ver1.1

アースショップのコーディング規約

Scratch 編

アースショップのプログラミング教室

Copyright © 2023 アースショップ All Rights Reserved.

目次

プログラムの品質について 2
コーディング規約の分類 3
① 命名規則 4
1. 名前は全てわかりやすいものにする 4
2. 分かりやすい名前にするための具体的なルール 5
3. 変数の種類が分かるような名前をつける 6
② コーディング スタイル 7
1. スクリプトの内容を、コメントを使って説明する 7
2. 変数の内容を、コメントを使って説明する 8
3. スクロールが必要なほど長くなったスクリプトはブロック定義を利用する 9
4. 機能としてまとめられるコードは、ブロック定義を利用する 10
5. if 文の入れ子構造は、深くしてはいけない 11
6. 不要なスクリプトは削除する 12
③ 推奨事項 13
1. マジックナンバーを使わない 13
2. 変数を作るときは、できるだけ「このスプライトのみ」で作成する 14
3. スクリプトは、それに最も関係があるスプライトや背景のコードに記述する . 15
4. 不要な処理を実行しないように、スクリプトの先頭でガード節を使う 16
④ Scratch 特有の規則 17
1. 作成できるクローンの数は、プロジェクト全体で 300 個まで 17
2. リストに入れられる項目の数は、20 万個まで 17

プログラムの品質について

プログラムを作るとき、プログラムの組み方（コーディングといいます）は作る人によってバラバラです。たとえ同じ人が同じものを作ろうとしても、1 年前に作ったプログラムと今日作ったプログラムは同じものではないでしょう。

このように、作る人によって、場合によっては同じ人が作る場合でも、コーディングの仕方は千差万別です。なんのルールや取り決めもなくコーディングされたプログラムは、「品質が低いプログラム」の可能性が高いです。逆に、他人が見ても分かりやすいプログラム、1 年後の自分が見ても分かるようなプログラムのことを「品質が高いプログラム」といいます。

「品質が高いプログラム」は、才能やセンスとは関係がありません。良いコーディングとはどういうものか、というルールを決定しそれを守ることが大事になります。そして、「品質が高いプログラム」を書くためのルールや取り決めのことを、「コーディング規約」といいます。

コーディング規約は、プログラミング言語ごとに定められていたり、会社やシステム開発のプロジェクトごとに作成されていたりします。チームメンバーのスキルを考慮して、コーディング規約を定めることもあります。アースショップのレッスンでは、本資料を Scratch のコーディング規約として利用します。

本資料を理解し活用することで、「品質が高いプログラム」を作成しましょう。

コーディング規約の分類

アースショップのコーディング規約では、ルールを次のように分類しています。

・① 命名規則：
　背景やスプライト、変数などにつける名前に関するルール

・② コーディング スタイル：
　コメントの付け方やブロック化の基準などのルール

・③ 推奨事項：
　必須ではないが、ルールを守ることで品質が向上するもの

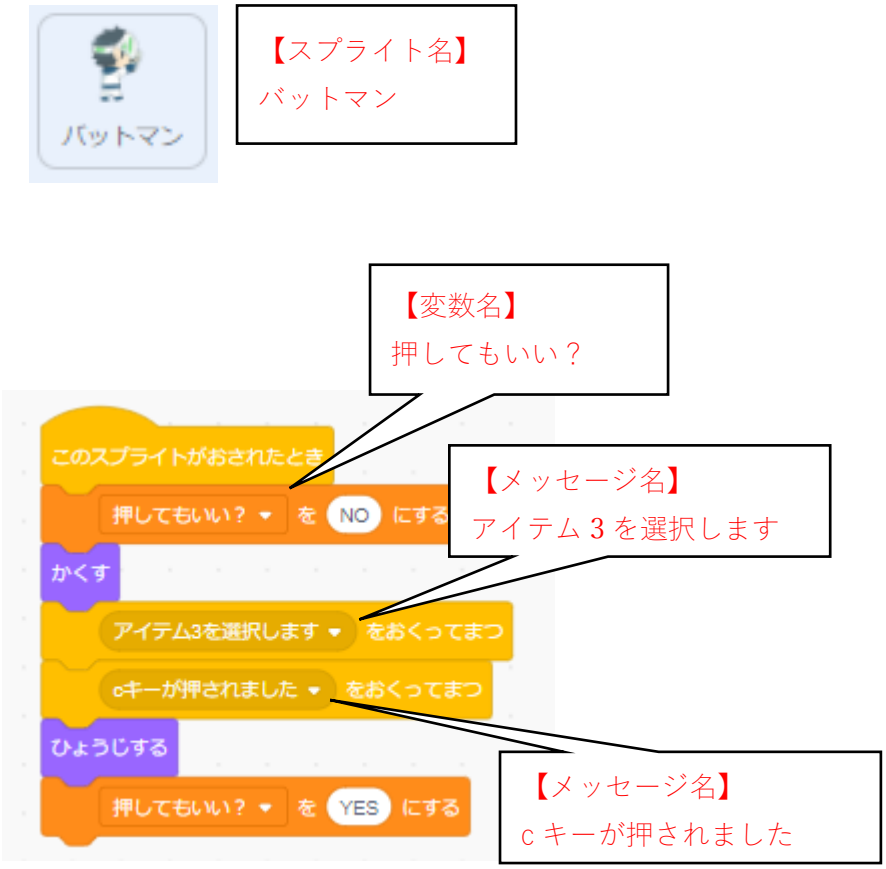
・④ Scratch 特有の制限：
　他のプログラミング言語にはない、Scratch 特有の制限

① 命名規則

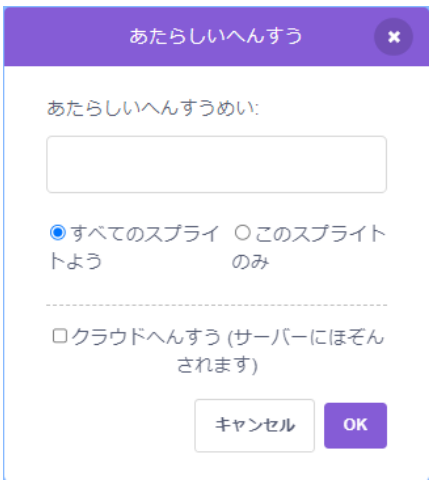
1. 名前は全てわかりやすいものにする

理由	分かりやすい名前をつけるほうが絶対いいに決まっている。「メッセージ 1 を送る」というプログラムから、きみは次に何が起きるか想像できるかな？
良い例	

2. 分かりやすい名前にするための具体的なルール

<table>
<tr><td>ルール</td><td>【背景、スプライト、コスチュームにつける名前】
「名詞」をつける
【メッセージ】
処理の前：語尾を「～する」とする
処理の後：語尾を「～した」とする
処理中　：語尾を「～している」とする
【変数（True か False の場合)】
疑問符「？」をつける</td></tr>
<tr><td>良い例</td><td> 
</td></tr>
</table>

3. 変数の種類が分かるような名前をつける

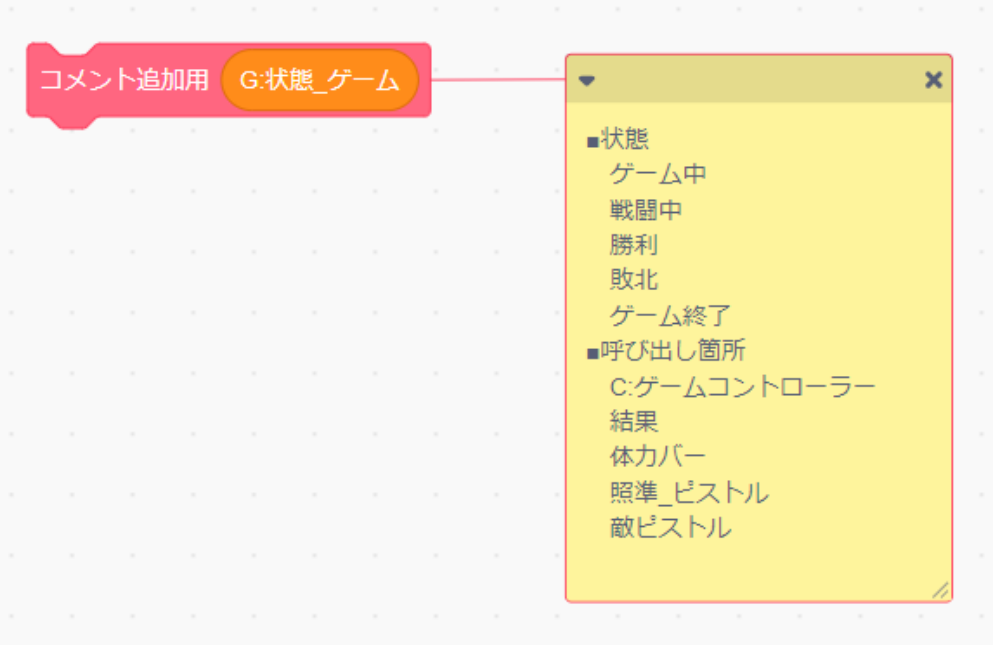
理由	スクラッチの変数は、次の 3 種類があります。 ①このスプライトのみ（ローカル変数） ②すべてのスプライト用（グローバル変数） ③クラウド変数（サーバー変数）。  どの種類で作られた変数か分かるように名前をつけます。
良い例	変数名の先頭に次の文字を付けることで区別します ①このスプライトのみ： なにもつけない ②すべてのスプライト用 グローバル変数の頭文字、「G_」をつける （例）G_現在のレベル ③クラウド変数 サーバー変数の頭文字、「S_」をつける （例）S_合計得点

② コーディング スタイル

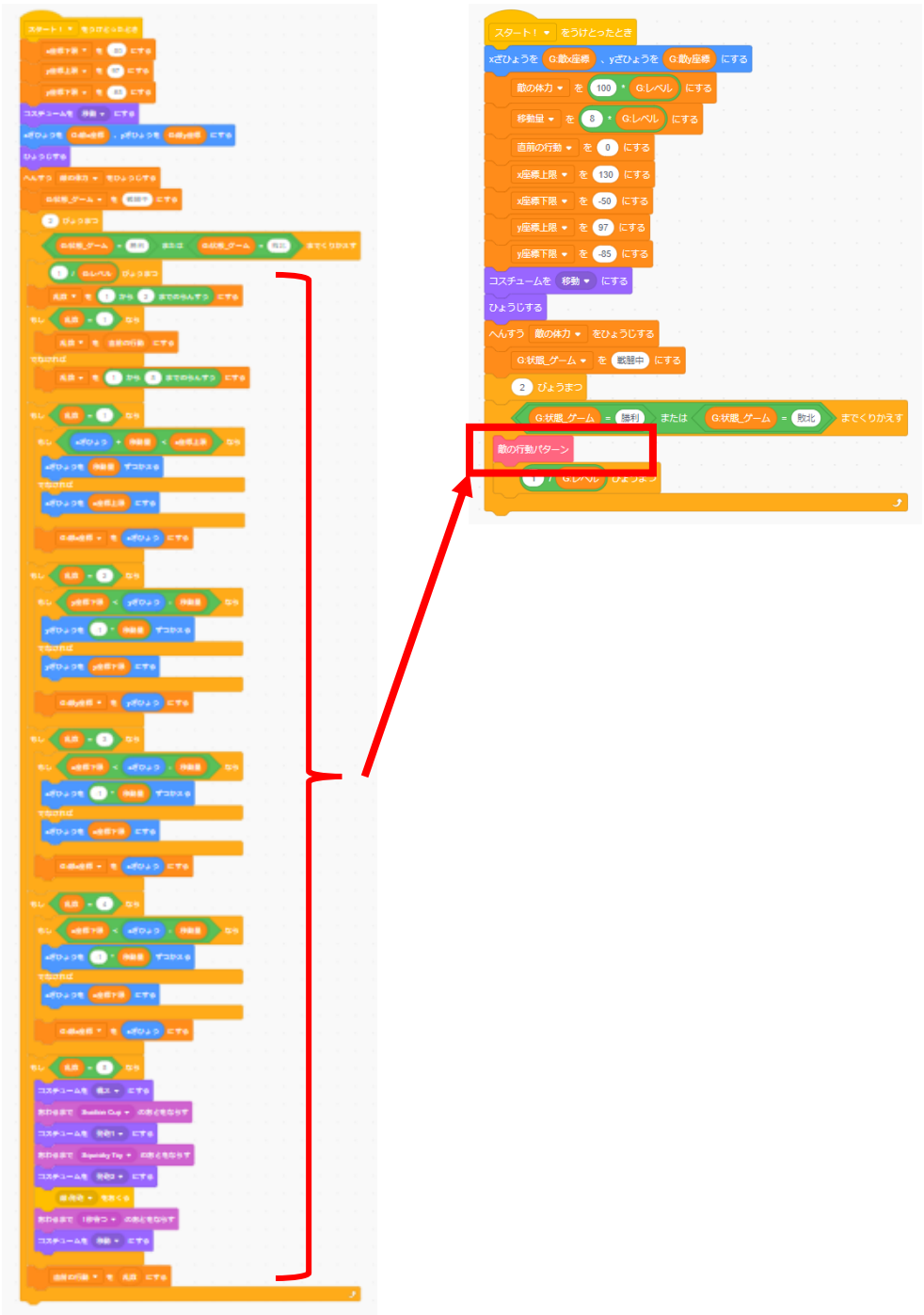
1. スクリプトの内容を、コメントを使って説明する

理由	下の例に出した「敵の行動パターン」というブロック定義の内容をみてください。コメントがなくても理解できますか？私はこのスクリプトの作者ですが、コメントがないと理解できません。
良い例	

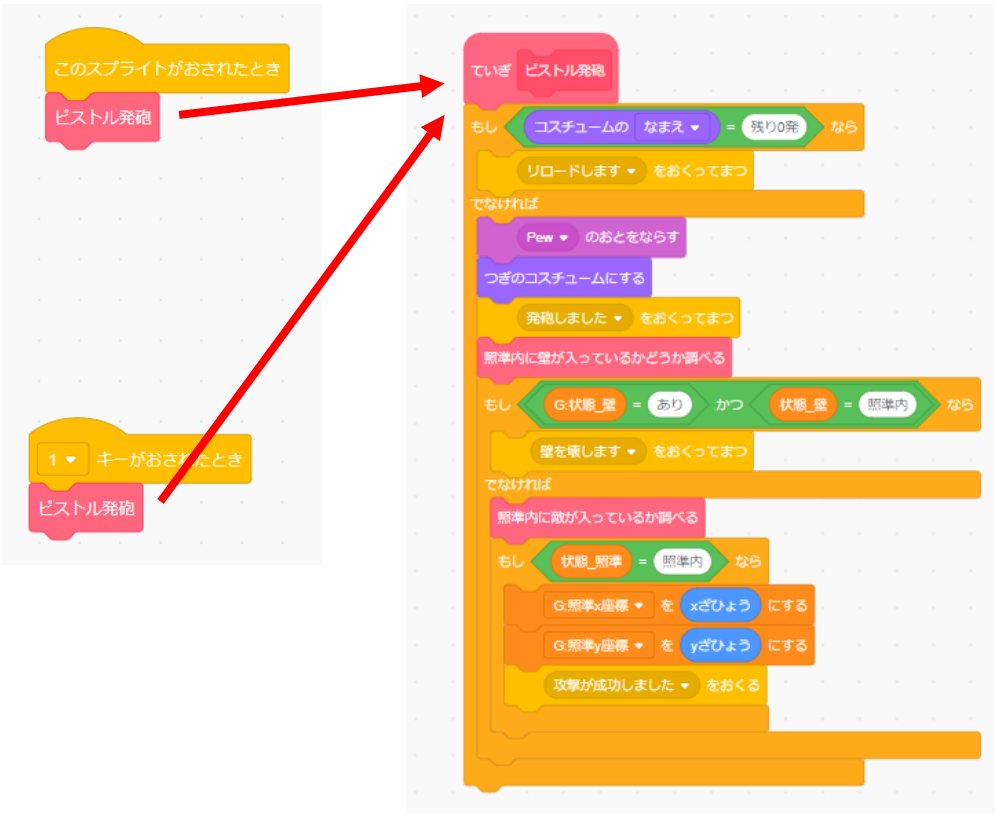
2. 変数の内容を、コメントを使って説明する

理由	ゲームが複雑になればなるほど変数の数が増えたり、色んなスプライトから呼び出されたりします。変数を使った目的や、どんな値が入るのか、どのスプライトから呼び出されるか、など後で困らないようにコメントを残そう。
良い例	

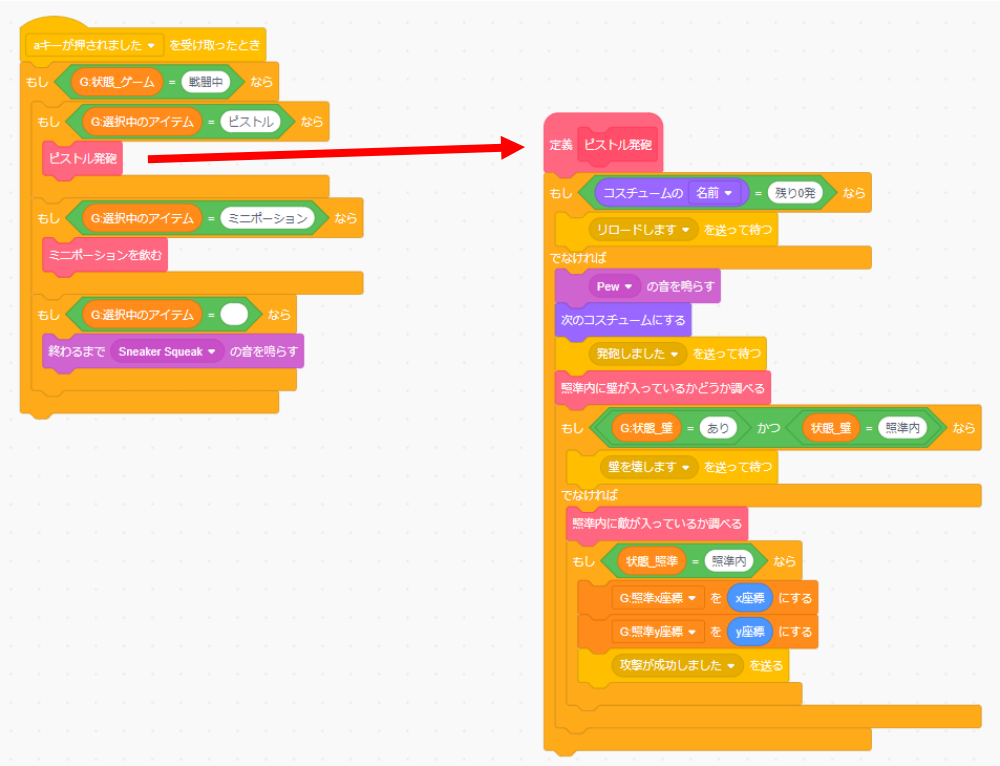
3. スクロールが必要なほど長くなったスクリプトはブロック定義を利用する

理由	どちらが見やすいかは一目瞭然！
良い例	

4. 機能としてまとめられるコードは、ブロック定義を利用する

理由	機能としてまとめておくとプログラムが見やすくなります。また、色んな所から呼び出すこともできて便利です。
良い例	

5. if 文の入れ子構造は、深くしてはいけない

理由	if 文の入れ子は 2 段以内におさめるようにしましょう。 if 文の入れ子構造を深くすると、プログラムが読みにくくなります。その結果、バグが発生しやすくなります。またその修正も困難です。ブロック定義を使ってスクリプトを分けましょう。
良い例	

6. 不要なスクリプトは削除する

理由	ここまで読み進めてきた君には説明不要だね！

③ 推奨事項

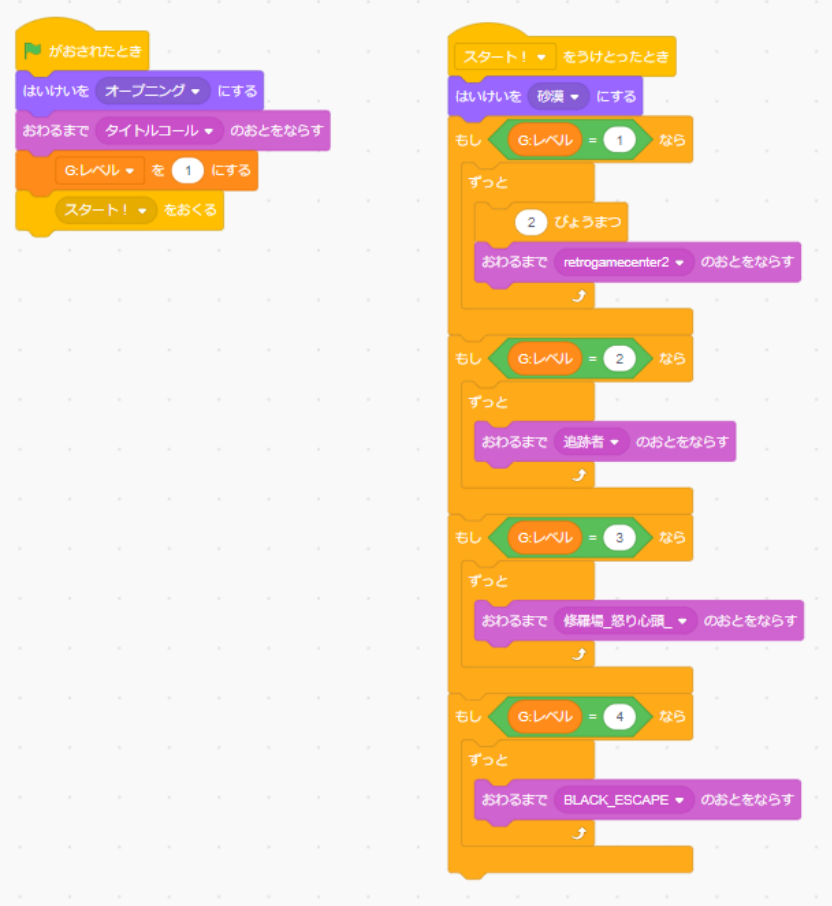
1. マジックナンバーを使わない

理由	下の例は、敵の体力や移動する速度、ステージ上のどこからどこまで動けるかといった範囲を指定しています。このように、数字や文字に意味があるものを「マジックナンバー」といいます。マジックナンバーがスクリプトにあると、プログラムを理解するのが難しくなります。変数で定義したり（もちろん分かりやすい変数名にしよう）、コメントを使って説明したりしましょう。
良い例	

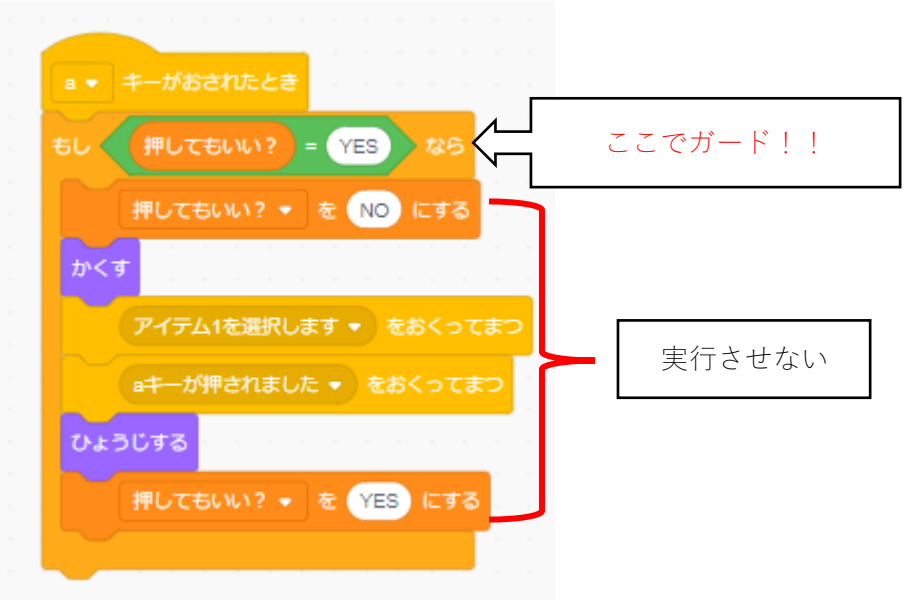
2. 変数を作るときは、できるだけ「このスプライトのみ」で作成する

理由	変数を「すべてのスプライト用」で作ると、他のスプライトからもその変数に自由にアクセスできるので作品が作りやすいと感じるかもしれません。ただ、プログラミングの世界では、この作り方はあまりよくない方法とされています。 少し難しい話になるのでここでは、これ以上説明はしません。詳しく知りたい場合は、「カプセル化」について調べてみましょう。

3. スクリプトは、それに最も関係があるスプライトや背景のコードに記述する

理由	例えば、ゲームの BGM を流すコードはどこに記述したらよいでしょうか。私は「背景」のコードに記述するようにしています。作品全体に関係するようなものは「背景」にまとめたい、という考えがあるからです。
良い例	

4. 不要な処理を実行しないように、スクリプトの先頭でガード節を使う

理由	下図は、キーが押されたときにある処理を実行させるスクリプトです。連続でキーを押しても処理が連続で実行されないように、ガード節を利用しています。 プログラムを実行してもよいのか、スクリプトの先頭でチェックをしましょう。
良い例	

④ Scratch 特有の規則

1. 作成できるクローンの数は、プロジェクト全体で 300 個まで

   300 を超えるクローンは生成されません。301 個目のクローンを生成しようとしても、プログラムが止まることはありません。

2. リストに入れられる項目の数は、20 万個まで

   20 万 1 個目の項目を追加しようとしても、プログラムが止まることはありませんでした。