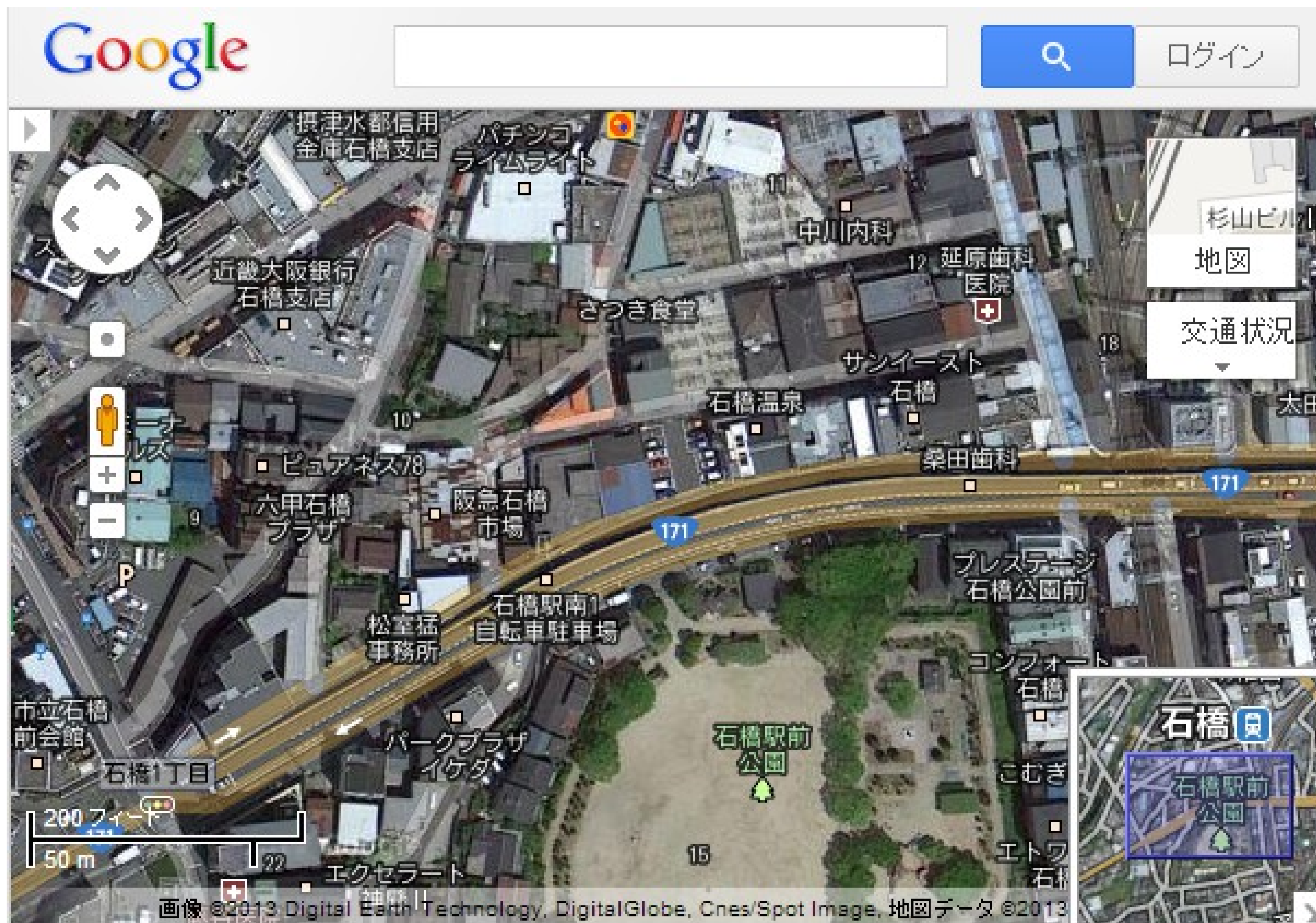
JavaScript講習会

+オブジェクト指向のなんたら

2013年4月20日
@大阪大学コンピュータクラブ

JavaScriptといえば？

Webページ上で色々するやつ？



© Google

今回は全く触れません

本日の目標

簡易電卓を作ろう

※文字列として式を読み取って計算するプログラム

とりあえず

JavaScriptの基礎

JavaScriptとは

プロトタイプベースのオブジェクト指向言語

・何言ってるんだこいつ

いろんな環境で動く

・ブラウザ (Firefox, Chrome, Opera等)
・サーバー向けの環境 (Node.js)
・ゲーム用フレームワーク (Unity等)
・マクロとか (Windowsにも標準搭載されている)

Webページに対して操作するだけじゃない

覚えておいて損は無さそう

注意

Javaとは違う

インドとインドネシアくらい違う

JavaScript基礎(1)

データ型 (データの種類)

プリミティブ型

・Number (数値)
・String (文字列)
・Boolean (真偽値)

参照型

・Object (連想配列)
・Array (配列)
・Function (関数)

他にもいくつかあるけど省略

JavaScript基礎(2)

Number (数値) の扱い

基本的に浮動小数です (ビット演算時のみ32bit符号付き整数)

加減乗除できます (+, -, *, /)
modもできます (%)
ビット演算とかもできます (&, |, ^, ~とか)

例)

```
((1 + 2) * (3 - 4) + 5) / 6
14 % 8
(6 & 4) | 2
```

大して予想外なことも無いと思います

JavaScript基礎(3)

String (文字列) の扱い

文字列です。

+ で連結できます

例)

```
“Hello, World!”
“Java”  +  “Script”
```

まあわかるよね

JavaScript基礎(4)

Boolean (真偽値) の扱い

真 (true) か偽 (false) かを扱います
「かつ」「または」「〜でない」って演算できます (&&, ||, !)

例)

```
true && false
true || false
!true
```

条件文とかでよく使います

JavaScript基礎(5)

Object (連想配列) の扱い

色々とデータを格納できます
言うよりも見たほうが早い

例)

```
{a : 1, b : 2, name : “Avocado” , isHuman : false}
{name : “Ichiro” , child :
   {name: “Jiro” , child : null}
}
{x : 10, y : 20}.x
```

このときのxをオブジェクトのxプロパティとか言ったりする

JavaScript基礎(6)

Array (配列) の扱い

連続したデータを格納できます

例)

```
[1, 1, 2, 3, 5, 8, 13, 21, 34]
[2, “foo” , 4, “bar” , 8, “baz” ]
[2, 3, 5, 7, 11, 13, 17, 19][3]
[0, 1, 2, 3, 4].length
```

JavaScript基礎(7)

データの比較

==　　等しいか?

!=　　等しくないか?

(大抵の場合) Numberに対して

>　　より大きい

<　　未満

>=　　以上

<=　　以下

例)

```
"test" == "rest"
2 <= 4
```

JavaScript基礎(8)

変数

データの容れ物 (ってよく言うよね)

例)

```
var a = 2
var str = “string”
var test = true
var obj = {x : 100, y : 200}
obj.x
var arr = [1, 2, 4, 8, 16, 32, 64]
arr[4]
```

JavaScript基礎(9)

変数(2)

最初に言ったプリミティブ型と参照型の違いに注意

```
var a = 2;
var b = a;
b = 5;
// この時点でaは2のまま

var obj1 = {x : 100, y : 200};
var obj2 = obj1;
obj2.x=10;
// この時点でobj1.xは10
```

ところで

こういうことして大丈夫？

```
var str = “Hello” ;
str = 20;
```

→大丈夫です
JavaScriptの変数に型はありません

JavaScript基礎(10)

制御構造

条件分岐

```
if(expr1){
    //A
}
else if(expr2){
    //B
}
else{
    //C
}
```

見ての通り、exp1が真 (true) ならAを、そうでなければ
exp2がtrueならBを、そうでなければCを実行する。

JavaScript基礎(11)

制御構造

繰り返し

```
for(expr1; expr2; expr3){
}

while(expr){
}
```

forはexp1を最初に実行、「exp2が真ならば{}内を実行し、exp3を実行」を繰り返す。

whileは「expが真ならば{}内を実行」を繰り返す。

JavaScript基礎(12)

関数

```
function 関数名(引数1, 引数2, ...){
    処理
    return 返り値;
}
```

呼び出す時は

関数名(引数1, 引数2, …)

JavaScript基礎(13)

例)

例えば足し算。

```
function add(arg1, arg2){
    return arg1 + arg2;
}

add(2, 3)
add(-1, 10)
add( “ECMA” ,  “Script” )
```

JavaScript基礎(14)

例)

階乗。

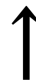
```
function fact(n){
    if(n <= 0){
        return 1;
    }
    else{
        return n * fact(n-1);
    }
}

fact(0)
fact(5)
fact(10)
```

注意

関数もオブジェクトです
(というか、データは全てオブジェクトです)

```
var func = function(message){
  print(message);
};
print(func.length);
```



こういうこともできます

※printは出力するための関数だと思って下さい
環境によって出力関数の名前は異なります

ちょっと休憩

練習

フィボナッチ数列の第n項を求める関数を作ろう
(できれば通常のループと再帰で2種類)

```
function fib(n){?}
```

JavaScript基礎 (15)

コンストラクタ

関数を特殊な呼び出し方 (new) をすることで、
オブジェクトを構築するのに使う。

```
function Point(x, y)
{
    this.x = x;
    this.y = y;
}
var point = new Point(10, 20);
print(point.x);   // 10
print(point.y);   // 20
```

JavaScript基礎 (16)

プロトタイプ

関数のprototypeプロパティをいじくってやる

```
function Point(x, y)
{
    this.x = x;
    this.y = y;
}
Point.prototype.getLength = function(){
    // Math.sqrtは平方根を返す関数
    return Math.sqrt(this.x * this.x + this.y * this.y);
};
var point = new Point(10, 20);
print(point.getLength()); // 22.360679774997898
```

Pointによって生成されたオブジェクト全てでgetLengthが使える

JavaScript基礎 (17)

プロトタイプ継承

Pointはさっき定義したやつ

```
function Circle(x, y, radius)
{
    this.x = x;
    this.y = y;
    this.radius = radius;
}
Circle.prototype = new Point(0, 0);
var circle = new Circle(10, 10, 5);
print(circle.getLength()); //14.142135623730951
```

CircleでもPointと同じgetLength関数 (メソッドとも言う)
が使える！

いよいよ本題に入ります

1+2

何と読みますか？

1足す2
1 + 2

1と2を足す
1 2 +

Add 1 to 2
+ 1 2

なんか3通りくらいありそう

中置記法

((1 + 2) * (3 - 4) + 5) / 6

後置記法

1 2 + 3 4 - * 5 + 6 /

前置記法

/ + * + 1 2 - 3 4 5 6

後置記法、前置記法では括弧が要らない！
→構文解析が楽

今回は後置記法の式を解析して、
前置記法・中置記法に変換しつつ
計算結果を表示するプログラムを作るよ。

式の構造を見てみる

((1 + 2) * (3 - 4) + 5) / 6

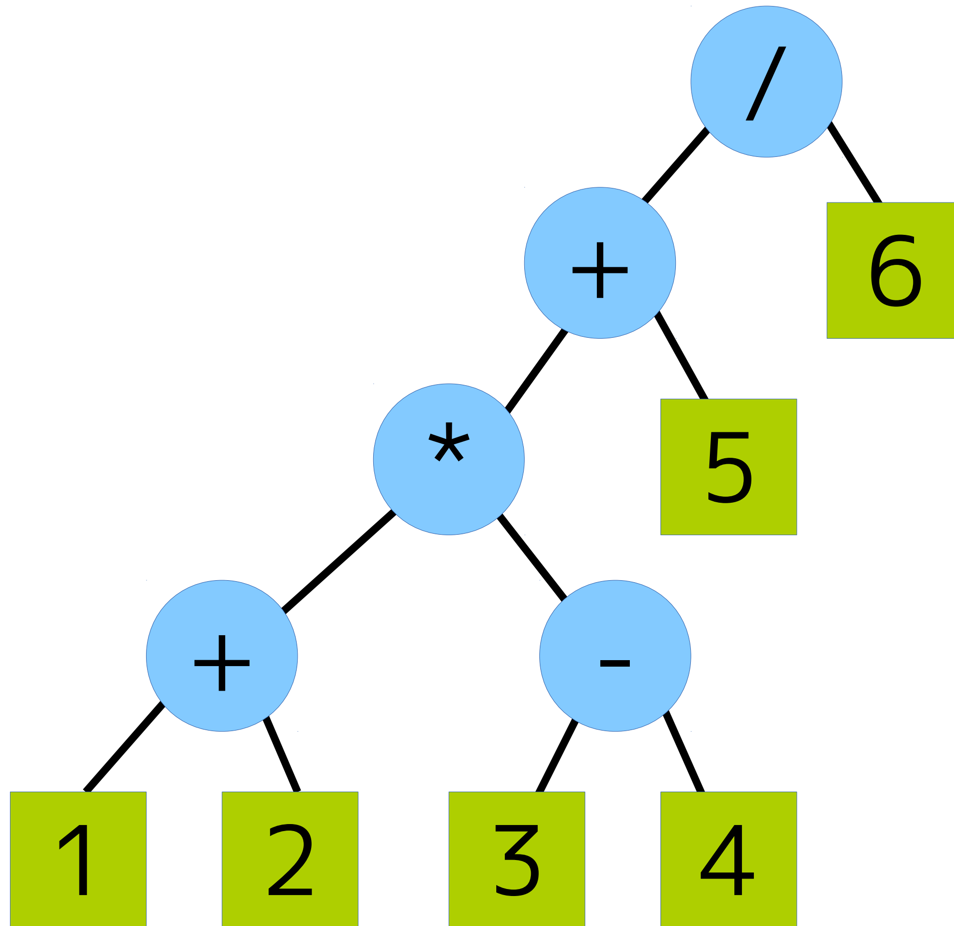
1 2 + 3 4 - * 5 + 6 /

/ + * + 1 2 - 3 4 5 6

扱いやすい形を考える

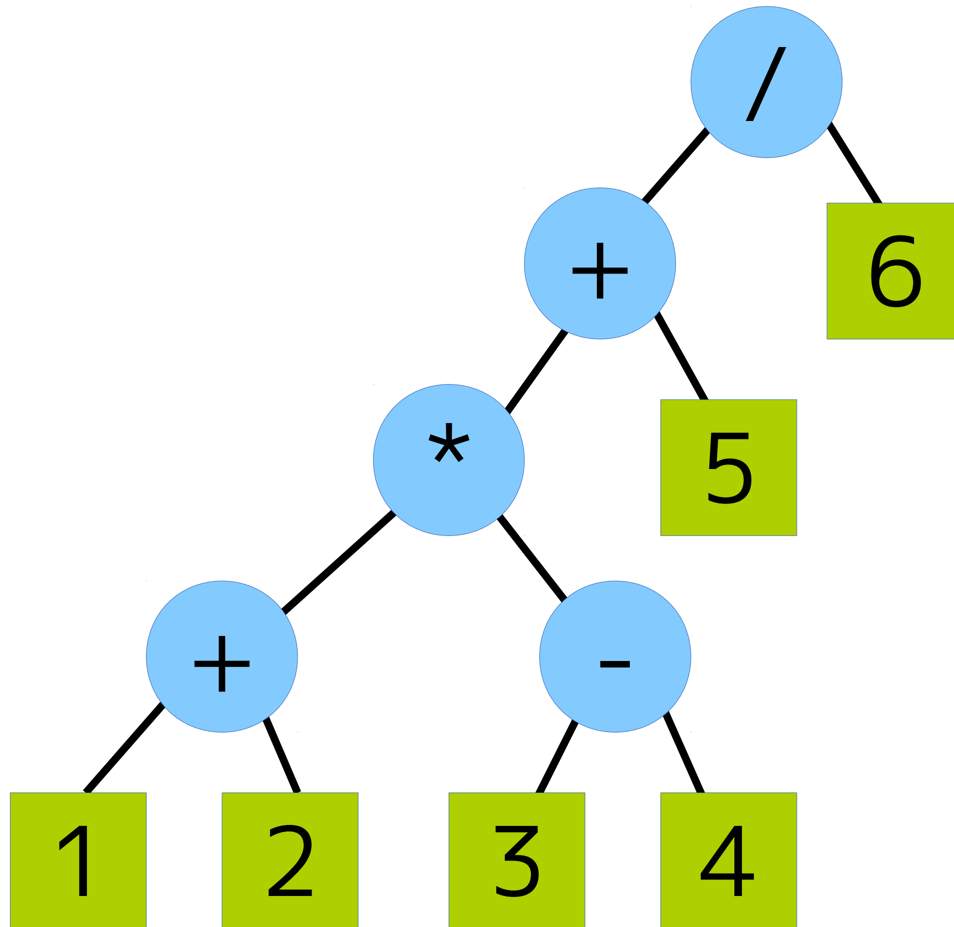
式の構造を見てみる

1 2 + 3 4 - * 5 + 6 /



ツリー構造になってる

式の構造を見てみる



←このツリーの各節点を「ノード」と呼びます

ノードに対応するオブジェクトを作ったら色々できそう

やってみる

ノードオブジェクトのコンストラクタを作る

```
function Node(){
}
```

これを拡張して数値と演算子のノードをそれぞれ作る

prototypeに何もしない関数を（一応）入れておく

→インターフェースのようなものとして機能（してほしい）
　そのメソッドが存在することを保証する

```
//前置記法に変換
Node.prototype.toPrefixNotation = function(){
    return "";
};
//中置記法に変換
Node.prototype.toInfixNotation = function(){
    return "";
};
//後置記法に変換
Node.prototype.toPostfixNotation = function(){
    return "";
};
//計算
Node.prototype.calc = function(){
    return NaN;
};
```

数値ノードを作る

Nodeを拡張してNumberNodeを作る

```
function NumberNode(value){
    //値（数値）
    this.value = value;
}
NumberNode.prototype = new Node();
```

メソッドを作成

※toStringは文字列表現を返すメソッド

```
NumberNode.prototype.toPrefixNotation = function(){
    return this.value.toString();
};
NumberNode.prototype.toInfixNotation = function(){
    return this.value.toString();
};
NumberNode.prototype.toPostfixNotation = function(){
    return this.value.toString();
};
NumberNode.prototype.calc = function(){
    return this.value;
};
```

演算子ノードを作る

Nodeを拡張してOperatorNodeを作る

```
function OperatorNode(type, left, right){
    // 演算子の種類 (+, -, *, /)
    this.type = type;
    // 左のノード
    this.left = left;
    // 右のノード
    this.right = right;
}
OperatorNode.prototype = new Node();
```

メソッドを作成

前置記法と後置記法は簡単に書ける

```
OperatorNode.prototype.toPrefixNotation = function(){
   return this.type + " "
      + this.left.toPrefixNotation() + " "
      + this.right.toPrefixNotation();
};

OperatorNode.prototype.toPostfixNotation = function(){
   return this.left.toPostfixNotation() + " "
      + this.right.toPostfixNotation() + " "
      + this.type;
};
```

メソッドを作成

中置記法は適宜括弧をつけなければダメ

```
OperatorNode.prototype.toInfixNotation = function(){
    var leftExp;
    if((this.type == "*" || this.type == "/")
        && (this.left.type == "+" || this.left.type == "-")){
        leftExp = "(" + this.left.toInfixNotation() + ")";
    }
    else{
        leftExp = this.left.toInfixNotation();
    }
    // (続く)
```

メソッドを作成

```
    // (続き)
    var rightExp;
    if((this.type == "*" || this.type == "/")
        && (this.right.type == "+" || this.right.type == "-")){
        rightExp = "(" + this.right.toInfixNotation() + ")";
    }
    else{
        rightExp = this.right.toInfixNotation();
    }
    return leftExp + " " + this.type + " " + rightExp;
};
```

メソッドを作成

計算は演算子の種類によって条件分岐

```
OperatorNode.prototype.calc = function(){
    if(this.type == "+"){
        return this.left.calc() + this.right.calc();
    }
    else if(this.type == "-"){
        return this.left.calc() - this.right.calc();
    }
    else if(this.type == "*"){
        return this.left.calc() * this.right.calc();
    }
    else if(this.type == "/"){
        return this.left.calc() / this.right.calc();
    }
    else{
        return NaN;
    }
};
```

後置記法の式の解析

1 2 + 3 4 - * 5 + 6 /

アイデア

・スペース区切りで最初から順番に見ていき、ノードに変換
・どこかに保存しておいて、演算子が来たら繋げる

後置記法の式の解析

１２＋３４－＊５＋６／

1. 1をNumberNode(1)としてスタック (山) に積む
   現在のスタック：N(1)

2. 2をNumberNode(2)としてスタックに積む
   現在のスタック：N(1), N(2)

3. 演算子が来た！
   スタックの上2つを取り出す (後ろから順にs1, s2とする)
   +をOperatorNode(“+”, s2, s1)としてスタックに積む
   現在のスタック：O(“+”, N(1), N(2))

後置記法の式の解析

1 2 + 3 4 - * 5 + 6 /

4. 3をNumberNode(3)としてスタック (山) に積む
   現在のスタック：O(“+”, N(1), N(2)), N(3)

5. 4をNumberNode(4)としてスタックに積む
   現在のスタック：O(“+”, N(1), N(2)), N(3), N(4)

6. 演算子が来た！
   スタックの上2つを取り出す (後ろから順にs1, s2とする)
   -をOperatorNode(“-”, s2, s1)としてスタックに積む
   現在のスタック：O(“+”, N(1), N(2)), O(“-”, N(3), N(4))

後置記法の式の解析

1 2 + 3 4 - * 5 + 6 /

7. 演算子が来た！
   スタックの上2つを取り出す (後ろから順にs1, s2とする)
   をOperatorNode(“”, s2, s1)としてスタックに積む
   現在のスタック：O(“*”, O(“+”, N(1), N(2)),
   O(“-”, N(3), N(4)))
8. …

こういう感じで処理していき、最後にスタックに残ったものが
欲しいツリー構造になっている。

こんな感じで (色々省略)

```
function parsePostfixNotation(expression){
    var arr = expression.split(" ");
    var len = arr.length;
    var stack = [];
    for(var i = 0; i < len; i++){
        var elem = arr[i];
        if(!isNaN(parseFloat(elem))){
            stack.push(new NumberNode(parseFloat(elem)));
        }
        else{
            var right = stack.pop();
            var left = stack.pop();
            stack.push(new OperatorNode(elem, left, right));
        }
    }
    return stack[0];
}
```

お疲れ様です。完成です。

ね、簡単でしょ？

実際に動かしてみる

```
var node = parsePostfixNotation("1 2 + 3 4 - * 5 + 6 /");
//中置記法で出力
print(node.toInfixNotation());
//前置記法で出力
print(node.toPrefixNotation());
//後置記法で出力
print(node.toPostfixNotation());
//計算結果を出力
print(node.calc());
```

オブジェクト指向とは

オブジェクト同士の相互作用として、
システムの振る舞いをとらえる考え方である。

　　ってWikipedia先生が言ってた

今回はノードをオブジェクトとして、それらの組み合わせで式を表現した。

オブジェクト指向プログラミング

ポリモーフィズム (多態性)

あるオブジェクトへの操作が呼び出し側ではなく、
受け手のオブジェクトによって定まる特性。

ってWikipedia先生が言ってた

今回の場合、Node, NumberNode, OperatorNodeに対して、
呼び出し側では全て同じ操作をしている (toInfixNotation, calc等)
→変に条件分岐を書かなくてもいいのでかなり楽になる

これが無いと、NumberNodeに対してはこの処理、OperatorNodeに対してはこの処理といったように、呼び出し側で分岐しなければならない。

オブジェクト指向プログラミング

カプセル化 (情報隠蔽)

オブジェクト内部のデータを隠蔽したり(データ隠蔽)、
オブジェクトの振る舞いを隠蔽したり、
オブジェクトの実際の型を隠蔽したりすることをいう。

ってWikipedia先生が言ってた

今回は使用していないが、例えばOperatorNodeのtypeを
後から不用意に変更されないようにすることなどができる。

カプセル化の例

コンストラクタ

```
Color(r, g, b) //new Color(255, 255, 255)
```

メソッド

```
getRGB() //{r : 255, g : 255, b : 255}
getHSV() //{h : 0, s : 0, v : 255}
```

↓

突然の変更

```
Color(h, s, v) //new Color(0, 0, 255)
```

Colorを生成する箇所以外は変更しなくていい！
(getRGBとgetHSVの中身さえ適当に変更すれば)

おわり

おまけ

今回作った電卓(?)のプログラム一式は
http://ideone.com/RUmcJ4
に置いておきました。

この資料もそのうちOUCCのページに
アップしておきます

おまけ

お手元のブラウザでJavaScriptを動かすには

1. 以下の内容をテキストエディタ (メモ帳とか)で入力し、
   (適当な名前).htmlという名前で保存

```
<html>
<head>
<title>なんでもいい</title>
<script>
//ここにJavaScriptを書く
//ただしprintはalertもしくはdocument.writeに置き換えること
</script>
</head>
<body>
</body>
</html>
```

2. ブラウザで開く (たぶんダブルクリックとかすれば良いです)

参考になりそうなサイト

- JavaScript初級者から中級者になろう
  http://uhyohyohyo.sakura.ne.jp/javascript.html
  初歩的なことから非常に丁寧に解説されているのでおすすめ。
  (今回言及できなかった、Webブラウザに絡んだことについても詳しく解説されています)

- MDNのJavaScript リファレンス
  http://developer.mozilla.org/ja/docs/JavaScript/Reference
  文法や用語にある程度慣れているならこちら。
  仕様などは網羅されている (はず)

くぅ～疲れましたw これにて完結です！
実は、ネタレスしたら講習会の話を持ちかけられたのが始まりでした
本当は話のネタなかったのですが←
ご厚意を無駄にするわけには行かないので流行りのネタで挑んでみた所存ですw
以下、ブラウザ達のみんなへのメッセジをどぞ

IE「みんな、見てくれてありがとう
 ちょっと腹黒なところも見えちゃったけど・・・気にしないでね！」

Chrome「いやーありがと！
 私のかわいさは二十分に伝わったかな？」

Opera「見てくれたのは嬉しいけどちょっと恥ずかしいわね・・・」

Firefox「見てくれありがとな！
 正直、alert中で言った私の気持ちは本当だよ！」

Safari「・・・ありがと」ﾌｧｻ

では、

IE、Chrome、Opera、Firefox、Safari、Node.js「皆さんありがとうございました！」

終

IE、Chrome、Opera、Firefox、Safari「って、なんでNode.jsくんが！？
 改めまして、ありがとうございました！」

本当の本当に終わり