上忍教材 No1:
大きな複雑なプログラムを作る方法(解説)

この資料は、学習を始める前の準備としてやった2つの課題の解説をしています。解説で、プログラムを作る時の、いろいろな設計方法と、大きな複雑なプログラムを作る時のポイントについて考えていきます。
この資料のプログラムは「高校自動販売機」のスタジオに入っています(Scratchの中で検索してみてください)。

2つの課題を、またやってみていない人は、こちらに課題があります。
http://beyondbb.jp/CDmama/materials/JouninNo.1_Module_Kadai201910.pdf

Go Ota, 2019 CoderDojo市川真間

課題1の解説: 簡単な自動販売機

簡単な自動販売機の動作

○旗を押したら(始め)
・お金の表示は0
・飲み物ボタンは 100円

○100円をクリックしたら
・お金の表示は100
・飲み物ボタンは

○　　をクリックしたら

・飲み物のスプライトを取り出し口に表示する。
・お金の表示を0にする。

「簡単自販機ベース」の名前で、スプライトが用意されています。これをリミックスして作ってみましょう。

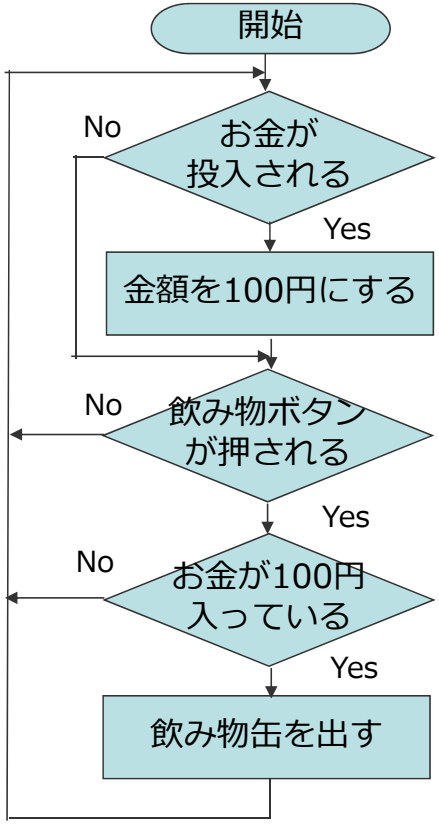
https://scratch.mit.edu/projects/335186683/

※ Scratch内で、「簡単自販機ベース」で検索できます。

一番簡単な自動販売機とフローチャート

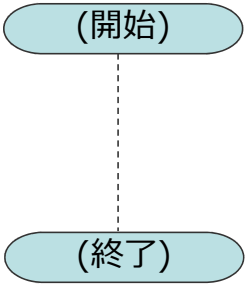
簡単な仕組みの自動販売機
・100円玉を1枚だけ入れられる
・商品は1つだけ
・飲み物ボタンだけ

まず、始めの一番簡単な自動販売機について考えてみましょう。お金を入れることと製品のボタンを押すことしかできません。
このプログラムの動作を、プログラムの設計図であるフローチャートで表すと右の図のようになります。



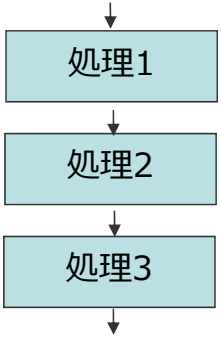
フローチャートの書き方

プログラムの開始と終わり

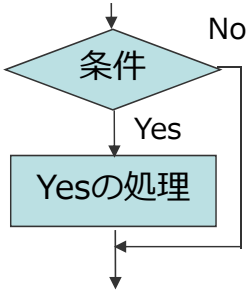


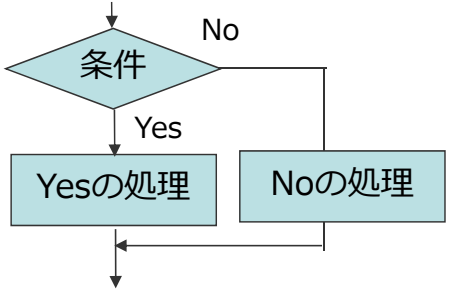
処理の流れの表記(アルゴリズム)

逐次型(直線型)



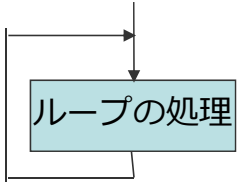
分岐型



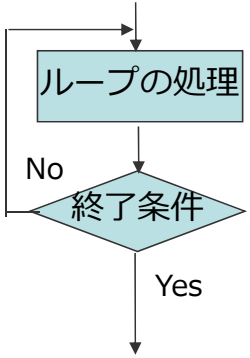


プログラムや人間の判断などのアルゴリズムは基本的に、逐次型、分岐型、ループ型の組み合わせで表現できますね。

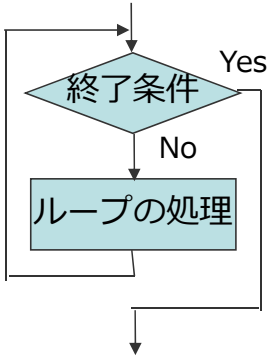
ループ型(繰返し型)



無限ループ



後判断ループ



前判断ループ

簡単自販機テキスト

簡単自販機の動作とプログラムを確認するため、まず文字(テキスト)で動作するもので確認してみましょう。

https://scratch.mit.edu/projects/142638853/

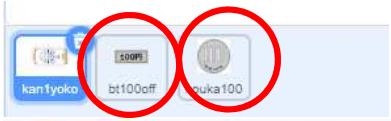
※ Scratch内で、「簡単自販機テキスト」で検索できます。

課題1の解答サンプル (缶のコード)

プログラムには、いろいろな方法があります。ここでは一つのサンプルを示します。
横になったカンにメインのプログラムがあります。

https://scratch.mit.edu/projects/335186400/
※ Scratch内で、「簡単自販機サンプル01」で検索できます。

課題1の解答サンプル(お金と商品ボタンのコード)



お金

商品ボタン

課題2: 本格的な自動販売機の解説

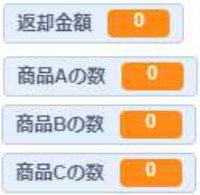
次のスライドで仕様を示した本格的な自動販売機はかなり難しいので。

まず、飲み物が1種類だけの本格的な自販機をつくりながら、いろいろな設計方法やプログラムを見ていきます。

設計方法としては、
次の3つの紹介します。

- フローチャート
  (前のスライドで紹介済み)
- 複合設計
- UML
  (シーケンス図とオブジェクト図

課題2: 本格的な自動販売機(1)



本格的な自動販売機を作ります。次のようなものが入ったプログラムが用意されています。これをリミックスして作ってみましょう。

・全部のスプライト
・基本的な変数
・初めに商品A～Cの数を入力する部分。
・100円と10円が効果投入口まで動くプログラム

難しいプログラムですが、出来るところまでつくってみましょう。

https://scratch.mit.edu/projects/332530818/

※ Scratch内で、「自販機ベース」で検索できます。

課題2: 本格的な自動販売機(2)

自動販売機の動作

〇旗を押したら(始め)

・質問に答えて、商品A,B,Cの数を入力(作成済です)

〇お金の動作

・100円玉、10円玉をクリックしたら投入口まで移動(作成済です)

・お金が投入口までいったら、対応する金額が[お金]として増えて表示

〇購入ボタンA/B/Cの動作

・対応する在庫[商品A/B/Cの数]があって、[お金]があったらonにする。(お金が入るたびにチェックしてonにします)

・上記以外はoff

・購入ボタンがonの時、対応する商品を取り出し口に表示する。

〇売り切れボタンの動作

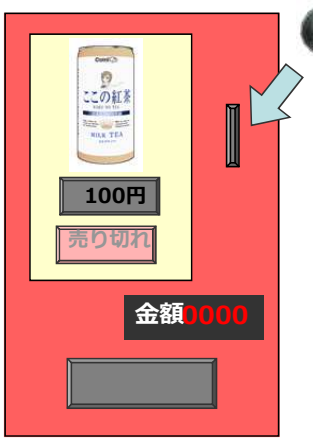
・対応する在庫[商品A/B/Cの数]あるときoff, ない時on

〇返却ボタン

・押すと、[金額]を0にする。

例えば、商品Aの数が1で金額が300の時、購入ボタンAがonの時、それを押すと、商品Aが表示され、その後、売り切れボタンAがoffになり、お金があっても購入ボタンはoffにする。

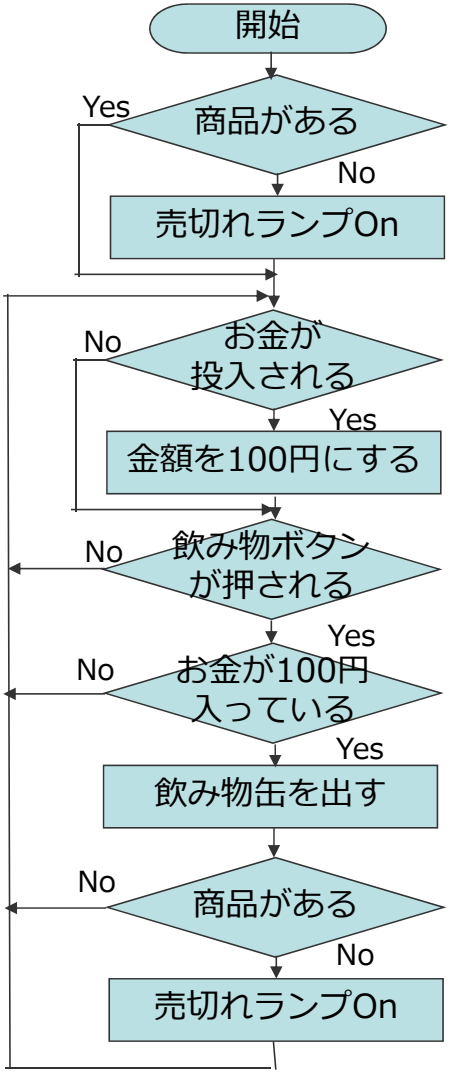
1種類自動販売機(その1)



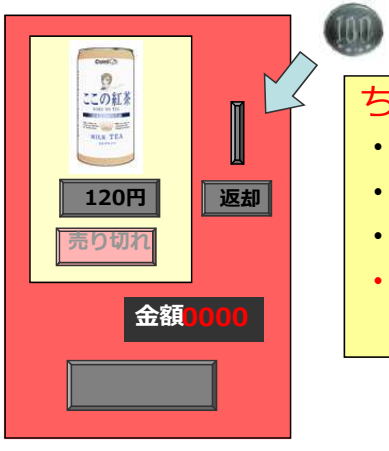
簡単な仕組みの自動販売機

・100円玉を1枚だけ入れられる
・商品は1つだけ
・商品切れランプは有り
・お金返却ボタンは無し

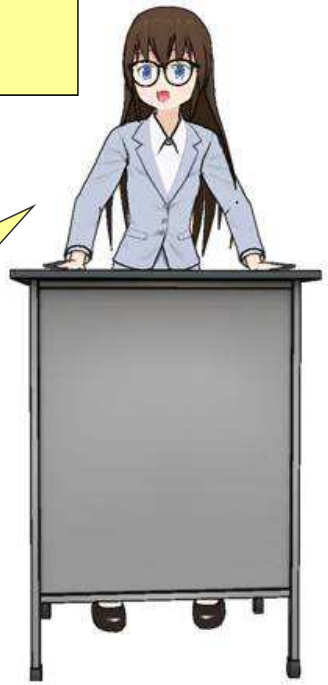
もう少し複雑な自動販売機を考えてみましょう。売切れランプを追加します。
フローチャートも少し複雑になりますね。ポイントは、開始直後と「飲み物缶を出す」の後の二か所に、商品の確認をして売切れランプの処理が入ることにあります。初めに品物が無い場合もありますから



ちょっとだけ複雑な1種類自動販売機



ちょっとだけ複雑な1種類自動販売機
- 100円玉を1枚だけ入れられる
- 商品は1つだけ
- 商品切れランプは有り
- お金返却ボタンは有り

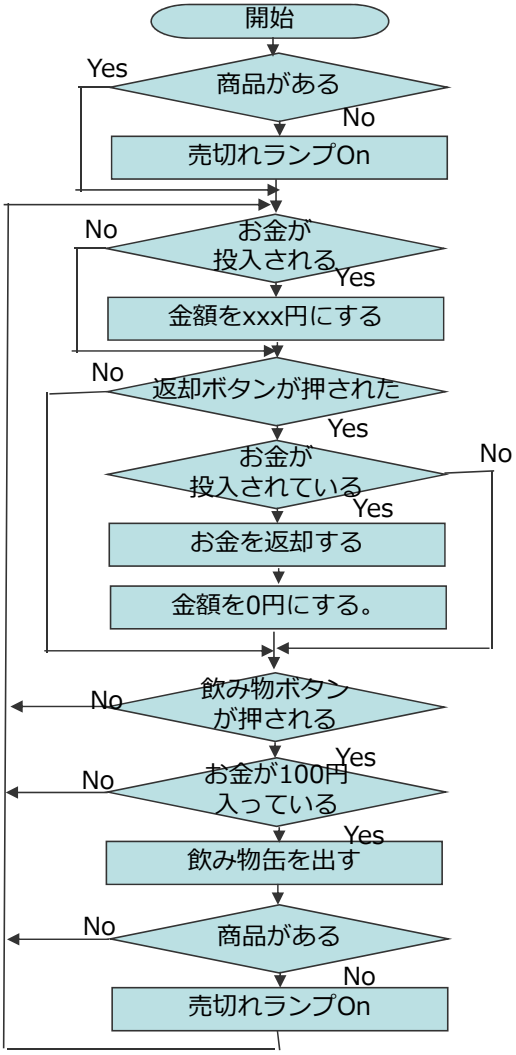


もう少し複雑な自動販売機を考えてみましょう。お金返却ボタンを追加します。
だいぶ、課題2にちかくものになります。

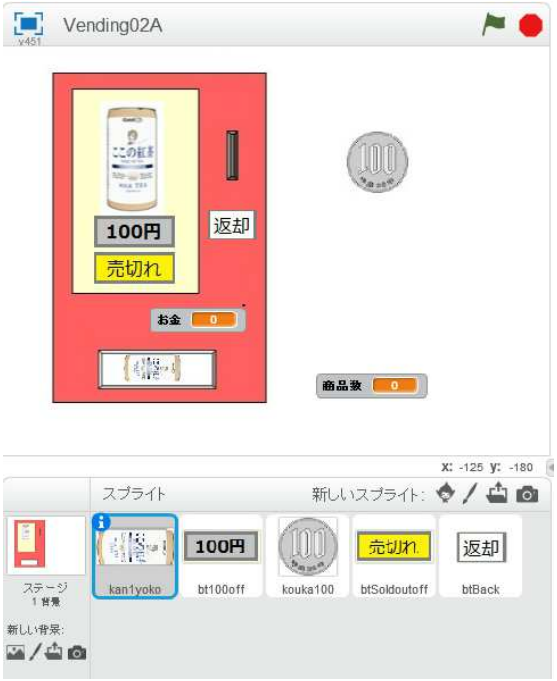
自動販売機といっても、いろいろな方法でプログラミングできます。そのいくつかを、その設計図といっしょに見ていきましょう。

単純なフローチャートを もとにしたプログラム

返却ボタンの追加では、お金が現在投入されたとか、金額表示を0円にするなどの処理も必要ですね。
　このぐらいになるとフローチャートもごちゃごちゃして判りにくくなりますね。



フローチャートをもとにした1種類自販機サンプル01

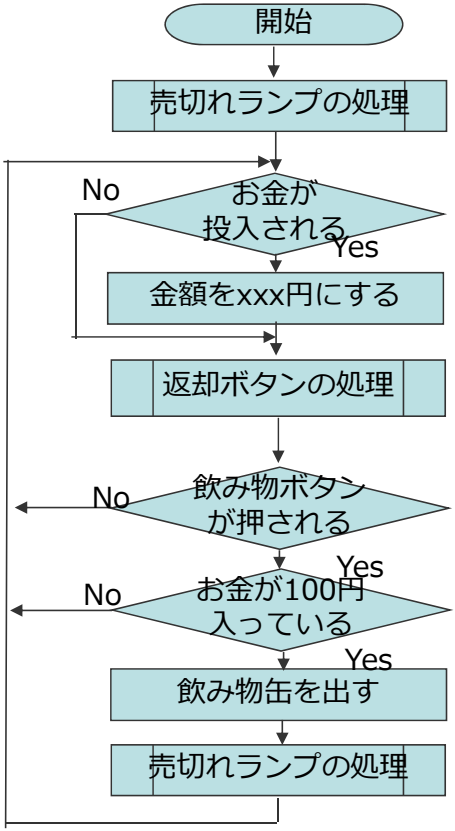


フローチャートをもとにしたプログラムです。
これもかなりぐちゃぐちゃしてきましたね。

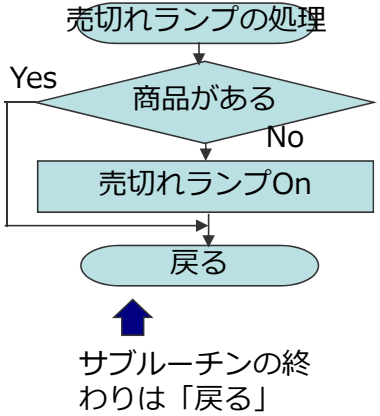
https://scratch.mit.edu/projects/142682702/
※ Scratch内で、「 1種類自販機サンプル01 」で検索できます。

サブルーチンですっきりさせたフローチャート

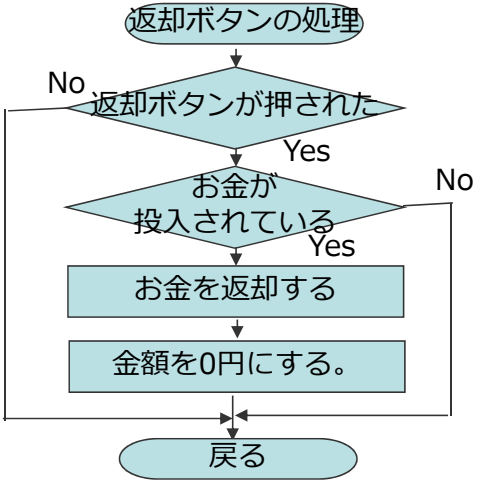
メインルーチン



サブルーチン1



サブルーチン2

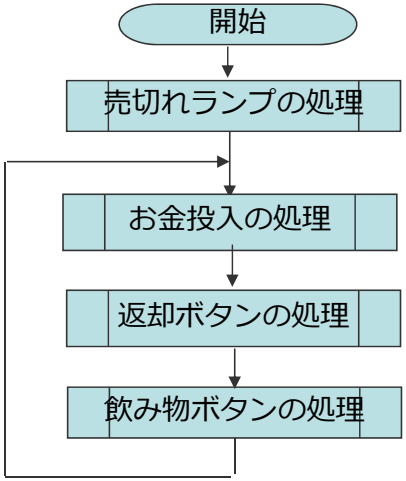


フローチャートが複雑になってきたら、サブルーチンを使いましょう。これは、　　を使って、フローチャートを分けることができます。サブルーチンに対して、大元のものをメインルーチンと呼びます。

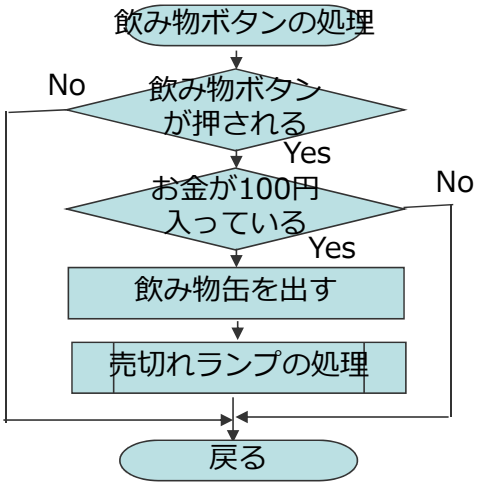
人間は沢山の情報をいっぺんに扱うことができません。そこでチャンクという個別の情報の塊にして扱います。

メインルーチンでもっとすっきりしたフローチャート

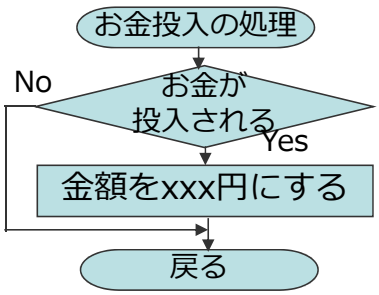
メインルーチン



サブルーチン4



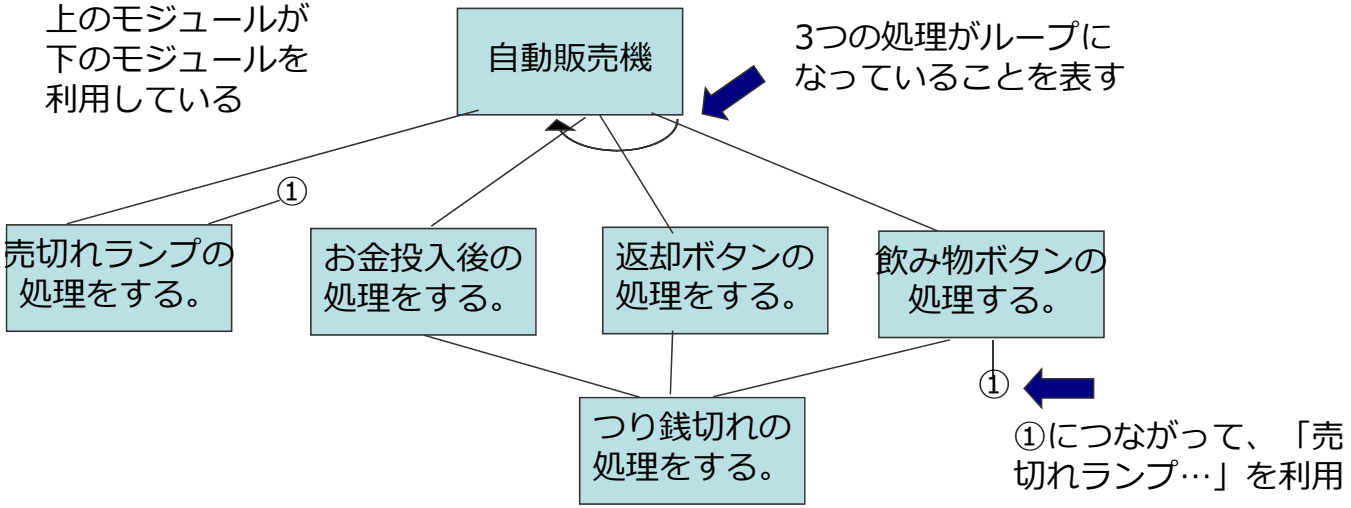
サブルーチン3



サブルーチンをもっと作るとメインルーチンがもっとすっきりします。こうすると、この時点の自動販売機のプログラムは1つのメインルーチンと4つのサブルーチンから構成されます。

売り切れランプの処理と、返却ボタンの処理は、前のスライドのサブルーチン1と2と同じになります。

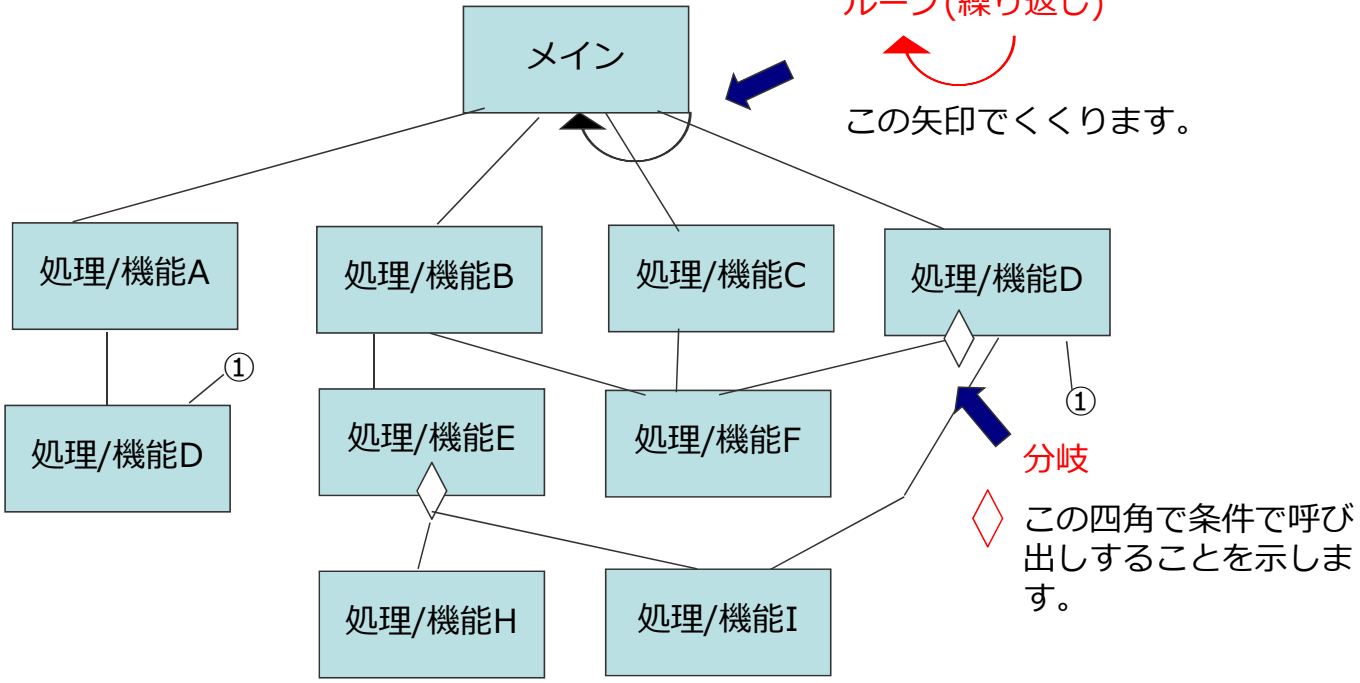
別の設計方法: 構造化設計



フローチャートはコンピュータのプログラムを設計する一つの手法ですが、あまり大きなプログラムには向いていません。現在では多くの設計手法がありますが、構造化設計は古いものですが、手軽に利用できます、また現在の設計手法の考え方のベースになっています。
フローチャートは処理の流れを中心に考えますが、構造化設計はモジュール(サブルーチン)の構造や関係を中心に表現します。この構造を考えると上図のようにシンプルなものになります。

参考図書:ソフトウェアの複合/構造化設計,G.J. マイヤーズ(1978)

複合/構造化設計の書き方



基本的なやりかた

プログラムの内容を、どんどん処理や機能で分解していく。分解していって、その処理や機能(モジュール)のプログラムの内容がたいだい想像できたら、分解は終了。

Scratchで新しいプログラムを作る時も、どんな構造(どんなモジュールから構成されるか)設計してつくるといいかもしれません。

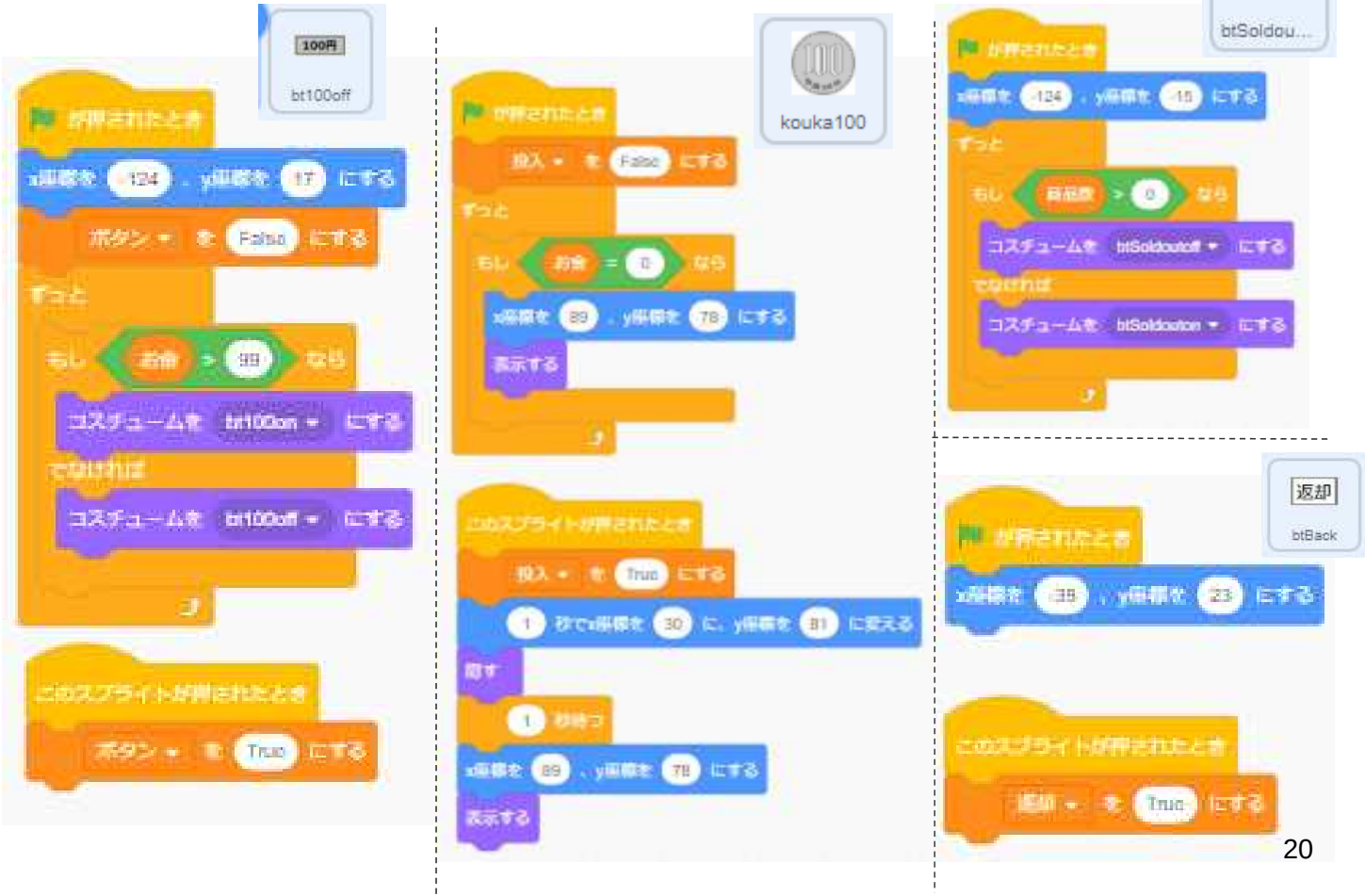
モジュール化した1種類自販機サンプル02(1)

モジュール(サブルーチン)使ったプログラムです。随分すっきりしたように見えます。

https://scratch.mit.edu/projects/142809844/

※ Scratch内で、「 1種類自販機サンプル02 」で検索できます。

モジュール化した1種類自販機サンプル02(2)



現在お勧めのUMLで設計してみよう

UMLは(統一モデリング言語: Unified Modeling Language）は、現在のシステム/ソフトウェア開発で主流になっている、オブジェクト指向分析や設計のための、設計のしかたです。全部で、13種類の設計の図式が決められていて、これらのいくつかを使って、分析・設計していきます。

ソフトウェア/システムの設計ですること

○振舞い

プログラムがどんな機能を持って、どう動くかの設計です。特に人間の使うプログラムの場合は、どんな操作をしたら、どう動くかを設計します。フローチャートでもどのような動きをするかを、ある程度表すことができます。

○構造

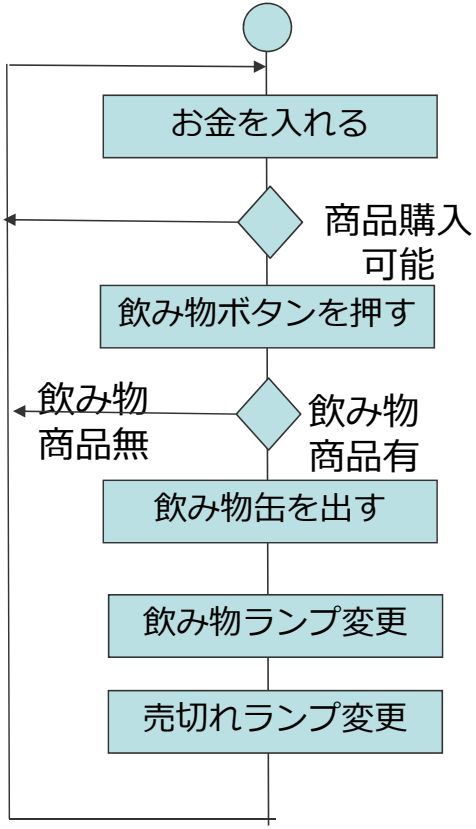
プログラムの中身がどのような処理や機能が組み合わさっているかを設計します。先に示した複合/構造化設計もこの構造やモジュールを簡易的にひょうげんするものです。UMLでは、モジュールをオブジェクトという単位として設計していきます。

○相互作用

振舞いの一種ですが、プログラムの中のモジュールやオブジェクトの動作に注目して設計していきます。

振舞いをアクティビティ図で設計してみる

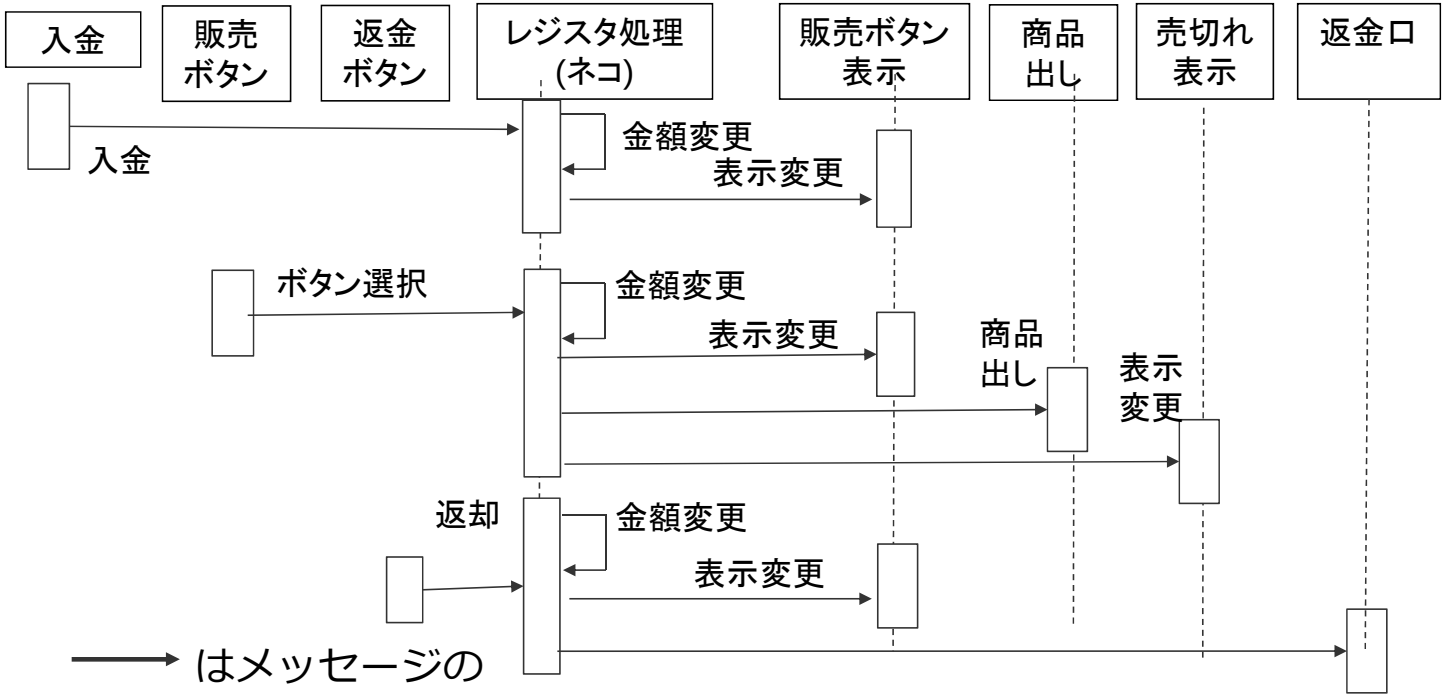
自動販売機を使う時の状態をフローチャートをアクティビティ図として書いてみました。



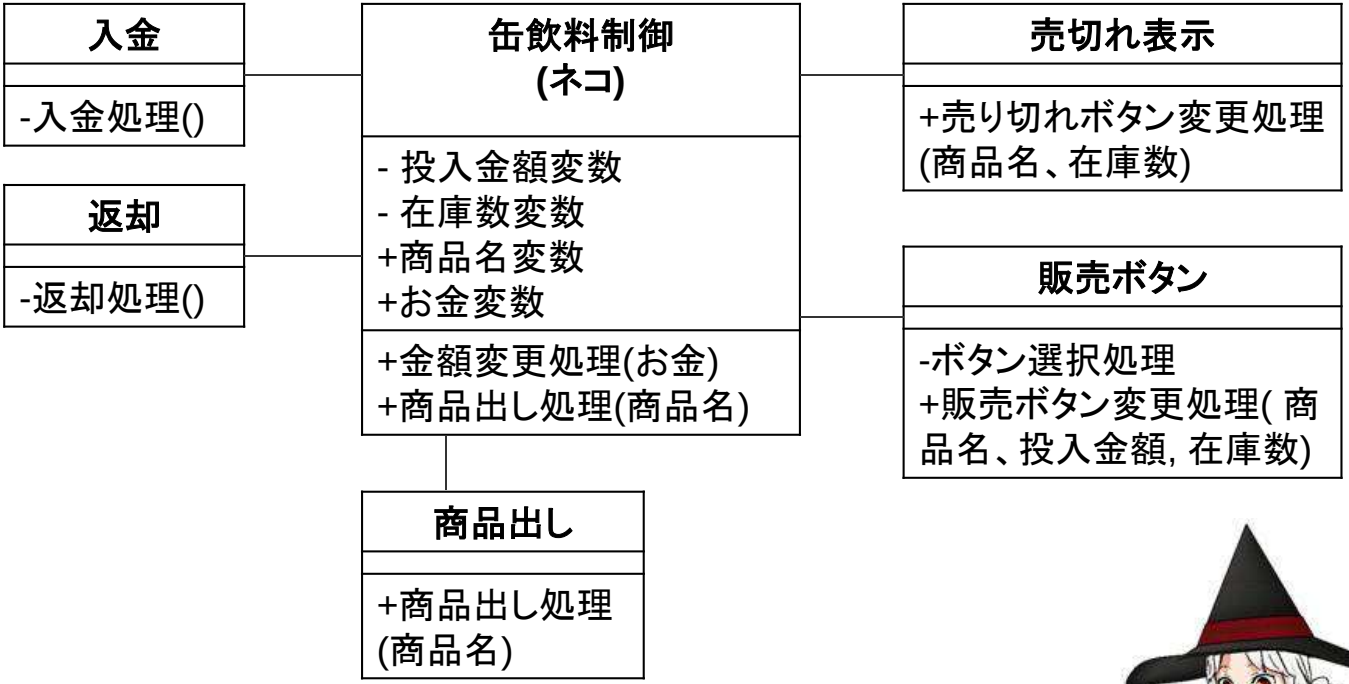
左のフローチャートは、プログラムのすべてを正しく表してはいませんが、人からみたプログラムの使い方を大まかに表しています。

相互作用をシーケンス図で設計してみる

プログラムの内部の動作を設計してみましょう。ここでは、UMLのシーケンス図を使って、それぞれの機能や処理がどのように連携して動くか設計しています。



構造をオブジェクト(クラス)図で設計してみる



現在多くのプログラムがオブジェクト指向をもとに開発されています。オブジェクトは個々のプログラムがある役割を持って、それが協働して動くことにより大きな仕事をするものです。ここでは、Scratchのスクリプトごとに機能をオブジェクト図類似したもので表現してみました。

オブジェクト(クラス)図の書き方

オブジェクト図でプログラムの構造を書き方です。(UMLでは、クラス図と、オブジェクト図に二種類あるますが、ここでは、Scratchで設計しやすいように、オブジェクト図にクラス図の要素をいれたもので説明します。)

オブジェクト名(クラス名)/ Scratchでのスプライト
属性(変数) - 変数 (そのスプライトだけが値を変更できる。他のスプライトは値を見れる) +変数 (いろいろなスプライトが値を変更できる。)
操作(モジュール/処理) -モジュール/処理 (そのスプライトの内部で拡張ブロックやメッセージで使用する) +モジュール/処理 (他のスプライトからメッセージなどで呼び出しできる)

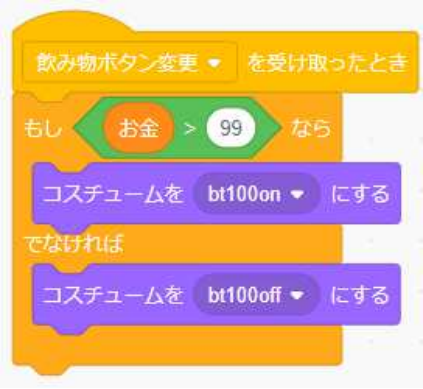
缶飲料制御 (ネコ)
- 投入金額変数 - 在庫数変数 +商品名変数 +お金変数
+金額変更処理(お金) +商品出し処理(商品名)

-変数 を使用することで、変数の値を変更している場所が明かになり、プログラムを正しく動かすことができるようになります。

シーケンスを主にした1種類自販機サンプル03(1)

シーケンス図とオブジェクト図をもとにしたプログラムです。やりとりの部分をできるだけScratchのメッセージを使って作っています。もともと、Scratchのスプライトは独立したオブジェクトなので、ランプの状態変化などはメッセージを使うとすっきりします。

1種類自販機サンプル02の場合
常にお金を監視して変更する必要がある。



1種類自販機サンプル03の場合
メッセージを使っているので、すっきりして確実。

https://scratch.mit.edu/projects/142911635/
※ Scratch内で、「 1種類自販機サンプル03 」で検索できます。

シーケンスを主にした1種類自販機サンプル03(2)

ネコのスプライトのコードが中心になっています。

シーケンスを主にした1種類自販機サンプル03(3)

大きな複雑なプログラムを作る方法(まとめ)

Scratchでの方法ですが。

○モジュール化

　Scratchは、もともとスクリプトでプログラが機能ごと分けられていますが、さらにスクリプトの中を適切な機能の小さなモジュールで作ります。簡単に設計する場合は複合/構造化設計つかうといいでしょう。しっかり設計するとはきUMLの各図を使います。

○オブジェクト化/変数のスコープ

　特に変数を利用できるスコープ(どのスプライトが変数の内容を変更できるか)を考えます。変数の変更を一つのスプライトに限定するとプログラムの動作がわかりやすくなります。

　さらに、スクリプト間の動作の引き渡しについては、スコープ図などで、どのようなメッセージを使うか明確にすると良いでしょう。

課題2の解答例: 自販機サンプル01

拡張ブロックを使って複合/構造化設計に近い形でプログラムを作成したものです。
繰り返し(ループ)処理で、各状態を確認しながら動作します。
下記のプログラムの中身を確認してください。

https://scratch.mit.edu/projects/332540159/
※ Scratch内で、「 自販機サンプル01 」で検索できます。

課題2の解答例: 自販機サンプル02

メッセージを使って割り込み処理を行い、シーケンス図に近い形でプログラムを作成したものです。各スクリプトがメッセージのやりとりで動作します。

https://scratch.mit.edu/projects/332753941/

※ Scratch内で、「 自販機サンプル02 」で検索できます。

自動販売機さん、
いつもご苦労様です

何気ない自動販売機でも、
いろいろなプログラムの作り方がありますね。