ななちゃんのIT教室

データ構造：複素数　の巻

by nara.yasuhiro@gmail.com

ななちゃんが、複素数 データ構造を
使ってみるという お話

第 0.1 版　2017 年 7 月 3 日

フリー素材
http://freeillustration.net

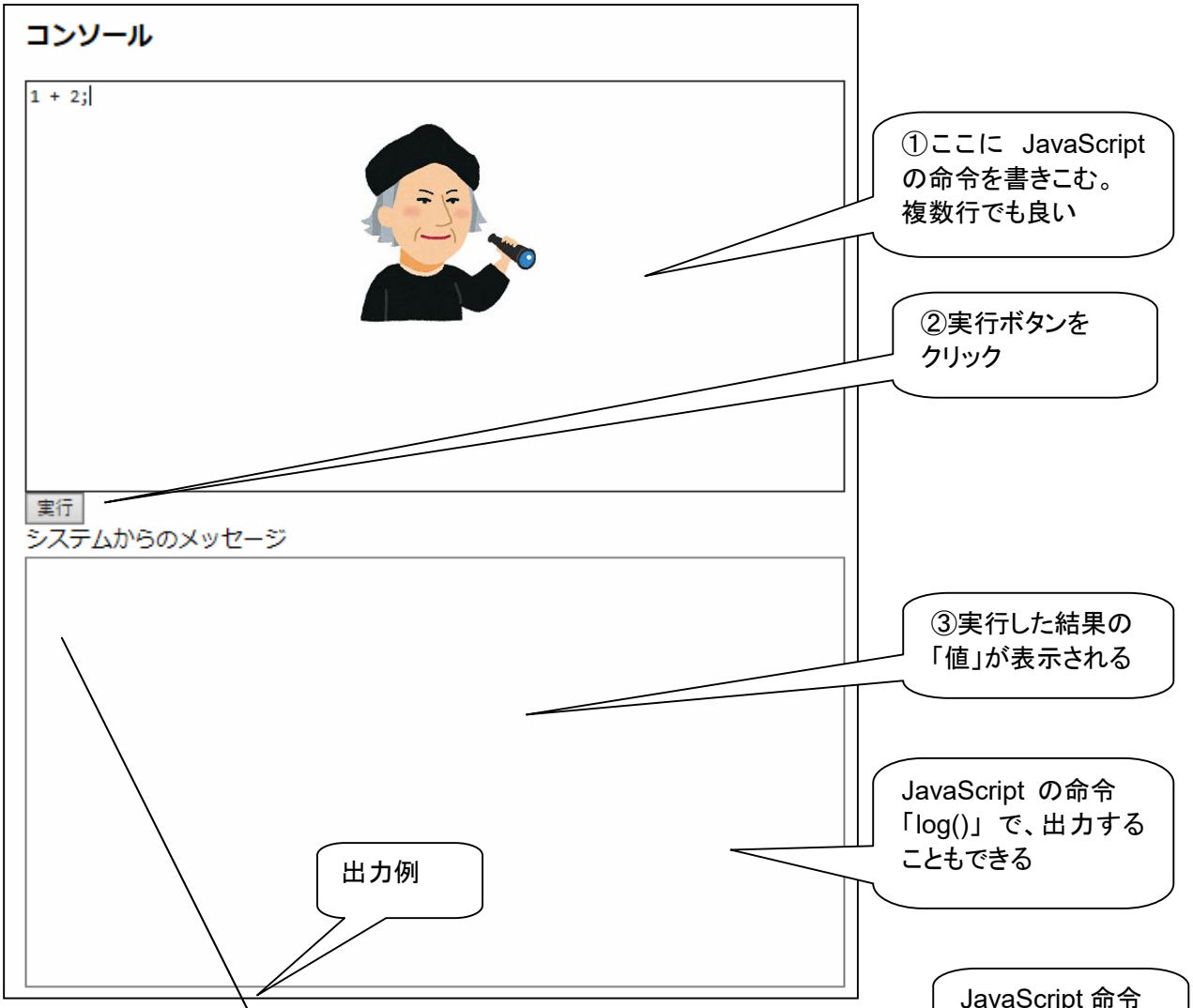
いらすとやフリー素材
http://www.irasutoya.com/

もくじ

第1回　秘密道具:マイ・コンソール

なな:　クリじい、「データ構造」の勉強をするんだけど、便利な秘密道具はない？

クリ:　あるぞ、あるぞ。定番秘密道具の「マイ・コンソール」。他の巻を読んでない読者のために、説明しよう。




```
<= 1 + 2;
=> Number number 3
<= "1" + "2"
=> String string "12"
<= 1;2;
=> Number number 2
<= var x = 1;
=> Undefined undefined undefined
<= x
=> Number number 1
<= var x; x= 1;
=> Number number 1
```

JavaScript 命令「1+2」を入力した

```
<= 1 + 2;
=> Number number 3
```

実行結果の「値」は 3

注意:var x = 1; の値は「undefined」

実行結果の「型」は Number

本 教材ではこのように圧縮表示しています

```
1 + 2;          // Number number 3
"1" + "2"       // String string "12"
1;2;            // Number number 2
var x = 1;      // Undefined undefined undefined
x               // Number number 1
var x; x= 1;    // Number number 1
```

「型」の判定方法は 2 種類 r

「○; ○」のように、複数の JavaScript 命令がある場合、一番右の命令の型、値だけ表示される

JavaScript 命令

実行結果の「型」と「値」

マイ・コンソールのプログラム。本資料のディレクトリには、複素数関連メソッド定義済みのバージョンを添付しています。

```
<!DOCTYPE html>
<html>
 <head>
    <meta charset="utf-8">
    <title>コンソール</title>
 </head>
 <body>
    <h3>コンソール</h3>
    <textarea rows="19" cols="80" id=pg autofocus>1 + 2;</textarea>
    <br><input type=button onClick=go() value="実行">
    <br>システムからのメッセージ
    <br><textarea rows="20" cols="80" id=log></textarea><br>
    <script>
var geval = eval;

var logp = document.getElementById("log");
var pgp    = document.getElementById("pg");
var logd;

function clog(s) { logp.value += s; }
function log(s) { logd += s; }
function typeIs(obj) {
      return(Object.prototype.toString.call(obj).slice(8, -1)); }
function isPrimitive(x) {
    return (typeof x)!="object";
}
function toLiteral(x) {
    if (typeIs(x)=="Number" && isNaN(x)) return "NaN";
    if (x === Infinity)                  return "Infinity";
    if ((typeIs(x)!="Symbol")&&(-x === Infinity)) return "-Infinity";
    if (typeIs(x)=="Set") return "Set("+JSON.stringify([...x])+")";
    if (typeIs(x)=="Map") return "Map("+JSON.stringify([...x])+")";
    return JSON.stringify(x);
}
function type(x) { return "" + (typeof x); }
function isInteger(n) { return n%1 === 0; }
function keys(obj) { return Object.keys(obj); }
function go() {
 logd = "";
 try {
            var v = geval(pgp.value);
            clog("<= " + pgp.value + "\n=> "
               + typeIs(v) + " " + type(v) + " " + toLiteral(v) + "\n");
            pgp.value = "";
            logp.scrollTop = logp.scrollHeight;
            pgp.focus();
 }
 catch(e) { clog("<= " + pgp.value + "\n=>! " + e + "\n");
                pgp.value = "";
                logp.scrollTop = logp.scrollHeight;
                pgp.focus();
 }
 if (logd != "") clog(logd + "\n");
}
   </script>
 </body>
</html>
```

第2回　複素数とは

なな：複素数って何に使うの？

先生：まずは、方程式の解。たとえば、 $x^3 = 1$ の解は、3 次方程式だから、3 つあるはずだけど、実数の世界では 1 しかありません。でも、複素数の世界では、-1/2 - √(3) / 2 i、-1/2 + √(3) / 2 i のふたつも解になり、合計 3 個存在することになります。

```
var x = new Complex(1,0);
var x3 = x.mul(x).mul(x); log(x3)
(1 + 0 i)

var x = new Complex(-1/2,-Math.sqrt(3)/2);
var x3 = x.mul(x).mul(x); log(x3)
(1 + 0 i)

var x = new Complex(-1/2,Math.sqrt(3)/2);
var x3 = x.mul(x).mul(x); log(x3)
(1 + 0 i)
```

1 の 3 乗は 1

-1/2 - √(3) / 2 i の 3 乗も 1

-1/2 + √(3) / 2 i の 3 乗も 1

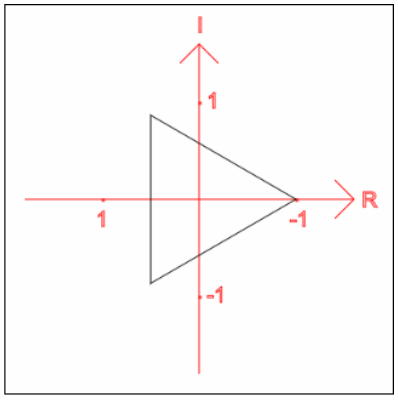
これを複素平面上に表示すると、正三角形の頂点になっていることが分かります。

```
var x1 = new Complex(-1/2,-Math.sqrt(3)/2);
var x2 = new Complex(-1/2,Math.sqrt(3)/2);
var x3 = new Complex(1,0);
var a = [x1, x2, x3];
var c = new Canvas();
c.ComplexAxis();
c.drawComplex(a);
```

配列化

複素平面軸表示

配列を図形化



複素数は、写像を表すのにも使えます。下記は、左に45° 回転し、大きさを 0.7 倍にする例です。

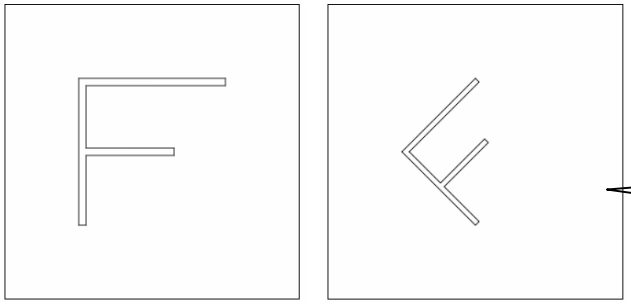
```
var a = [new Complex(-1,1), new Complex(1,1), new Complex(1,0.9),
    new Complex(-0.9,0.9), new Complex(-0.9,0.05), new Complex(0.3,0.05),
    new Complex(0.3,-0.05), new Complex(-0.9,-0.05), new Complex(-0.9,-1),
    new Complex(-1,-1)];

var c = new Canvas();
c.drawComplex(a);

var a2 = a.map(function(c) { return c.mul(new Complex(0.5,0.5)); });
var a2 = a.map((c) => c.mul(new Complex(0.5,0.5)));
c.drawComplex(a2);
```

複素平面上での F の形

写像前の F の形を表示

各点に 0.5 + 0.5 i を掛ける

ECMAScript 2015～のアロー関数を使う場合



45° 左回転 長さ 0.7 の写像になる

第3回　複素数のコンストラクタと表示

なな: 複素数をどうやってプログラムにするの？

先生: 複素数は、a + b i という形で、a と b は実数なので、下記のようなコンストラクタになります。

```
function Complex(r,i) {
    this.r = r;
    this.i = i;
}
```

複素数データの
コンストラクタ

なな: とても単純なのね。

先生: 大切なのは、複素数に関連する演算、足し算とか、掛け算などを、メソッドとして組み込むこと。まずは、複素数のインスタンスを作ったときに、それを文字列として表示するメソッド。小数以下の桁数が多いと見にくいので、小数点以下 2 桁で丸めています。データそのものは正確なままで、表示だけ丸めようということです。

```
Complex.prototype.toString = function() {
    return "("      + Math.round(this.r*100)/100 +
              ((this.i<0)?" - ":" + ") +
              Math.abs(Math.round(this.i*100)/100 ) + " i)";
}
```

複素数データを
文字列化する
メソッド

```
var x = new Complex(1,0.5); log(x);
(1 + 0.5 i)
```

実行例

```
Complex.prototype.equal = function(x) {
    return (this+"") == (x+"");
}
```

二つの
複素数データの
値が等しいかを
判定するメソッド

文字列化すると
同じ文字列に
なるかを判定

第4回 基本的な演算子(和/差、積、共役、距離)

なな: 複素数の和/差は？。

$$c_1 = a_1 + b_1 i, \quad c_2 = a_2 + b_2$$
$$(a_1 + b_1 i) + (a_2 + b_2 i) = (a_1 + a_2) + (b_1 + b_2)i$$

```
Complex.prototype.add = function(x) {
    return new Complex(this.r + x.r, this.i + x.i);
}
```

$$(a_1 + b_1 i) - (a_2 + b_2 i) = (a_1 - a_2) + (b_1 - b_2)i$$

```
Complex.prototype.sub = function(x) {
    return new Complex(this.r - x.r, this.i - x.i);
}
```

なな: 複素数の積は？

$$\begin{aligned}&(a_1 + b_1 i)(a_2 + b_2 i)\\ &= a_1 a_2 + b_1 b_2 i^2 + a_1 b_2 i + a_2 b_1 i\\ &= (a_1 a_2 - b_1 b_2) + i(a_1 b_2 + a_2 b_1)\end{aligned}$$

```
Complex.prototype.mul = function(x) {
    var a = this.r;
    var b = this.i;
    var c = x.r;
    var d = x.i;
    return new Complex(a*c-b*d,a*d+b*c);
}
```

なな: 複素数の共役は？

$$\overline{z} = a - ib$$

```
Complex.prototype.cnj = function() {
    var a = this.r;
    var b = this.i;
    return new Complex(a,-b);
}
```

なな: ふたつの複素数の距離は？

$$P_1P_2 = |z_2 - z_1| = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

```
Complex.prototype.distance = function(c) {
    return new Complex(
        Math.sqrt((this.r-c.r)*(this.r-c.r) + (this.i-c.i)*(this.i-c.i)), 0);
}
```

第5回　少し複雑な演算子（除算、平方根、極形式、指数関数、対数関数、べき乗）

なな:　複素数の割り算はどうするの？

先生:　下記のように計算します。証明は関係書籍で確認してね。

$$
\begin{aligned}
&\frac{a_1+b_1i}{a_2+b_2i} \\
&= \frac{(a_1+b_1i)(a_2-b_2i)}{(a_2+b_2i)(a_2-b_2i)} \\
&= \frac{a_1a_2+b_1b_2+(a_2b_1-a_1b_2)i}{a_2^2+b_2^2} \\
&= \frac{a_1a_2+b_1b_2}{a_2^2+b_2^2} + i\frac{a_2b_1-a_1b_2}{a_2^2+b_2^2}
\end{aligned}
$$

```
Complex.prototype.div = function(x) {
    var a = this.r;
    var b = this.i;
    var c = x.r;
    var d = x.i;
    return new Complex((a*c+b*d)/(c*c+d*d),(b*c-a*d)/(c*c+d*d));
}
```

なな:　平方根は？

先生:　下記のようになります。

b≧0の時

$$
\begin{aligned}
&\sqrt{a+bi} \\
&= \pm\left(\sqrt{\frac{a+\sqrt{a^2+b^2}}{2}} + \sqrt{\frac{-a+\sqrt{a^2+b^2}}{2}}i\right)
\end{aligned}
$$

b<0の時

$$
\begin{aligned}
&\sqrt{a+bi} \\
&= \pm\left(\sqrt{\frac{a+\sqrt{a^2+b^2}}{2}} - \sqrt{\frac{-a+\sqrt{a^2+b^2}}{2}}i\right)
\end{aligned}
$$

```
Complex.prototype.sqrt = function() {
    var x = this.r;
    var y = this.i;
    var m = x * x + y * y;
    var r = Math.sqrt(m);
    var x1 = Math.sqrt((r + x) / 2.0);
    if (y > 0) var y1 =    Math.sqrt((r - x) / 2.0);
    else                   y1 = -Math.sqrt((r - x) / 2.0);
    return new Complex(x1,y1);
}
```

先生: 極形式 (polar form) に関するメソッド群です。

$z = r(\cos(\theta) + i\sin(\theta))$

$r = |z| = \sqrt{a^2 + b^2}$

$\theta = \arctan(b/a)$

```
function PComplex(r,th) {
    this.r = r;
    this.th = th;
}
```

極形式データの
コンストラクタ

```
PComplex.prototype.toString = function(x) {
    return "< r:"    + Math.round(this.r*100)/100 +
               ", th:" + Math.round(this.th*100)/100 +
               "(" + Math.round(this.th/Math.PI*100)/100 + "π) >";
}
```

極形式データを
文字列化する
メソッド

```
PComplex.prototype.toComplex = function() {
    var r = this.r * Math.cos(this.th);
    var i = this.r * Math.sin(this.th);
    return new Complex(r, i);
}
```

極形式データを
複素数表現に
変換するメソッド

```
Complex.prototype.toPcomplex = function() {
    var r = Math.sqrt(this.r*this.r + this.i*this.i);
    var th = Math.atan2(this.i,this.r);
    return new PComplex(r, th);
}
```

複素数データを
極形式表現に
変換するメソッド

```
var c = new Complex(1,1); log(c);
(1 + 1 i)
var p = c.toPcomplex(); log(p);
< r:1.41, th:0.79(0.25π) >
log(p.toComplex());
(1 + 1 i)
```

実行例
1 + i を極形式
に変換してから、
元に戻してい
す

先生：指数関数（exponential function、エクスポネンシャル）を計算するメソッドです。

$e^z = e^a (\cos(b) + i \sin(b))$

```
Complex.prototype.exp = function() {
    var r = Math.exp(this.r) * Math.cos(this.i);
    var i = Math.exp(this.r) * Math.sin(this.i);
    return new Complex(r, i);
}
```

```
log((new Complex(0,Math.PI/2)).exp());
(0 + 1 i)
log((new Complex(0,Math.PI)).exp());
(-1 + 0 i)
```

実行例

先生：対数関数（logarithmic function、log）を計算するメソッドです。

$Log(w) = \ln(|w|) + i \arg(w)$

$Log(a + b\,i) = \ln(\sqrt{a^2 + b^2}) + i \arctan(b/a)$

```
Complex.prototype.log = function() {
    var r = Math.log(Math.sqrt(this.r*this.r + this.i*this.i));
    var i = Math.atan2(this.i,this.r);
    return new Complex(r, i);
}
```

```
log((new Complex(0,1)).log());
(0 + 1.57 i)
log((new Complex(-1,0)).log());
(0 + 3.14 i)
```

実行例

先生：複素数のべき乗を計算するメソッドです。

$(\cos(\theta) + i \sin(\theta))^n = \cos(n\theta) + i \sin(n\theta)$

$z^m = r^m e^{im\theta} = r^m (\cos(m\theta) + i \sin(m\theta))$　　$r = |z| = \sqrt{a^2 + b^2}$　　$\theta = \arctan(b/a)$

$m=1 \rightarrow z = r e^{i\theta} = r (\cos(\theta) + i \sin(\theta))$

```
Complex.prototype.pow = function(n) {
    var rr = Math.pow(Math.sqrt(this.r*this.r + this.i*this.i),n);
    var th = Math.atan2(this.i,this.r);
    var r = rr * Math.cos(n*th);
    var i = rr * Math.sin(n*th);
    return new Complex(r, i);
}
```

```
var x = new Complex(-1/2,-Math.sqrt(3)/2);log(x);
(-0.5 - 0.87 i)
log(x.pow(3));
(1 - 0 i)
log(x.mul(x).mul(x));
(1 - 0 i)
```

実行例

第6回 複素数の配列とキャンバス

なな: 複素数を、複素数平面上の点と考えると、図形はどう表現すれば良いの？

先生: 複素数の配列と考えます。複素数の配列データを文字表現したり、図形表現するメソッドを用意しました。

```
var x1 = new Complex(-1/2,-Math.sqrt(3)/2);
var x2 = new Complex(-1/2,Math.sqrt(3)/2);
var x3 = new Complex(1,0);
var a = [x1, x2, x3];
```

複素数の配列はこのように作ります

```
function dispCArray(d) {
    return d.reduce((p,c) => p+c.toString(),"");
}
```

複素数の配列を文字列表現に変換するメソッドを用意しました

```
function Canvas(size) {
    this.canvas = document.createElement('canvas');
    this.canvas.width  = (size===undefined)?400:size;
    this.canvas.height = (size===undefined)?400:size;
    this.canvas.style = "border:solid 1px";
    this.ctx = this.canvas.getContext('2d');
    document.body.appendChild(this.canvas);
}
```

正方形のキャンバスを表示する個なうトラクタです。複数回 new すれば、複数個のキャンバスを作れます。サイズを指定しないと一辺 400 にします。

```
Canvas.prototype.clear = function() {
    var ctx = this.ctx;
    var canvas = this.canvas;
    ctx.clearRect(0,0,this.canvas.width,this.canvas.height);
    return this;
}
```

キャンバスの描画内容を消去するメソッドです

```
Canvas.prototype.drawComplex = function(cpx) {
    var ctx = this.ctx;
    var canvas = this.canvas;
    ctx.beginPath();
    ctx.moveTo(cv(cpx[0].r),canvas.width-cv(cpx[0].i));
    for (var i=1; i<cpx.length; i++)
            ctx.lineTo(cv(cpx[i].r),canvas.width-cv(cpx[i].i));
    ctx.closePath();
    ctx.stroke();
    return this;
    function cv(d) { return (d + 2)*(canvas.width/4); }
}
```

複素数の配列を引数で与えると、要素複素数を頂点とする多角形を描画します。終点～始点を結びます。

```
Canvas.prototype.plotComplex = function(cpx,b) {
    var color = (b===undefined)?0:b;
    var ctx = this.ctx;
    ctx.fillStyle = "rgb(" + color + "," + color + "," + color + ")";
    var canvas = this.canvas;
    ctx.fillRect(cv(cpx.r)-1,canvas.width-cv(cpx.i)-1,2,2);
    ctx.fillStyle = "rgb(" + color + "," + color + "," + color + ")";
    return this;
    function cv(d) { return (d + 2)*(canvas.width/4); }
}
```

引数で複素数を一つ与え、指定した明るさ（0～255）の点を表示する、Canvasクラスのメソッド

```
Complex.prototype.plotComplex = function(c,b) {
    var color = (b===undefined)?0:b;
    var ctx = c.ctx;
    ctx.fillStyle = "rgb(" + color + "," + color + "," + color + ")";
    var canvas = c.canvas;
    ctx.fillRect(cv(this.r)-1,canvas.width-cv(this.i)-1,2,2);
    return this;
    function cv(d) { return (d + 2)*(canvas.width/4); }
}
```

引数で複素数を一つ与え、指定した明るさ（0～255）の点を表示する、Complexクラスのメソッド

実行例

```
var d = new Complex(1,1); log(d);
(1 + 1 i)
var a = [new Complex(1,1), new Complex(2,2), new Complex(3,3)];
log(dispCArray(a));
(1 + 1 i)(2 + 2 i)(3 + 3 i)
log(a.reduce(function(p,c) { return p+c.toString(); },""));
(1 + 1 i)(2 + 2 i)(3 + 3 i)
log(a.reduce((p,c) => p+c.toString(),""));
(1 + 1 i)(2 + 2 i)(3 + 3 i)
```

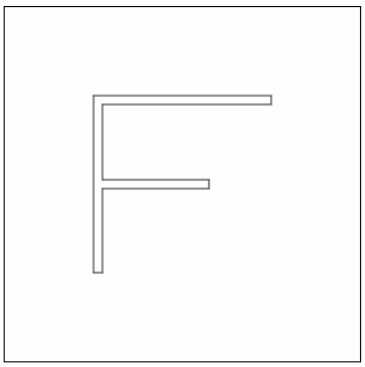
複素数の配列を生成

配列の内容を表示

reduce を使っても表示できます

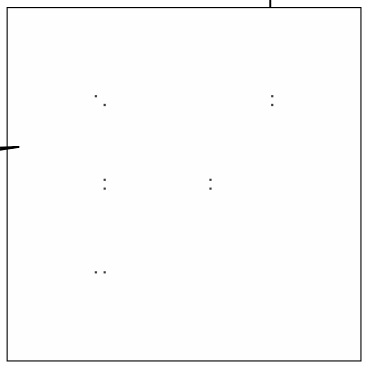
実行例

```
var a = [new Complex(-1,1), new Complex(1,1), new Complex(1,0.9),
    new Complex(-0.9,0.9), new Complex(-0.9,0.05), new Complex(0.3,0.05),
    new Complex(0.3,-0.05), new Complex(-0.9,-0.05), new Complex(-0.9,-1),
    new Complex(-1,-1)];

var c = new Canvas();
c.drawComplex(a);
```



10 個の複素数で表現したＦのパターンを表示

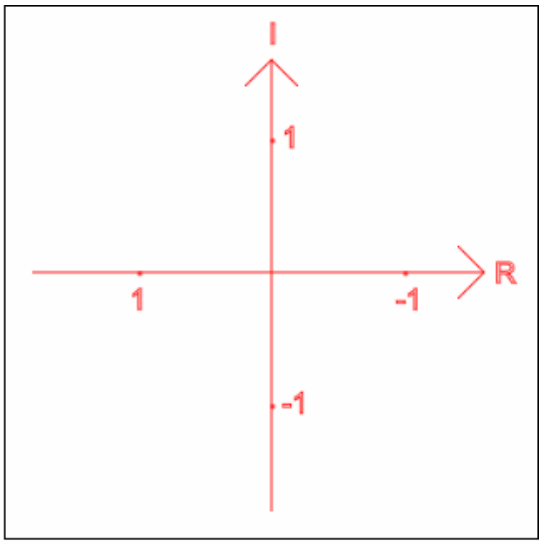
```
var a = [new Complex(-1,1), new Complex(1,1), new Complex(1,0.9),
    new Complex(-0.9,0.9), new Complex(-0.9,0.05), new Complex(0.3,0.05),
    new Complex(0.3,-0.05), new Complex(-0.9,-0.05), new Complex(-0.9,-1),
    new Complex(-1,-1)];
var c = new Canvas();
a.forEach(function(e){e.plotComplex(c);});
```



10 個の複素数で表現したＦのパターンの頂点を点として表示

```
Canvas.prototype.ComplexAxis = function() {
    var ctx = this.ctx;
    var canvas = this.canvas;
    ctx.strokeStyle = "rgb(255,0,0)";
    ctx.beginPath();
    ctx.moveTo(canvas.width*0.05,canvas.height*0.5);
    ctx.lineTo(canvas.width*0.9, canvas.height*0.5);
    ctx.stroke();
    ctx.beginPath();
    ctx.moveTo(canvas.width*0.85,canvas.height*0.45);
    ctx.lineTo(canvas.width*0.9, canvas.height*0.5);
    ctx.lineTo(canvas.width*0.85,canvas.height*0.55);
    ctx.stroke();
    ctx.beginPath();
    ctx.moveTo(canvas.width*0.5,canvas.height*0.95);
    ctx.lineTo(canvas.width*0.5, canvas.height*0.1);
    ctx.stroke();
    ctx.beginPath();
    ctx.moveTo(canvas.width*0.45,canvas.height*0.15);
    ctx.lineTo(canvas.width*0.5, canvas.height*0.1);
    ctx.lineTo(canvas.width*0.55,canvas.height*0.15);
    ctx.stroke();
    ctx.strokeRect(canvas.width*0.25,canvas.height*0.5,2,2);
    ctx.strokeRect(canvas.width*0.75,canvas.height*0.5,2,2);
    ctx.strokeRect(canvas.width*0.5,canvas.height*0.25,2,2);
    ctx.strokeRect(canvas.width*0.5,canvas.height*0.75,2,2);
    ctx.font = "22px sans-serif";
    ctx.strokeText("R",canvas.width*0.92,canvas.height*0.52);
    ctx.strokeText("I",canvas.width*0.495,canvas.height*0.07);
    ctx.strokeText("1",canvas.width*0.52,canvas.height*0.265);
    ctx.strokeText("-1",canvas.width*0.52,canvas.height*0.765);
    ctx.strokeText("1",canvas.width*0.234,canvas.height*0.57);
    ctx.strokeText("-1",canvas.width*0.734,canvas.height*0.57);
    ctx.strokeStyle = "rgb(0,0,0)";
}
```

複素数平面の座標軸を描くメソッド



第7回　さあ、いよいよ使ってみよう！

なな: これまでに説明でててきた複素数演算は、どう使うの？

先生: 複素数演算のプログラムがどうなっているかより、それを使って、複素数の性質をいろいろ調べたり、実験したりすることが大切ね。コンピュータを使わないと、性質を感じるというより、公式を暗記するとか、計算問題になってしまったり、ささいな計算間違いで疲れ果ててしまったり、数学がきらいになってしまいがちだったと思います。ここでは、公式の暗記や、計算はコンピュータに任せて、複素数の性質を感じてみましょう。

```
var x = new Complex(1,1); log(x);
(1 + 1 i)
var y = new Complex(1,1); log(y);
(1 + 1 i)
x == y                      // Boolean boolean false
(x+"") == (y+"")            // Boolean boolean true
x.equal(y)                  // Boolean boolean true
// ------------------------------------------------------------
var a = new Complex(1,2); log(a);
(1 + 2 i)
var b = new Complex(3,4); log(b);
(3 + 4 i)
var c = new Complex(5,6); log(c);
(5 + 6 i)
(a.mul(b)+"") == (b.mul(a)+"")                    // Boolean boolean true
(a.mul(b).mul(c)+"") == (a.mul(b.mul(c)))         // Boolean boolean true
(a.mul(b.add(c))) == ((a.mul(b).add(a.mul(c)))+"") // Boolean boolean true
```

x ← 1 + i

y ← 1 + i

== はアドレス比較

文字列表記で比較

専用メソッド

交換則成立

結合則成立

分配則成立

交換則は、ab = ba、結合則は、(ab) c = a (bc)、分配則は、a (b +c) = ab + ac。

```
var c = new Complex(1,1); log(c);
(1 + 1 i)
var p = c.toPcomplex(); log(p);
< r:1.41, th:0.79(0.25π) >
log(p.toComplex());
(1 + 1 i)
```

極形式相互変換

一般形式→極形式変換

極形式→一般形式変換

1 + i は、極形式では、「1.41 (cos(π/4) + i sin(π/4))」になります。

```
var x = new Complex(-1/2,-Math.sqrt(3)/2); log(x);
(-0.5 - 0.87 i)
log(x.pow(3));
(1 - 0 i)
log(x.mul(x).mul(x));
(1 - 0 i)
log(x.toPcomplex());
< r:1, th:-2.09(-0.67π) >
var y = new Complex(-1/2,Math.sqrt(3)/2);log(y);
(-0.5 + 0.87 i)
log(y.toPcomplex());
< r:1, th:2.09(0.67π) >
```

x = -1/2 - √(3)/2

pow メソッドで 3 乗

mul メソッドで 3 乗

極形式に変換

x = -1/2 + √(3)/2

極形式に変換

-1/2 ± √(3)/2 は、1 とともに、1 の 3 乗根。極形式に変換すると、

1 (cos(±2π/3) + i sin(±2π/3)) になります。

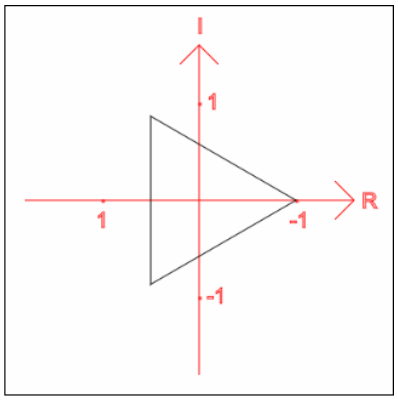
これを複素平面上に表示すると、正三角形の頂点になっていることが分かります。

```
var x1 = new Complex(-1/2,-Math.sqrt(3)/2);
var x2 = new Complex(-1/2,Math.sqrt(3)/2);
var x3 = new Complex(1,0);
var a = [x1, x2, x3];
var c = new Canvas();
c.ComplexAxis();
c.drawComplex(a);
```

配列化

複素平面軸表示

配列を図形化




```
var z1 = new Complex(1,2); log(z1);     // (1 + 2 i)
var z2 = z1.cnj(); log(z2);             // (1 - 2 i)
log(z1.toPcomplex());                   // < r:2.24, th:1.11(0.35π) >
log(z2.toPcomplex());                   // < r:2.24, th:-1.11(-0.35π) >
var z12 = z1.mul(z2); log(z12);         // (5 + 0 i)
```

共役複素数

互いに共役の関係のある複素数の積は実数。値は絶対値（ｒ）の積

第8回　写像としての複素数

なな: 複素数は写像にも使えるということだったけど？

先生: a+bi という複素数は、r (cosθ +isinθ) という「極形式」に変換することができます。ここで、
$r = \sqrt{(a^2 + b^2)}$、$\theta = \tan^{-1}(b/a)$ という関係があります。この複素数を、別の複素数に乗ずるということは、複素数平面上で、

1. 長さ方向に r 倍する（原点からその点に向かうベクトルの長さが r 倍になる）
2. θ だけ回転する（原点を中心として回転）

という操作になります。θ >0 なら左回転、θ <0 なら右回転。

ちなみに、点 z を、原点中心ではなく、点 α 中心に角 θ だけ回転した点 w は

$w = (z - \alpha)(\cos\theta + i\sin\theta) + \alpha$ で計算できます。

また、w = z * A + B という形で、回転に加え、移動も表現できます。

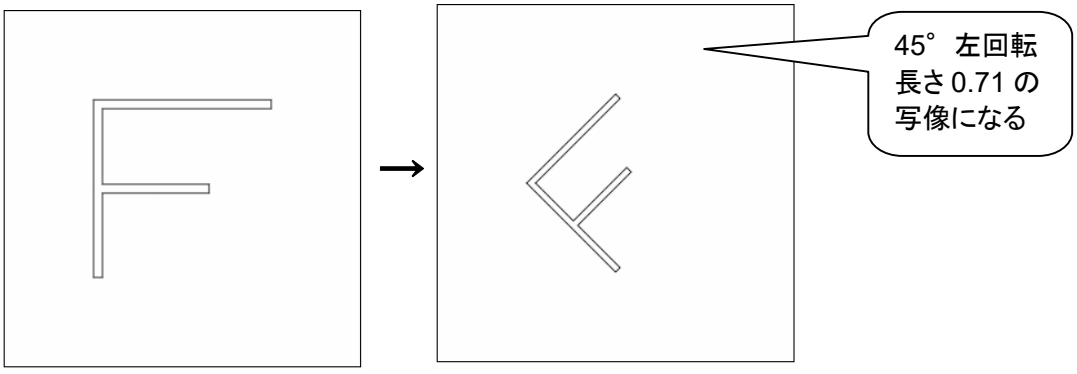
```
var a = [new Complex(-1,1), new Complex(1,1), new Complex(1,0.9),
    new Complex(-0.9,0.9), new Complex(-0.9,0.05), new Complex(0.3,0.05),
    new Complex(0.3,-0.05), new Complex(-0.9,-0.05), new Complex(-0.9,-1),
    new Complex(-1,-1)];

var c = new Canvas();
c.drawComplex(a);

var a2 = a.map(function(c) { return c.mul(new Complex(0.5,0.5)); });
var a2 = a.map((c) => c.mul(new Complex(0.5,0.5)));
c.drawComplex(a2);
log((new Complex(0.5,0.5)).toPcomplex());
< r:0.71, th:0.79(0.25π) >
```

複素平面上での F の形

各点に 0.5 + 0.5 i を掛ける

ECMAScript 2015～のアロー関数を使う場合

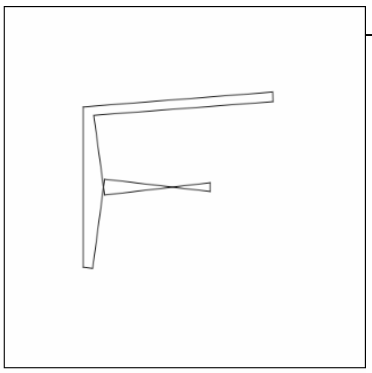
極形式。
0.25π = 45° 左回転、長さ 0.71 倍



45° 左回転 長さ 0.71 の写像になる

先生：ａ＋ｂｉを掛けるという操作は線形操作であり、回転と移動の範囲です。形は変わらず、相似形という形になります。これに対し、二乗とか、逆数というような非線形操作を加えると、形が変わります。元の形が想像できないほど変形してしまうと、何が起きているのか分からなくなるので、ここでは、「わずかな非線形操作」を実験してみましょう。1.05 乗とか、0.95 乗を試してみましょう。

```
var a = [new Complex(-1,1), new Complex(1,1), new Complex(1,0.9),
    new Complex(-0.9,0.9), new Complex(-0.9,0.05), new Complex(0.3,0.05),
    new Complex(0.3,-0.05), new Complex(-0.9,-0.05), new Complex(-0.9,-1),
    new Complex(-1,-1)];
c = new Canvas();
var a2 = a.map((c) => c.pow(1.05));
c.drawComplex(a2);
```



```
var a = [new Complex(-1,1), new Complex(1,1), new Complex(1,0.9),
    new Complex(-0.9,0.9), new Complex(-0.9,0.05), new Complex(0.3,0.05),
    new Complex(0.3,-0.05), new Complex(-0.9,-0.05), new Complex(-0.9,-1),
    new Complex(-1,-1)];
c = new Canvas();
var a2 = a.map((c) => c.pow(0.95));
c.drawComplex(a2);
```