

「作図チャレンジ」を使ったプログラミング入門

齋藤文康

2023 年 7 月 29 日

「作図チャレンジ」のプロジェクトは Snap! (<https://snap.berkeley.edu/>) のサイトで検索すると利用することができます。



現時点では「作図チャレンジ」の検索にヒットするのはひとつだけですが、ezushi 版を選択してください。表示されたプロジェクトで [Edit] をクリックすると、プロジェクト内で作図することができますようになります。

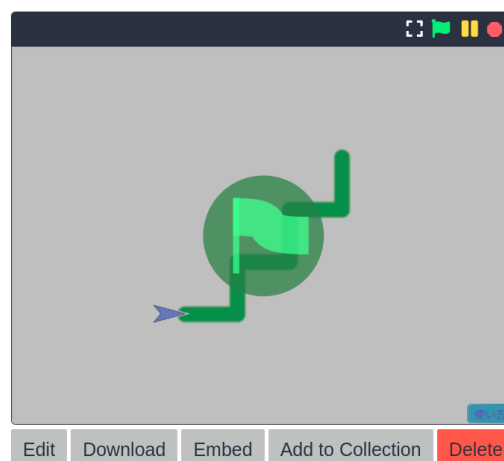
作図チャレンジ

Search Results: 作図チャレンジ

Projects



作図チャレンジ
by ezushi



Snap! はコンピューターのプログラミングを文字のコードで記述するのではなく、それぞれの動作を表すブロックを組み合わせることでプログラミングしていくものです。Scratch と同様に命令などのブロックを左側にあるパレットから選択して使用します。しかし、いろいろな機能のカテゴリーの中から必要なブロックをあちこち移動して選び出すのは初心者にとってはなかなかたいへんです。プログラミングの基本が学べる図形の作図に必要なブロックを Pen (ペン) パレットにまとめてみました。

学習の進め方としては broadcast で選択できるメニュー順に 階段 → 正方形 → 正三角形 → 正五角形 → 正六角形を想定してします。それぞれの図形は一辺を描いて向きを変えることを繰り返すことで作図できます。スクリプト例は示しませんので、じっくり考えることを楽しんでください。


このプロジェクトにはスケール設定ブロックがあるので、ステージ上の $x y$ 座標に仮定の座標を対応させてその座標上に点や線や文字を描くことができます。(グラフの描画など)

1 階段

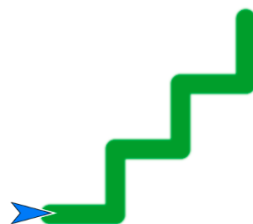
階段の作図は単純に見えますが、三通りのやり方を考えることで順次実行、条件分岐、繰り返しというプログラミングの基本を学ぶことができます。この考え方を発展させると迷路探索ができます。

1.1 順次実行

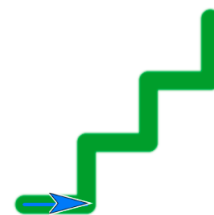



始めのスク립トのメニューから階段を選択して  をクリックします。

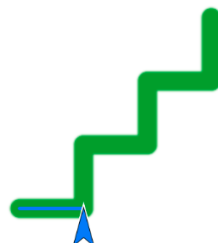
1. 右図のように下書きが表示され、初期位置に置かれます。



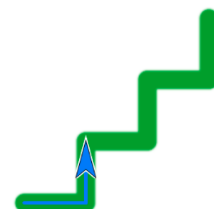
2. ブロック定義にある **進む** をクリックすると一辺が描かれます。




3. ブロック定義にある **上を向く** をクリックすると  が上を向きます。



4. ブロック定義にある **進む** をクリックすると次の一辺が描かれます。



一連の動作をスクリプトとして加えていくとこのようになります。まだ途中ですが、うまくできたかどうか  をクリックして確かめます。

このように指定したブロックを順番に行うことを順次実行と言います。Snap! でもスクリプトの先頭から下に向かって順に実行するのが基本です。



1.2 繰り返し

同じ動作を 100 回も繰り返す場合、順次実行だけでやるのは無理だし、後で動作を変更することになった場合にはそのすべての箇所を変更する必要があります。

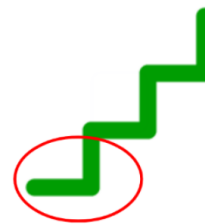
このプロジェクトでも繰り返しのためのブロックが用意されています。



左側のブロックでは繰り返しの回数を指定して C 型ブロックの中に繰り返したいスクリプトを入れてやります。右側のブロックでは **i** の変数が指定された値の範囲をステップの値ずつ増加しながら繰り返します。(10 から 1 まで ステップ -1 にすれば減少です。) **i** はクリックして名前を変えることができますし、C 型ブロックの中のスクリプトにドラッグ & ドロップしてその値を利用することができます。

右図の赤で囲った部分に注目すると、この動作を 3 回繰り返すと全体が描けます。

[3 回繰り返す] のなかに赤で囲った部分を描くスクリプトを入れてやります。



1.3 条件分岐

右図の赤で囲った部分に注目すると、[一辺を描いた後に角度を変える] ことを 6 回繰り返すと全体が描けます。変更する角度を計算で求めることもできますが、「もし~なら~をする。そうでなかったら~をする。」という条件分岐を使ってやってみてください。



角度はその時の向きが右向きならば上向きに、上向きならば右向きに変更する必要があります。

指定した条件なら、E型ブロックの上の段のスクリプトだけを、そうでなかったら下の段のスクリプトだけを実行します。

もし右向きならば

上を向く

そうでなければ

右を向く


2 スクリプトの実行制御

各ブロックの実行をゆっくり或いは一ブロックごとに確認しながら行うことができます。



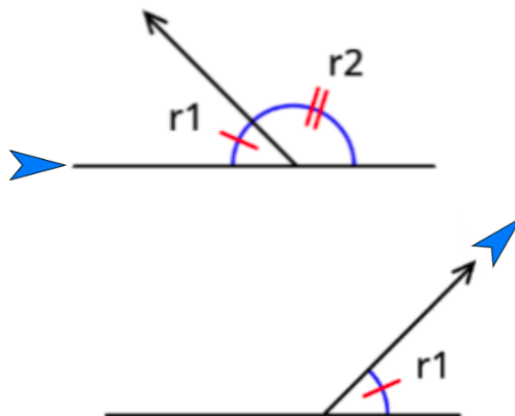
初期値としてブレークフラグに 0.5 が設定されています。これは一ブロックごとに 0.5 秒の間隔を空けて実行するという事です。0 で待ち時間なし、負数で一ステップごとにスペースキーの入力待ちになります。

発表モードを true にすると、下書きの背景を表示しません。

スペースキーで再開 でポーズの際の再開をスペースキーで行うか  で行うかを指定します。

3 ブロックの説明

▶ を右図上段のように移動させたい場合、[左へ r1 度回す]としてしまうと下段のようになるので [左へ r2 度回す] としなければなりません。



▶ の向きに関するブロックです。

向き ⇒ 0 0 ブロックで ▶ の向きを一般的な 0 度 ~ 359 度で指定できます。角度の指定は直接数値を入れることもできますし、三角ボタンをクリックするとマウスでの指定もできます。

向きを 反時計回りに (左) に 度回す ブロックで ▶ の向きを右や左へ指定の角度変更できます。

向き ⇒ 0 ブロックで ▶ の向きの値を参照することができます。

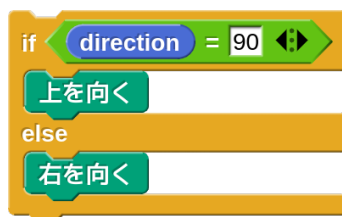
簡略化してカスタムブロックにしてしまった条件分岐について補足説明しておきます。

条件分岐用のブロックは次の 2 つがあり、条件判断のためのブロックは六角形のものになります。



条件分岐用のブロックを使うと右のようになります。

角度はシステム変数を使ってみました。





direction = 90 が条件判断のためのブロックで、**direction** は ▶ の向き (角度) を示すシステム変数です。これで、「向きは 90 度に等しいか?」「(「右向きか?」) の意味になります。(Snap! では 90 度が右)

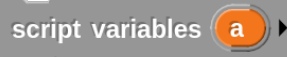

このような六角形のブロックはクリックされたり実行されると、指定された条件をテストしてその結果をレポートします。

この時点では **direction** 90 なので、**direction = 90** true (真) ということになります。したがって、この場合 E 型ブロックの上の段のスクリプトだけを実行することになります。(**上を向く**)

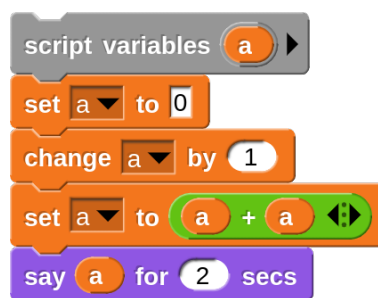
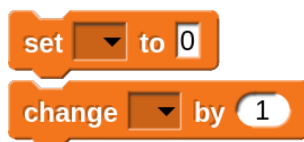
ちなみに、  (偽) になります。

 は指定された左右の条件がどちらも成立した場合だけ true (真) になります。

 は指定された左右の条件がどちらかでも成立すれば true (真) になります。


少し複雑なことをしようとすると、変数が必要になってきます。Scratch には、すべてのスプライトで使えるものとそのスプライトだけで使えるものがありました。Snap! には加えてスクリプト変数という、それに続くスクリプト内だけで使えるものがあります。[Variables (変数)] のところにある  で、右向きの三角ボタンをクリックすると変数がふやせます。すると、逆向きの三角ボタンが出現します。それをクリックすると変数を削除できます。そこにある変数  のところを右クリックすると名前の変更ができます。ここから変数をドラッグしてこの後のスクリプトで使用します。

変数に値を入れたり増加させたりするには右のブロックを使用します。



算術演算子  は、割り算 \div の演算子です。

 は、「 \wedge 」の乗  で x^2 の意味です。

 では、下向きの三角ボタンをクリックするとメニューから演算を選ぶことができます。

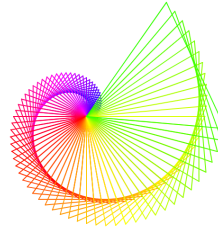
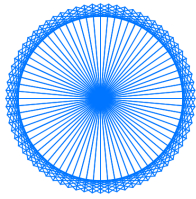
4 その他の図形

向きを変える角度と繰り返しの回数を指定してやれば、正方形や正三角形、星形まで描くことができます。それぞれの図形での外角、作図のために向きを変える角度を考えることは価値があることだと思います。それぞれ角度を指定して描くことができたならば、角度をスクリプトで求める方法も考えてみてください。

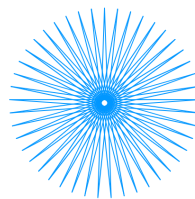
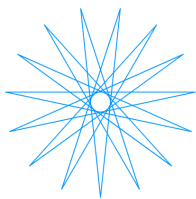
5 図形なし

下書きに沿った図形問題をやり終えたなら、下書きを表示しないで正三角形などを描きながら 5 度ずつ回転して一周させたり、始めの歩数を小さくして回転させながら徐々に色を変え大きくさせると面白い図になります。この場合、描き始めはステージの中心になります。

参考文献: Peter Farrell 著、鈴木幸敏 訳『Python ではじめる数学の冒険』オライリー・ジャパン
p13 課題 1-2 四角形による円、p21 課題 1-5 カメのらせん の応用





また、星形の発展形として角の数を増やした図形を描くことにチャレンジしてみてください。




6 グラフ表示

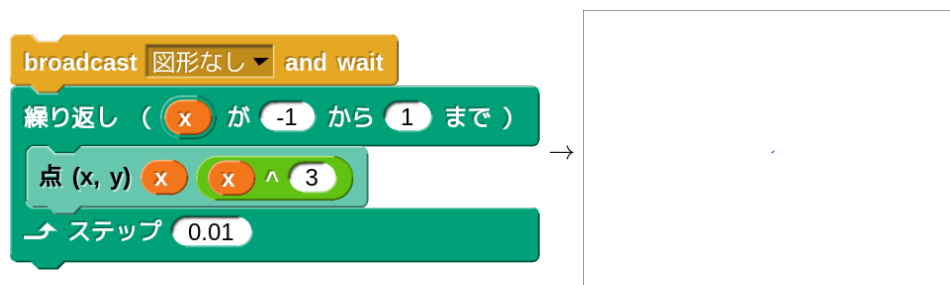
ここまでは **進む** で線を描いていました。x 座標 (-240 ~ 240) y 座標 (-180 ~ 180) の値を指定して点や線を描くこともできます。

点 (x, y)  ブロックは指定した x 座標 y 座標の位置に点を描きます。

線 (x1, y1, x2, y2)  ブロックは指定した始点と終点の x 座標 y 座標の位置を結ぶ線を描きます。

 キーを押すとデモスクリプトが実行されます。スクリプトを使い方の参考にしてください。

グラフなどを描く場合にスケールが問題になる場合があります。 $y = x^3$ のグラフを、x の値が -1 から 1 まで 0.01 刻みの y の位置を描くスクリプトは次のようになります。



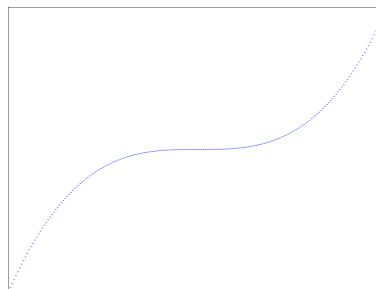
x の値が -1 から 1 までということは、画面上の 3 画素の範囲なので形がまったくわかりません。スケールブロックで、指定した x y 座標の範囲を座標値とする仮想ステージを作ることができます。(前出 参考文献 P80 例 4-7 参照)

仮想ステージ上の座標を指定して **点 (x, y)** などを実行すれば実際の Snap! のステージの位置に変換して描画されます。

(-1) (1) (-1) (1) を指定すれば x の値の範囲をステージの大きさとしたグラフが描けます。グラフを描くスクリプトの部分は同じです。

```
broadcast 図形なし ▶ and wait
スケール (xmin = -1 , xmax = 1 , ymin = -1 , ymax = 1 )
繰り返し ( x が -1 から 1 まで )
  点 (x, y) x x ^ 3
  ↳ ステップ 0.01
```

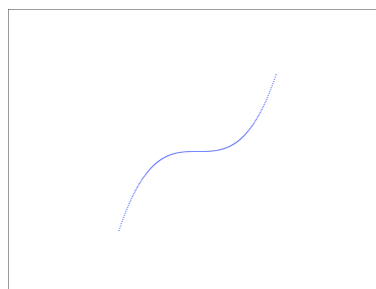
ステージの縦と横の長さが違うのに、どちらも -1 から 1 までと同じ値を指定したので 1 目盛り分の縦と横の長さが違う状態になります。



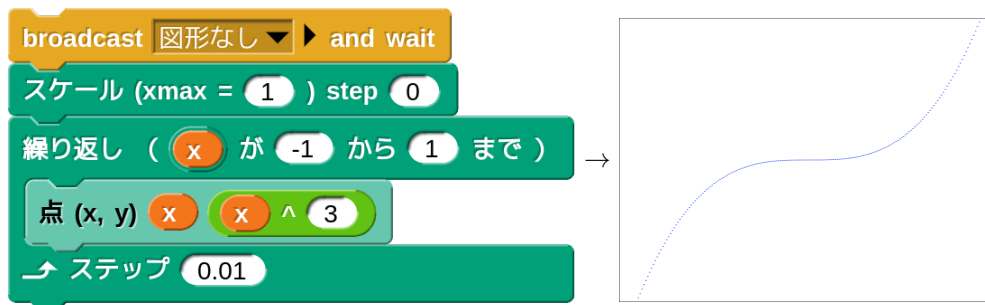
Snap! のステージの画面比率に合わせて、(-2.4) (2.4) (-1.8) (1.8) を指定すれば次のようになります。



```
broadcast 図形なし ▶ and wait
スケール (xmin = -2.4 , xmax = 2.4 , ymin = -1.8 , ymax = 1.8 )
繰り返し ( x が -1 から 1 まで )
  点 (x, y) x x ^ 3
  ↳ ステップ 0.01
```

x の範囲が -1 ~ 1 のままなのでステージ全体を使っていません。

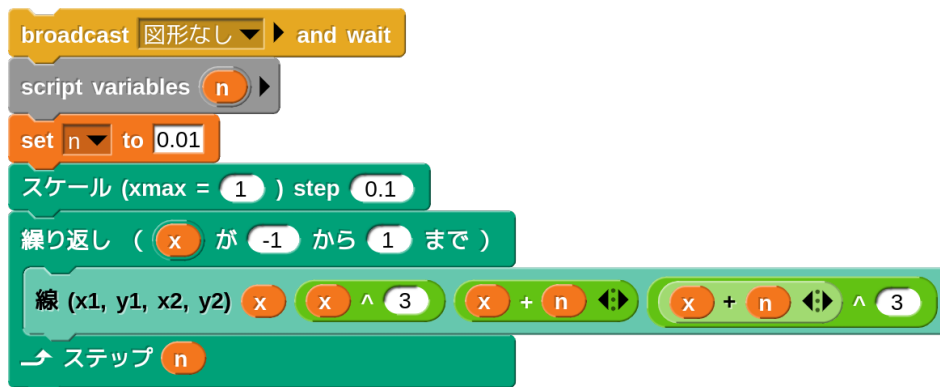


次の例のスケールブロックを使うと、仮想ステージ x 座標の最大値を指定するだけで Snap! のステージの画面比率に合わせて設定されます。



これまでの [点] で描いたグラフは x の値の刻み幅が 0.01 だったので、間が合いた点のグラフになっていました。刻み幅を 0.001 とかにすると曲線のようなグラフになります。(それを実行する場合は [Shift] キーを押しながら  をクリックしてターボモード  にしてください。)

[線] ブロックを使って補間処理をすると繰り返しの回数を減らせます。



スケール (xmax = 1) step 0.1 ブロックは step を指定しないか 0 の場合はグリッド線なしで、そうでなければ指定した間隔でグリッド線を描きます。既定値は 0 グリッド線なしです。

