アルゴリズム・データ構造 I 第4回

JavaScript入門２ー関数と配列を使ってみよう

名城大学理工学部情報工学科
山本修身

関数の作り方と利用方法

- 大域環境に関数を定義する場合,

```
function foo(x, y, z){
    var s = x + y + z
    return s
}
```

のように書く．これは,

```
var foo = function(x, y, z){
     var s = x + y + z
     return s
}
```

と同じことを意味する．**x, y, z**は**引数**であり，**s**は**返り値**と呼ばれる．

- 関数を実際に実行する場合には,

```
foo(2, 3, 4)
```

のように値を後ろにつけて呼び出す．実は，**foo(2, 3, 4, 5, 6)**として呼び出しても間違いではない．後ろの**5, 6**は読み込まれないだけ．

配列と関数の練習１：ヒストグラムを作る (1)

- 乱数を発生させて，その乱数がどのように分布しているかを調べてみる．
  - まず，Math.random()を用いて，分布を調べる．

```
function rand_test(n, m){
   var a = []
   for (var i = 0; i < m; i++) a[i] = 0
   for (var i = 0; i < n; i++){
     x = Math.random()
     a[Math.floor(x * m)] += 1
   }
   for (var i = 0; i < m; i++)
    puts(i + " " + a[i])
}

rand_test(10000, 100)
```

実数が与えられたとき，それを超えない最大の整数を返す

配列と関数の練習１：ヒストグラムを作る (2)

このプログラムを実行すると，以下のようになる．

```
0 93
1 94
2 106
3 113
4 83
5 85
6 87
7 111
8 102
.....
93 105
94 111
95 105
96 95
97 115
98 101
99 115
```

結果は良さそうだが，結果の
全体を見渡すのが難しい！
スクロールしないと見えない

見やすく表示することを目指す

配列と関数の練習１：ヒストグラムを作る (3)

プログラムを改変する場合には，まず，必要道具を作ってそれを使って改造する．　まず，整数を右詰めで表示するための関数を書く．その前に，空白がいくつか続く文字列を返す関数を作る．<u>関数を作ったらそれの単体テストをすぐに行う．</u>

```
function spaces(n){
   var s = ''
   for (var i = 0; i < n; i++) s += ' '
   return s
}
```

spaces(n) の単体テスト

```
function test_spaces(){
   for (var i = 2; i < 10; i++)
     puts("[" + spaces(i) + "]")
}

test_spaces()
```

```
[  ]
[   ]
[    ]
[     ]
[      ]
[       ]
[        ]
[         ]
```


配列と関数の練習１：ヒストグラムを作る (4)

改めて整数を右詰めで表示するための関数を書く．

spaces(n)の定義がこの上に必要

```
function i2s(i, n){
   var m = 10
   for (var j = 0; j < 5; j++){
     if (i >= m) n -= 1
     m = m * 10
   }
   return spaces(n) + i
}
```

i2s(i, n) の単体テスト

```
for (var i = 0; i <10; i++){
   s = ""
   for (var j = 0; j < 10; j++)
      s+= i2s(i * j, 3)
   puts(s)
}
```

九九の表

```
  0  0  0  0  0  0  0  0  0  0
  0  1  2  3  4  5  6  7  8  9
  0  2  4  6  8 10 12 14 16 18
  0  3  6  9 12 15 18 21 24 27
  0  4  8 12 16 20 24 28 32 36
  0  5 10 15 20 25 30 35 40 45
  0  6 12 18 24 30 36 42 48 54
  0  7 14 21 28 35 42 49 56 63
  0  8 16 24 32 40 48 56 64 72
  0  9 18 27 36 45 54 63 72 81
```

配列と関数の練習１：ヒストグラムを作る (5)

- ヒストグラムの100個の要素を10x10の並びで表示する.

spaces(n), i2s(n) の定義がこの上に必要

```
function rand_test(n, m){
   var a = []
   for (var i = 0; i < m; i++) a[i] = 0
   for (var i = 0; i < n; i++){
     x = Math.random()
     a[Math.floor(x * m)] += 1
   }
   for (var i = 0; i < m / 10; i++){
     s = ""
     for (var j = 0; j < 10; j++)
        s += i2s(a[i * 10 + j], 4)
     puts(s)
   }
}

rand_test(10000, 100)
```

87	102	112	113	108	88	93	91	85	98
104	94	87	103	93	96	96	101	100	83
103	107	114	103	100	113	95	99	100	97
109	88	108	85	93	102	100	96	106	100
101	95	94	101	89	103	97	92	121	105
85	104	88	106	107	97	95	112	107	117
99	112	103	92	100	89	106	95	105	91
103	96	96	90	105	96	96	92	109	99
102	105	94	107	96	82	107	119	89	98
99	109	85	109	114	110	117	100	99	117

配列の初期化についての注意

JavaScriptの配列はCの配列とは違います.

```
int a[20] = {0};
```

これはC言語 ✕

のような書き方はできません. 空の配列を作ってそこにデータを入れるのか初期化する最も簡単な方法です. したがって,

```
var a = []
for (var i = 0; i < 20; i++) a[i] = 0
```

○

のように書きます.

配列の長さについて

- 配列の長さは.lengthを付けることで調べることができる. すなわち,

```
a = [3, 4, 5, 6]
puts(a.length)
```

を実行すれば, 結果は4となります. また, 効率の良い方法ではありませんが,

```
var i = 0
while (a[i] != undefined) i += 1
puts(i)
```

で調べることもできます.

再帰的な関数定義とは

再帰的な関数：ある関数の定義の中でその関数を参照している関数のこと.

```
function foo(n){
  .....
  .....
  foo(n - 1)
  .....
}
```

のような形をしている.

$$s_0 = 0$$
$$s_n = s_{n-1} + n$$

1 + 2 + ... + n を計算するのであれば,

```
function sum(n){
     if (n == 0) return 0
     else return sum(n - 1) + n
}
```


階乗と組合せ数の計算 (1)

階乗を用いて組み合せの計算を行う. 表現できる桁数に制限があるので, 入力がある程度大きくなると正しく計算できなくなる

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

function comb(n, m){
   return fact(n) / fact(m) / fact(n - m)
}

for (n = 0; n < 8; n++){
  var s = ""
  for (m = 0; m <= n; m++)
     s = s + " " + comb(n, m)
  puts(s)
}
```

$$_nC_m = \frac{n!}{m!(n-m)!}$$

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
```


階乗と組合せ数の計算 (2)

- 階乗を用いないで組み合せを計算する関数comb(n, m)をそのまま再帰的に計算することもできる. $_nC_0 = {}_nC_n = 1$

$$_nC_m = {}_{n-1}C_m + {}_{n-1}C_{m-1}$$

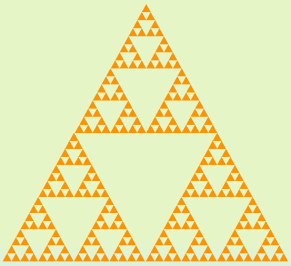
```
function comb(n, m){
   if (n == m || m == 0) return 1
   else return comb(n - 1, m - 1) + comb(n - 1, m)
}

for (n = 0; n < 8; n++){
  var s = ""
  for (m = 0; m <= n; m++)
     s = s + " " + comb(n, m)
  puts(s)
}
```

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
```

◇◇ 再帰構造の例 - シェルピンスキーガスケット - (1) ◇◇

- シェルピンスキーガスケットはシェルピンスキー（1882-1969）によって考案された以下のような図形である．

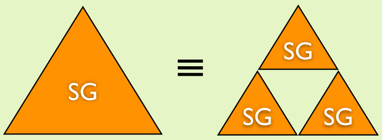


Waclaw Sierpinski

ドローツールで簡単に作れる

◇◇ 再帰構造の例 - シェルピンスキーガスケット - (2) ◇◇

- シェルピンスキーガスケット (SG) は以下のように定義される



SGの作り方（再帰的な説明）：

1. SGを１つ用意する
2. このSGのコピーを2つ用意する
3. それぞれを半分に縮小する
4. 3つの縮小されたSGを三角状に並べる

そもそもSGがないからSGを作るわけで最初からSGがあればSGは作らない…

鶏が先か卵が先か？

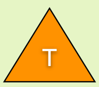
◇◇ 再帰構造の例 - シェルピンスキーガスケット - (3) ◇◇

もともとシェルピンスキーガスケットを持っていないので，シェルピンスキーガスケットは作れない．

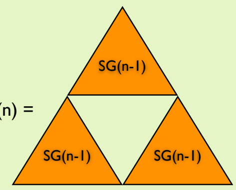
ところが…

シェルピンスキーガスケットの近似を考えれば作ることができる！

SG(0) =



SG(n) =



SG(3)を作るにはSG(2)があればよい．
SG(2)を作るにはSG(1)があればよい．
SG(1)を作るにはSG(0)があればよい．
SG(0)は普通の三角形なので作れる

n→∞のときSG(n)→SGとなる．SGの近似を求めるアルゴリズムとなる．

◇◇ 再帰構造の例 - シェルピンスキーガスケット - (4) ◇◇

シェルピンスキーガスケットを実際に描いてみる．

```
function middle_point(p1, p2){
   x1 = p1[0]; y1 = p1[1]
   x2 = p2[0]; y2 = p2[1]
   return [(x1 + x2) / 2, (y1 + y2) / 2]
}

function SG(p1, p2, p3, level){
   if (level == 0)
      draw_triangle(p1, p2, p3)
   else {
      var m1 = middle_point(p1, p2)
      var m2 = middle_point(p2, p3)
      var m3 = middle_point(p3, p1)
      SG(p1, m1, m3, level - 1)
      SG(m1, p2, m2, level - 1)
      SG(m3, m2, p3, level - 1)
   }
}

SG([0.5, 1], [0, 0], [1, 0], 7)
```

draw_triangle(p1, p2, p3)によって黒い三角形を描画することができる．

プログラミング環境は「プログラミング環境（描画環境付き）」を用いる

◇◇ 再帰構造の例 - シェルピンスキーガスケット - (5) ◇◇

- プログラムを実行すると以下のように表示される．