第 2 章

Scratch

2.1　Scratch とは

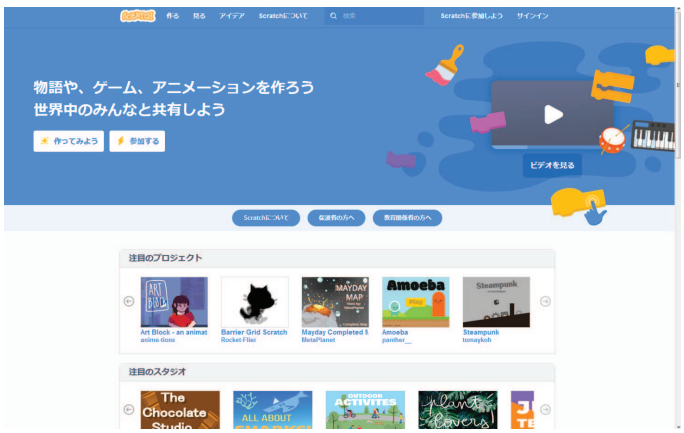
初心者が最初に正しい構文の書き方を覚えること無く結果を得られるプログラミング言語学習環境である。

2 年生の講義では C 言語を用いてプログラミングを行う。プログラミングを行うとき、最初に必要なのは **アルゴリズム** (すなわち答えまでの手順) を考えることである[*1]。実は、この作業が苦手な人が多いようなので、難解なプログラミングの構文やフローチャートなど意識せず、プログラミングが出来る Scratch を紹介する。

Scratch は、web 上で起動して使用する方法と、アプリをインストールして起動して使用する方法がある。Scratch のホームページ

`https://scratch.mit.edu/`

を表示すると



の画面が表示される。

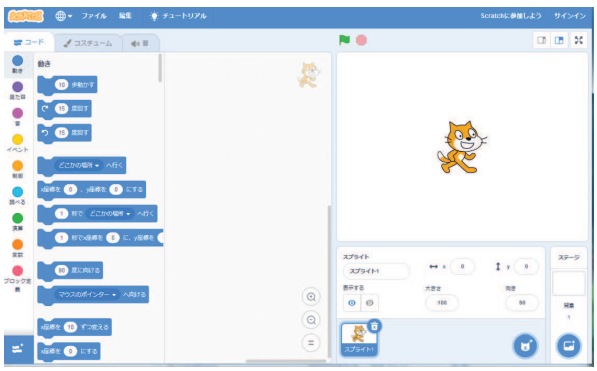
[*1] この点は、数学の証明問題と似ていると思う。

★ インターネットに接続された環境で使用する場合、このサイトの 作ってみよう をクリックすると Scratch を使うことが可能になる。

★ インターネットに接続されていないパソコンや Android スマホで使用する場合は、アプリをインストールして行う。

上記ホームページの下段に、[リソース] の下に [ダウンロード] があるので、クリックする。

下記の画面が表示されたら適切なアプリをダウンロードし、インストールする。残念ながら、iPhone, iPad 用のアプリはない。



Scratch を起動すると右の画面が表示される。(チュートリアルが開く場合は、チュートリアルを閉じる)

左側はコードの一覧があり、この一覧のなかにはプログラムを作るために必要な命令 (ブロック) がある。

真ん中には (ブロックを組み立てる) スクリプトエリアがあり、右側がスプライトが表示されたステージ (上段) とスプライト一覧 (下段) がある。

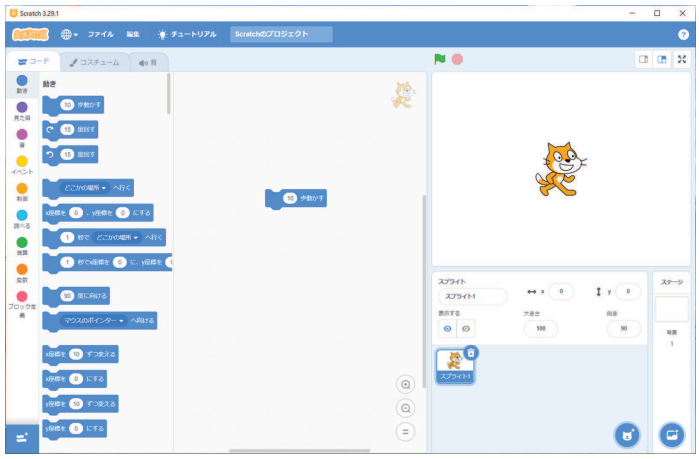
プログラムの作成は、コードにあるブロック一覧から命令や制御文をスクリプトエリアにドラッグし、繋げていく。

2.1.1 プログラム作成

実際に簡単なプログラムを作成して、動作の仕組みを理解する。

(1) ブロックパネル内の【動き】から、 

を選択し、スクリプトエリアにドラッグする。



ドラッグした 

をダブルクリックすると、実際にどのような動きか確認できる。

(2) プログラムの開始動作を確実にするために、【イベント】から 

をスクリプトエリアにドラッグし、これら 2 つを 

のように繋げる。

繋げたのち、右上の 
の旗をクリックすると scratch cat が 10 歩進む。

(3) これだけでは解りにくいので、今度は【制御】から 

を選択し、スクリプトエリアにドラッグする。

そのとき、右下図のように繋がるようにドラッグする。

(4) 繋げたら先ほどと同様に右上の 
の旗をクリックする。今度は scratch cat が右端まで行き、止まる。

このとき、スクリプトエリア内のブロックが黄色枠で囲まれた状態になっている。これは、プログラムが動き続けていることを意味している。

そこで、プログラムを停止するために 
の●をクリックする。すると、黄色枠が無くなり停止したことを表している。

(5) 次に【動き】から 

を選択し、右図のブロックの組になるようにドラッグする。

再び右上の  の旗をクリックすると、今度は右端まで行くと折り返して左端まで行き、また折り返して右端へと向かう。

ただし、右から左に移動するとき上下反転していて不自然なので、 

を間に入れる。このときの折り返し方は、

[左右のみ]、[回転しない]、[自由に回転]

の 3 種類から選択できる。指定していない場合は、[自由に回転] になっている。

スプライトの向き (角度) を自由に変えるには 

を用いる。

また、スプライトの移動する方向 (角度) を変えるには、

と 

の間に 

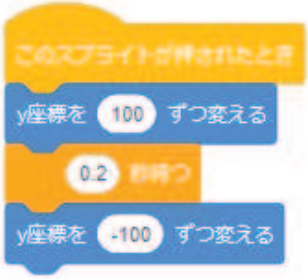
を入れる。

その後、90 度の部分をクリックすると角度を変えられる (右図)。

このように Scratch ではブロックを繋ぎ合わせることでプログラムを作成することが出来る。

もちろん C 言語や Java のように複雑なプログラムは組めないが、アルゴリズムの学習には十分な命令 (ブロック) があるので、各自で試してほしい。

例 2.1.1. 右のプログラムは、ステージの Scratch cat を押すと *(*雑ではあるが*)* ジャンプするプログラムである。



このプログラムでは、別のブロックの組を作成し、ある条件で動作を実行するようになっている。

このように、複数のブロックの組を作成し、*1* つのスプライトを制御することができる。もちろん、複数のスプライトをステージに登場させることができる。

♠ **補足** ステージの座標は中心が $(0, 0)$ で、x 軸が -240 から 240、y 軸が -180 から 180 のサイズになっている。ただし、スプライトはこの範囲を超えて移動する。

例 2.1.2. 右の例では、*Scratch Cat* が四角形を移動するプログラムとなっている。

を *3* つ別々にプログラムしているのは、

は滑らかに移動するのではなく、瞬時に移動するためである *(*右のプログラムでは *50* 歩にしている*)*。

そのため、*50* 歩動いた後に、 *1* 秒待つ をいれることによって、少しずつ移動している感じをだしている。

50 歩動かす を 3 回繰り返した後、角度を 90° 回転させている。

この動作をずっと繰り返すことによって、四角形を描いて移動している。

また、初期値として x 座標と y 座標および、最初の向き を指定している。

課題 2.1.1. *2* つのブロック

と

をふくめたプログラムを作成せよ。

この *10* 歩と、*15* 度は適時変更してよい。

例 2.1.3. 途中で コスチューム を変えて移動するプログラムである。

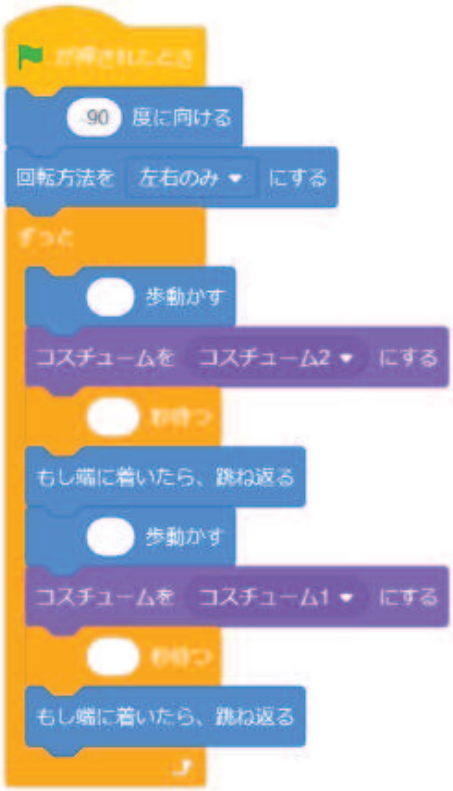
歩数と待つ時間を空白にしているので、いろいろな数値を入力してどのように動きが変わるか確認してみる。

♠ **補足** スプライトを追加したい場合は、画面右下にある

をクリックする。

背景を変えたい場合は、右下の

をクリックすると背景を選ぶことが出来る。



課題 2.1.2. 上記の例を参考に、スプライトが動くプログラムを作成せよ。

動きは自由に考えて良く、*(*詳しく説明していないが*)* 背景を変えたり、スプライトを増やしても良い。

2.1.2 ファイルの保存, 読み込み

作成したプログラムは、[ファイル] から [コンピューターに保存する] を押すと下の図の右図が出るので、[ファイルを保存する] を選択する (図 1)。

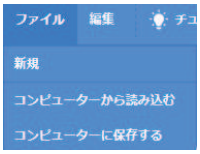


図 1

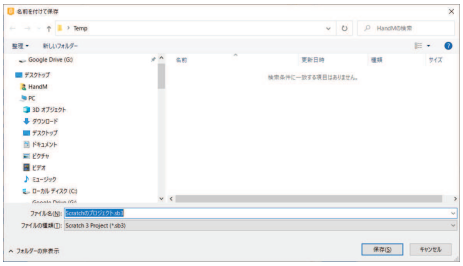


図 2

パソコンにインストールされたアプリを使っている場合は、保存先を聞かれる (図 2) ので、好みの場所に保存する。その際、ファイル名を変更してもよい。

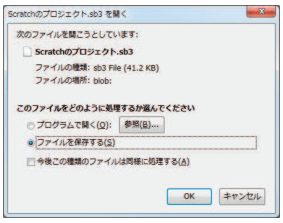


図 3

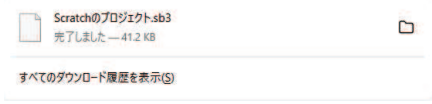


図 4

web ブラウザ上で行っている場合、ブラウザによって処理が異なる。

ファイルの処理方法を聞かれた場合 (図 3) は、“ファイルを保存する (S)”を選ぶと、(通常) ダウンロードフォルダに保存される。

Firefox の場合、図 4 のように表示されファイルの保存が行われる。

また、スマホやタブレット (iPad 等 Windows OS 以外) でも同様に保存が出来る (場合もある)。もし上手くいかない場合は、[Scratch アカウント] を作成して保存する。

スマホの場合だけでなく、複数のパソコンで作業する場合は、[Scratch アカウント] の作成をする。まず、Scratch のホームページの [Scratch に参加しよう] をクリックして、質問に答えていく。

このとき、ユーザー名は本名や学生番号を使わないようにするように。

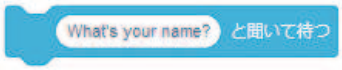
作成が完了したら、[サインイン] から、いつでもサインインしてファイルの保存、呼び出しが可能になる。

保存したプログラムを読み込む場合は、“コンピュータから読み込むを”選び、保存した場所からファイルを選択する。

2.2　入出力と演算

2.2.1　入力と出力

キーボードから入力されたデータを利用するプログラムを考える。まずは、キーボードからデータを入力し、表示するプログラムを作ってみる。

キーボードからデータを入力するには、【調べる】にある 

を使用する。このブロックを選択し、スクリプトエリアにドラッグする。

♠ **補足** ブロック内の “Wat’s your name?” は環境によって、“あなたの名前は何ですか？ ” になっている。どちらの場合もスクリプトエリアにドラッグした後、変更できる。

このブロックが実行されると、(Scratch Cat のいる) 実行画面にデータ入力欄が表示される。

入力されたデータの値は 

に格納される。このブロックもスクリプトエリアにドラッグし、さらに、ブロックパネルの【見た目】から 

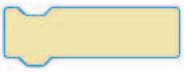
もスクリプトエリアにドラッグする。

先にドロップした 

を 

の “こんにちは!” の上にドラッグし、

のようにプログラムする。このプログラムの実行結果の確認は、各自に委ねる。

♣ **補足**　コードにあるブロックには、上下 (もしくは一方のみ) に窪みと出っ張りのあるブロック  と、上下のどちらにも窪みも出っ張りも無いブロックがあり、さらに左右が丸まったもブロック  と、とがったブロック  に分けられる。

これらの種類は意味があり、はめ込むことが出来ない場合は利用できない (意味が通らない) ことを意味している。実際にプログラムを作成していると解る (と思う)。

2.2.2 四則演算

【演算】の中に、まず四則演算がある

これらは、数の四則演算と同じである。以下の例を見て、説明とする。

例 2.2.1. 下のようにブロックを組み、旗マークをクリックすると、*Scratch cat* が *“3.50”* と言ってくれる。ただし、有効桁数は あまり多くない。

例 2.2.2. 前節の入力も含めたプログラミングとして、下のようにブロックを組んでみる。

“自然数を入力してください”と問われるので、自然数を入力すると *3* で割った値が表示される。

♡ **注意!** 四則演算のブロックは (+),(−),(*),(/) のようにカッコがあると思うように。  は、((+)*) と (+(*)) である。

【演算】の中には、四則演算以外、(とがったブロックの形をした) 制御の条件や、文字の演算をおこなうブロックもある。また、簡単ではあるが三角関数なども扱える。

例 2.2.3. 【演算】の中にある  を選び、“絶対値”の部分をプルダウンすると、平方根や $\sin, \cos, \tan$ の値を扱うことが出来る。

2.2.3　文字演算

Scratch では文字や文字列に対しても演算を行うことが出来る[*2]。

【演算】にある文字の演算をおこなうブロッのうち、ここでは両側が丸みのあるブロックとして

の 3 つを紹介する。

それぞれは、

・左側は文字列と文字列を併せて 1 つの文字列とする演算、
・真ん中は文字列の指定番目の文字を表す演算、
・右側は文字列の長さを表す演算

となっている。

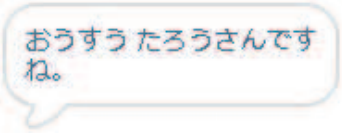
使い方は、

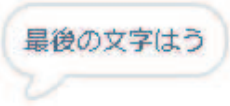
のように使う。

・ まず、キーボードからの入力を受け付け、 入力されたデータが 答え に格納される。

・ 次に、 答え と文字列 “さんですね。” を組み合わせて 1 つの文字列として、扱っている。

・ また、 答え の長さを表す演算で長さを求め、 答え のその長さ番目の文字を表す演算で得られた文字を文字列 “最後の文字は” に併せて表示させている。

実行すると、“あなたの名前はなんですか？ ” と聞かれるので、例えば “おうすう たろう”と入力すると、まず 

と 2 秒表示され、その後 

と 2 秒表示される。

♠ **補足**　演算が長くなり、見づらくなる場合は次で紹介する変数をもちいて、適時変数に代入をするのも良い。

[*2] ここでいう演算は、文字や文字列に対して、ある文字, 文字列または数値を返す関数と考えると解りやすい

2.2.4 変数

上の話では 1 つの変数 (答え) だけを扱っていたが、実際のプログラムを作るときは複数の変数が必要となることが多い。

ここでは複数の変数の準備 (定義) を行い、それら変数を用いたプログラムを考える。

【変数】を選択すると、右の画面が表示される。

ここで、変数を作る を選ぶと、

新しい変数名：

と聞かれるので、変数名 (例えば x) を入力して OK を押す。

入力すると変数が作成され、変数を作る の下にその変数名のブロックが表示される。

後は 答え と同様に扱えばよい。

変数を作る を繰り返すことによって、複数の変数を定義することが出来る。

もちろん、これら定義した変数は **演算**を行うことが可能である。ただし、以下の点に注意する。

♡ **注意!** 演算で、数値の代入 (y の値を x に代入) をするとき、

を用いてはいけない。このブロックは比較のとき使うもので、変数への代入は【変数】にある

を用いて

としなければならない。また、x に $3y$ の値を代入するなら、演算の積を用いて

とする。

例 2.2.4. *2* つの数を入力し、和と差を表示するプログラムを考える。考え方は、

1) *1* つめの数値を聞き、入力された数値が 答え に入る。

2) *2* つめの数値を聞く前に、変数 *(*例えば、 x *)* を用意し 答え の値を変数に代入しておく。

3) *2* つめの数値を聞き、 答え に入る。

4) x と 答え の和と差を表示する。表示は apple と banana を使って、

のようにする。プログラムとしては、もう *1* つ変数を用意した方が見た目は良い。

例 2.2.5. マウスのポインタの座標を表示し続けるプログラムを考える。

文字の和を *2* つ用いて、右のようにプログラムを作る。しかし、この場合旗マークをクリックしたときのマウスのポインタがある場所の座標を表示して終わる。

表示させ続けるには、動作を継続させることを忘れないように。

2.2.5　乱数

変数は入力したものだけでは無く、生成させたものを使うことも出来る。乱数を生成させる場合は 1 から 10 までの乱数 を使う。初期では 1 から 10 までの自然数となっているが、−10 から 10 までとすることで負の数も生成する。

また、範囲を −10.0 から 10.0 とすることで、有理数の乱数を生成することができる。

課題 2.2.1. *2* つの自然数を乱数で生成し、*2* 秒ずつそれらの数を言い、その後、和と差を *2* 秒言うプログラムを作成せよ。ただし、生成する数は *1* から *100* までとする。

2.3 制御

プログラムの流れの中で、条件によって分岐 (制御) を行う代表的な 2 つを紹介する。

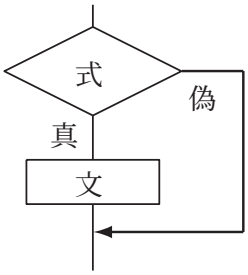
2.3.1 if 文に相当するブロック

まず、“if 文” と呼ばれる分岐方法で、

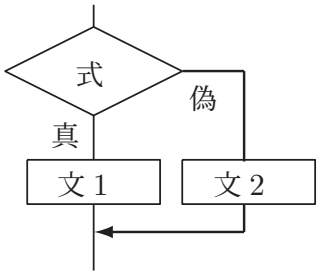
図 1

図 2

の 2 種類 (if 形式と if-else 形式) がある。フローチャートと呼ばれる図では



if 形式



if-else 形式

のように表される。

上のブロックの、条件  はフローチャートの “式” に当てはまる。

条件  を満たしているなら、その間にあるブロック (フローチャートでは “文”) を実行することを意味している。

図 2 のブロックは、条件  を満たしているなら、1 つ目の間にあるブロック (フローチャートでは “文 1”) を実行し、満たしていないなら、2 つ目の間にあるブロック (フローチャートでは “文 2”) を実行する。

ここでいう “条件  ” とは [演算] にある

などである。これらの条件分岐は良く使われるので、しっかり活用して欲しい。

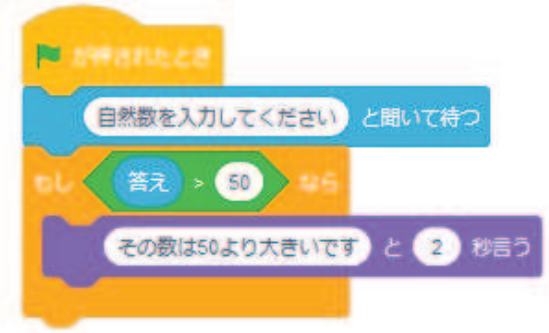
実際、制御を使ったプログラムの例を見ながら考えてみる。

例 2.3.1. (分岐の例 1)

キーボードから入力されたデータが、*50* より大きい数か否かを判断し、*50* より大きい数のときは、

“その数は *50* より大きいです”

と表示するプログラムである。



条件として、

“ 答え が *50* より大きい”

が使われている。この条件を満たしていれば、

“もし〜なら.....” ブロックの中にある “.....” ブロックが実行される。

例 2.3.2. (分岐の例 2)

キーボードから入力されたデータが、*50* より大きい数か否かを判断し、*50* より大きい数のときは、

“その数は *50* より大きいです”

と表示し、そうでないときは、

“その数は *50* 以下です”

と表示するプログラムである。

条件として、

“ 答え が *50* より大きい”

が使われているのは、上記の例と同じである。

この条件を満たしていれば、“もし〜なら..... でなければ · · · · · ”ブロックの中にある “.....”ブロックが実行され、条件を満たしていなければ、“· · · · · ”ブロックが実行される。

♡ **注意!**　この例では、文字など数以外を入力した場合、正しく表示されない。*(a, A, +, −* などを入力してみる。*)*

例 2.3.3. (分岐の例 3)

キーボードから入力されたデータが、偶数か奇数か判断するプログラムを考える。

右のようなブロックの組になるが、*3* つ目のブロックの判定 *(*= 0 となっている*)* 部分を = 1 に変えると、“偶数ですと *2* 秒言う” ブロックと “奇数ですと *2* 秒言う”ブロックを入れ替えても正しく動く。

このようにプログラムを考えるとき、これが唯一正解というものはないので、例と異なるプログラムを考えてみるのも楽しい。

条件を複数指定したい場合、例えば、右のブロックの例のような場合、組み合わせる順番に注意が必要な場合がある。

“かつ”と “かつ”を組み合わせる場合は、気にしなくても良いが、“かつ”と “または”を組み合わせる場合、(A または B) かつ C と、A または (B かつ C) では異なる。

(偶数または 3 の倍数) かつ 5 の倍数 と 偶数または (3 の倍数かつ 5 の倍数) でみれば解る。前者は 4 や 6 は含まないが後者は含む。

また、if 文に相当するブロック (図 1, 図 2) は、それらのブロックの中に if 文に相当するブロックを入れることができる。これらを組み合わせることによって、3 分岐、4 分岐も可能である。さらには、上記の “かつ”と “かつ”を組み合わせる場合に似ているが、後者の “かつ”を満たさない場合の分岐も作ることができる。

2.3.2 for, while 文に相当するブロック

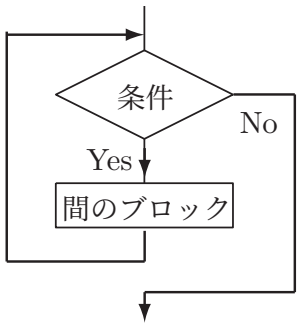
for 文、while 文は条件を満たしている間は、繰り返して使うことが出来る。Scratch において、これらに相当するブロックは次の 2 つである。

図 3

図 4

図 3 のブロックは、回数が 10 回を超えるまで間にあるブロックを繰り返し、図 4 のブロックは、条件  を満たしているなら、間にあるブロックを実行し続ける。

流れ図としては、どちらも右図のようになる。



この 2 つのブロックの使い方の違いとして、図 3 のブロック (for 文に相当) は、条件の部分で回数が明確なときに使い、図 4 のブロック (while 文に相当) は、終了条件が明確なときに使うことが多い。

例 2.3.4. (総和の例)

1 から *100* までの和を求めるなら、左のブロック、*1* から n までの和が *1000* を超えるまで求めるなら右のブロックを使う。

それぞれの場合、変数の扱いを理解していないと正しい結果を得られない。特に、初期化 は大切で、忘れないようにする。

2.3.3　初期化

初期化とは、変数の値を 0 や 1 などにしておくことであり、初期化していない変数は、前回実行したときの値が残っていたり、ブロックを組み立て中に値が変化している。

プログラムの途中で初期化（リセット) をすることもある。

例 2.3.5. (初期化)

1 から 10 までの和と、1 から 10 までの積を求めるプログラムを作成する。

まず 考えてみると、どちらも 1 から 10 と範囲は同じなので、まとめて計算する方法と、和と積は別々で計算する方法が思いつく。

後者は 例 2.3.4 の 1 から 100 までの和を参考に出来るので各自に委ねる。

まとめて行う場合は、変数をもう 1 つ追加する。(右にあるプログラムは変数 sum と pro を使用している。)

このとき初期化として、和は 0 で積は 1 となることに注意する。

和は sum に値を加えていき、積は pro にかけていく。

例 2.3.6. (ユークリッドの互除法)

キーボードから入力された 2 つの数値の最大公約数を求めるプログラムを考える。

まずは、最大公約数の求め方の復習をする。

(ユークリッドの互除法)

大きい方を a 小さい方を b として以下の試行を繰り返す。

$$\begin{array}{lcll} a \div b & = & n_1 & \text{余り } r_1 \\ b \div r_1 & = & n_2 & \text{余り } r_2 \\ r_1 \div r_2 & = & n_3 & \text{余り } r_3 \\ & \cdots & & \\ r_{k-1} \div r_k & = & n_{k+1} & \text{余り } 0 \end{array}$$

このとき、r_k が最大公約数がとなる。

このアルゴリズムに従い、以下の点に注意し考える。

・ 入力された 2 つの自然数のうち大きい方を a, 小さい方を b とする。

そのため “もし〜なら” ブロックを使って 1 つめに入力された数値 (答え) を a に代入し、この a と 2 つめに入力された数値 (答え) の大小を判定する。

(答え) が a より大きいならば、b を a として、a を (答え) とする。

そうでなければ、a はそのままで、b に (答え) を代入する。

・ 1 回の演算の後、変数の入替え (代入) の順序に注意する。

・ 終了判定は、余りが 0 となったときである。

余りの判定の段階では、余り (r) は 0 であり、変数 b の値も 0 になっている。

そのときの 1 つ前の余りには、変数 a に代入されているので、答えは a となることに注意する。

以上に注意し、考えたものが右のプログラムとなる。

2.4　データの扱い、リストとソート

2.4.1　リスト

変数をたくさん使ったプログラムを考える場合は、リスト (配列) を使う。リストは変数を束ねたようなもので、a, b のようなリスト名に対して、a の 1 番目や、a の 2 番目のようにして使うことが出来る。

変数を作ったときと同じように【変数】を選択し、リストを作る を選ぶと、新しいリスト名：と聞かれるので、リスト名 (例えば a) を入力して OK を押す。作成されるとステージに右のようなリスト名が書かれたリストが作成される。

リストを扱うブロックが並んでいるのでこれまでと同様に見て貰えば解るかもしれないが、少し注意が必要なブロックもあるので、少し説明をしておく。

まず、 

は、単なる変数と思うことが出来、 

と同様に扱うことが出来る。リストにデータを追加して行く場合は、 

を使う。

また、既にあるデータを変更する場合は、 

を使って、値を代入することができる。

実際にデータを追加して行くと、右のような状態になる。数値以外にも文字列も同時に扱うことができる。(1 つのリストで両方扱うことはあまり推奨はしない。)

ちなみに、これはリストの中に 3 つのデータ $(4, 6, -3)$ が入っている状態で 

が実行された状態である。このブロックはリストの最後に新たなものを追加するブロックである。

次に、 

のブロックであるが、これはリストの (1) 番目のデータを削除し、(2) 番目以降を繰り下げて格納し直す。右の例だと、1 番目が 6、2 番目が -3、3 番目が “なにか” になる。

注意が必要なのは、 

のブロックであり、これはリストの (1) 番目以降を繰り上げて、その後 (1) 番目にデータを格納する。右の例だと、1 番目が “なにか”、2 番目が 4、3 番目が 6、4 番目が -3、5 番目が “なにか” になる。

また、リスト a の “長さ” が 10 のとき 

は実行されない。(11) 番目に挿入はしてくれない。

もうひとつ注意が必要なものは  で、これはリスト a の中に “なにか”があるか探し、見つかればその格納番号を表してくれる。右上の例なら、格納番号 4 が値となる。また、見つからない場合は 0 が値となるが、複数見つかった場合は小さい格納番号のみが値となる。

その他  や  はそれぞれリストの内容を消すものと、リストの成分の数を扱っている。また、  は制御のときの条件として扱う。

 は、リストに格納されたデータの表示の ON/OFF を行う。

例 2.4.1. (乱数のリストへの格納)

1 から *10* までの乱数を *10* 個生成し、それら乱数をリストに格納するプログラムを作成する。ここまで学習していると、右のようになることは容易に解る。

実際作成した後、旗マークを押すと、リストとして 10 個の乱数がが格納されている。

ただ、注意が必要なのはこの後で、再度旗マークをクリックすると新たに 10 個の乱数が生成され、全部で 20 個になる。

新たに 10 個の乱数を生成したい場合は  を付けくわえればよい。

例 2.4.2. (約数のリストへの格納)

入力された数の約数をリストに追加するプログラムを考える。

まず、リストの初期化を忘れず行う。

入力された数 (答え) を 1 から順に割っていき、割り切れたとき(余りが 0 のとき)は約数なのでその数をリストに追加する。

これを 答え まで行う。

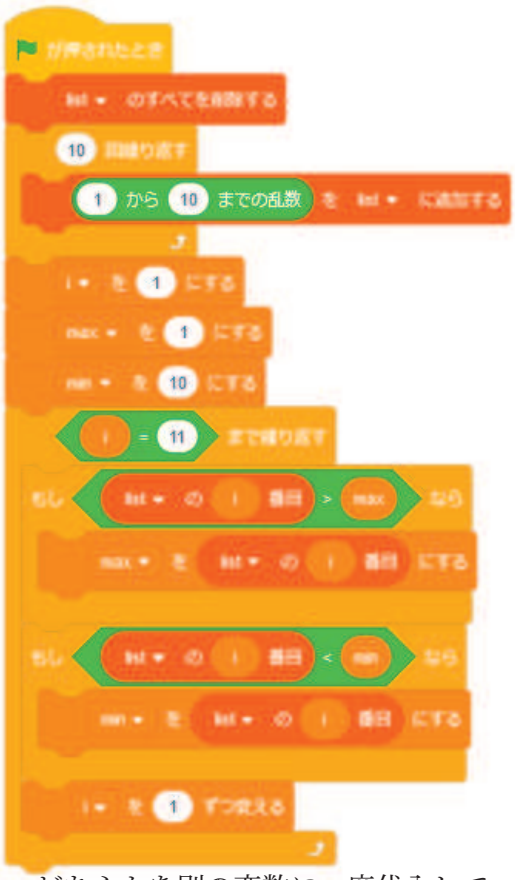
(1 から 答え までなので、 答え 回行うことと同義である。)

例 2.4.3. (最大値、最小値)

例 2.4.1 で生成させた乱数のうち、最大値と最小値を求めるプログラムを考える。

右図は最大値 (max) と、最小値 (min) を求めるプログラムの例である。最大値と最小値を探す前に、変数 max と min をそれぞれ 1 と 10 にしていることに注意する。

表示を行う場合は、このプログラムの続きにブロックを追加する。



2.4.2　ソート

生成したり、与えられた複数のデータを、ある規則に従って並べ替えることを “ソート”といい、ソートの方法はデータの構造やデータ数によってさまざまな方法が存在する (興味のある人はネット検索で)。

いろんなソート方法を試すと良いが、どの方法でも注意しなければならないことがある。

♡ **注意!**　リスト内の i 番目と j 番目を入れ替える場合、どちらかを別の変数に一度代入しておかなければならない。

例えば、list の 3 番目と 5 番目を入れ替えるには以下のようにしなければならない。

(list の 5 番目の値を tmp に退避させ、list の 5 番目に list の 3 番目の値を代入し、list の 3 番目に tmp の値を代入している。)

ここでは、アルゴリズムが簡単な、単純ソートを紹介する。

単純ソートは、リストの 1 番目 (比較元) の値と、2 番目以降 (比較対象に) ある各値を順に比較して、順序関係が条件 (大きい順、や小さい順など) の結果の逆になっているものがあれば、1 番目とそれらの値の位置を入替える。

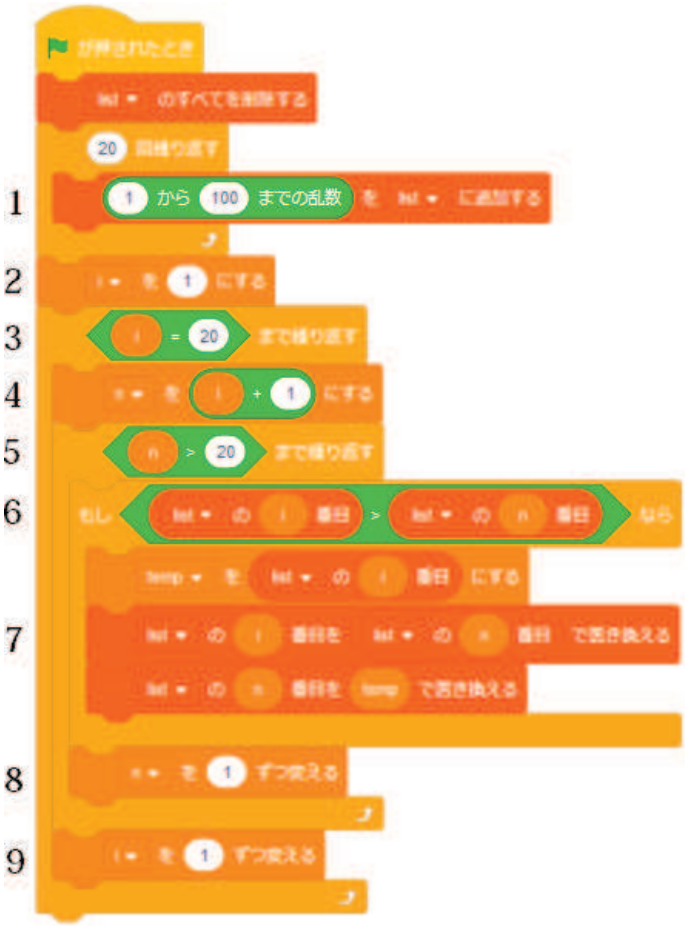
リストの最後の値まで調べ終えたら、2 番目の値と、3 番目以降にある各値を比較する。これを $n-1$ 番目と n 番目の比較まで行ったら終了となる。

例 2.4.4. リスト $\{3,7,2,5,1,4\}$ において、数の小さい順に並べ替える手順を考える。

$\{3,7,2,5,1,4\}$ 1 番目 (比較元) と 2 番目 (比較対象) を比較
$\{3,7,2,5,1,4\}$ 1 番目 (比較元) と 3 番目 (比較対象) を比較 (入替える)
$\{2,7,3,5,1,4\}$ 新たな 1 番目 (比較元) と 4 番目 (比較対象) を比較
$\{2,7,3,5,1,4\}$ 1 番目 (比較元) と 5 番目 (比較対象) を比較 (入替える)
$\{1,7,3,5,2,4\}$ 新たな 1 番目 (比較元) と 6 番目 (比較対象) を比較
$\{1,7,3,5,2,4\}$ 1 番目 (比較元) との比較、入替え終了

$\{1,7,3,5,2,4\}$ 2 番目と 3 番目を比較 (入替える)
$\{1,3,7,5,2,4\}$ 新たな 2 番目と 4 番目を比較
$\{1,3,7,5,2,4\}$ 2 番目と 5 番目を比較 (入替える)
$\{1,2,7,5,3,4\}$ 新たな 2 番目と 6 番目を比較
$\{1,2,7,5,3,4\}$ 2 番目との比較、入替え終了

以下同様に繰り返し、ソートを完成させる。20 個のデータの例は以下の通り。



最初の旗マークが押されたときの後、リストの全てのデータを削除している。

1 では、1 から 100 までの乱数をリストに追加することを 20 回行っている。

2 では、最初の比較元 (i 番目) を 1 にしている。

比較元は 19 までなので、20 になったら終わる (20 になるまで続ける) ようにしている (**3**)。

そして、最初の比較対象 (n 番目) は $i+1$ 番目なので、n を $i+1$ にする (**4**)。

n は 20 番目まで比較するため、n が 20 を超えるまで続けている。“$n=21$ まで繰り返す”でも良いが、何カ所かある 20 を 答え に変えると、汎用性のあるプログラムになる (**5**)。

6 では、i 番目 (比較元) の値が n 番目 (比較対象) の値より大きい場合を見つけている。その場合、**7** の部分で入れ替えを行っている。

入れ替えを行った場合も行わない場合も、比較対象は次のデータに移る (**8**)。

比較対象が最後の値を比較した後、比較元を次のデータに移す (**9**)。

2.4.3　拡張機能

Scratch には最初に表示されているコード（ブロック) 以外に、いくつかの拡張機能がある。それら機能は、画面左下の  から呼び出すことが出来る。

例えば、音楽を選択すると右図のブロックが追加される。これらも見るとどのような動作をするか解りやすくなっている。

その他、LEGO や micro:bit を扱えるブロックも存在する。Scratch に対応した LEGO ブロックがあると楽しさが増えるので、環境があれば試して欲しい。

ここでは、“ペン”の紹介を行う。ペンを選択すると、コード (ブロック) が追加される。これらのブロックも見てもらえば解りやすいが、2 つの説明をしておく。

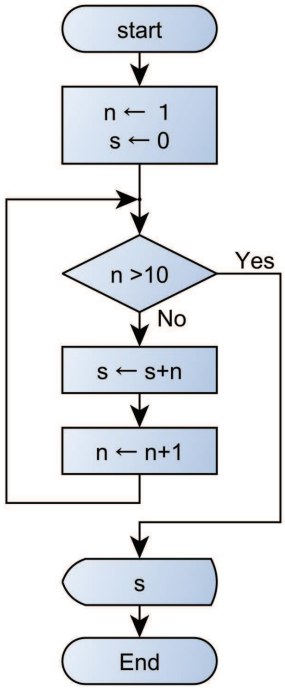
まず、“ペンを下ろす” と “ペンを上げる” は、Scratch ネコが移動中に、ペンを下ろして線を書き、ペンを上げると書くのを止める動作となっている。

“スタンプ”は、スプライトを張り付ける動作である。それぞれ確認するために、右のようなブロックの組 (4 組) を作ってみる。

スペースキーや上下の矢印キーはスマホやタブレットでは押すことが出来ないが、スマホやタブレットの場合は、それらブロックをタップすることで同様の動作となる。

例題. として、正三角形や正五角形を書く。等間隔の点線を書くなど。

2.4.4 フローチャート



フローチャートとは、作業の流れを図で表したものである。

各ステップを様々な形の箱で表し、それらの間の流れを矢印 (または実線) で繋ぐことでアルゴリズムやプロセスを表現する。

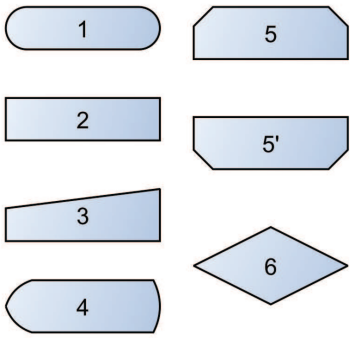
フローチャートを用いることで作成したいプログラムの大まかな流れを図示し、プログラムを組む際に役立てられる。また、他者へ説明する場合にもプログラムを見せるより視覚的で解りやすい。フローチャートはプログラミングのみならず、いろいろな分野で用いられている。

右のフローチャートは 1 から 10 までの和を求め、表示する流れを表したものである。

♡ **注意!** フローチャートの描き方にはいくつかの流儀があり、細かい点で異なっている。

例えば、start の直後の→のように、明らかに進む方向が解るものは実線で書く場合がある。

☆パーツ (様々な形の箱)



フローチャートに使われるパーツ (箱) のうち、よく目にするものを紹介しておく。

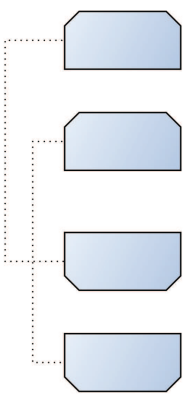
右の 1 は、手続きの始まりや終わりを意味する。

2 は処理や命令

3 は手動 (主にキーボード) からの入力

4 は画面への出力

5,5' はペアーで使い、ループの開始と終了で、6 は条件分岐である。



♡ **注意!** 5, 5' を用いてループを考える場合、右のようなループは出来ないことに注意する。

また、6 の条件分岐は成立つか否かの二択で用いることが多い。

2.5 Scratch 課題

課題 2.5.1. 入力された数値が、4 で割り切れる, 4 で割った余りが 1, 4 で割った余りが 2, 4 で割った余りが 3 か判別して 2 秒言うプログラムを作成せよ。

課題 2.5.2. 名字と名前をアルファベットで入力し、イニシャルを 2 秒表示するプログラムを作成せよ。例えば、 Ridai と Taro と入力すると T.R と言う。

課題 2.5.3. 入力された 2 つの数値の最小公倍数を 2 秒言うプログラムを作成せよ。

課題 2.5.4. 数値を入力していき、0 が入力されるまでに入力された数値の合計を 2 秒言うプログラムを作成せよ。

課題 2.5.5. 100 個の乱数データを作成し、それらを順に 0.5 秒言うプログラム を 右のように作成したが、正しく 100 個の乱数を 表示してくれなかった。

正しく表示するようにプログラムを作り変えよ。

課題 2.5.6. 入力された数 ($\geqq 2$) までの素数をリストに追加するプログラムを作成せよ。

例えば 12 と入力すると、 2,3,5,7,11 がリストの上から順に登録される。

課題 2.5.7. 乱数を用いて 10 個の自然数 (1 から 10 まで) を生成し、偶数と奇数で分けて別のリストに格納するプログラムを作成せよ。

課題 2.5.8. 乱数を用いて 10 個の自然数 (1 から 10 まで) を生成し、偶数と奇数で分けて別のリストに格納し、それぞれのリストの中で数の小さい順に並べるプログラムを作成せよ。

右のリストのウインドウは課題 2.5.7 の結果である。

課題 2.5.9. 自然数を入力してくださいと聞き、入力した自然数 答え に対して、2 の 答え 乗を 2 秒言うプログラムを作成せよ。

♡ **注意!** あまり大きな数にすると、1.245321654315e+19 のような表示になる。これは、$1.245321654315 \times 10^{19}$ を意味している。さらに大きい数にすると動作しなくなるので、ほどほどの数で試してみる。

課題 2.5.10. 自然数入力してくださいと 2 回聞き、それぞれを a, b とし、つるとかめの頭の数が合計 a で、足の数が合計 b 本の時、つるとかめの頭数をそれぞれ言うプログラムを作成せよ。ただし、存在しない場合はそのことを言うようにすること。

課題 2.5.11. “自然数を入力してください” と聞き、入力された値 答え に対して、n を 1 から 答え まで動くとする。n を除く n の約数を全て足し合わせたとき、n となる数を 2 秒ずつ言うプログラムを作成せよ。(上限は各自の好みで)

例 条件を満たす最小の数は 6 である。6 の約数は $1, 2, 3, 6$ で、自身を除く約数の和は $1 + 2 + 3 = 6$ となる。

課題 2.5.12. 2 から 10000 までの数を 1 つ入力し (自然数が入力されることは仮定し)、その数が素数なら “素数”、合成数なら “合成数” と言うプログラムを作成せよ。

課題 2.5.13. 乱数を用いて、0 から 100 までの整数を 50 個生成し、それらの数の平均、最大値、最小値を言うプログラムを作成せよ。

ただし、表示は一度に 3 つをまとめて言っても良いし、平均、最大値、最小値をそれぞれ数秒ずつ言っても良い。

課題 2.5.14. (懸賞金 1 億 4 千万円) 自然数を入力してくださいと聞き、 答え を n とし、以下のコラッツの予想で 1 になるまでの過程を 1 秒ずつ言うプログラムを作成せよ。

例えば、3 が入力されたとすれと

$$3 \to 10 \to 5 \to 16 \to 8 \to 4 \to 2 \to 1$$

の順で 1 になるので、10, 5, 16, 8, 4, 2, 1 をそれぞれ 1 秒言う。

コラッツの予想

1. 自然数 n を考える。
2. n が偶数ならば n を 2 で割る。
   n が奇数なら 3 倍して 1 加える。
3. 2. の結果 (を新たな n とし、n) が 1 なら終わり、(n が) 2 以上なら再度 2. を行う。

以上の方法でどんな自然数 n も必ず 1 になる。(という予想)