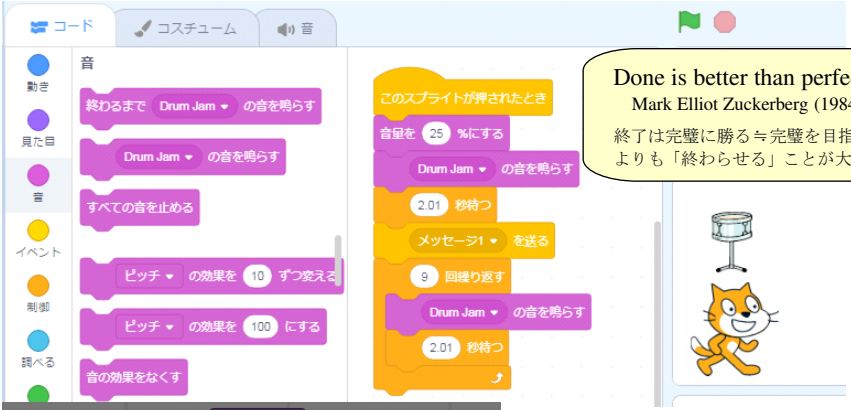
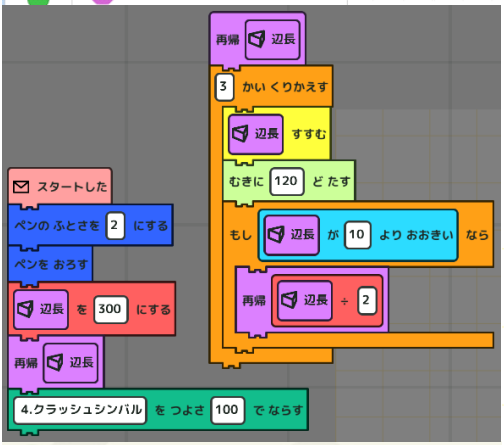
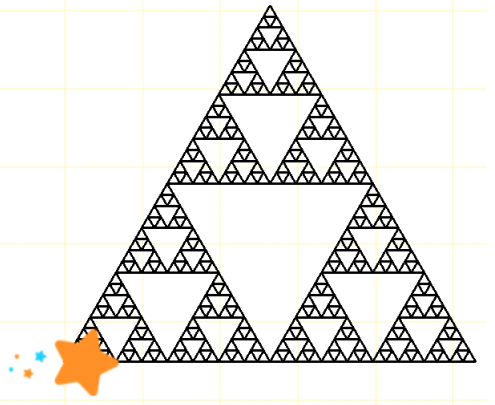
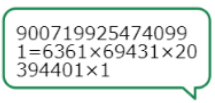
部品型言語を３つ試してみました。 2021 年 9 月 04 日　加藤 厚

― Scratch もプログラミングゼミも LINE entry も offline で使えます！ ―









目次

これまで文字型プログラミング言語（JavaScript など）を使ってきた筆者から見ると、部品型のそれは「なんだか面白そう」です。そこでその代表といえる Scratch と国産の２言語を（少しだけ）試してみました。

音や線画が簡単に扱えるのは期待どおりの「面白さ」でしたが、計算や関数処理の速度、（殆ど言及していませんが）data 入出力などについては限界も感じました。

本冊子の内容は、筆者が NEC VersaPro VX-B（CPU Intel Core i3-M370、RAM 4.0GB）上の Windows 7 Home Premium（64-bit）で確認した結果をまとめたものですが、部品型言語の素人が試した結果ですので、無知や誤解などに基づく不適切な内容があるかもしれません。お気づきの点がありましたら atsushi.kato.1958@gmail.com までお知らせ頂ければ幸いです。なお、旧式 OS の活用や JavaScript による計算・統計処理に興味がある方には、筆者の以下の頁から link されている諸資料がお役に立つかもしれません：
http://mmua.html.xdomain.jp/kato/@vector.html

【用語】

関数（function‥機能・働きなどの意味もあります）：
本来は「与えられた値に一定の処理を行って結果を出力する一連の命令のまとまり」です。Scratch とプログラミングゼミではブロック（block=塊）、部品のカテゴリをブロックと呼ぶ LINE entry では関数と呼びますが、ユニットあるいはモジュールと考えると分かりやすいかもしれません。02～05 では値は与えず部品をまとめて扱うために、06～08 では値を与えて再帰呼び出しを行うために、そして 15 では「与えた値（=被除数）の素数判定結果を“戻り値”として得る処理」を分離して処理の流れを簡明にするために使っています。

再帰呼び出し／再帰関数(recursive call/recursive function)：
処理の過程で自分自身を呼び出すことです(処理中に自分自身を呼び出す関数が再帰関数)。処理が無限に続くとメモリを使い切ってしまうので「呼び出し終了条件」の適切な設定が必要です。

CC BY-SA 2.0(クリエイティブ・コモンズ 表示－継承 2.0)：
表示（適切なクレジットの表示）と継承（改変作品などの同一ライセンスでの配布）を条件として共有（コピーや再配布）と翻案（改変・加工作品の作成）が営利を含め目的に制限なしで可能というライセンス方式です。

素因数分解（prime factorization）：
整数をその素因数（＝その整数の約数である素数）の積で表現することです。

デフォルト（default）
特に指定しない場合に使われる「標準的な選択肢」です。

引数（argument）と**戻り値**（return value）
「処理開始時に関数に渡す値」と「処理終了時に関数が返す値」です。

部品型（プログラミング）言語（visual programming language）：
文字で記述する一般的なプログラミング言語とは異なり、特定機能を持つ部品を連結してプログラムを作成する言語を本冊子では部品型（プログラミング）言語と呼びます。スクラッチ（Scratch）が有名ですが、国産のプログラミングゼミや LINE entry、英国製の Kano Code、中国製の mBlock などもあります。

フラクタル（fractal）：
部分が全体に相似している自己相似の状態で、コッホ曲線、樹木曲線、バーンスレイのシダ、メンガーのスポンジなどが有名です。06～08 で描画する図形（シェルピンスキーの三角形／ガスケット）は有限段階の相似なので「近似」に留まります。

メリさんの羊（Mary Had a Little Lamb）：
合衆国の童謡。新聞記者を集めた蝋管式蓄音機の発表会（1877 年）でエジソン自身が歌って録音・再生して見せたそうです。

01　部品型言語には「国産」もあり offline での使用も可能

義務教育への導入も始まり、プログラミングへの関心が近年高まっています。そして、文字によるプログラミングが困難な年少者の段階などでは、様々な機能の部品（≒ブロック）を組み合わせてプログラミングを行う部品型の言語が広く用いられています。

代表的な部品型プログラミング言語といえば MIT メディアラボのスクラッチ（Scratch：2006～）ですが、あれこれ調べてみたところプログラミングゼミ（DeNA：2014～）、LINE entry（LINE：2019～）などの「国産」もありました。筆者はこれまで JavaScript などの文字型言語で統計処理用の tool などを作ってきましたが､部品型言語は未経験。そこで、３つの部品型言語で音楽・線画・計算の各領域の簡単なプログラムを作ってそれぞれの使い勝手、特徴などを検討してみることにしました。

部品型言語では online（ネット接続状態）のブラウザ上での実行が近年の流行のようですが、実は offline（ネット接続なし）でのプログラムの作成・実行・保存も可能です。そこで、まず以下の URL から筆者が使用中の 64bit Windows 7 PC へのインストール用ファイル(exe や zip)を download しました（資料 01 参照）※。

※Scrach 3 は公式には「Windows 10 以降に対応」ですが 7 でも動きます（Android 用、macOS 用、ChromeOS 用もあります）。プログラミングゼミには macOS 用や Android 用が、LINE entry には 32bit Windows 用や macOS 用があります。使用中の OS に適合するものを選んで download してください。

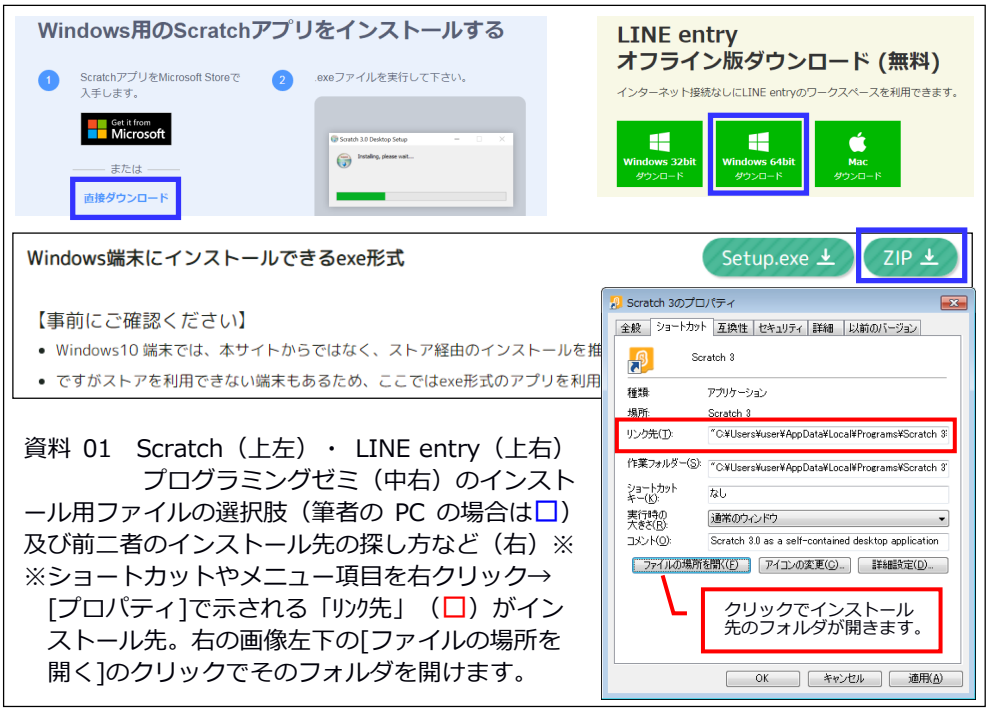
Scratch： ― または ― の下の「直接ダウンロード」をクリック（約 150MB）
https://scratch.mit.edu/download

プログラミングゼミ：「Windows 端末‥」の右の[ZIP↓]をクリック（約 100MB）
https://programmingzemi.com/download_pc.html

LINE entry：[Windows 64bit]をクリック（約 480MB←かなり大！）
https://entry.line.me/policy/download

Scratch と LINE_entry は Setup.exe のWクリックでインストールされます。プログラミングゼミは zip（圧縮）ファイルのWクリックで示されるフォルダ※を drag&drop（以下、D&D）でハードディスクや USB メモリなどに展開すればＯＫです。
※ProgrammingWin32_2021‥（以下は日付など）

補足：Scratch と LINE entry もポータブル使用が可能です。資料１下部の情報も参照しつつ、Scratch 3 及び LINE_entry_offline フォルダを USB メモリなどに D&D でコピーすれば、フォルダ内の Scratch 3.exe や LINE_entry_offline.exe のWクリックで各言語が起動します（資料 13 参照）。ちなみに、本冊子の内容は全て Windows 7 PC にセットした USB メモリから起動・実行した、つまりポータブルで使用した各言語について確認したものです。



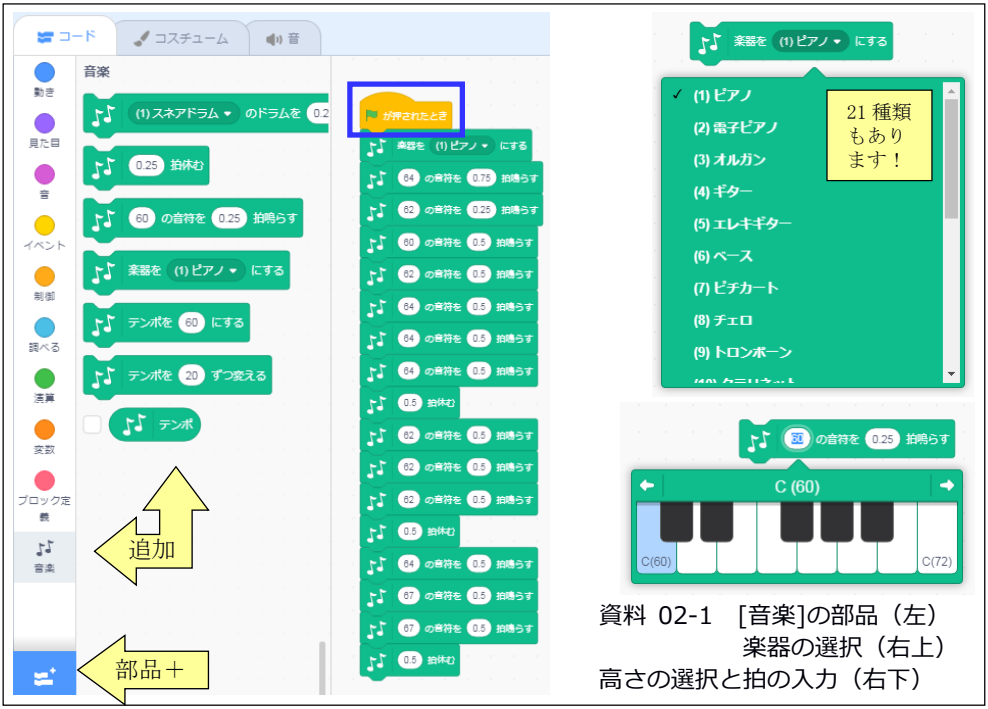
資料 01　Scratch（上左）・ LINE entry（上右）
プログラミングゼミ（中右）のインストール用ファイルの選択肢（筆者の PC の場合は□）及び前二者のインストール先の探し方など（右）※
※ショートカットやメニュー項目を右クリック→[プロパティ]で示される「ﾘﾝｸ先」（□）がインストール先。右の画像左下の[ファイルの場所を開く]のクリックでそのフォルダを開けます。

02　Scratch で「メリさんの羊」：繰り返しとブロック作成

年少者へのアピールを意識してか、部品型言語では音や線画が扱いやすいことが多いようです。ということで、まずは Scratch で「メリさんの羊」をプログラムしてみます。

ところが、最初に起動した状態では、資料 02-1 の左端の[コード]には[音楽]のカテゴリはありません。そこで、[コード]の下端の「部品＋」の青色部分のクリックで示される「音楽」をクリックして追加すると、左から２列目の音楽の部品が使用可能になります。

Scratch では[イベント]（＝出来事）のどれか（今回は「旗が押されたとき」）を「開始の合図」として画面中央の上部に置き、その下に部品を連結していくのが基本です。楽器の種類、音の高さと長さ（拍）などは部品の中の楕円部分をクリックして選択あるいは入力します。

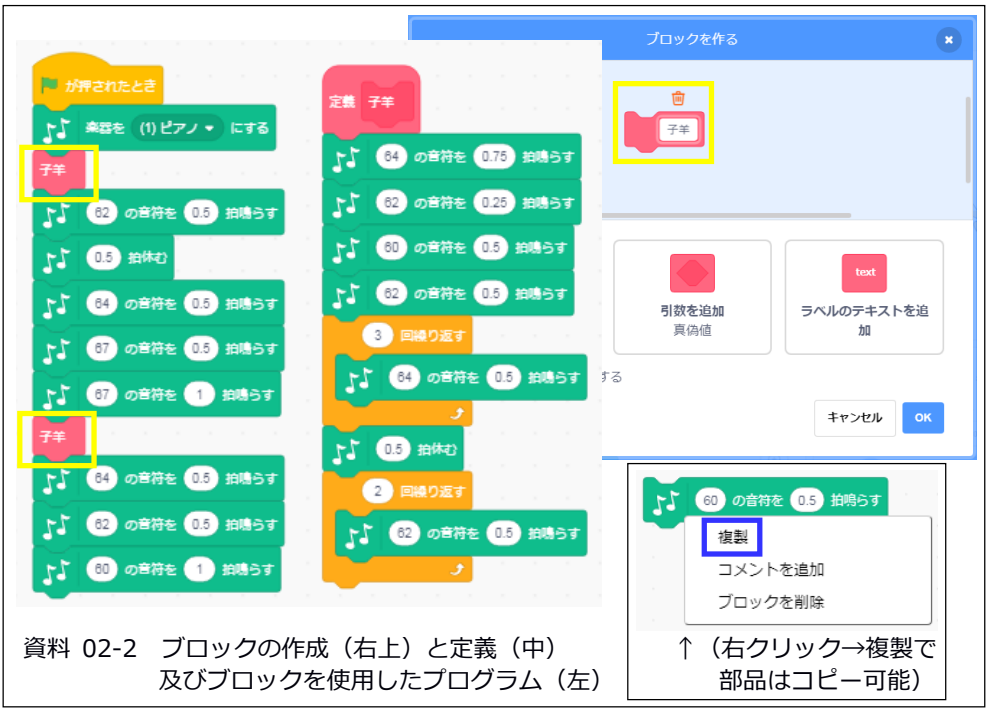


資料 02-1　[音楽]の部品（左）
楽器の選択（右上）
高さの選択と拍の入力（右下）

この指定で旗や連結した部品をクリックすると確かに演奏がなされるのですが、曲の前半だけで既に中央のプログラミングエリアは一杯※。そこで、省スペースの工夫として[制御]の[○回繰り返す]と[ブロック定義]（≒関数）を使います。

※この辺りで一度[ファイル]→[コンピューターに保存する]しておきます。

「ミレドレミミミレレ」をブロックAにすれば、メリさんの羊はA＋レミソソ、A＋ミレドです。そこで、[ブロック定義]→[ブロックを作る]→ブロック名（例：「子羊」）を入力して[OK]で[定義　子羊]という赤いブロックが画面中央に追加されます。その下にブロックの内容を繰り返しや複製も使いながら移し、ブロック＋反復されない音符だけにした結果は資料 02-2 のとおりです。これでかなりコンパクトになりました。



資料 02-2　ブロックの作成（右上）と定義（中）
及びブロックを使用したプログラム（左）

↑（右クリック→複製で部品はコピー可能）

03　Scratch で「メリさんの羊」：メッセージでキャラ連携

Scratch のプログラムは特定のスプライト（＝妖精：以下、キャラ）を対象にしています。つまり、02 で作った「メリさんの羊」は（デフォルトキャラの）ネコを対象としたプログラムです※。
※キャラを削除するとそのプログラムも消えてしまうので要注意。

そこで、プログラムの工夫の一例として、伴奏をする別キャラからネコにメッセージを送って音楽を始めさせる「メリさんの羊：伴奏付」を作ってみます。

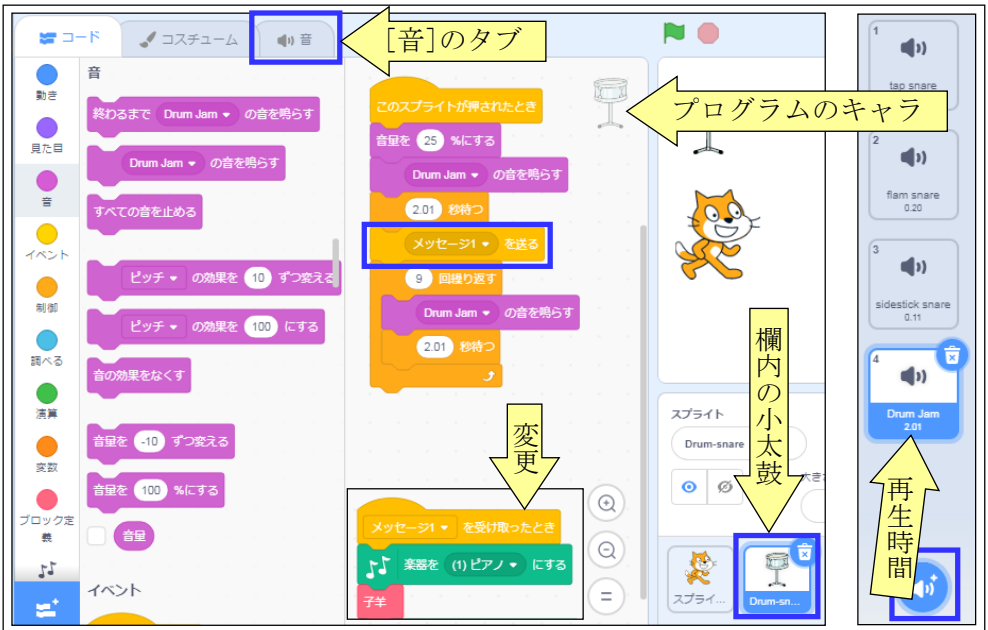
画面右下の[ネコの顔＋]→[虫眼鏡]でキャラ（今回は Drum-snare＝小太鼓）を選択＝追加します。そして、画面右下の欄内の小太鼓をクリックすると画面中央は（デフォルトのネコ用から）小太鼓用のプログラミングエリアに切り替わります（資料 03 参照）。

そこに「このキャラ（＝スプライト：小太鼓）のクリックで開始、（伴奏なので）音量は 25%、音は Drum Jam、１回鳴らしたところでネコに[メッセージ 1]を送る‥」を内容とする部品を組み立てます（資料 03 の上の中参照）。Drum Jam は（最初は選択できないので）[音]のタブ→その画面の左下の[音＋]→打楽器やループなどで絞り込んで Drum Jam のクリックで追加してから選択します（資料 03 の下など参照）。[2.01 秒待つ]が必要な理由は、Drum Jam の再生時間＝2.01 秒だけ待たないと、再生が終わる前に次の部品が始まってしまうからです。

資料 03 の中央下部に示したようにネコの方のプログラムの開始の合図を「メッセージ 1 の受け取り」に変更して小太鼓をクリックすると音量 25%の Drum Jam が 1 回→続いて 9 回反復される Drum Jam を伴奏とした「メリさんの羊」の演奏が行われます。

Scratch は 2006 年の公開以来、2013 年の 2.0、2019 年の 3.0 と改訂を重ねてきており、2021 年の時点で全世界の登録ユーザーが 6000 万人以上（日本国内は約 80 万人）と、長い歴史と厚いユーザーを持つ安定・充実した部品型言語です。

資料 03 の下部にその一端を示したように標準添付の音やキャラなどの素材も多数かつ良質なので、それらを組み合わせるだけでも当分は楽しめそうです（‥[音]の選択肢は 300 以上あります！）。



資料 03　追加したキャラクタ（右）とその部品（中）及びネコの部品の変更（中下）

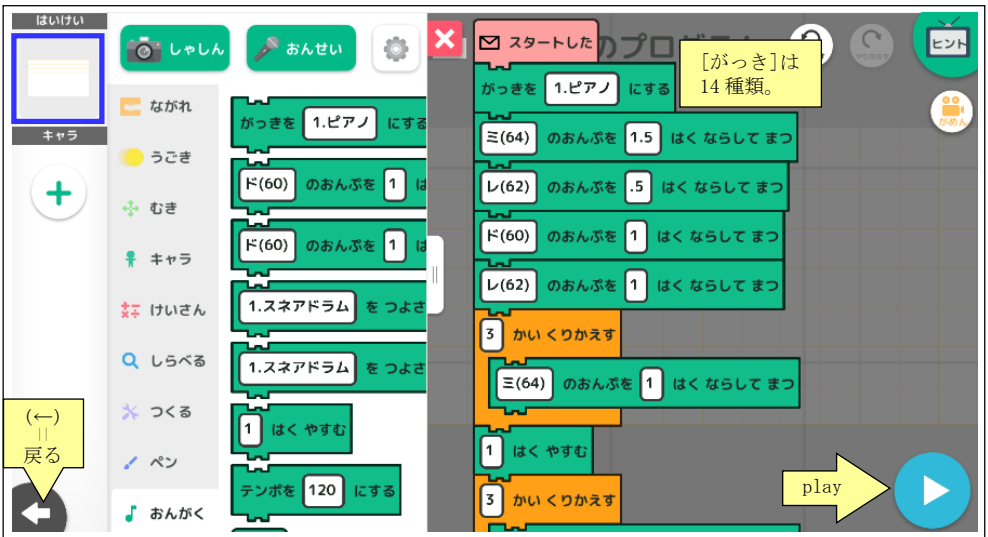
[音]のタブ→音＋→Drum Jam のクリックによる音の選択肢の追加（右）

04　プログラミングゼミで「メリさんの羊」

Programming.exe（資料 13 参照）のWクリックなどでプログラミングゼミを起動すると初回のみニックネーム※の登録を求められます。適当な名前を入力して[ＯＫ]→「設定の確認」で[はい]→レベルを 999（＝全機能使用可能）にして[ＯＫ]します。
※プログラムを保存するとファイル名の先頭にこの「名前」が付きます。

「はじめてのプログラム」や「おすすめ」は左下隅の(←)や(×)で閉じ、[あたらしくつくる]※→[じゆうにつくる]と進んで画面左上の[はいけい]をクリック＃→[てがみ]から始まるカテゴリ欄を上に drag して一番下の[おんがく]をクリックすると必要な部品が表示されます。
※２回目以降の起動時には[じぶんのさくひん]が選べます。
＃プログラミング（部品の組み込み）は[はいけい]に対しても可能です。

資料 02-1 と同等のプログラムを作ってみると、デフォルトサイズでは部品が大きいため「くりかえす」を使っても画面に収まりません※。
※部品の表示サイズはパッド右端のスワイプで変更可能です。なお、画面右下の[play]のクリックに対応する[スタートした]は[てがみ]のカテゴリにあります。

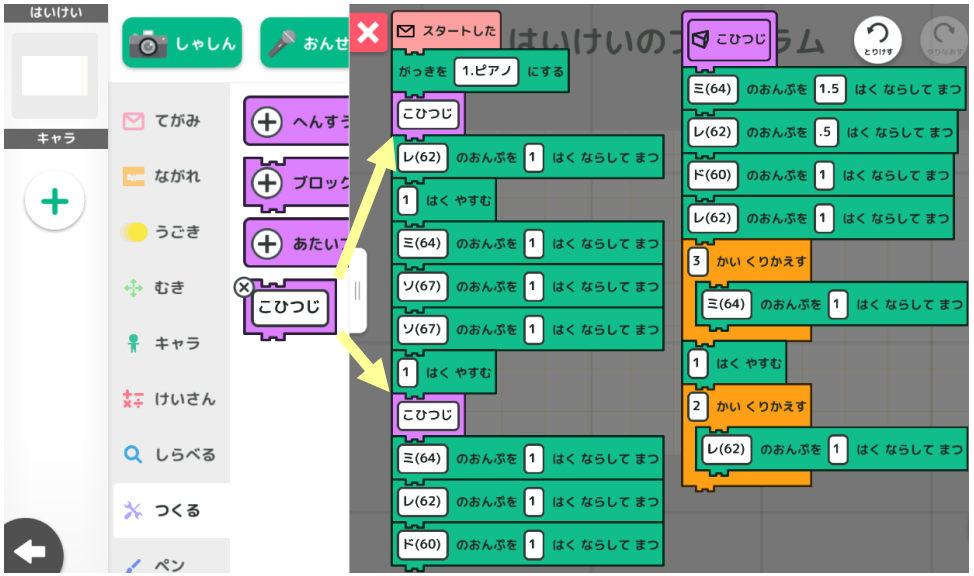


資料 04-1　[あたらしくつくる]→[じゆうにつくる]での最初の部品配置など

そこで資料 02-2 と同様にブロック（≒関数）を利用します。[つくる]→[ブロックをつくる]→[＋もじ]と進み、「もじ」に名称（例：「こひつじ」）を入力して[ＯＫ]で[こひつじ]ブロックが画面に追加されます。[こひつじ]ブロックの下に共通部分の部品を連結して定義し（‥コピーは長押しで）、[スタートした]以下に左の欄から D&D した（白抜きの）[こひつじ]とその他の部品を連結して完成したプログラムは資料 04-2 のとおりです。ブロックを使ったこのプログラムは（表示サイズを若干縮小したこともあり）画面に収まっています※。

※３言語とも部品の背景にはカテゴリのマークと同じ色が使われています。

左下隅の(←)を２回クリックすると[じぶんのさくひん]画面になります。今回のプログラムは「さくひん 1」として自動保存されており、作品の右クリックか歯車のクリック→[だいめいをかえる]で名称変更が、[おくる]で任意の場所への保存が可能です。保存したプログラム（ファイル名.prozemiproj）は画面への D&D で読み込めます。



資料 04-2　新しい[ブロック]（こひつじ）を定義・使用したプログラム

05　LINE entry で「メリさんの羊」

プログラミングゼミの[てがみ]にも「メッセージ」関連の部品があるので機能的には伴奏も可能そうですが、（効果音は 20 ほどあるものの）Scratch の Drum Jam のような伴奏向きの素材が見当たらない※ため割愛して LINE entry でのプログラミングに進みます。※14 を参照

LINE_entry_offline.exe のWクリックなどで LINE_entry を起動すると Scratch のそれとよく似た画面が現れ、クマを歩かせるプログラムの例が表示されます（[play]のクリックで作動）。[ブロック]のタブの[音]をクリックすると 9 つの部品がありますが、最初は音が[対象なし]なので[音]のタブ→[音を追加]と進んで必要な音（今回はピアノのドレミソ）を選択→[追加する]のクリックでこれらの音が選択できるようになります※。音の長さ指定は「秒」ですが、「拍」という変数を作成して基本値を 60 などとし（30 / [拍]の値）秒とすれば「拍」による全曲一貫指定及び変更も可能です（資料 05-2 の左下参照）。
※音階を指定できる楽器はピアノとマリンバの 2 種類のみ。

資料 05-1　追加した音の長さを秒で指定した曲前半の部品配置など

資料 05-1 に示したように、前半のみで画面の大部分を使ってしまうので、[ブロック]のタブの[関数]→[関数を作成する]で共通部分を「メリさんの羊」という名前の関数（ユニット≒モジュール）にします※。
※他の２言語とは異なり、関数は灰色の別レイヤ(layer)=層≒画面で定義します。

元の画面から共通部分のみをコピペした関数が資料 05-2 の左側（灰色レイヤ‥背景に元の部品が透けて見えます）、関数[メリさんの羊]を利用した全曲のプログラムが右側です※。
※部品などのコピペは右クリックなどで異なるレイヤ間でも可能です。

完成したらファイル関連アイコン（資料 05-1 上の右）で保存※しておきます（‥その右の矢印は編集の undo=取り消しと redo=その逆）。
※[保存する]は（左の枠内のファイル名での）「上書き保存」で、[コピーとして保存]が「名前を付けて保存」です。↑今回の例では「メリさんの羊：5-1」

別レイヤでの関数定義、右クリックや[Cntl]+[C]/[V]でのコピペ、ゴミ箱の常時明示などは「使いやすい工夫」と感じました。

資料 05-2　別レイヤでの関数定義（左）と関数を利用した全曲のプログラム（右）
<－－－－－及び「拍」の利用例（左下）

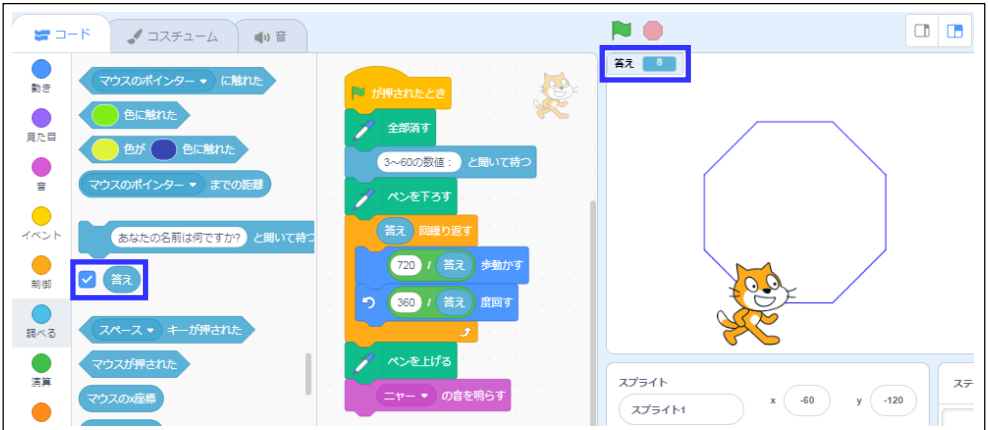
06　Scratch で正多角形と近似フラクタル：再帰呼び出し

LINE entry の[開始]には「信号」関連の部品があるので機能的には伴奏も可能そうですが、（効果音は 100 ほどあるものの）伴奏向きの素材が見当たらないため、割愛して線画のプログラミングに進みます。

Scratch を起動し、「旗が押されたとき」の下に部品を連結していきます。線による描画のため、[コード]の下端の「ブロック＋」の青色部分をクリックして追加した「ペン」のカテゴリから[全部消す]、[ペンを下ろす]、[ペンを上げる]※の部品を追加します。
※[ペンを上げる]は使わなくてもＯＫですが‥。

角数が選べるように[調べる]から[○と聞いて待つ]を選んで質問を「3～60 の数値：」とし、[答え]に基づいて線分の長さを「720/[答え]」※、角度を「360/[答え]」として[答え]回繰り返すようにします。
※「歩動かす」の単位はピクセルです。

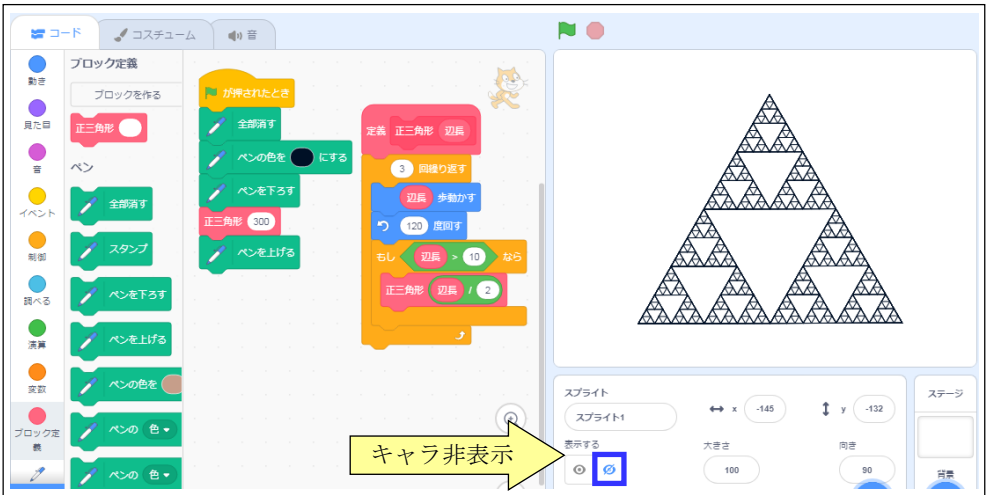
旗を押し、質問に半角数字の 8 を入力して[Enter]すると、ネコが資料 06-1 の右側に示した正八角形を描いて「ニャー」と鳴きます※。
※コードの左から２列目中ほどの[答え]がチェックされているため、実行画面に入力値（今回は 8）が表示されています（位置は D&D で移動可能）。



資料 06-1　指定された角数の正多角形を描くための部品（中）と実行結果（右）

発展課題として、関数の中でその関数を呼ぶ再帰呼び出しを使って近似フラクタルの描画をしてみます。02 と同様、[ブロック定義]→[ブロックを作る]→ブロック名（「正三角形」）を入力し、今回は[引数を追加]※をクリックして[number or text]に「辺長」と指定して[OK]します。※変数「辺長」で渡した引数の値に従って処理が行われます。

資料 06-2 の中ほどに示したようにブロック「正三角形」は与えられた辺長の正三角形を描く関数ですが、その中に「辺長が 10 より大きい」なら辺長を半分にして自分自身を再実行する指定をします。するとまず辺長 300 の 1 辺、次に 150、次に 75、次に 37.5、次に 18.75 の辺が角度を変えながら描かれ、18.75 の半分=9.375≦10 なのでそれ以上の再帰呼び出しはされずに残り 2 辺が描かれて最初で最小の正三角形になります。一番深い再帰呼び出しの繰り返しが終わるので辺長は一つ前の 37.5 に戻り、それを 1 辺とする正三角形の中が 1 辺 18.75 の正三角形で埋められ‥という循環への出入りを繰り返して、最終的には資料 06-2 に示した作図（シェルピンスキーの三角形／ガスケット）がなされます。



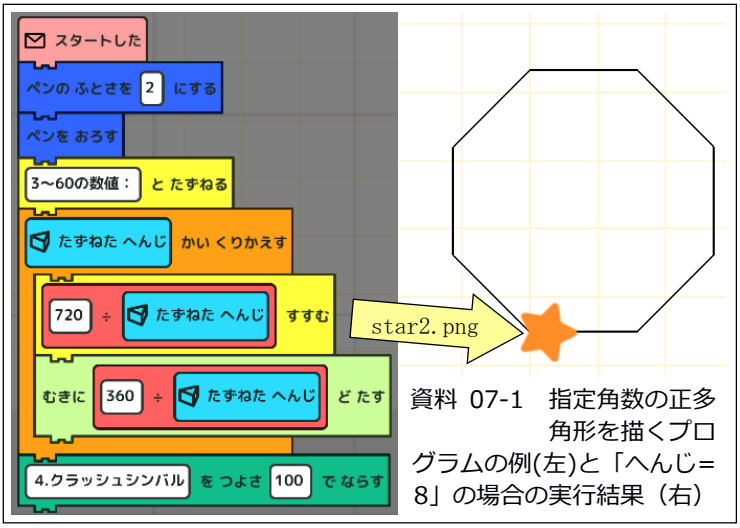
資料 06-2　関数の再帰呼び出しを利用した描画プログラム（中）と実行結果（右）

07　プログラミングゼミで正多角形と近似フラクタル

プログラミングゼミを起動したら（04 で設定した）自分のニックネームを選択し[あたらしくつくる]→[じゆうにつくる]と進みます。線で描画するにはキャラ（＝キャラクタ）が必要なので[キャラ]の下の＋をクリック→[しゃしんをえらぶ]の右下の風景アイコンをクリックして適当なイメージを選択→[ＯＫ]で追加します※。

※今回はプログラミングゼミの custom フォルダ内の characters から star2.png を選択しました（資料 07-1 参照）。

追加したキャラは、それをクリックして示される部品を組み立てたプログラムを、それが[はいけい]上に D&D された位置から実行します。



資料 07-1　指定角数の正多角形を描くプログラムの例(左)と「へんじ=8」の場合の実行結果（右）

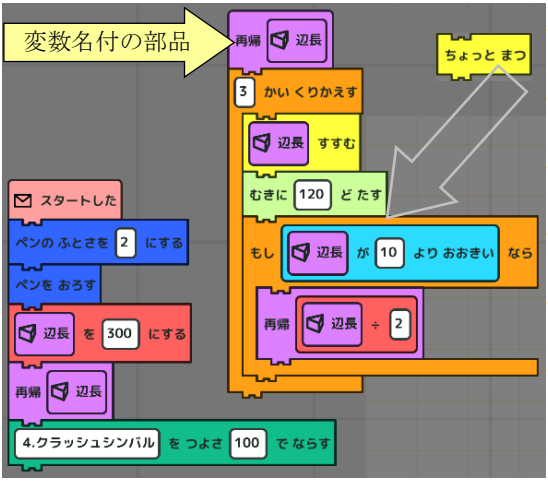
資料 06-1 と同等のプログラム及び実行結果の一例は資料 07-1 に示したとおりです。[〇とたずねる]は[うごき]カテゴリに、[たずねた へんじ]は[しらべる]カテゴリにあります。

「再帰」を使って近似フラクタルを描画するには、[つくる]→[ブロックをつくる]で[＋ラベル]をクリックしてブロック名（例：再帰）を、[＋へんすう]をクリックして変数名（例：辺長）を指定して[ＯＫ]で引数付の部品を作り※（06 と同様）その下に「辺長＞10 ならその半分

の値を引数として渡して自分自身を実行する」ように部品を連結します＃。
※部品欄の下部には上下につなぎ目の付いた変数白抜きの部品が、プログラムエリアにはその部品＝ブロックの働きを定義する変数名付の部品が追加されます。
＃変数[辺長]は変数名付の部品からの D&D で白抜き部分に paste できます。

[スタートした]の方（＝本体プログラム）の[ペンをおろす]以下には[辺長]を 300 にする部品、上下につなぎ目の付いた引数付の部品（白抜き部分には[辺長]を D&D）、終了の合図の音（クラッシュシンバル）の部品を連結します。以上のプログラムと実行結果は資料 07-2 に示したとおりで、Scratch では描画に 30 秒程度かかったのに対し、描画時間は一瞬（１秒未満）でした※。
※描画過程を確認したいなら[もし]の前などに[ちょっと まつ]を追加しましょう。

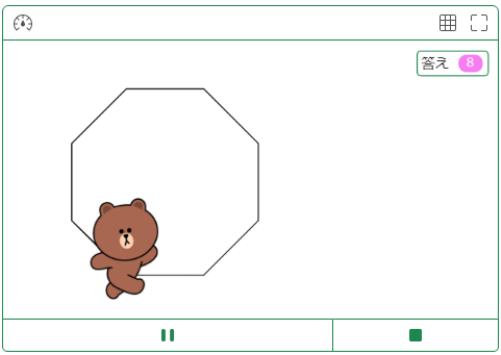
終了時に自動保存される「さくひん○」は右クリックなど→[だいめいを かえる]で名称変更が、[おくる]で任意の場所への任意ファイル名での保存が可能で、保存したプログラム（ファイル名.prozemiproj）は画面への D&D で読み込めます。



資料 07-2　再帰関数を利用した描画プログラムと実行結果

08　LINE entry で正多角形と近似フラクタル

LINE entry を起動し、「スタートボタンをクリックした時」の下に部品を連結していきます。描画画面の縦方向がやや狭いため線分の合計を 720 から 600 にし、回転方向が逆なので 360 を-360 とした以外はこれまでのプログラムと同様です※。実行・確認したら、正多角形.ent などのファイル名で一旦保存しておきます。
※[クマの鳴き声]もありますがリアル過ぎるので[ホイッスル]にしました。



資料 08-1　指定角数の正多角形を描くプログラム（右）と実行結果（上）

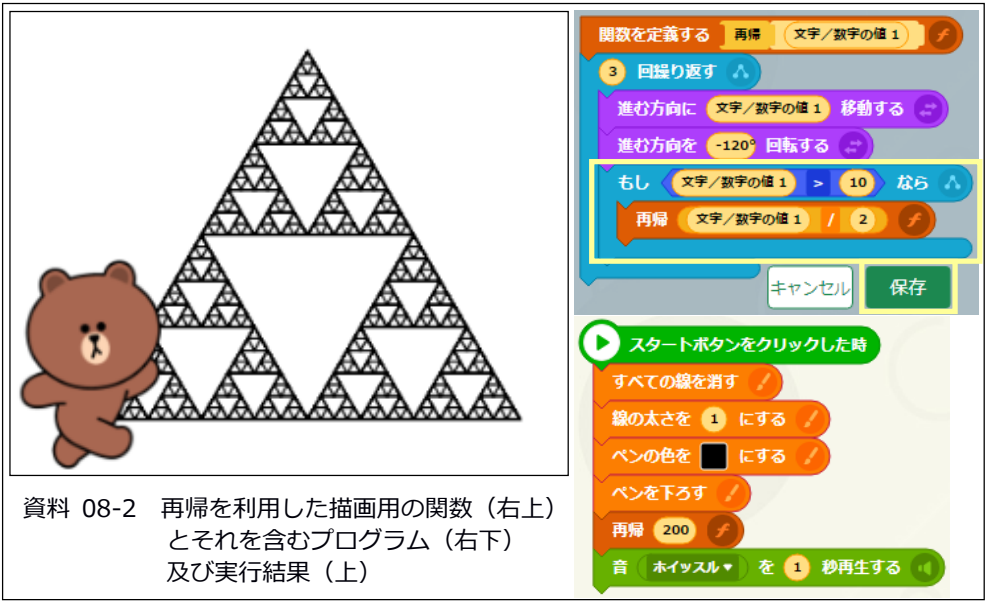
次に「再帰」を使って近似フラクタルを描画します。不要になる[答えを待つ]と[答え]を削除し、関数にする「繰り返す」の部分を右クリックなどでコピー（＝メモリに保存）しておきます。

[関数]→[関数を作成する]のクリックで灰色レイヤが開くので[関数を定義する]の部品の[関数]に適切な名称（例：再帰）を記入し、その後に[文字/数字の値]のピースを D&D で結合します※。その下に、コピーしておいた「繰り返す」を右クリックなどで貼り付けて連結し、回数を 3、移動を[文字/数字の値１]＃、回転を-120°に変更します。
※カチッと音がして[文字/数字の値１]になればＯＫです。
＃[定義する]ブロックなどから右クリックで複製して D&D します。

念のため、ここで一旦[保存]（＝関数の保存）し、[ペンを下ろす]と[音]の間に[再帰]の部品を連結したら 10 を 200 にして（資料 08-2 の右下参照）実行してみます。大きい△が１つ描かれればＯＫです。

[再帰]の部品の[繰り返す]中の[回転する]の下に[制御]カテゴリから[もし<正しい>なら]を D&D して条件を<(文字/数字の値１)＞10>※とし、その中に[関数]から[再帰]の部品を D&D して引数を(文字/数字の値１)/2＃にして（関数を）[保存]します（資料 08-2 の右上）。
※[正しい]の上に[判断]カテゴリから<10＞10>を D&D し[文字‥１]で入れ替え。
＃[計算]ブロックから(10 / 10)を D&D し[文字‥１] / 2 に入れ替え及び入力。

このプログラムを実行すると資料 08-2 の左側に示した描画が（筆者の環境では Scratch よりやや早く 20 秒程度で）行われます。なお、プログラムの保存時に[保存する]を選ぶと資料 08-1 のそれ（正多角形.ent など）に「上書き」されてしまうので[コピーとして保存]を選んで別名（例：近似フラクタル.ent）で保存しましょう。



資料 08-2　再帰を利用した描画用の関数（右上）とそれを含むプログラム（右下）及び実行結果（上）

09　Scratch で素因数分解：より効率的な処理の工夫

音や線画だけでなくコンピュータ＝計算機らしいプログラムとして素因数分解をしてみます。そのためには「正の整数 y1 の入力を求め、2 から 1 ずつ増やしながら値 x で y1 を割っていって割り切れたらその x を x1、商（y1/x1）を y2 として後者を再度 x で割っていく」という手続きを「約数がありうる範囲」で繰り返し、約数がなくなったら y1＝x1×x2×‥×1 のように示すプログラムを組めばＯＫです。

資料 09-1 は最も素朴に 2～各 y までの範囲を検討するプログラムの一例です。まず、[変数]カテゴリ→[変数を作る]で[結果]、[被除数]、[除数]の３変数※を作り、入力を求める[○と聞いて待つ]は[調べる]から D&D。[変数]の[□を○にする]を使って[被除数]に[答え]＝y1 を、[結果]に(＝)を代入します。
※左の□チェックで内容を画面表示。

主な処理は[制御]の[く　>まで繰り返す]及び[変数]の[○にする]と[○ずつ変える]で行い、割り切れるたびに[結果]に[除数]の値と(×)を書き足して商（＝被除数/除数）の値を被除数に代入します。

[被除数]=1 まで繰り返したら[見た目]カテゴリの[○と言う]でキャラに[答え]＝[除数]×[除数]×‥×[被除数]（=1）という文字列で出力させます※。
※例えば 223092870 を処理すると、瞬時に 2x3x5x7x11x23 に素因数分解します。

資料 09-1　素朴で非効率なプログラムの例

ここで「素因数分解 0」などのファイル名で保存しておきます。さて、実はこのプログラムには大きなムダが含まれています。注目すべきは「約数がありうる範囲」。例えば 100 の約数は 2, 4, 5, 10, 20, 25, 50、200 の約数は 2, 4, 5, 10, 20, 40, 50, 100 というように、約数は元の値の平方根の上下で対を成します。従って、2～√y までに約数が無ければ y は約数を持たない素数であり、それ以上の値は検討不要なのです。

そこで[除数]の値が√[被除数]を超えた時点で[被除数]を[除数]に代入します（‥「+1 しながら最後まで検討した状態」と同じ）。[被除数]＝[除数]なので余り＝0 で内側の繰り返しが終了。[結果]にその時点での[除数]（＝[被除数]）と×が追加され、[被除数]に代入される[被除数]/[除数]の値が 1 のため外側の繰り返しも終了し、その時点の[結果]と[被除数](=1)が出力されます（資料 09-2 の□部分参照）。

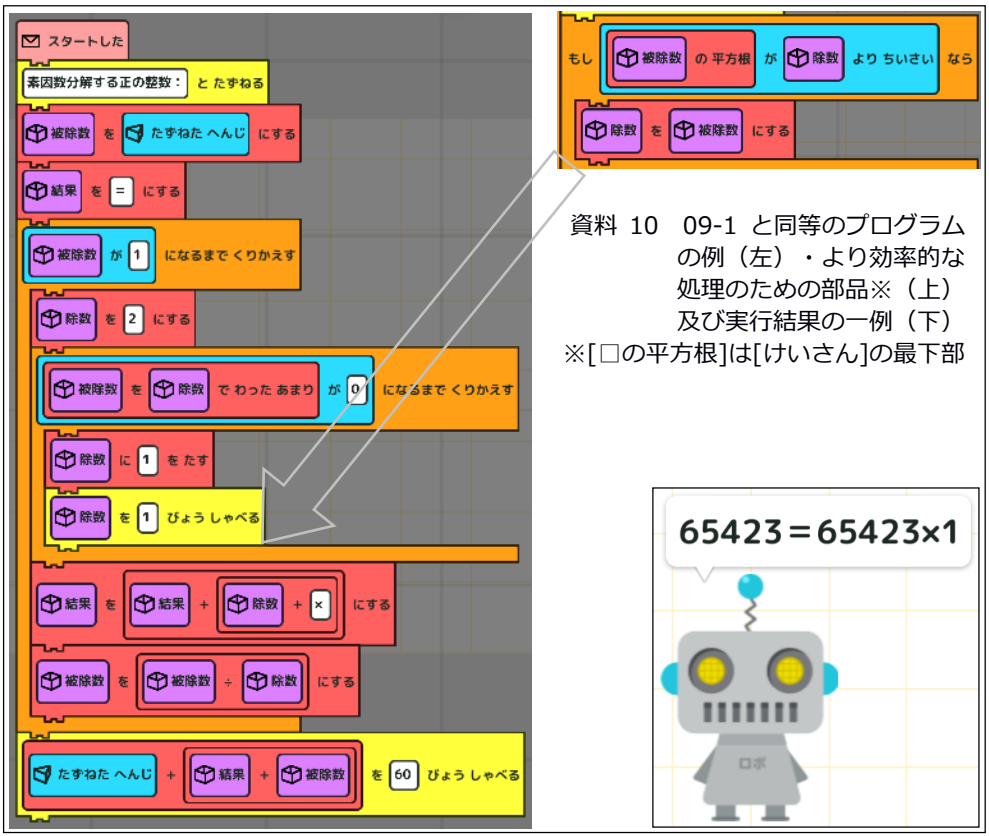
□部分を追加したプログラムを「素因数分解 1」などとして保存し、まずは先の素因数分解 0 で 5555567 という 7 桁の素数を分解してみると筆者の環境では 5555567＝5555567×1 と出力されるまでに 20 秒ほどかかりました。他方、素因数分解 1 では同じ結果（資料 09-2 の右側）が 1 秒未満で得られ、少しの工夫で格段に効率的になっています。

資料 09-2　追加した部品（左の□）と実行結果（右）

10　プログラミングゼミで素因数分解

07 と同じ手順で適当なキャラ（例では robot.png）を登録します。「たずねる」や[しゃべる]は[うごき]から、[にする]や[たす]などは[けいさん]から、[くりかえす]は[ながれ]から取り出し、[つくる]→[へんすうをつくる]で作った[被除数]・[除数]・[結果]と組み合わせた 09-1 と同等のプログラムは資料 10 の左に示したとおりです。

キャラを背景上に D&D して右下隅の[play]（資料 04-1 参照）をクリックし「素因数分解する正の整数：」に適当な数字を入力して試してみましょう。09-1 と同等のこのプログラムでは、例えば 65423 という５桁の素数判定に 20 秒ほどかかります※が、資料 10 右上の２部品を追加すれば結果は１秒未満で得られます。※速度は Scratch の約 1/100。



資料 10　09-1 と同等のプログラムの例（左）・より効率的な処理のための部品※（上）及び実行結果の一例（下）
※[□の平方根]は[けいさん]の最下部

11　LINE entry で素因数分解

「スタートボタンをクリックした時」の下に部品を結合していきます。[データ]から[〇と言って答えを待つ]を取り出し、[変数を作成する]で[被除数]、[除数]、[結果]の３変数を作ります＃。
＃[結果]と[除数]に基本値（＝と 2）を指定しておけば☆の２行は省略可。

[表示]の[〇を言う]を取り出して〇の部分に[データ]から取り出した(□の値)を配置し、[制御]の[〈正しい〉になるまで繰り返す]の中に必要な処理を組み込みます（資料 11 参照）。

プログラムができたら[play]（資料 05-1 参照）をクリックし「素因数分解する正の整数：」に適当な数字を入力して試してみましょう。09-1 と同等のこのプログラムでは、例えば 1213 という４桁の素数判定に 20 秒ほどかかります※が、資料 11 右上の２部品を追加すれば右下の結果が１秒程度で得られます＃。※速度は Scratch の約 1/4000。
＃09～11 では、各プログラムの処理速度を検討するためにワザと素数（5555567、65423、1213）を処理させています。例えば素数ではない 111546435 なら、いずれの非効率なプログラムでもほぼ瞬時に＝3x5x7x11x23x1 と素因数分解します。

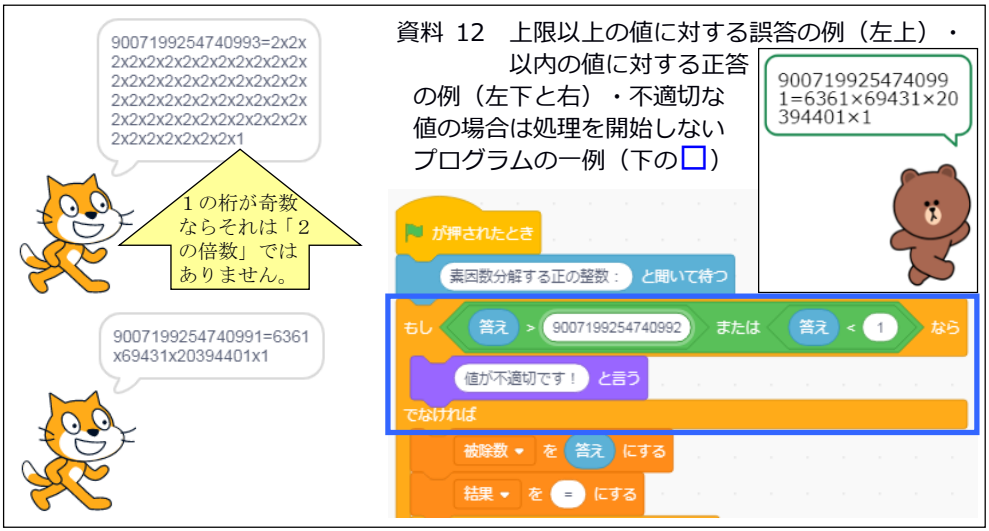
資料 11　09-1 と同等のプログラムの例（下）

12　各部品型言語の数値的制約と誤答させない対策の例

意外かもしれませんが、プログラミング言語には「処理できる値の上限」があります（例：JavaScript なら 2 の 53 乗=9007199254740992）。そこで、それを超える 9007199254740993 を各部品型言語のプログラムで因数分解させてみた結果は以下のとおりでした※：

①Scratch：2x2x‥x2x1 という誤答を返す（資料 12 の左上）
②プログラミングゼミ：同上の誤答（＋出力が画面からはみだす）。
③LINE entry：9007‥0993 を入力しても 9007‥0992 として処理する。
※ちなみに、9007199254740991 なら Scratch は 6361×69431×20394401 という正答を約 1 秒で返します（資料 12 の左下）。プログラミングゼミはやはり 2x2x2x‥という誤答を返します。LINE entry は正答しますが処理完了に約 20 分を要します（資料 12 の右）。

Scratch と LINE entry の上限は 2 の 53 乗で、プログラミングゼミのそれはもっと低いようです。「そんな極端な値を指定する方が変」という良識を期待したい所ですが「誤答を返すのはやはり問題」と考えるなら「正しく処理できない入力は受け付けない改訂」が必要です。Scratch の場合の一例を資料 12 の右下に示しました。[制御]カテゴリの[もしく　>なら　でなければ]を使って 1 未満や 9007199254740992 より大きい[答え]の場合には処理を始めないようにしています。



資料 12　上限以上の値に対する誤答の例（左上）・以内の値に対する正答の例（左下と右）・不適切な値の場合は処理を開始しないプログラムの一例（下の□）

13　試した範囲での各部品型言語の特徴など

これまでに試したのは３つの部品型言語の機能などのごく一部に過ぎませんが、その範囲内での印象は以下のとおりです：

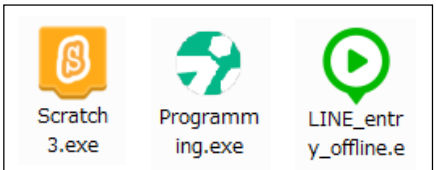
①Scratch：全般的に高機能で「不得意領域」がありません。標準添付の音やキャラなどの素材も多数かつ高品位です。言語を[にほんご]に変更すれば部品などの表記はかな／カナのみにできます。

②プログラミングゼミ：部品の表記はその殆どがかな／カナ、プログラムは自動保存、長押しでコピー、入力画面では 50 音キーボードが現れるなど、年少者のタブレットによる利用を考慮した工夫が盛り込まれています。線画が超高速な一方、処理できる値の上限がやや低めのようです（「常識」的には十分ですが‥）。
なお、キャラ間で部品をコピーする方法と[しゃべる]での出力の改行方法は発見できませんでした（
や¥n は無効＠後者）。

③LINE entry：縦幅が狭い部品、変数の選択指定、右クリックや[Cntl]+[C]/[V]でコピペ、関数は別画面で定義など、PC での利用者が長めのプログラムを書く時に便利そうな機能が盛り込まれています。他方、音階のある楽器が少なく、11 ・ 12 の※で触れたように増分=1 の繰り返し処理は Scratch などと比べてかなり低速でした。

機能以外では、Offline 使用時の容量はプログラミングゼミが約 0.1 GB で最小、Scratch は約 0.3GB、そして LINE entry は約 1.0GB で最大でした。しかし、16GB のメモリスティックやカードが「小容量」の現在、このような属性はもはやさほど重要ではないかもしれません。

ということで、一般的には Scratch が、高速線描が必要／巨大な数値が不要な処理を主にタブレットで行う場合や極力小容量の媒体で作業したい場合にはプログラミングゼミが、そして多様な楽器や高速の繰り返し処理が不要で「Scratch は使いたくない」「容量は不問」といった場合には LINE entry がオススメかもしれません。



資料 13　各言語の起動ファイル

14　プログラミングゼミで大きめの数値を処理（BGM 付き）

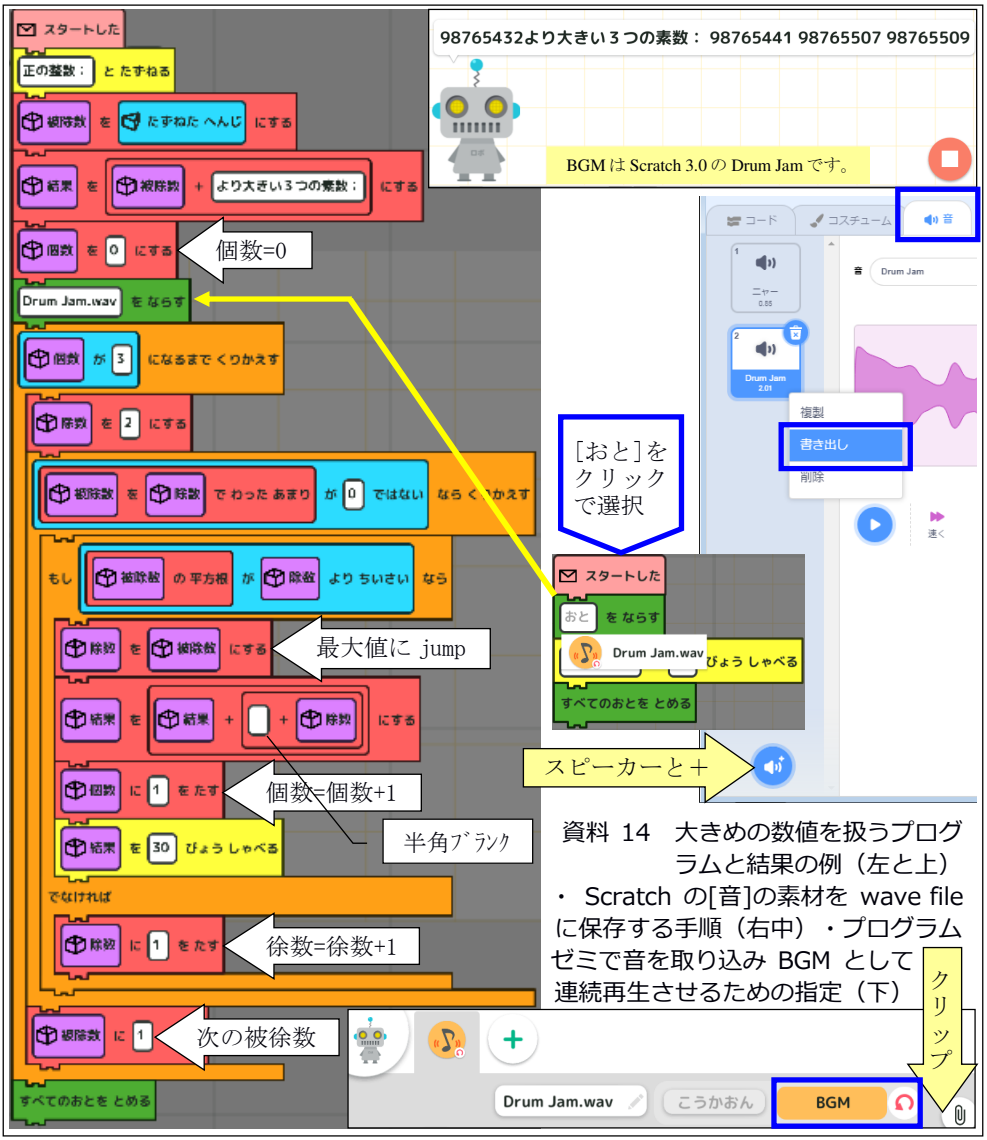
05 で指摘したように LINE entry には「音階付の楽器が少ない」という限界が、また 12 で指摘したようにプログラミングゼミには「数値の上限が低め」という限界がありました。加えて、国産の両部品型言語は Scratch と比較して計算の処理速度が遅く、標準添付の素材も見劣る印象があります（‥自作推奨という事かもしれませんが）。

実は、素材については Scratch のそれの利用が可能です※。また（2 の 53 乗≒9000 兆とかではなく）「常識的な範囲の数値」ならプログラミングゼミでも正しい処理が期待できることでしょう。ということで、Scratch の素材を BGM に使いつつ「常識的な範囲」で大きめの数値を処理するプログラムをプログラミングゼミで作成してみます。
※Scratch の素材は CC BY-SA 2.0 に従えば（≒出典を明示すれば）利用可能。

まず BGM 用に Scratch の素材を file 保存します。Scratch を起動したら[音]のタブをクリック→左下のスピーカーと＋のアイコン（音を選ぶ)をクリック→いずれかの音(今回は[ループ]のカテゴリから Drum Jam）を選択→左の欄のアイコンの右クリックで[書き出し]を選ぶとその音が Drum Jam.wav として保存されます（資料 14 の右中参照）。

次に、プログラミングゼミを起動し、10 で作成したプログラムを右クリック→[かいぞうする]で例えば「BGM 再生」といった名称にします。クリックでロボットのプログラムを開き上部の[おんせい]をクリック→右側のクリップのアイコンをクリック→Drum Jam.wav を読み込み→[BGM]と[回転矢印]＝連続再生をクリック→[ＯＫ]で file が取り込まれ、そのプログラム内での再生が可能になります※。[キャラ]のカテゴリから[□をならす]と[すべてのおとをとめる]を D&D して時間確保用の[□びょう　しゃべる]を挟み、灰色の[おと]をクリック→[Drum Jam.wav]をクリック選択で BGM 再生の設定は完了です。
※「取り込み済み」なので元の wave file は削除などしてかまいません。

処理としては「指定数値より大きい最初の３素数」を求めさせます。プログラムの内容は資料 14 の左に示したとおりで、数値が（それなりに大きい）98765432 の場合、３素数を得るまでの所要時間は数秒程度で常識的には十分な性能と言えそうです（資料 14 の右上参照）。



資料 14　大きめの数値を扱うプログラムと結果の例（左と上）
・ Scratch の[音]の素材を wave file に保存する手順（右中）・プログラムゼミで音を取り込み BGM として連続再生させるための指定（下）

15　部品型言語はスケートボード？

これまで検討してきたように、今回の３部品型言語は音楽・線画・計算といった古典的領域※については（得手不得手はあれ）ほぼ十分な性能を持っていました（13 参照）。

※キャラ接触などに基づく game、カメラや傾きセンサーなどからの入力の利用、ハードウェアの制御といった領域が得意分野のようですが、筆者は関心希薄。

これらの言語は、文字型言語の場合に不可欠な「文法事項」に煩わされること無く、文部科学省のいう「プログラミング的思考」※を利用者がその興味・関心に基づいて試行・体験し習得・上達できる道具となりうる点で、子どもにとっても大人（含高齢者）にとっても有意義な「優れたオモチャ」であると考えます。

※「自分が意図する一連の活動を実現するために、どのような動きの組合せが必要であり、一つ一つの動きに対応した記号を、どのように組み合わせたらいいのか、記号の組合せをどのように改善していけば、より意図した活動に近づくのか、といったことを論理的に考えていく力」
小学校プログラミング教育の手引き（第三版） 2020

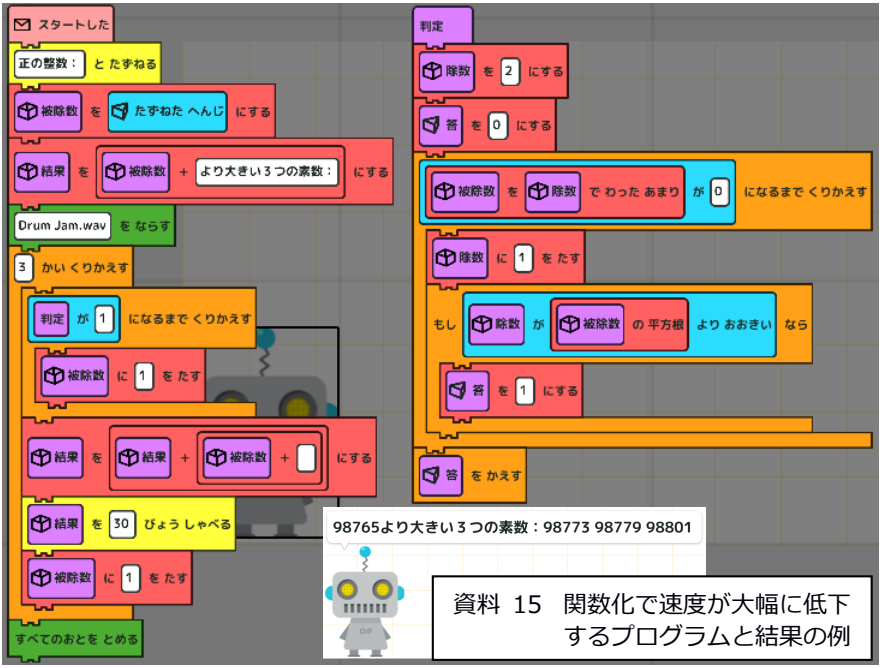
他方「これで実用プログラムを作るか？」と問われたら、筆者の答は「否」です。資料 14 のプログラムの（BGM を除く）実用的機能なら、JavaScript で関数を用いて以下のように記述することでしょう。構成は form１つと function２つで簡明、手続は「素数になるまで被除数に+1」と単純で処理も高速です（出力までに要する時間は約２秒）。では、なぜ

```
<form>正の整数：<input type="text" size="12" value="98765432">
 <input type="button" value="より大きい３つの素数：" onClick="探索()">
 <input type="text" size="36">
</form>
<script>
function 探索(){
 被除数=document.forms[0].elements[0].value;結果=""
 for(i=1;i<4;i++){do{被除数++}while(判定(被除数)==0)
 結果=結果+被除数+" ";document.forms[0].elements[2].value=結果}
}
function 判定(){
 除数=2;答=0
 while(被除数%除数!=0){除数++;if(Math.sqrt(被除数)<除数)答=1}
return 答
}
</script>
```

資料 14 では関数を使っていないのか？　それは（理由は不明ですが）「関数を用いると処理速度が極端に遅くなる」からです（資料 15）※。
※資料 14 の構成なら 1 秒もかからない 98765 の処理に 1 分近くかかります。

部品型言語に限らず、複雑な処理はその流れを辿るのが困難です。実用プログラムには、入力の容易さ・出力の整形などと共に複雑な処理の簡明化に有効な関数の高速処理が不可欠であり、これらの要件を満たさない言語は実用プログラムの作成に適切とは言えません。

部品型言語はスケートボードのようなものかもしれません。ストリートやパークで遊びながら技を試していれば、楽しみながらの模倣と工夫を通して様々な技能が習得できることでしょう。その上で、より現実的・日常的な仕事などでの利用が必要になったり、道の彼方まで行ってみたくなったなら、文字型言語という自動車の仕組みを学び、ハンドルを握ってアクセルを踏む時が来たのでしょう。



資料 15　関数化で速度が大幅に低下するプログラムと結果の例