ななちゃんのIT教室

プチプログラム集

by nara.yasuhiro@gmail.com

1 ページに収まるような小さいプログラム例で
プログラミングの楽しさを発見しましょう

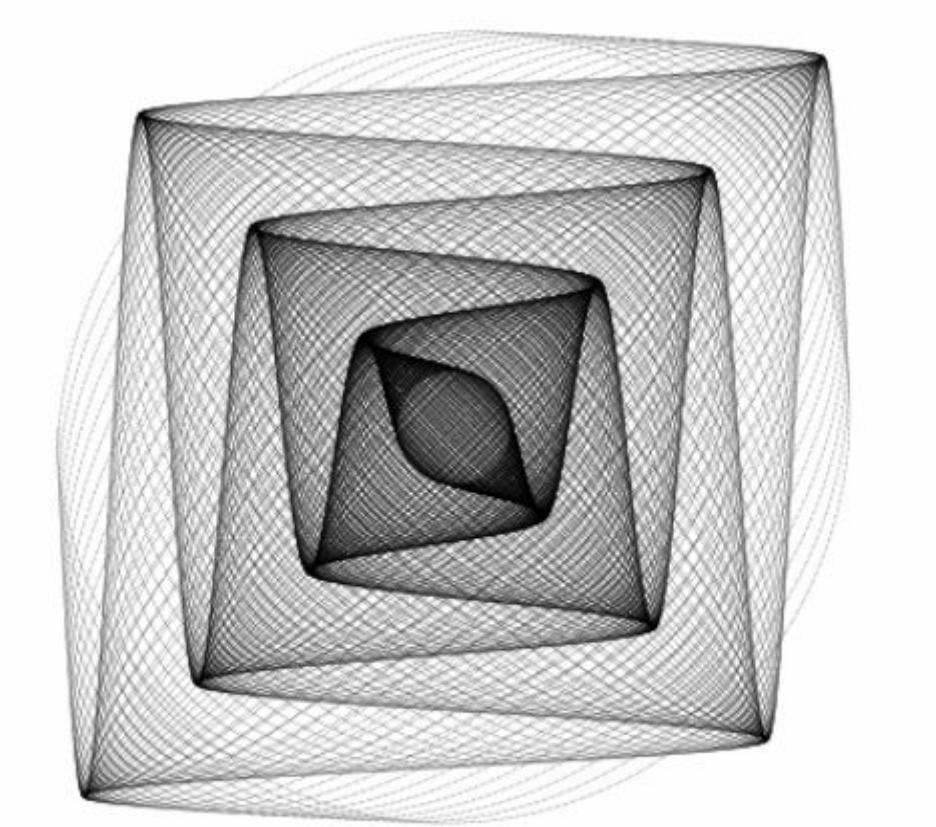
第 2.2 版　2017 年 6 月 11 日

フリー素材
http://freeillustration.net

1. リサージュ図形

x 軸と、y 軸に、周波数や位相の異なる正弦波を入力することによって得られる図形。周波数が同じだと円、倍数関係にあると「８」の字や「ねじりパン」のような、ねじれた円になるが、最小公倍数が大きいと複雑な図形になる。原理的には、周波数の比が無理数の場合は閉曲線にはならず、軌道は有限の平行四辺形領域を稠密に埋める。（コンピュータは有限桁数しか扱えないので、厳密に無理数を扱うことはできない）

1855 年にフランスの物理学者ジュール・アントワーヌ・リサジュー（J.A. Lissajous, 1822-1880）が考案したとされる。1815 年にナサニエル・バウディッチ(Nathaniel Bowditch) の先行的研究があるので、バウディッチ曲線、ボウディッチ曲線と呼ばれることもある。



```
<!DOCTYPE html>
<html lang="ja">
  <head>
    <meta charset="utf-8">
    <title>Lissajous</title>
  </head>
  <body>
    <canvas id="canvas" width="400" height="400"></canvas><br>
    F1: <input type=input value=79 id=F1>
    F2: <input type=input value=80 id=F2>
    <input type=button value=go onclick=go()>
    <script>
var canvas = document.getElementById('canvas');
var ctx = canvas.getContext('2d');
var F1p = document.getElementById("F1");
var F2p = document.getElementById("F2");

go();

function go() {
    ctx.clearRect(0,0,400,400);
    var lim = 7;
    for (t=1*Math.PI; t<=lim*Math.PI; t+=0.0001) {
        var x = 0.9 * Math.cos(Number(F1p.value) * t) * t/(lim*Math.PI);
        var y = 0.9 * Math.sin(Number(F2p.value) * t) * t/(lim*Math.PI);
        dot(x,y);
    }
}

function dot(x,y) {
    ctx.fillRect(200+x*200, 200-y*200,0.5,0.5);
}
    </script>
  </body>
</html>
```

2. ランレングス圧縮法

ランレングス圧縮（run length compression、RLE、Run Length Encoding、連長圧縮）法は、最も基本的な情報圧縮アルゴリズムの一つで、可逆圧縮方式の一つ。連続して現れるデータを、繰り返しの回数で置き換えることによりデータ量を削減する方式。

連続したデータを、そのデーター つ分と、連続した長さで表現することで圧縮する。例えば、「A A A A A B B B B B B B B B A A A」は「A 5 B 9 A 3」と表せる。これは、A が5 回続き、そのあとに B が 9 回、そしてA が 3 回続いていることを表している。3 文字以上連続するデータは元の長さより短くなる。

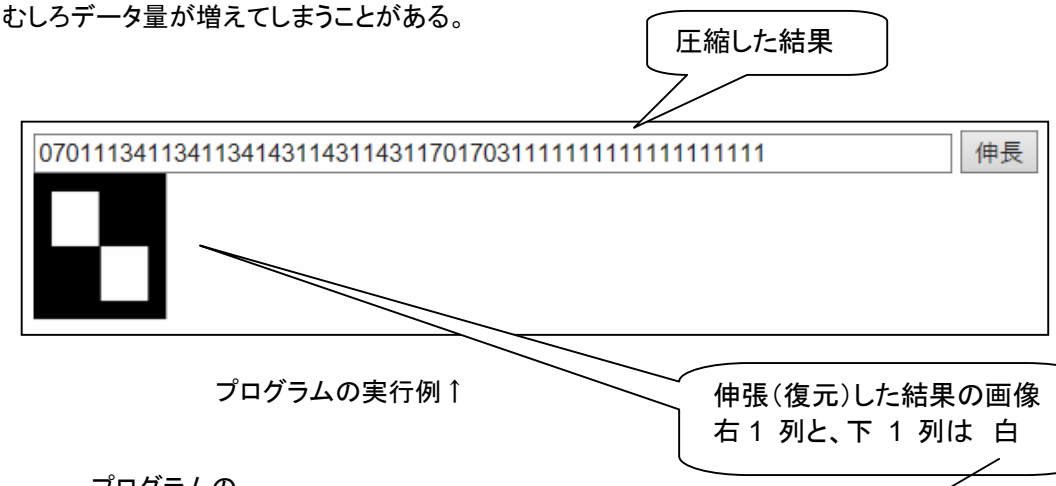
さらに、データが 2 種類（A＝白 と B＝黒）だけで、最初に A が来ることにしておけば、上記の「A 5 B 9 A 3」は、「5 9 3」だけで表せる。こういったことから、白と黒（モノクロ）のファクシミリ用の情報圧縮方法として利用されている。

このルールに従ったとき、B が最初に見つかった場合は、最初に A が 0 回連続していることにすれば良い。例えば、「B B B A A A A A B B B B B A A A」は「0 3 5 5 3」で表せることになる。

「3 5」が、35 回なのか、3 回 + 5 回なのか区別できないので、回数は 1 桁に制限すると、35 回は、9 回 + 9 回 + 9 回 + 8 回で、間に B の 0 回がはさまっていると考え、「9 0 9 0 9 0 8」とする。さらに、数を、2 進数 3 桁（3 ビット）に対応付けができるように、 0 ～ 7 に制限すると、「7 0 7 0 7 0 7 0 7」となる。

ここで作る JavaScript プログラムでは、データが、0 が 白、1 が 黒、とし、9 × 9 のマス目の画像であると考える。

白や黒が連続する部分が多ければデータ量は小さくなり、白と黒が1個ずつ交互に現れるとデータ量は大きくなる。データ量が小さい画像は 「冗長度が高い画像」、データ量が大きい画像は 「冗長度が低い画像」 という。冗長度とは「無駄が多い」、「含まれる情報量が少ない」という意味の言葉。冗長度の小さい、情報量の多い画像は、圧縮すると、むしろデータ量が増えてしまうことがある。



プログラムの

```
var dt = "000000001" +          // 0 = black, 1 = white
         "011100001" +
         "011100001" +
         "011100001" +
         "000011101" +
         "000011101" +
         "000011101" +
         "000000001" +
         "111111111";
```

の部分を書き換えると、他の画像データになるが、 1 行 9 文字で、9 行になるようにしないと、正しく表示できなくなる。「//」 以下は、コメント。

```
<!DOCTYPE html>
<html>
   <head>
      <meta charset="utf-8">
      <title>Run-length encoding</title>
   </head>
   <body>
      <input type=text id=in size=80>
      <input type=button onClick=decode() value="伸長"><br>
      <canvas id=canvas width=100 height=100></canvas>
      <script>
var canvas = document.getElementById('canvas');
var ctx = canvas.getContext('2d');
var inp = document.getElementById('in');
var x=0, y=0, c=1, cnt=0, line="";
var dt = "000000001" +                  // 0 = black, 1 = white
            "011100001" +
            "011100001" +
            "011100001" +
            "000011101" +
            "000011101" +
            "000011101" +
            "000000001" +
            "111111111";

function encode() {
    for (var k=0; k<100; k++) {
        if (c == dt [ k ] )   {
            cnt++;
            if (cnt > 7) { line += "70" ; cnt=1; }
        }
        else { line += cnt; cnt=1; c = 1 - c; }
    }
    line += cnt; inp.value = line;
}
encode();

function decode() {
    var c = 1;
    var line = inp.value;
    for (var j in line)   {
       var n = line [ j ] ;
       for (var i = 0; i < n; i++ ) push(c);
       c = 1 – c;
    }
}

function push(c) {
    if (y > 80) return;
    if (c==0) ctx.fillRect(x, y, 10, 10);
    x += 10;
    if (x > 80) { x = 0; y += 10; }
}
      </script>
   </body>
</html>
```

3. スネークゲーム

タイピング練習プログラムのつもり。画面上、黒い線が伸びてゆく、はじめは、左上から、右方向に延びてゆく。キーボードの「u」を押すと「上 (up)」、「d」を押すと「下 (down)」、「r」を押すと「右 (right)」、「l」を押すと「左 (left)」に方向を変える。画面のへりにぶつかったり、すでに黒くなっている部分にぶつかるとゲーム終了。50 秒以内に、何歩すすめるか、というゲーム。

注意：このプログラムは、マイクロソフトの Edge ブラウザでは正しく動作しない場合があります。

実行結果

```
<!DOCTYPE html>
 <html>
  <head>
    <meta charset="utf-8">
    <title>スネークゲーム</title>
    <style> canvas { border: solid 2px; } </style>
  </head>
  <body>
  <canvas id=canvas width=100 height=100></canvas>
   右＝r、左＝l、上＝u、下＝d<br>
   得点<input type=text id=out><br>
   <script>
var outp = document.getElementById('out');
outp.value = 0;
var canvas = document.getElementById('canvas');
var ctx = canvas.getContext('2d');
var n=0, x=5, y=5, dx=1, dy=0;
var tid;

ctx.fillStyle = "white";
ctx.fillRect(0,0,100,100);
ctx.fillStyle = "black";

onkeydown = function (e) {
    if         (e.keyCode==85) { dy=-1; dx=0; } // u
    else if (e.keyCode==68) { dy=1;   dx=0; } // d
    else if (e.keyCode==82) { dx=1;   dy=0; } // r
    else if (e.keyCode==76) { dx=-1; dy=0; } // l
}

function step() {
    if ((n++ < 500) && (n % 5 == 0)) outp.value = n;
    x+=dx; y+=dy;
    if ((n >= 500) ||
          (x<=0) || (x>=90) || (y<=0) || (y>=90) ||
          ((dx==-1||dy==-1)&&(ctx.getImageData(x,   y,   1,1).data[0]==0)) ||
          ((dx== 1||dy==-1)&&(ctx.getImageData(x+9,y,   1,1).data[0]==0)) ||
          ((dx==-1||dy== 1)&&(ctx.getImageData(x,   y+9,1,1).data[0]==0)) ||
          ((dx== 1||dy== 1)&&(ctx.getImageData(x+9,y+9,1,1).data[0]==0)) ) {
        clearInterval(tid);
        document.onkeydown = null;
        return;
    }
    ctx.fillRect(x,y,10,10);
}

tid = setInterval(step, 100);
</script>
  </body>
 </html>
```

ASCII コードとは別物のキーボードコード

タイムアウト判定

画面外判定

衝突判定

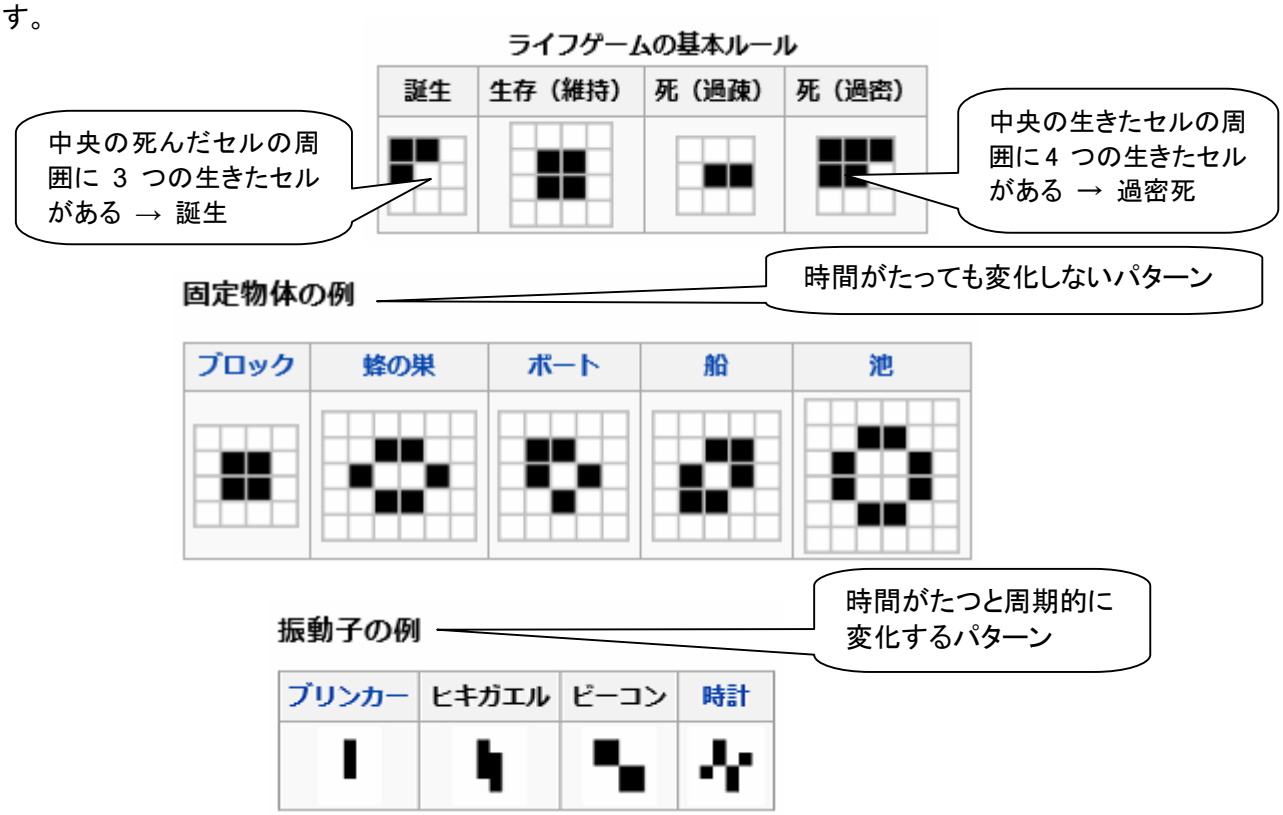
4. ライフゲーム（Game of Life）

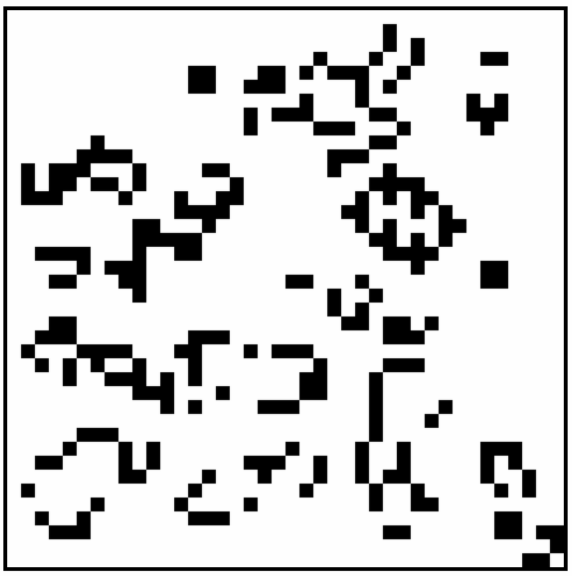
1970 年にイギリスの数学者ジョン・ホートン・コンウェイ (John Horton Conway) が考案した生命の誕生、進化、淘汰などのプロセスを簡易的なモデルで再現した、セルオートマトン方式のシミュレーションゲーム。

ライフゲームでは初期状態のみでその後の状態が決定される。碁盤のような格子があり、一つの格子はセル（細胞）と呼ばれる。各セルには 8 つの近傍のセルがある (ムーア近傍) 。各セルには「生」と「死」の 2 つの状態があり、あるセルの次のステップ（世代）の状態は周囲の 8 つのセルの今の世代における状態により決定される。

セルの生死は次のルールに従う。
誕生：死んでいるセルに隣接する生きたセルがちょうど 3 つあれば、次の世代が誕生する。
生存：生きているセルに隣接する生きたセルが 2 つか 3 つならば、次の世代でも生存する。
過疎：生きているセルに隣接する生きたセルが 1 つ以下ならば、過疎により死滅する。
過密：生きているセルに隣接する生きたセルが 4 つ以上ならば、過密により死滅する。

下に中央のセルにおける次のステップでの生死の例を示す。生きているセルは■、死んでいるセルは□で表す。





実行例 （初期パターンは乱数で決めているので、毎回結果が異なる）

```
<!DOCTYPE html>
 <html lang="en">
 <head>
    <meta charset="UTF-8">
    <title>lifeGame</title>
    <style> canvas { border: solid; } </style>
 </head>
 <body>
    <canvas id="canvas1" width="400" height="400"></canvas>
 </body>
    <script> // http://blog.csdn.net/qq_22187867/article/details/51337190
var canvas=document.getElementById('canvas1');
var ctx=canvas.getContext("2d");
var i, j, ii, jj, points=[], npoints=[];

for (i = 0; i < 42; i++) {
    points[i] = []; npoints[i] = [];
    for (j = 0; j < 42; j++) {
        points[i][j] = ((i>0)&&(i<41)&&(j>0)&&(j<41)&&(Math.random()<0.3))?1:0;
        npoints[i][j] = 0;
}   }

setInterval( function() {
    ctx.clearRect(0,0,400,400);
    for (i = 1; i < 41; i++) for (j = 1; j < 41; j++)
        if(points[i][j] == 1) ctx.fillRect((i-1)*10,(j-1)*10,10,10);
    for (i = 1; i < 41; i++) for (j = 1; j < 41; j++) {
        var sum=0;
        for (ii=-1; ii<=1; ii++) for (jj=-1; jj<=1; jj++)
            if (points[i+ii][j+jj] == 1) sum++;
        npoints[i][j] = ((sum==3) || (sum==4 && points[i][j]==1))?1:0;
    }
    for (i = 1; i < 41; i++) for (j = 1; j < 41; j++)
        points[i][j] = npoints[i][j];
}, 100);
   </script>
 </body>
</html>
```

0、41 は不変化枠
なので 1～40

5. 基数変換

10進数を入力すると、それを、8進数、16進数などに変換できます。変化先の数を修正すると、それに対応する10進数のほうがが自動的に変化します。→をクリックすると変換の方向を逆にします。

10進 10 → 16 進 a

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>応用（基数変換）</title>
    <style type=text/css> input { font-size:150%; }</style>
  </head>
  <body>
    10 進
    <input type=text id=dec value=0 style="text-align:right;" onchange=right()>
    <input type=text id=dir value="→" readonly onclick=chdir() size=1>
    <input type=text id=rdx value="16" size=2   onchange=recalc()>進
    <input type=text id=tgt value=0 style="text-align: right;" onchange=left()>
    <script>
var decp, tgtp, dirp, rdxp;
    decp = document.getElementById("dec");
    tgtp = document.getElementById("tgt");
    dirp = document.getElementById("dir");
    rdxp = document.getElementById("rdx");

function right() {
    dirp.value = "→";
    tgtp.value = parseInt(decp.value).toString(rdxp.value);
}

function left() {
    dirp.value = "←";
    decp.value = parseInt(tgtp.value,rdxp.value);
}

function chdir() {
    if(dirp.value == "←") {
        dirp.value = "→";
        tgtp.value = parseInt(decp.value).toString(rdxp.value);
    }
    else {
        dirp.value = "←";
        decp.value = parseInt(tgtp.value,rdxp.value);
}   }

function recalc() {
    if(dirp.value == "→") {
        tgtp.value = parseInt(decp.value).toString(rdxp.value);
    }
    else {
        decp.value = parseInt(tgtp.value,rdxp.value);
}   }
    </script>
  </body>
</html>
```

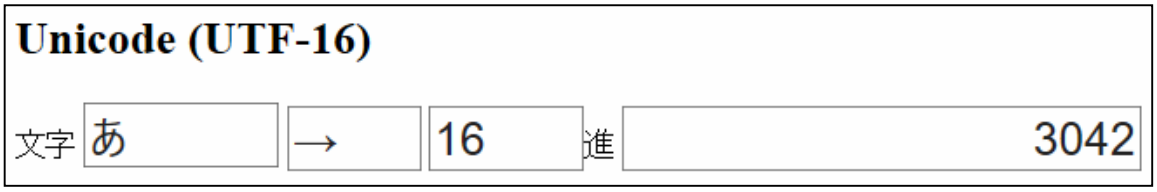
10 進→他

10 進←他

基数変換

6. 文字コード

文字を入力すると、その文字コードを表示します。表示は、10進数、16進数を指定できます。文字コード欄を修正すると、それに対応する文字を表示します。



```
<!DOCTYPE html>
<html>
   <head>
      <meta charset="utf-8">
      <title>応用（文字コード）</title>
      <style type=text/css> input { font-size:150%; }</style>
   </head>
   <body>
   <h2>Unicode (UTF-16)</h2>
     文字<input type=text id=dec value="あ" style="text-align:left;" onchange=right() size=4>
     <input type=text id=dir value="→" readonly onclick=chdir() size=1>
     <input type=text id=rdx value="16" size=2    onchange=recalc()>進
     <input type=text id=tgt value="3042" style="text-align: right;" onchange=left()>
     <script>
function chCode(s) { return s.charCodeAt(0); }
function toChar(code) { return String.fromCharCode(code); }
var decp, tgtp, dirp, rdxp;
     decp = document.getElementById("dec");
     tgtp = document.getElementById("tgt");
     dirp = document.getElementById("dir");
     rdxp = document.getElementById("rdx");
function right() {
     dirp.value = "→";
     tgtp.value = chCode(decp.value).toString(rdxp.value);
}
function left() {
     dirp.value = "←";
     decp.value = toChar(parseInt(tgtp.value,rdxp.value));
}
function chdir() {
     if(dirp.value == "←") {
          dirp.value = "→";
          tgtp.value = chCode(decp.value).toString(rdxp.value);
     }
     else {
          dirp.value = "←";
          decp.value = toChar(parseInt(tgtp.value,rdxp.value));
}   }
function recalc() {
     if(dirp.value == "→") {
          tgtp.value = chCode(decp.value).toString(rdxp.value);
     }
     else {
          decp.value = toChar(parseInt(tgtp.value,rdxp.value));
}   }
     </script>
   </body>
</html>
```

文字コードを得る

基数解釈

基数変換

文字コードを文字に

7. タイピング練習

a ～ z の文字がランダムに表示される。2 秒以内にそのキーを押すと「◎」が表示されて得点になる。違うキーだと表示文字が消える。10 点満点。

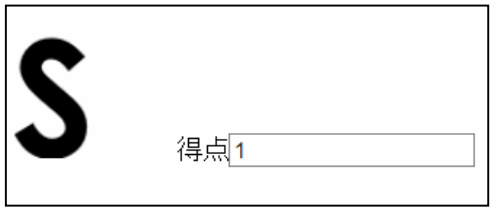
```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
  </head>
  <body>
    <canvas id="canvas" width="100" height="100">描画領域</canvas>
    得点<input type=text id=out>
    <script>
var outp = document.getElementById("out");
outp.value = 0;
var canvas = document.getElementById("canvas");
var ctx = canvas.getContext("2d");
ctx.font='bold 100px Century Gothic,"Hiragino Kaku Gothic ProN"';
var c;
var n=0;

function func1() {
    if (n++ >= 10) {
        ctx.fillStyle="rgb(255,255,255)";
        ctx.fillRect(0, 0, 100, 100);
        ctx.fillStyle="rgb(0,0,0)";
        ctx.fillText(" ",0, 99);
        clearInterval(intervalID);
        document.onkeydown = null;
        return;
    }
    var rand = Math.floor(Math.random()*26);
    c = "A".charCodeAt(0) + rand;
    var str = String.fromCharCode(c);
    ctx.fillStyle="rgb(255,255,255)";
    ctx.fillRect(0, 0, 100, 100);
    ctx.fillStyle="rgb(0,0,0)";
    ctx.fillText(str,0, 99);
}

document.onkeydown = function (e)  {
    if(!e) e = window.event;
    ctx.fillStyle="rgb(255,255,255)";
    ctx.fillRect(0, 0, 100, 100);
    ctx.fillStyle="rgb(0,0,0)";
    if (e.keyCode == c) {
        outp.value++;
        ctx.fillText("◎",0, 95);
    }
    else {
        ctx.fillText(" ",0, 95);
    }
    c = 0;
}

var intervalID = window.setInterval(func1, 2000);
    </script>
  </body>
</html>
```

注意：このプログラムは、マイクロソフトの Edge ブラウザでは正しく動作しない場合があります。



0～25 の乱数発生

A～Z のコードに変換

文字コードを文字に変換

文字を表示

正しい文字入力の場合

間違った文字入力の場合

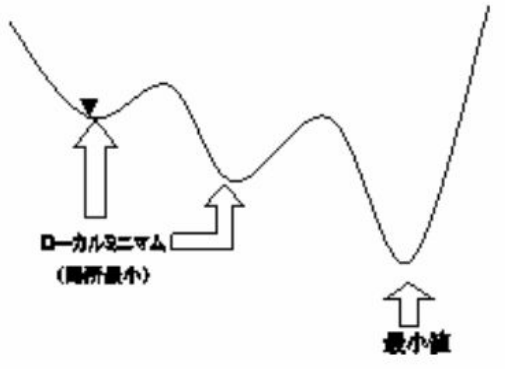
8. 巡回セールスマン問題 (Traveling Salesperson Problem、TSP)

セールスマンがある街から出発し、N個の都市を1回ずつ通って、元の街に戻ってくる最短経路を求める問題。組み合わせ最適化問題の一つ。旅商人問題、Hamiltonian game とも言う。運送会社の配送ルートの計画、VLSI の設計などの場面で利用されている。もともとは米ランド社が 1940 年代なかばに提出した懸賞問題。ランド社の Robinson が 1949 年に書いた論文の中で、この懸賞の存在について述べているという。

都市数を n とすると、n! / 2n 種類の巡回経路がある(n! 個の経路のうち、反対方向のものを除き(÷2)、出発点が特定点以外のものを除く(÷n)ので)。10 都市のとき、組み合わせ総数は 181,440、15 都市のとき約 400 億、30 都市のとき、4.42×10 の 30 乗。計算速度 10 tara FLOPS の計算機を用いて、25 京年(250,000 億年)以上かかる。数十都市になるとすべての巡回経路を調べることは計算量的に難しく、厳密解を求めることはできない。そこで、近似アルゴリズムで求めることになる (最適である保証はないが、より良い解を検索)。 近似アルゴリズムにはいくつかの種類がある。それぞれに対し、いろいろなプログラミング上の工夫が考えられている。 巡回セールスマン問題は、量子コンピュータが得意とする問題でもある。

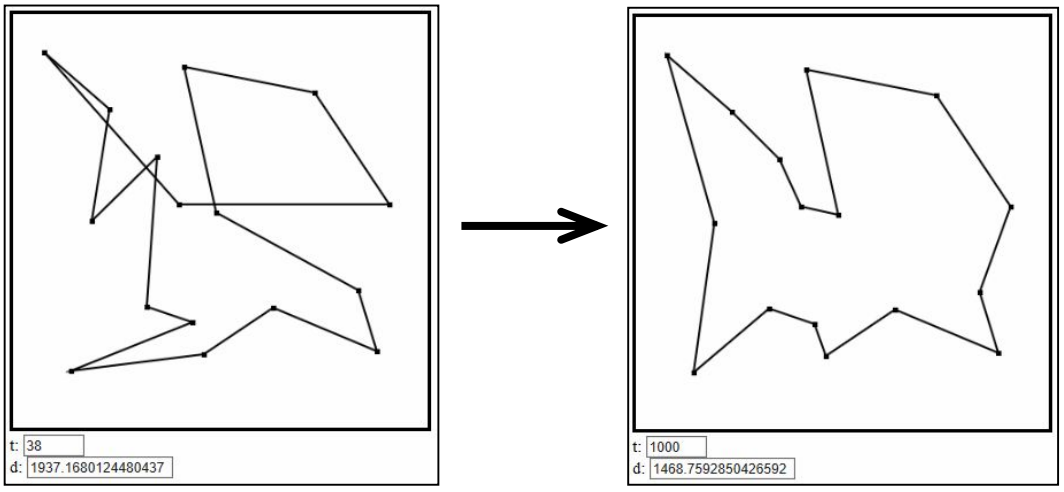
(1) ランダムサーチ (Random Search)
ランダムに経路を作成し、それまででもっとも良い解と比較。並列処理が可能。全体的な探索が行なえ、計算時間を無視すれば必ず最適解を求めることが出来る。それまでの探索の結果が生かされず、非効率的のため、大きな問題
では時間的に不向き。

(2) 山登り法 (Hill Climbing、HC)
初期経路の近傍を探索し、解が改善されれば、
それに置き換えてゆく。探索が局所的に良い方に
しか進行しないので、局所解に陥りやすくなる。



(3) 焼きなまし法 (Simulated Anealing、SA)
溶解状態にある物質を徐々に冷却して結晶状態に
するプロセスから ヒントを得たアルゴリズム。初期
経路の近傍を確率的に探索するが、最初は、この範囲を大きくし、少しずつ減らしてゆく。

(4) 遺伝的アルゴリズム(Genetic Algorithm、GA)
生物の進化をモデルとした手法。選択(淘汰)、突然変異、交叉、といった遺伝的操作を用いて 問題を解く。初期経路の近傍を確率的に探索(選択、淘汰)するが、たまに大きくジャンプ(突然変異)。二つの経路候補間で、パターンの部分的交換を行う(交差)。



初期の訪問順序　　発見した より良い訪問順序

巡回セールスマン問題のプログラム。焼きなまし法。

16 地点。16 地点の座標乱数で決定し agnts [ ] に記憶。訪問順序は vectr [ ] に記憶し、経路長がより短いものをみつけ、順次書き換えてゆく。

```
<!DOCTYPE html>
<html><head><meta charset="utf-8"></head><body>
      <canvas id="canvas" width="400" height="400" style="border:solid"></canvas><br>
      t: <input type=text id=tm value=0 size=4><br>
      d: <input type=text id=dist value=0 size=18><br>
      <script>
var tmp      = document.getElementById('tm');
var distp    = document.getElementById('dist');
var canvas   = document.getElementById('canvas');
var ctx      = canvas.getContext('2d'); ctx.lineWidth = 2;
var NofPnts = canvas.width, NofAgnts = 16;
var agnts = new Array(NofAgnts); for (i=0; i<NofAgnts; i++) agnts[i] = new agent(0,0);
var vectr = new Array(NofAgnts); for (i=0; i<NofAgnts; i++) vectr[i] = i;
var bckup = new Array(NofAgnts); for (i=0; i<NofAgnts; i++) bckup[i] = i;
for (var i in agnts) {
    while(true) {
        var x = Math.floor(Math.random()*NofPnts*0.9) + Math.floor(NofPnts*0.05);
        var y = Math.floor(Math.random()*NofPnts*0.9) + Math.floor(NofPnts*0.05);
        var ng = 0;
        for (j in agnts) { if ((agnts[j].x == x) && (agnts[i].y == y)) ng = 1; }
        if (ng == 0) break; }
    agnts[i].x = x; agnts[i].y = y; }
var tm = 0; tmp.value = tm; disp(); var t = 1000; var tid = setInterval(step,10);
function agent(x1,y1) { this.x = x1; this.y = y1; }
function save()      { for (i in bckup) bckup[i] = vectr[i]; }
function restore()   { for (i in vectr) vectr[i] = bckup[i]; }
function disp() {
    ctx.clearRect(0,0,NofPnts,NofPnts);
    ctx.beginPath();
    ctx.moveTo(agnts[vectr[vectr.length-1]].x, agnts[vectr[vectr.length-1]].y);
    for (var i in agnts) {
        var ag = agnts[vectr[i]];
        ctx.lineTo(ag.x, ag.y); ctx.fillRect(ag.x-2, ag.y-2, 5, 5); }
    ctx.stroke(); distp.value = calcdist(); }
function calcdist() {
    var d = 0; var x1 = agnts[vectr[vectr.length-1]].x; var y1 = agnts[vectr[vectr.length-1]].y;
    for (var i in vectr) {
        var ag = agnts[vectr[i]];
        d += Math.sqrt(Math.pow(ag.x-x1,2) +Math.pow(ag.y-y1,2));
        x1 = ag.x; y1 = ag.y; }
    return d; }
function swap() {
    var n1 = Math.floor(Math.random()*NofAgnts);
    while(true) { var n2 = Math.floor(Math.random()*NofAgnts); if (n1 != n2) break; }
    var vs = vectr[n1]; vectr[n1] = vectr[n2]; vectr[n2] = vs; }
function step() {
    var d1=calcdist(); save();
    for (var i=3-Math.floor(Math.log10(1001-t)); i > 0; i--) swap(); // Simulated Anealing
    var d2 = calcdist(); if (d1 < d2) restore(); disp();
    if (--t<=0) clearInterval(tid); tmp.value = ++tm; }
</script></body></html>
```

地点座標

訪問順序

地点座標を乱数で決定

10 ms 毎に step を呼び出す

地点座標オブジェクト

地点と経路を表示

総経路長を計算

訪問順序 2 点交換

短経路なら更新、長経路なら破棄

遺伝的アルゴリズム（突然変異のみ）の場合の差し替え

```
    if (Math.random()<0.01) for (var i=0; i<10; i++) swap(); // Genetic Algorithm
    else                    swap();
```

1%の確率で突然変異

普段は僅かな変異

9. パーサ (parser)

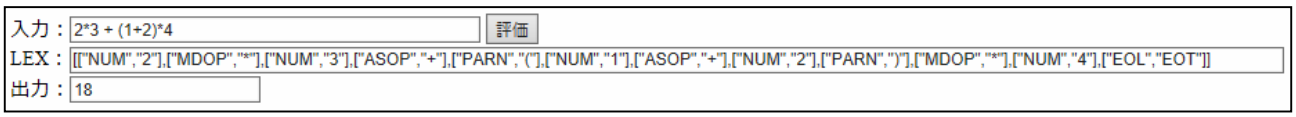
プログラミング言語のソースコードが構文規則に則っているかどうかを判定する（構文解析、parse）ために用いるプログラム。構文解析器、 パーザ、パーサー。通常は、字句解析器（lexer）によって字句レベルでの分析（語句解析、lexical analysis））が行われた後に実行される。

パーサ単体だけではソースコードのエラーチェックの機能しか持たないが、機械語などのコード生成機能を付加するとコンパイラになり、値の評価機能を付加するとインタプリタになる。コンパイラやインタプリタが利用できるツリー構造を出力する場合もある。JavaScript の場合、インタプリタ本体は eval()、式の評価だけなら JSON.parse() の形で利用できる。

この原理を示す、限定機能のプログラムを作る。整数定数の加減算式を扱える。ベースとなる文法は、

```
式       := 加減算式
加減算式 := 乗除算式 + 加減算式 | 乗除算式 - 加減算式 | 乗算式
乗除算式 := 項       + 乗除算式 | 項       - 乗除算式 | 項
項       := （式） | +項 | -項 | 数値
```

とする。「 | 」は、「または」の意味。



123
[["NUM","123"],["EOL","EOT"]]
123

123+111
[["NUM","123"],["ASOP","+"],["NUM","111"],["EOL","EOT"]]
234

123 + 111 ;
[["NUM","123"],["ASOP","+"],["NUM","111"],["EOL",";"],["EOL","EOT"]]
234

1*2+3*4
[["NUM","1"],["MDOP","*"],["NUM","2"],["ASOP","+"],["NUM","3"],["MDOP","*"],["NUM","4"],["EOL","EOT"]]
14

2*3+(1+2)*4
[["NUM","2"],["MDOP","*"],["NUM","3"],["ASOP","+"],["PARN","("],["NUM","1"],["ASOP","+"],["NUM","2"],
["PARN",")"],["MDOP","*"],["NUM","4"],["EOL","EOT"]]
18

```
<!DOCTYPE html>
<html><head><meta charset="utf-8"></head><body>
入力:<input type=text id=in size=50 autofocus>
        <input type=button value="評価" onclick=ev()><br>
LEX:<input type=text id=lx size=160><br>
出力:<input type=text id=out><br>
<script>
var inp  = document.getElementById('in');
var outp = document.getElementById('out');
var lxp  = document.getElementById('lx');
var itext, iseq;

function ev() {
    outp.value = lxp.value = ""; itext = inp.value; iseq = [];
    if (lex() == false) return;
    lxp.value = JSON.stringify(iseq); outp.value = parse(); inp.focus(); }
function parse() {
    var v = expr(0);
    if (iseq[v[1]][0] != "EOL") { alert("parse error(eol): " + JSON.stringify(iseq[v[1]]));
                                  return undefined; }
    return v[0]; }
function expr(is) { return asexpr(is); }
function asexpr(is) {
    var v = mdexpr(is);
    if (iseq[v[1]][1] == "+") { var v2 = asexpr(v[1]+1); return [ v[0] + v2[0], v2[1] ]; }
    if (iseq[v[1]][1] == "-") { var v2 = asexpr(v[1]+1); return [ v[0] - v2[0], v2[1] ]; }
    return v; }
function mdexpr(is) {
    var v = term(is);
    if (iseq[v[1]][1] == "*") { var v2 = mdexpr(v[1]+1); return [ v[0] * v2[0], v2[1] ];   }
    if (iseq[v[1]][1] == "/") { var v2 = mdexpr(v[1]+1); return [ v[0] / v2[0], v2[1] ]; }
    return v; }
function term(is) {
    var v;
    if (iseq[is][0] == "NUM") return [Number(iseq[is][1]), is+1];
    if (iseq[is][1] == "+")     return [ term(is+1)[0],term(is+1)[1]];
    if (iseq[is][1] == "-")     return [-term(is+1)[0],term(is+1)[1]];
    if (iseq[is][1] == "(") {
        var v = expr(is+1);
        if (iseq[v[1]][1] == ")") return [v[0],v[1]+1];
        alert("parse error(rpar): " + JSON.stringify(iseq[v[1]]));      }
    alert("parse error(term): " + JSON.stringify(iseq[is])); }
function lex() {
    var num = "";
    for (var i=0; i<itext.length; i++) {
        if ((num == "") && (itext[i] == " ")) continue;
        if ((itext[i]>="0") && (itext[i]<="9")) { num = num + itext[i]; continue; }
        else if (num != "") { iseq[iseq.length] = ["NUM", num]; num = ""; i--; continue; }
        if (itext[i] == ";")   { iseq[iseq.length] = ["EOL", ";"]; continue; }
        if ((itext[i] == "*") || (itext[i] == "/")) { iseq[iseq.length] = ["MDOP", itext[i]]; continue; }
        if ((itext[i] == "+") || (itext[i] == "-")) { iseq[iseq.length] = ["ASOP", itext[i]]; continue; }
        if ((itext[i] == "(") || (itext[i] == ")")) { iseq[iseq.length] = ["PARN", itext[i]]; continue; }
        alert("lex error: " + itext[i]); return false; }
    if (num != "") { iseq[iseq.length] = ["NUM", num]; }
    iseq[iseq.length] = ["EOL", "EOT"]; return true; }
</script></body></html>
```

10. ハノイの塔 (Tower of Hanoi)

パズルの一種。 バラモンの塔、ルーカスタワー とも呼ばれる。伝説によると、今から5000年前，インドのベナレスという町に大寺院があり、そこに世界の中心といわれるドームがあった。その中に台が作られていて、その上にダイヤモンドでできた棒が 3 本立っていた。インドの神ブラーマは、世界が始まるとき、この棒に黄金でできた円盤を 64 枚 差しておいた。この円盤は下が大きく、上に行くほど小さくできていて、ピラミッド状に積み上げられていた。ブラーマは僧侶たちに次のような修行を与えた。

1. 積み上げられた円盤を，すべて他の棒に移すこと。
2. その際、1 回に 1 枚しか動かしてはならない。また、小さな円盤の上にそれより大きな円盤を乗せてはならない。
3. すべてこの 3 本の棒を使って移すこと。棒以外のところに円盤を置いてはならない。

ブラーマは、この円盤がすべて他の棒に移った瞬間、世界は消滅してしまうと予言している。5000 年たった今も、寺院ではこの修行が続けられているという。n 枚の円盤すべてを移動させるには最低 $2^n - 1$ 回の手数がかかる。プログラミングにおける再帰的呼出しの例題としてもよく用いられる。

解き方

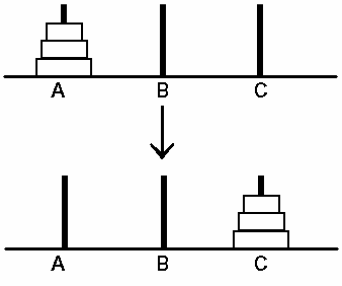
3本の棒を、A、B、C と名前付ける。
最初 A に n 個の円盤があり、Cにすべての円盤を移動させるとすると、次のようにする。

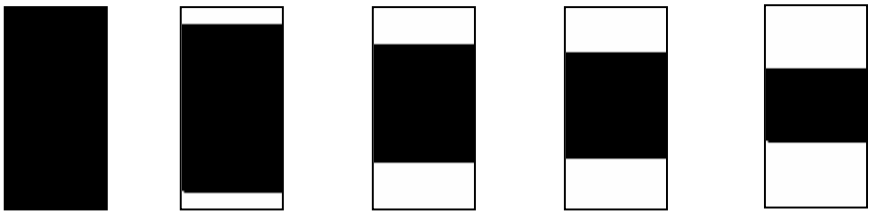
1. 上から n - 1 個目までの円盤を何らかの方法で A から B に移動。
2. 残った 1 枚を A から C に移動。
3. B にある円盤を何らかの方法で B から C に移動。

1 は最初Aに n - 1 個の円盤があり、Bにすべての円盤を移動させるという問題ととらえることができる。
そこで、次のようにする。

1. 上から n - 2 個目までの円盤を何らかの方法で A から C に移動。
2. 残った 1 枚を A から B に移動。
3. C にある円盤を何らかの方法で C から B に移動。

3 も同様にして行うことができ、「何らかの方法」の部分を分解していくと解ける。





1.png 15.png 2.png 25.png 3.png

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset=utf-8>
    <style> div { width:301px; height:102px; margin:1px;
                  background-color:#ffffff; border:1px solid #0000ff; }
    </style>
  </head>
  <body>
    <h1>ハノイの塔</h1>
    <p>移動元</p>
    <div id="p1"><img src="1.png"><img src="15.png"><img
                  src="2.png"><img src="25.png"><img src="3.png"></div>
    <p>中継点</p>
    <div id="p2"></div>
    <p>移動先</p>
    <div id="p3"></div>
    <input type=button onclick=go() value="step">
    <input type=button onclick=go2() value="run">
    <script>
      var step=0, froms = Array(100), tos = Array(100);
      var s = 0, timerId = null;
      var p1 = document.getElementById("p1");
      var p2 = document.getElementById("p2");
      var p3 = document.getElementById("p3");
      hanoi(p1,p2,p3,5);

      function hanoi(from,via,to,n) {
        if (n<=1) { tos[step] = to; froms[step++] = from; }
        else      { hanoi(from,to,via,n-1);
                    hanoi(from,via,to,1);
                    hanoi(via,from,to,n-1);
      }  }

      function go() {
        if (s >= step) {
          if (timerId!=null) clearInterval(timerId);
          timerId = null; return; }
        tos[s].appendChild(froms[s].lastChild);
        s++;
      }

      function go2() { setInterval(go,2000); }
    </script>
  </body>
</html>
```

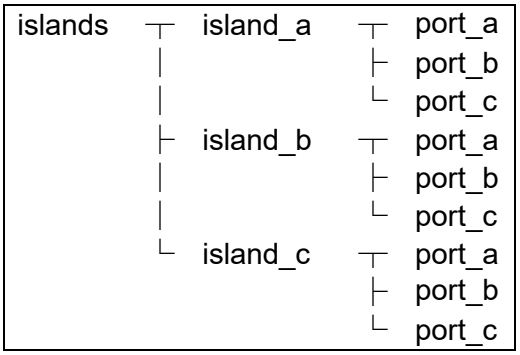
11. 宝さがしゲーム

islands 諸島へようこそ。ここには3つの島 (island-a、b、c という名の3つのディレクトリ) があり、各々の島には3つの港 (port-a、b、c という名のディレクトリ) があります。どこかの港に宝 (treasure という名のファイル) があります。これをみつけて、% cat treasure で宝物を手に入れて下さい。鍵 (key という名のファイル) をみつけたら、% cat key でヒントを得ることができます。

自分が今どこにいるかを知る: % pwd
何が見えるかを知る: % ls
見える port_a に移動する % cd port_a
1つ手前の場所へ戻る: % cd ..
メッセージを読む: % cat key

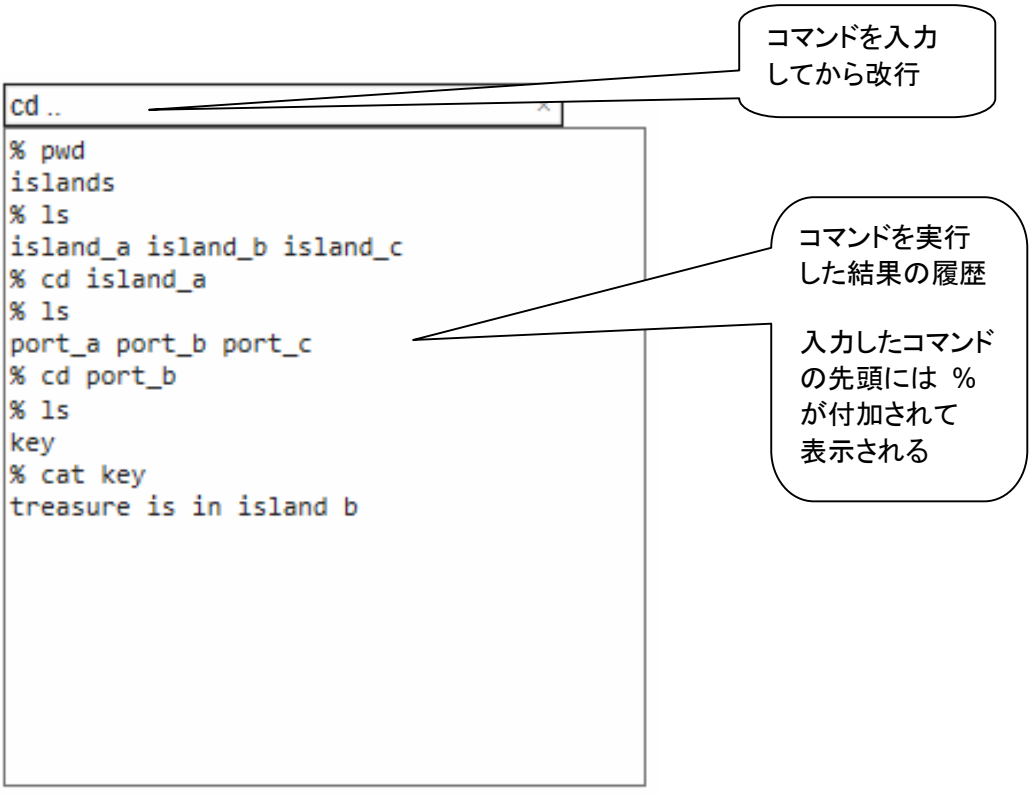
無料イラスト素材

ディレクトリ「islands」以下の構造:



treasure、key はどこかの port_? にあります。

いらすとやフリー素材



```
<!DOCTYPE html>
<html>
  <head><meta charset="utf-8"></head>
  <body>
     <input type=text size=40 id=pg autofocus value="pwd"><br>
     <textarea rows="20" cols="40" id=log></textarea>
     <script>
var geval = eval;
var logp = document.getElementById("log");
var pgp = document.getElementById("pg");
var islands = { island_a: { port_a: {}, port_b: { key: "treasure is in island b" }, port_c: {} },
                island_b: { port_a: {}, port_b: { treasure: "congratulations!" },     port_c: {} },
                island_c: { port_a: {}, port_b: { key: "treasure is in island b" }, port_c: {} } };
var cwd = ["islands"];
function pwd() {
    var o = cwd[0];
    for (var i=1; i<cwd.length; i++) o = o + "/" + cwd[i];
    return o; }
function cd(dir) {
    if (dir == "..") { if (cwd.length > 1) { cwd.length--; return ""; }
                       else                { return "parameter error"; }}
    var o = cwd[0];
    for (var i=1; i<cwd.length; i++) o = o + "." + cwd[i];
    var d = geval(o); var ok = false;
    for (var j in d) { if (j == dir) { cwd[cwd.length] = dir; ok = true; } }
    if (ok == false) return "error: " + dir + "not found";
    else return ""; }
function ls() {
    var o = cwd[0];
    for (var i=1; i<cwd.length; i++) o = o + "." + cwd[i];
    var d = geval(o); o = "";
    for (var j in d) o = o + j + " ";
    return o;
}
function cat(file) {
    var o = cwd[0];
    for (var i=1; i<cwd.length; i++) o = o + "." + cwd[i];
    var d = geval(o); var ok = false; o = "";
    for (var j in d) { if (j == file) { o = d[file]; ok = true; } }
    if (ok == false) return "error: " + file + "not found";
    else return o; }
function go() {
    var cmd = [], cmdp = 0, word = "", inl = pgp.value, opt="";
    for (var i=0; i<inl.length; i++) {
        if (inl[i] == " ") {
            if (word != "") { cmd[cmdp++] = word; word = ""; }
            continue; }
        word = word + inl[i]; }
    if (word != "") cmd[cmdp] = word;
    switch(cmd[0]) {
        case "pwd": if (cmd.length == 1) opt = pwd();            else opt = "parameter error"; break;
        case "ls":    if (cmd.length == 1) opt = ls();           else opt = "parameter error"; break;
        case "cd":    if (cmd.length == 2) opt = cd(cmd[1]);   else opt = "parameter error"; break;
        case "cat": if (cmd.length == 2) opt = cat(cmd[1]); else opt = "parameter error"; break;
    }
    if (opt != "") opt = opt + "¥n";
    logp.value += "% " + inl + "¥n" + opt;
    pgp.value = "";
    logp.scrollTop = logp.scrollHeight;
}
document.onkeydown = function (e) { if (e.keyCode == 13) go(); }
    </script>
  </body>
</html>
```

12. 魔方陣（Magic square）

n×n 個の正方形の碁盤目（方陣）に数字を配置し、縦、横、対角線いずれの列についても、列の数字の合計が同じになるもの。n×n の場合、n 次の魔方陣という。特に1から方陣のマスの総数 n^2 までの数字を1つずつ過不足なく使ったものを言う。このときの一列の和（定和）は、1～n^2 の総和を n で割ったものであり、$n(n^2+1)/2$ で計算できる。回転や裏返しによる派生形を除くと、3次は1種、4次は880種、5次は275,305,224種あることが計算機を使って確認されている。、6次は約1800京種と推定されている。

奇数×奇数の魔方陣の作り方の例。

(1) ヒンズーの連続方式

1.上段の中央を1にする
2.右上に次の数字を置いていく
3.右上が埋まっていたら一つ下に次の数字を置く
4.再び右上へと数字を埋めていく
5.後は3,4の繰り返しで完成

上下左右で、魔方陣の外に出てしまったら、モジュラの形で折り返す。

–	–	–	1	–	–	–
–	–	–	–	–	–	–
–	–	–	–	–	–	–
–	–	–	–	–	–	–
–	–	–	–	–	–	–
–	–	–	–	–	–	–
–	–	–	–	–	–	–

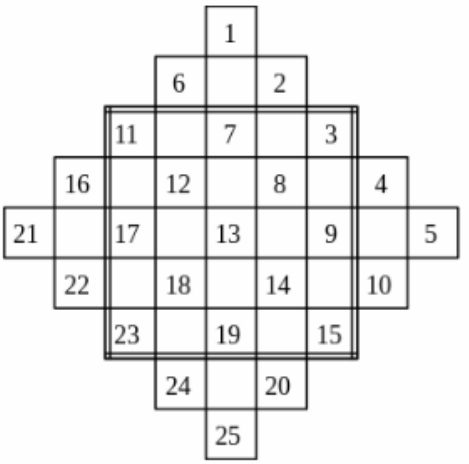
–	–	–	1	–	–	–
–	–	7	–	–	–	–
–	6	–	–	–	–	–
5	–	–	–	–	–	–
–	–	–	–	–	–	4
–	–	–	–	–	3	–
–	–	–	–	2	–	–

–	–	–	1	–	–	–
–	–	7	–	–	–	–
–	6	8	–	–	–	–
5	–	–	–	–	–	–
–	–	–	–	–	–	4
–	–	–	–	–	3	–
–	–	–	–	2	–	–

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

30	39	48	1	10	19	28
38	47	7	9	18	27	29
46	6	8	17	26	35	37
5	14	16	25	34	36	45
13	15	24	33	42	44	4
21	23	32	41	43	3	12
22	31	40	49	2	11	20

(2) バシェー方式



11	24	7	20	3
4	12	25	8	16
17	5	13	21	9
10	18	1	14	22
23	6	19	2	15

ヒンズーの連続方式


```
<style>
   body,input { font-size: 24px; }
   table      { border-collapse: collapse; font-size: 24px; }
   td         { text-align: center; border: solid 1px; padding: 10px; }
</style>
サイズ（奇数）:<input type=text size=3 value=3 id=dim>
<input type=button value=go onclick=go()><p>
<div id=out></div>
<script>
function go() {
    var N = Number(document.getElementById("dim").value);
    var m = new Array(N); for (var i=0; i<N; i++) m[i] = new Array(N);
    var x = Math.floor(N/2);
    var y = -1;
    var n = 0;
    for (var c=0; c<N; c++) {
        y++; x--; y++;
        for (var d=0; d<N; d++) {
            x++; y--; n++;
            if (x<0) x+=N; else if (x>N-1) x-=N;
            if (y<0) y+=N; else if (y>N-1) y-=N;
            m[x][y] = n;
    }   }
    var o = "<table>";
    for (x=0; x<N; x++) {
        o += "<tr>";
        for (y=0; y<N; y++) { o += "<td>" + m[x][y] + "</td>"; }
        o += "</tr>";
    }
    o += "</table>";
    var s = N * (N*N + 1) / 2;
    o += "<br>S=" + s ;
    document.getElementById("out").innerHTML = o;
}
</script>
```

バシェー方式


```
<style>
   body,input { font-size: 24px; }
   table      { border-collapse: collapse; font-size: 24px; }
   td         { text-align: center; border: solid 1px; padding: 10px; }
</style>
サイズ（奇数）:<input type=text size=3 value=3 id=dim>
<input type=button value=go onclick=go()><p>
<div id=out></div>
<script>
function go() {
    var N = Number(document.getElementById("dim").value);
    var m = new Array(N); for (var i=0; i<N; i++) m[i] = new Array(N);
    for (var x=N-1,y=0; x>N/2; x--,y--) ;
    for (var c=0; c<N; c++,x--,y++) {
        for(var d=0,xx=x,yy=y; d<N; d++,xx++,yy++) {
            var xxx = xx; if (xxx<0) xxx+=N; if (xxx>N-1) xxx-=N;
            var yyy = yy; if (yyy<0) yyy+=N; if (yyy>N-1) yyy-=N;
            var n = (d + N*c) + 1;
            m[xxx][yyy] = n;
    }   }
    var o = "<table>";
    for (x=0; x<N; x++) {
        o += "<tr>";
        for (y=0; y<N; y++) { o += "<td>" + m[x][y] + "</td>"; }
        o += "</tr>";
    }
    o += "</table>";
    var s = N * (N*N + 1) / 2;
    o += "<br>S=" + s ;
    document.getElementById("out").innerHTML = o;
}
</script>
```

4次の魔方陣の作り方

(1) 1 ～ 16 までを順に並べる。
(2) 4 隅と中央の 4マスは動かさず、他の数は、中心に対して対称の位置にあるマスに移動させる。

```
［ 1，  2，  3，  4］    ［ 1，15，14，  4］
［ 5，  6，  7，  8］ →［12，  6，  7，  9］
［ 9，10，11，12］    ［ 8，10，11，  5］
［13，14，15，16］    ［13，  3，  2，16］
```

1	15	14	4
12	6	7	9
8	10	11	5
13	3	2	16

S=34

```
<style>
   body,input { font-size: 24px; }
   table      { border-collapse: collapse; font-size: 24px; }
   td         { text-align: center; border: solid 1px; padding: 10px; }
</style>
<div id=out></div>
<script>

go();

function go() {
    var N = 4, M = N-1, xx, yy;
    var m = new Array(N); for (var i=0; i<N; i++) m[i] = new Array(N);

    for (var x=0; x<N; x++) {
        for (var y=0; y<N; y++) {
            if ((x%M==0)&&(y%M!=0) || (x%M!=0)&&(y%M==0)) { xx = M-x; yy = M-y; }
            else                                          { xx = x;   yy = y;   }
            var n = xx*N + yy + 1;
            m[x][y] = n;
        }
    }
    var o = "<table>";
    for (x=0; x<N; x++) {
        o += "<tr>";
        for (y=0; y<N; y++) { o += "<td>" + m[x][y] + "</td>"; }
        o += "</tr>";
    }
    o += "</table>";
    var s = N * (N*N + 1) / 2;
    o += "<br>S=" + s ;
    document.getElementById("out").innerHTML = o;
}
</script>
```

13. 円周率

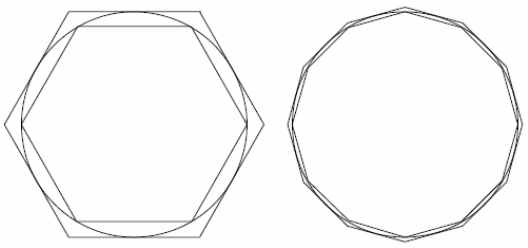
「最新のコンピュータで円周率を〇〇桁計算した」などのニュースを見たことはないだろうか？ どんな複雑な計算をしているかというと、実は単純。

アルキメデスの漸化式（Archimedes' Recurrence Formula）、またはその改良したものを使っている。アルキメデス（BC 287-212）は円周率の近似値をきわめて厳密に計算したことでも知られている。円に対する外接多角形と内接多角形の周長を、辺数を倍々にしながら、次々に計算した。その際に使ったのが、アルキメデスの漸化式。

円に外接、内接するそれぞれの正 3 × 2n 角形の辺の長さを a[n], b[n] としたとき

a[n+1] = 2 * a[n]*b[n]　　a[n+1] は、外接正多角形の周長
b[n+1] = Math.sqrt(a[n+1] * b[n])　　b[n+1] は、内接正多角形の周長

1回の計算ごとに、n角形の辺の数が倍になる。はじめの多角形として正六角形をアルキメデスは選んだ。円の直径を 1、半径を 0.5 とすれば、a[0] = 2 √3 と b[0] = 3になる。アルキメデスは 96 角形まで計算したとされる。n = 1 から n = 5 まで計算することにより 223/71 < π < 22/7 を求めた。
小数だと 3.14084 < π < 3.14286 である。



プログラムはこんなに単純です。

```
<textarea cols=65 rows=28 id=out></textarea>
<script>
var outp = document.getElementById("out");
var a = 2 * Math.sqrt(3);
var b = 3;
var k = 6;
var ap = 0;

while (ap != a) {
    outp.value += "k:" + k + "¥ta:" + a + "¥tb:" + b + "¥n";
    ap = a;
    a = 2 * (a * b) / (a + b);
    b = Math.sqrt(a * b);
    k = k * 2;
}
</script>
```

```
k:6     a:3.4641016151377544    b:3
k:12    a:3.215390309173473     b:3.1058285412302493
k:24    a:3.1596599420975004    b:3.132628613281238
k:48    a:3.146086215131435     b:3.139350203046867
k:96    a:3.1427145996453682    b:3.1410319508905093
k:192   a:3.141873049979824     b:3.141452472285462
k:384   a:3.1416627470568485    b:3.1415576079118574
k:768   a:3.1416101766046895    b:3.141583892148318
k:1536  a:3.141597034321526     b:3.14159046322805
k:3072  a:3.1415937487713514    b:3.1415921059992713
k:6144  a:3.1415929273850964    b:3.141592516692157
k:12288 a:3.1415927220386135    b:3.1415926193653836
k:24576 a:3.1415926707019976    b:3.1415926450336906
k:49152 a:3.1415926578678443    b:3.1415926514507672
k:98304 a:3.1415926546593056    b:3.1415926530550364
k:196608        a:3.141592653857171     b:3.1415926534561036
k:393216        a:3.1415926536566365    b:3.14159265355637
k:786432        a:3.1415926536065033    b:3.1415926535814367
k:1572864       a:3.1415926535939697    b:3.1415926535877032
k:3145728       a:3.1415926535908367    b:3.14159265358927
k:6291456       a:3.141592653590053     b:3.141592653589661
k:12582912      a:3.141592653589857     b:3.1415926535897593
k:25165824      a:3.141592653589808     b:3.141592653589784
k:50331648      a:3.1415926535897962    b:3.14159265358979
k:100663296     a:3.141592653589793     b:3.1415926535897913
k:201326592     a:3.1415926535897922    b:3.141592653589792
k:402653184     a:3.141592653589792     b:3.141592653589792
```

14. フィボナッチ数列と螺旋葉序（らせんようじょ）

＜フィボナッチ数列＞

- 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, ・・・・・
- 最初の 2 つの 1 を除いた数列のそれぞれの数は，1 つ前の数と2 つ前の数との和になっている。
- 2 ＝ 1 ＋ 1, 3 ＝ 1 ＋ 2, 5 ＝ 2 ＋ 3, 8 ＝ 3 ＋ 5, ・・・・・・
- フィボナッチ数列の隣り合う 2 項の比は黄金比（(1 + √5) / 2 = 1.618）に収束する。
- 黄金比のおおまかな値は 5:8。黄金比はギリシア時代から最も調和のとれた比といわれ，絵画，建築等に応用され，本やはがきの縦横比もこれに近い。名刺も。

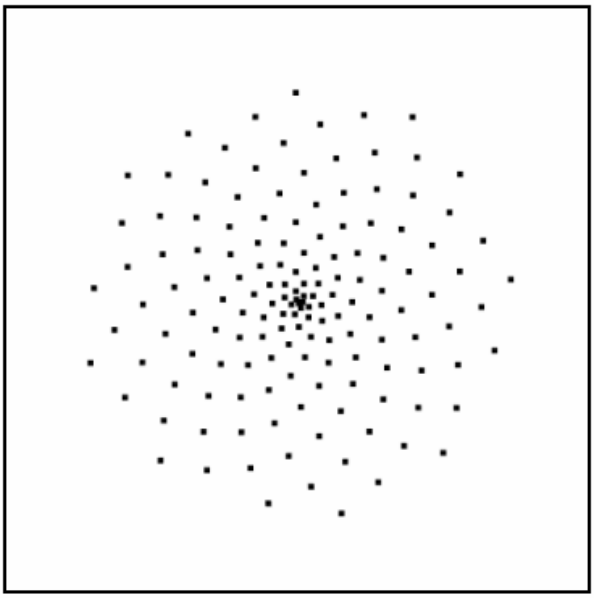
＜植物の葉のつき方の規則(螺旋葉序)＞

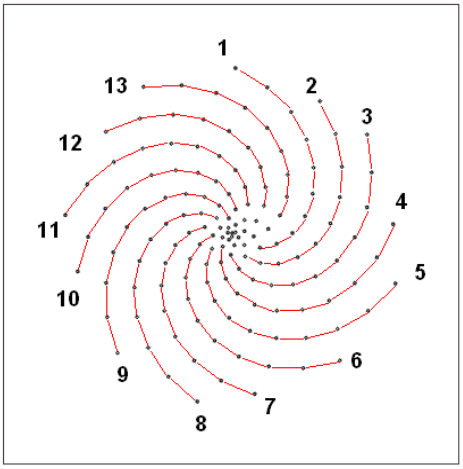
- 茎から葉が出る方向は，1 枚ずつ 2/5 回転，3/8 回転，5/13 回転，8/21 回転などずつ変わってゆく。
- 回転角が順次変わるという意味ではない。植物によって、回転角は一定。
- これらの回転数の分母，分子ともフィボナッツィ数列の数。
- n／m × 360° ずつ回転しながら生えてゆくと、n 点、m 点ごとに 360° に近づくので、つながって見える。
  360° ×黄金分割比 ずつ回転しながらだと、連続するフィボナッチ数の倍数に近いが、わずかに異なるので、つながって見える線が少しずつずれて、うず状に見える。

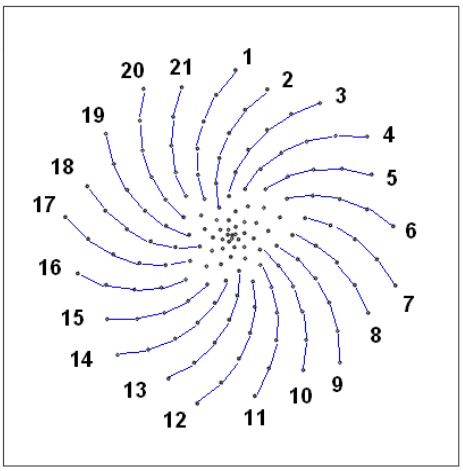
```
<!DOCTYPE html>
<html><body><script>
var canvas = document.createElement('canvas');
document.body.appendChild(canvas);
canvas.width    = canvas.height = 400;
canvas.style.border = "solid 2px";
var ctx = canvas.getContext('2d');
var x = y = a = 0;

function mv(step) {
   x += step * Math.sin(a * Math.PI / 180.0);
   y += step * Math.cos(a * Math.PI / 180.0);
}

var r=0, c=150;
for(;;) {
     mv(r);
     ctx.fillRect(200+x, 200-y, 4, 4);;
     mv(-r);
     r += 1;
     a += 360*1.6180339887;
     if(c-- == 0) break;
}
</script></body></html>
```







15. 小町算（こまちざん）

数の遊び、数学パズルの一種。1□2□3□4□5□6□7□8□9 = 100 という数式の□の中に、＋,－,×,÷,空白 のいずれかを一つずつ入れて正しい数式を完成させるというもの。解析的な解法はない。答はトライ＆エラー、または総当たりで探すしかないとされる。

□＝空白の場合、数字を結合して、多桁とする。□として、＋,－,空白 のいずれかだけを許すルール、1の前に「-」も置ける、というルール、括弧も使えるというルール、9□8□7□6□5□4□3□2□1 のように並べるというルールもある。江戸時代には「一ら十までの数字を使って九十九を表す」というルールだったとする説がある。

□が、＋,－,×,÷,空白 のいずれかで、1の前に「-」も置けるというルールの場合、162種の解がある。□が、＋,－,空白 のいずれかで、1の前に「-」も置けるというルールの場合、12種の解がある。

江戸時代の寛保年間（1743年頃）には既に知られていた。小町の名称は、平安時代の歌人、小野小町に由来する。小野小町に「100夜続けて通えば愛を受け入れる」と約束された深草少将が、99夜通い、あと1夜というところで亡くなってしまったという「百夜通い（ももよがよい）」の伝説があり、後に、小野小町が思い出すたびに1から9までの数字を使って100を作る計算を考えたという話が作られたとする説がある。その他、小野小町のように美しい数式だからという説、このパズルにはまると時間のたつのを忘れて年寄りになってしまうので、「花の色はうつりにけりないたづらに、我が身世にふるながめせしまに」という小野小町の歌にかけたとする説、などがある。

「世界大百科事典」（平凡社）によれば、田中由真(1651-1719)著の「雑集求笑算法」(1698)が、小町算や目付字などを含め、22の題材を扱っているとされる。この本を種本とする古典パズルの教典「勘者御伽双紙」(1743)もある。

ヨーロッパでは，このような計算を「センチュリーパズル」とよぶ。センチュリーは、一世紀＝100年 のこと。

「＋、－」だけの場合のプログラム

```
<textarea cols=50 rows=14 id=out></textarea>
<script>
var outp = document.getElementById("out");
var op = [ "", "+", "-" ];
var d = [0,0,0,0,0,0,0,0], dg, a=1;
while (true) {
    var str = "1";
    for (var n=0; n<8; n++) { str = str + op[d[n]] + (n+2); }
    if (eval(str) == 100) outp.value += a++ + ": " + str + "¥n";
    if (eval("-"+str) == 100) outp.value += a++ + ": " + "-"+str + "¥n";
    dg = 0;
    while(true) {
        if          (d[dg] == 0) { d[dg] = 1; break; }
        else if (d[dg] == 1) { d[dg] = 2; break; }
        else if (dg < 7)        { d[dg++] = 0; continue; }
        else                            { dg = 8; break; }
    }
    if (dg > 7) break;
}
</script>
```

```
1: 123-45-67+89
2: 12-3-4+5-6+7+89
3: 12+3+4+5-6-7+89
4: 123+4-5+67-89
5: -1+2-3+4+5+6+78+9
6: 1+2+3-4+5+6+78+9
7: 12+3-4+5+67+8+9
8: 1+23-4+56+7+8+9
9: 1+2+34-5+67-8+9
10: 1+23-4+5+6+78-9
11: 123+45-67+8-9
12: 123-4-5-6-7+8-9
```

「＋、－、、×、÷」の場合。実行に数分かかるので、ブラウザで「このWebページは応答していません」などの警告メッセージが出る場合があるが、しばらくがまんしていると、結果が表示される。

```
<textarea cols=50 rows=24 id=out></textarea>
<script>
var outp = document.getElementById("out");
var op = [ "", "+", "-", "*", "/" ];
var d = [0,0,0,0,0,0,0,0], dg, a=1;
while (true) {
    var str = "1";
    for (var n=0; n<8; n++) { str = str + op[d[n]] + (n+2); }
    if (eval(str) == 100) outp.value += a++ + ": " + str + "¥n";
    if (eval("-"+str) == 100) outp.value += a++ + ": " + "-"+str + "¥n";
    dg = 0;
    while(true) {
        if          (d[dg] == 0) { d[dg] = 1; break; }
        else if (d[dg] == 1) { d[dg] = 2; break; }
        else if (d[dg] == 2) { d[dg] = 3; break; }
        else if (d[dg] == 3) { d[dg] = 4; break; }
        else if (dg < 7)        { d[dg++] = 0; continue; }
        else                            { dg = 8; break; }
    }
    if (dg > 7) break;
}
</script>
```

1: -12*3-4*5+67+89
2: 123-45-67+89
3: 1*2-3+4-5+6+7+89
4: 1+2*3-4-5+6+7+89
5: 1-23+4*5+6+7+89
6: -1*2+3+4+5-6+7+89
7: 12-3-4+5-6+7+89
8: 1+2+3*4-5-6+7+89
9: 1-23-4+5*6+7+89
10: -1+2/3/4*5*6+7+89
11: -1/2-3+45/6+7+89
12: -1+2/3*45/6+7+89
13: 1*2/3+4*5/6+7+89
14: 1/2*34-5+6-7+89
15: 12+3+4+5-6-7+89
16: 1*23-4+5-6-7+89
17: -1+2+3+4*5-6-7+89
18: 12/3+4*5-6-7+89
19: -1-2/3*45+6*7+89
20: -1*2-34+5+6*7+89
21: 1-23-4-5+6*7+89
22: 1*2-3+4+56/7+89
23: 1+2*3-4+56/7+89
24: 12+3+4-56/7+89
25: 1*23-4-56/7+89
26: -1+2*3/4*56/7+89
27: 123+4-5+67-89
28: 1+234*5/6-7-89
29: 12+3*45+6*7-89
30: 1+2*34-56+78+9
31: -1+2-3+4+5+6+78+9
32: 1+2+3-4+5+6+78+9
33: 1*2*3-4+5+6+78+9
34: 1*2+3*4+5-6+78+9
35: 12+3*4-5-6+78+9
36: 1*2*3*4-5-6+78+9
37: 1*2-3+4*5-6+78+9
38: -1/2*34+5*6+78+9
39: 1+2+3*4*5/6+78+9
40: -1+23+4+5/6*78+9
41: 1+234*5*6/78+9
42: 1+2*3+4+5+67+8+9
43: 12+3-4+5+67+8+9
44: 1-2+3*4+5+67+8+9
45: 1-2-3+4*5+67+8+9
46: -12/3+4*5+67+8+9
47: 12*3-4*5+67+8+9
48: 1/2/3*456+7+8+9
49: 1+23-4+56+7+8+9
50: 12+34+5*6+7+8+9
51: -1+23*4+5-6-7+8+9
52: 1-2-3+45+6*7+8+9
53: -12/3+45+6*7+8+9
54: 1*2+34+5+6*7+8+9
55: 12+34-5+6*7+8+9
56: -123-4+5*6*7+8+9
57: -1+23*4-56/7+8+9
58: 1*23+4+5+67-8+9
59: 1+2+34-5+67-8+9
60: 1*2+34+56+7-8+9
61: -1+23*4-5+6+7-8+9
62: 1+23*4+5-6+7-8+9
63: -1-2+3*4*5+6*7-8+9
64: -1+23*4+56/7-8+9
65: 1+2+3*4*56/7-8+9
66: -1+23*4-56+7*8+9
67: -1+2/3*45+6+7*8+9
68: 12+3*4+5+6+7*8+9
69: 1*2*3*4+5+6+7*8+9
70: 12-3+4*5+6+7*8+9
71: 1-2-3+45-6+7*8+9
72: -12/3+45-6+7*8+9
73: 1*2+34+5-6+7*8+9
74: 12+34-5-6+7*8+9
75: -1*2+3+4+5*6+7*8+9
76: 12-3-4+5*6+7*8+9
77: -1+23*4+56-7*8+9
78: -123+45*6-7*8+9
79: -1+23*4/56*7*8+9
80: 1*23+4+56/7*8+9
81: 1*23*4-56/7/8+9
82: -1+23*4*56/7/8+9
83: -1-2*3*4+56+78-9
84: 1+23-4+5+6+78-9
85: 1*2+3+4*5+6+78-9
86: -1+2*3+4*5+6+78-9
87: -1*2+34+5-6+78-9
88: 12*3-4+5-6+78-9
89: 1*2+3-4+5*6+78-9
90: -1+2*3-4+5*6+78-9
91: 12/3/4+5*6+78-9
92: 123+45-67+8-9
93: -1*2+34*5-67+8-9
94: -1*234+5*67+8-9
95: 1+23*4-5+6+7+8-9
96: 123-4-5-6-7+8-9
97: 1-2+3*4*5+6*7+8-9
98: 123+4*5-6*7+8-9
99: 1+23*4+56/7+8-9
100: 1*2+3+45+67-8-9
101: -1+2*3+45+67-8-9
102: 1*2*34+56-7-8-9
103: -1+2+3+4*5*6-7-8-9
104: 12/3+4*5*6-7-8-9
105: -1-23*4+5*6*7-8-9
106: -1-2*3+4+56+7*8-9
107: -1*2+3-4+56+7*8-9
108: 1-2+3+45+6+7*8-9
109: 1-2+3*4*5-6+7*8-9
110: -1+2*3*4+5*6+7*8-9
111: -1+2+34*5-6-7*8-9
112: -1+2+3+4*5*6*7/8-9
113: 12/3+4*5*6*7/8-9
114: 1+2+3-45+67+8*9
115: 1*2*3-45+67+8*9
116: 1-2-34+56+7+8*9
117: 1/2*3/4*56+7+8*9
118: 1+2+3+4+5+6+7+8*9
119: 1*2*3+4+5+6+7+8*9
120: -1*2+3*4+5+6+7+8*9
121: 1+23-4-5+6+7+8*9
122: -1*2-3+4*5+6+7+8*9
123: 1-2*3+4*5+6+7+8*9
124: -12+34+5-6+7+8*9
125: -1*2+34-5-6+7+8*9
126: 12*3-4-5-6+7+8*9
127: 1+2*3+4*5-6+7+8*9
128: -1*2-3-4+5*6+7+8*9
129: 1-2*3-4+5*6+7+8*9
130: 1+2-3*4+5*6+7+8*9
131: -12*3/4+5*6+7+8*9
132: 1+2*3*4*5/6+7+8*9
133: -1+23*4-56-7+8*9
134: -1+2/3*45+6-7+8*9
135: 12+3*4+5+6-7+8*9
136: 1*2*3*4+5+6-7+8*9
137: 12-3+4*5+6-7+8*9
138: 1-2-3+45-6-7+8*9
139: -12/3+45-6-7+8*9
140: 1*2+34+5-6-7+8*9
141: 12+34-5-6-7+8*9
142: -1*2+3+4+5*6-7+8*9
143: 12-3-4+5*6-7+8*9
144: -1*23+4+5+6*7+8*9
145: -12-3-4+5+6*7+8*9
146: -1*2-3-4-5+6*7+8*9
147: 1-2*3-4-5+6*7+8*9
148: 1+2-3*4-5+6*7+8*9
149: -12*3/4-5+6*7+8*9
150: 1+2+3-4*5+6*7+8*9
151: 1*2*3-4*5+6*7+8*9
152: 1+23-4+56/7+8*9
153: 1*2+34-56/7+8*9
154: 1-2-3+4*56/7+8*9
155: -12/3+4*56/7+8*9
156: 1*234+5-67-8*9
157: -1+2+34*5-6+7-8*9
158: 1+234-56-7-8*9
159: 1+2+34*5+6-7-8*9
160: -1+2/3*45*6-7-8*9
161: -1+23*4+56/7/8*9
162: -12+3*4*5+6*78/9

16. ツェラーの公式（Zeller's congruence）

西暦年月日から曜日を計算する公式。クリスティアン・ツェラー (Christian Zeller) が考案した。ユリウス通日（西暦1年 1月 1日 からの日数）を求め、そこから曜日を求める計算と本質は同じ。

h百y年m月d日の曜日を求めるとする。1月、2月は、前の年の13月、14月として計算する。例えば、2001年11月3日の場合、h = 20、y = 1、m = 11、d = 3 となる。

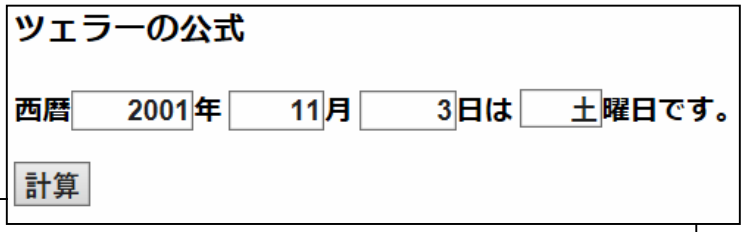
w = y + [y/4] ＋ [h/4] - 2*h + [13 * (m ＋1) / 5] ＋ d の値を求める。

ここで、[X] は、ガウスの記号。X を超えない最大の整数、つまり、X の整数部分を表す。

w を 7 で割った余りを r とする。

曜日は、次の表で求まる。

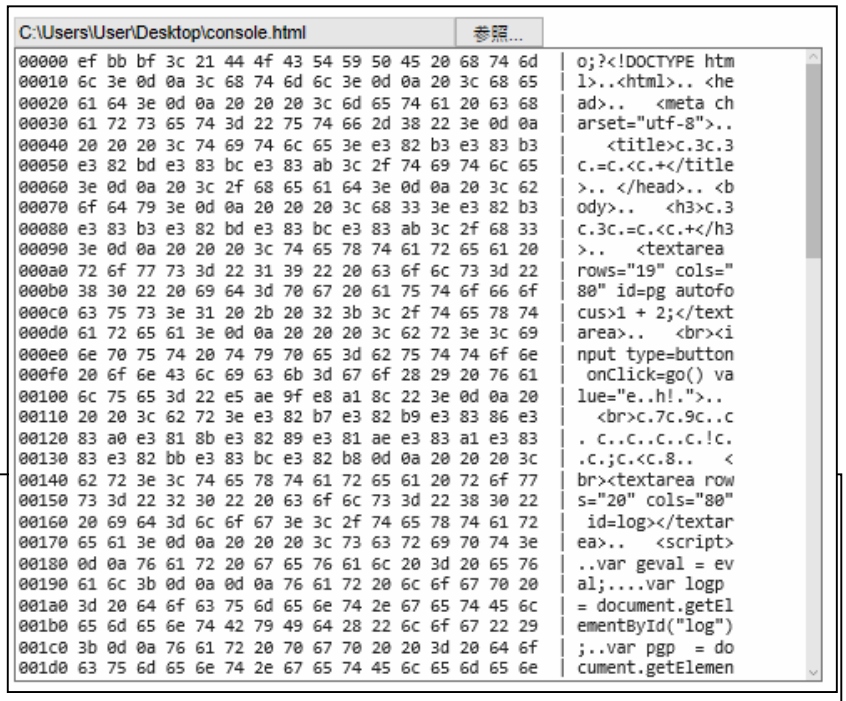
r	0	1	2	3	4	5	6
曜日	土	日	月	火	水	木	金




```
<!DOCTYPE html>
<html> <head>
      <meta charset="utf-8">
      <title>Ant feromon</title>
      <style>
            body,input { font:sans-serif; font-weight:600; font-size:x-large; }
            input    { text-align: right; }
      </style>
   </head> <body>
<h3>ツェラーの公式</h3>
西暦<input type=text id=yearp value=2001 size=4>年
      <input type=text id=monthp value=11 size=2>月
      <input type=text id=dayp value=3 size=2>日は
      <input type=text id=youbip size=1>曜日です。<p>
<input type=button onclick=go() value="計算"><br>
<script>
function go() {
    var year   = Number(yearp.value);
    var month = Number(monthp.value);
    var day     = Number(dayp.value);
    if (month > 2) { var m = month; }
    else                  { m = month + 12; year += 1; }
    var h = Math.floor(year/100);
    var y = year % 100;
    var d = day;
    var w = y + Math.floor(y/4) + Math.floor(h/4) - 2*h + Math.floor(13*(m+1)/5) + d;
    var r = w % 7;
    var table = ["土","日","月","火","水","木","金"];
    youbip.value = table[r];
}
</script>
</body>
</html>
```

17. 16 進ダンプ

ファイルの内容を、16進数で表示するプログラム。




```
<!DOCTYPE html>
<html>
 <head>
    <meta charset="utf-8">
    <title>File menu</title>
 </head>
 <body>
   <input id="fileSel" type="file" multiple="true" onchange="readFile()" size=50>
   <br><textarea rows=30 cols=80 id=out></textarea>
   <script>
function printFile(evt) {
    var cntnt = new Uint8Array(evt.target.result);

var s = "", ss = "", i;
for (i=0; i<cntnt.length; i++) {
   if (i%16==0) s += ("0000" + i.toString(16)).slice(-5) + " ";
   s += ("0"+cntnt[i].toString(16)).slice(-2) + " ";
   var d = cntnt[i]&0x7F;
   ss += (d<0x20)?".":String.fromCharCode(d);
   if (i%16==15) { s += " | " + ss + "¥n"; ss = ""; }
}
if (i%16 != 0) {
    for ( ; i%16 != 0; i++) s += "     ";
    s += " | " + ss + "¥n";
}
out.value = s;
}

function readFile() {
    var reader = new FileReader();
    reader.onload = printFile;
    var files = document.getElementById("fileSel").files;
    reader.readAsArrayBuffer(files[0]);
}

function typeIs(obj) {
      return(Object.prototype.toString.call(obj).slice(8, -1)); }

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