ななちゃんのIT教室

カンフーじゃないの？（関数）の巻

by nara.yasuhiro@gmail.com

ななちゃんが、JavaScript の
関数の使い方の基礎を学ぶという お話

第 0.1 版　2017 年 6 月 17 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　関数の使い途

なな：カンスーって何？　カンフーと関係あるの？

先生：関数ね。JavaScript のプログラムのかたまりに名前をつけたものよ。下記の上が定義、下が呼び出し。

```
function add(x, y) {
    return x + y;
}

add(1, 2)          // Number number 3
```

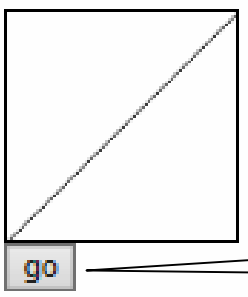
値が数値型の 3 ということ。
alert(add(1,2)) と書けば、
「3」と表示するアラート画面が出る
ということ。

なな：関数を使うと、どんなご利益があるの？

先生：関数の名前を適切に決めれば、その部分のプログラムの役目が分かりやすくなるので、プログラムが読みやすくなるという利点があります。それから、プログラムの多くの箇所から呼び出すなら、プログラム行数の削減につながります。それから、その部分のアルゴリズムの改良を行う場合、一か所を修正するだけで済みます。さらに、使いやすい、良い関数ができた場合、他人にコピーして利用してもらうことができます。ファイルに書き込んでおき、<script src=…>で読み込むこともできます。複数のプログラムや、プログラマ間で、共有することができます。便利な関数をいくつかまとめたものをライブラリといいます。インターネット上で、多くのライブラリが公開されています。

下記のプログラムでは、関数で計算した結果をグラフ表示しています。この関数を書き換えることで、別の関数のグラフを表示することができます。一か所からしか呼び出さない関数ですが、プログラムの可読性（人間による読みやすさ、理解のしやすさ）に貢献していると思います。

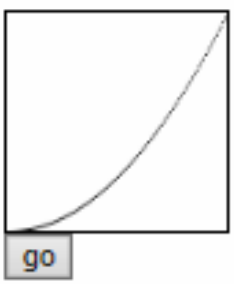
```
<style>canvas { border: solid 1px; }</style>
<canvas width=100 height=100 id=canvas></canvas>
<br><input type=button onclick=go() value=go>
<script>
var canvas = document.getElementById('canvas');
var ctx = canvas.getContext('2d');

function go() {
    for (var x=0; x<1; x+=0.01) {
        var y = func(x);
        ctx.fillRect(x*100, 100-y*100, 1, 1);
    }
}

function func(x) {
    return x;
}
</script>
```



「go」をクリックすると描画

```
function func(x) {
    return x*x;
}
```

このように書き換えると

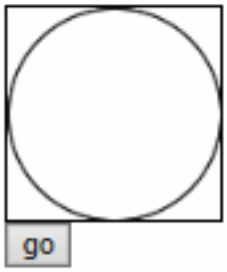


第2回　データをふたつ返すには？

なな：関数からデータを複数個返すことはできるの？

先生：配列やオブジェクトの形で返せば実現できます。下記は、配列を使う例です。

```
<style>canvas { border: solid 1px; }</style>
<canvas width=100 height=100 id=canvas></canvas>
<br><input type=button onclick=go() value=go>
<script>
var canvas = document.getElementById('canvas');
var ctx = canvas.getContext('2d');

function go() {
    for (var r=0; r<2*Math.PI; r+=0.01) {
        var d = func(r);
        var x = d[0];
        var y = d[1];
        ctx.fillRect((x+1)*50, 100-(y+1)*50, 1, 1);
    }
}

function func(r) {
    return [Math.cos(r), Math.sin(r)];
}
</script>
```



先生：ECMAScript 2015 からは、分割代入も可能になりました。

```
<style>canvas { border: solid 1px; }</style>
<canvas width=100 height=100 id=canvas></canvas>
<br><input type=button onclick=go() value=go>
<script>
var canvas = document.getElementById('canvas');
var ctx = canvas.getContext('2d');

function go() {
    var x, y;
    for (var r=0; r<2*Math.PI; r+=0.01) {
        [x, y] = func(r);
        ctx.fillRect((x+1)*50, 100-(y+1)*50, 1, 1);
    }
}

function func(r) {
    return [Math.cos(r), Math.sin(r)];
}
</script>
```

先生：下記は、return ではなく、引数として配列を渡し、その中身を書き換えることでデータを返す例です。配列は値渡しではなく、参照渡しであることを利用しています。

```
<style>canvas { border: solid 1px; }</style>
<canvas width=100 height=100 id=canvas></canvas>
<br><input type=button onclick=go() value=go>
<script>
var canvas = document.getElementById('canvas');
var ctx = canvas.getContext('2d');

function go() {
    var x, y, d = [0,0];
    for (var r=0; r<2*Math.PI; r+=0.01) {
        func(r, d);
        x = d[0];
        y = d[1];
        ctx.fillRect((x+1)*50, 100-(y+1)*50, 1, 1);
    }
}

function func(r, d) {
    d[0] = Math.cos(r);
    d[1] = Math.sin(r);
}
</script>
```

配列のアドレスをもらい、
配列にデータを書き込んでいる

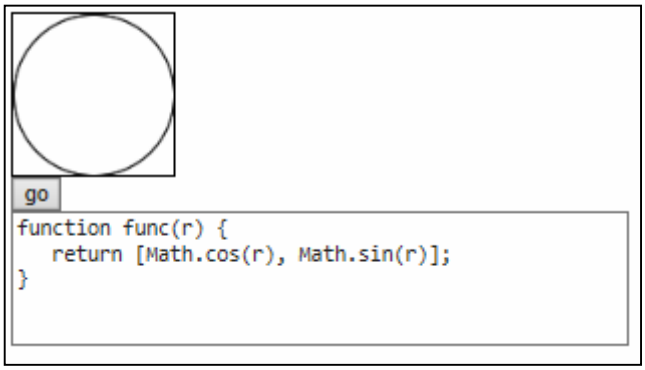
第3回　実行時に関数を定義する

なな：描画対象の関数をエディタで書き換えて実行しなおさないと変更できないのは不便ね。関数を実行時に書き換える（定義し直す）ことはできないの？

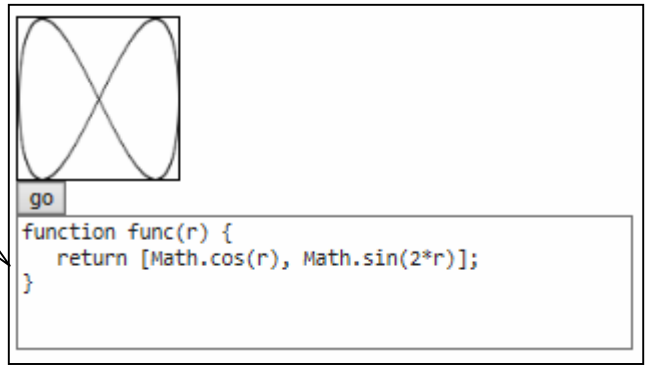
先生：できます。eval() を使って、実行中に関数を再定義することができます。

```
<style>canvas { border: solid 1px; }</style>
<canvas width=100 height=100 id=canvas></canvas>
<br><input type=button onclick=go() value=go>
<br><textarea cols=50 rows=5 id=pg>
function func(r) {
    return [Math.cos(r), Math.sin(r)];
}
</textarea>
<script>
var canvas = document.getElementById('canvas');
var ctx = canvas.getContext('2d');

function go() {
    ctx.clearRect(0,0,100,100);
    eval(document.getElementById('pg').value);
    for (var r=0; r<2*Math.PI; r+=0.01) {
        var d = func(r);
        var x = d[0];
        var y = d[1];
        ctx.fillRect((x+1)*50, 100-(y+1)*50, 1, 1);
    }
}
</script>
```

関数定義（文字列）を読み出して評価している



この欄を書き換えて、「go」ボタンをクリックすると、描画内容が変わる。



先生：関数定義のプログラムを、HTML の script タグ追加で評価することもできます。

```
<style>canvas { border: solid 1px; }</style>
<canvas width=100 height=100 id=canvas></canvas>
<br><input type=button onclick=go() value=go>
<br><textarea cols=50 rows=5 id=pg>
function func(r) {
    return [Math.cos(r), Math.sin(r)];
}
</textarea>
<script>
var canvas = document.getElementById('canvas');
var ctx = canvas.getContext('2d');

function draw() {
    var x, y;
    for (var r=0; r<2*Math.PI; r+=0.01) {
        [x, y] = func(r);
        ctx.fillRect((x+1)*50, 100-(y+1)*50, 1, 1);
    }
}

function go() {
    var sc = document.createElement("script");
    sc.innerHTML = document.getElementById('pg').value
                 + "draw()";
    document.body.appendChild(sc);
}

</script>
```

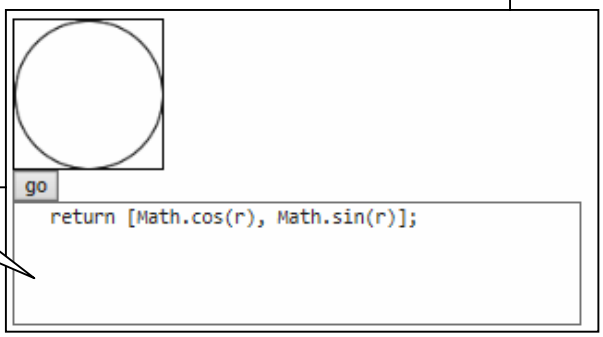
<script>
function func(r) {
    return [Math.cos(r), Math.sin(r)];
}
draw()
</script>
というタグを<body>に動的に追加。

評価が完了すると
draw() が
呼び出される

先生：それから、Function オブジェクトを、コンストラクタから生成することもできます。

```
<style>canvas { border: solid 1px; }</style>
<canvas width=100 height=100 id=canvas></canvas>
<br><input type=button onclick=go() value=go>
<br><textarea cols=50 rows=5 id=pg>
    return [Math.cos(r), Math.sin(r)];
</textarea>
<script>
var canvas = document.getElementById('canvas');
var ctx = canvas.getContext('2d');

function go() {
    ctx.clearRect(0,0,100,100);
    var func = new Function('r',document.getElementById('pg').value);
    for (var r=0; r<2*Math.PI; r+=0.01) {
        var d = func(r);
        var x = d[0];
        var y = d[1];
        ctx.fillRect((x+1)*50, 100-(y+1)*50, 1, 1);
    }
}

</script>
```

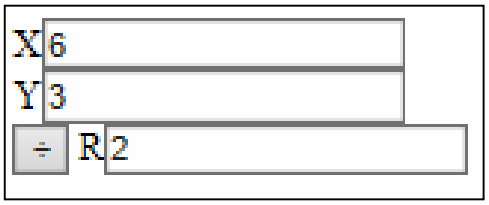
ここを買い替えて
「go」ボタンを
クリックすると描画



第4回　便利な関数はライブラリ化

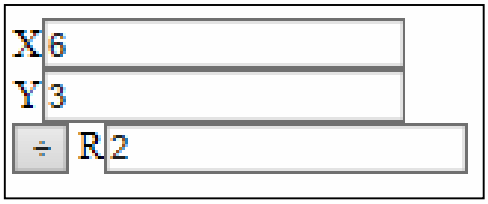
なな：「document.getElementById("…").value」って、長たらしくて、いつもタイプミスするんだけど。

```
X<input type=text id=x><br>
Y<input type=text id=y><br>
<input type=button onclick=calc() value="÷">
R<input type=text id=r><br>
<script>
function calc() {
    var x = document.getElementById("x").value;
    var y = document.getElementById("y").value;
    var r = x / y;
    document.getElementById("r").value = r;
}
</script>
```



先生：「document.getElementById("…").value」を、「$」という名前の関数として定義すると楽よ。
「x = document.getElementById("x").value」の代わりに「x = $("x");」
「document.getElementById("r").value = r」の代わりに「$("r",r)」と書けばよくなります。

```
X<input type=text id=x><br>
Y<input type=text id=y><br>
<input type=button onclick=calc() value="÷">
R<input type=text id=r><br>
<script>
function $ (ipt, data) {
    var port = document.getElementById(ipt);
    if (data == undefined) return port.value;
    else port.value = data;
}
</script>
<script>
function calc() {
    var x = $("x");
    var y = $("y");
    var r = x / y;
    $("r",r);
}
</script>
```



こういうのをたくさんまとめてライブラリにしたものとして、「jQuary」なんかが有名です。

第5回　チーム開発のプライバシー保護

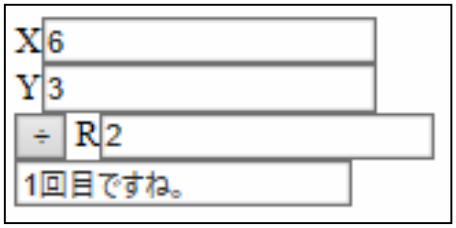
なな：前回のプログラムを山田君が作ってくれたので、私が、それを 3 回繰り返すプログラムに拡張しようとしているんだけど、うまく動作しないの。 1 回目だけで終わってしまうの。

```
X<input type=text id=x><br>
Y<input type=text id=y><br>
<input type=button onclick=go() value="÷">
R<input type=text id=r><br>
<input type=text id=m><br>
<script>
function $ (ipt, data) {
    var port = document.getElementById(ipt);
    if (data == undefined) return port.value;
    else port.value = data;
}
</script>
<script>
var x, y, r;
function calc() {
    x = $("x");
    y = $("y");
    r = x / y;
    $("r",r);
}
</script>
<script>
$("m","3 回練習しましょう！");
var x = 1;
function go() {
    if (x <= 3) {
        $("m",x + "回目ですね。");
        calc();
    }
    x++;
    if (x == 4) $("m","お疲れ様！");
}
</script>
```

山田君が開発した部分

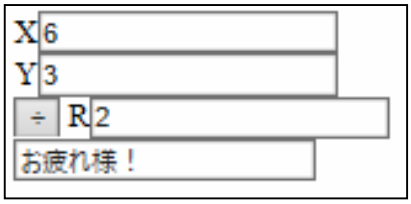
ななちゃんが開発した部分

山田君が開発した部分を呼び出し



先生：ななちゃんは、回数を調べるのに、変数 x を使っているけど、山田君も x を使っているので、衝突してしまったようね。山田君のプログラムで、x が 6 になるから、6 回目をやったように見えてしまい、終了してしまうのよ。ななちゃんが、x のかわりに、 i とか、n とかを使えば大丈夫。

なな：動いた！



先生：関数の外で定義した変数があると、衝突する可能性があるの。プログラムをチーム開発しているとき、プログラムが大きくなると、「僕は x, y, i, j, k, n を使っているから、他の人は使わないでね」、「え？ もう使っているよ。僕のプログラム書き直すの面倒だな」などということが起きてしまうの。

なな：困るわね。

先生：さっきの例では、山田君が、

```
var x, y, r;
function calc() {
    x = $("x");
    y = $("y");
    r = x / y;
    $("r",r);
}
```

→

```
function calc() {
    var x, y, r;
    x = $("x");
    y = $("y");
    r = x / y;
    $("r",r);
}
```

のように、x を、関数内で宣言すれば、ローカル変数になって、ななちゃんの x とぶつからなくなります。でも、山田君の部分が、これから、もっと複雑になって、山田君の複数の関数間で、x を共有したい場合などに困ることになります。

下記のように、山田君のプログラム全体を、さらに関数の中に入れてしまい、他の人に利用してもらう関数だけを return で通知するようにすれば、衝突を防げます。山田君は、他の人のことは気にせずに、関数の外側に 変数を定義して大丈夫です。

```
X<input type=text id=x><br>
Y<input type=text id=y><br>
<input type=button onclick=go() value="÷">
R<input type=text id=r><br>
<input type=text id=m><br>
<script>
function $ (ipt, data) {
    var port = document.getElementById(ipt);
    if (data == undefined) return port.value;
    else port.value = data;
}
</script>
<script>
yamada = (function() {

var x, y, r;
function calc() {
    x = $("x");
    y = $("y");
    r = x / y;
    $("r",r);
}

return { "calc":calc }; })()
</script>
<script>
var x;
$("m","3 回練習しましょう！");
x = 1;
function go() {
    if (x <= 3) {
        $("m",x + "回目ですね。");
        yamada.calc();
    }
    x++;
    if (x == 4) $("m","お疲れ様！");
}
</script>
```

関数の内部で定義した変数はローカル変数になり、関数外に見えない、というのがポイント！

関数で包み込む

他の人に利用してもらう関数をリターン。山田君の calc() を、他の人に calc() という名前で使ってもらう。名前を変えることもできます。

包み込む関数は、すぐに実行

ななちゃんが、山田君の関数を利用している

ななちゃんは、yamada という名前の変数以外は、自由に定義して使ってよい。
var calc; も OK。

なな：「東京都 多摩市 桜が丘」と、「神奈川県 横浜市 保土ヶ谷区 桜が丘」が区別できるのと同じ理屈ね！