Scratch

スクラッチによるはじめての
アルゴリズム入門

河西 朝雄 著

KASAI.SOFTWARELAB

定価1,500円＋税

Scratch によるはじめての

アルゴリズム入門

河西　朝雄著

カサイ．ソフトウエアラボ

はじめに

プログラムを使って問題を解くための論理または手順をアルゴリズム（algorithms : 算法）といいます。プログラム処理では多量のデータを扱うことが多く、この場合、取り扱うデータをどのようなデータ構造（data structure）にするかで、問題解決のアルゴリズムが異なってきます。データ構造とアルゴリズムは密接な関係にあり、良いデータ構造を選ぶことが良いプログラムを作ることにつながります。

コンピュータ言語が持つデータ型には数値型、文字型などの基本データ型とデータをまとめて管理するリストがあります。しかし、このようなコンピュータ言語が持つデータ型だけでは、大量のデータや複雑な構造のデータを効率よく操作することはできません。そこでデータ群を都合よく組織化するための抽象的なデータ型をデータ構造と呼びます。代表的データ構造として、表（table：テーブル)、棚（stack：スタック)、待ち行列（queue：キュー)、リスト（list)、木（tree：ツリー)、グラフ（graph）があります。

プログラムの世界に特有なアルゴリズムとして再帰という考え方があります。再帰とデータ構造の木やグラフを組み合わせることで効率的なアルゴリズムができます。

ある問題を解くためのアルゴリズムは複数存在しますが、人間向きのアルゴリズムが必ずしもコンピュータ向きのアルゴリズムにならないこともあります。本書では先達が研究した伝統的なアルゴリズムを中心に以下の１２のカテゴリで解説します。

１章　画面出力

通常のプログラミング環境と異なり Scratch には print のような画面出力を行う命令がありません。そこで output という名前のリストを出力画面の代用にします。

２章　数値処理

コンピュータで扱うデータをおおざっぱに分けると、数値データと文字列データです。この章では数値データを処理する各種アルゴリズムを紹介します。

３章　文字列処理

２章では数値データを処理する各種アルゴリズムを紹介しました。この章では文字列データを処理する各種アルゴリズムを紹介します。

４章　乱数の利用

規則性がなくまったくでたらめに並ぶ数を乱数といいます。プログラムの世界では乱数を利用することで様々な処理が行えます。

５章　リストデータの処理

リストに格納されているデータから標準偏差や度数分布などの各種統計を取ったり、ソー

トやサーチといったリストデータの処理を説明します。

６章　再帰
再帰という考え方は人間の一般的な感覚からは縁遠いものですがプログラムの世界では重要な考え方です。再帰を用いると、複雑なアルゴリズムを明解に記述することができます。

７章　データ構造
データ群を都合よく組織化するための抽象的なデータ型をデータ構造と呼びます。代表的データ構造として表、スタック、キュー、リスト、木、グラフがあります。

８章　２次元グラフィックス
図形を方程式で表し、コンピュータにより解析的に解けば、平行移動、回転、拡大・縮小などの座標変換が簡単に行えます。

９章　３次元グラフィックス
３次元空間にある物体を紙やデイスプレイという２次元平面に作画するには、それなりの方法が必要になります。

１０章　リカーシブ・グラフィックス
リカーシブ・グラフィックスはグラフィックスの世界を解析的に表現せずに、再帰的に表現しようとするものです。

１１章　パズル・ゲーム
子ども達がプログラミングで一番作りたいものがゲームです。ゲーム機のゲームのようなものは簡単にはできませんので、この章ではパズル的な簡単にできるゲームを紹介します。

１２章　アラカルト
今までの各章で説明したアルゴリズムのカテゴリに属さない雑多なアルゴリズムを紹介します。

Scratch を用いて高度なプログラミングを勉強したい人にとって、本書で紹介したアルゴリズムやデータ構造を学ぶことが少しでもお役に立てば幸いです。

2016 年 10 月
河西　朝雄

目次

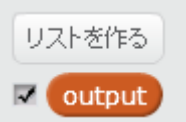
１章　画面出力

通常のプログラミング環境と異なりScratchにはprintのような画面出力を行う命令がありません。
そこでoutputという名前のリストを出力画面の代用にします。
そこに結果を表示するためのinit、println、print、newlineというユーザーブロックを作ります。

・initはリストoutputの内容をクリアします。
・println(arg)は引数argの内容を画面に表示し、改行（次のリスト要素に進める）します。
・print(arg,dig)は引数argの内容を、digで指定した桁数で右揃えして画面に表示します。改行（次のリスト要素に進める）はしません。
・newlineは改行（次のリスト要素に進める）をします。

１－１　画面出力 println

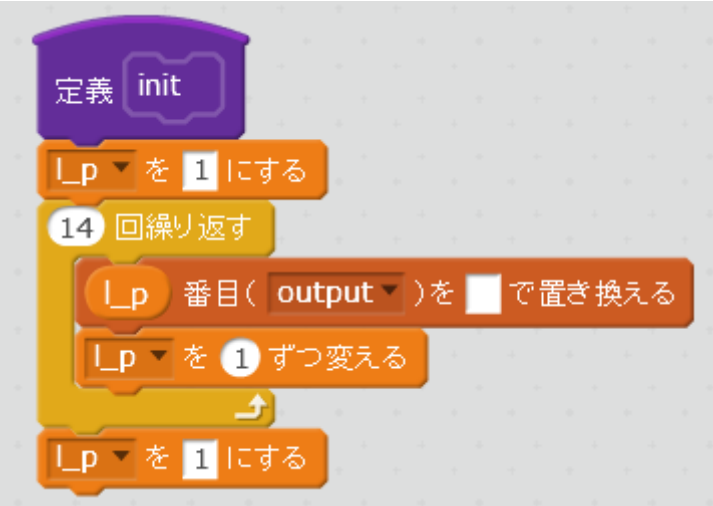
・output という名前のリストを作成し、表示状態にします。要素数を 14 とし画面一杯に幅を広げて表示します。



・output への表示位置は変数 l_p で管理します。

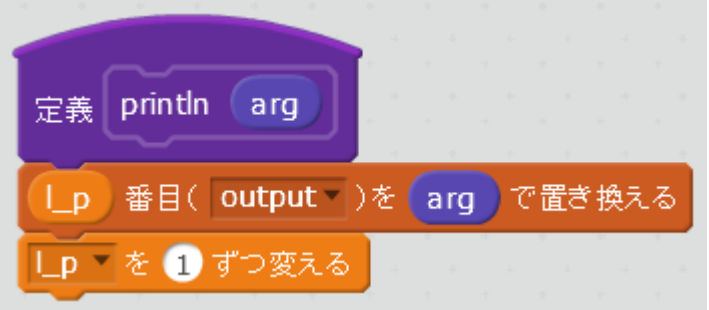
■init

init はリスト output の内容をクリアします。



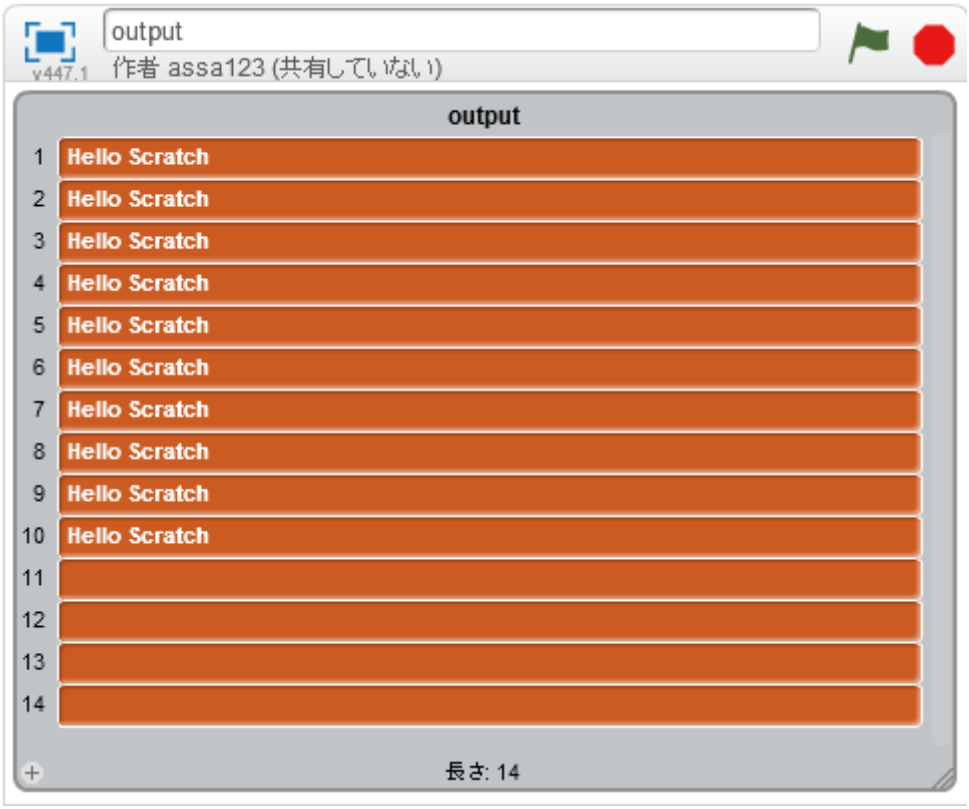
■println(arg)

println(arg)は引数 arg の内容を画面に表示し、改行（次のリスト要素に進める）します。



例題 1-1　println の例

「Hello Scratch」を 10 回表示します。

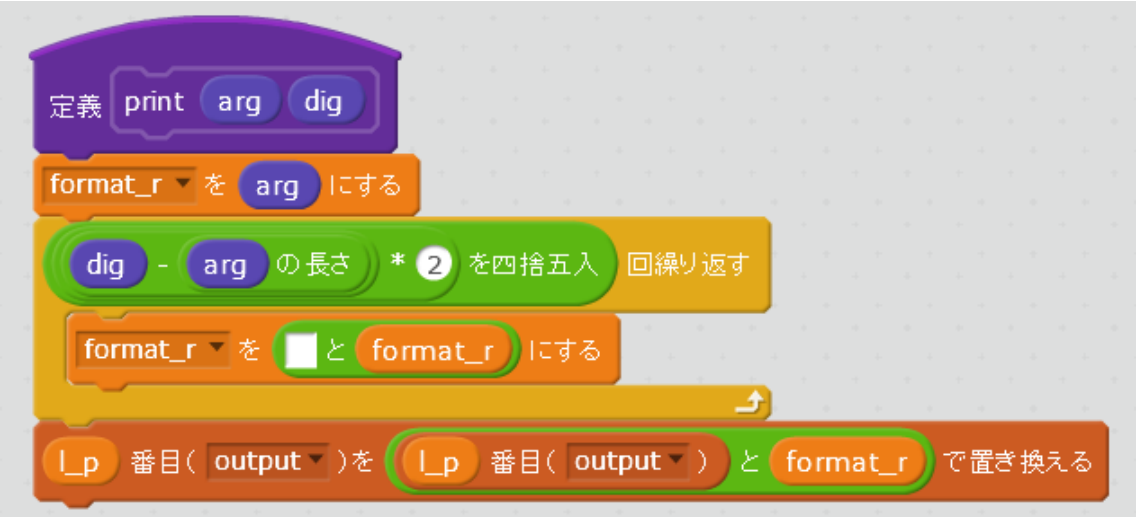


１－２　書式付き出力 print

■print(arg,dig)

print(arg,dig)は引数 arg の内容を、dig で指定した桁数で右揃えして画面に表示します。改行（次のリスト要素に進める）はしません。桁数を会わせるために左に詰める空白は、本来埋めるべき数の 2 倍で埋めています。これは空白が英小文字、数字などのフォントサイズの半分程度になっているためです。読者の使用環境でこの 2 倍という値が異なる場合は調整してください。

・変数 format_r に指定した桁数で右揃えした内容を格納します。

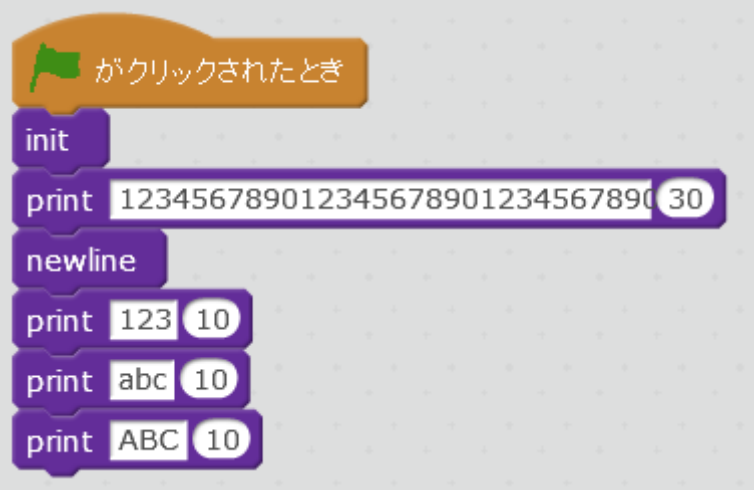


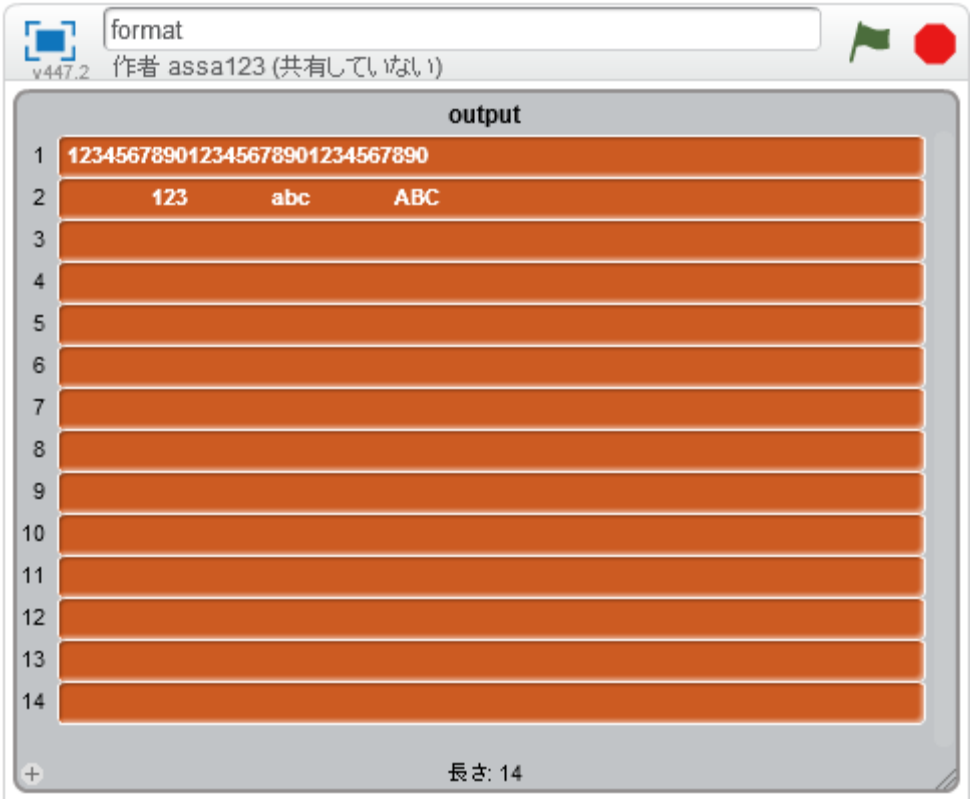
■newline

newline は改行（次のリスト要素に進める）をします。

例題 1-2-1　print の例

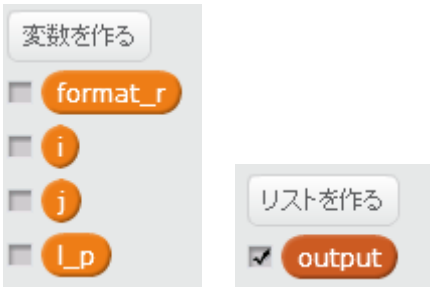
「123」、「abc」、「ABC」を 10 桁の右揃えで表示します。英大文字は英小文字、数字のフォントサイズより大きいので、表示位置が多少ずれます。

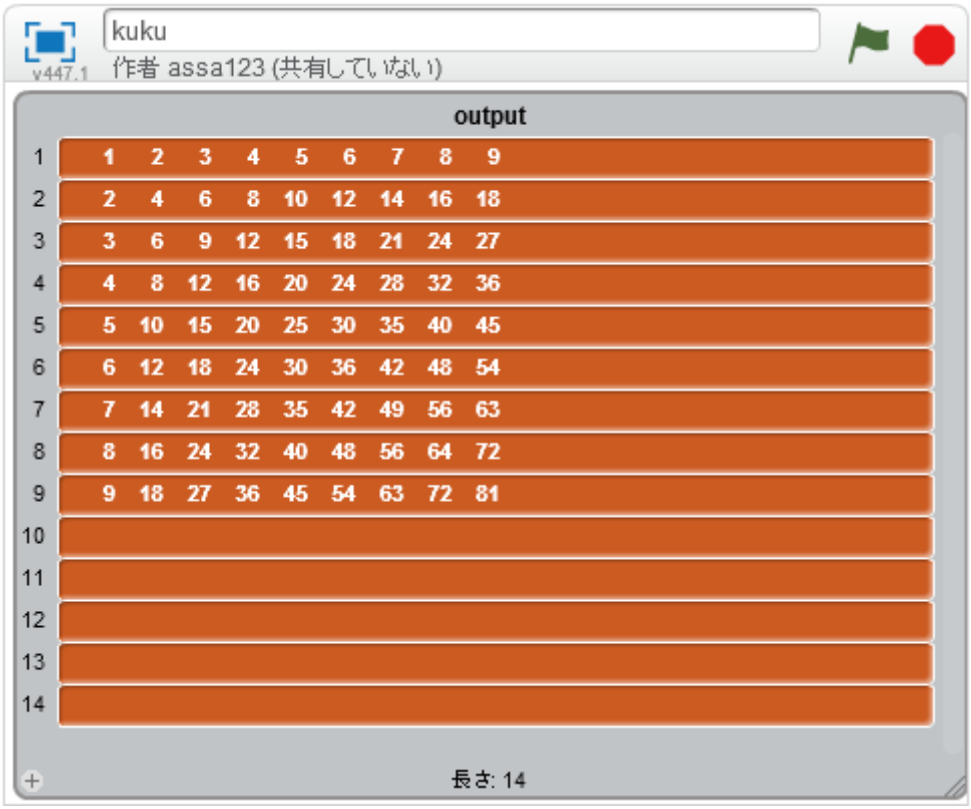




例題 1-2-2　九九の表

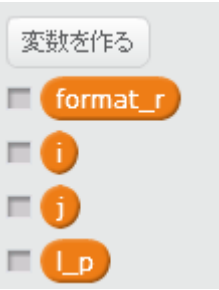
九九の値を４桁で表示します。

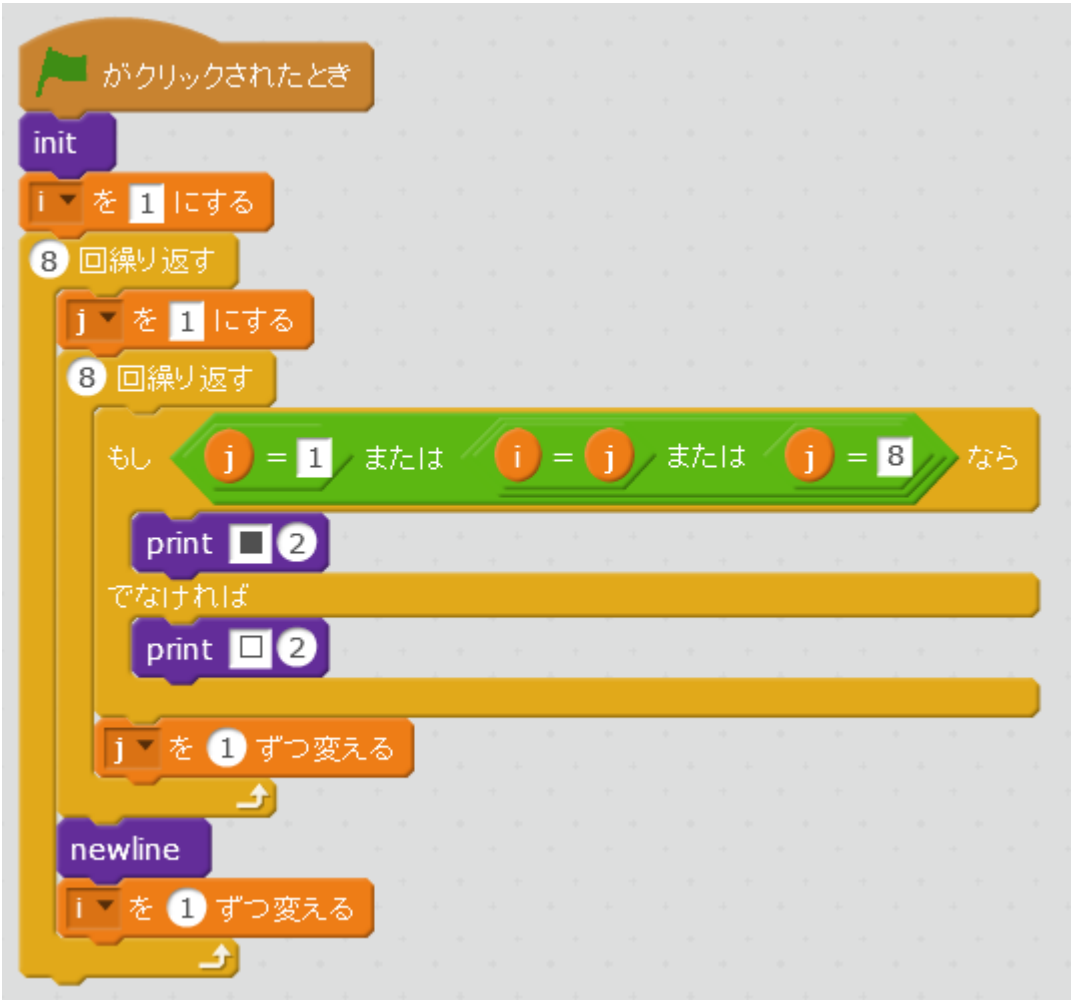


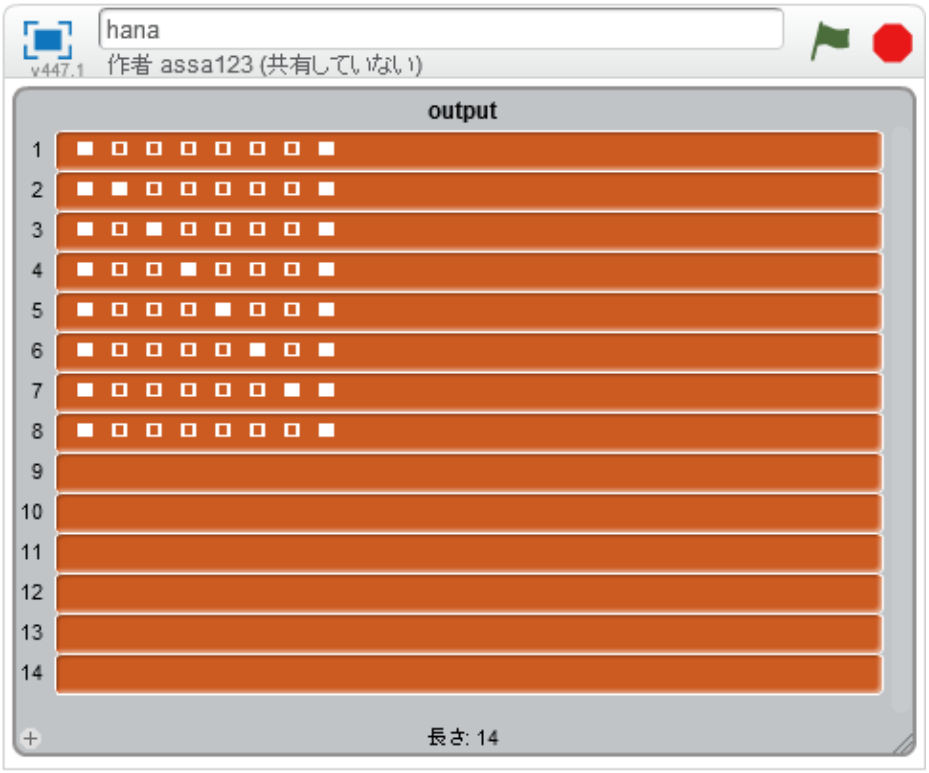

v447.1
kuku
作者 assa123 (共有していない)
output
1 1 2 3 4 5 6 7 8 9
2 2 4 6 8 10 12 14 16 18
3 3 6 9 12 15 18 21 24 27
4 4 8 12 16 20 24 28 32 36
5 5 10 15 20 25 30 35 40 45
6 6 12 18 24 30 36 42 48 54
7 7 14 21 28 35 42 49 56 63
8 8 16 24 32 40 48 56 64 72
9 9 18 27 36 45 54 63 72 81
10
11
12
13
14
長さ: 14

例題 1-2-3　花文字

N という花文字を表示します。■と□を 2 桁で表示します。






hana
v447.1
作者 assa123 (共有していない)
output
1
2
3
4
5
6
7
8
9
10
11
12
13
14
長さ: 14

２章　数値処理

コンピュータで扱うデータをおおざっぱに分けると、数値データと文字列データです。この章では数値データを処理する各種アルゴリズムを紹介します。

・進数変換
・ユークリッドの互除法
・エラトステネスのふるい
・漸化式
・数値積分
・非線形方程式の解法
・補間
・テイラー展開
・多桁計算
・長いπ
・連立方程式の解法

２－４　漸化式

階乗、べき乗、フィボナッチ数列、組み合わせは、n 項を n-1 項などの前の項を使って表現できるのが特徴です。このようなものを漸化式といいます。代表的な漸化式の例を以下に示します。

①階乗

$n!=n\cdot(n-1)!$

$0!=1$

②べき乗

$x^n=x\cdot x^{n-1}$

$x^0=1$

③フィボナッチ数列

$F_n=F_{n-1}+F_{n-2}$

$F_1=F_2=1$

④組み合わせ

${}_nC_r={}_{n-1}C_{r-1}+{}_{n-1}C_r$

${}_rC_0={}_rC_r=1$

漸化式は n 次の定義を n-1 次を使って定義しているので、再帰的に表現することもできます。 6 章の「6-2　漸化式の再帰版」参照。

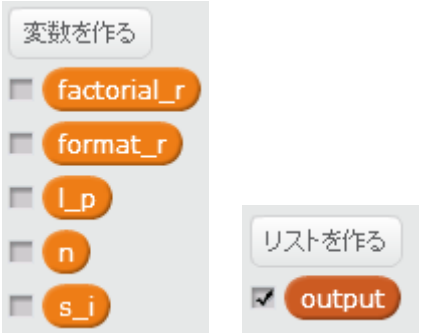
例題 2-4-1　階乗

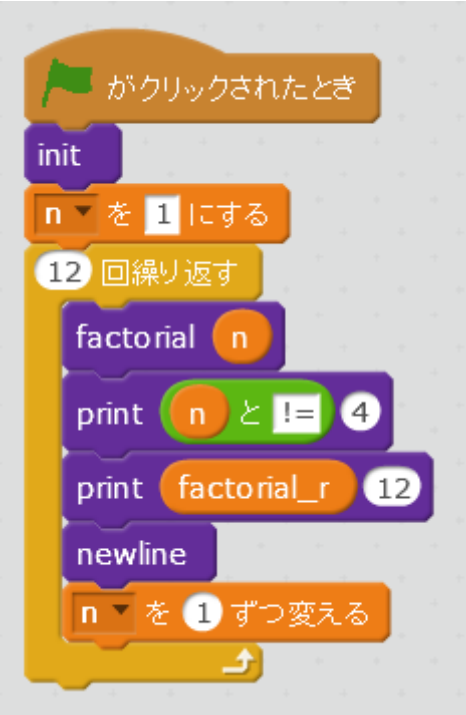
1!~12!を求めて表示します。n!を求めるユーザブロックを factorial(n)とします。結果は変数 factorial_r に得られます。

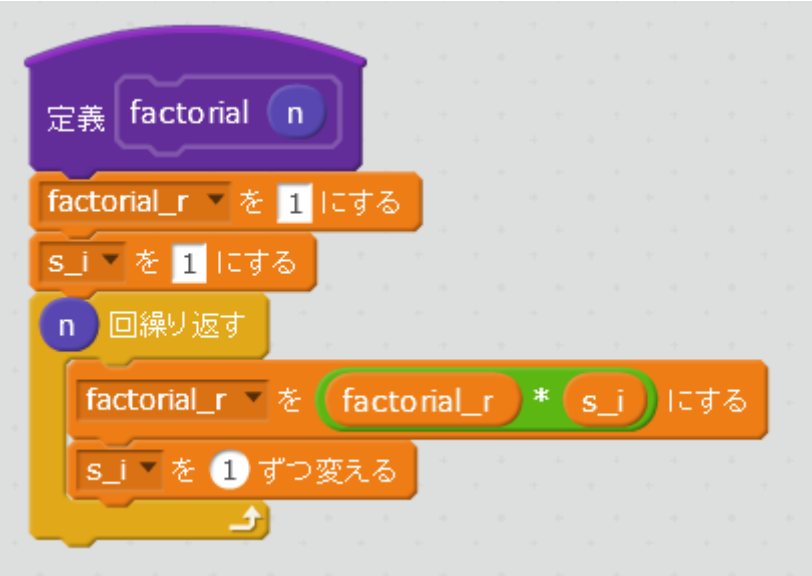
「注」本書では、ユーザブロック内の変数とメインでの変数を区別するために以下の規則を設けます。

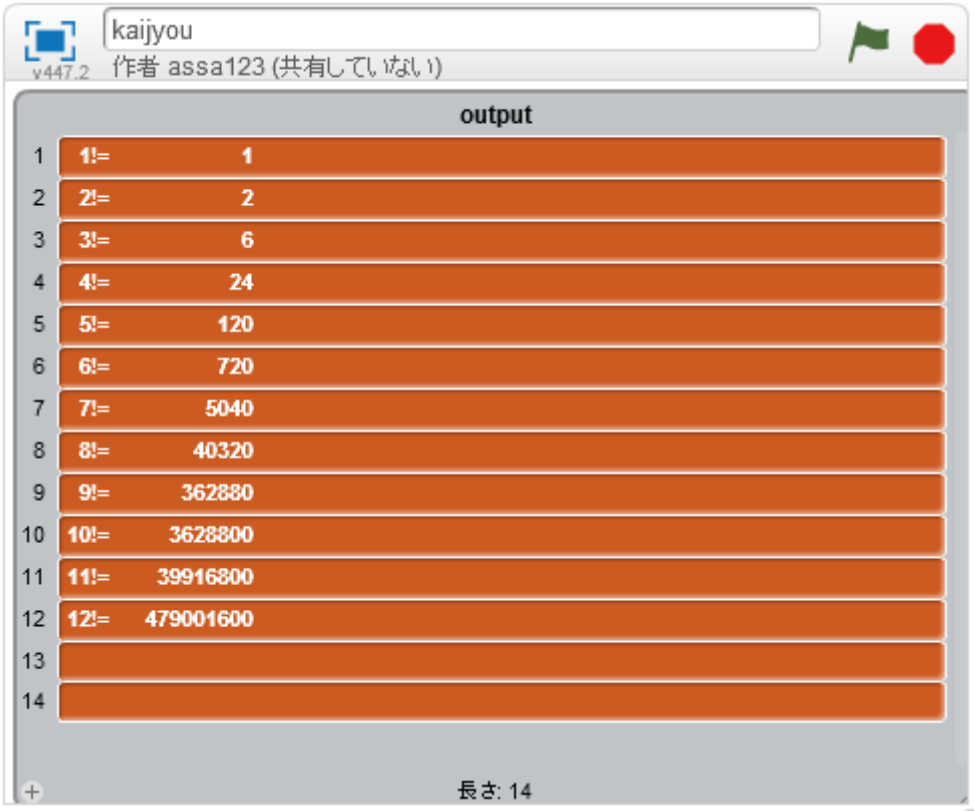
・ユーザブロックの結果を返す変数には「_r」を末尾に付けます。

・ユーザブロック内で使うループ変数は s_i のように「s_」を先頭に付けます。



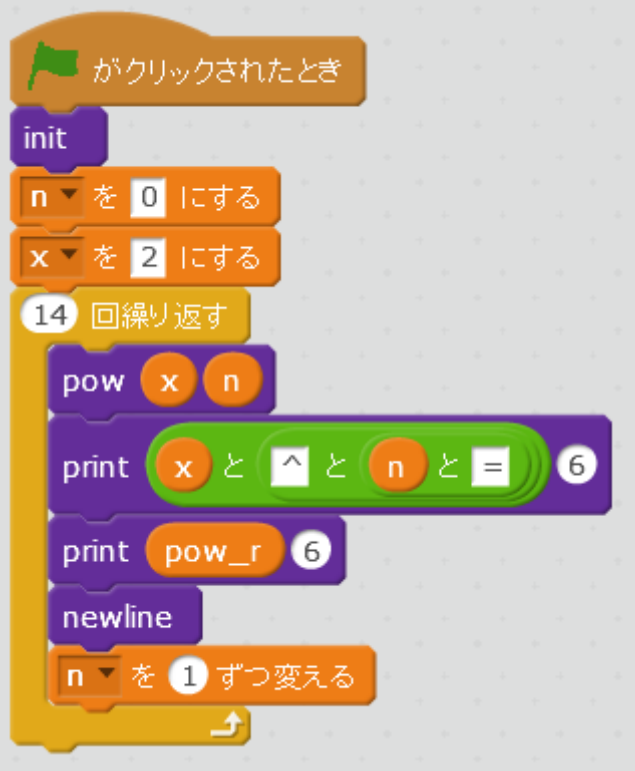

がクリックされたとき
init
n を 1 にする
12 回繰り返す
factorial n
print n と != 4
print factorial_r 12
newline
n を 1 ずつ変える

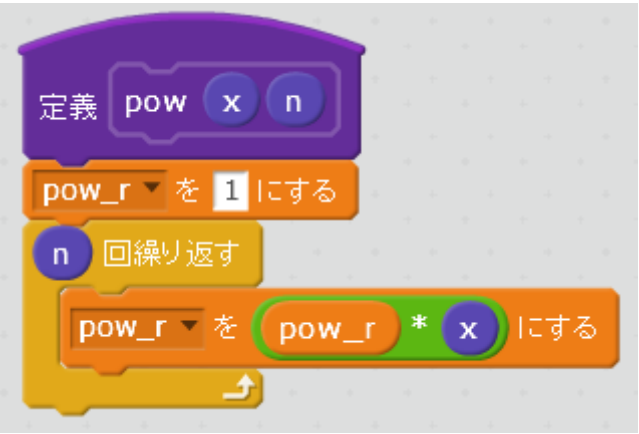

定義 factorial n
factorial_r を 1 にする
s_i を 1 にする
n 回繰り返す
factorial_r を factorial_r * s_i にする
s_i を 1 ずつ変える

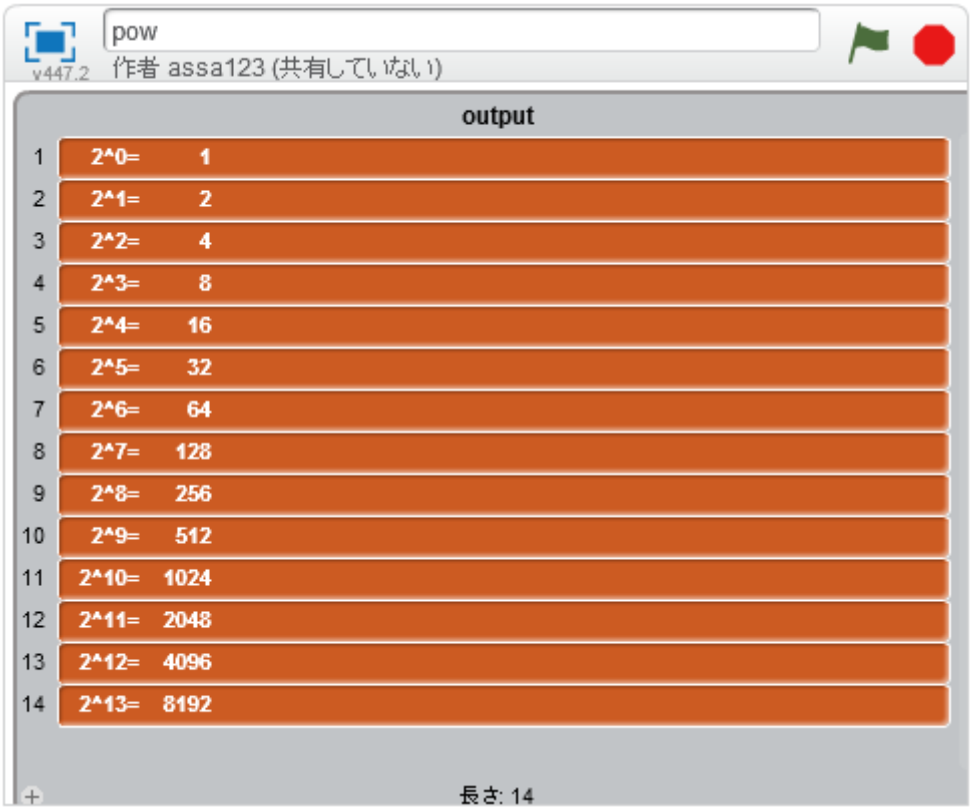


例題 2-4-2　べき乗

2^0~2^{12} を求めて表示します。x^n を求めるユーザブロックを pow(x,n)とします。結果は変数 pow_r に得られます。


がクリックされたとき
init
n を 0 にする
x を 2 にする
14 回繰り返す
pow x n
print x と ^ と n と = 6
print pow_r 6
newline
n を 1 ずつ変える


定義 pow x n
pow_r を 1 にする
n 回繰り返す
pow_r を pow_r * x にする



例題 2-4-3　フィボナッチ数列

フィボナッチ数列は以下のように定義された数列です。

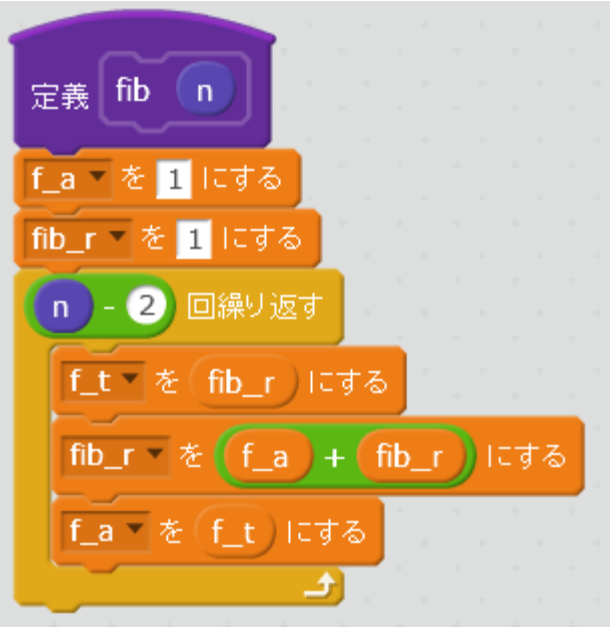
・n 項のフィボナッチ数は n−1 項と n−2 項を足したものである。

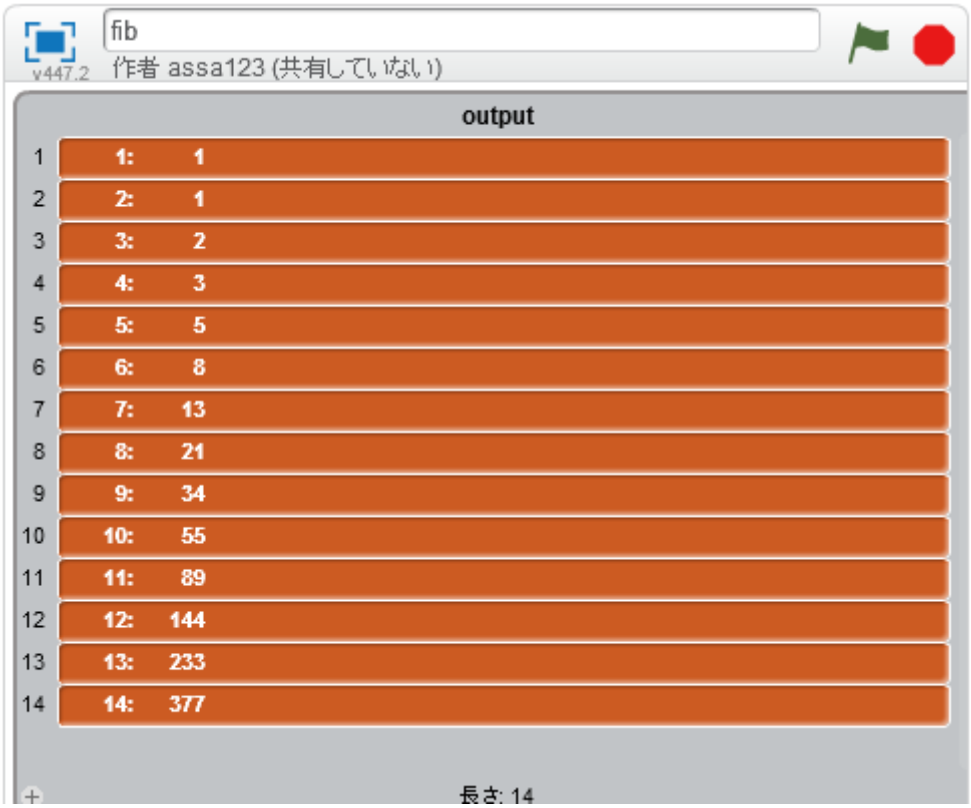
・第 1,2 項は「1」である。

1 1 2 3 5 8 13 21 34 55 89 ・・・

n のフィボナッチ数を求めるユーザブロックを fib(n)とします。結果は変数 fib_r に得られます。


がクリックされたとき
init
n を 1 にする
14 回繰り返す
fib n
print n と : 6
print fib_r 6
newline
n を 1 ずつ変える


定義 fib n
f_a を 1 にする
fib_r を 1 にする
n - 2 回繰り返す
f_t を fib_r にする
fib_r を f_a + fib_r にする
f_a を f_t にする

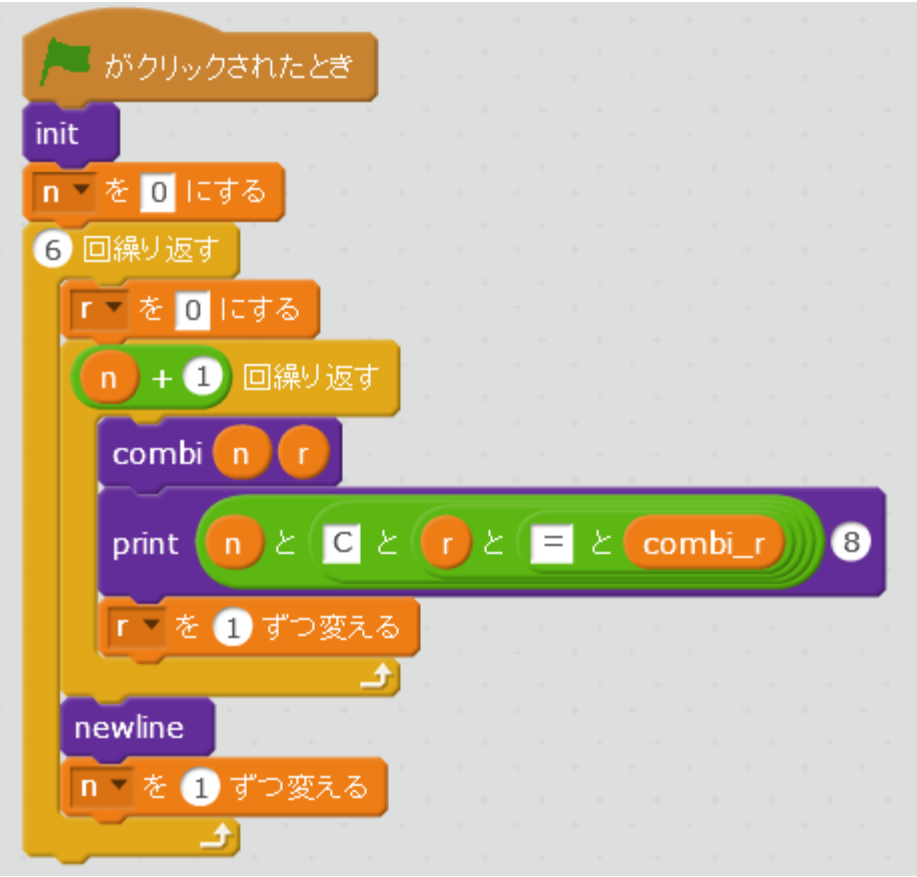


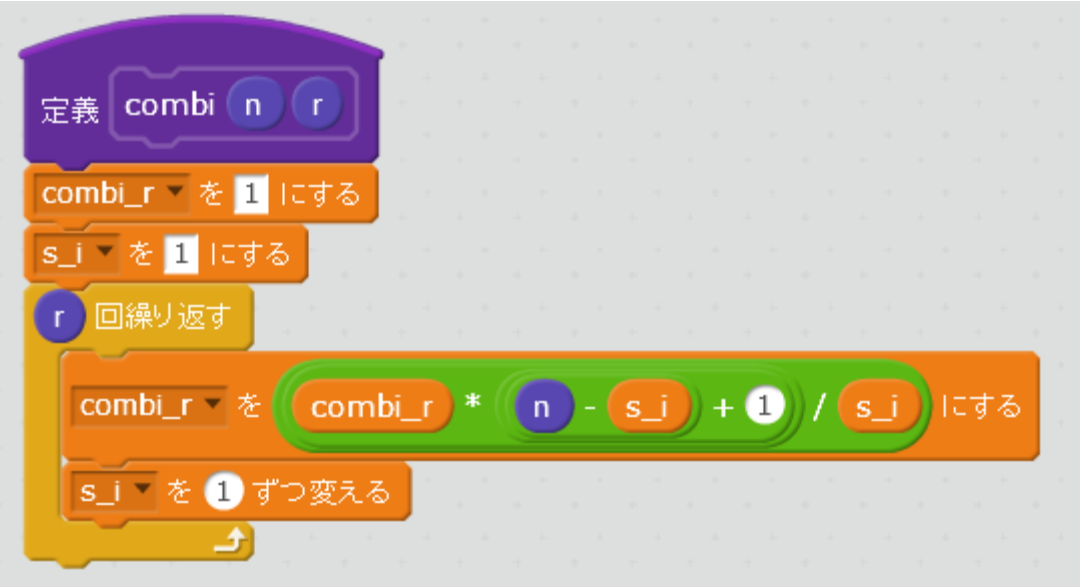
例題 2-4-4　組み合わせ

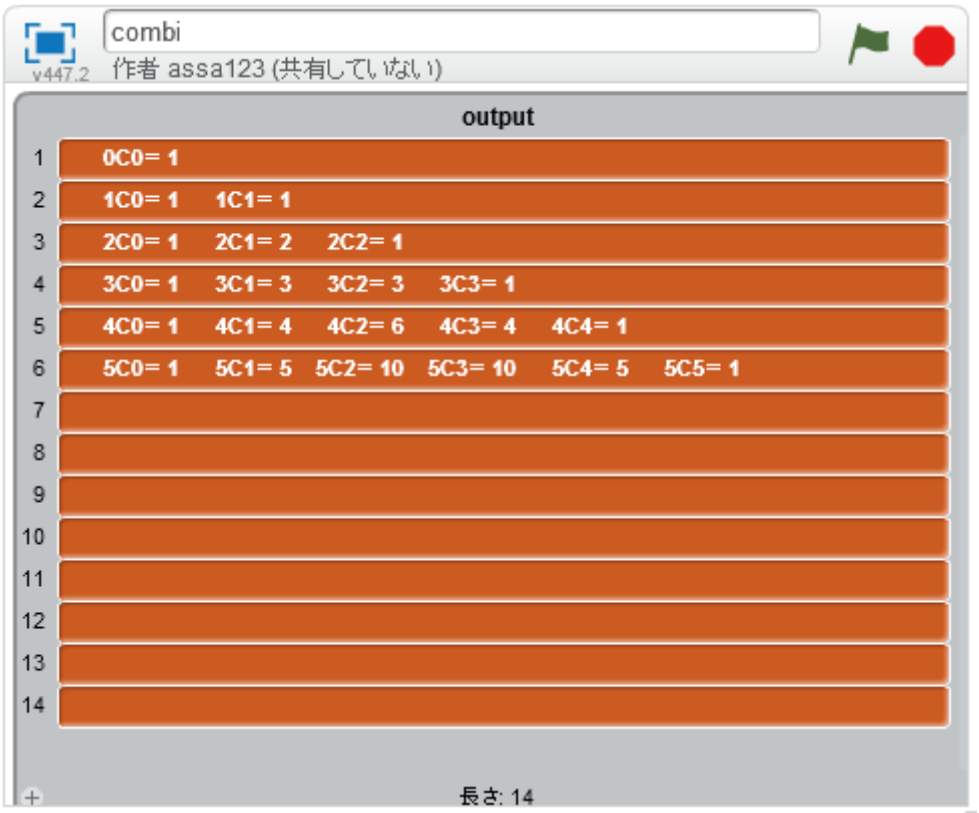
n 個の中から r 個を選ぶ組み合わせ（Combination）の数を ${}_nC_r$ と書き、次の式で定義されます。

$${}_nC_r = n!/r!(n-r)!$$

${}_nC_r$ を求めるユーザブロックを combi(n,r)とします。結果は変数 combi_r に得られます。


がクリックされたとき
init
n を 0 にする
6 回繰り返す
r を 0 にする
n + 1 回繰り返す
combi n r
print n と C と r と = と combi_r 8
r を 1 ずつ変える
newline
n を 1 ずつ変える

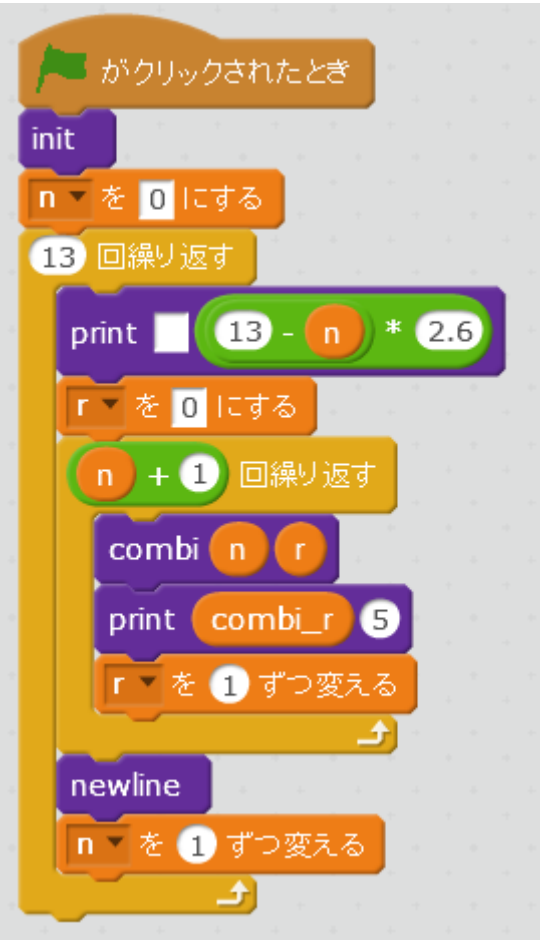

定義 combi n r
combi_r を 1 にする
s_i を 1 にする
r 回繰り返す
combi_r を combi_r * n - s_i + 1 / s_i にする
s_i を 1 ずつ変える

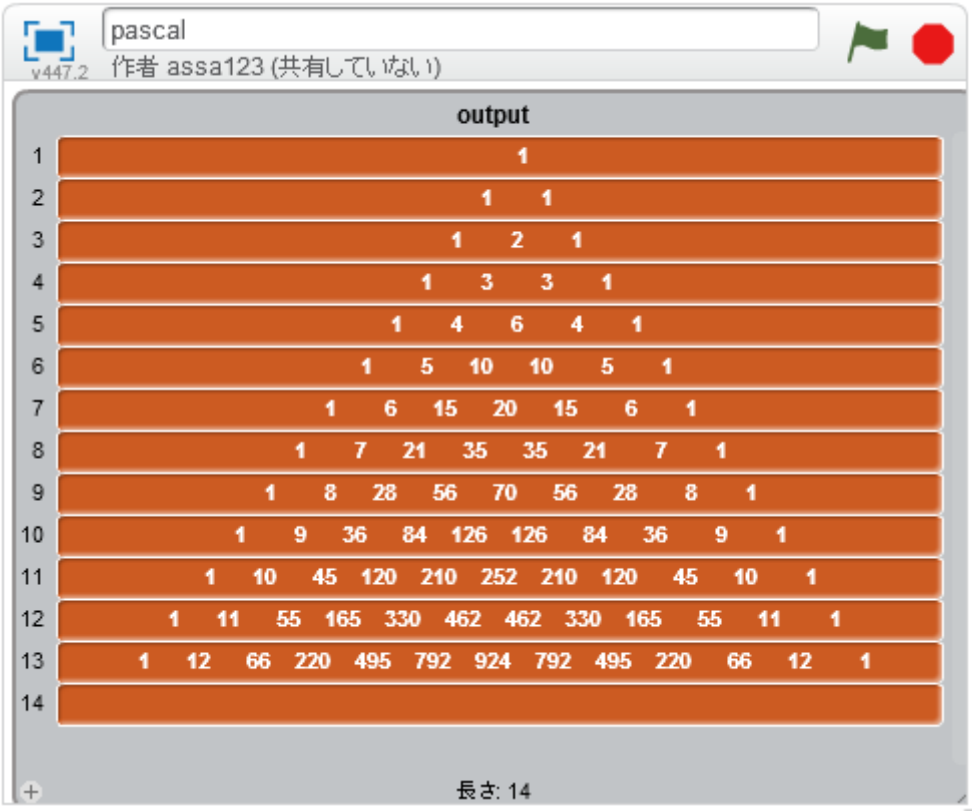

v447.2
combi
作者 assa123 (共有していない)
output
1 0C0= 1
2 1C0= 1 1C1= 1
3 2C0= 1 2C1= 2 2C2= 1
4 3C0= 1 3C1= 3 3C2= 3 3C3= 1
5 4C0= 1 4C1= 4 4C2= 6 4C3= 4 4C4= 1
6 5C0= 1 5C1= 5 5C2= 10 5C3= 10 5C4= 5 5C5= 1
7
8
9
10
11
12
13
14
長さ: 14

例題 2-4-5　Pascal の 3 角形

${}_nC_r$ を次のように並べたものを Pascal の三角形と呼びます。

$${}_0C_0$$
$${}_1C_0 \quad {}_1C_1$$
$${}_2C_0 \quad {}_2C_1 \quad {}_2C_2$$
$${}_3C_0 \quad {}_3C_1 \quad {}_3C_2 \quad {}_3C_3$$
$${}_4C_0 \quad {}_4C_1 \quad {}_4C_2 \quad {}_4C_3 \quad {}_4C_4$$




pascal
v447.2
作者 assa123 (共有していない)
output
1 1
2 1 1
3 1 2 1
4 1 3 3 1
5 1 4 6 4 1
6 1 5 10 10 5 1
7 1 6 15 20 15 6 1
8 1 7 21 35 35 21 7 1
9 1 8 28 56 70 56 28 8 1
10 1 9 36 84 126 126 84 36 9 1
11 1 10 45 120 210 252 210 120 45 10 1
12 1 11 55 165 330 462 462 330 165 55 11 1
13 1 12 66 220 495 792 924 792 495 220 66 12 1
14
長さ: 14

２－１０　長いπ

e の値やπの値を 1000 桁まで正確に求めます。「２－９　多桁計算」で作った、ladd,lsub,ldiv を使います。繰り返し回数は N→L に変更します。

例題 2-10-1　e の 1000 桁計算

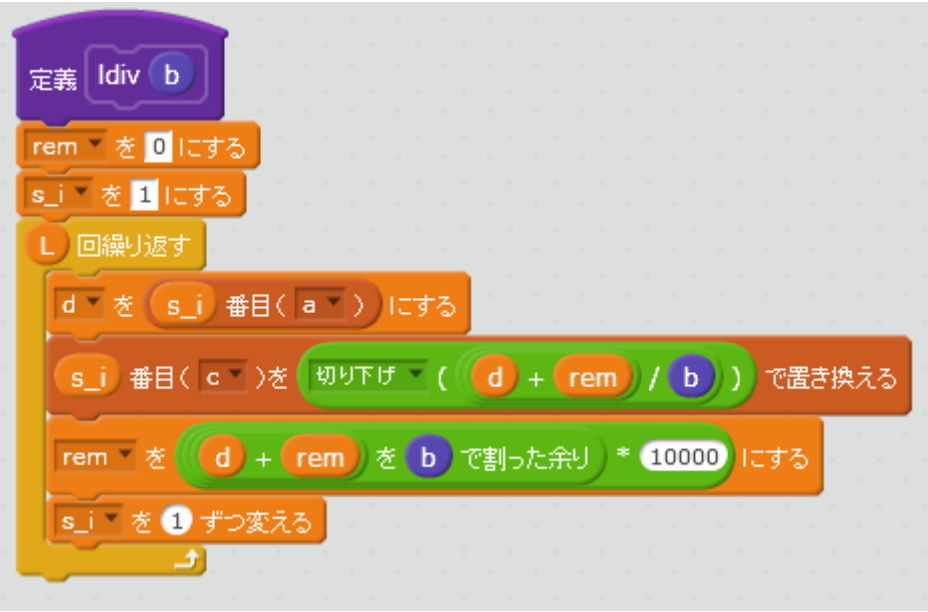
テイラー展開によると e(自然対数の底)の値は次式で求められます。

e=1+1/1!+1/2!+1/3!+・・・+1/n!+・・・

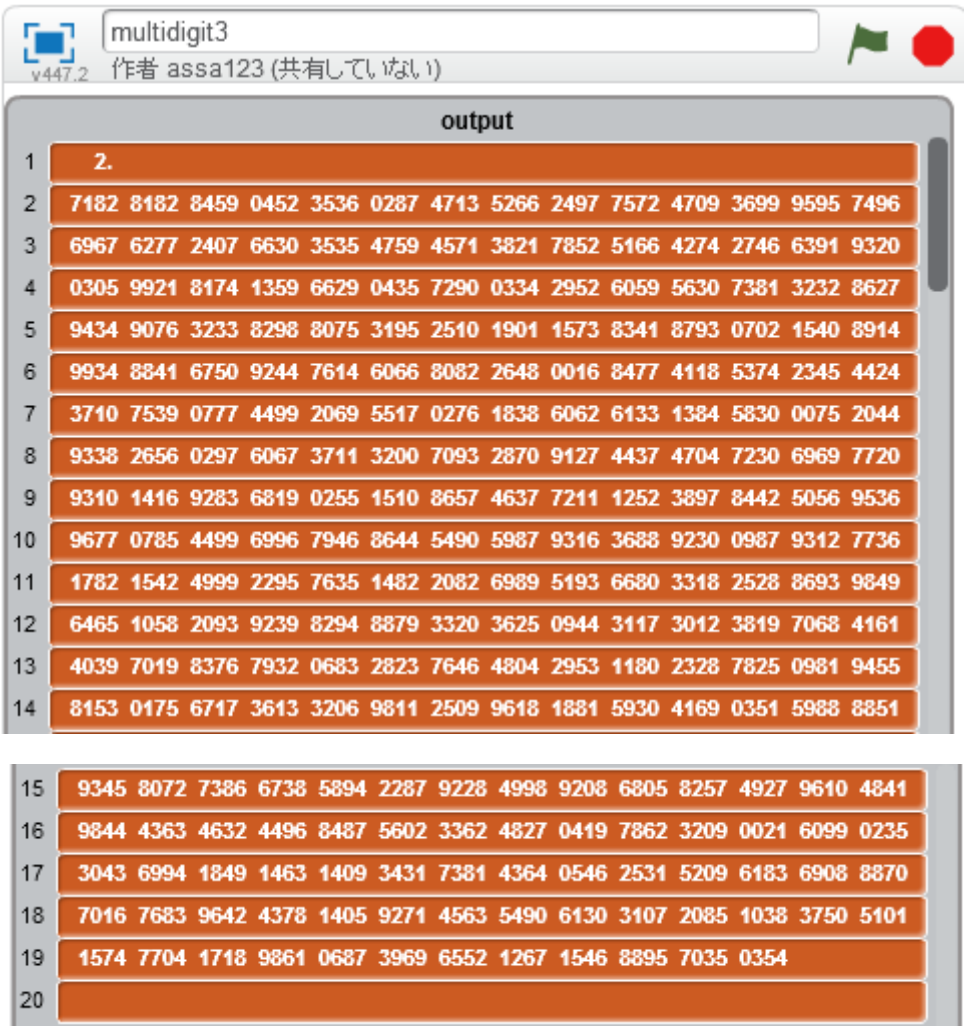

がクリックされたとき
KETA を 1000 にする
L を 切り下げ (KETA / 4 + 3) にする
N を 451 にする
s の長さ 回繰り返す
1 番目を s から削除する
1 番目を w から削除する
1 番目を a から削除する
1 番目を b から削除する
1 番目を c から削除する
k を 1 にする
L 回繰り返す
0 を k 番目に挿入する(s)
0 を k 番目に挿入する(w)
0 を k 番目に挿入する(a)
0 を k 番目に挿入する(b)
0 を k 番目に挿入する(c)
k を 1 ずつ変える
1 番目(s)を 1 で置き換える
1 番目(w)を 1 で置き換える


k を 1 にする
N 回繰り返す
i を 1 にする
s の長さ 回繰り返す
i 番目(a)を i 番目(w) で置き換える
i を 1 ずつ変える
ldiv k
i を 1 にする
s の長さ 回繰り返す
i 番目(w)を i 番目(c) で置き換える
i を 1 ずつ変える
i を 1 にする
s の長さ 回繰り返す
i 番目(a)を i 番目(s) で置き換える
i 番目(b)を i 番目(w) で置き換える
i を 1 ずつ変える
ladd
i を 1 にする
s の長さ 回繰り返す
i 番目(s)を i 番目(c) で置き換える
i を 1 ずつ変える
k を 1 ずつ変える
init
disp


定義 ladd
cy を 0 にする
s_i を L にする
L 回繰り返す
s_i 番目(c)を s_i 番目(a) + s_i 番目(b) + cy で置き換える
もし s_i 番目(c) < 10000 なら
cy を 0 にする
でなければ
s_i 番目(c)を s_i 番目(c) - 10000 で置き換える
cy を 1 にする
s_i を -1 ずつ変える


定義 ldiv b
rem を 0 にする
s_i を 1 にする
L 回繰り返す
d を s_i 番目(a) にする
s_i 番目(c)を 切り下げ (d + rem / b) で置き換える
rem を d + rem を b で割った余り * 10000 にする
s_i を 1 ずつ変える

・zeropad は例題 2-9-1 と同じです。


multidigit3
v447.2
作者 assa123 (共有していない)
output
1 2.
2 7182 8182 8459 0452 3536 0287 4713 5266 2497 7572 4709 3699 9595 7496
3 6967 6277 2407 6630 3535 4759 4571 3821 7852 5166 4274 2746 6391 9320
4 0305 9921 8174 1359 6629 0435 7290 0334 2952 6059 5630 7381 3232 8627
5 9434 9076 3233 8298 8075 3195 2510 1901 1573 8341 8793 0702 1540 8914
6 9934 8841 6750 9244 7614 6066 8082 2648 0016 8477 4118 5374 2345 4424
7 3710 7539 0777 4499 2069 5517 0276 1838 6062 6133 1384 5830 0075 2044
8 9338 2656 0297 6067 3711 3200 7093 2870 9127 4437 4704 7230 6969 7720
9 9310 1416 9283 6819 0255 1510 8657 4637 7211 1252 3897 8442 5056 9536
10 9677 0785 4499 6996 7946 8644 5490 5987 9316 3688 9230 0987 9312 7736
11 1782 1542 4999 2295 7635 1482 2082 6989 5193 6680 3318 2528 8693 9849
12 6465 1058 2093 9239 8294 8879 3320 3625 0944 3117 3012 3819 7068 4161
13 4039 7019 8376 7932 0683 2823 7646 4804 2953 1180 2328 7825 0981 9455
14 8153 0175 6717 3613 3206 9811 2509 9618 1881 5930 4169 0351 5988 8851
15 9345 8072 7386 6738 5894 2287 9228 4998 9208 6805 8257 4927 9610 4841
16 9844 4363 4632 4496 8487 5602 3362 4827 0419 7862 3209 0021 6099 0235
17 3043 6994 1849 1463 1409 3431 7381 4364 0546 2531 5209 6183 6908 8870
18 7016 7683 9642 4378 1405 9271 4563 5490 6130 3107 2085 1038 3750 5101
19 1574 7704 1718 9861 0687 3969 6552 1267 1546 8895 7035 0354
20

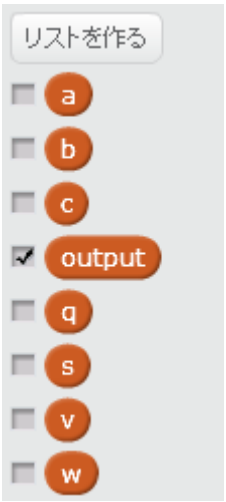
例題 2-10-2　π の 1000 桁計算

マチンの公式によるとπの多項式の第 n 項は以下の式で表せます。

$$(16/5^{2n-1}-4/239^{2n-1})\cdot 1/(2n-1)$$

リスト w,v,q,s に対して多桁計算を行うユーザブロックは以下です。

w/25 を ldiv1 で行います
v/239 を ldiv2 で行います
q/(2*k-1)を ldiv3 で行います
s+q を ladd1 で行います
w-v を lsub1 で行います
s-q を lsub2 で行います

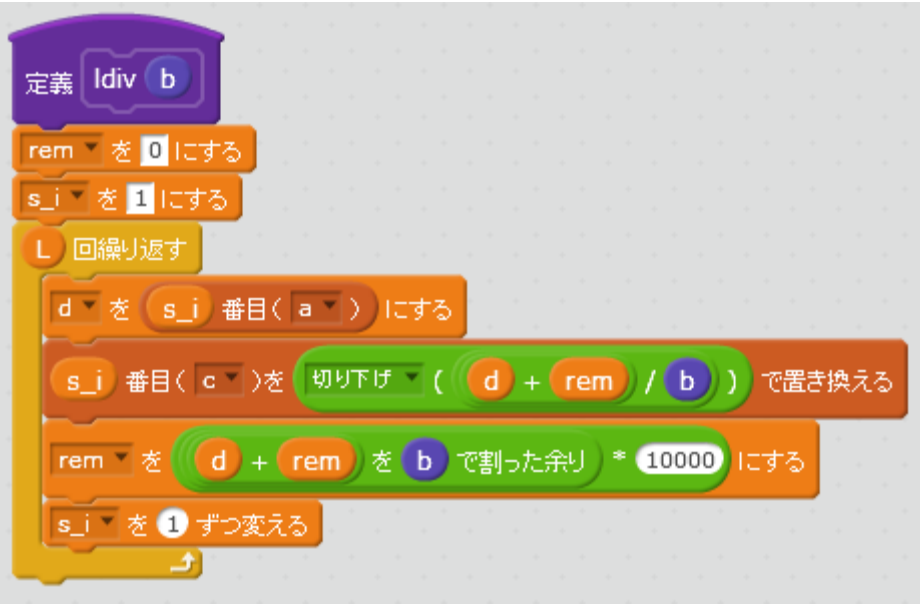



がクリックされたとき
KETA を 1000 にする
L を 切り下げ (KETA / 4 + 3) にする
N を 切り下げ (KETA / 1.39794 + 1) にする
s の長さ 回繰り返す
1 番目を s から削除する
1 番目を w から削除する
1 番目を v から削除する
1 番目を q から削除する
1 番目を a から削除する
1 番目を b から削除する
1 番目を c から削除する
k を 1 にする
L 回繰り返す
0 を k 番目に挿入する(s)
0 を k 番目に挿入する(w)
0 を k 番目に挿入する(v)
0 を k 番目に挿入する(q)
0 を k 番目に挿入する(a)
0 を k 番目に挿入する(b)
0 を k 番目に挿入する(c)
k を 1 ずつ変える
1 番目(w)を 16 * 5 で置き換える
1 番目(v)を 4 * 239 で置き換える


k を 1 にする
N 回繰り返す
ldiv1
ldiv2
ldiv2
lsub1
ldiv3
もし k を 2 で割った余り = 0 なら
lsub2
でなければ
ladd1
k を 1 ずつ変える
init
disp


定義 ladd
cy を 0 にする
s_i を L にする
L 回繰り返す
s_i 番目(c)を s_i 番目(a) + s_i 番目(b) + cy で置き換える
もし s_i 番目(c) < 10000 なら
cy を 0 にする
でなければ
s_i 番目(c)を s_i 番目(c) - 10000 で置き換える
cy を 1 にする
s_i を -1 ずつ変える


定義 lsub
brrw を 0 にする
s_i を L にする
L 回繰り返す
s_i 番目(c)を s_i 番目(a) - s_i 番目(b) - brrw で置き換える
もし s_i 番目(c) < 0 なら
s_i 番目(c)を s_i 番目(c) + 10000 で置き換える
brrw を 1 にする
でなければ
brrw を 0 にする
s_i を -1 ずつ変える


定義 ldiv b
rem を 0 にする
s_i を 1 にする
L 回繰り返す
d を s_i 番目(a) にする
s_i 番目(c)を 切り下げ (d + rem / b) で置き換える
rem を d + rem を b で割った余り * 10000 にする
s_i を 1 ずつ変える


定義 ldiv1
i を 1 にする
s の長さ 回繰り返す
i 番目(a)を i 番目(w) で置き換える
i を 1 ずつ変える
ldiv 25
i を 1 にする
s の長さ 回繰り返す
i 番目(w)を i 番目(c) で置き換える
i を 1 ずつ変える

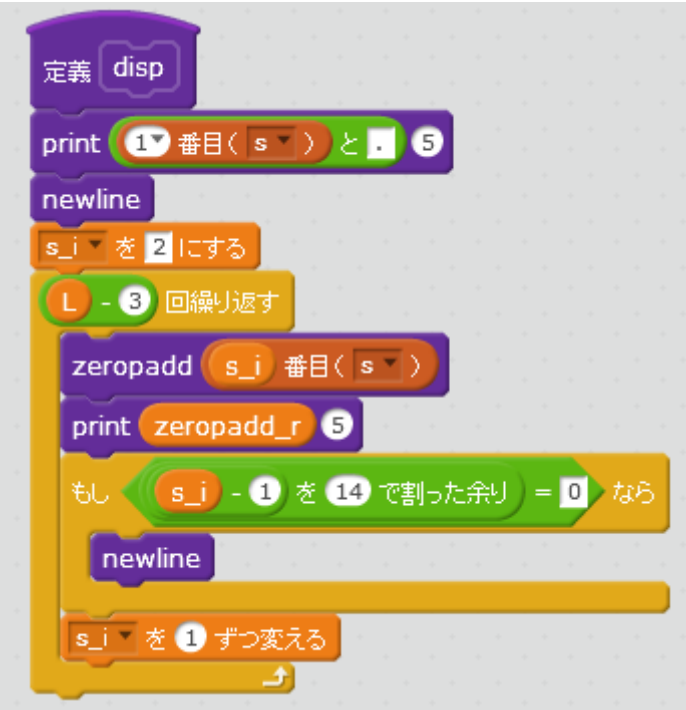

定義 ldiv2
i を 1 にする
s の長さ 回繰り返す
i 番目(a)を i 番目(v) で置き換える
i を 1 ずつ変える
ldiv 239
i を 1 にする
s の長さ 回繰り返す
i 番目(v)を i 番目(c) で置き換える
i を 1 ずつ変える

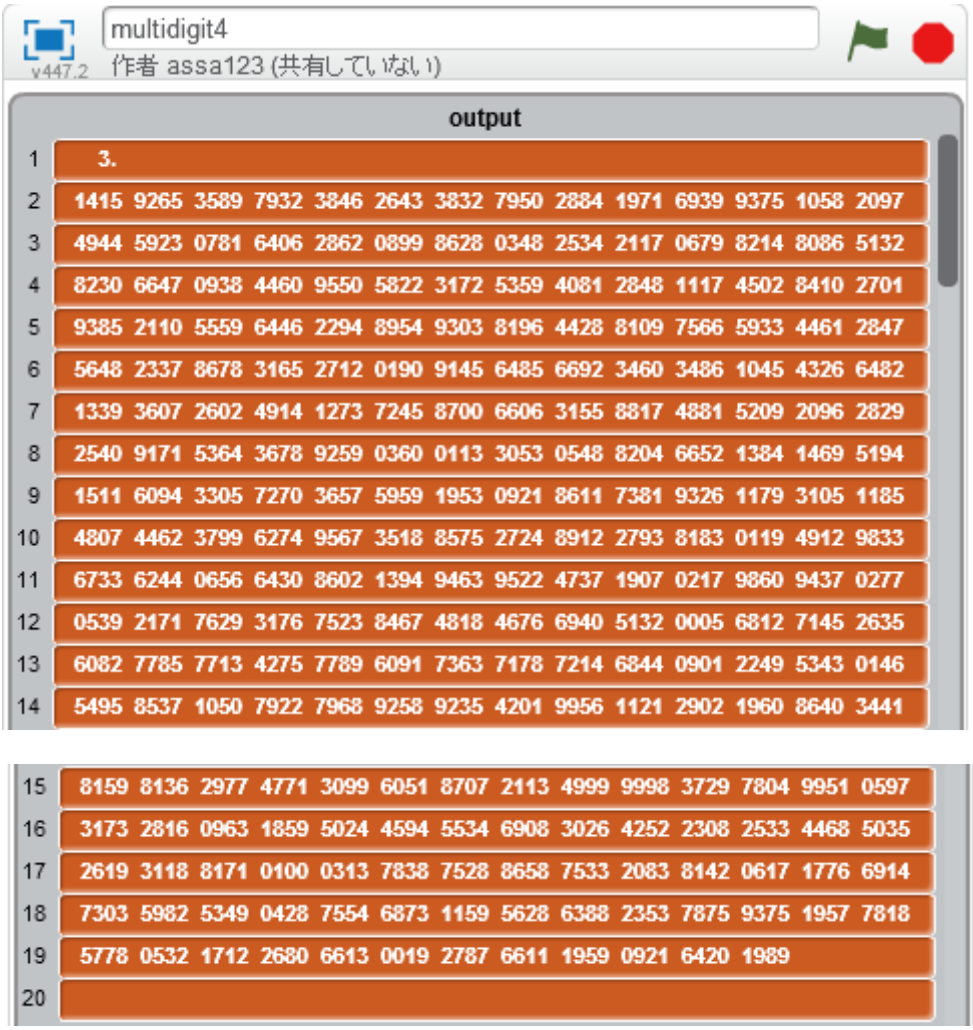

定義 ldiv3
i を 1 にする
s の長さ 回繰り返す
i 番目(a)を i 番目(q) で置き換える
i を 1 ずつ変える
ldiv 2 * k - 1
i を 1 にする
s の長さ 回繰り返す
i 番目(q)を i 番目(c) で置き換える
i を 1 ずつ変える


定義 ladd1
i を 1 にする
s の長さ 回繰り返す
i 番目(a)を i 番目(s) で置き換える
i 番目(b)を i 番目(q) で置き換える
i を 1 ずつ変える
ladd
i を 1 にする
s の長さ 回繰り返す
i 番目(s)を i 番目(c) で置き換える
i を 1 ずつ変える


定義 lsub1
i を 1 にする
s の長さ 回繰り返す
i 番目(a)を i 番目(w) で置き換える
i 番目(b)を i 番目(v) で置き換える
i を 1 ずつ変える
lsub
i を 1 にする
s の長さ 回繰り返す
i 番目(q)を i 番目(c) で置き換える
i を 1 ずつ変える


定義 lsub2
i を 1 にする
s の長さ 回繰り返す
i 番目(a)を i 番目(s) で置き換える
i 番目(b)を i 番目(q) で置き換える
i を 1 ずつ変える
lsub
i を 1 にする
s の長さ 回繰り返す
i 番目(s)を i 番目(c) で置き換える
i を 1 ずつ変える


定義 disp
print 1 番目(s) と . 5
newline
s_i を 2 にする
L - 3 回繰り返す
zeropadd s_i 番目(s)
print zeropadd_r 5
もし s_i - 1 を 14 で割った余り = 0 なら
newline
s_i を 1 ずつ変える


multidigit4
v447.2
作者 assa123 (共有していない)
output
1 3.
2 1415 9265 3589 7932 3846 2643 3832 7950 2884 1971 6939 9375 1058 2097
3 4944 5923 0781 6406 2862 0899 8628 0348 2534 2117 0679 8214 8086 5132
4 8230 6647 0938 4460 9550 5822 3172 5359 4081 2848 1117 4502 8410 2701
5 9385 2110 5559 6446 2294 8954 9303 8196 4428 8109 7566 5933 4461 2847
6 5648 2337 8678 3165 2712 0190 9145 6485 6692 3460 3486 1045 4326 6482
7 1339 3607 2602 4914 1273 7245 8700 6606 3155 8817 4881 5209 2096 2829
8 2540 9171 5364 3678 9259 0360 0113 3053 0548 8204 6652 1384 1469 5194
9 1511 6094 3305 7270 3657 5959 1953 0921 8611 7381 9326 1179 3105 1185
10 4807 4462 3799 6274 9567 3518 8575 2724 8912 2793 8183 0119 4912 9833
11 6733 6244 0656 6430 8602 1394 9463 9522 4737 1907 0217 9860 9437 0277
12 0539 2171 7629 3176 7523 8467 4818 4676 6940 5132 0005 6812 7145 2635
13 6082 7785 7713 4275 7789 6091 7363 7178 7214 6844 0901 2249 5343 0146
14 5495 8537 1050 7922 7968 9258 9235 4201 9956 1121 2902 1960 8640 3441

15 8159 8136 2977 4771 3099 6051 8707 2113 4999 9998 3729 7804 9951 0597
16 3173 2816 0963 1859 5024 4594 5534 6908 3026 4252 2308 2533 4468 5035
17 2619 3118 8171 0100 0313 7838 7528 8658 7533 2083 8142 0617 1776 6914
18 7303 5982 5349 0428 7554 6873 1159 5628 6388 2353 7875 9375 1957 7818
19 5778 0532 1712 2680 6613 0019 2787 6611 1959 0921 6420 1989
20

3章　文字列処理

コンピュータで扱うデータをおおざっぱに分けると、数値データと文字列データです。2章では数値データを処理する各種アルゴリズムを紹介しました。この章では文字列データを処理する各種アルゴリズムを紹介します。

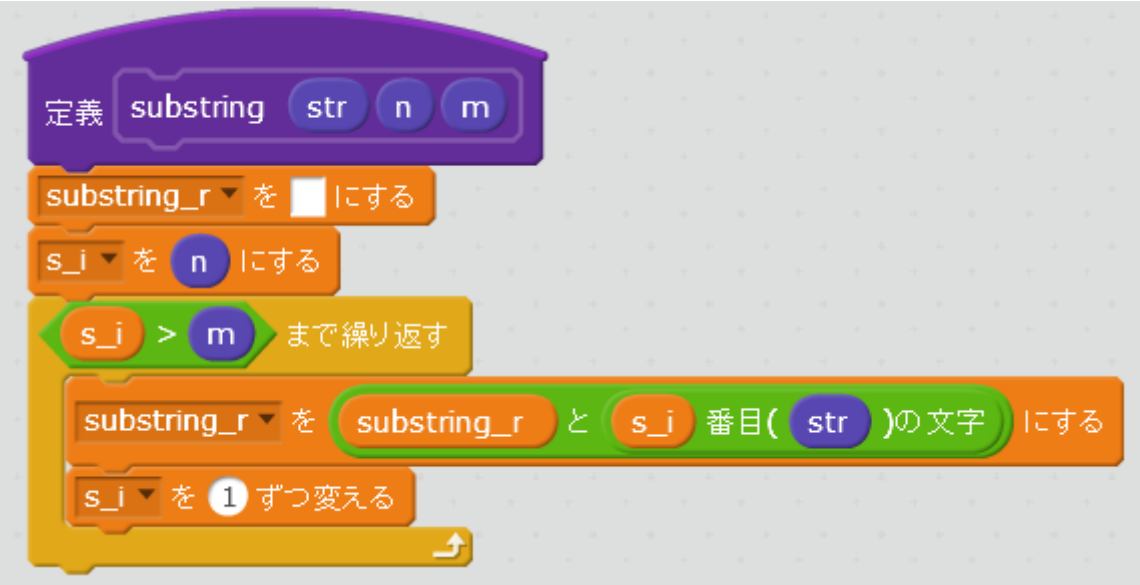
・部分文字列の取得
・逆文字列の取得
・文字検査
・文字列のサーチ
・文字列の置き換え（リプレイス）
・トークンの切り出し
・文字列の大小比較
・暗号
・ASCII コード

３－１　部分文字列の取得

部分文字列を取得するユーザーブロックを作ります。

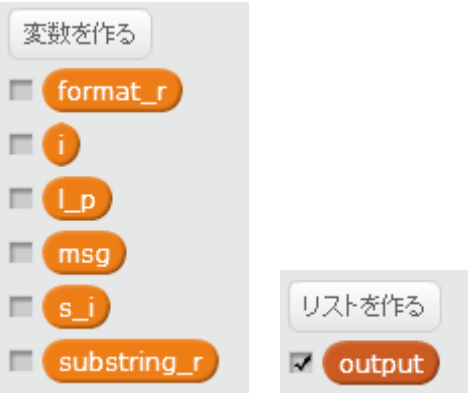
■substring(str,n,m)

str で与えられた文字列の n 番目の文字からm番目の文字までを切り取ります。切り取った部分文字列は substring_r に格納されています。

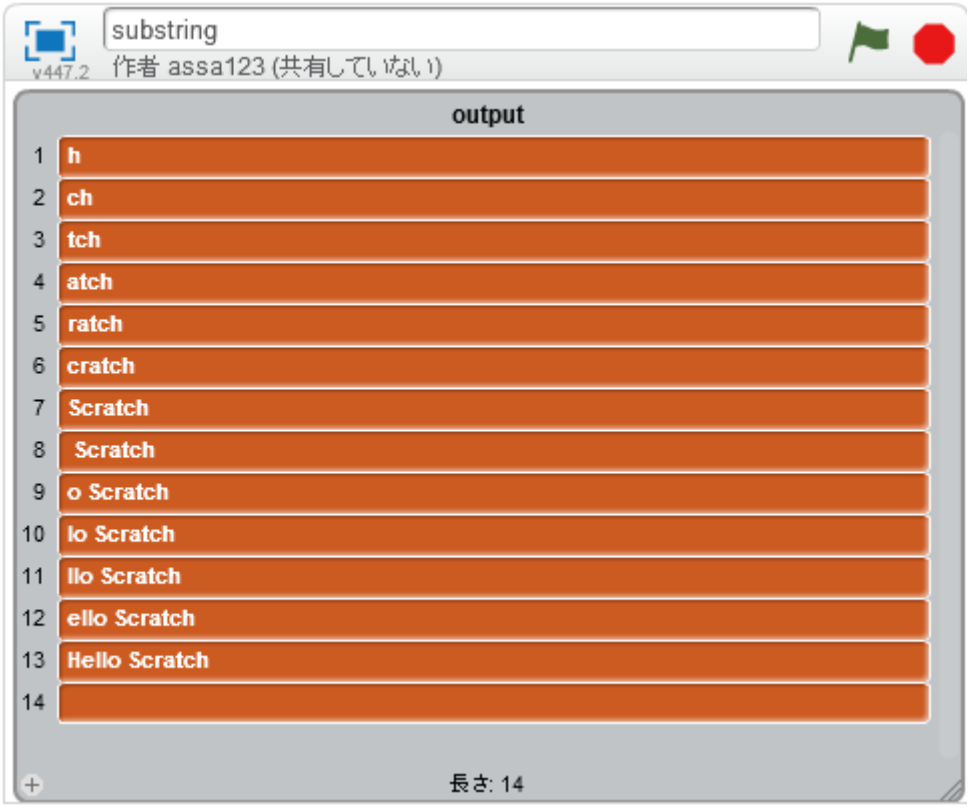


例題 3-1　切り取った文字列の表示

「Hello Scratch」という文字列を後ろから１文字、２文字、３文字・・・と切り取り表示します。




がクリックされたとき
init
msg を Hello Scratch にする
i を 1 にする
i > msg の長さ まで繰り返す
substring msg msg の長さ - i + 1 msg の長さ
println substring_r 0
i を 1 ずつ変える


substring
v447.2
作者 assa123 (共有していない)
output
1 h
2 ch
3 tch
4 atch
5 ratch
6 cratch
7 Scratch
8 Scratch
9 o Scratch
10 lo Scratch
11 llo Scratch
12 ello Scratch
13 Hello Scratch
14
長さ: 14

３－６　トークンの切り出し

文字列を指定した区切り文字でトークンに分離するユーザブロックを作ります。

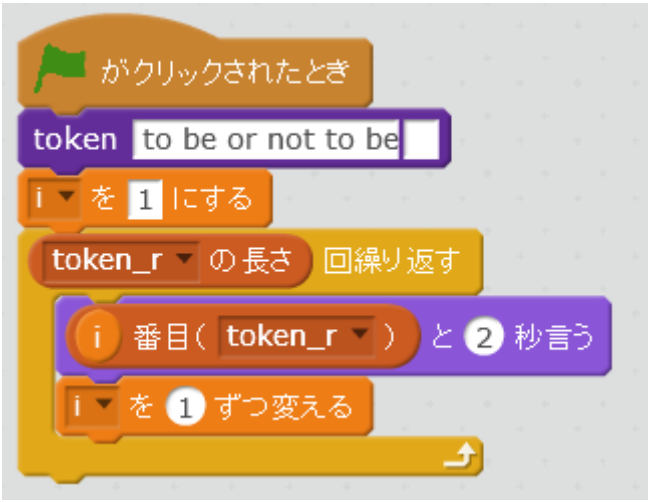
■token(str1,str2)

文字列 str1 を文字列 str2 で示す区切り文字でトークンに分離します。分離されたトークンはリスト token_r に格納します。

・substring は「３－１　部分文字列の取得」で示したものです。

例題 3-6　トークンの切り出し

「to be or not to be」という文字列を空白を区切りにしてトークンに分離します。




token
v447.2
作者 assa123 (共有していない)
token_r
1 to
2 be
3 or
4 not
5 to
6 be
長さ: 6
to

４章　乱数の利用

規則性がなく、まったくでたらめに並ぶ数を乱数といいます。一見するとでたらめな数など役に立たないと思いがちですが、プログラムの世界では乱数を利用することで様々な処理が行えます。たとえば、乱数でπの値を求めたり、面積を求めたりすることができます。これはモンテカルロ法というものです。ドリルなどの問をいつも同じ順序で出していていたのでは、意味がないので、乱数で適当な順序を作ることができます。また、２つのサイコロの目の和の出現分布をシミュレーションすることなどもできます。この章では乱数を利用したアルゴリズムについて説明します。

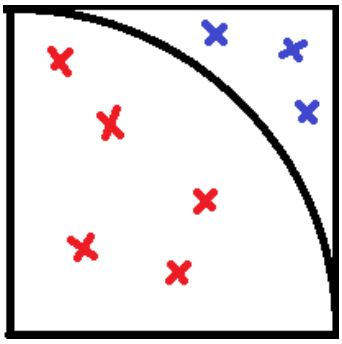
・乱数の一様性
・サイコロの目の和
・線形合同法による乱数の生成
・ボックスミューラー法による正規乱数
・彷徨う
・ランダムな順列
・英単語ドリル
・モンテカルロ法

４－８　モンテカルロ法

ある問題を、数値計算のような解析的な方法でなく、乱数を用いたシミレーションにより問題を解決する方法をモンテカルロ法といいます。乱数を用いた一種の賭けのような方法で問題を解くことからカジノで有名なモンテカルロ（Monte-Carlo）という名前が付けられたそうです。

例題 4-8-1　モンテカルロ法によるπの計算

円周率πをこの方法で求めるには次のようになります。



0～1 の一様乱数を２つ発生させ、それらを x,y とします。こうした乱数の組をいくつか発生させると、１×１の正方形の中に(x,y)で示される点は均一にばらまかれると考えられます。

したがって、正方形の面積と１／４円の面積の比は、そこにばら撒かれた乱数の数に比例するはずです。

今、1／4 円の中にばらまかれた乱数の数を a、全体にばらまかれた乱数の数を n とすると、次の関係が成立します。

$\pi/4 : 1 = a : n$

$\therefore \pi = 4a/n$

具体的にモンテカルロ法でπの値を求めるには以下のようにします。

・0～100 の乱数を x,y に得ます。

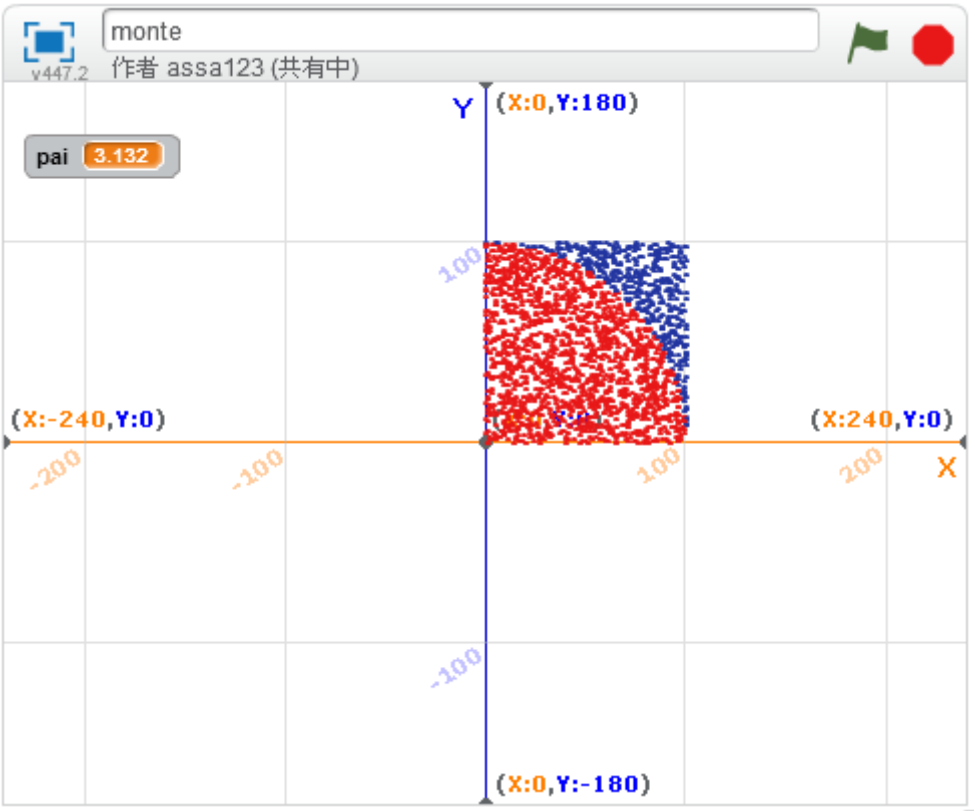
・$x^2+y^2<1000$ が円内に入る条件です。

・円内に入れば変数 a をカウントアップします。

・円内なら赤で点を打ちます、円外なら青で点を打ちます。点とは 1 歩歩いた直線です。


変数を作る
a
n
pai
x
y


がクリックされたとき
隠す
消す
ペンの太さを 2 にする
n を 2000 にする
a を 0 にする
n 回繰り返す
x を 0 から 100 までの乱数 にする
y を 0 から 100 までの乱数 にする
もし x * x + y * y < 10000 なら
ペンの色を にする
a を 1 ずつ変える
でなければ
ペンの色を にする
ペンを上げる
x座標を x 、y座標を y にする
ペンを下ろす
1 歩動かす
pai を 4 * a / n にする


v447.2
monte
作者 assa123 (共有中)
pai 3.132
Y
(X:0,Y:180)
100
(X:-240,Y:0)
(X:240,Y:0)
-200
-100
100
200
X
-100
(X:0,Y:-180)

例題 4-8-2　モンテカルロ法による面積の計算

以下で示される楕円の面積をモンテカルロ法で求めます。

$x^2/4+y^2=1$

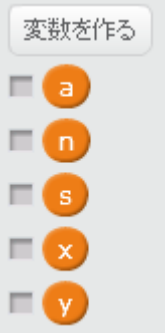
x に 0.0~2.0 の乱数、y に 0.0~1.0 の乱数を対応させ、２×１の長方形の中に均一にばらまきます。1/4 の楕円の中に入った乱数の数を a、乱数の総数を n、1/4 の楕円の面積を s とすると、

$2:s=n:a$
$\therefore s=2a/n$

となり求める楕円の面積 S は

$S=4・s=4・2a/n$

となります。




がクリックされたとき
n を 2000 にする
a を 0 にする
n 回繰り返す
x を 0.0 から 2.0 までの乱数 にする
y を 0.0 から 1.0 までの乱数 にする
もし x * x / 4 + y * y < 1 なら
a を 1 ずつ変える
s を 4 * 2 * a / n にする
楕円の面積は と s と言う


monte2
v447.2
作者 assa123 (共有していない)
楕円の面積は6.368

５章　リストデータの処理

リストに格納されているデータから標準偏差や度数分布などの各種統計を取ったり、データを小さい順または大きい順に並べ換え（ソート）たり、指定したデータを検索（サーチ）したりといったリストデータの処理を説明します。

・標準偏差
・度数分布
・順位付け
・バブルソート
・直接選択法
・シェルソート
・ポインタのソート
・逐次探索と番兵
・二分探索

５－４　バブルソート

プログラムにおいて、データを一定の規則で並べ変えることをソート（整列）といいます。小さい順に並べることを昇順、大きい順に並べることを降順といいます。たとえば、次のようなデータがあったとします。

70　　51　　30　　80　　50　　56

このデータに対し「隣合うデータを比較し、左側に小さいデータ、右側に大きいデータになるように交換する。」という操作を行なうことでソートを行ないます。

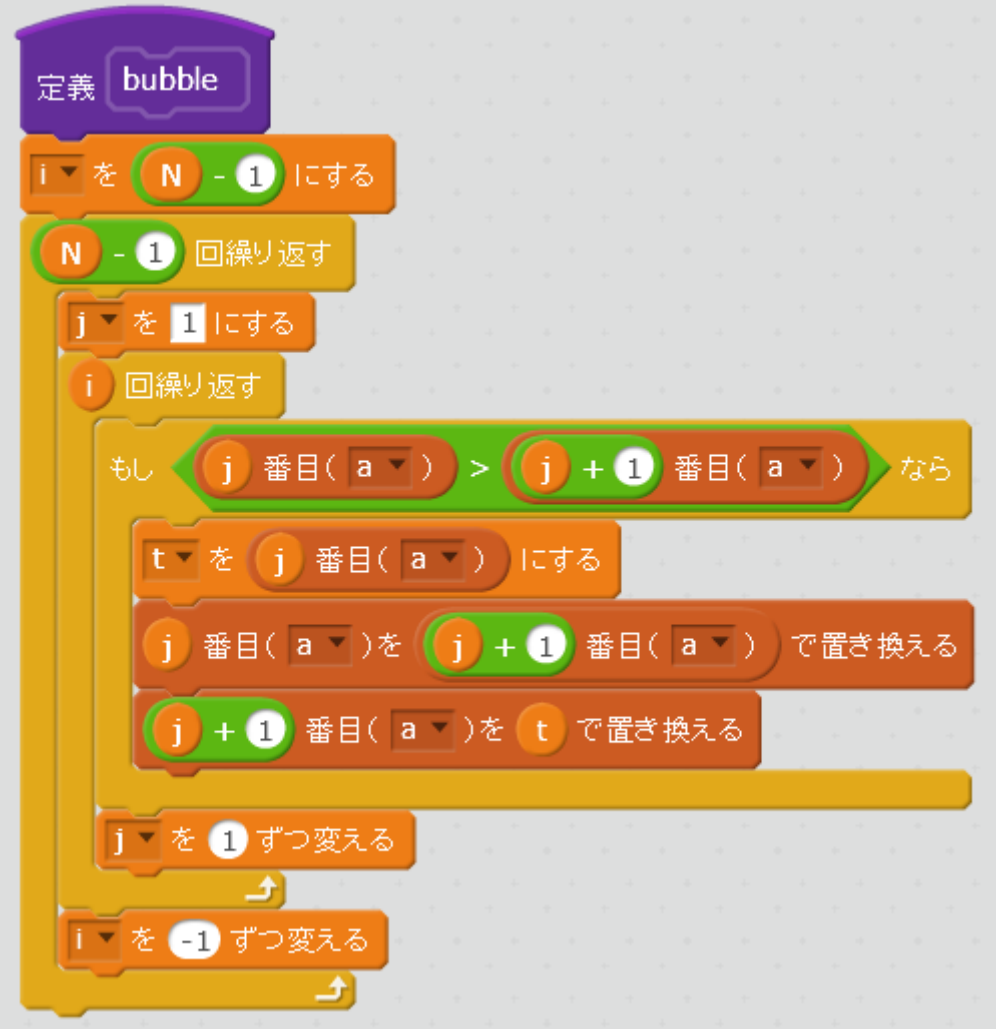
・まず「70」と「51」を比較し、「70」の方が大きいので、「70」と「51」を交換する。
・次に「70」と「30」を比較し、「70」の方が大きいので、「70」と「30」を交換する。
・次に「70」と「80」を比較し、「70」の方が小さいので、交換はしない。
・次に「80」と「50」を比較し、「80」の方が大きいので交換する。
・次に「80」と「56」を比較し、「80」の方が大きいので、「80」と「56」を交換する。

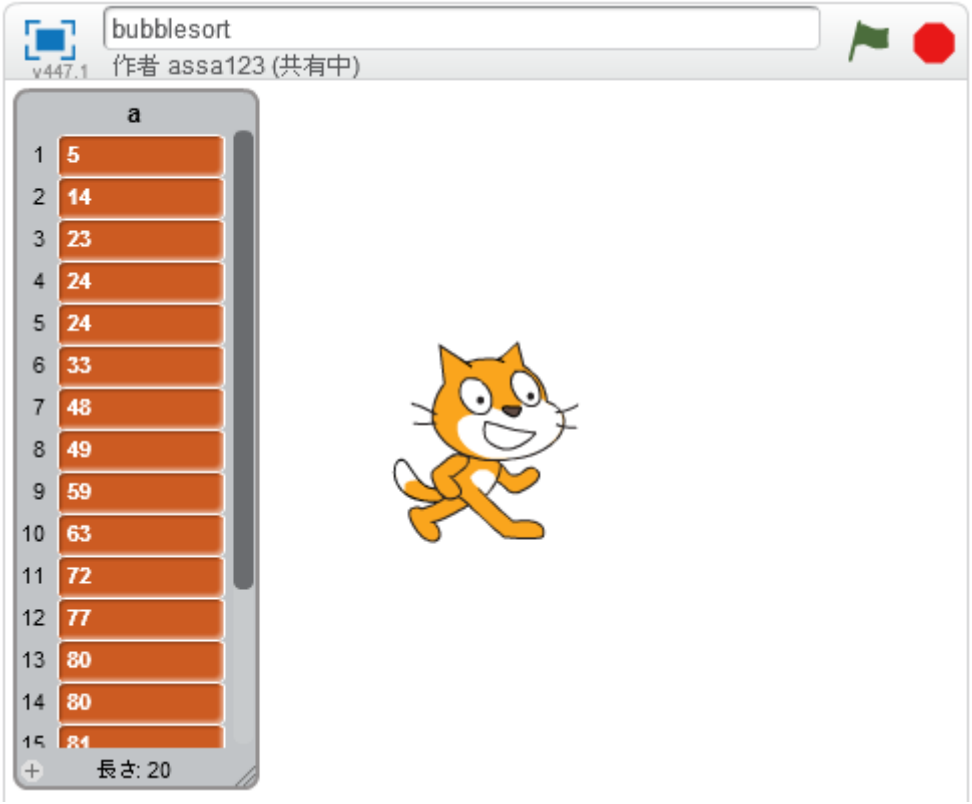
この結果データは「51 30 70 50 56 80」と並び変わります。一番大きい「80」が右端に押し出されて確定します。
こんどは「51 30 70 50 56」という１つデータ数の少ない数列に対し同じ処理をします。すると、「30 51 50 56 70」となり、「70」が確定します。以後「30 51 50 56」、「30 50 51」、「30 50」と数列の数が減少して行き、「30 50」の数列の比較と交換（必要なら）を行って、ソート作業は終了します。大きいデータ（あるいは小さいデータ）が泡のように右端にあがっていくイメージから、このようなソート法をバブル・ソートと呼びます。

例題 5-4　バブルソート

リスト a に 20 個のデータを乱数を使って格納します。データの範囲は 1 ~100 とします。このデータをバブルソートで小さい順に並べ替えます。


がクリックされたとき
a の長さ 回繰り返す
1 番目を a から削除する
N を 20 にする
N 回繰り返す
1 から 100 までの乱数 を a に追加する
bubble


定義 bubble
i を N - 1 にする
N - 1 回繰り返す
j を 1 にする
i 回繰り返す
もし j 番目(a) > j + 1 番目(a) なら
t を j 番目(a) にする
j 番目(a)を j + 1 番目(a) で置き換える
j + 1 番目(a)を t で置き換える
j を 1 ずつ変える
i を -1 ずつ変える


bubblesort
v447.1
作者 assa123 (共有中)
a
1 5
2 14
3 23
4 24
5 24
6 33
7 48
8 49
9 59
10 63
11 72
12 77
13 80
14 80
長さ: 20

５－５　直接選択法

部分数列 a_i~a_n の中から最小項を探し、それと a_i を交換することを、部分数列 a_1~a_n から始め、部分数列が a_n になるまで繰り返します。これが直接選択法と呼ばれるソートです。
これをアルゴリズムとして記述すると次のようになります。
①対象項 i を 1~n-1 まで移しながら、以下を繰り返す。
　②対象項を最小値の初期値とする。
　③対象項+1~n 項について以下を繰り返す。
　　④最小項を探し、その番号を s に求める。
　⑤i 項と s 項を交換する。

例題 5-5　直接選択法

リスト a に 20 個のデータを乱数を使って格納します。データの範囲は 1 ~100 とします。
このデータを直接選択法で小さい順に並べ替えます。


定義 direct
i を 1 にする
N - 1 回繰り返す
min を i 番目(a) にする
s を j にする
j を i + 1 にする
j > N - 1 まで繰り返す
もし j 番目(a) < min なら
min を j 番目(a) にする
s を j にする
j を 1 ずつ変える
t を i 番目(a) にする
i 番目(a)を s 番目(a) で置き換える
s 番目(a)を t で置き換える
i を 1 ずつ変える


tyokusetsu
v447.1
作者 assa123 (共有中)
a
1 4
2 5
3 10
4 12
5 21
6 22
7 25
8 26
9 28
10 31
11 37
12 45
13 45
14 52
15 60
長さ: 20

6章　再帰

再帰という考え方は人間の一般的な感覚からは縁遠いものですがプログラムの世界では重要な考え方です。
再帰的（リカーシブ）な構造とは、自分自身（n 次）を定義するのに、自分自身より 1 次低い部分集合（n-1 次）を用い、さらにその部分集合は、より低次の部分集合を用いて定義するということを繰り返す構造です。このような構造を一般に再帰と呼んでいます。
ある手続きの内部で、再び自分自身を呼び出すような構造の手続きを再帰的手続きと呼び、手続き内部で再び 1 次低い自分自身を呼び出すことを再帰呼び出し（リカーシブ・コール）と呼びます。
再帰では一般に再帰からの脱出口を置かなければいけません。これがないと、再帰呼び出しが永遠に続いてしまうことになります。
再帰を用いると、複雑なアルゴリズムを明解に記述することができます。ここではハノイの塔や迷路を説明します。

・ユークリッドの互除法の再帰版
・漸化式の再帰版
・ハノイの塔
・ハノイの塔シュミレーション
・迷路

６－３　ハノイの塔

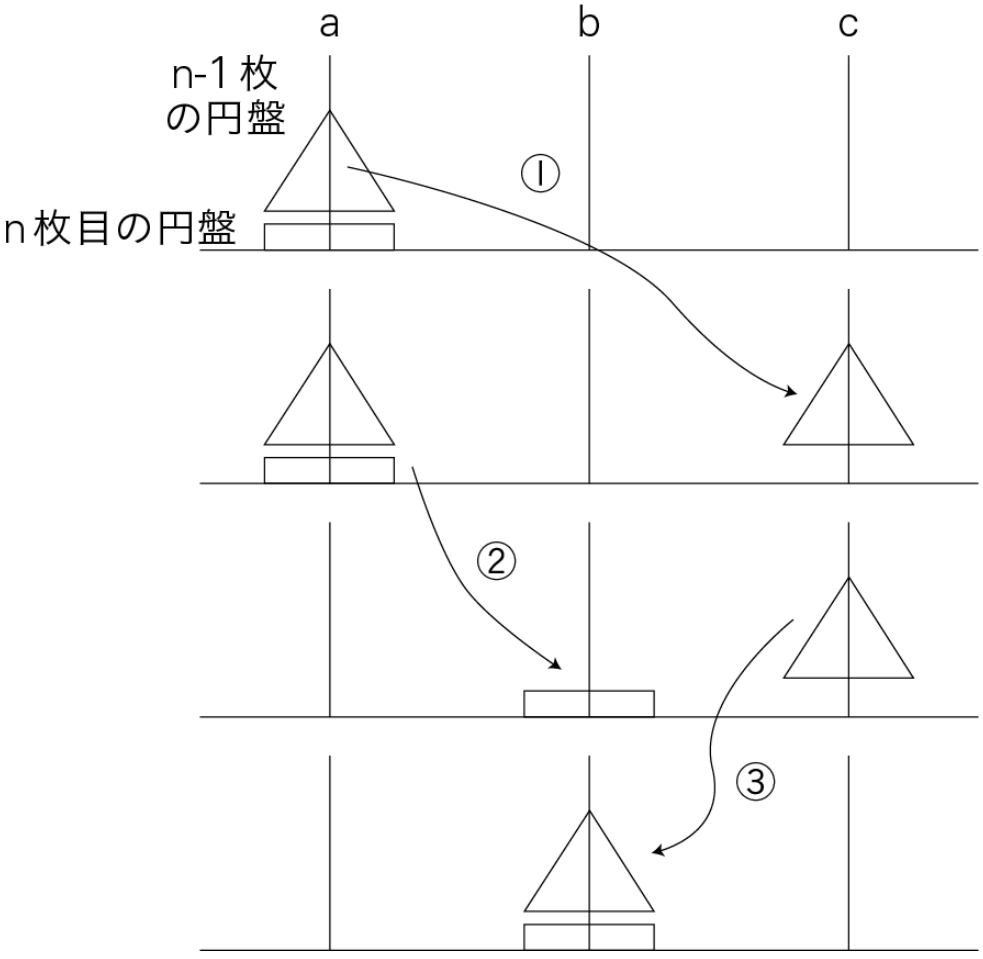
ハノイの塔とは次のようなパズルゲームです。

「３本の棒 a、b、c がある。棒 a に、中央に穴の空いた n 枚の円盤が大きい順に積まれている。これを１枚ずつ移動させて棒 b に移す。ただし、移動の途中で円盤の大小が逆に積まれてはならない。また、棒 c は作業用に使用するものとする。」

n 枚の円盤を a⇒b に移す作業は、次のような作業に分解できます。①と③の作業が再帰的な作業となります。

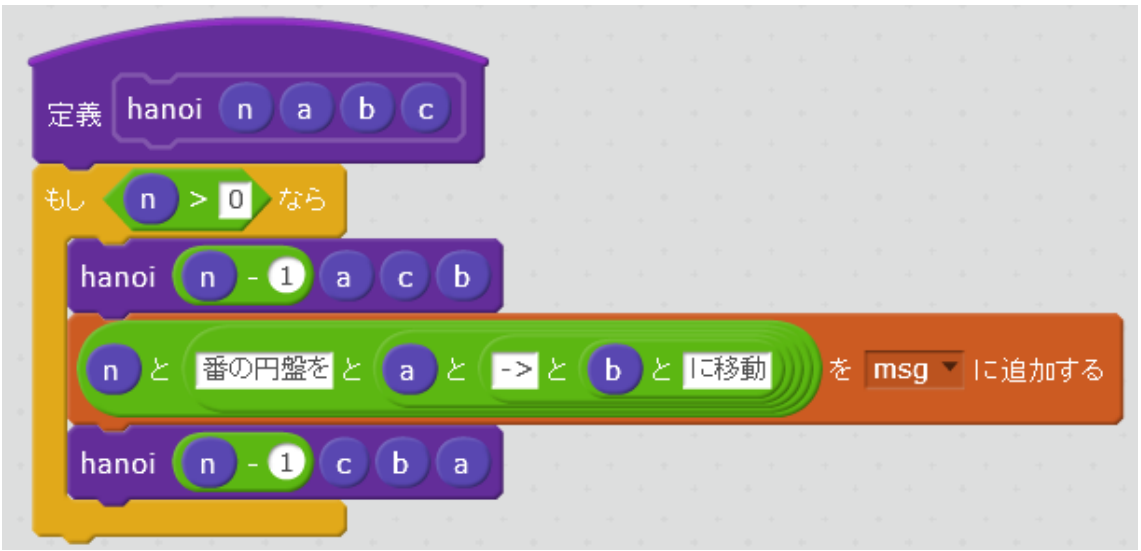
①a の n-1 枚の円盤を a⇒ c に移す。
②n 枚目の円盤を a⇒b に移す。
③c の n-1 枚の円盤を c⇒b に移す。

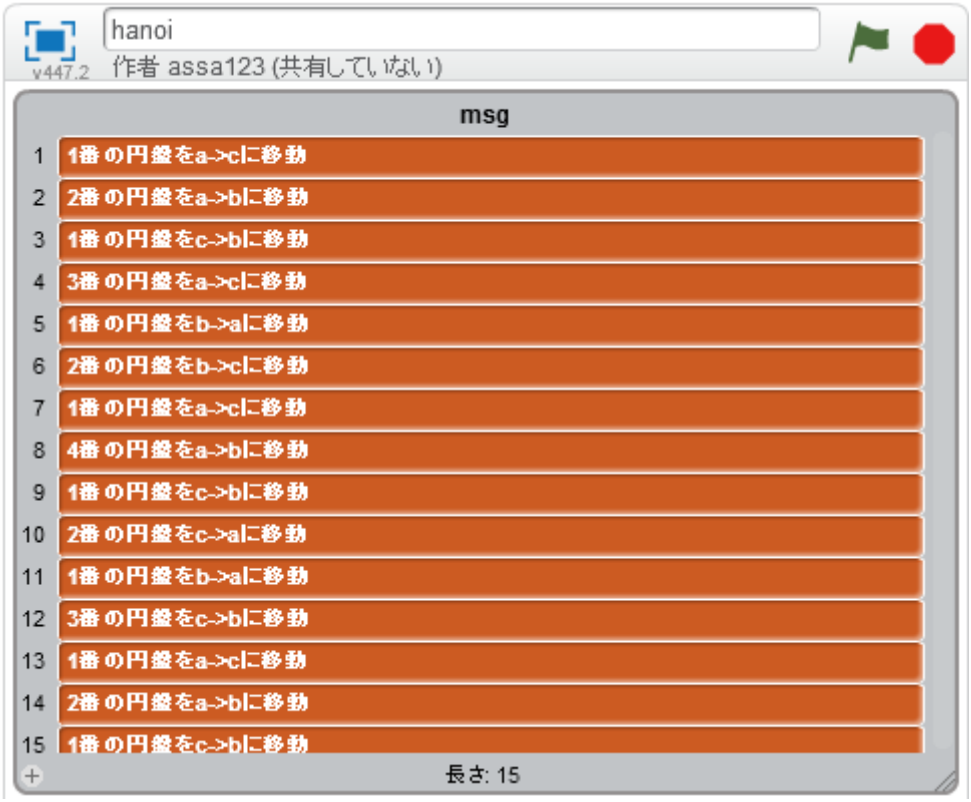
ハノイの塔



例題 6-3　ハノイの塔

ハノイの塔を解くユーザブロック hanoi(n,a,b,c)を作ります。



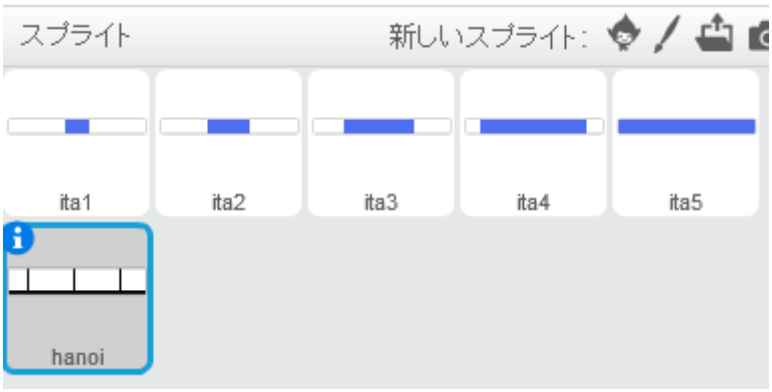

hanoi
v447.2
作者 assa123 (共有していない)
msg
1 1番の円盤をa->cに移動
2 2番の円盤をa->bに移動
3 1番の円盤をc->bに移動
4 3番の円盤をa->cに移動
5 1番の円盤をb->aに移動
6 2番の円盤をb->cに移動
7 1番の円盤をa->cに移動
8 4番の円盤をa->bに移動
9 1番の円盤をc->bに移動
10 2番の円盤をc->aに移動
11 1番の円盤をb->aに移動
12 3番の円盤をc->bに移動
13 1番の円盤をa->cに移動
14 2番の円盤をa->bに移動
15 1番の円盤をc->bに移動
長さ: 15

６－４　ハノイの塔シュミレーション

例題 6-3 ではハノイの塔の解は円盤の移動を文字でテキストエリアに表示しましたが、視覚的に円盤を実際に移動させます。

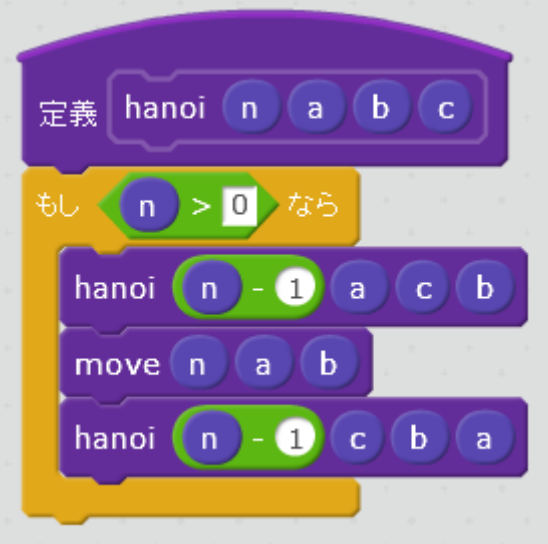
例題 6-4　ハノイの塔シュミレーション

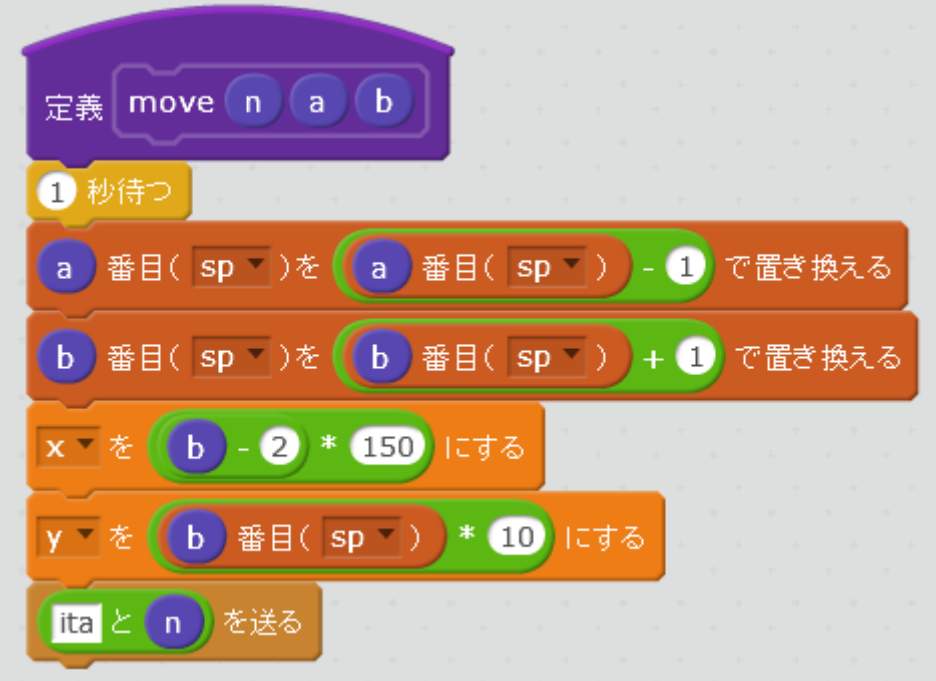
円盤を乗せる３本の棒がある台を hanoi とします。
円盤は ita1(最小)~ ita5（最大）の最大 5 枚まで用意します。sp[1]~sp[3]に a~c の各棒にある円盤の枚数を格納します。
move(n,a,b)ユーザブロックで n 番目の円盤を「引数 a で示す棒」⇒「引数 b で示す棒」に移動します。



・hanoi

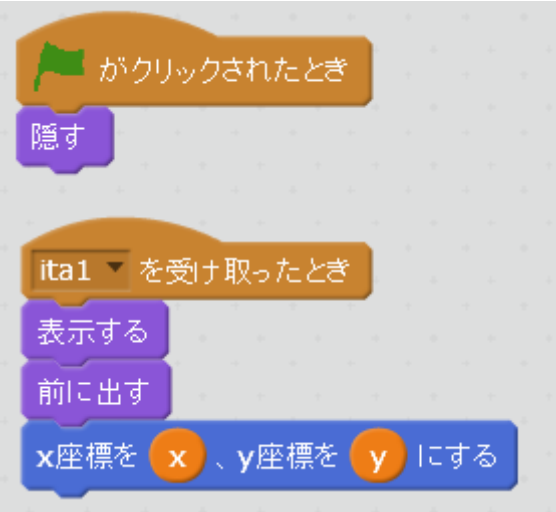

がクリックされたとき
x座標を 0 、y座標を 0 にする
何枚？ と聞いて待つ
n を 答え にする
1 番目(sp)を n で置き換える
2 番目(sp)を 0 で置き換える
3 番目(sp)を 0 で置き換える
y を 10 にする
x を -150 にする
i を n にする
n 回繰り返す
ita と i を送る
0.5 秒待つ
y を 10 ずつ変える
i を -1 ずつ変える
hanoi n 1 2 3

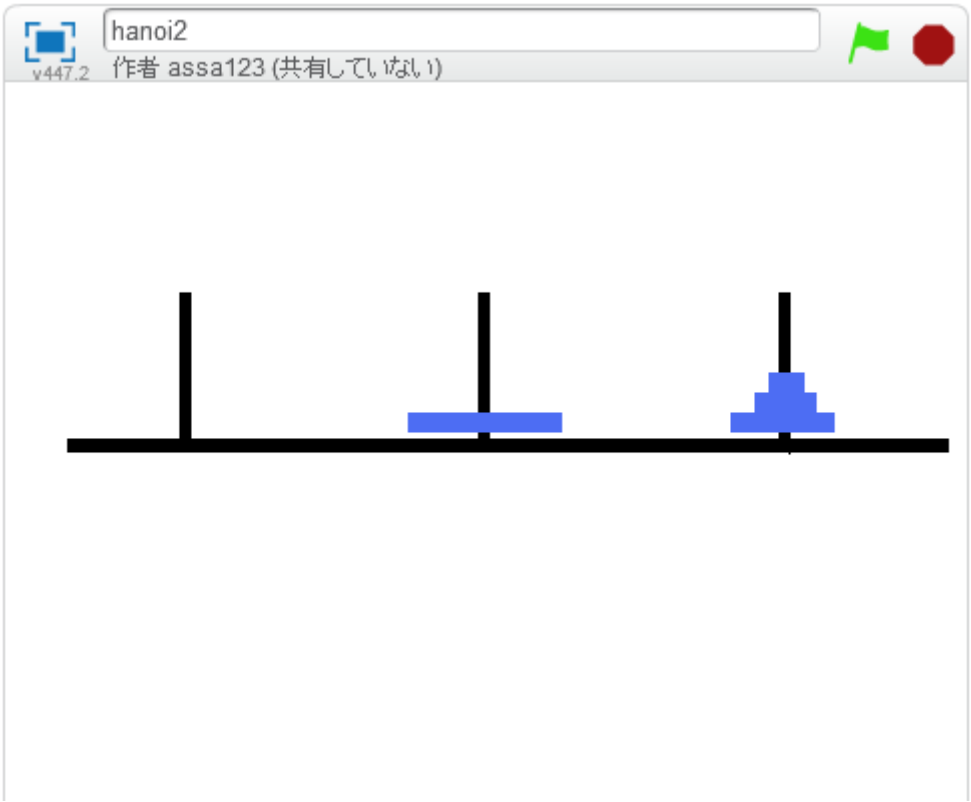

定義 hanoi n a b c
もし n > 0 なら
hanoi n - 1 a c b
move n a b
hanoi n - 1 c b a


定義 move n a b
1 秒待つ
a 番目(sp)を a 番目(sp) - 1 で置き換える
b 番目(sp)を b 番目(sp) + 1 で置き換える
x を b - 2 * 150 にする
y を b 番目(sp) * 10 にする
ita と n を送る

・ita1~ita5

メッセージはそれぞれ ita1~ita5 とします。





7章　データ構造

コンピュータ言語が持つデータ型には数値型、文字列型などの基本データ型とデータをまとめて管理するリストがあります。しかし、このようなコンピュータ言語が持つデータ型だけでは、大量のデータや複雑な構造のデータを効率よく操作することはできません。そこでデータ群を都合よく組織化するための抽象的なデータ型をデータ構造と呼びます。代表的データ構造として以下があります。

・表（table：テーブル）
・棚（stack：スタック）
・待ち行列（queue：キュー）
・リスト（list）
・木（tree：ツリー）
・グラフ（graph）

この章ではこれらのデータ構造を扱ったアルゴリズムについて説明します。

・リストの操作
・循環リスト
・自己再編成探索
・スタック
・キュー
・リストの２次元化（表：テーブル）
・逆ポーランド記法
・逆ポーランド式のパージング
・決定木
・二分探索木のサーチ
・木のトラバーサル
・式の木
・グラフの探索
・Euler の一筆書き
・知的データベース

７－３　自己再編成探索

逐次探索では、データの先頭から１つ１つ調べていくので、後ろにあるものほど探索に時間がかかります。
一般に一度使われたデータというのは再度使われる可能性が高いので、探索ごとに探索されたデータを前の方に移すようにすると、自ずと使用頻度の高いデータが前の方に移って来ます。このような方法を自己再編成探索といいます。
身近な例としては、ワープロで漢字変換をした場合、直前に変換した漢字が、今回の第一候補になる、いわゆる学習機能といわれるものがそうです。
自己再編成探索はデータの入れ替えを探索のたびに行うので、データの挿入・削除が容易なリストで実現するのがよいです。
データを再編成する方法として、次のようなものが考えられます
・探索データを先頭に移す
・探索データを１つ前に移す

例題 7-3-1　自己再編成探索で先頭に移す

「しよう」に対する漢字候補を探索し、見つかったらそれをリストの先頭に移動します。


がクリックされたとき
ずっと
単語を入力してください と聞いて待つ
i を 1 にする
flag を 0 にする
i > word の長さ または flag = 1 まで繰り返す
もし 答え = i 番目(word) なら
i 番目を word から削除する
答え を 1 番目に挿入する(word)
t を 0 にする


selforg
v447.2
作者 assa123 (共有していない)
word
1 使用
2 私用
3 仕様
4 止揚
5 枝葉
6 子葉
長さ: 6
単語を入力してください
私用

例題 7-3-2　自己再編成探索で一つ前に移動

「しよう」に対する漢字候補を探索し、見つかったらそれを一つ前に移動します。


selforg2
v447.2
作者 assa123 (共有していない)
word
1 使用
2 子葉
3 私用
4 仕様
5 止揚
6 枝葉
長さ: 6
単語を入力してください
私用

７－７　逆ポーランド記法

数式を

a+b-c*d/e

のように、オペランド（演算の対象となるもの）の間に演算子を置く書き方を挿入記法（中置記法）と呼び、数学の世界で一般に用いられています。これを、

ab+cd*e/-

のようにオペランドの後に演算子を置く書き方を、後置記法または逆ポーランド記法と呼びます。この式は「a と b を足し、c と d を掛け、それを e で割ったものを引く」というように式の先頭から読んでいけばよいのと、かっこが不要なため、演算ルーチンを簡単に作れることから、コンピュータの世界ではよく使われます。

■逆ポーランド記法に変換するアルゴリズム

挿入記法の式から、逆ポーランド記法の式に変換するアルゴリズムは以下のとおりですが、問題を簡単にするため、次のような条件をつけます。

・オペランドは１文字からなる
・演算子は＋、－、＊、／の４つの演算子だけとする
・式が誤っている場合のエラー処理はつけない

式の各要素を因子と呼びます。因子にはオペランドと演算子がありますが、これを評価する優先順位を次のように決めます。数字の大きいものが優先順位が高いとします。

因子	優先順位
オペランド	3
＊，／	2
＋、－	1

この優先順位表の値は priority(factor)ユーザブロックで得ることにします。

式を評価するとき、取り出した因子を格納する作業用の stack、逆ポーランド記法の式を作る polish という２つのスタックを用いて次のように行います。

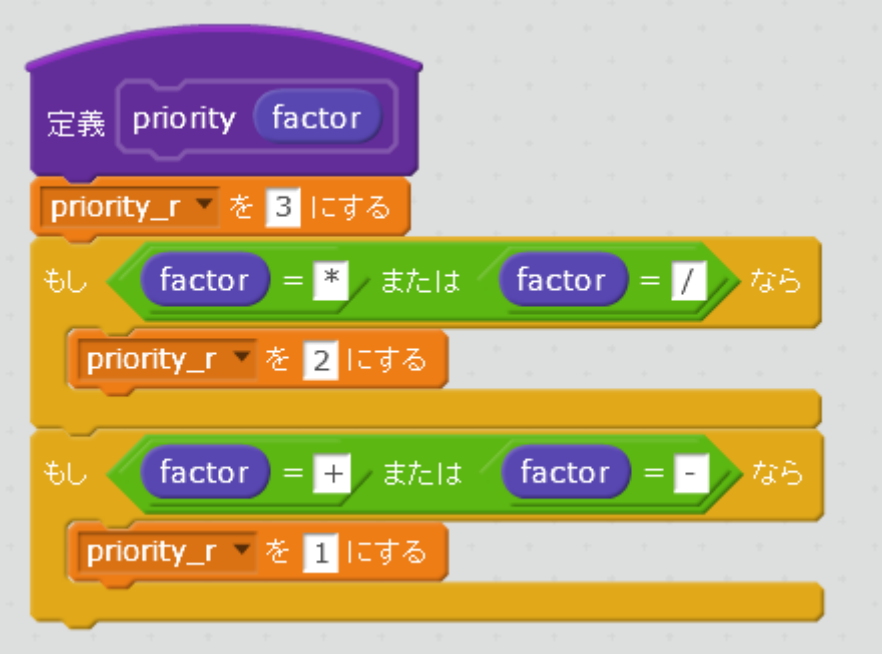
①式の終わりになるまで以下を繰り返す
　②式から１つの因子を取り出す
　③「取り出した因子の優先順位」＜＝「スタック・トップの因子の優先順位」である間、
　　polish に stack の最上位の因子を取り出して積む
　④②で取り出した因子を stack に積む
⑤stack に残っている因子を取り出し polish に積む

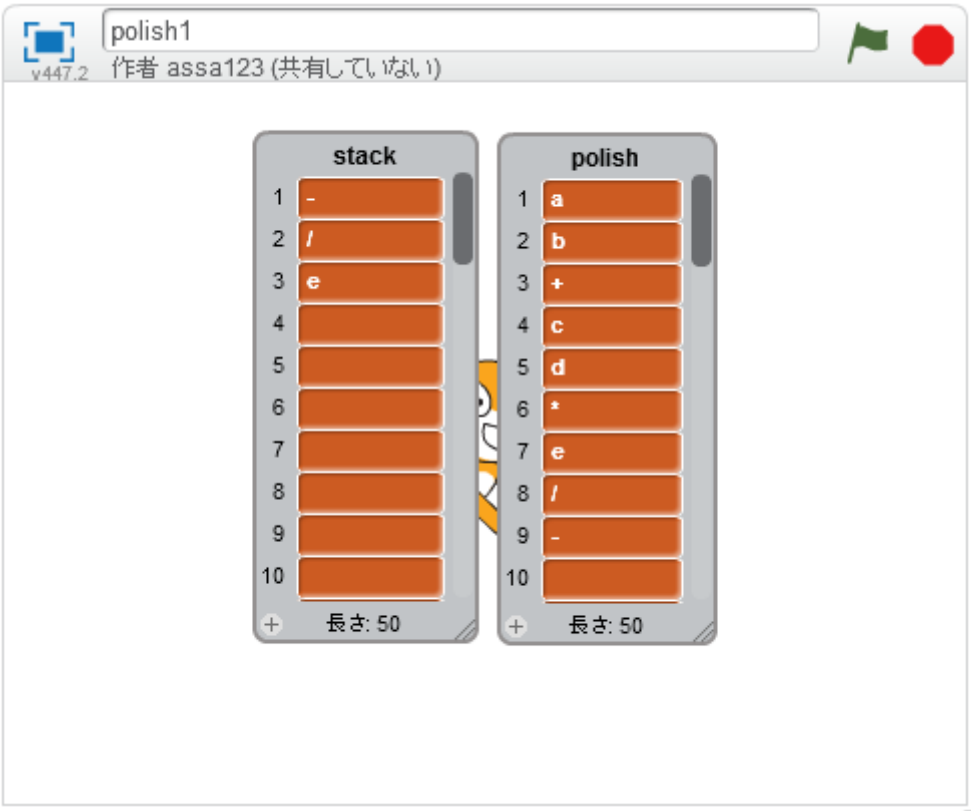
例題 7-7-1　逆ポーランド記法に変換

「a+b-c*d/e」という式を逆ポーランド記法に変換して結果を polish に格納します。


がクリックされたとき
arrayinit
exp を a+b-c*d/e にする
1 番目(stack)を 1 番目(exp)の文字 で置き換える
sp1 を 1 にする
sp2 を 1 にする
n を 2 にする
exp の長さ - 1 回繰り返す
p を n 番目(exp)の文字 にする
priority p
pri1 を priority_r にする
priority sp1 番目(stack)
pri2 を priority_r にする
pri1 > pri2 または sp1 < 1 まで繰り返す
sp2 番目(polish)を sp1 番目(stack) で置き換える
sp2 を 1 ずつ変える
sp1 を -1 ずつ変える
priority sp1 番目(stack)
pri2 を priority_r にする
sp1 を 1 ずつ変える
sp1 番目(stack)を p で置き換える
n を 1 ずつ変える
i を sp1 にする
sp1 回繰り返す
sp2 番目(polish)を i 番目(stack) で置き換える
sp2 を 1 ずつ変える
i を -1 ずつ変える


定義 arrayinit
stack の長さ 回繰り返す
1 番目を stack から削除する
1 番目を polish から削除する
50 回繰り返す
を stack に追加する
を polish に追加する


定義 priority factor
priority_r を 3 にする
もし factor = * または factor = / なら
priority_r を 2 にする
もし factor = + または factor = - なら
priority_r を 1 にする



■かっこの処理を含む

かっこを含む挿入記法の式を逆ポーランド記法の式に変換します。逆ポーランド記法の式ではかっこは取り除かれます。かっこの処理は次のように行います。

・(はそのまま stack に積む

・)は stack のトップが(になるまで、stack に積まれている因子を取り出し polish に積む。スタック・トップに来た(は捨てる。)は stack には積まない。

(と)の処理を特別扱いにしないようにするため次のようなルールをつけます。

・(の優先順位を最高にし、これをスタックに積む処理を別扱いしないようにする。しかしこの結果スタックからの取り出しのときに(を突き抜けてしまうので、スタック・トップが(なら取り出しを行わないという条件を付ける。

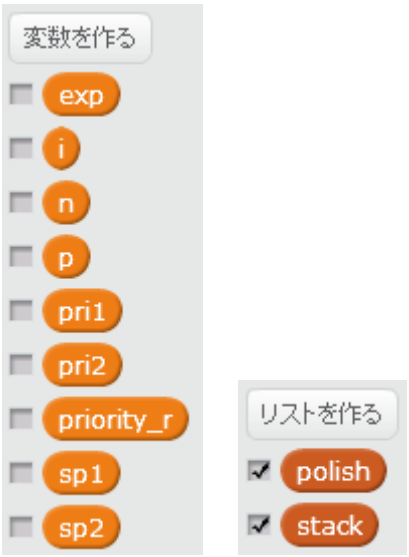
・)にも優先順位を与える。)の優先順位を最低にすることで、stack の内容が)に来るまで全部取り出される。この処理が終了し、スタック・トップに来た)を取り除く。

この場合の優先順位は次のようになります。数字が大きいものが優先順位が高いとします。

因子	優先順位
（	4
オペランド	3
＊，／	2
＋、－	1
）	0

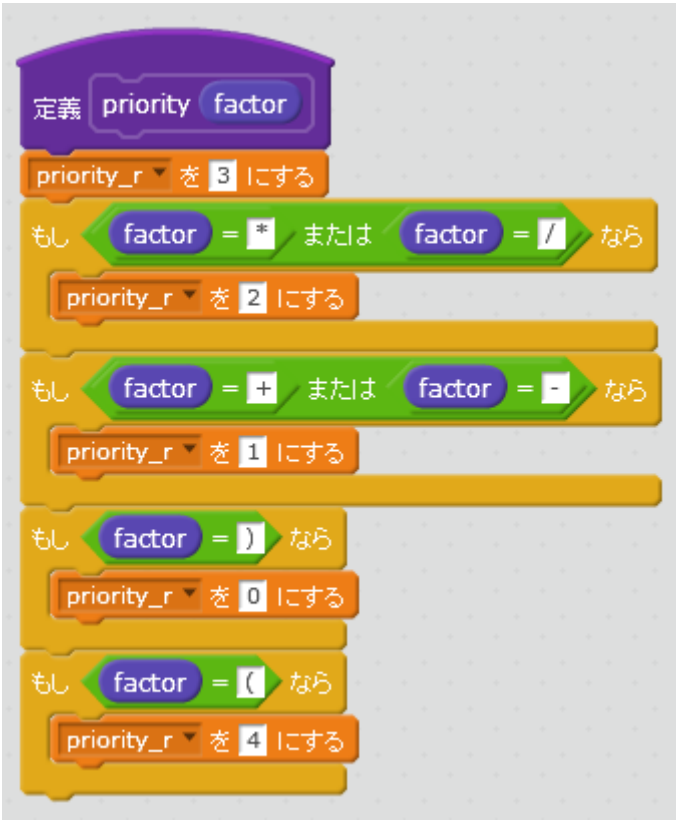
例題 7-7-2　かっこを含む式を逆ポーランド記法に変換

「(a+b)*(c+d)」というかっこを含む式を逆ポーランド記法に変換して、結果を polish に格納します。

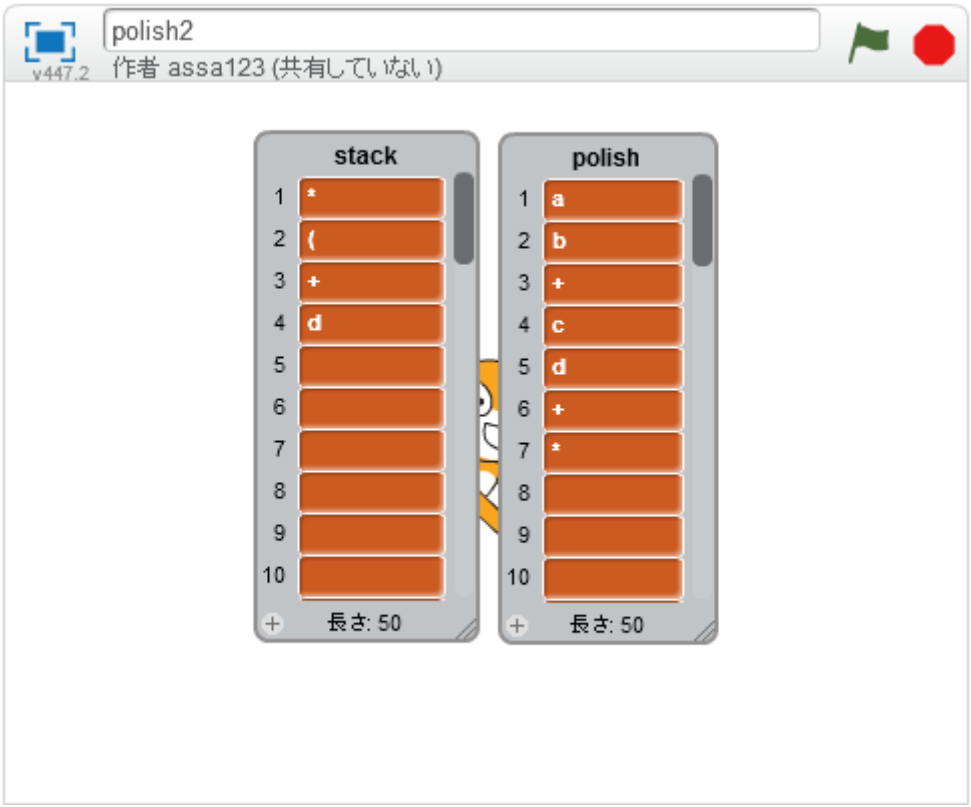


・上の青枠には以下が入ります。


もし p =) なら
sp1 を -1 ずつ変える
でなければ
sp1 を 1 ずつ変える
sp1 番目(stack)を p で置き換える
n を 1 ずつ変える
i を sp1 にする
sp1 回繰り返す
sp2 番目(polish)を i 番目(stack) で置き換える
sp2 を 1 ずつ変える
i を -1 ずつ変える


定義 priority factor
priority_r を 3 にする
もし factor = * または factor = / なら
priority_r を 2 にする
もし factor = + または factor = - なら
priority_r を 1 にする
もし factor =) なら
priority_r を 0 にする
もし factor = (なら
priority_r を 4 にする


定義 arrayinit
stack の長さ 回繰り返す
1 番目を stack から削除する
1 番目を polish から削除する
50 回繰り返す
を stack に追加する
を polish に追加する

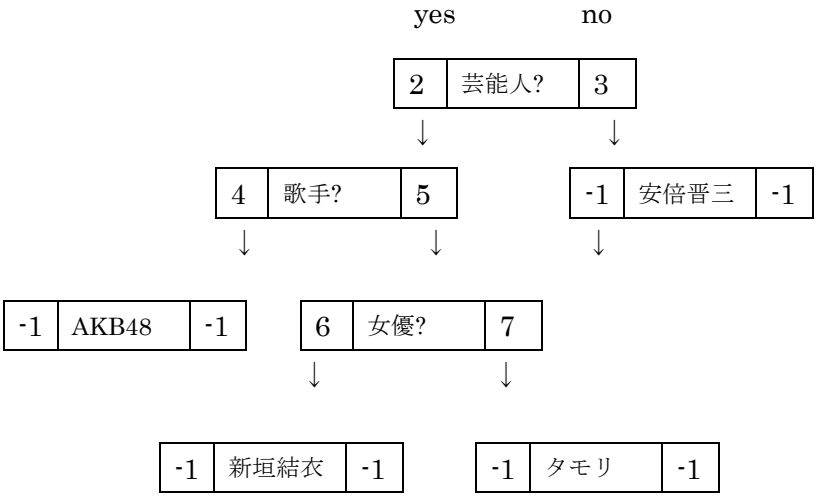

polish2
v447.2
作者 assa123 (共有していない)
stack
1 *
2 (
3 +
4 d
5
6
7
8
9
10
長さ: 50
polish
1 a
2 b
3 +
4 c
5 d
6 +
7 *
8
9
10
長さ: 50

７－１５　知的データベース

コンピュータはどんなに高速で正確な計算を行えても、自ら考えることはできません。しかし、自ら考え出すことはできなくても、いろいろな局面で起きた状況を記憶しておいて、次の局面でその情報を利用して、戦略なり、動作を決めることができます。このようにして構築したデータを知的データベースと呼びます。
ここではコンピュータと人間が対話を進める過程で、コンピュータに学習機能を持たせ、知的データベースを構築していくプログラムを作ります。

■知的データベースを構築する決定木

ある質問（たとえば「芸能人ですか」）に対し、答えを yes と no の２つに限定するなら、質問をノードとする二分木と考えられます。このような二分木を特に決定木といいます。



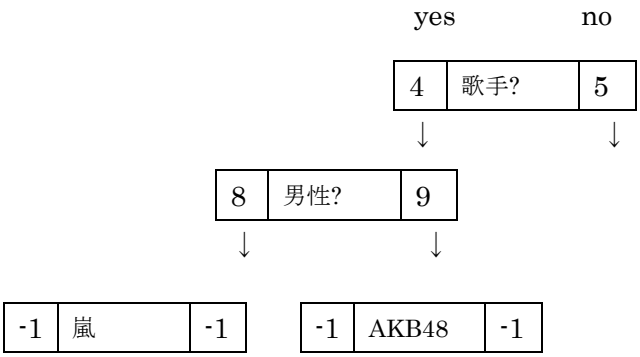
上の決定木を配列で表すと以下のようになります。

p	left[p]	node[p]	right[p]
1	2	芸能人ですか	3
2	4	歌手ですか	5
3	-1	安倍晋三	-1
4	-1	AKB48	-1
5	6	女優ですか	7
6	-1	新垣結衣	-1
7	-1	タモリ	-1
8			

■学習機能

上図の決定木では「芸能人ですか」という質問に yes と答えると、「歌手ですか」と聞いてきます。それに yes と答えると「答えは AKB48 です」とコンピュータが答えをだします。もし対話者が考えているものが、「嵐」だったら、コンピュータの出した答えは誤りであったことになります。この場合はコンピュータに「嵐」を導く情報を与えておかなければなりません。これが（簡単ではあるが）学習機能です。この情報を与えるには、「歌手ですか」の下に、「AKB48」と「嵐」を区別する質問を対話者に聞き、追加すればよいです。

「AKB48」と「嵐」を区別する質問ノードを「男性ですか」とし、この質問ノードを「AKB48」の位置に入れます。質問に対する yes のノードを左、no のノードを右にして、現在使用しているリスト要素の最後に追加します。質問ノードの左ポインタおよび右ポインタに「嵐」と「AKB48」のリスト要素位置を与えます。



例題 7-15　知的データベース

left,node,right で示す知的データベースの決定木をたどり質問に答え、回答が違っていたら新しい項目を決定木に追加して学習しながらデータベースを増やします。


がクリックされたとき
lp を node の長さ にする
ずっと
p を 1 にする
p 番目(left) = -1 まで繰り返す
p 番目(node) と聞いて待つ
もし 答え = y なら
p を p 番目(left) にする
でなければ
p を p 番目(right) にする
結果は と p 番目(node) と ですか と聞いて待つ


もし 答え = n なら
あなたの考えは と聞いて待つ
newnode を 答え にする
p 番目(node) と と と newnode と を区別する質問は と聞いて待つ
ques を 答え にする
yesの項目は と newnode と でよいですか と聞いて待つ
もし 答え = y なら
p 番目(left)を lp + 2 で置き換える
p 番目(right)を lp + 1 で置き換える
でなければ
p 番目(left)を lp + 1 で置き換える
p 番目(right)を lp + 2 で置き換える
-1 を left に追加する
-1 を left に追加する
-1 を right に追加する
-1 を right に追加する
p 番目(node) を node に追加する
newnode を node に追加する
p 番目(node)を ques で置き換える
lp を 2 ずつ変える


database
v447.2
作者 assa123 (共有していない)
left
1 2
2 4
3 -1
4 9
5 6
6 -1
7 -1
8 -1
9 -1
長さ: 9
node
1 芸能人ですか
2 歌手ですか
3 安倍晋三
4 男性ですか
5 女優ですか
6 新垣結衣
7 タモリ
8 AKB48
9 嵐
長さ: 9
right
1 3
2 5
3 -1
4 8
5 7
6 -1
7 -1
8 -1
9 -1
長さ: 9
結果は嵐ですか

８章　２次元グラフィックス

点の位置を示すのに座標を導入し、各点の位置関係が直線上にあるなら１次方程式、各点が円、楕円、双曲線、放物線などの曲線上にあるなら２次方程式で表せます。図形を方程式で表し、コンピュータにより解析的に解けば、平行移動、回転、拡大・縮小などの座標変換が簡単に行えます。この章では２次元平面上の図形を描くアルゴリズムを説明します。

・タートル・グラフィックス
・グラフィックス用ブロック
・円周上の点を結んでできる図形
・関数のグラフ
・２次元座標変換
・ウィンドウとビューポート
・ジオメトリック・グラフィックス

８－１　タートル・グラフィックス

方向と長さを与えて直線を引いていく形式のグラフィックスをタートル・グラフィックスといいいます。タートル・グラフィックスは LOGO や UCSD Pascal で有名になったもので、△形のカーソルをタートル（亀）にみたて、この亀に方向と長さを指定して作画していくというものです。Scratch も 

や 

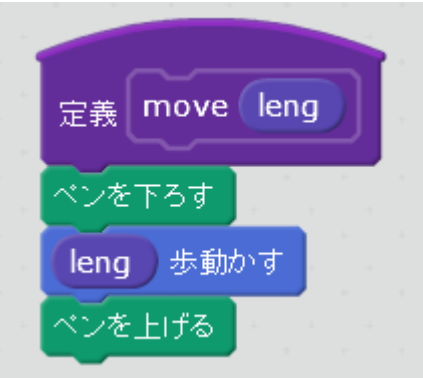
でタートル・グラフィックスを行うことができます。しかし実際に作画するにはペンの上げ下げが入ってくるので、これを含めたユーザーブロックとして setpoint と move を作ります。

■setpoint(x,y)

ペンを上げて位置を(x,y)位置に移動します。

■move(leng)

現在の描画方向に向かって長さ leng の直線を引きます。

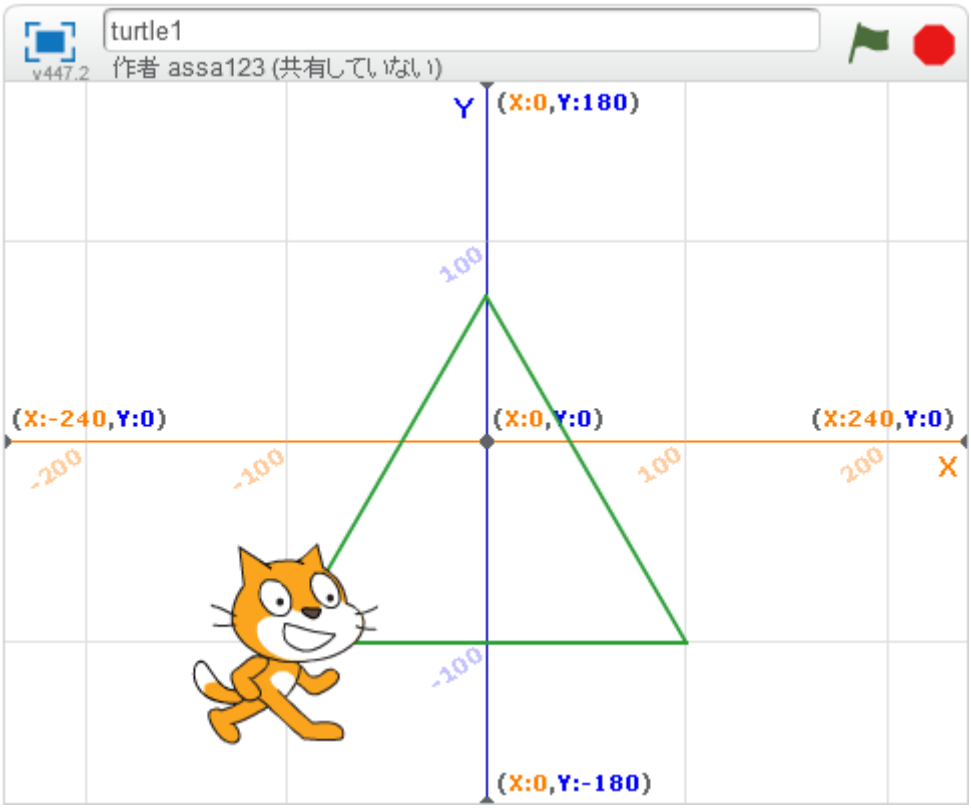


■init

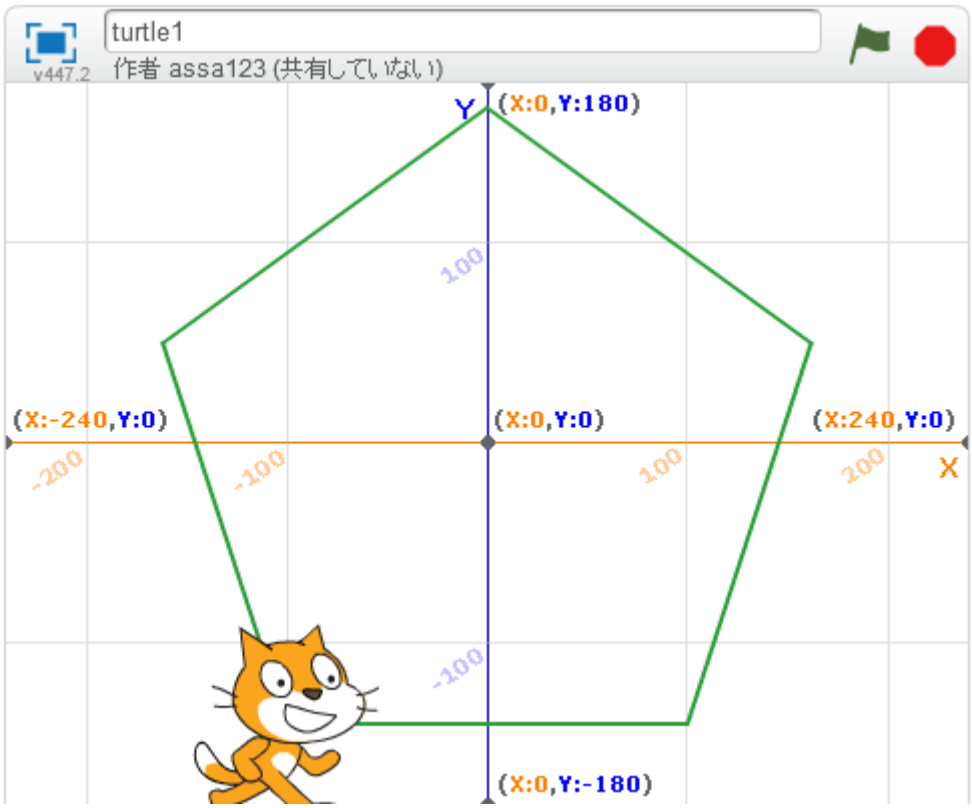
ペンの色、太さを設定します。描画方向を右方向に設定します。

例題 8-1-1　多角形を描く

正三角形は move(l)と turn(120)を３回行えばよいです。

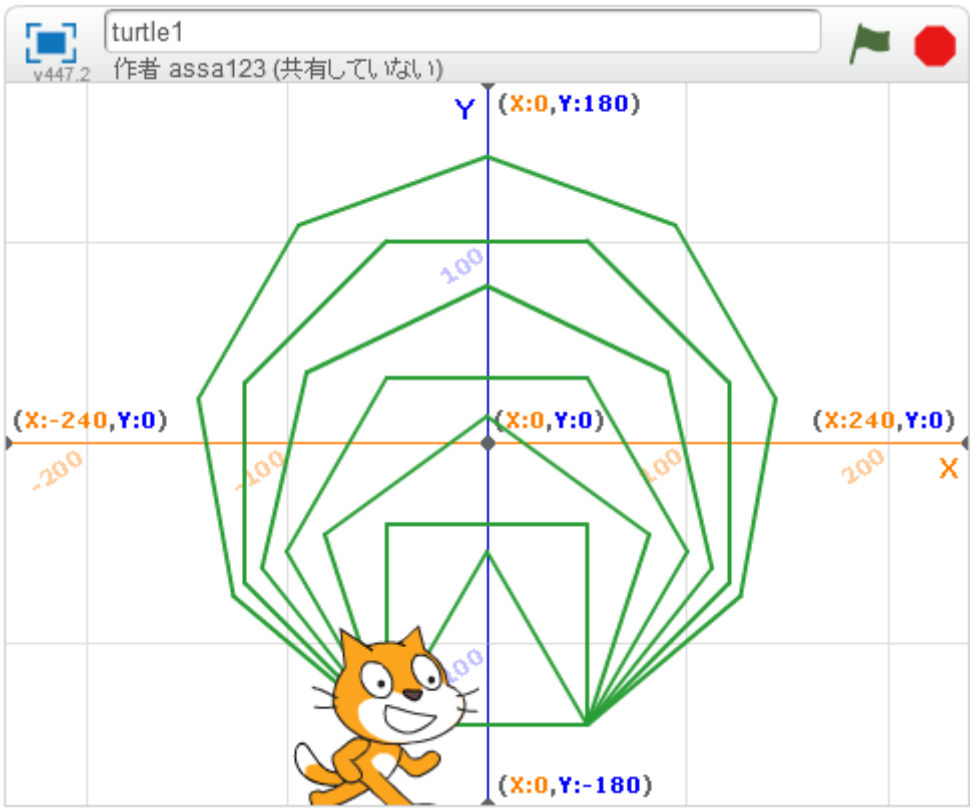


・正五角形は move(l)と turn(72)を 5 回行えばよいです。



例題 8-1-2　正三角形～正九角形を描く

正 n 角形のときの回転角は 360/n° です。



例題 8-1-3　星型多角形

一筆書きで描ける星型多角形は 5 角形、 7 角形、 9 角形・・・の奇数多角形です。

星型 n 角形の回転する角度は次式で求めることができます。

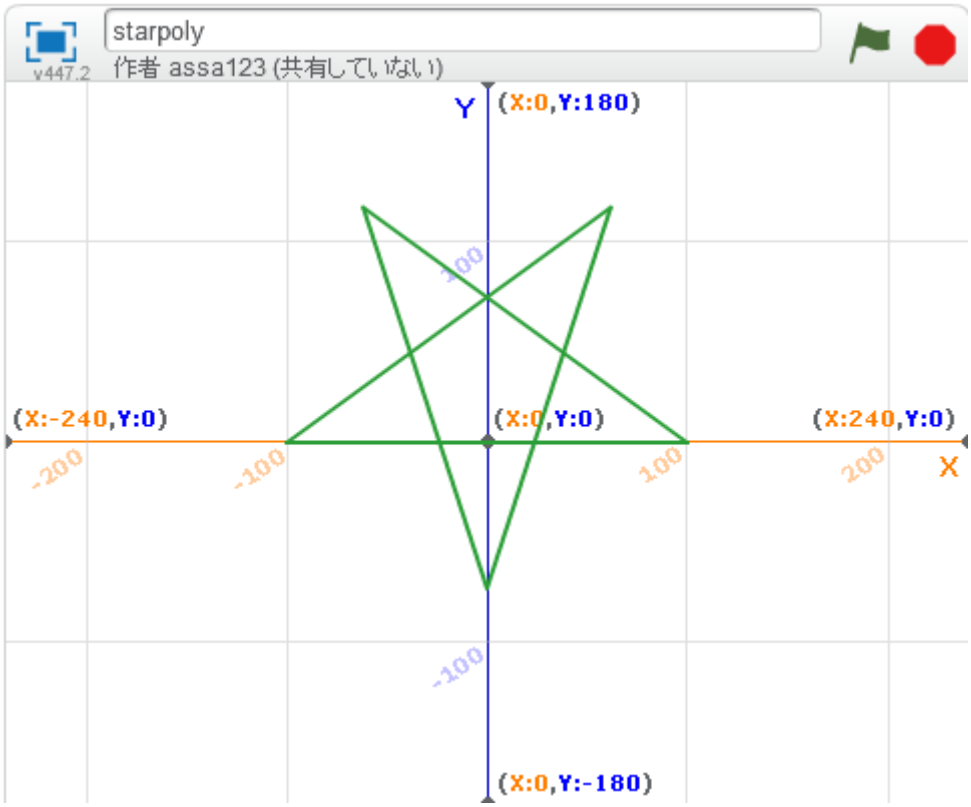
180-180/n

たとえば n が 5 なら

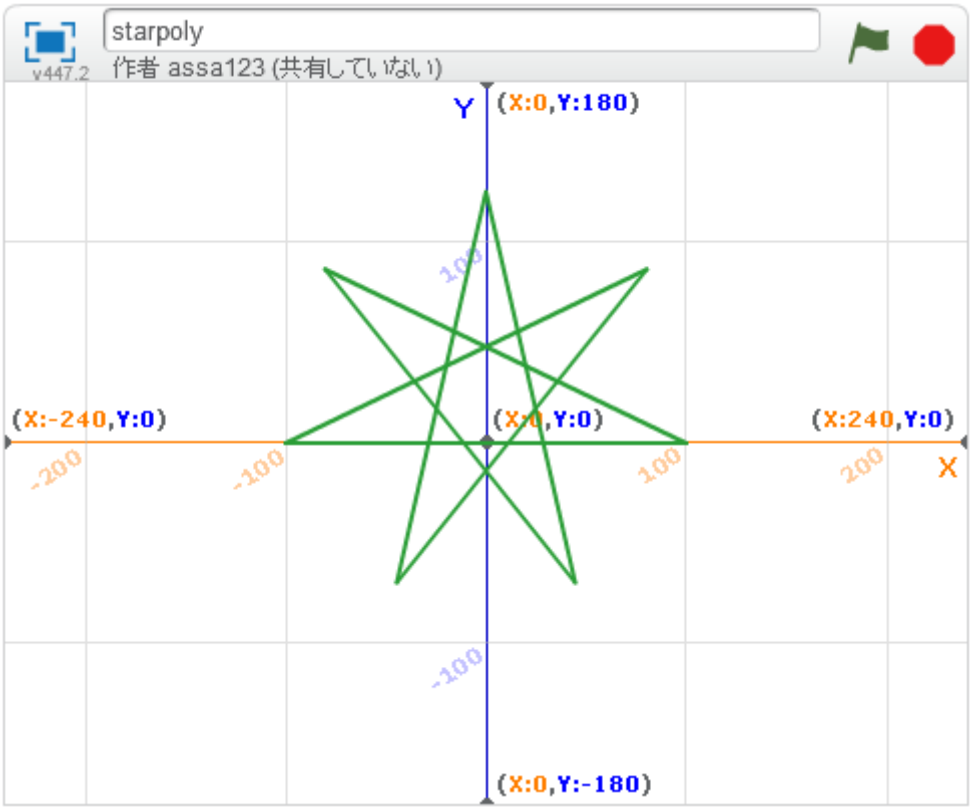
180-180/5=180-36=144

となります。


がクリックされたとき
init
setpoint -100 0
n を 5 にする
n 回繰り返す
move 200
180 - 180 / n 度回す
1 秒待つ


starpoly
v447.2
作者 assa123 (共有していない)
Y
(X:0,Y:180)
100
(X:-240,Y:0)
(X:0,Y:0)
(X:240,Y:0)
-200
-100
100
200
X
-100
(X:0,Y:-180)

・n=7

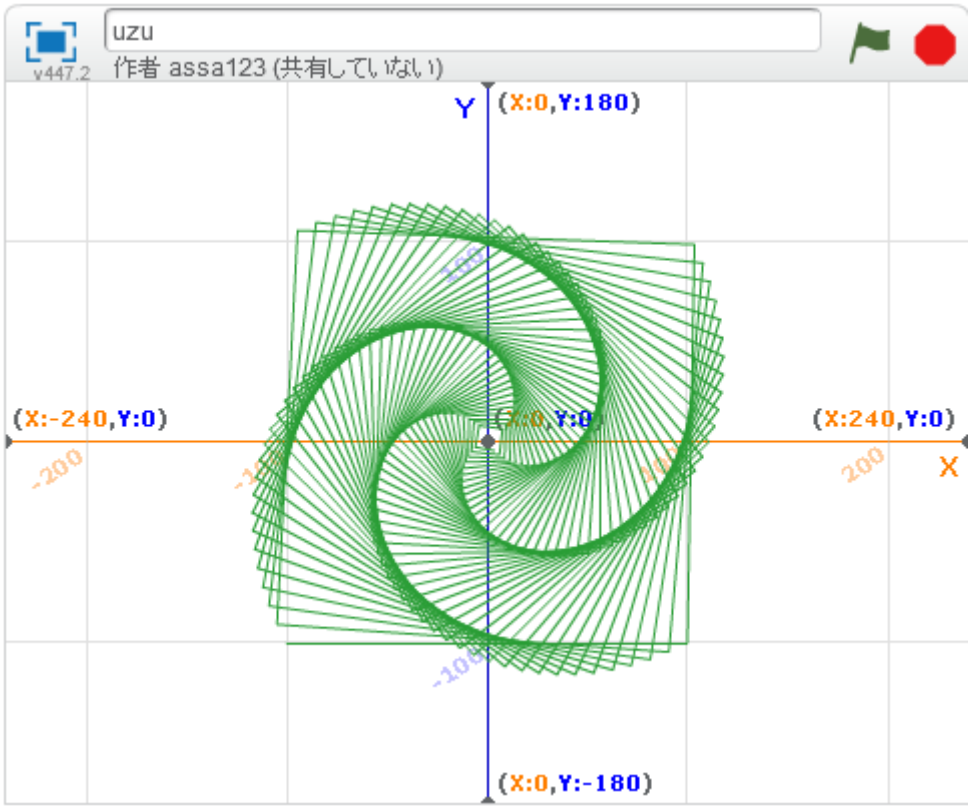


例題 8-1-4　渦巻き模様

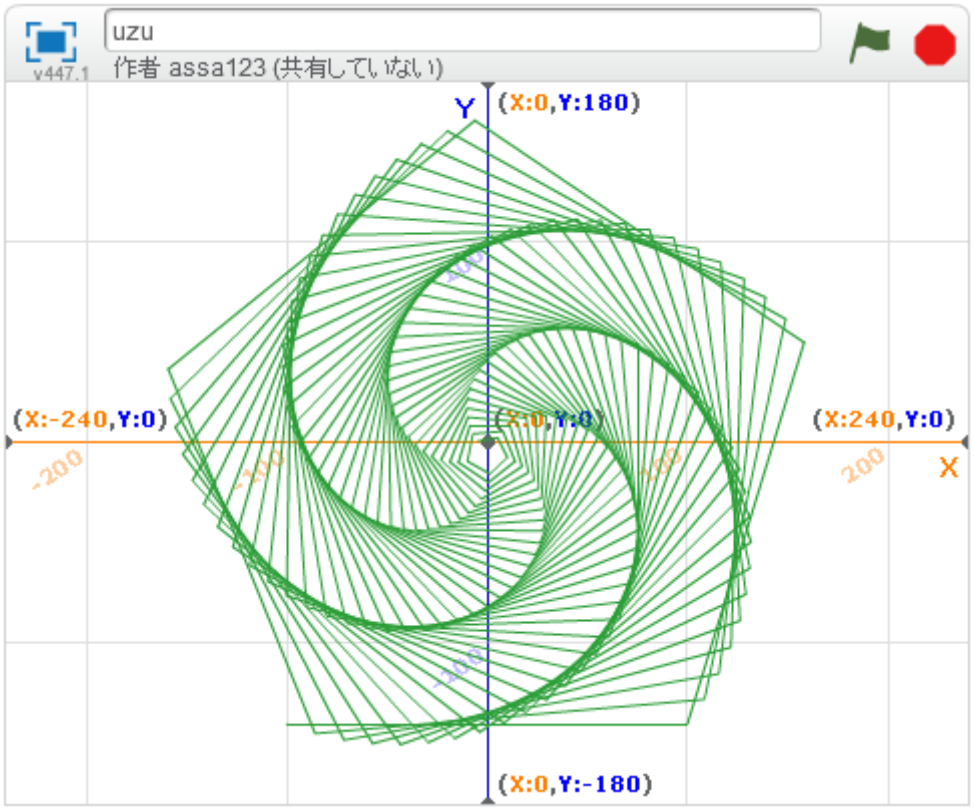
move と turn をくり返しながら、引く直線の長さを徐々に短くしていくと次のような渦巻き模様が得られます。turn する角度を angle、減少させていく長さを dec とし、直線の長さが「10」になるまで繰り返します。

・ペンの太さを 1 にします

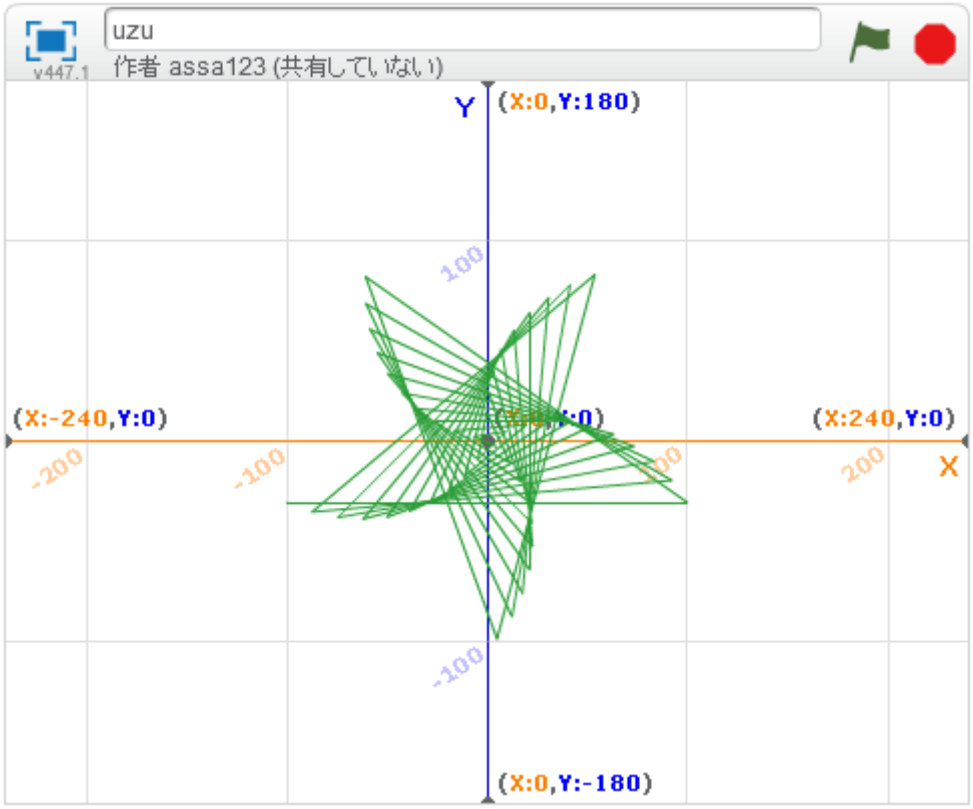

がクリックされたとき
init
setpoint -100 -100
隠す
leng を 200 にする
leng < 10 まで繰り返す
move leng
89 度回す
leng を -2 ずつ変える


uzu
v447.2
作者 assa123 (共有していない)
Y
(X:0,Y:180)
(X:-240,Y:0)
(X:0,Y:0)
(X:240,Y:0)
X
-200
200
(X:0,Y:-180)

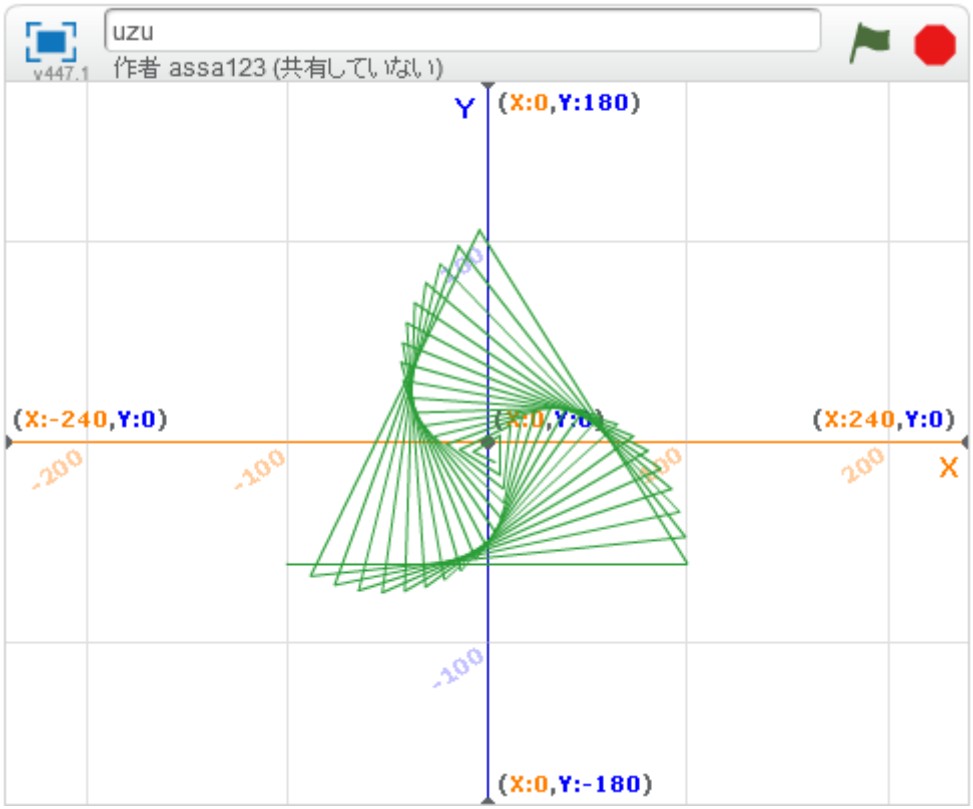
・開始点（-100,-140）、角度 73、きざみ 1



・開始点(-100,-30)、角度 145、きざみ 4



・開始点(-100,-60)、角度 122、きざみ 4



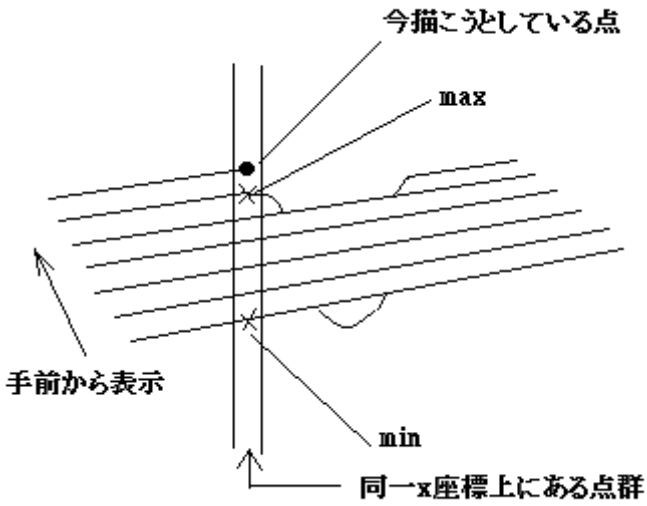
９章　３次元グラフィックス

私達の世界は３次元空間ですので、そこにある物体を紙やデイスプレイという２次元平面に作画するには、それなりの方法が必要になります。この章では３次元グラフィックスを行うためのアルゴリズムについて説明します。

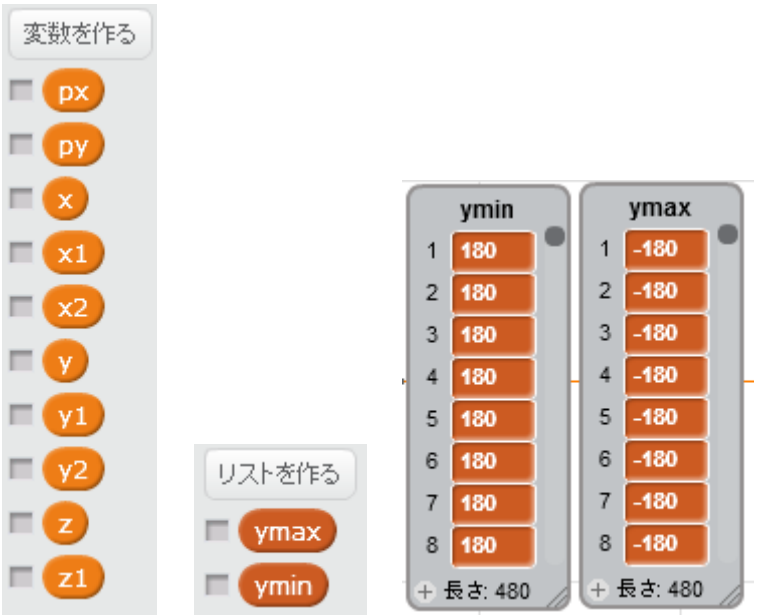
・3 次元座標変換（軸測投影）
・透視
・柱体モデル
・回転体モデル
・ワイヤーフレームモデル
・３次元関数
・陰線処理

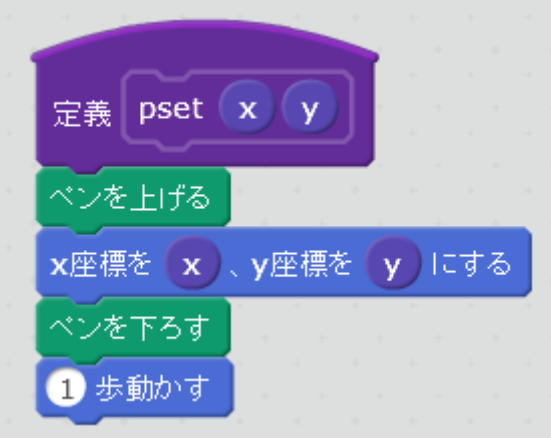
９－７　陰線処理

前方の面の後ろに隠れて見えない線を消すことを陰線処理と呼びます。陰線処理の方法は各種ありますが、ここでは max・min 法というきわめて単純な方法を説明します。
max・min 法では描く図形を必ず手前から描いて行きます。そして以後に描く点が、前に描いた点群（表示画面の同じ x 座標に位置する前の点群）の最大点（max）と最小点（min）の間（点群の内側）にあれば、その点を表示せず、逆に前に描いた点群の外にあれば、その点を表示します。

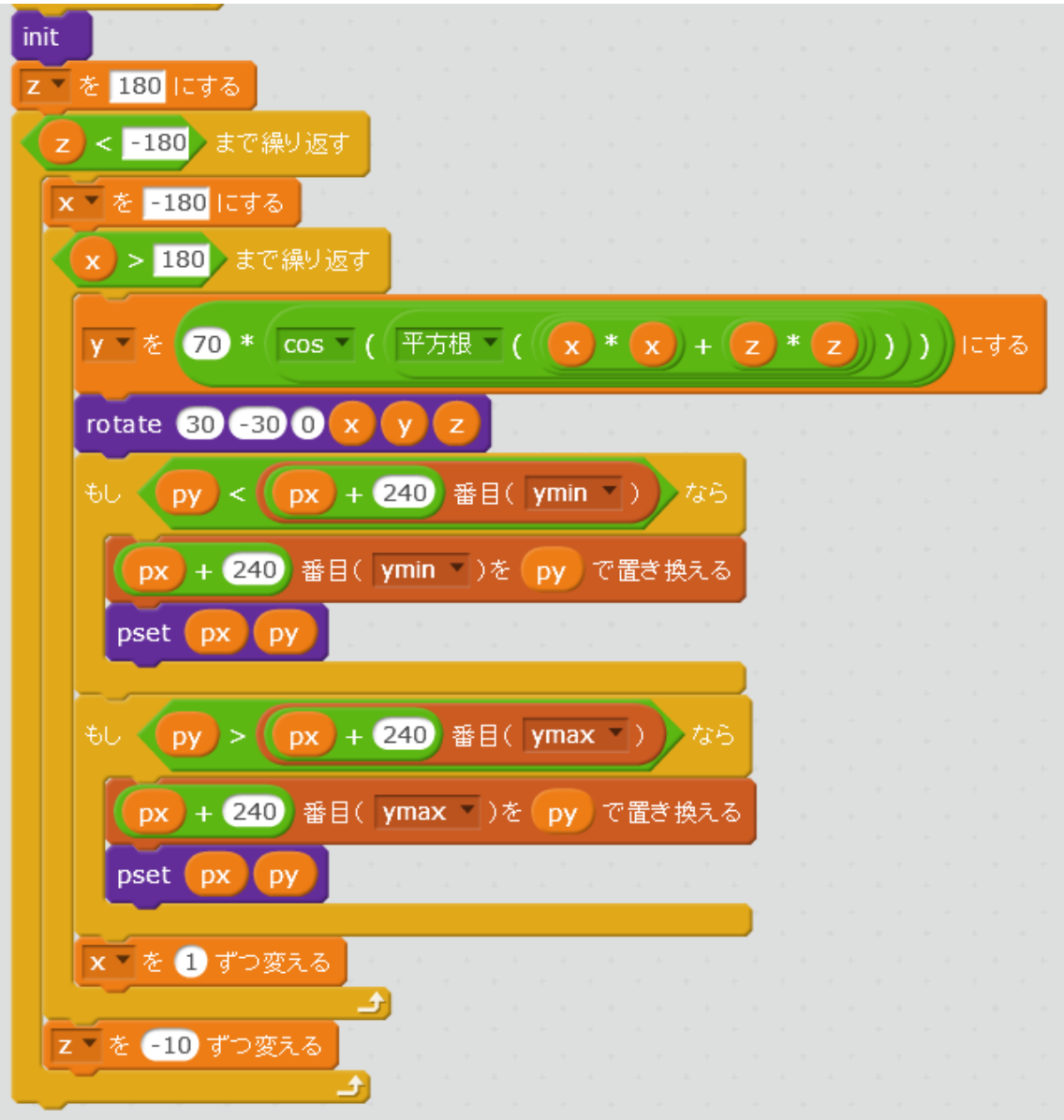


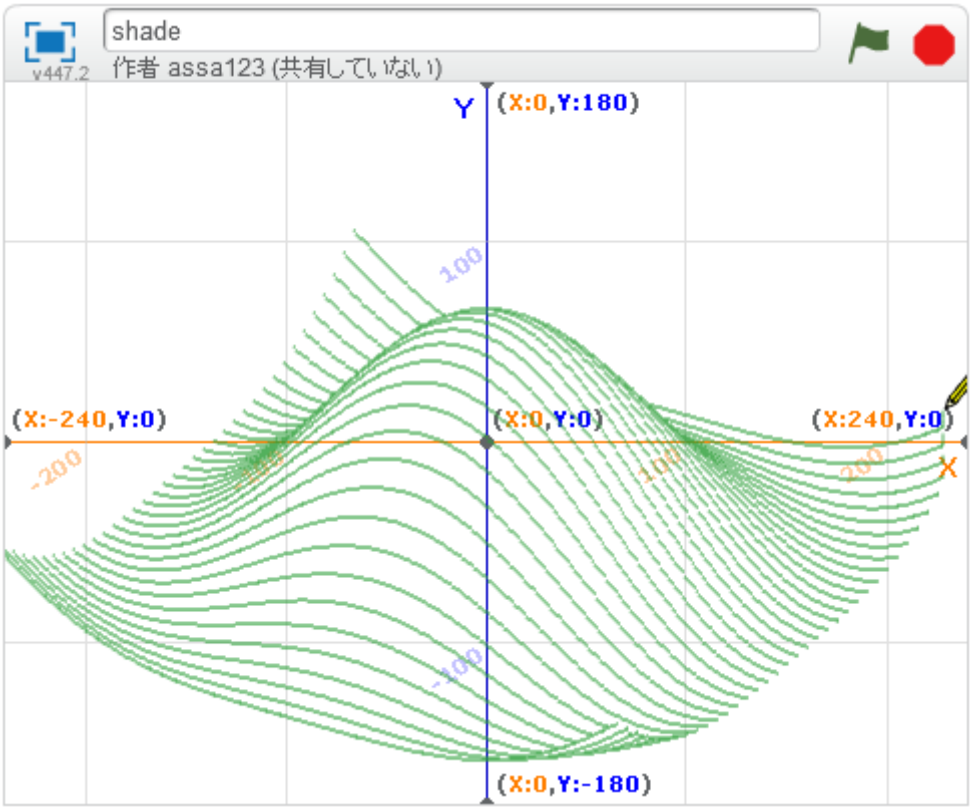
例題 9-7　陰線処理
$y=70\cos(\sqrt{x^2+z^2})$という３次元関数を陰線処理して表示します。



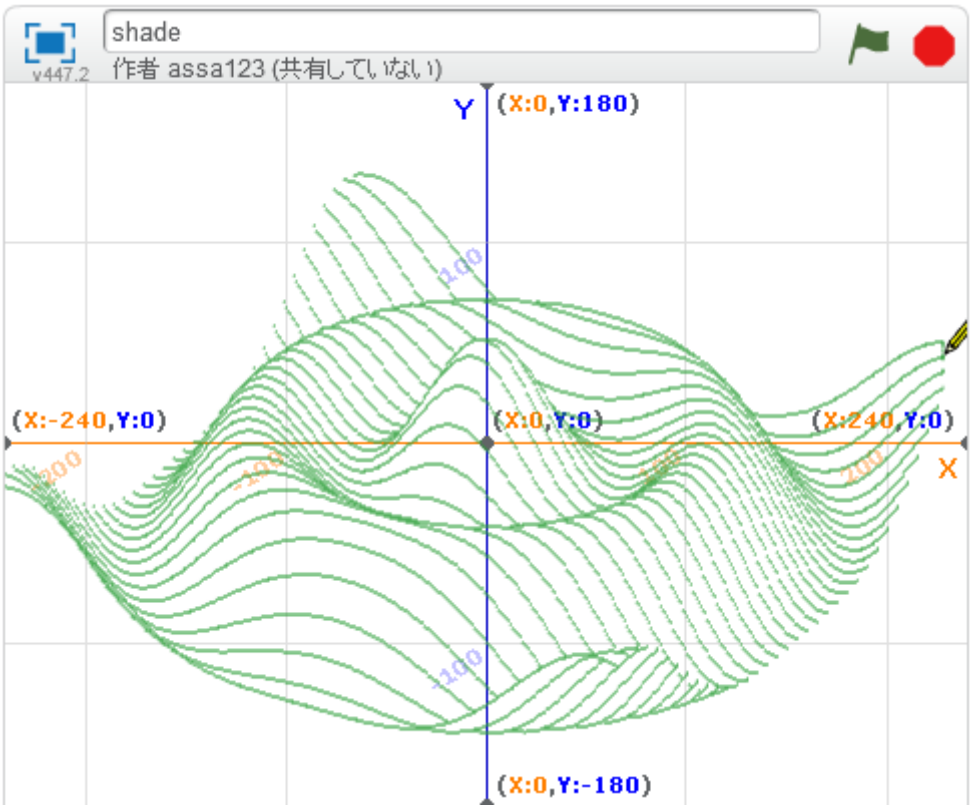

定義 pset x y
ペンを上げる
x座標を x 、y座標を y にする
ペンを下ろす
1 歩動かす


がクリックされたとき
ymin の長さ 回繰り返す
1 番目を ymin から削除する
1 番目を ymax から削除する
480 回繰り返す
180 を ymin に追加する
-180 を ymax に追加する

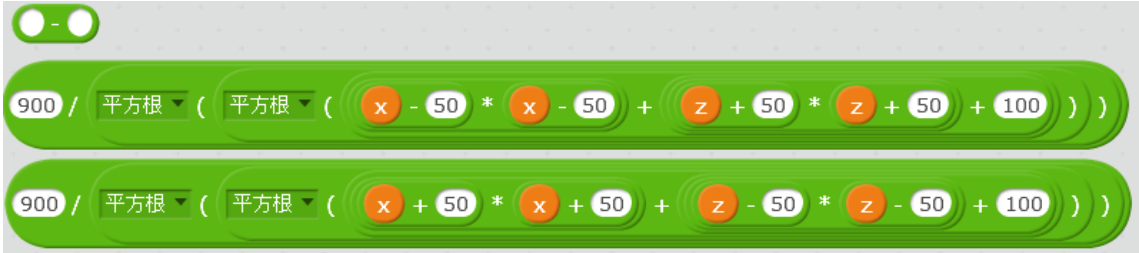

init
z を 180 にする
z < -180 まで繰り返す
x を -180 にする
x > 180 まで繰り返す
y を 70 * cos (平方根 (x * x + z * z)) にする
rotate 30 -30 0 x y z
もし py < px + 240 番目(ymin) なら
px + 240 番目(ymin)を py で置き換える
pset px py
もし py > px + 240 番目(ymax) なら
px + 240 番目(ymax)を py で置き換える
pset px py
x を 1 ずつ変える
z を -10 ずつ変える

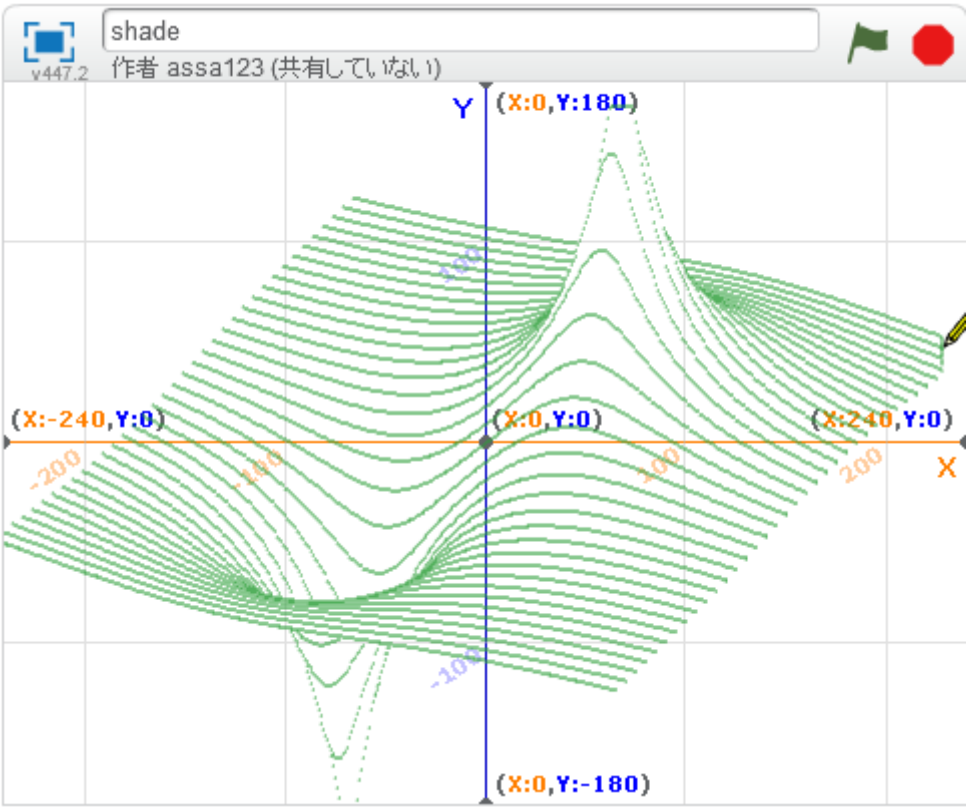

shade
v447.2
作者 assa123 (共有していない)
Y
(X:0,Y:180)
100
(X:-240,Y:0)
(X:0,Y:0)
(X:240,Y:0)
-200
-100
100
200
X
-100
(X:0,Y:-180)

・y=30(cos(√x²+z²)+cos(3√x²+z²))



・$y=900/\sqrt{\sqrt{(x-50)^2+(z+50)^2}+100}-900/\sqrt{\sqrt{(x+50)^2+(z-50)^2}+100}$





１０章　リカーシブ・グラフィックス

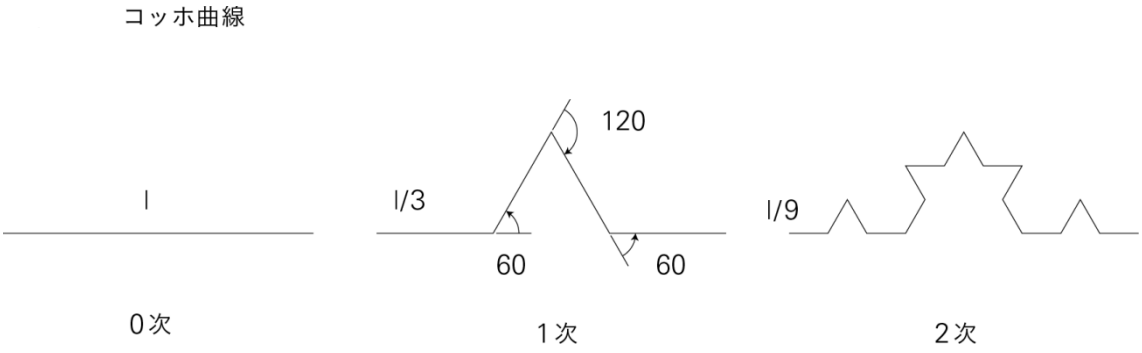
リカーシブ・グラフィックスはグラフィックスの世界を解析的に表現せずに、再帰的に表現しようとするものです。再帰を使うと自然に近い図形（入り組んだ海岸線や樹木）が、いとも簡単に表現できます。

・コッホ曲線
・クロスステッチ
・樹木曲線Ⅰ
・樹木曲線Ⅱ
・C 曲線
・ドラゴン曲線
・ヒルベルト曲線
・シェルピンスキー曲線

１０－１　コッホ曲線

コッホ曲線は、数学者のコッホにより発見されたものです。コッホ曲線は以下のように定義されています。

0 次のコッホ曲線は長さ l の直線である。1 次のコッホ曲線は、1 辺の長さが l/3 の大きさの正三角形状のでっぱりを出す。2 次のコッホ曲線は、1 次のコッホ曲線の各辺（4 つ）に対し、1 辺の長さが l/9 の大きさの正三角形状のでっぱりを出す。

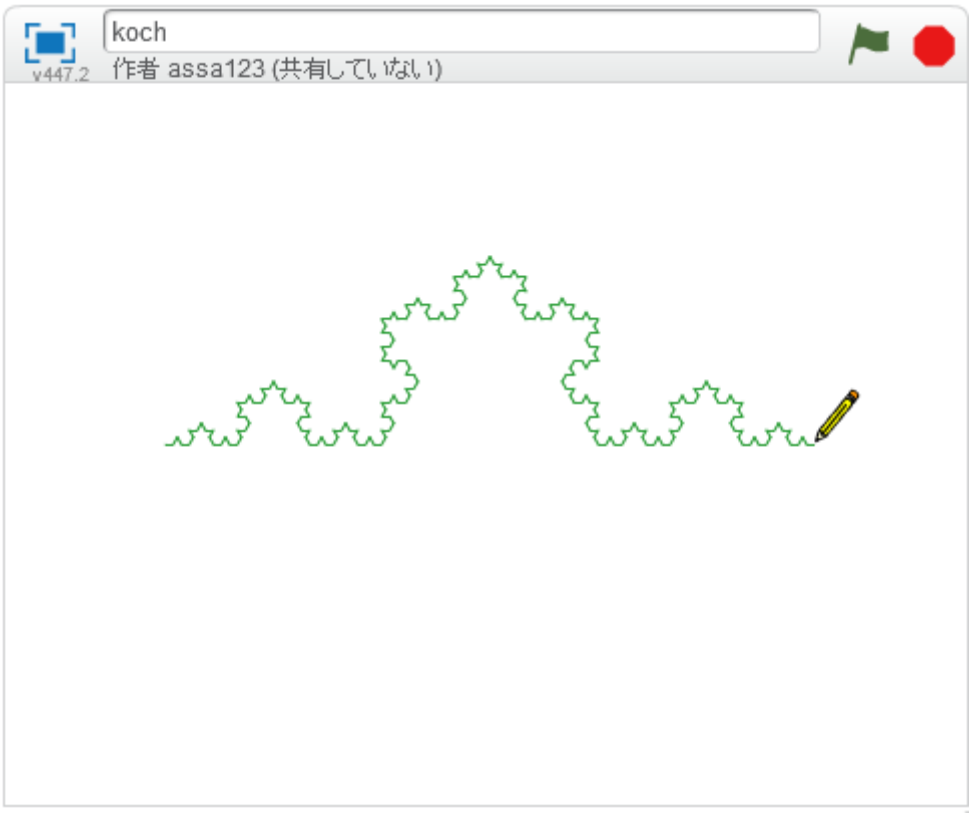


例題 10-1-1　コッホ曲線

n 次のコッホ曲線を描く手順は以下です。

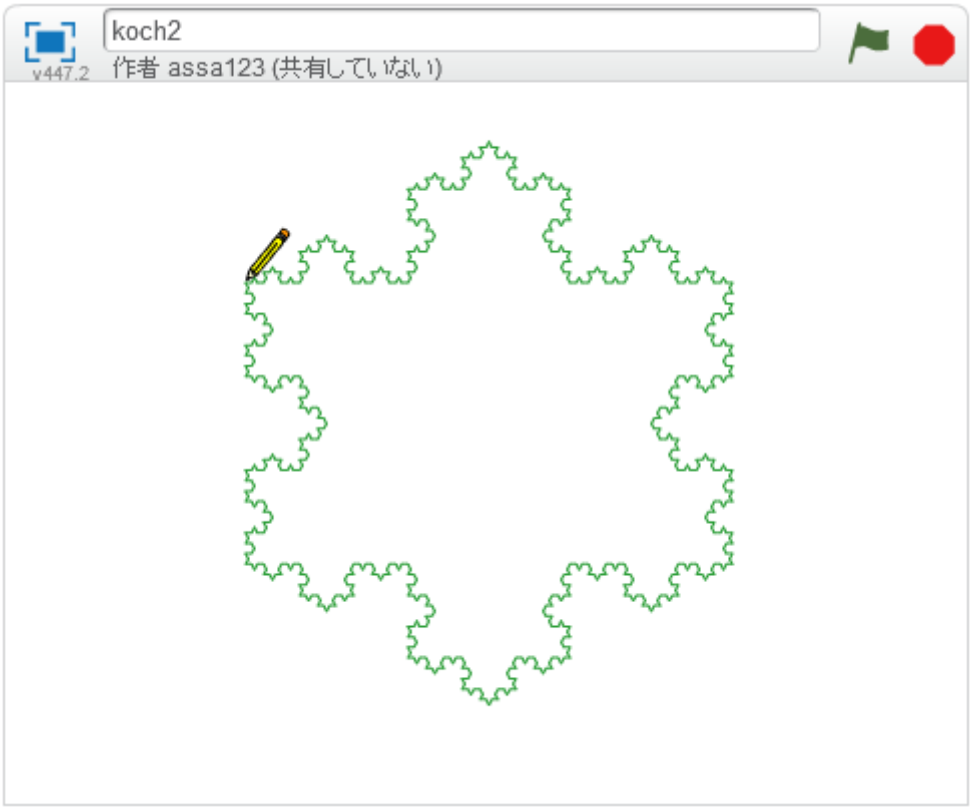
①n-1 次のコッホ曲線を 1 つ描く。
②向きを 60°変えて n-1 次のコッホ曲線を 1 つ描く。
③向きを-120°変えて n-1 次のコッホ曲線を 1 つ描く。
④向きを 60°変えて n-1 次のコッホ曲線を 1 つ描く。


定義 koch n leng
もし n = 0 なら
move leng
でなければ
koch n - 1 leng
60 度回す
koch n - 1 leng
-120 度回す
koch n - 1 leng
60 度回す
koch n - 1 leng



練習問題 10-1-2　コッホ島

4 次のコッホ曲線を「-120°」回転して、3 個描きます。描かれた図形をコッホ島と呼びます。


koch2
v447.2
作者 assa123 (共有していない)

１１章　パズル・ゲーム

子ども達がプログラミングで一番作りたいものがゲームです。ゲーム機のゲームのようなものは簡単にはできませんので、この章ではパズル的な簡単にできるゲームを紹介します。

・魔方陣
・戦略を持つじゃんけん
・リバーシー
・移動板パズル

１１－３　リバーシー

8×8 のマスに 3 種類のイメージを配置します。

green　緑（石を置いていない状態）のイメージ

white　白を置いた状態のイメージ

black　黒を置いた状態のイメージ

例題 11-3-1　石を置けるか調べるブロック ok

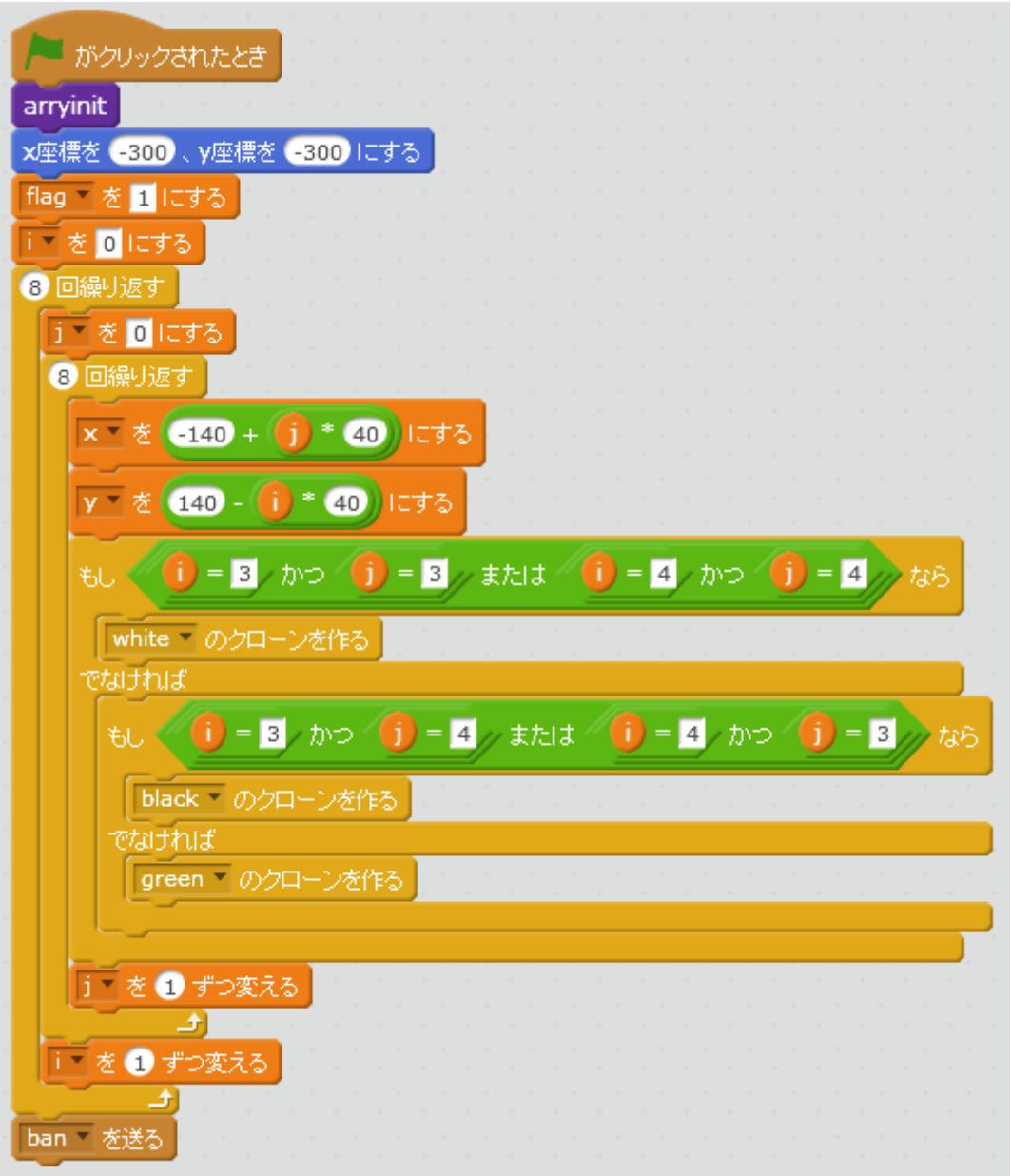
■盤面の情報を配列に置く

2 次元配列 m[i][j]に盤面の(i、j)位置のマスの情報（0：緑、1:白、2:黒）を与えます。2 次元配列 m[ ][ ]を初期化するユーザーブロックを arrayinit とします。盤面の初期状態は(4、4)と(5、5)位置が白、(4、5)と(5、6)位置が黒、その他は緑とします。

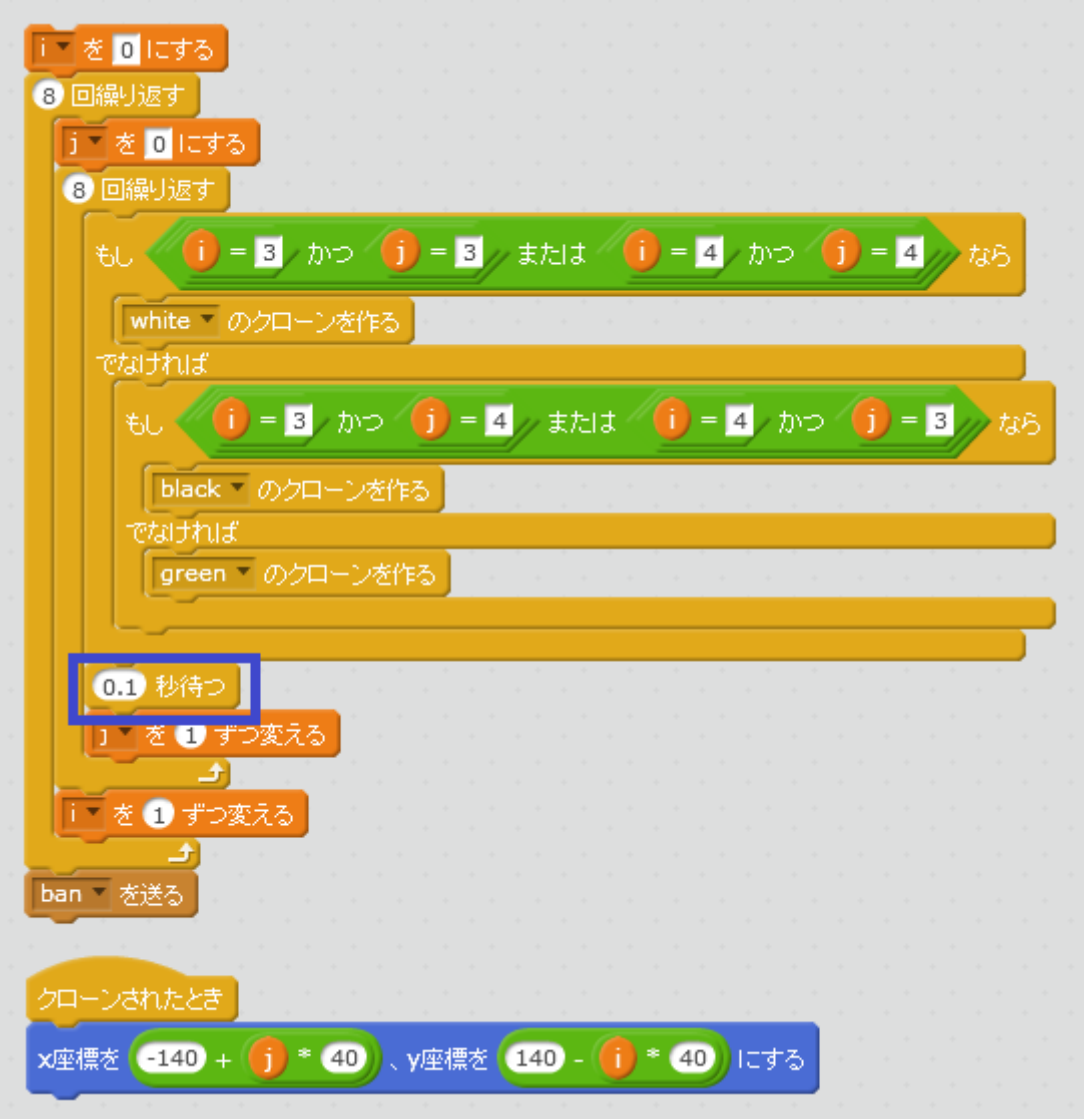
配列要素は 8×8 で良いのですが、ok ユーザブロックと reverse ユーザーブロックでマスの要素を超えた参照を行うのでそれに備えて 10×10 の要素とします。盤面に相当する要素は 1～8 で行います。盤面の外枠に相当する要素 0 と要素 9 の内容は 0 とします。

		j									
		0	1	2	3	4	5	6	7	8	9
	0	0	0	0	0	0	0	0	0	0	0
	1	0									0
	2	0									0
	3	0									0
i	4	0				1	2				0
	5	0				2	1				0
	6	0									0
	7	0									0
	8	0									0
	9	0	0	0	0	0	0	0	0	0	0

・green のスクリプトブロック



「注」上のプログラムではループ内で i と j の値から変数 x,y にクローンの表示位置を計算しています。もしこれをクローンの生成側で「-140+j*40」のように指定すると、クローンを作る動作の前にループ変数の j の値は＋１されてしまうので、「0.1 秒待つ」を入れて、クローンの生成が終わるのを待って、j の値が変わるようにします。変数 x,y への代入操作が「0.1 秒待つ」の役割をしていることになります。「６－５　迷路」参照。

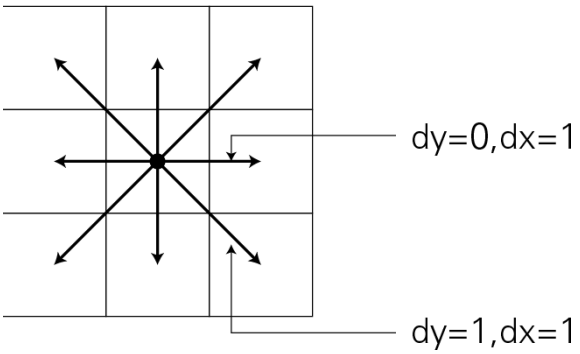



定義 arryinit
m の長さ 回繰り返す
1 番目を m から削除する
10 * 10 回繰り返す
0 を m に追加する
4 * 10 + 4 + 1 番目(m)を 1 で置き換える
5 * 10 + 5 + 1 番目(m)を 1 で置き換える
4 * 10 + 5 + 1 番目(m)を 2 で置き換える
5 * 10 + 4 + 1 番目(m)を 2 で置き換える
クローンされたとき
x座標を x 、y座標を y にする

■ok ユーザブロック

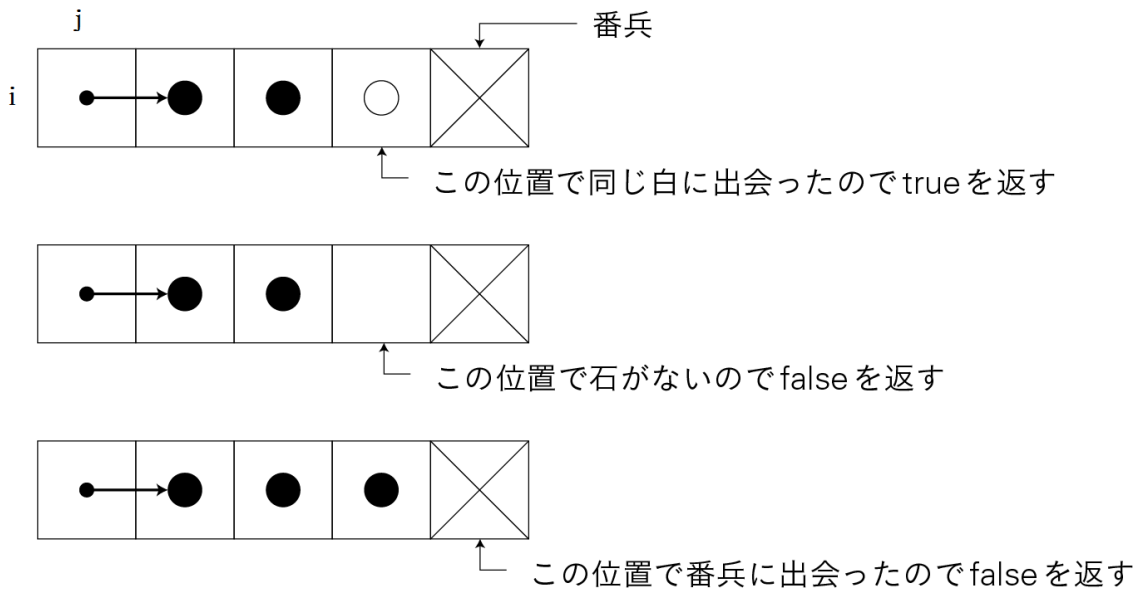
クリックした位置に石が置ければ okflag=1、置けなければ okflag=0 を設定する ok ユーザーブロックを作ります。クリックした位置を i、j とすると。そこを中心に 8 方向でチェックを行います。i、j 位置から dy、dx で示す値を加えることで i、j 位置を進めます。たとえば dy=0、dx=1 ならマスの右方向への移動となり、dy=1、dx=1 ならマスの右斜め下方向への移動となります。8 方向の dy、dx の値は for の二重ループで-1、0、1 と変化させます。組み合わせは 9 通りできますがが dy=0、dx=0 はどこにも進めないので、除外すると 8 通りとなります。

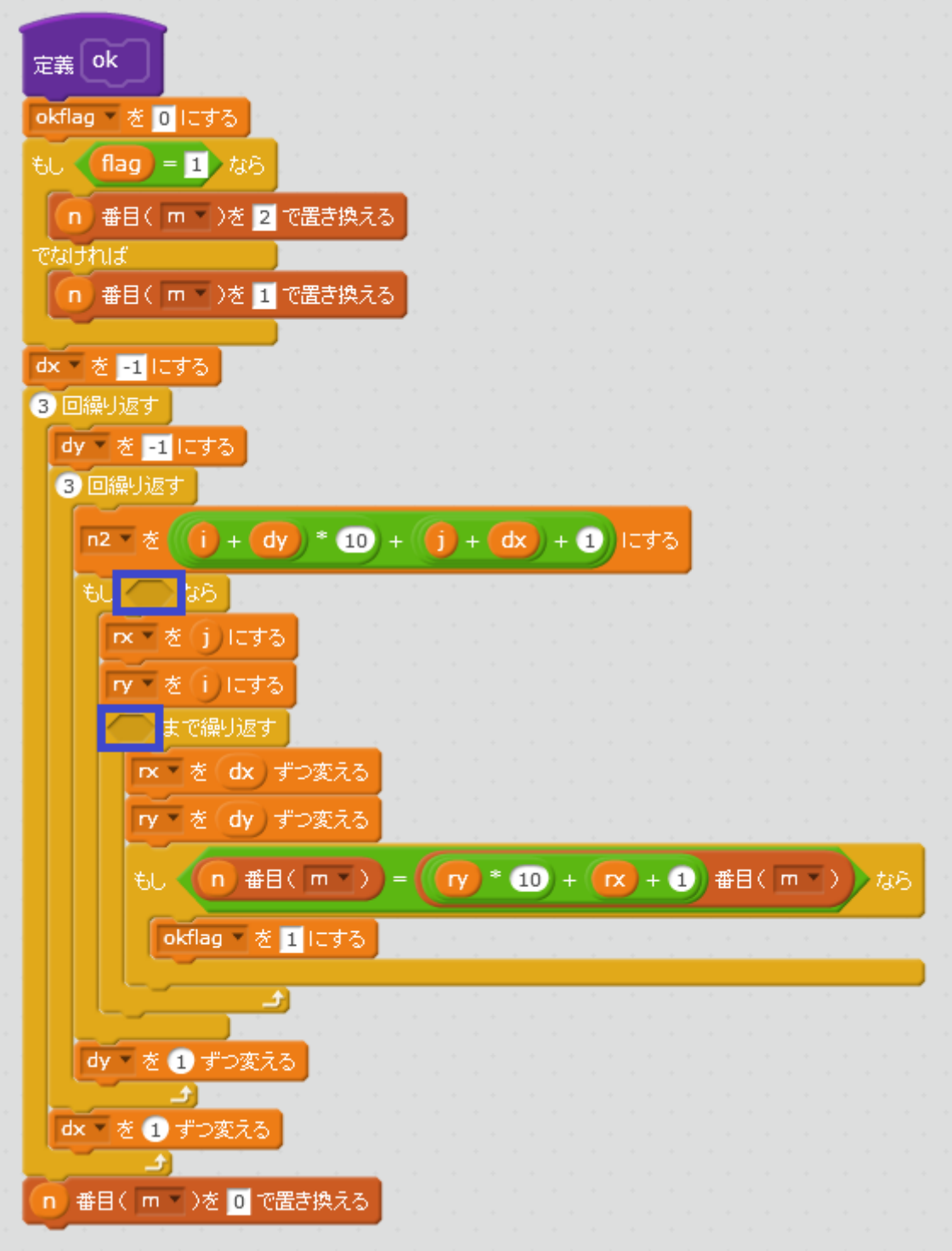
８方向の移動方向



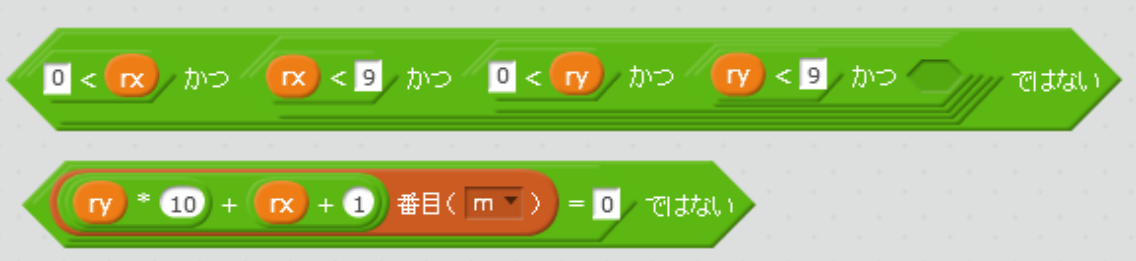
i、j 位置に対応する配列にその手番の石を置いたという情報を格納します。for の二重ループ内で、位置を進める操作を行うのは dy=0、dx=0 でない場合でかつ進む方向の隣が同じ色でない場合です。
位置を進めるのは端になるまでか緑になるまでです。マスを進めるときの変数として ry、rx を使います。i、j 位置と ry、rx 位置のマスの石が同じ色なら okflag=1 を設定します。okflag=1 になる条件がなく for の 2 重ループを完了した場合は i、j 位置に置けないということなので okflag=0 を設定し、i、j 位置に仮に置いたものを取り去ります。以下に i、y 位置に白が置けるか右方向にチェックする場合を示しました。

右方向へのチェック




定義 ok
okflag を 0 にする
もし flag = 1 なら
n 番目(m)を 2 で置き換える
でなければ
n 番目(m)を 1 で置き換える
dx を -1 にする
3 回繰り返す
dy を -1 にする
3 回繰り返す
n2 を i + dy * 10 + j + dx + 1 にする
もし なら
rx を j にする
ry を i にする
まで繰り返す
rx を dx ずつ変える
ry を dy ずつ変える
もし n 番目(m) = ry * 10 + rx + 1 番目(m) なら
okflag を 1 にする
dy を 1 ずつ変える
dx を 1 ずつ変える
n 番目(m)を 0 で置き換える

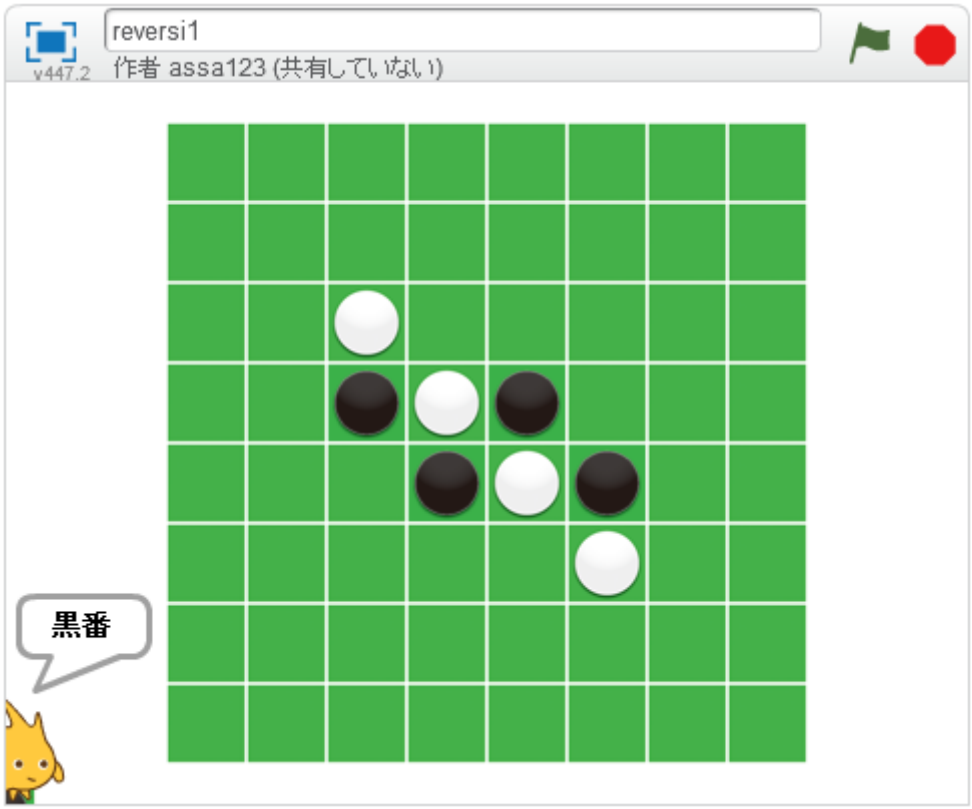
・青枠には以下が入ります。



・white のスクリプトブロック

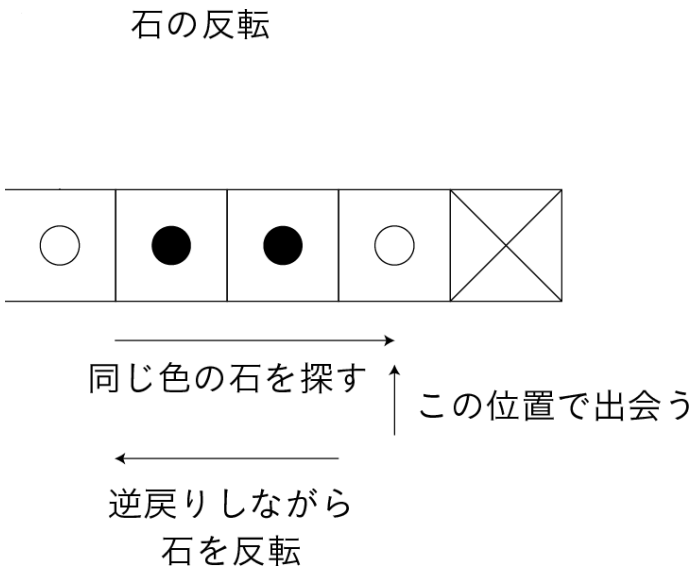
・black のスクリプトブロック

・Gobo のスクリプトブロック

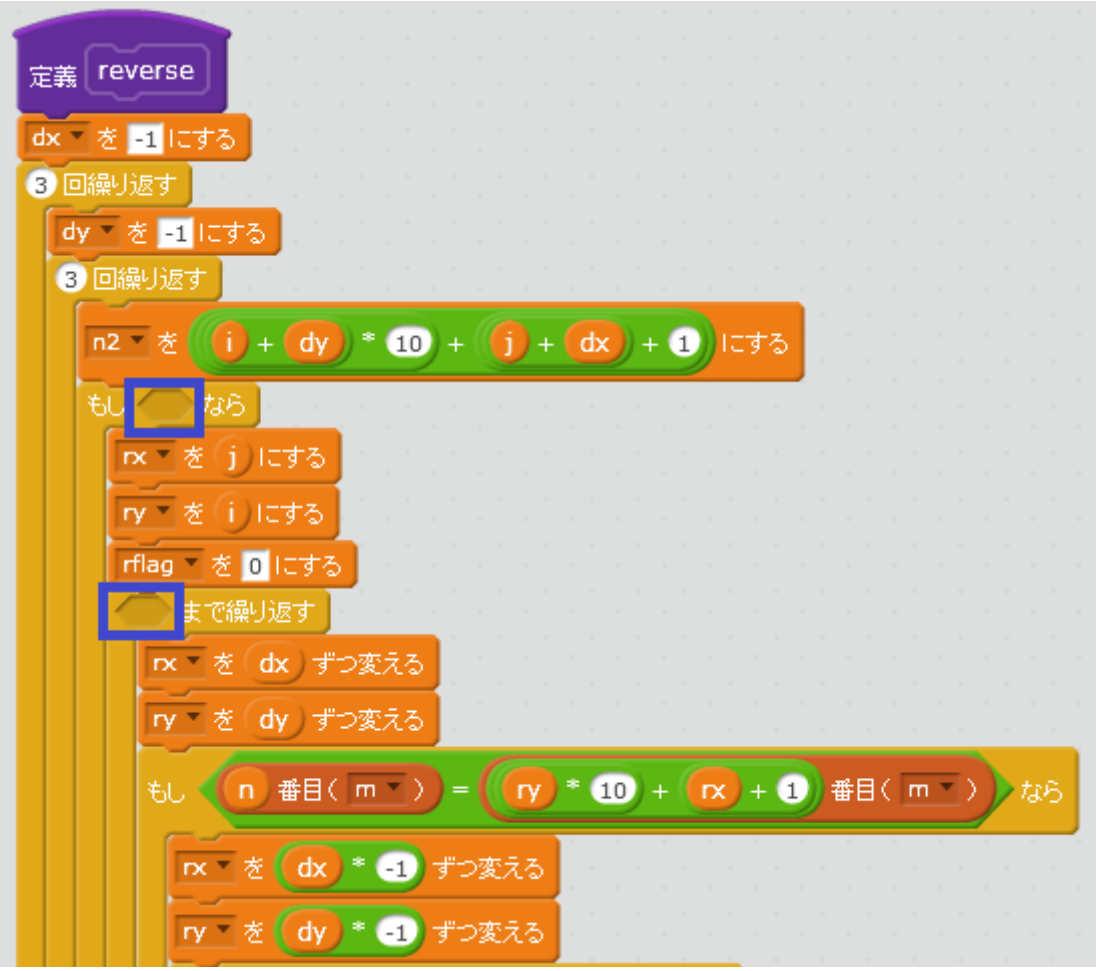


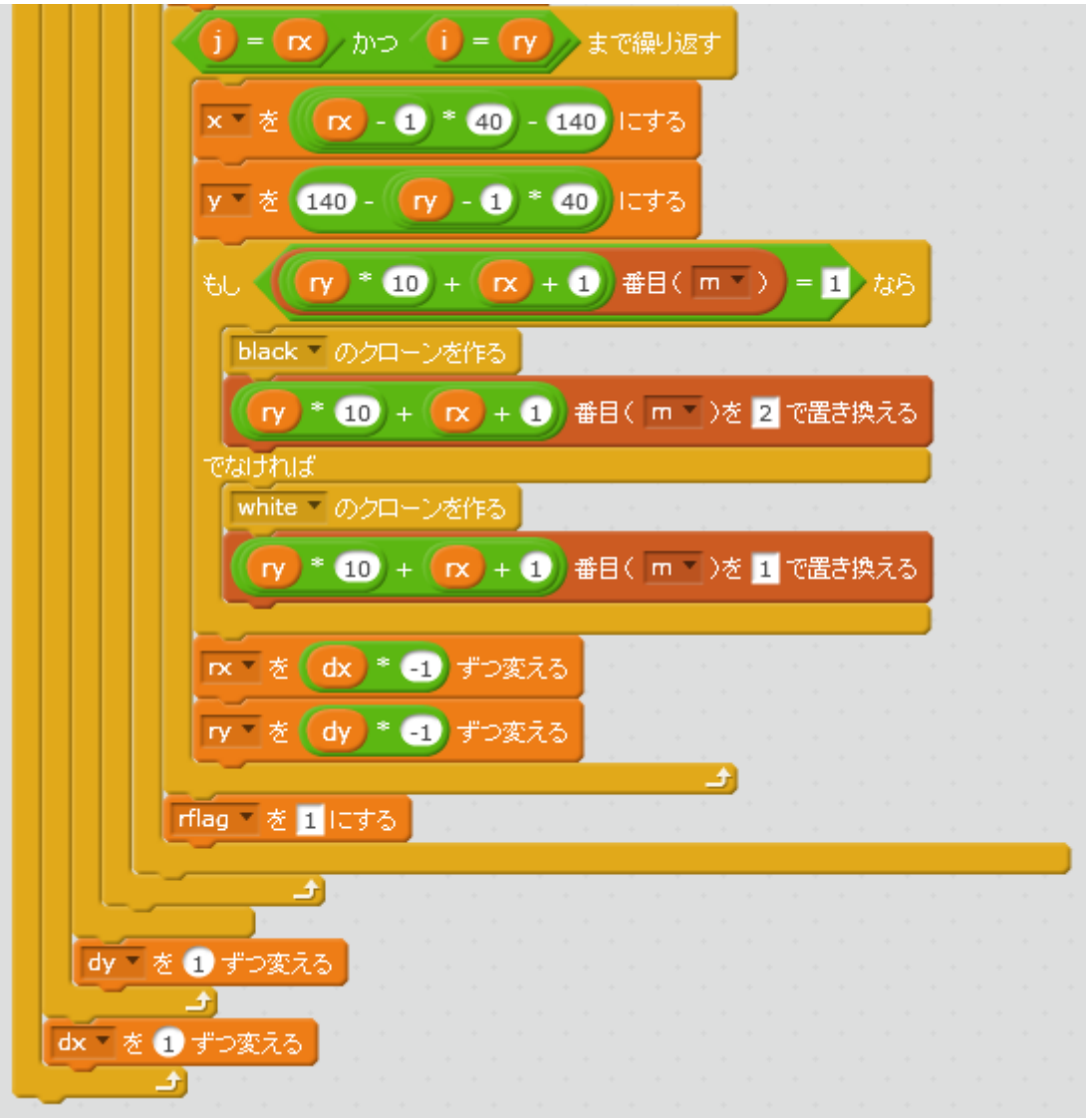
例題 11-3-2　自動反転 reverse

はさんだ石を自動的に反転する reverse ユーザーブロックを作ります。先に作成した ok ユーザーブロックと同様な方法で i、j 位置を中心に 8 方向を検査します。i、j 位置に置いた石と同じ色の石が見つかったら、そこから逆戻りしながら石を反転する処理を追加します。



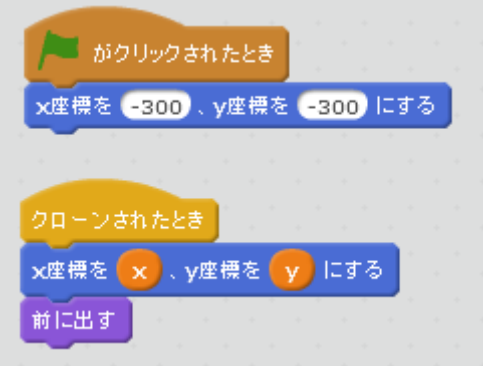
・green のスクリプトブロック


定義 reverse
dx を -1 にする
3 回繰り返す
dy を -1 にする
3 回繰り返す
n2 を i + dy * 10 + j + dx + 1 にする
もし なら
rx を j にする
ry を i にする
rflag を 0 にする
まで繰り返す
rx を dx ずつ変える
ry を dy ずつ変える
もし n 番目(m) = ry * 10 + rx + 1 番目(m) なら
rx を dx * -1 ずつ変える
ry を dy * -1 ずつ変える

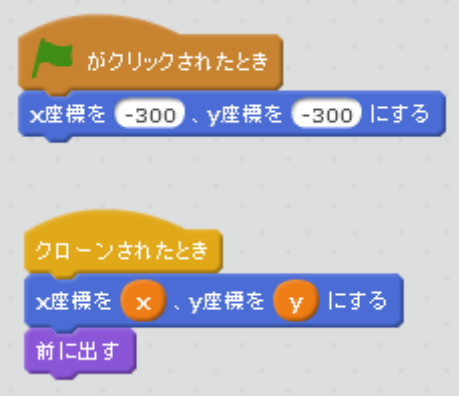


・青枠には以下が入ります。

・white のスクリプトブロック



・black のスクリプトブロック



・Gobo のスクリプトブロック

reversi1 と同じ

例題 11-3-3　コンピュータと対戦（手はランダム）

ここではコンピュータは何も考えずにランダムに白石を置く conputer ユーザーブロックを作ります。コンピュータは 1～8 の乱数を 2 組発生し、その値を i、j 位置とし、ok(i,j)でそこに石を置けるか判定します。

・green のスクリプトブロック

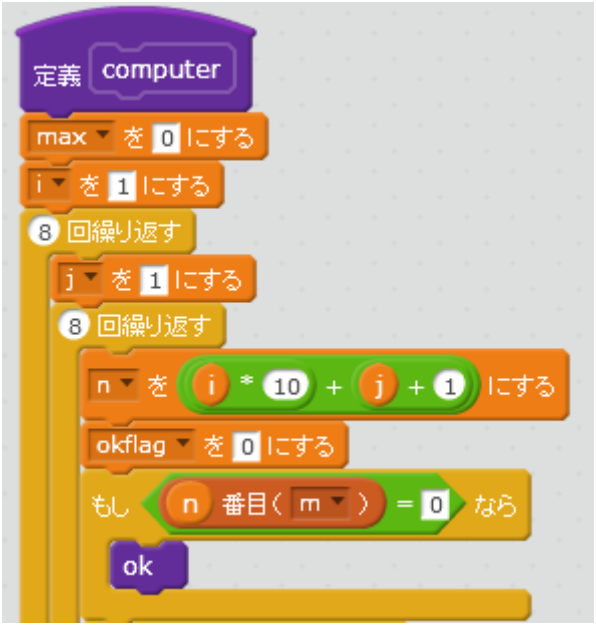
・青枠が変更した部分です。

・white,black,Gobo のスクリプトブロック

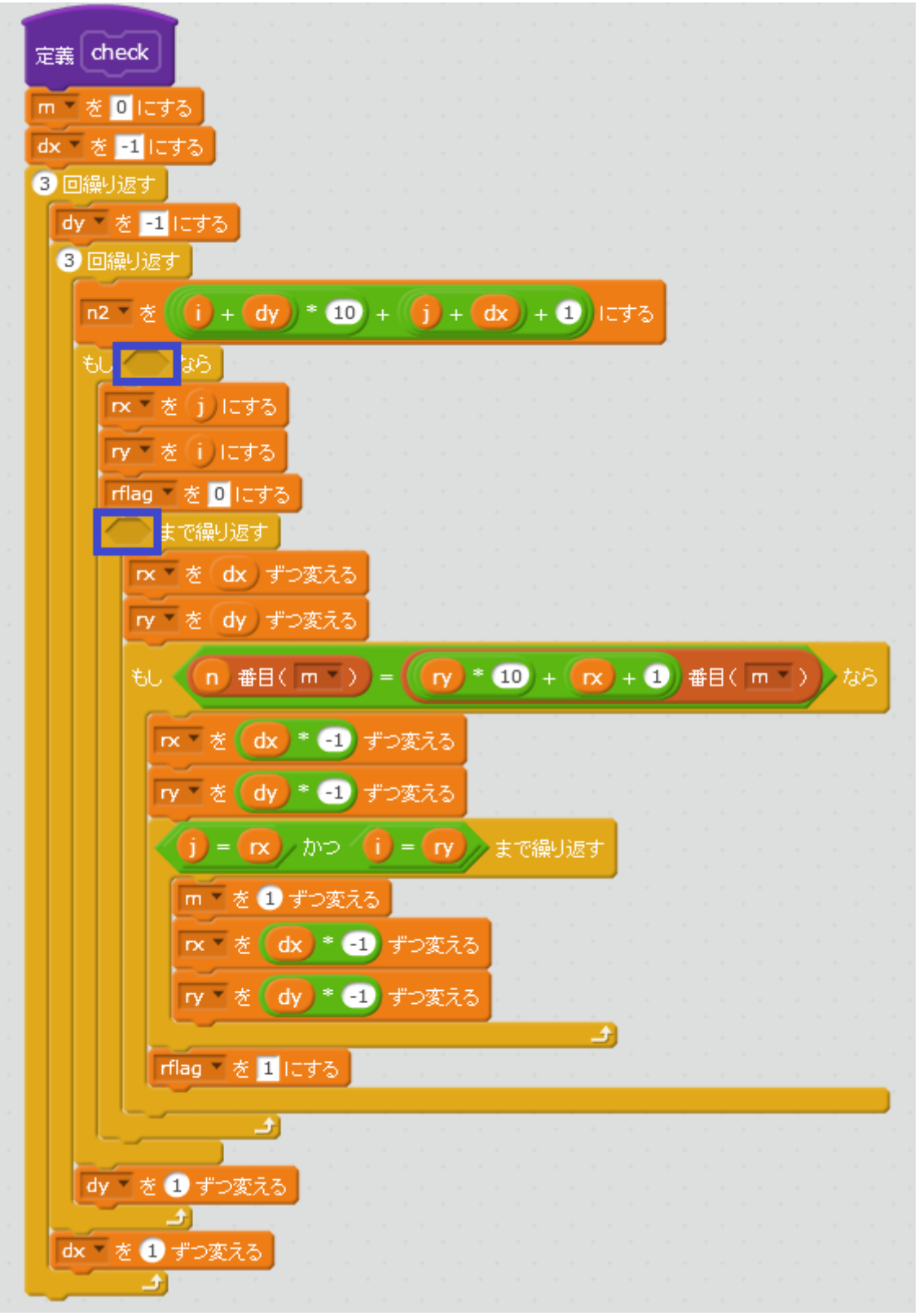
reversi2 と同じ

例題 11-3-4　コンピュータが手を考える

左上隅から始めそこに白石（コンピュータの手）が置けたら、黒石をひっくり返せる枚数を求める check ユーザーブロックを作ります。reverse ユーザーブロックを参考にします。一番ひっくり返す枚数が多い位置をコンピュータの手にします。




もし okflag = 1 なら
n 番目(m)を 1 で置き換える
check
もし m > max なら
max を m にする
ii を i にする
jj を j にする
n 番目(m)を 0 で置き換える
j を 1 ずつ変える
i を 1 ずつ変える
i を ii にする
j を jj にする
もし max = 0 ではない なら
n を i * 10 + j + 1 にする
x を j - 1 * 40 - 140 にする
y を 140 - i - 1 * 40 にする
white のクローンを作る
n 番目(m)を 1 で置き換える
reverse
flag を flag * -1 にする
ban を送る


定義 check
m を 0 にする
dx を -1 にする
3 回繰り返す
dy を -1 にする
3 回繰り返す
n2 を i + dy * 10 + j + dx + 1 にする
もし なら
rx を j にする
ry を i にする
rflag を 0 にする
まで繰り返す
rx を dx ずつ変える
ry を dy ずつ変える
もし n 番目(m) = ry * 10 + rx + 1 番目(m) なら
rx を dx * -1 ずつ変える
ry を dy * -1 ずつ変える
j = rx かつ i = ry まで繰り返す
m を 1 ずつ変える
rx を dx * -1 ずつ変える
ry を dy * -1 ずつ変える
rflag を 1 にする
dy を 1 ずつ変える
dx を 1 ずつ変える

・青枠には以下が入ります。

・white,black,Gobo のスクリプトブロック
reversi2 と同じ

例題 11-3-5　４隅を優先

黒石をひっくり返せる枚数の最大を優先する前に、四隅に置けたらそちらを優先するようにします。


i を ii にする
j を jj にする
もし max = 0 ではない なら
n を i * 10 + j + 1 にする
x を j - 1 * 40 - 140 にする
y を 140 - i - 1 * 40 にする
white のクローンを作る
n 番目(m)を 1 で置き換える
reverse
flag を flag * -1 にする
ban を送る


または または または
i = 1 かつ j = 1
i = 8 かつ j = 1
i = 1 かつ j = 8
i = 8 かつ j = 8

１２章　アラカルト

今までの各章で説明したアルゴリズムのカテゴリに属さない雑多なアルゴリズムを紹介します。

・羅針盤
・ドットアート
・万年歴
・３拓クイズ
・電卓
・お絵描きツール

１２－１　羅針盤

羅針盤の上の針をマウス位置の方向を指すように回転させます。

例題 12-1　羅針盤の針を回す

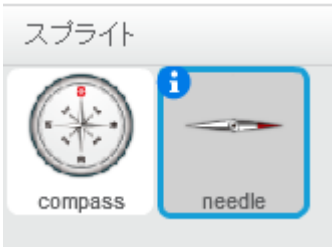
羅針盤 compass と針 needle を画面中央に配置します。針の角度 angle はマウス位置(x,y)からアークタンジェントを使って「atan(y/x)」で求め、 

で回転します。

なお、Scratch の 

の値と実際の角度の違いが以下のようにあります。従って angle° に向ける場合は「90-angle」とします。

90（右）	0
0(上)	90
-90（左）	180
180（下）	270




がクリックされたとき
x座標を 0 、y座標を 0 にする
ずっと
angle を 90 - atan (マウスのy座標 / マウスのx座標) にする
もし マウスのx座標 < 0 なら
angle を angle + 180 にする
angle 度に向ける


compass
v447.2
作者 assa123 (共有していない)
angle 29.153789
北
北西
北東
西
東
南西
南東
南

著者略歴

河西 朝雄（かさいあさお）

山梨大学工学部電子工学科卒（1974 年）。長野県岡谷工業高等学校情報技術科教諭、長野県松本工業高等学校電子工業科教諭を経て、現在は「カサイ．ソフトウエアラボ」代表。

「主な著書」

「入門ソフトウエアシリーズ C 言語」、「同シリーズ Java 言語」、「同シリーズ C++」、「入門新世代言語シリーズ VisualBasic4.0」、「同シリーズ Delphi2.0」、「やさしいホームページの作り方シリーズ HTML」、「同シリーズ JavaScript」、「同シリーズ HTML 機能引きテクニック編」、「同シリーズホームページのすべてが分かる事典」、「同シリーズ i モード対応 HTML と CGI」、「同シリーズ i モード対応 Java で作る i アプリ」、「同シリーズ VRML2.0」、「チュートリアル式言語入門 VisualBasic.NET」、「はじめての VisualC#. NET」、「C 言語用語辞典」ほか（以上ナツメ社）

「構造化 BASIC」、「Microsoft Language シリーズ Microsoft VISUAL C++初級プログラミング入門上、下」、「同シリーズ VisualBasic 初級プログラミング入門上、下」、「C 言語によるはじめてのアルゴリズム入門」、「Java によるはじめてのアルゴリズム入門」、「VisualBasic によるはじめてのアルゴリズム入門」、「VisualBasic6.0 入門編、中級テクニック編、上級編」、「Internet Language 改訂新版シリーズ ホームページの制作」、「同シリーズ JavaScript 入門」、「同シリーズ Java 入門」、「New Language シリーズ標準 VisualC++プログラミングブック」、「同シリーズ標準 Java プログラミングブック」、「VB.NET 基礎学習 Bible」、「原理がわかるプログラムの法則」、「プログラムの最初の壁」、「河西メソッド：C 言語プログラム学習の方程式」、「基礎から学べる VisualBasic2005 標準コースウエア」、「基礎から学べる JavaScript 標準コースウエア」、「基礎から学べる C 言語標準コースウエア」、「基礎から学べる PHP 標準コースウエア」、「なぞりがき C 言語学習ドリル」、「C 言語標準ライブラリ関数ポケットリファレンス[ANSI C,ISO C99 対応]」、「 C 言語　標準文法ポケットリファレンス[ANSI C,ISOC99 対応]」、「[標準] C 言語重要用語解説 ANSI C / ISO C99 対応」ほか（以上技術評論社）

「電子書籍：カサイ．ソフトウエアラボ」

「Android プログラミング Bible 初級 基礎編」、「Android プログラミング Bible 中級 Android 的プログラミング法」、「Android プログラミング Bible 上級　各種処理」、「Android プログラミング完全入門」、「iPhone&iPad プログラミング Bible[上] 」、「iPhone&iPad プログラミング Bible[下] 」、「JavaScript によるはじめてのアルゴリズム入門 」、「Web アプリ入門（HTML5+JavaScript）」、「HTML5 を使った JavaScript 完全入門 」、「Scratch プログラミング入門」、「小・中学生のための Scratch プログラミング入門」、「ideon で学ぶ小・中学生のためのプログラミング入門 C 言語編」、「同 Java 言語編」

Scratch によるはじめての
アルゴリズム入門

2016 年 10 月 1 日　初版　第 1 刷

著者＝河西 朝雄
発行者＝河西 朝雄
発行所＝カサイ．ソフトウエアラボ
長野県茅野市ちの 813　TEL.0266-72-4778

表紙デザイン＝河西 朝樹

定価＝1,500 円＋税