="back" src="img/space.jpg" style="position:absolute;">
    <img id="ship" src="img/syatorul.png" width=60 style="position:absolute;left:150;top:300;">
    <img id="beem" src="img/beem.png" style="position:absolute;">
    <script type="text/javascript">
        resizeTo(380,600);                    // ウィンドサイズ指定
        document.body.style.overflow = "hidden";// スクロールバー非表示
</script>
・・・
```

※

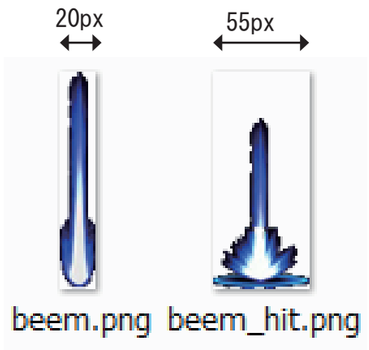


JavaScript g4-04

流星が宇宙船に当たったら「流星の画像」を「爆発した画像」に
置き換えるスクリプトを追加せよ。流星と宇宙船の衝突判定は
簡易ラケットゲームの「js29-5.html」の関数を用いればよい。

jsg4-4.html

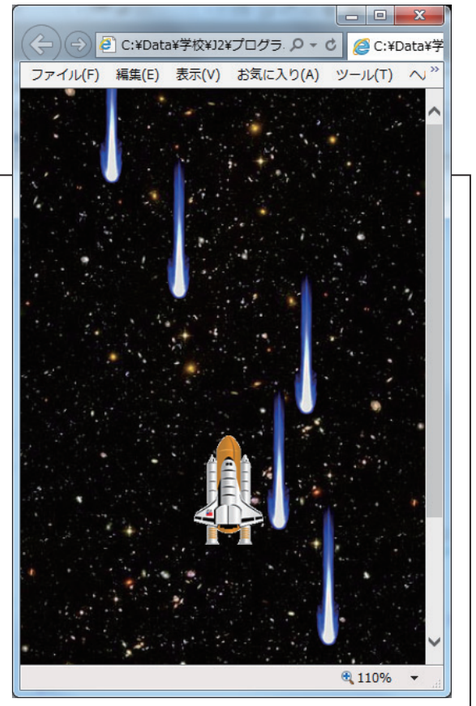
```
<html>
<head>
<script type="text/javascript">
    var x=0, y= -500, dx=10, dy=2, sx=150, sy=300, bx=150, by= -100;
    function move()
    {
        setTimeout("move()",20);
        ・・・省略 (背景を動かす部分)
        var ship = document.getElementById("ship");
        var beem = document.getElementById("beem");
        ・・・省略 (前ページの  ※ {  の部分が入る)

        if( [                                ] ){ // 衝突判定
            beem.src =  "img/beem_hit.png";
        }

        beem.style.left = bx - beem.width/2;  // 流星と爆発画像の中央揃え
        beem.style.top  = by;

    }
    function keydown(event)
    {
      ・・・省略 (キー入力処理)
    }
    function overlap( obj1 , obj2 )
    {
        ・・・省略 (衝突判定関数  ラケットゲームと同じ)
    }
</script>
</head>
<body  onLoad="move()" onKeydown="keydown(event)" >

    ・・・以下省略
```



発展課題

JavaScript g4-05

流星の数を配列を使用して増やすよう追加修正せよ。



jsg4-5.html

```
<html>
<head>
<script type="text/javascript">

    var x=0,y= -500,dx=10,dy=2,sx=150,sy=300;
    var bx = new Array(0,100,200,300,400);     // 流星用配列　初期位置
    var by = new Array(0,-100,-200,-300,-400);

    function move()
    {
        setTimeout("move()",20);

        var back = document.getElementById("back");
       if(y>=0 )
            y = -500;
        else
            y = y +  dy;
        back.style.left = x;
        back.style.top  = y;
        var ship = document.getElementById("ship");

        for(var i=0;i<5;i++){                          // 配列分 繰り返す
            var beem = document.getElementById( i );
            by[i] = by[i] + 10;
            if( by[i] > 600 ) {
                by[i] = -100;
                bx[i] = Math.floor(Math.random()*300);
                beem.src ="img/beem.png";
            }

            if( overlap( ship , beem) == 1 ){
                beem.src ="img/beem_hit.png";
            }

            beem.style.left = bx[i] - beem.width/2;
            beem.style.top  = by[i];
        }

    }
```

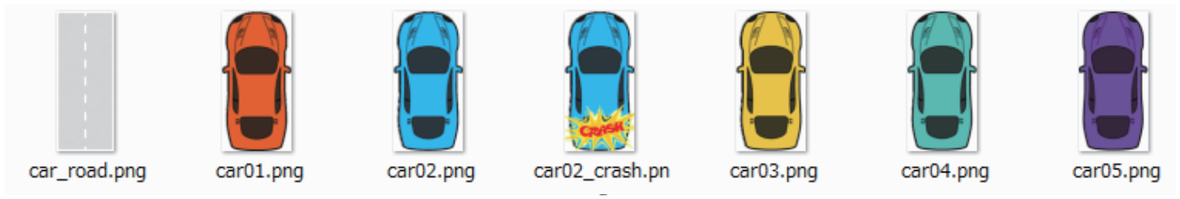


```
    function keydown(event)
    {
        var ship = document.getElementById("ship");
        var key = event.keyCode;
        if(key == 37 ) sx= sx-2;
        if(key == 39 ) sx= sx+2;
          ship.style.left = sx;
          ship.style.top  = sy;

    }

    function overlap(obj1,obj2)
    {
        var x1,y1,w1,h1,x2,y2,w2,h2;
        x1 = obj1.offsetLeft;    x2 = obj2.offsetLeft;
        y1 = obj1.offsetTop;     y2 = obj2.offsetTop;
        w1 = obj1.width;         w2 = obj2.width;
        h1 = obj1.height;        h2 = obj2.height;

        if(( x2 < x1+w1  &&  x2+w2 > x1 ) && (  y2 < y1+h1  &&  y1 < y2+h2 ) )
            return 1;
        else
            return 0;
    }

</script>
</head>
<body  onLoad="move()" onKeydown="keydown(event)" >
    <img id="back" src="img/space.jpg" style="position:absolute;">
    <img id="ship" src="img/syatoru1.png" width=60 style="position:absolute;left:150;top:300;">
    <script type="text/javascript">
        resizeTo(380,600);
        for(var i=0;i<5;i++){              // 流星を５つ表示
            document.write("<img id='" + i + "' src='img/beem.png' style='position:absolute;'>");
        }
    </script>
</body>
</html>
```

発展課題２

JavaScript g4-06

画像を車と道路に置き換えてドライブゲーム風にしてみる
画像ファイルは img/car フォルダにある。

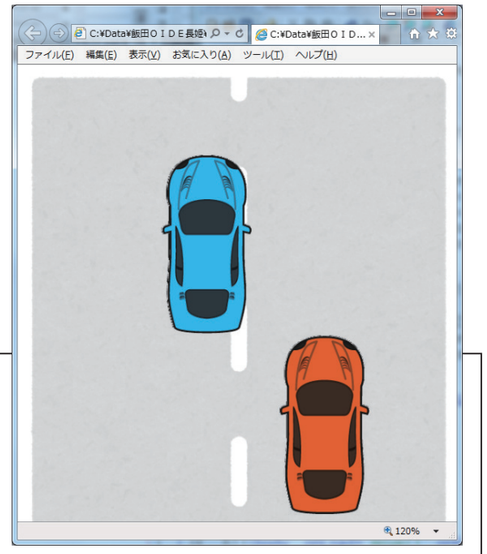


jsg4-6.html

jsg4-6.html を改変

プログラミング技術 WEB提出

```
<html>
<head>
<script type="text/javascript">
    var x = 0,y = -500;
    var dx=10,dy=10,bx=150,by= -100;
    var sx = 150 ,  sy = 300;    // 自動車の位置

    function move()
    {
        setTimeout("move()",50);
        var back = document.getElementById("back");
        if(   y>=0    )
            y = -500;
        else
            y = y +  dy;
        back.style.left = x;
        back.style.top  = y;

        var ship = document.getElementById("ship");
          var beem = document.getElementById("beem");
          by = by +10;
          if( by > 600 ) {
            by = -200;
            bx = Math.floor(Math.random()*300);
            beem.src ="img/car/car02.png";
        }
        beem.style.left = bx;
        beem.style.top  = by;

          if( overlap( ship , beem) == 1){    // 衝突判定
               beem.src = "img/car/car02_crash.png";
          }
    }
```


```
function keydown(event)
{
    var ship = document.getElementById("ship");
    var key = event.keyCode;
    if(  key == 37) sx = sx-10;
    if(  key == 39) sx = sx+10;
    ship.style.left = sx;
    ship.style.top  = sy;
}

function overlap( obj1 , obj2 )
{
    var x1,y1,w1,h1,x2,y2,w2,h2;
    x1 = obj1.offsetLeft;    x2 = obj2.offsetLeft;
    y1 = obj1.offsetTop;     y2 = obj2.offsetTop;
    w1 = obj1.width;         w2 = obj2.width;
    h1 = obj1.height;        h2 = obj2.height;
    if(( x2 < x1+w1  &&  x2+w2 > x1 ) && (  y2 < y1+h1  &&  y1 < y2+h2 ) )
        return 1;  // ２つのオブジェクトが重なっていれば 1 を返す
    else
        return 0;
}


</script>
</head>
<body  onLoad="move()" onKeydown="keydown(event)" >
    <img id="back" src="img/car/car_road.png" style="position:absolute;">
    <img id="ship" src="img/car/car01.png" width=100 style="position:absolute;left:150;top:300;">
    <img id="beem" src="img/car/car02.png" width=100 style="position:absolute;">
    <script type="text/javascript">
        resizeTo(500,600);                          // ウィンドサイズ指定
       document.body.style.overflow = "hidden";     // スクロールバー非表示
    </script>
</body>
```

JavaScript による　ミニゲームベース⑤　落ちモノゲーム

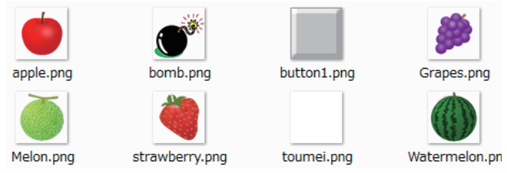
JavaScript g5-01

「落ちものゲーム」もどきを作る。
ゲーム盤面は横１０×縦１５とし、２次元配列に各コマのデータを格納する。各コマの画像ファイルは配列に設定し管理する。

jsg5-1.html

```
<html>
<head>
<script type="text/JavaScript">
    var gazou= new Array("toumei.png","apple.png","melon.png","grapes.png","watermelon.png","button1.png");
    var n,x,y;
    var dat = [ [ 5,0,0,1,2,3,4,0,0,5 ] ,
                [ 5,0,0,0,0,0,0,0,0,5 ] ,
                [ 5,0,0,0,0,0,0,0,0,5 ] ,
                [ 5,0,0,0,0,0,0,0,0,5 ] ,
                [ 5,0,0,0,0,0,0,0,0,5 ] ,
                [ 5,0,0,0,0,0,0,0,0,5 ] ,
                [ 5,0,0,0,0,0,0,0,0,5 ] ,
                [ 5,0,0,0,0,0,0,0,0,5 ] ,
                [ 5,0,0,0,0,0,0,0,0,5 ] ,
                [ 5,0,0,0,0,0,0,0,0,5 ] ,
                [ 5,0,0,0,0,0,0,0,0,5 ] ,
                [ 5,0,0,0,0,0,0,0,0,5 ] ,
                [ 5,0,0,0,0,0,0,0,0,5 ] ,
                [ 5,0,0,0,0,0,0,0,0,5 ] ,
                [ 5,5,5,5,5,5,5,5,5,5 ] ,
              ];

    function disp()  // 配列データ表示関数
    {
        for(var dy=0;dy<15;dy++){
            for(var dx=0;dx<10;dx++){
                var id = dy*10+dx;
                var img = document.getElementById( id );
                img.style.left = dx*48;
                img.style.top  = dy*48;
                img.src="img/" + gazou[ dat[dy][dx] ];
            }
        }
    }
</script>
</head>
<body onLoad="disp()">
    <img src="img/mori2.jpg" style=" position:absolute;left:0;top:0;"> 背景画像
    <script type="text/JavaScript">
        for(var i=0;i<150;i++){
            document.write("<img id='" + i + "' src='img/toumei.png' width=48 style='position:absolute;'>");
        }
    </script>
</body>
</html>
```

盤面のデータ
（2 次元配列）

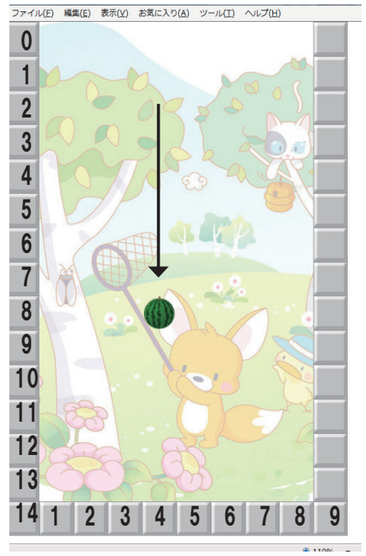


JavaScript g5-02

落とす画像番号と横位置（ｘ）を乱数で設定し、
上から下へ一コマづつ落ちていくスクリプトを追加する。
何回かリロードして動作を確認する。

jsg5-2.html

```
<html>
<head>
<script type="text/JavaScript">
    var gazou= new Array("toumei.png","apple.png","melon.png","grapes.png","watermelon.png","button1.png");
    var n,x,y;
    var dat = [ [ 5,0,0,0,0,0,0,0,0,5 ] ,
                        ・・・省略
                  [ 5,5,5,5,5,5,5,5,5,5 ] ,
                ];

    function disp()
    {
        ・・・省略
    }

    function start()
    {
        n = [          ]  // 画像番号
        x = Math.floor(Math.random()*8)+1;  // 横位置
        y = 0;        // 縦位置
        drop();
    }

    function drop()              // 自動で落ちる関数
    {
        setTimeout("[      ]", 200);
        dat[y][x] = 0;           // 現在位置を消去
        y[      ]                // １コマ下へ移動
        dat[y][x] = [      ]     // 画像番号をセット
        disp();                  // 表示関数呼び出し
    }
</script>
</head>
<body onLoad="start()">
    <img src="img/mori2.jpg" style=" position:absolute;left:0;top:0;" >
    <script type="text/JavaScript">
        for(var i=0;i<150;i++){
            document.write("<img id='" + i + "' src='img/toumei.png' width=48 style='position:absolute;'>");
        }
    </script>
</body>
</html>
```

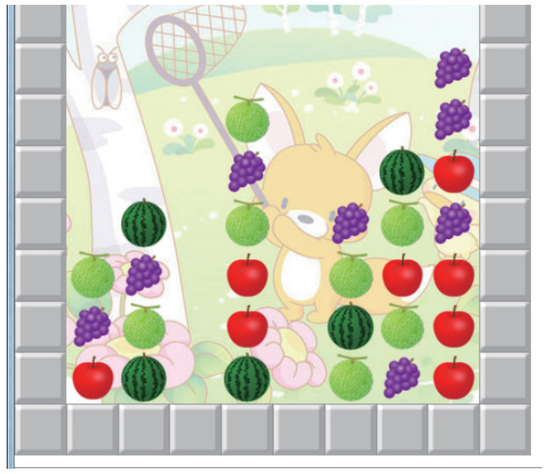


JavaScript g5-03

落ちていく先にすでに画像があったらそこで止め、新たな画像を上から落としていくスクリプトを追加する。
下の壁も画像の一つとして考え処理する。

jsg5-3.html

```
<html>
<head>
<script type="text/JavaScript">
    var gazou= new Array("toumei.png","apple.png","melon.png","grapes.png","watermelon.png","button1.png");
    var n,x,y;

    var dat = [ [ 5,0,0,0,0,0,0,0,0,5 ] ,
                    ・・・省略
                [ 5,5,5,5,5,5,5,5,5,5 ] ,
              ];

    function disp()
    {
          ・・・省略
    }

    function start()
    {
          ・・・省略
    }

    function drop()
    {
        if(                              ){
            setTimeout("drop()",300);
            dat[y][x] = 0;
            y++;
            dat[y][x] = n;
            disp();
        }
        else{
            start();
        }
    }

</script>
</head>
<body onLoad="start()">

      ・・・以下、省略
```



6	5	0	0	0	0	0	0	0	0	5
7	5	0	0	0	0	0	0	0	3	5
8	5	0	0	0	2	0	0	0	3	5
9	5	0	0	0	3	0	0	4	1	5
10	5	0	4	0	2	0	3	2	3	5
11	5	2	3	0	1	0	2	1	1	5
12	5	3	2	0	1	0	4	2	1	5
13	5	1	4	0	4	0	2	3	1	5
14	5	5	5	5	5	5	5	5	5	5
	0	1	2	3	4	5	6	7	8	9

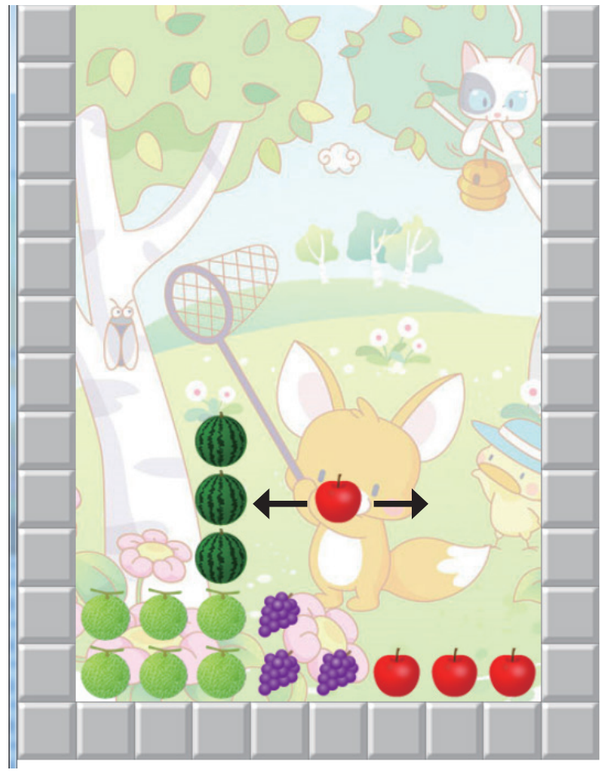
配列データ

JavaScript g5-04

左右（←→）のキー入力で落ちていく画像を移動するスクリプトを追加する。

jsg5-4.html

```
<html>
<head>
<script type="text/JavaScript">


    ・・・  ここまで省略

    function drop()
    {
        ・・・省略
    }

    function keyin(event)
    {
        var key = event.keyCode;
        dat[y][x] =
        if(key == 37 )
        if(key == 39 )
        dat[y][x] =
        disp();
    }

</script>

</head>
<body onLoad="start()" onKeydown="keyin(event)">
    <img src="img/mori2.jpg" style="z-index: -999; position:absolute;left:0;top:0;" >
    <script type="text/JavaScript">
        for(var i=0;i<150;i++){
            document.write("<img id='" + i + "' src='img/toumei.png' width=48 style='position:absolute;'>");
        }
    </script>
</body>
</html>
```



JavaScriptによる　ミニゲームベース⑥　ダンジョンRPG

JavaScript g6-01

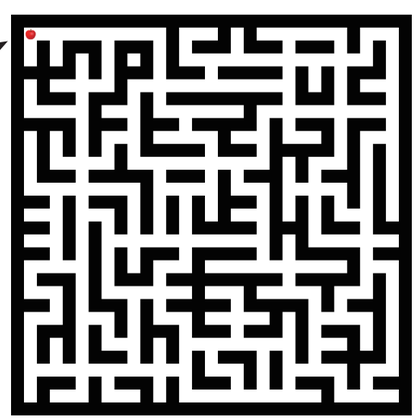
「迷路ゲーム」キャラクタが迷路の中を移動するゲームベース。
迷路のプログラムを元に追加・修正して作成する。

キャラクタが移動するたびに再描画するため、
表示方法を innerHTML を使用したものに変更。

jsg6-1.html

```
<html>
<head>
<meta charset="UTF-8">
<script type="text/javascript">
    var W=32,H=32,CX,CY;  //キャラクタ位置
    var maze = [];          ・・・以下省略
     ・・・
    function maze_disp2()
    {
      var msg = document.getElementById("msg");
      var str="";
      for(var y=0;y<=H;y++){
        for(var x=0; x<=W;x++){
            switch( maze[y][x] ){                    // １マス 20px で表示
                case 0 : str += "<img src='img/white.png' width=20>"; break;
                case 1 : str += "<img src='img/black.png' width=20>"; break;
                case 2 : str += "<img src='img/red.png'   width=20>"; break;
                case 3 : str += "<img src='img/apple.png' width=20>"; break;
            }    // キャラクタは３で示す          キャラクタ画像を表示
        }
        str += "<br>";
      }
      msg.innerHTML = str;       // 表示内容を書き換える
    }
</script>
<body>
<div id="msg"> ここに迷路を描画 </div>   // 移動するたびに再描画するため領域を指定
<script type="text/javascript">
    var x,y;

    for(y=0;y<=H;y++){
        maze[y] = [];
        for(x=0; x<=W;x++){
            maze[y][x] = 0;
        }
    }
    for(var y=1; y<=H-1; y++ ){
        for(var x=1; x<=W-1; x++ ){
            if(x==1 || x==W-1 || y==1 || y==H-1)
                maze[y][x] = 1;
        }
    }
    CX=2 , CY=2 ;        // キャラクタ位置
    maze[CY][CX] = 3;    // 配列にキャラクタ (3) を設定
    boutaosi();          // 迷路生成（アルゴリズム選択）
    maze_disp2();        // 迷路表示関数（再描画版）
</script>
</body>
</html>
```

キャラクタ

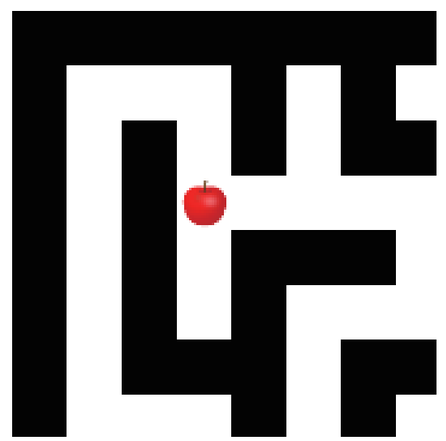


JavaScript g6-02

キーボード入力でキャラクタを移動する。

jsg6-2.html

```
<html>
<head>
<meta charset="UTF-8">
<script type="text/javascript">
    var W=32,H=32,CX,CY;
    var maze = [];          ・・・以下省略
     ・・・
    function maze_disp2()
    {
     ・・・省略
    }

    function keydown(event) // キー入力処理関数
    {
        var key = event.keyCode;
        maze[CY][CX] = 0;    // 現在位置のキャラクタデータを消す
        switch(key){     // 進みたい方向に壁がなければ移動する
            case 38 : if( maze[CY-1][CX  ] == 0 ) CY--;     break;  // ↑ キー
            case 40 : if(                       ) CY++;     break;  // ↓ キー
            case 37 : if(                       ) CX--;     break;  // ← キー
            case 39 : if(                       ) CX++;     break;  // → キー
        }
        maze[CY][CX] = 3;    // 移動後の位置にキャラクタデータを設定
        maze_disp2();
    }
</script>
<body onkeydown="keydown(event)">   // キー入力イベントを設定
<div id="msg"> ここに迷路を描画 </div>   // 移動するたびに再描画するため領域を指定
<script type="text/javascript">
    var x,y;

    for(y=0;y<=H;y++){
        maze[y] = [];
        for(x=0; x<=W;x++){
            maze[y][x] = 0;
        }
    }
    for(var y=1; y<=H-1; y++ ){
        for(var x=1; x<=W-1; x++ ){
            if(x==1 || x==W-1 || y==1 || y==H-1)
                maze[y][x] = 1;
        }
    }
    CX=2 , CY=2 ;
    maze[CY][CX] = 3;
    boutaosi();
    maze_disp2();
</script>
</body>
</html>
```

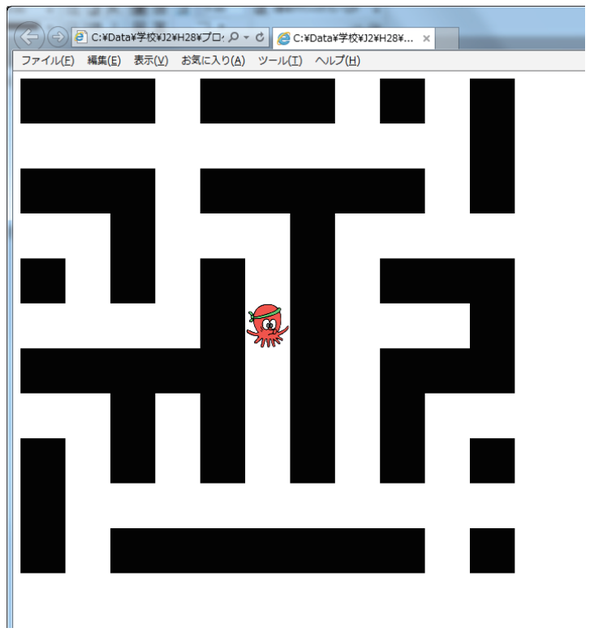


JavaScript g6-03

迷路を大きく表示し、一部分だけクリッピングして表示する。

jsg6-3.html

```
<html>
<head>
<meta charset="UTF-8">
<script type="text/javascript">
    var W=32,H=32,CX,CY;
    var maze = [];        ・・・以下省略
     ・・・
    function maze_disp2()
    {
      var msg = document.getElementById("msg");
      var WW=5;         // キャラクタのいる位置から上下左右 WW だけを表示
      var str="";
      for( var y = CY-WW;  y <= CY+WW;  y++ ){
        for( var x = CX-WW;  x <= CX+WW;  x++ ){
            if( y<0  ||  x<0  ||  y>H  ||  x>W  ){            // 迷路外側の処理
                str += "<img src='img/white.png' width=50>";
            }
            else{
                switch( maze[y][x] ){                    // １マス 50px で表示
                    case 0 : str += "<img src='img/white.png' width=50>"; break;
                    case 1 : str += "<img src='img/black.png' width=50>"; break;
                    case 2 : str += "<img src='img/red.png'   width=50>"; break;
                    case 3 : str += "<img src='img/たこキャラ.png' width=50>"; break;
                }
            }
        }
        str += "<br>";
      }
       msg.innerHTML = str;
    }
    function keydown(event) // キー入力処理関数
    {
      ・・・省略
    }
<body onkeydown="keydown(event)">
<div id="msg"> ここに迷路を描画 </div>
<script type="text/javascript">
    var x,y;

    for(y=0;y<=H;y++){
        maze[y] = [];
        for(x=0; x<=W;x++){
            maze[y][x] = 0;
        }
    }
    for(var y=1; y<=H-1; y++ ){
        for(var x=1; x<=W-1; x++ ){
            if(x==1 || x==W-1 || y==1 || y==H-1)
                maze[y][x] = 1;
        }
    }
     CX=2 , CY=2 ;
     maze[CY][CX] = 3;
     boutaosi();
     maze_disp2();
</script>
</body>
</html>
```



JavaScript g6-04

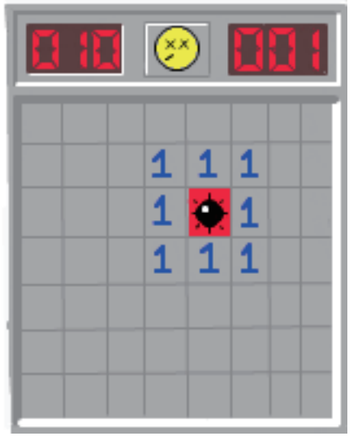
キャラクタの表示を進む方向によって変える。
img/work フォルダ内に下記アニメＧＩＦ画像があるので
自身のフォルダへ walk フォルダごとコピーする。

jsg6-4.html

```
・・・省略

    var walk = ["back.gif","front.gif","left.gif","right.gif"]; // 画像ファイル
    var wn=1;

    function maze_disp2()
    {
       ・・・省略
                    case 3 : str += "<img src='img/walk/" + walk[ wn ] + "' width=50>";break;
                }
            }
        }
        str += "<br>";
       }
        msg.innerHTML = str;
    }

    function keydown(event)
    {
        var key = event.keyCode;
        maze[CY][CX] = 0;
        switch(key){
            case 38 : if( maze[CY-1][CX  ] == 0 ) CY--; wn=0; break;
            case 40 :                                   wn=1; break;
            case 37 :                                   wn=2; break;
            case 39 :                                   wn=3; break;
        }
        maze[CY][CX] = 3;
        maze_disp3();
    }
・・・省略
```

JavaScriptによる　ミニゲームベース⑦　マインスイーパ

マインスイーパは、地雷が隠れているマスを開かずに、できるだけ短い時間で地雷の無いマス全てを開く（＝全ての地雷を見つける）ゲームである。地雷のある所を開いてしまうと失敗になる。（右図参照）
名前の由来は、マイン＝地雷。スイーパー＝除去、掃除。
Windows 3.1 の時代から Windows に標準搭載されており、パソコンに触った事の有る人であれば一度はプレイした事が有るであろう有名なゲームである。不朽の名作として紹介される事も多く、恐らく Windows 標準搭載ゲームの中で最も人気のあるゲームだと思われる。 Windows 標準搭載のものが最も有名ではあるが、単純なゲーム故に作成も容易であるためか多数の実装が存在しており、Linux などの Windows 以外の OS においても標準で類似のゲームが搭載されている。



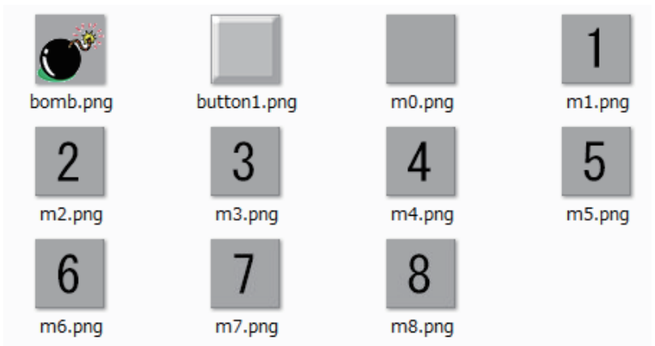
「js26.html」の（マインスイーパーもどき）をベースに発展させます。

X →　　配列 m[][]

Y ↓	0	1	2	3	4	5	6	7	8	9
0										
1		1	1	1		1	1	1		
2		1	-9	1		1	-9	1		
3		1	1	2	2	3	2	1		
4				1	-9	-9	1			
5				1	2	2	1			
6										
7										
8										
9										

ゲーム等の多くの情報を扱うプログラムをわかりやすく簡潔に記述するには、どのようなデータ構造を用いるかを考える事が重要である。今回は種々の判定を効率よく行うため次のような工夫をした。

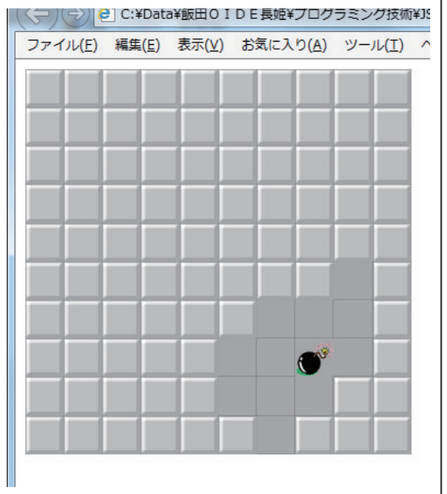
○爆弾、その周りの数値は２次元配列で表現
○爆弾はマイナスの値
○数値は画像として０～８まで８個を準備
○０の画像は数値を入れず空白としている



JavaScript g7-01　盤面を表示し、爆弾をセット、クリックでマスを開く

jsg7-1.html

```
<html>
<head>
  <script type="text/javascript">
      var m=[];
      var img_path;
    function change(obj)
    {
        var id = obj.id;
        var y=Math.floor(id/10);
       var x=id%10;
        if( m[y][x] < 0    )
           obj.src = "img/bomb.png";     // 爆弾

        else
           open(y,x);                    // それ以外
    }

     function open(y,x)     // マスを開く関数
     {
          var id = y*10+x;
          var obj=document.getElementById(id)
          obj.src="img/m" + m[y][x] + ".png";
     }

  </script>
</head>
<body>
    <script type="text/javascript">
        var i,j,id;
        for(i=0;i<10;i++){   // 全部で100個のマス
            m[i]=[];         // データは２次元配列に
            for(j=0;j<10;j++){
                m[i][j] = 0;      // 最初は中身はゼロ
                id = i*10+j;      // ＩＤ　0 ～ 99
                document.write("<img id='" + id + "' src='img/button1.png'
                                                width=32 onClick ='change(this)'>");
            }
            document.write("<br>");
        }
        for(i=0;i<10;i++){   // まれに爆弾がダブって設定されることあり
            var bx = Math.floor(Math.random()*10);// 爆弾の横位置を乱数で設定
            var by = Math.floor(Math.random()*10); //  〃  縦位置を乱数で設定
            m[by][bx] = -9;  //爆弾をセット（マイナス値）
        }
       img_path = (document.getElementById("0" )).src  //ボタン画像のフルパス
    </script>
</body>
</html>
```

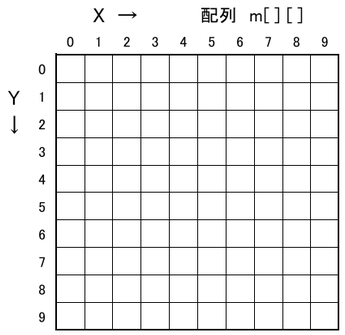


	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0

JavaScript g7-02　爆弾のデータの周りの数値データをカウントする。

jsg7-2.html

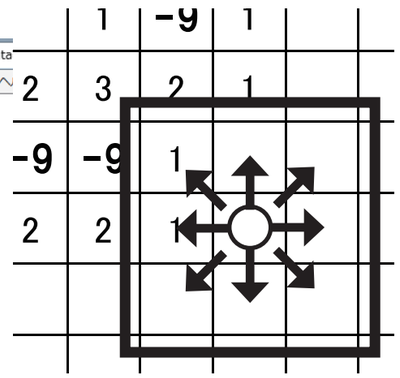
```
<html>
<head>
  <script type="text/javascript">
      var m=[];
      var img_path;

    function change(obj)
    {
      ・・・省略
    }

      function open(y,x)
      {
      ・・・省略
      }

      function cnt(y,x)      // 周りの爆弾の数をカウント
      {
            if(y<0 || y>9 || x<0 || x>9) return; // 盤面外ならリターン
            m[y][x]++;
      }

  </script>
</head>
<body>
    <script type="text/javascript">
        var i,j,id;
        for(i=0;i<10;i++){   // 全部で 100 個のマス
            m[i]=[];         // データは２次元配列に
            for(j=0;j<10;j++){
                m[i][j] = 0;       // 最初は中身はゼロ
                id = i*10+j;       //  ＩＤ　0 ～ 99
                document.write("<img id='" + id + "' src='img/button1.png'
                                        width=32 onClick ='change(this)'>");
            }
            document.write("<br>");
        }
        for(i=0;i<10;i++){
            var bx = Math.floor(Math.random()*10);// 爆弾の横位置を乱数で設定
            var by = Math.floor(Math.random()*10); //  〃  縦位置を乱数で設定
            m[by][bx] = -9;  //爆弾をセット（マイナス値）
            cnt(by-1,bx-1);  cnt(by-1,bx+0);  cnt(by-1,bx+1);      //爆弾周囲の数値設定
            cnt(by-0,bx-1);                   cnt(by+0,bx+1);
        }
        img_path = (document.getElementById("0" )).src   //ボタン画像のフルパス
    </script>

</body>
</html>
```



JavaScript g7-03　隣接するゼロのマスを連続して開ける（再帰呼び出し）

jsg7-3.html

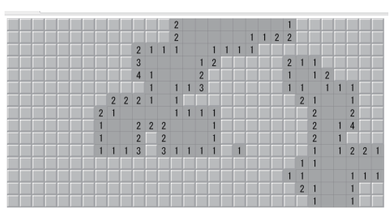
```
<html>
<head>
  <script type="text/javascript">
      var m=[];
      var img_path;
    function change(obj)
    {
      ・・・省略

    }

      function open(y,x)
      {
            if(y<0 || y>9 || x<0 || x>9) return;      // 盤面外ならリターン
            var id = y*10+x;
            var obj=document.getElementById(id)
            if( obj.src != img_path ) return;         // すでに開かれていればリターン
            obj.src="img/m" + m[y][x] + ".png";
            if(m[y][x] != 0) return;                  // 数値がゼロでなければリターン
            open(y-1,x-1);                  // 現在位置を中心に８方向を開く（再帰）
                                            //
                                            //
                                            //
                                            //
                                            //
                                            //
                                            //
                                            //
      }

      function inc(y,x)
      {
            ・・・省略
      }

  </script>
</head>
<body>

      ・・・省略

</body>
</html>
```



JavaScript g7-04

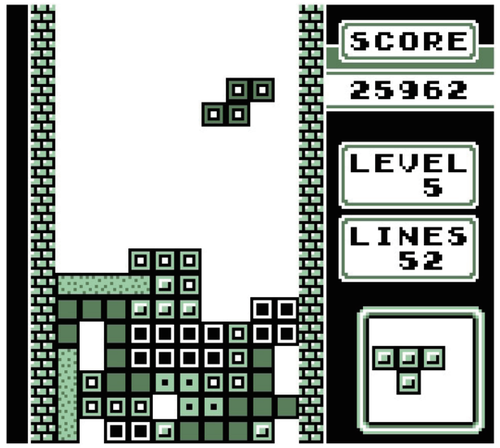
盤面のマス数を横３０、縦１５、
爆弾数を８０個に変更せよ。

jsg7-4.html



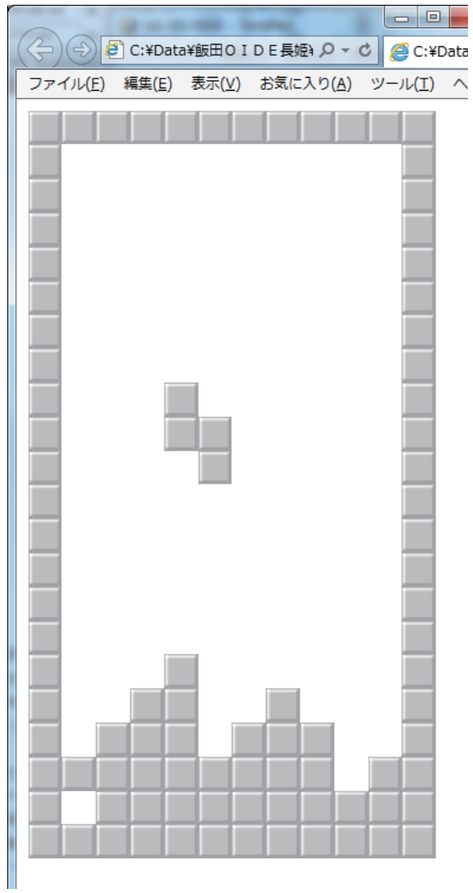
JavaScriptによる　ミニゲームベース⑧　テトリス

テトリス（ロシア語：Тетрис）は、1980年代末から1990年代初めにかけ、世界各国で大流行したコンピューターゲーム。落ち物パズルの元祖である。オリジナルはソビエト連邦のコンピュータ科学者アレクセイ・パジトノフが開発した。1984年6月6日に初めてプレイ可能な版が開発され、その後ライセンス供給が様々なゲーム制作会社に対してなされ、各種のプラットフォーム上で乱立する状態になった。パジトノフは当初､オモチャ屋でみかけた（正方形５つからなる）ペントミノのパズルピースセットを、勤めていた研究所（ドラドニーツィン・コンピューティングセンター）のコンピュータで６日かけて再現し、そのゲームを「遺伝子工学」(Genetic Engineering）と名付けていた。次いで扱いやすいよう図形を（正方形４つからなる）テトロミノに変更し、その７種の「テトロミノス」を十字キーで画面に隙間なく置いていくというゲームにした。しかしまだコンピュータ・ゲームならではのアクション性が足りないように思え、枠の中に置いたテトロミノスの数で得点を競うようにもしてみたが満足せず、やがて枠を画面全体でなく縦長の「通路」にし、上から下へ落ちてくるテトロミノスを操作するというゲームにした。しかしそれだけだと枠が短時間で埋まってしまう。そこから「横一列に並んだテトロミノスは消滅する」という発想が生まれた。



◆ゲーム画面は横１０マス、縦２０マスで構成。データは２次元配列で管理し、ブロック、壁の有無を１，０で処理する。
◆ブロックの移動、重なり、壁の判定などは配列へデータをセットすることで行い、処理が終わった後に一括で画面に再表示させる。

	0	1	2	3	4	5	6	7	8	9	10	11
0												
1												
2												
3												
4												
5												
6												
7												
8												
9												
10						1						
11						1	1					
12						1						
13												
14												
15												
16												
17	1	0	0	0	0	0	0	0	0	0	0	1
18	1	0	0	0	0	0	0	0	0	0	0	1
19	1	0	0	0	0	0	0	0	0	0	0	1
20	1	0	0	0	0	0	0	0	0	0	0	1
21	1	1	1	1	1	1	1	1	1	1	1	1



JavaScript g8-01　画面（壁）の初期化と表示

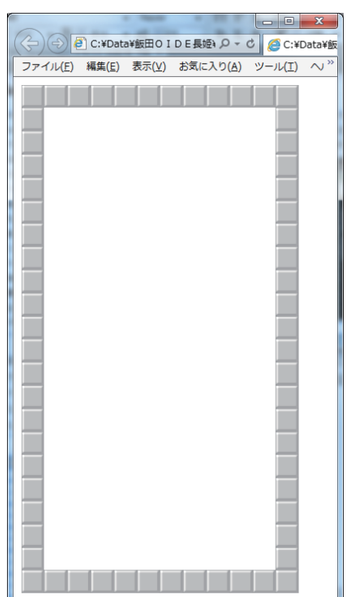
jsg8-1.html

```
<html>
<head>
  <script type="text/javascript">
    var x,y,xx,yy,n;
    var ban=[];
    for(y=0;y<=21;y++){
        ban[y] = [];
        for(x=0;x<=11;x++)
            ban[y][x]=1;        // 縦２２横１２の
    }                                配列全てを１にセット
    for(y=1;y<=20;y++){
        for(x=1;x<=10;x++)
            ban[y][x] = 0;      // 外側の壁を残して
    }                                内側を０にセット

    function disp()             // 配列表示関数
    {
        var dp = document.getElementById('disp');
        str="";
        for(y=0;y<=21;y++){
            for(x=0;x<=11;x++){
                if(ban[y][x] == 1)
                    str += "<img src='img/button1.png' width=24 >";
                else
                    str += "<img src='img/toumei.png' width=24 >";
            }
            str += "<br>";
        }
        dp.innerHTML = str;
    }

  </script>
</head>
<body>
    <p id="disp"></p>
    <script type="text/javascript">
        disp();
    </script>
</body>
</html>
```

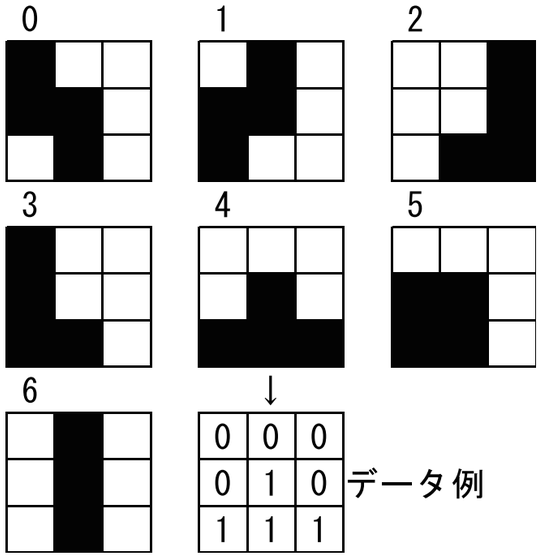
	0	1	2	3	4	5	6	7	8	9	10	11
0	1	1	1	1	1	1	1	1	1	1	1	1
1	1	0	0	0	0	0	0	0	0	0	0	1
2	1	0	0	0	0	0	0	0	0	0	0	1
3	1	0	0	0	0	0	0	0	0	0	0	1
4	1	0	0	0	0	0	0	0	0	0	0	1
5	1	0	0	0	0	0	0	0	0	0	0	1
6	1	0	0	0	0	0	0	0	0	0	0	1
7	1	0	0	0	0	0	0	0	0	0	0	1
8	1	0	0	0	0	0	0	0	0	0	0	1
9	1	0	0	0	0	0	0	0	0	0	0	1
10	1	0	0	0	0	0	0	0	0	0	0	1
11	1	0	0	0	0	0	0	0	0	0	0	1
12	1	0	0	0	0	0	0	0	0	0	0	1
13	1	0	0	0	0	0	0	0	0	0	0	1
14	1	0	0	0	0	0	0	0	0	0	0	1
15	1	0	0	0	0	0	0	0	0	0	0	1
16	1	0	0	0	0	0	0	0	0	0	0	1
17	1	0	0	0	0	0	0	0	0	0	0	1
18	1	0	0	0	0	0	0	0	0	0	0	1
19	1	0	0	0	0	0	0	0	0	0	0	1
20	1	0	0	0	0	0	0	0	0	0	0	1
21	1	1	1	1	1	1	1	1	1	1	1	1


button1.png


toumei.png



JavaScript g8-02 ブロックデータの初期化



0	0	0
0	1	0
1	1	1

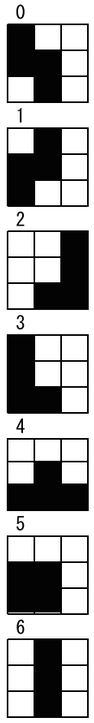
データ例

３×３の７個分の３次元配列にブロックのパターンを設定する。ブロックのある位置に「１」をセットしておく。

続いてこのブロックパターンを画面に表示する。このときブロックの左上の座標を「ｘｘ、ｙｙ」としブロックの番号を「ｎ」とする。
これらの変数は様々な関数で使用するため、グローバル宣言しておく。

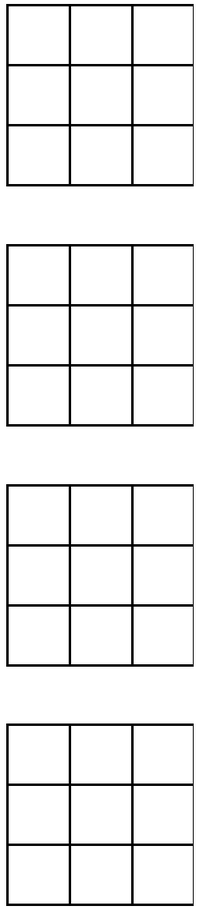
jsg8-2.html

```
<html>
<head>
  <script type="text/javascript">
    var b =[[[1,0,0],
             [1,1,0],
             [0,1,0]],
            [[0,1,0],
             [1,1,0],
             [1,0,0]],
            [[0,0,1],
             [0,0,1],
             [0,1,1]],
            [[1,0,0],
             [1,0,0],
             [1,1,0]],
            [[0,0,0],
             [0,1,0],
             [1,1,1]],
            [[0,0,0],
             [1,1,0],
             [1,1,0]],
            [[0,1,0],
             [0,1,0],
             [0,1,0]]
          ];
  var x,y,xx,yy,n;
  var ban=[];
・・・以下、省略
```



0	0	0
0	1	0
1	1	1

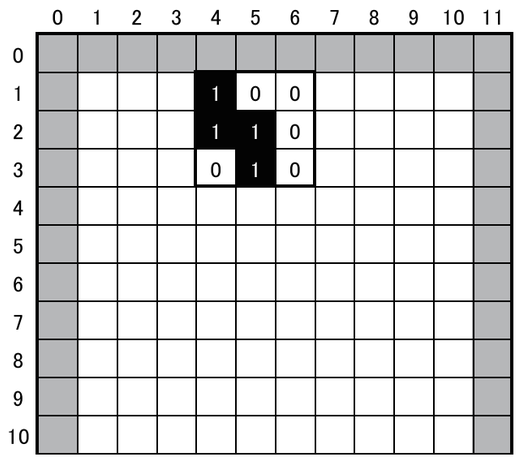
オリジナルブロックのパターンデザイン

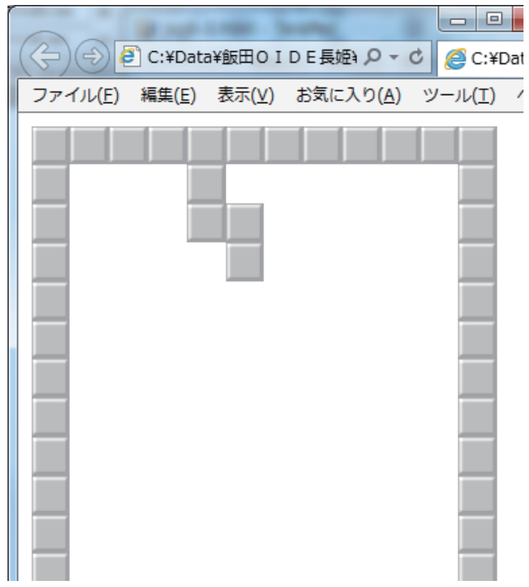


JavaScript g8-03 指定位置（xx,yy）にブロック（n番）データを表示

jsg8-3.html

```
<html>
<head>
  <script type="text/javascript">

    ・・・省略

    function disp()
    {
        ・・・
        ・・・省略
    }
                        // ブロックセット関数
    function bset(f)
    {
        for(y=0;y<3;y++){
            for(x=0;x<3;x++){
                if(                    )
                    ban[yy+y][xx+x] = f;    // 1:表示    2:消去
            }
        }
        disp();
    }

  </script>
</head>
<body>
    <p id="disp"></p>
    <script type="text/javascript">
        xx=4; yy=1; n=0;
        bset(1);
    </script>
</body>
</html>
```



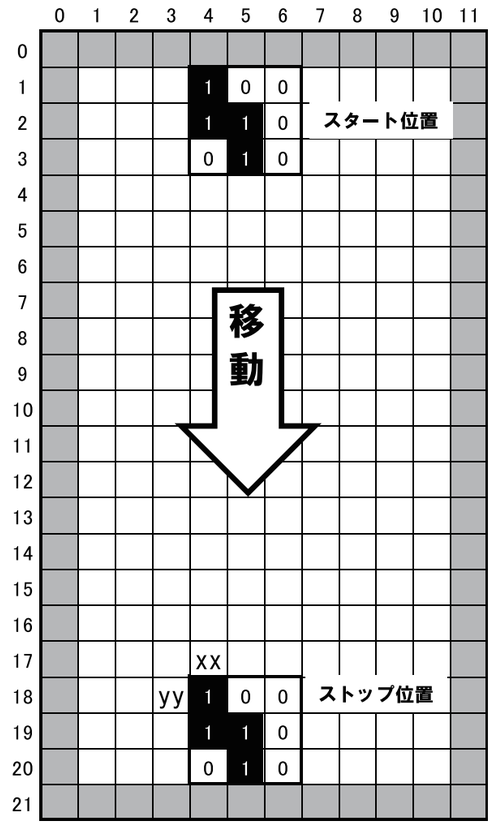


関数 bset(f) で　ｎ番のブロックをｘｘ、ｙｙの位置にデータセット
関数 disp() で実際に表示する。
bset(f) の引数「f」は１か０で表示するか消去するかを指定する。

JavaScript g8-04 ブロックの移動（自動的に落ちて、下で止める）

jsg8-4.html

```
<html>
<head>
  <script type="text/javascript">
     ・・・省略

    function disp()
    {
        ・・・省略
    }

    function bset(f)
    {
        ・・・省略
    }

    function fall()
    {
        bset(0);          // ブロック消去
        if( yy==18 ){     // 下まで到達
            bset(1);
            xx=4; yy=1; n = Math.floor(Math.random()*7);
        }                 // 次のブロック （０～６の乱数）
        else
            [          ]  // 下へ移動
        bset(1);          // ブロックセット・表示
        setTimeout("fall()",200);   // 連続して呼び出す
    }

  </script>
</head>
<body>
    <p id="disp"></p>
    <script type="text/javascript">
          xx=4; yy=1; n=0;
          fall();   // ブロック移動関数
    </script>
</body>
</html>
```

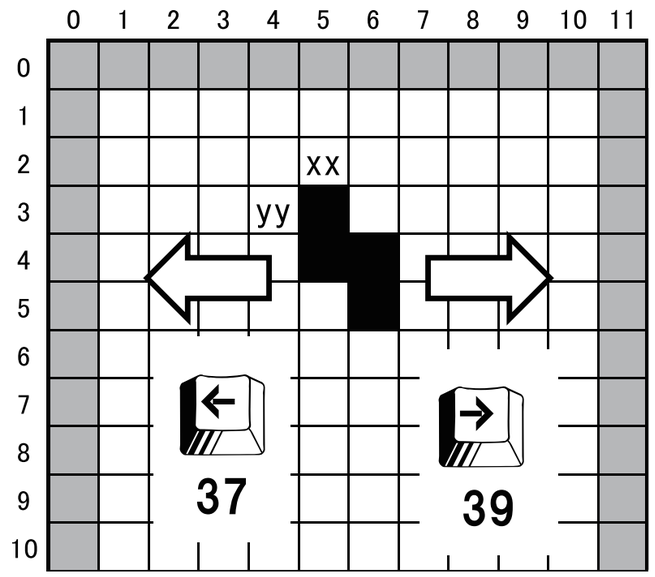


タイマーを設定しブロックを下へ移動する処理を加える。ブロックのｙ座標（ｙｙ）を増加しながら表示、消去を繰り返していく。このままだとブロックはブラウザ外へ動き続けてしまうので下（ｙ座標値１８）まで到達したら、再び上の位置から新たなブロックが出現するようにプログラムする。
ブロックの種類は乱数を使用して０～６まで発生させる。ブロックが重なった時の処理は次の段階とする。

JavaScript g8-05 キー入力でブロックを左右に移動

jsg8-5.html

```
<html>
<head>
  <script type="text/javascript">

     ・・・省略

    function disp()
    {
     ・・・省略
    }
    function bset(f)
    {
     ・・・省略
    }
    function fall()
    {
     ・・・省略
    }

    function keydown(event)
    {
        var key = event.keyCode;
        bset(0);                          // ブロック消去
        switch( key ){
            case 37 : [          ] break;  // 左カーソルキー
            case 39 : [          ] break;  // 右カーソルキー
        }
        bset(1);                          // ブロックセット・表示
    }

  </script>
</head>
<body onKeydown="keydown(event)">
    <p id="disp"></p>
    <script type="text/javascript">
          xx=4; yy=1; n=0;
          fall();
    </script>
</body>
</html>
```



キー入力イベント

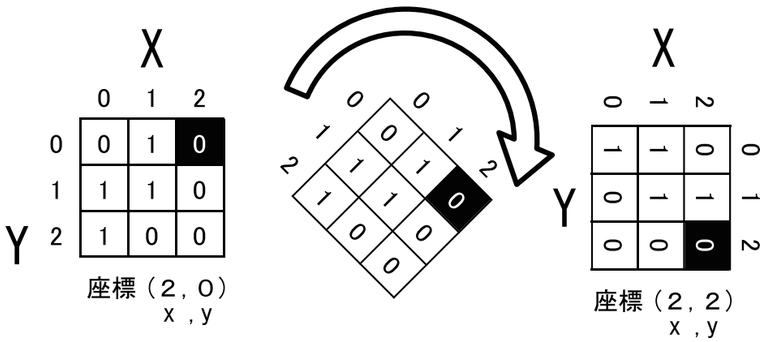
keydown イベントを仕掛けて左右のカーソルキーの入力でブロックを左右に移動します。この段階では、左右の壁で止まる処理はしていません。

JavaScript g8-06

キー入力でブロックを回転させる

ブロックを回転させるという事は、ブロックのデータが格納されている２次元配列のデータを回転させるという事である。方法はいろいろあるが、回転前と回転後のデータを比較して規則性があるかどうかを調べてみよう。

配列データを回転



回転前と回転後のデータには表の様な規則性がある。これを数式に表すと…

配列データの座標

回転前		回転後	
X	Y	X	Y
0	0	2	0
1	0	2	1
2	0	2	2
0	1	1	0
1	1	1	1
2	1	1	2
0	2	0	0
1	2	0	1
2	2	0	2

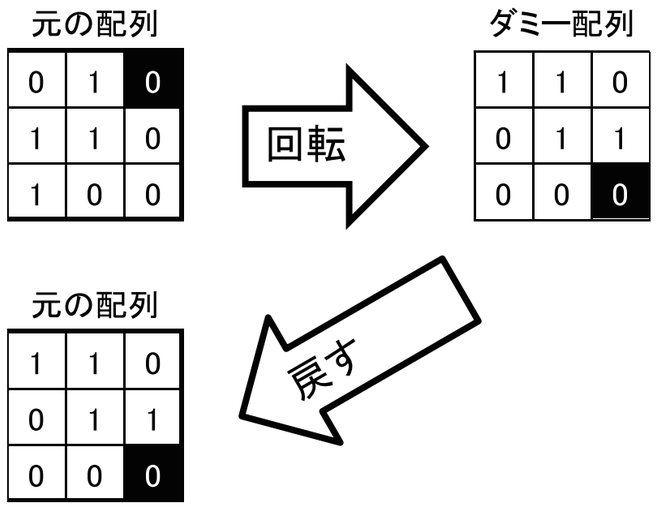
回転前のX座標値が
回転後のY座標値に

回転前 Y座標	回転後 X座標
0	2
1	1
2	0

X→Y座標の変換

x	x-2	abs(x-2)
0	-2	2
1	-1	1
2	0	0

この処理を〃配列内で行う事は難しいので、回転後のデータを一旦、別の配列に格納し、そのデータを元の配列に戻すようにする。



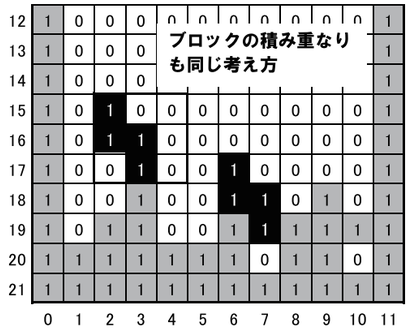
jsg8-6.html

```
・・・省略
    function disp()
    {
    ・・・省略
    }
    function bset(f)
    {
    ・・・省略
    }
    function fall()
    {
    ・・・省略
    }

     function keydown(event)
     {
         var key = event.keyCode;
         bset(0);                              // ブロック消去
         switch( key ){
             case 37 : xx--;   break;          // 左カーソルキー
             case 39 : xx++;   break;          // 右カーソルキー
             case 32 : turn(); break;          // スペースキー　で回転
         }
         bset(1);                              // ブロックセット・表示
     }

     function turn()
     {
         var dmy = [[0,0,0],[0,0,0],[0,0,0]];  // ダミー配列
         for(x=0;x<3;x++){
             for(y=0;y<3;y++){
                 dmy[y][x] = b[n][Math.abs(x-2)][y];     // 回転処理
             }
         }
         for(x=0;x<3;x++){
             for(y=0;y<3;y++){
                 b[n][y][x] = [          ]     // 元の配列に戻す
             }
         }
     }
  </script>
</head>
<body onKeydown="keydown(event)">
    ・・・省略
    <script type="text/javascript">
          xx=4; yy=1; n=0;
          fall();
```

JavaScript g8-07

ブロックが壁、他のブロックに重なるかを判定

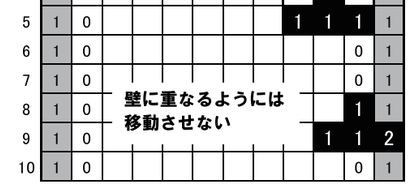
移動中のブロックが壁で止まったり、他のブロックが積み重なっているところで止まったりする処理を付け加えます。具体的にはこれから移動しようとする場所に「壁」や「ブロック」があるかどうかを判定する関数を作り、あれば移動せず、なければ移動処理をするようにします。壁、ブロックは２次元配列の情報としてはどちらも「１」なのでひとつの処理として記述できます。



・・・省略

jsg8-7.html

```
function fall()
{
    bset(0);
    if( check(yy+1,xx) == 1 ){              // １つ下に壁、ブロックがあれば止める
        bset(1);
        xx=4; yy=1; n = Math.floor(Math.random()*7);
    }
    else
        yy++;
    bset(1);
    setTimeout("fall()",200);
}

function keydown(event)
{
    var key = event.keyCode;
    bset(0);
    switch( key ){
        case 37 : if(check(yy,xx-1) != 1) xx--;  break;   // 左壁がなければ移動
        case 39 : if(check(yy,xx+1) != 1) xx++;  break;   // 右壁     〃
        case 32 : turn(); break;
    }
    bset(1);
}

function check(cy,cx)                       // 重なりチェック関数
{
    for(y=0;y<3;y++){
        for(x=0;x<3;x++){
            if( ban[cy+y][cx+x]==1 && b[n][y][x]==1)
                return 1;                   // 重なれば「１」を返す
        }
    }
                                            // 重ならなければ「０」を返す
}
```

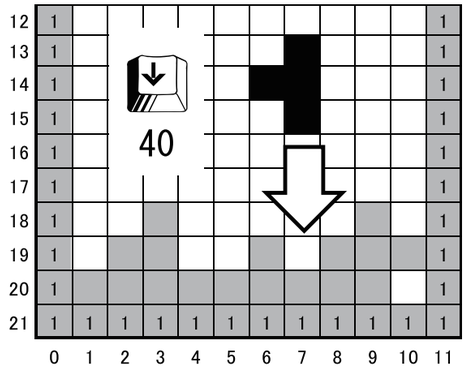


・・・省略
</script>

JavaScript g8-08

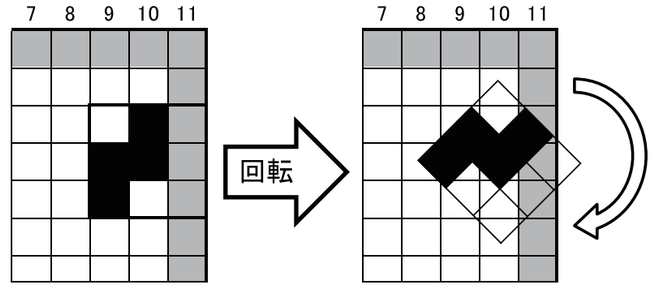
・キー入力でブロックを落とす

落す位置が決定したら「↓」キーでブロックが止まる位置（１番下か、すでにブロックが積み重なっている位置（check() 関数で判定））まで移動させる。



・壁際でのブロック回転

ブロックを壁際で回転させるとき、壁と重なる場合がある。このような場合には、回転後にブロックん位置を壁から１コマ離す処理を付け加える。



・・・省略

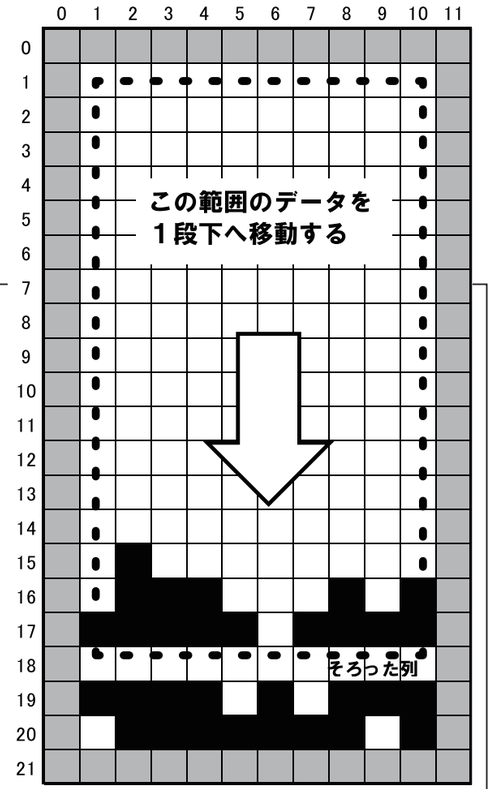
jsg8-8.html

```
function keydown(event)
{
    var key = event.keyCode;
    bset(0);
    switch( key ){
        case 37 : if(check(yy,xx-1) != 1) xx--;  break;
        case 39 : if(check(yy,xx+1) != 1) xx++;  break;
        case 38 : turn();
                  if( check(yy,xx) == 1){       // 壁と重なるなら
                      if( x<2 )
                          xx++;                 // 左壁なら右へ移動
                      else
                          [          ]          // 右壁なら左へ移動
                  }                               break;

        case 40 : while( check(yy+1,xx ) == 0) // 壁かブロックが
                          [          ]          // あるまで下へ移動
                  break;
    }
    bset(1);
}
```

・・・省略

JavaScript g8-09 ブロックが１列そろったら消す

ブロックを積み重ねていき、１列揃ったらその列を消去する処理を付け加える。調べる列の全てのデータが１なら、横１列の合計は「１０」となるので、そこから上のデータを下へ移動する。

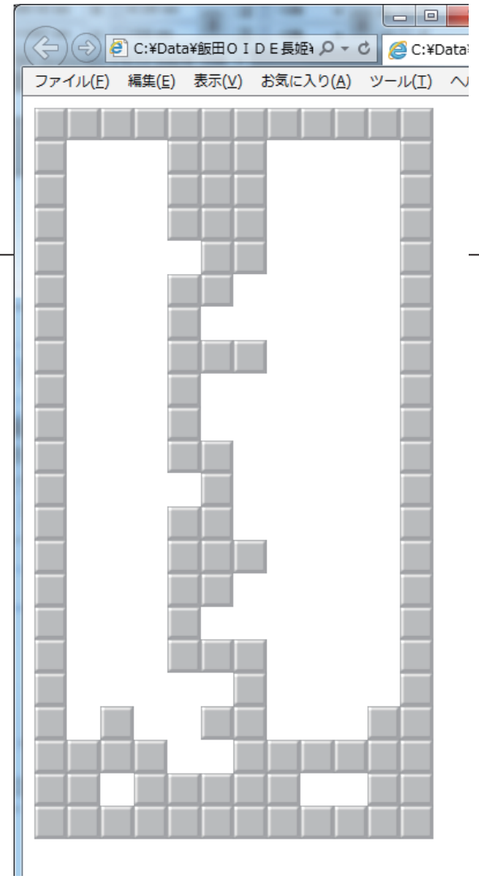


jsg8-9.html

```
・・・省略

    function fall()
    {
        bset(0);
        if( check(yy+1,xx) == 1 ){
            bset(1);
            del();  // del() 関数呼び出し
            xx=4; yy=1; n = Math.floor(Math・・・
        else
            yy++;
        bset(1);
        setTimeout("fall()",200);
    }

  ・・・省略

    function del()
    {
        var sum;
        for(y=20;y>=1;y--){                    // 下壁の１コマ上から調べる
            sum=0;                             // カウント用
            for(x=1;x<=10;x++)                 // 横１列をカウント
                sum += ban[y][x];
            if(          ){                    // もし横１列が揃ったら
                for(var dy=y;dy>1;dy--){       // 上のブロックを下へ移動
                    for(x=1;x<=10;x++)
                        ban[dy][x] = ban[dy-1][x];
                }
                y++;                           // 同じ位置をもう一度調べるため
            }
        }
        disp();
    }

  </script>
・・・省略
```


JavaScript g8-10 ブロックが上まで積み重なったらゲームオーバー

ブロックが積み重なっていき、上まで到達したらゲームオーバーとする処理を付け加える。



jsg8-10.html

```
・・・省略

    function del()
    {
        var sum;
        if(yy<=1){   // 上まで積み重なったら
            document.bgColor="black";
            exit();  // 背景色を黒にして停止
        }
        for(y=20;y>=1;y--){
            sum=0;
            for(x=1;x<=10;x++)
                sum += ban[y][x];
            if( sum == 10 ){
                for(var dy=y;dy>1;dy--){
                    for(x=1;x<=10;x++)
                        ban[dy][x] = ban[dy-1][x];
                }
                y++;
            }
        }
        disp();
    }

  </script>

・・・省略
```

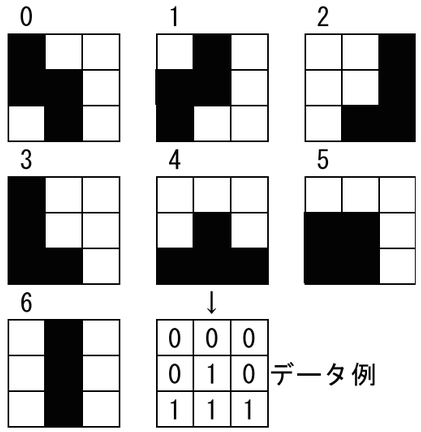
JavaScript g8-11 ブロックパターン、ブロック画像等を変更する。

ブロックに使用する画像やサイズ、ブロックの並び方のパターンなどを変更してオリジナルゲームに仕上げよ。

Javascript 版

テトリスもどき

jsg8-10.html



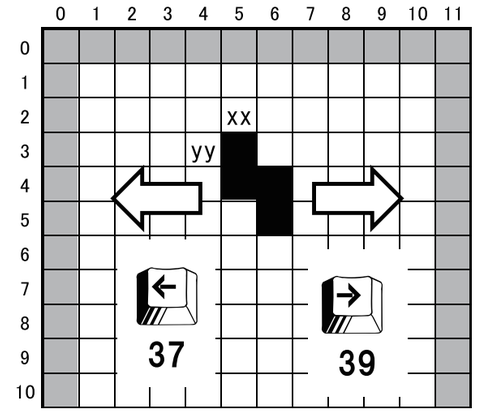
↓

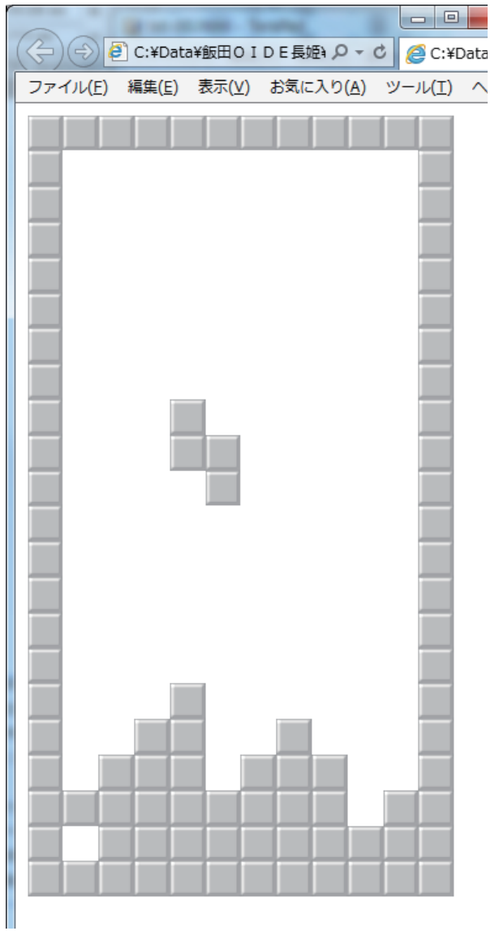
0	0	0	
0	1	0	データ例
1	1	1	

```
<html>
<head>
  <script type="text/javascript">
    var b =[[[1,0,0],
             [1,1,0],
             [0,1,0]],
            [[0,1,0],
             [1,1,0],
             [1,0,0]],
            [[0,0,1],
             [0,0,1],
             [0,1,1]],
            [[1,0,0],
             [1,0,0],
             [1,1,0]],
            [[0,0,0],
             [0,1,0],
             [1,1,1]],
            [[0,0,0],
             [1,1,0],
             [1,1,0]],
            [[0,1,0],
             [0,1,0],
             [0,1,0]]
           ];
    var x,y,xx,yy,n;
    var ban=[];
    for(y=0;y<=21;y++){
        ban[y] = [];
        for(x=0;x<=11;x++)
            ban[y][x]=1;
    }
    for(y=1;y<=20;y++){
        for(x=1;x<=10;x++)
            ban[y][x] = 0;
    }

    function disp()
    {
              var dp = document.getElementById('disp');
              str="";
        for(y=0;y<=21;y++){
            for(x=0;x<=11;x++){
                if(ban[y][x] == 1)
                    str += "<img src='img/button1.png' width=24 >";
                else
                    str += "<img src='img/toumei.png' width=24 >";
            }
                     str += "<br>";
        }
        dp.innerHTML = str;
    }

    function bset(f)
    {
        for(y=0;y<3;y++){
            for(x=0;x<3;x++){
                if( b[n][y][x] == 1 )
                    ban[yy+y][xx+x] = f;
            }
        }
        disp();
    }

    function fall()
    {
        bset(0);
        if( check(yy+1,xx) == 1 ){
            bset(1);
            del();
            xx=4; yy=1; n = Math.floor(Math.random()*7);
        }
        else
            yy++;
        bset(1);
        setTimeout("fall()",200);
    }
```

	0	1	2	3	4	5	6	7	8	9	10	11
0												
1												
2												
3												
4												
5												
6												
7												
8												
9												
10						1						
11						1	1					
12						1						
13												
14												
15												
16												
17	1	0	0	0	0	0	0	0	0	0	0	1
18	1	0	0	0	0	0	0	0	0	0	0	1
19	1	0	0	0	0	0	0	0	0	0	0	1
20	1	0	0	0	0	0	0	0	0	0	0	1
21	1	1	1	1	1	1	1	1	1	1	1	1


```
    function keydown(event)
    {
        var key = event.keyCode;
        bset(0);
        switch( key ){
            case 37 : if(check(yy,xx-1) != 1) xx--;  break;
            case 39 : if(check(yy,xx+1) != 1) xx++;  break;
            case 38 : turn();
                      if( check(yy,xx) == 1){
                          if( x<2 )
                              xx++;
                          else
                              xx--;
                      }                              break;

            case 40 : while( check(yy+1,xx ) == 0)
                          yy++;
                      break;
        }
        bset(1);
    }
    function turn()
    {
        var dmy = [[0,0,0],[0,0,0],[0,0,0]];
        for(x=0;x<3;x++){
            for(y=0;y<3;y++){
                dmy[y][x] = b[n][Math.abs(x-2)][y];
            }
        }
        for(x=0;x<3;x++){
            for(y=0;y<3;y++){
                b[n][y][x] = dmy[y][x];
            }
        }
    }
    function check(cy,cx)
    {
        for(y=0;y<3;y++){
            for(x=0;x<3;x++){
                if( ban[cy+y][cx+x]==1 && b[n][y][x]==1)
                    return 1;
            }
        }
        return 0;
    }

    function del()
    {
        var sum;
        if(yy<=1){
            document.bgColor="black";
            exit();
        }
        for(y=20;y>=1;y--){
            sum=0;
            for(x=1;x<=10;x++)
                sum += ban[y][x];
            if( sum == 10 ){
                for(var dy=y;dy>1;dy--){
                    for(x=1;x<=10;x++)
                        ban[dy][x] = ban[dy-1][x];
                }
                y++;
            }
        }
        disp();
    }

  </script>
</head>
<body onKeydown="keydown(event)">
    <p id="disp"></p>
    <script type="text/javascript">
              xx=4; yy=1; n=Math.floor(Math.random()*7);
              fall();
    </script>
</body>
</html>
```



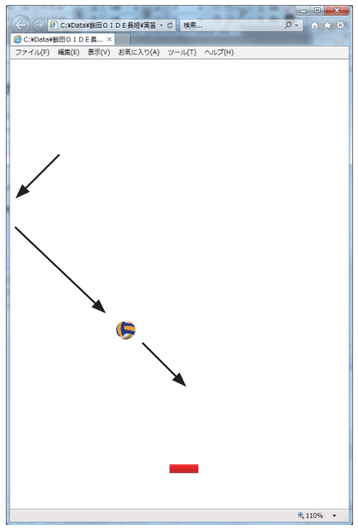


JavaScript による　ミニゲームベース⑨　ブロック崩し

JavaScript g9-01　課題「11-06」をベースに作成していきます。（画像は変更）
ボールを飛び回らせ、キー入力でラケットを移動する

jsg9-1.html

```
<html>
<head>
<script type="text/javascript">
    var x=0,y=200,dx=4,dy=8,rx,ry;
    function move()
    {
        setTimeout("move()",20);
        var ball = document.getElementById("ball");
        var bar = document.getElementById("bar");
        x += dx;
        y += dy;
        ball.style.left = x;
        ball.style.top  = y;
        if ( x<=0 || x+32>=document.body.clientWidth )
            dx = -dx;
        if ( y<=0 || y+32>= document.body.clientHeight )
            dy=-dy;
        if( overlap( ball , bar ) == 1 )
            dy = -dy;    ボールとラケットが重なっていれば跳ね返す
    }
    function racket(event)
    {
        var obj = document.getElementById("bar");
        rx = event.clientX;
        ry = 700;
        obj.style.left = rx;
        obj.style.top  = ry;
    }
    function overlap( obj1 , obj2 )   重なり判定関数
    {
        var x1,y1,w1,h1,x2,y2,w2,h2;
        x1 = obj1.offsetLeft;    x2 = obj2.offsetLeft;
        y1 = obj1.offsetTop;     y2 = obj2.offsetTop;
        w1 = obj1.width;         w2 = obj2.width;
        h1 = obj1.height;        h2 = obj2.height;
        if(( x2 < x1+w1  &&  x2+w2 > x1 ) && (  y2 < y1+h1  &&  y1 < y2+h2 ) )
            return 1;      // 2つのオブジェクトが重なっていれば 1を返す
        else
            return 0;
    }
</script>
</head>
<body bgcolor="#ffffff" onLoad="move()" onMouseMove="racket(event)">
    <img id="ball" src="img/block/ba01.png" width=32 style="position:absolute;">
    <img id="bar" src="img/block/btn01.png" width=50 height=15 style="position:absolute">
    <script type="text/javascript">
        resizeTo(600,900);
    </script>
</body>
</html>
```



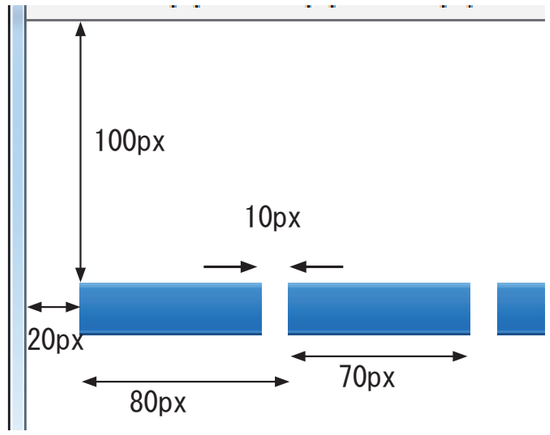
JavaScript g9-02　ブラウザ上部にブロックを表示します。

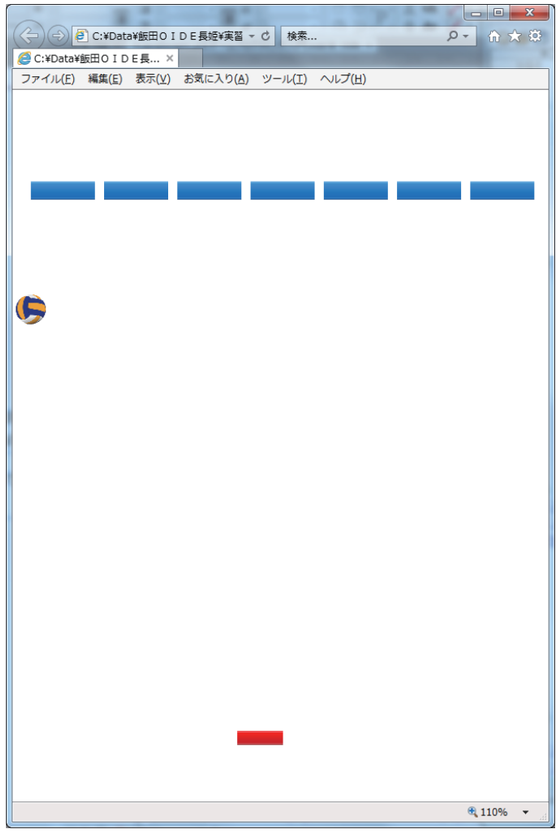
jsg9-2.html

```
・・・省略
<body bgcolor="#ffffff" onLoad="move()" onMouseMove="racket(event)">
    <img id="ball" src="img/block/ba01.png" width=32 style="position:absolute;">
    <img id="bar" src="img/block/btn01.png" width=50 height=15 style="position:absolute">
    <script type="text/javascript">
        resizeTo(600,900);
        for( var j=0; j<7; j++ ){          var id = j;   配置した画像に番号をつける（０～６）
            document.write("<img id='" + id + "' src='img/block/btn05.png' width=70 height=20 ¥
                style='position:absolute; left:" + (j*80+20) + ";top:" + (100) + "'>");
        }
    </script>
</body>
</html>
```

２行に分けて記述する場合は
行の終わりに「￥」を入れます。

ブラウザの原点からＹ座標１００ピクセルの位置
へブロックを１列に７個表示しています。
ブロックの幅 70px、高さ 20px に指定
ブロックの間は 10px 開ける設定です。

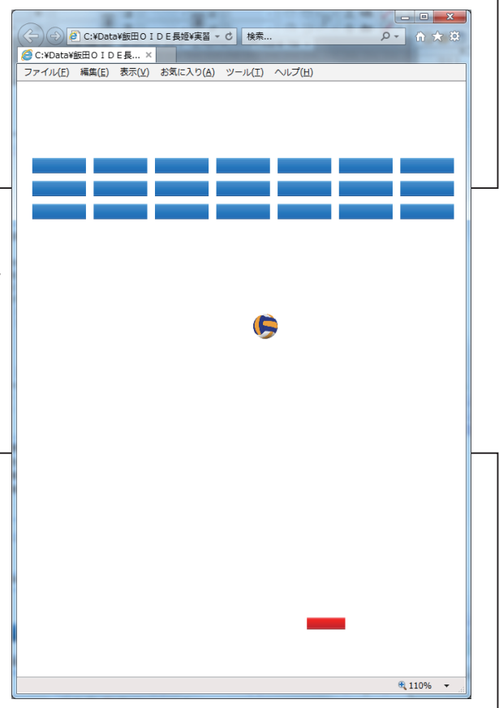




JavaScript g9-03 ブロックの列数を３列に増やします。

```
・・・省略
<body bgcolor="#ffffff" onLoad="move()" onMouseMove="racket(event)">
    <img id="ball" src="img/block/ba01.png" width=32 style="position:absolute;">
    <img id="bar" src="img/block/btn01.png" width=50 height=15 style="position:absolute">
    <script type="text/javascript">
        resizeTo(600,900);
        for( var i=0; i<3; i++ ){
            for( var j=0; j<7; j++ ){
                var id = i*7+j;    配置した画像に番号をつける（０～２０）
                document.write("<img id='" + id + "' src='img/block/btn05.png' width=70 height=20 ¥
                style='position:absolute; left:" + (j*80+20) + ";top:" + (i*30+100) + "'>");
            }                                                    ブロックのタテ位置
        }
    </script>
</body>
</html>
```

jsg9-3.html



JavaScript g9-04 ボールがブロックに当たった処理を追加

・ボールがブロックに当たったか判定し
・ボールをブロックで跳ね返し
・ブロックの表示を消す

必要な処理を move 関数に追加します。

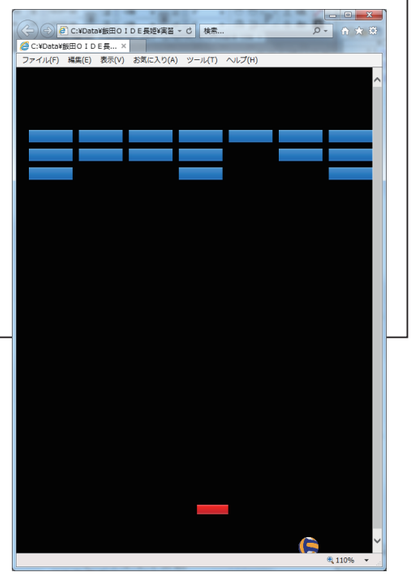
```
・・・省略
    function move()
    {
        ・・・省略
        if( overlap( ball , bar ) == 1 )
            dy = -dy;

          for(var i=0;i<21; i++ ){                    ０～２０までのブロックを判定
            var block = document.getElementById( i );
            if( overlap( ball , block) == 1 ) {       もしブロックに当たったら
                dy = [      ]                             ボールを反転
                block.style.display="none";               ブロックを消す
                break;
            }
        }
    }
・・・以下省略
```

jsg9-4.html

JavaScript g9-05 ボールを打ち損ねたときのゲームオーバー処理を move 関数に加えます。

```
・・・省略
    function move()
    {
        setTimeout("move()",20);
        var ball=document.getElementById("ball");
        var bar = document.getElementById("bar");
        x+=dx;
        y+=dy;
        ball.style.left=x;
        ball.style.top=y;
        if (x<=0 || x+70>=document.body.clientWidth) dx=-dx;
        if (y<=0)    dy=-dy;                              上の壁は跳ね返す
        if (y+32>= document.body.clientHeight ) {         ボールが領域から出たら
            document.bgColor="#000000";                       背景を黒色に
            dx=0; dy=0;                                       ボールの移動量を０に
        }
        if( overlap( ball , bar) == 1 ) dy = -dy;

          for(var i=0;i<21; i++ ){
            var obj3 = document.getElementById( i );
            if( overlap( ball , bar ) == 1 ) {
                dy = -dy;
                    obj3.style.display="none";
                break;
            }
        }

    }
・・・省略
```

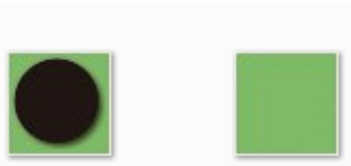
jsg9-5.html

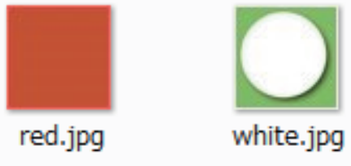


画像やブロックの数や大きさ、スピードなどを変えて、オリジナルなゲームに仕上げましょう！

※ボールの位置などにより、時々おかしな動きをする場合があります。
完全な動作をさせるためにはもう少しの工夫が必要です・・・

JavaScript による　ミニゲームベース⑩　リバーシ（オセロ）





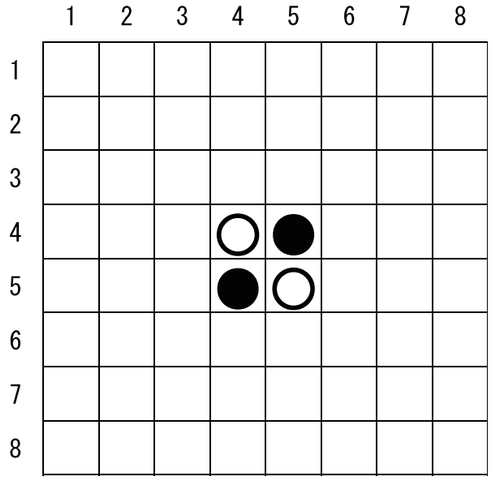
リバーシゲームのデータを２次元配列に設定し、
対応する画像を縦横８×８マスづつ表示します。

０・・・空白のマス（green.jpg）
１・・・黒コマ（black.jpg）
２・・・白コマ（white.jpg）

それぞれに０～６３のＩＤを付けて管理します。

	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	**2**	**1**	0	0	0	0
5	0	0	0	0	**1**	**2**	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0

２次元配列



盤面イメージ

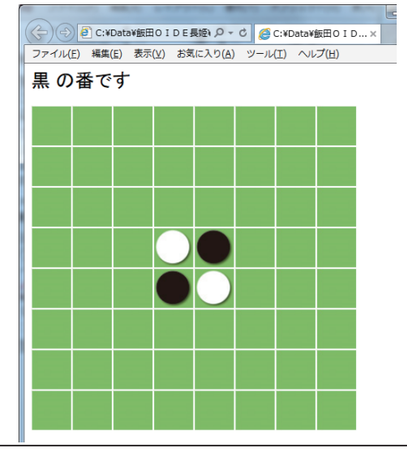
次にコマを打つ順番のプレイヤーの番号（１か２）を変数「PLAYER」で管理しています。
最終的に人間とコンピュータを対戦させることにします。

JavaScript g10-01

リバーシゲームのデータを２次元配列にセットし、盤面を表示する。

jsg10-1.html

```
<html>
<head>
<meta content="text/html; charset=utf-8" http-equiv="Content-Type">
<script type="text/javascript">

    var W = 50;         // １コマのサイズ px
    var BAN = [] , PLAYER ,k=["  ","黒","白"];

    function init()                         // 盤面初期化関数
    {
        for(var i=0;i<=9;i++){
            BAN[i] = [];
            for(j=0;j<=9;j++)
                BAN[i][j] = 0;
        }
          BAN[4][5]=1; BAN[5][4]=1;         // 初期コマ　黒
          BAN[4][4]=2; BAN[5][5]=2;         //           白

        PLAYER = 1;                         // 最初はプレイヤー１の順番
        ban_disp();                         // ゲーム盤の表示
    }

    function ban_disp()
    {
        var y,x,img,n,str="";
        msg = document.getElementById("msg");
        str = "<font size=5>" + k[PLAYER] + " の番です</font><p>"; // 順番表示
        for (y=1;y<=8;y++){
            for (x=1;x<=8;x++){
                switch( BAN[y][x] ){
                    case 0 : img = "img/green.jpg"; break;          // コマなし
                    case 1 : img = "img/black.jpg"; break;          // 黒コマ
                    case 2 : img = "img/white.jpg"; break;          // 白コマ
                }
                n = (y-1)*8+(x-1);                                  // イメージのＩＤ
                str += "<img id='" + n + "' src='" + img + "' width=" + W +">";
            }
            str += "<br>";
        }
        msg.innerHTML = str;
    }
</script>
</head>
<body>
     <div id="msg" >リバーシ</div>
     <script type="text/javascript">
          init();
     </script>
</body>
</html>
```

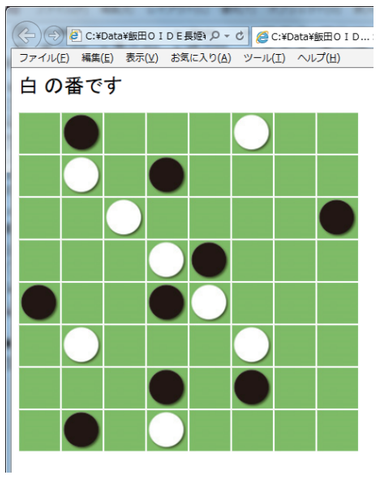


JavaScript g10-02 マウスの操作で盤面にコマを打てるようにする。（重ね打ち）

この段階ではマウス位置に応じて黒白のコマを交互に置いていく。
そこにすでに駒があっても重ねて表示する形になる。

jsg10-2.html

```
<html>
<head>
<meta content="text/html; charset=utf-8" http-equiv="Content-Type">
<script type="text/javascript">

    var W = 50;        //  1コマのサイズ px
    var BAN = [] , PLAYER ,k=["  "," 黒 "," 白 "];

    function init()
    {
      ・・・省略
    }

    function ban_disp()
    {
      ・・・省略
    }

    function manual(obj)     // 手動入力（マウス）関数
    {
        var id = obj.id;
        var y = Math.floor(id/8)+1;
        var x = Math.floor(id%8)+1;
        BAN[y][x] = [          ];          // 打った場所にコマをセット
        PLAYER = PLAYER==1?2:1;            // 次の順番
        ban_disp();                        // 盤面の表示
    }

</script>
</head>
・・・以下　省略
```



X →　　配列 BAN[][]

Y ↓	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	8	9	10	11	12	13	14	15
2	16	17	18	19	20	21	22	23
3	24	25	26	27	28	29	30	31
4	32	33	34	35	36	37	38	39
5	40	41	42	43	44	45	46	47
6	48	49	50	51	52	53	54	55
7	56	57	58	59	60	61	62	63
8								
9								

イメージのＩＤ　０～６３

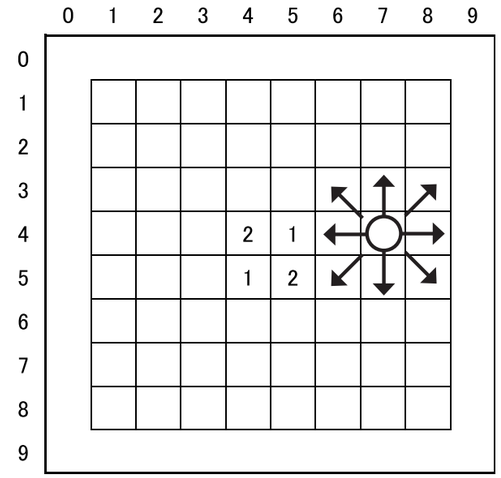
ｙ位置　ＩＤ番号／８
ｘ位置　ＩＤ番号％８

JavaScript g10-03 コマを置く事が出来る位置にだけコマをセット出来るようにする。

コマを置けるのは相手を挟んで反転できる場所のみ。次への発展も考えて、その場所に置けば、相手のコマを何枚反転させる事が出来るのかを返す関数として設計する。（０なら置けない）
この段階では相手のコマは反転しない。

jsg10-3.html

```
<html>
<head>
<meta content="text/html; charset=utf-8" http-equiv="Content-Type">
<script type="text/javascript">

    var W = 50;        //  1コマのサイズ px
    var BAN = [] , PLAYER ,k=["  "," 黒 "," 白 "];

    function init()
    {
      ・・・省略
    }

    function ban_disp()
    {
      ・・・省略
    }

     function manual(obj)  // 手動入力関数
     {
          var id = obj.id;
          var y = Math.floor(id/8)+1;
          var x = Math.floor(id%8)+1;
          if( check(x,y) == 0 )
              return;
          else{
              BAN[y][x] = PLAYER;                  // 打った場所にコマをセット
          }
          PLAYER = PLAYER==1?2:1;                  // 次の順番
          ban_disp();                              // 盤面の表示
     }

     function check(x,y)            // x,y 位置に置くと何枚取れるかを返す関数
     {

          if(BAN[y][x] != 0) return 0;             // 空白コマでなければリターン（０枚）
          var cnt=0,ct,rx,ry;
          var aite = PLAYER==1?2:1;                // 相手のプレイヤー　黒 :1 白 :2

          for(var dy=-1; dy<=1; dy++){             // 上下左右、斜めの８方向を調べる
              for(var dx=-1;dx<=1;dx++){
                  ct = 0;                          // １つのラインで反転できるコマ数 ct
                  rx=x+dx; ry=y+dy;
                  while( BAN[ry][rx] == aite ){    // 相手のコマが並んでいれば繰り返す
                      ct++;                        // 反転できるコマをカウント
                      rx += dx; ry[          ]     // 次の位置へ進める
                  }
                  if( BAN[ry][rx] == PLAYER ) cnt += ct;   // 合計 cnt に累計
              }
          }
          return cnt;                                      // 反転できる枚数を返す
     }
</script>
</head>
・・・以下　省略
```

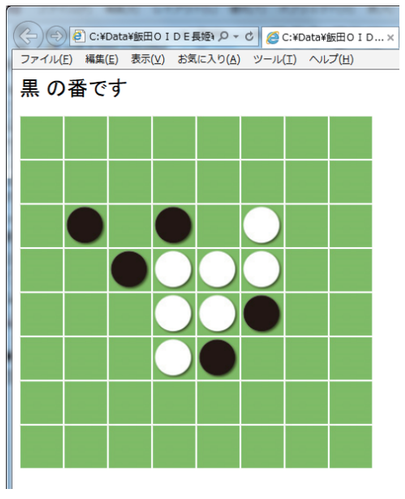


JavaScript g10-04

相手のコマを反転させる処理を加える。コマを置けるかどうかを判定する関数 check() と共通部分が多いが、別にしておいた方が使いやすい。この段階でマウスを使って交互に手動入力する、ゲームのベースが出来上がる。

jsg10-4.html

```
<html>
<head>
<meta content="text/html; charset=utf-8" http-equiv="Content-Type">
<script type="text/javascript">

    var W = 50;         // １コマのサイズ px
    var BAN = [] , PLAYER ,k=["  ","黒","白"];

    function init()                          // 盤面初期化関数
    {
        ・・・省略
    }

    function ban_disp()
    {
        ・・・省略
    }

    function manual(obj)                     // 手動入力（マウス）関数
    {
        ・・・省略
     }

    function check(x,y)                      // x,y 置くと何枚取れるかを返す関数
    {
        ・・・省略
     }

     function reverse(x,y)                                  // コマを反転する関数
     {
        var rx,ry;
        var aite = PLAYER==1?2:1;                           // 相手のプレイヤー　黒 :1 白 :2
        for(var dy=-1; dy<=1; dy++) {                       // 上下左右、斜め８方向を調べる
            for(var dx=-1;dx<=1;dx++) {
                if( dx!=0 || dy!=0) {
                    rx=x+dx; ry=y+dy;
                    while( BAN[ry][rx] == [    ]) {       // 相手のコマが並んでいたら繰り返す
                        rx += dx; ry += dy;
                        if( BAN[ry][rx] == [      ]) {     // 挟めていたら
                            while( !(rx==x && ry==y) ) {    // 戻りながら反転する
                                rx -= dx; ry -= dy;
                                BAN[ry][rx] = PLAYER;
                            }
                            break;
                        }
                    }
                }
            }
        }
    }

</script>
</head>
<body>
<div id="msg" >リバーシ</div>
<script type="text/javascript">
      init();
</script>
</body>
</html>
```



JavaScript g10-05

人間（マウス入力）ｖｓコンピュータ対戦にする。

　コンピュータに自動的に手を打たせる処理を追加します。ここではもっとも簡単な「コマを置く場所を乱数で決定」というアルゴリズムの例を示します。

jsg10-5.html

```
<html>
<head>
<meta content="text/html; charset=utf-8" http-equiv="Content-Type">
<script type="text/javascript">

    var W = 50;         // １コマのサイズ px
    var BAN = [] , PLAYER ,k=["  ","黒","白"];

    function init()                          // 盤面初期化関数
    {
        ・・・省略
    }

    function ban_disp()
    {
        ・・・省略
    }

     function manual(obj)                          // 手動入力（マウス）関数
     {
         var id = obj.id;
         var y = Math.floor(id/8)+1;
         var x = Math.floor(id%8)+1;
         if( check(x,y) == 0 )
            return;
         else{
             BAN[y][x] = PLAYER;                   // 打った場所にコマをセット
             reverse( x , y );                     // 反転処理
         }
         PLAYER = PLAYER==1?2:1;                   // 次の順番
         ban_disp();                               // 盤面の表示
        setTimeout("computer()",1000);  // １秒後にコンピュータの手を呼び出す
     }

    function check(x,y)         // x,y 置くと何枚取れるかを返す関数
    {
        ・・・省略
    }

    function reverse(x,y)       // コマを反転する関数
    {   ・・・省略
    }

     function computer()      // コンピュータの手
     {
          do {
             var x = Math.floor(Math.random()*8)+1;
             var y = Math.floor(Math.random()*8)+1;
           }while( check(x,y) ==[    ]);  // 打てる場所が見つかるまで繰り返す

         BAN[y][x] = PLAYER;                   // コマをセット
         reverse( x , y );                     // 反転処理
         PLAYER = PLAYER==1?2:1;               // 次の順番
         ban_disp();                           // 盤面の表示
     }

</script>
</head>
  ・・・以下、省略
```

プログラミング技術 アルゴリズム勝負 第2弾!

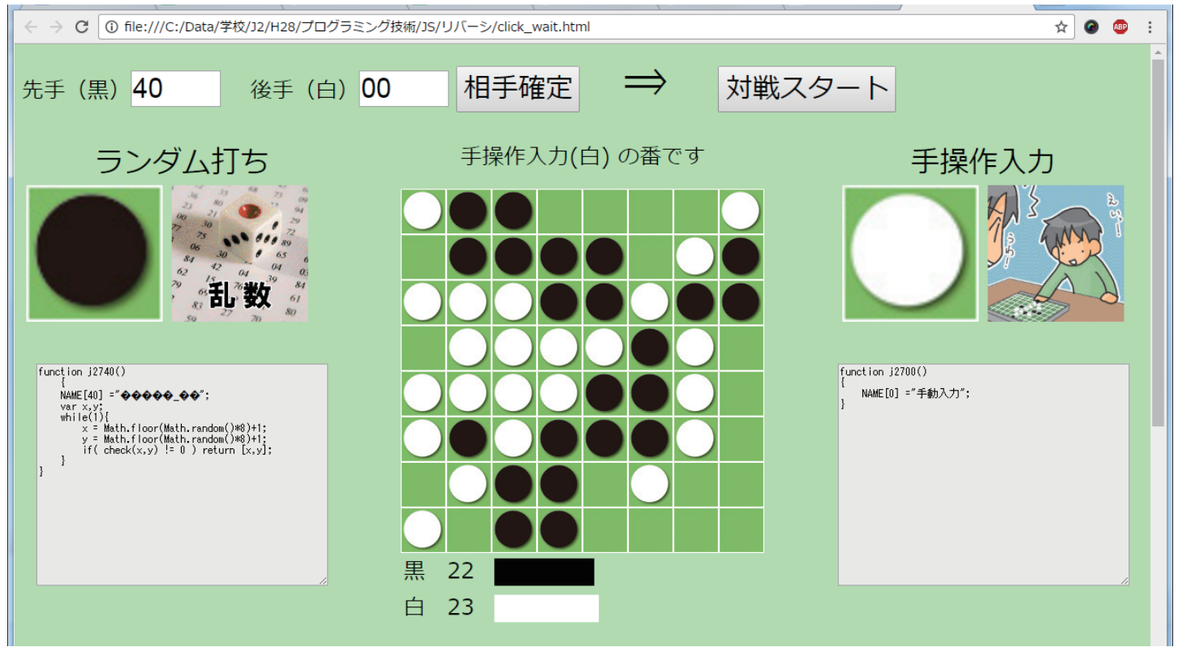
人工知能オセロ大会

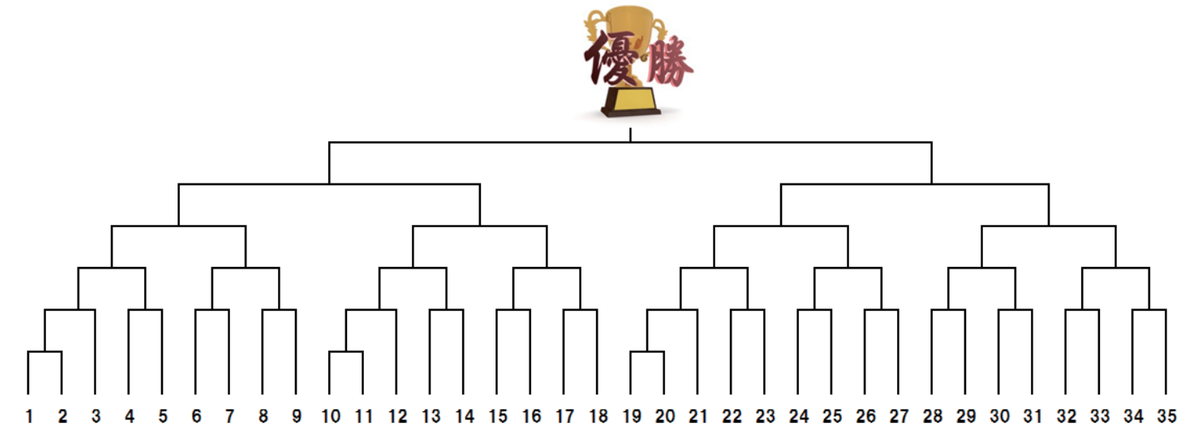
日　時：
場　所：電算室
参加者：プ技　選択者

オセロ（Othello）は、２人用のボードゲーム。交互に盤面へ石を打ち、相手の石を挟むと自分の石の色に変わり最終的に石の多い方が勝ち。単純なルールながらゲームとしての複雑さは人間がゲームの木の全展開を把握可能な程度を超えており、いまだにコンピュータによる全解析は達成されていない。“A minute to learn, a lifetime to master”（覚えるのに１分、極めるのは一生）がオセロのキャッチフレーズである。オセロは商標であることから、ほぼ同一ルールのゲームとして「リバーシ」「白黒ゲーム」「源平碁」の名称が使われることもある。オセロにはじゃんけんなどと違って<u>ルール上偶然の要素はない</u>。ゲーム理論では、オセロは将棋やチェス、囲碁などと同じく、二人零和有限確定完全情報ゲーム（ふたり れいわ ゆうげん かくてい かんぜんじょうほう ゲーム）に分類される。**<u>偶然（運）に左右されないゲーム</u>**であり、ルールが単純なため昔から人工知能（ＡＩ）の良い題材として研究されてきている。

ルールと競技方法

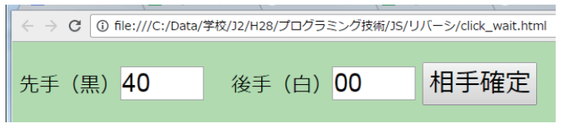
- **各自、オセロの指し手のアルゴリズムを考案し、プログラム化する。**
- **ベースのプログラムから関数化したプログラムを呼び出して実行。**
- **自動的に1回勝負させ、トーナメント方式で1位～3位までを決定。**
- **最終的（全てのマスが埋まるか、どちらも打てる場所がない状態）にコマ数が同じ場合は勝敗がつくまで再戦を行う。**
- **最初の対戦時に自身のアルゴリズムをプレゼンする。（1～2分間）**





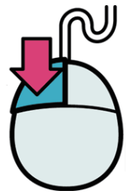
othello-base.html

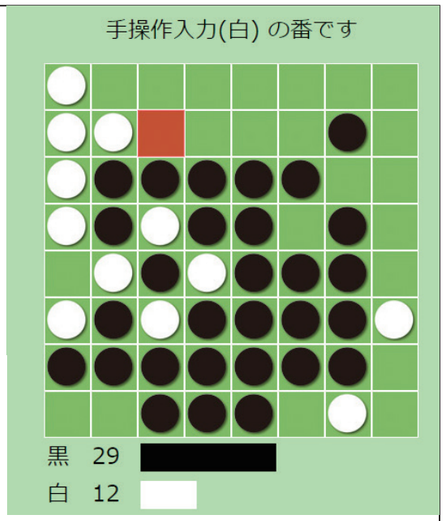
二つのアルゴリズムプログラムを読み込んで自動対戦させるベースプログラム




```
<html>
<head>
    <meta content="text/html; charset=utf-8" http-equiv="Content-Type">
<script type="text/javascript" src="j2700-40.js"></script>

<script type="text/javascript">

    var W = 50;  // 1コマのサイズ px
    var BAN = [] , PLAYER ,k=[" ","黒","白 "],N=["","",""];
    var T_FLAG ,MX,MY,msg2,tID1 2

    function init()  // 盤面初期化関数
    {
        for(var i=0;i<=9;i++){
            BAN[i] = [];
            for(j=0;j<=9;j++)
                BAN[i][j] = 0;
        }
                BAN[4][5]=1; BAN[5][4]=1;   // 初期コマ  黒
                BAN[4][4]=2; BAN[5][5]=2;   //           白

        var player1 = document.getElementById("text1").value;  // オブジェクトＩＤを取得
        var player2 = document.getElementById("text2").value;
        msg = document.getElementById("msg");
        var s1 = document.getElementById("source1");
        var s2 = document.getElementById("source2");
        kekka = document.getElementById("t_kekka");
        var plimg = document.getElementById("plimg");
        var p2img = document.getElementById("p2img");

        p1_func = "j27" + player1 ;                                 // プレイヤー１の関数名
        p2_func = "j27" + player2 ;                                 // プレイヤー２   〃

        p1_no = Number(player1);                                    // プレイヤー１の番号
        p2_no = Number(player2);                                    // プレイヤー２   〃

        plimg.src = "img/" + p1_func + ".png";                  // プレイヤー１の画像
        p2img.src = "img/" + p2_func + ".png";                  // プレイヤー２   〃

                p1name.innerHTML = "<font size=6>" + NAME[p1_no] + "</font>"; N[1] = NAME[p1_no];      // 名前と画像の表示
                p2name.innerHTML = "<font size=6>" + NAME[p2_no] + "</font>"; N[2] = NAME[p2_no];

        s1.value = window[ p1_func ];                           // プレイヤー１の関数をテキストボックスへ表示
        s2.value = window[ p2_func ];                           // プレイヤー２    〃

        PLAYER = 1;                                                     // 最初はプレイヤー１の順番
        if( p1_no == 0)                                             // プ１が「手操作入力」ならフラグを設定
                                T_FLAG = true;
        else
                                T_FLAG = false;

        msg2 = document.getElementById("msg2");                 // デバッグ用の表示領域ＩＤ
        ban_disp();                                                     // ゲーム盤の表示
    }

    function ban_disp()
    {
        var y,x,img,n,str="";
            var cnt=[60,2,2];                                                   // コマカウント用配列（空白、黒、白を初期設定）
        str = "<center><font size=5>" +  N[PLAYER] +"(" + k[PLAYER] + ") の番です</font></center><p>";// 順番表示
        cnt=[0,0,0];                    // コマカウント用配列

        for (y=1;y<=8;y++){
            for (x=1;x<=8;x++){
                cnt[ BAN[y][x] ]++;                                     // コマカウント
                switch( BAN[y][x] ){
                    case 0 : img = "green.jpg"; break;          // コマなし
                    case 1 : img = "black.jpg"; break;          // 黒コマ
                    case 2 : img = "white.jpg"; break;          // 白コマ
                }
                n = (y-1)*8+(x-1);                                      // イメージのＩＤ
                str += "<img id='" + n + "' src='" + img + "' width=" + W + " onClick='manual(this)' onmouseover='over(this)' onmouseout='out(this)'>";
            }
            str += "<br>";
        }
        str += "<table border=0 width=" + W*8 + ">";            // 黒白コマ 棒グラフ表示
        str += "<tr><td width=25%><font size=5>黒 " + cnt[1] + "</td><td><img src='img/black.png' height=30 width=" + cnt[1]*(W/10) +"></font></td></tr>";
        str += "<tr><td width=25%><font size=5>白 " + cnt[2] + "</td><td><img src='img/white.png' height=30 width=" + cnt[2]*(W/10) +"></font></td></tr>";
        str += "</td></tr></table>";

        msg = document.getElementById("msg");
        msg.innerHTML = str;                                    // 表示描き替え
                if(cnt[0] == 0 ){                               // 空白のコマがなくなったら終了処理
                    PLAYER = cnt[1]>cnt[2]?1:2;                 // 勝利者の判定
            str += "<center><font size=6>" +  N[PLAYER] +"(" + k[PLAYER] + ") の勝利です！！</font></center>";
                msg.innerHTML = str;
                exit;                   // 強制的に止める（エラー）
        }
        msg.innerHTML = str;
    }

    function check2()    // PLAYER が打てる場所があるかどうか判定
    {
        for(var y=1;y<=8;y++){
            for(var x=1;x<=8;x++){
                if( check(x,y) != 0 ) return true;              // 打てる場所がある true
            }
        }
        return false;                                           // 打てる場所がない false
    }

    function reverse(x,y)               // コマを反転する関数
    {
        var aite = PLAYER==1?2:1;                               // 相手のプレイヤー   黒:1 白:2

        for(var dy=-1; dy<=1; dy++){                            // 上下左右、斜めを調べる
            for(var dx=-1;dx<=1;dx++){
                if( dx!=0 || dy!=0){
                    var rx=x+dx; var ry=y+dy;
                    while( BAN[ry][rx] == aite){
                        rx += dx; ry += dy;
                        if( BAN[ry][rx] == PLAYER ){            // 挟めていたら
                            while( !(rx==x && ry==y) ){         // 戻りながら反転する
                                rx -= dx; ry -= dy;
                                BAN[ry][rx] = PLAYER;
                            }
                            break;
                        }
                    }
                }
            }
        }
    }

    function manual(obj)                // 手動入力（マウス）関数
    {

        var id = obj.id;                // クリックされた盤面の画像オブジェクトＩＤ
        var y = Math.floor(id/8)+1;     // ＩＤをｘ、ｙ座標に変換
        var x = Math.floor(id%8)+1;
        if( check(x,y) != 0 ){          // クリックされた場所に置ければ値を更新
            MX=x; MY=y;
            T_FLAG = false;             // シンキングの終了（フラグ）
        }
                                        // 置けなければ何もしない
```


```
function over(obj)                      // マウスオーバーの処理
{
            var id = obj.id;
    var y = Math.floor(id/8)+1;
    var x = Math.floor(id%8)+1;
    if( check(x,y) == 0) return;        // 置けない場所ならリターン
    obj.src = "red.jpg";                // 位置を示す画像（赤）を表示
}

function out(obj)                       // マウスアウトの処理
{
            var id = obj.id;
    var y = Math.floor(id/8)+1;
    var x = Math.floor(id%8)+1;         // 置けない場所ならリターン
    if( check(x,y) == 0) return;
    obj.src = "green.jpg";              // 空白の画像（緑）を表示
}

function check(x,y)                     // x,y 置くと何枚取れるかを返す関数
{

    if(BAN[y][x] != 0) return 0;        // 空白コマでなければリターン
    var cnt=0,ct;
    var aite = PLAYER==1?2:1;           // 相手のプレイヤー   黒:1 白:2

    for(var dy=-1; dy<=1; dy++){        // 上下左右、斜めを調べる
        for(var dx=-1;dx<=1;dx++){
            ct = 0;
            var rx=x+dx; var ry=y+dy;
            while( BAN[ry][rx] == aite ){   // ひとつのラインで反転できるコマ数 ct
                ct++;
                rx += dx; ry += dy;
            }
            if( BAN[ry][rx] == PLAYER ) cnt += ct;      // 合計 cnt に累計
        }
    }
    //if(cnt != 0) cnt++;  // 打ったコマを足す場合
     return cnt;
}

MX = -1; MY = -1;
function loop()
{
    if( T_FLAG == true ) return;                    // シンキングフラグ（T_FLAG）がオンならリターン

                clearInterval(tID1);                // loop() の繰り返しタイマをクリア
                //msg2.innerHTML =  ">> " + T_FLAG;
    var x,y;
    if( PLAYER == 1){                               // プレイヤー１の場合
        if( check2() == true ){
            if( p1_no == 0){                        //   手動入力なら
                x = MX; y = MY;
                T_FLAG = false;
            }
            else {                                  // 自動アルゴリズムなら
                T_FLAG = true;
                var xy = eval( p1_func + "();" );   // プレイヤー１のアルゴリズム関数を呼び出す
                T_FLAG = false;
                x= xy[0]; y=xy[1];                  // 返された手のｘｙ座標
            }
            BAN[y][x] = PLAYER;                     // 打った場所にコマをセット
            reverse( x , y );                       // 反転処理
        }
    }
    else {
        if( check2() == true ){                     // プレイヤー２の場合
            if( p2_no == 0){                        //   手動入力なら
                x = MX; y = MY;
                T_FLAG = false;
            }
            else {
                T_FLAG = true;
                var xy = eval( p2_func + "();" );   // プレイヤー２のアルゴリズム関数を呼び出す
                T_FLAG = false;
                x= xy[0]; y=xy[1];
            }
            BAN[y][x] = PLAYER;                     // 打った場所にコマをセット
            reverse( x , y );                       // 反転処理
        }
    }
    PLAYER = PLAYER==1?2:1;                         // 次の順番

    if(PLAYER == 2 && p2_no == 0) T_FLAG = true;    // 手動入力の時はシンキングフラグをオン
    if(PLAYER == 1 && p1_no == 0) T_FLAG = true;

    ban_disp();                                     // 盤面の表示
    loop_start();                                   // 次の繰り返しを呼ぶ
}

function loop_start()                               // loop を定期的に呼び出す処理
{
    tID1 = setInterval(loop,100);                   // 次の順番に渡すまでの時間 ms
}


</script>
</head>
```





対戦番号を「００」とした場合には手動入力モードとなり、マウスでコマの位置を指定する。自身のプログラム開発時に動作確認などの用途としても使用できる。

```
<body bgcolor="#BBFFBB">
<div id="but">
<font size=5>
先手（黒）<input type ="text" id="text1" size=2 style="font-size:30px;" value="40">
後手（白）<input type ="text" id="text2" size=2 style="font-size:30px;" value="00">
<input type ="button" id="button1" value=" 相手確定 " onClick="init()" style="font-size:30px;"><font size=7>  ⇒  </font>
<input type ="button" id="button2" value=" 対戦スタート " onClick="loop_start()" style="font-size:30px;"><p>
</div>

<table border=0 width=100%><tr>
<td valign=top align=center width=25% >
<div id="p1name"><font size=6>先手（黒）</div>
<img src="black.jpg" width=150> <img id="plimg" src="img/man.png" width=150>  <p>
<textarea id="source1" rows="20" cols="50" style=" font-size:12px; background: #eeeeee;">プレイヤー１のソースリスト</textarea>
</td>
<td width=40% align="center">
<div id="msg" >リバーシ</div>
</td>
<td valign=top  align=center width=25% >
<div id="p2name"><font size=6>後手（白）</div>
<img src="white.jpg" width=150> <img id="p2img" src="img/man.png" width=150 align=top><p>

<textarea id="source2" rows="20" cols="50" style="width=100%; font-size:12px; background: #eeeeee;">プレイヤー２のソースリスト</textarea>
</td>
</tr></table>

<div id="msg2">メッセージ２</div>
<textarea id="t_kekka" rows="30" style="width:10%; font-size:14px; background: #dddddd;">対戦結果</textarea>

</font>
<p>


</body>
</html>
```

人工知能オセロ大会

インクルードファイル（m18a00-40.js）

```
var NAME = [    ];              // 名前用の配列

                        // それぞれのオセロ関数をインクルード

document.write("<script type='text/javascript' src='h28c00.js'></script>");
document.write("<script type='text/javascript' src='h28c01.js'></script>");
document.write("<script type='text/javascript' src='j2802.js'></script>");
・・・
・・・
document.write("<script type='text/javascript' src='h28c38.js'></script>");
document.write("<script type='text/javascript' src='h28c39.js'></script>");
document.write("<script type='text/javascript' src='h28c40.js'></script>");
```

イメージファイル（m18a____.png）

対戦中に表示する自分の写真、または好きなキャラクター
などを次の形式で作成、保存する。

サイズ　：　１５０ｐｘ　×　１５０ｐｘ（正方形）
ファイル形式　：　ＰＮＧ
ファイル名は　：　ｍ１８ａ＿＿＿＿．ＰＮＧ

名簿番号２ケタ

保存先　：　img フォルダ内

フォルダの構成
othello-base.html　メインプロ
m18a00-40.js　インクルードファイル
m18a40.js　対戦相手サンプル
m18a___.js　自分のプログラム
（名簿番号）
--- img フォルダ（イメージ）
　　m18a___.png　自分の画像
　　（名簿番号）
　　m18a40.png　サンプル画像

自分のプログラム と 対戦相手のプログラム

（自分で入力し、ベースプログラムと同じフォルダへ保存する）

ファイル名は「ｍ１８ａ＿＿＿＿．ｊｓ」とする。

名簿番号２ケタ

作成する関数は　指し手のｘ、ｙ座標（配列）を返すものとする。

自分の例（m18a41.js）　４１番の場合


```
NAME[41] ="乱れ打之介";
function m18a41()
{
    var x,y;
    ・・・
    return [ x , y ];     // 戻り値として配列を返す［ x座標、y座標 ］
}
```

対戦相手の例（m18a42.js）　４２番の場合

```
NAME[42] ="上調べ太郎";
function m18a42()
{
    var x,y;
    ・・・
    return [ x, y ];
}
```

特別な番号

対戦番号を「００」とした場合には手動入力モードとなり、マウスでコマの位置を指定する。自身のプログラム開発時に動作確認などの用途としても使用できる。

オセロアルゴリズム

プログラム例

・関数名は「ｍ１８ａ」＋自分の名簿番号２ケタ　m18a01 m18a20 m18a32

・何番と何番を対戦させるのかはテキストボックスに入力

・盤面の情報は配列 **BAN[][]** に記録されていく（0: 空　1: 黒　2: 白）

・打つ場所がない場合はメインプログラムで自動的にパスとなり、
相手に順番が移り続行される。（両方ともパスの場合はエラー…）

ベースプログラムで用意されている変数（配列）と関数

（自分のプログラム関数の中で参照できる）

BAN[ ][ ]　**盤面の情報**（0: 空　1: 黒　2: 白）

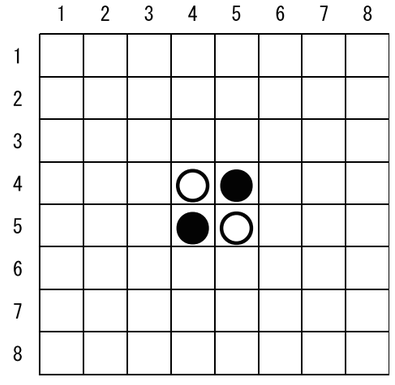
PLAYER　**自分のコマ（1: 黒　2: 白）** 自分の関数内で参照する

応用　aite = PLAYER==1?2:1;

check(x, y)　**ｘ、ｙ座標に自分のコマを打った時、相手のコマを何枚ひっくり返すことができるかを返す。**
（戻り値が０ならその座標には置けないということ）

	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	**2**	**1**	0	0	0	0
5	0	0	0	0	**1**	**2**	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0

２次元配列



盤面イメージ

例１　乱数で発生させた座標に打つ

m18a41.js

```
NAME[41] =″乱れ打之介″;
function m18a41()
{
    var x,y;
    while(1){                       // 打てるところがあるまで探す
        x = Math.floor(Math.random()*8)+1;
        y = Math.floor(Math.random()*8)+1;
        if( check(x,y) != 0 )  // その場所に打てれば…
            return [x,y];       // ｘ、ｙ座標を配列として返す
    }
}
```

※打てるところがあるかどうか（パスかどうかは）メインプログラムの中で判断しているので、この関数が呼び出されれば、必ずその順番のプレイヤーに盤面のどこかに打てる場所があるという事。打てる場所があれば、それが悪手でも必ず打たなくてはいけないのがオセロのルールである。

例２　盤面の左上から調べて、最初に打てる座標に打つ

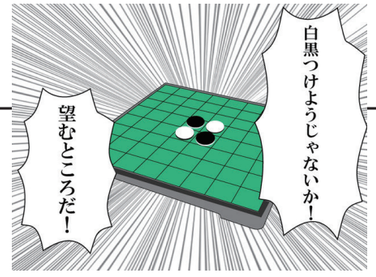
m18a42.js

```
NAME[42] =″上方調太郎″;
function m18a42()
{
    var x,y;
    var jibun = PLAYER;             // 自分のコマ（1: 黒　2: 白）
    var aite = PLAYER==1?2:1;       // 相手のコマ
    for( y=1; y<=8; y++ ){          // 左上から順番に調べて…
        for( x=1; x<=8; x++ ){
            if( check(x,y) != 0 ) // 打てるところがあれば
                return [x,y];     // xy 座標を配列として返す
        }
    }
}
```

例３　相手のコマを一番多くひっくり返せる場所に打つ

m18a43.js

```
NAME[43] =″沢山取太郎″;
function m18a43()
{
    var x,y,max_x, max_y, ct1, maxcnt=0;   // 仮の最大枚数
    var jibun = PLAYER;              // 自分のコマ（1:黒　2:白）
    var aite = PLAYER==1?2:1;        // 相手のコマ

    for( y=1; y<=8; y++ ){               // 全てのマスを調べる
        for( x=1; x<=8; x++ ){
            ct1 = check(x,y);        // ひっくり返せる相手の枚数
            if( ct1 > maxcnt ){  // 現在の最大より多かったら
                maxcnt = ct1;            // 枚数を記憶
                max_x=x; max_y=y;        // 座標を記憶
            }
        }
    }
    return [ max_x , max_y ];        // 最多枚数の座標を返す
}
```



ＡＩ対決オセロ大会　提出課題

①自分のプログラムリストと
　実行結果（対戦相手は自由）を印刷
②ＡＩ対決オセロ大会の感想

} これで ＡＴサイズ１枚

③プレゼン資料をＡ４用紙１枚に収まるようにして印刷

オセロのアルゴリズム

番　氏名

アルゴリズムのタイトル