教師用指導書

2018年8月27日

BeCREO TECHNOLOGIES

もくじ

もくじ

もくじ

もくじ

もくじ

スコッティ・ゴー!の準備（1/2）

室内の照明を明るくしましょう

スコッティ・ゴー!で遊ぶには、生徒は自分で書いたコードの写真を撮る必要があります。照明が十分に明るくないと、コードを正しくスキャンできないことがあります。

別のセットのタイルを組み合わせてはいけません

セット内の各タイルには、携帯機器のカメラで正しく識別する、固有のID番号と図があります。1つのコードで複数のセットのタイルを混ぜると、正しく識別されないことがあります。各セットのタイルの裏に別々の色で印を付けて、他のセットのタイルと区別するよう、お薦めします。

家庭版タイルと教育版タイルを組み合わせてはいけません

最適モデルでは、生徒2人あたり、1個のセットボックスの使用がお薦めです

生徒と**スコッティ・ゴー！教育版**を使い始める前に、生徒にルールを伝えましょう。例えば、生徒に誓約をさせるとよいかもしれません。その際、セットとは別のタイルをお互いに貸し借りしないというルールを決めておきましょう。授業終了後、他のセットのタイルを混ぜないよう、タイルをそれぞれのボックスに戻させてください。小さい子供の場合、必要なタイルをすべて事前に用意しておくようお薦めします。

すべてのクエストにアクセスできる特別な教師用アカウントを作成しましょう

モジュール内のすべてのクエストにアクセスできる教師用アカウントを作成できます。スコッティ・ゴー!コースをまだ完了していなくても、クエストにアクセス可能です。
教師用アカウントでアクセスできるオプションにより、生徒の年齢や技能レベルに応じて、選択したプログラミング概念を取り扱う様々な科目を教えながら、スコッティ・ゴー!が使えます。

教師用アカウントを作るには、**スコッティ・ゴー！教育版アプリ**で次のオプションを選んでください。

プレイヤーを選択 -> 新しいプレイヤー 次に、このコードを入力:

Teacher_13_cps#

このコードは生徒に見せないでください!

スコッティ・ゴー!の準備（2/2）

教師用アカウントを使うと、生徒はチームで作業できます。このモデルでは、クエストの説明の表示用に機器1台、生徒が作ったプログラムのスキャン用に携帯機器1台のみ必要です。

教師用アカウントによって、スコッティ・ゴー!進度カードを最大限に活用できます。追加の教材をダウンロードするには、アカウントにログインしてください:

www.scottiego.com/education

教師用メモ（1/2）

正解は1つ以上あります

この指導書には、各クエストに解答があります。様々なプログラムの同じ問題を解くには、多くの方法があります。中には、解答案よりもはるかに短いものもあります。

ゲームは、モジュールやクエストで活用できる179タイルで構成されています

新しいプログラム概念や、後でクエストやモジュールに導入する新しいタイルを使って、各クエストを何度も繰り返し解くことができます。

スコッティ・ゴー！教育版アプリは、ゲーム方式を使っていますが、本格的な教育ツールです

スコッティ・ゴー！教育版アプリを使うと、キーコード能力に関する指導カリキュラムの目標を達成し、また他の状況で計算論的思考をする際にも最新の世界的開発を活用できます:

• 論理的思考や抽象的思考、計算論的思考、データの視覚提示に基づく理解、分析、問題解決、

• コンピューターやその他のデジタル機器を使ったプログラミング、問題解決: コンピューターアプリケーションを使ったアセンブリングやプログラミングのアルゴリズム、情報の整理、検索、共有、

• 社会的能力の開発、例えば: 仮想環境、グループプロジェクトやプロジェクト管理への参加を含む、グループコミュニケーションや協力、

ゲームは生徒が学ぶのを助ける単なるツールであり、目標ではないことを覚えておきましょう!

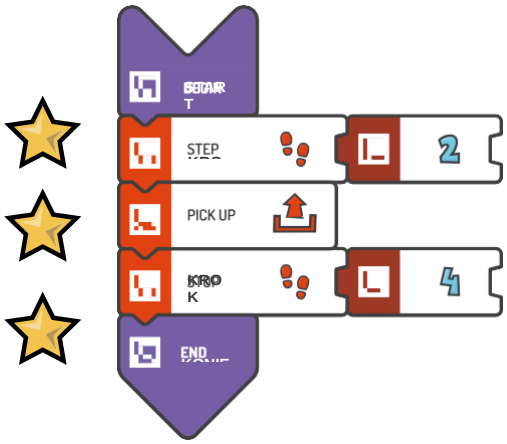
授業でスコッティ・ゴー！教育版を使用する際、能動方式、レッスン目標、レッスン管理、評価などに関するルールに従うようお薦めします。スコッティ・ゴー!教育版は、新しく導入した概念に、スパイラル法を取り入れています。つまり、**後のクエストでも異なる状況で使われたコマンドを1つのクエスト内で使っていきます。クエストの難易度は、ゲームが進行するにつれ、クエストごとに上がります。そのため、教師と生徒はアルゴリズム技能を養うことができます。毎回、新しいコマンドが導入され、画面のゲームキャラクターの右上隅の感嘆符（!）で合図します。**

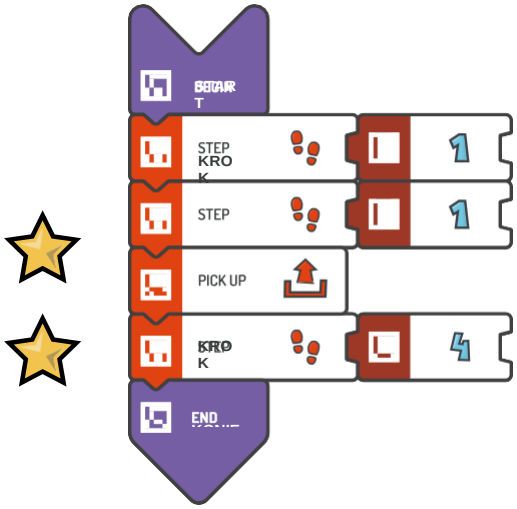
教師用メモ（2/2）

プログラミングが上手な人や意欲的な生徒は、問題を解決する最適な方法を常に探します。できるだけ少ないコードで構成されていながら、目的を達成している場合、そのプログラムは「最適」です。

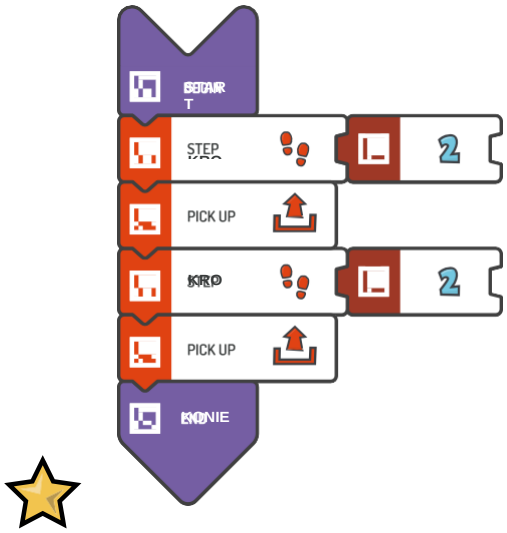
授業中、教師は学習を促進する雰囲気を作らなければいけません。**スコッティ・ゴー！教育版アプリ**には、興味深いゲームに基づく環境での形成的評価の要素という特徴があります。各クエスト完了後、生徒は、コードの質や最適化に対するごほうびとして１つから3つの星をもらいます。お互いに競争をさせなくても、とても楽しいはずです。

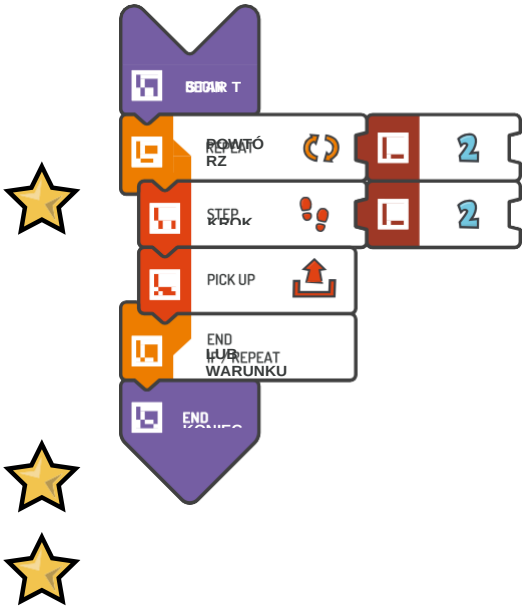
プログラムが最高に良い場合は星3つ、解決が少しだけ効率的ではない場合は星2つ。クエストは解決されたけれども、プログラムが長すぎる場合は星1つです。





クエストによっては、1つ以上の星が欲しい場合、**REPEAT**タイルのような特定のタイルを解答で使う必要があります。





モジュール1
ヨーロッパ1

がらくた集め競走が大好きです。あなたはどれぐらいうまくできるかな。「X」のスクエアを全部調べよう。

モジュール導入: コマンド

スコッティ・ゴー!では、簡単なコマンドと複雑なコマンドの、2種類を区別します:

簡単なコマンド

簡単なコマンドは、あらゆるプログラムの基本要素です。各コマンドで行う動作や操作は、1つです。簡単なコマンドの例には、スコッティがどのように動くか、振舞うかに関連した動作があります。例えば:

通常、スコッティ・ゴー!コマンドのように、1行のコードにはコマンドが1つあり、別の行の下に並んで置かれ、その順番に実行されます。

複雑なコマンド

複雑なコマンドは、作成するプログラムの一部、または全体の解釈を指示します。通常、一連の簡単なコマンドまたは別の複雑なコマンドで構成されています。一般的に、複雑なコマンドは角かっこの中に入れます。スコッティ・ゴー!の代わりに、2つの揃いのオレンジ色のブロックの間に配置されます。複雑なコマンドの例には、**REPEAT**ループ、**REPEAT WHILE**ループ、条件文コマンド**IF**などがあります。

注意!
ゲームキャラクターの管理、最初のコマンド

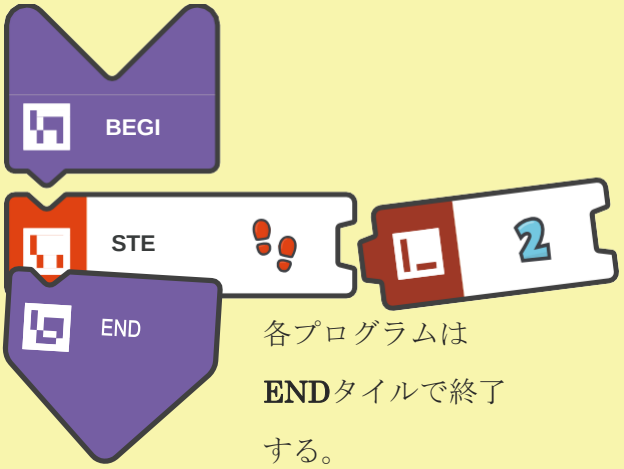
最初のクエストの最も重要な点は、**スコッティ・ゴー！教育版アプリ**にあるチュートリアルの使い方を生徒に教えることです。生徒は、**スコッティ・ゴー!ユーザーマニュアルの14ページ**に記載されたスキャンプログラムの方法を学習する必要があります。

通常、生徒は問題なくチュートリアルが使えますが、ヒントの意味に注目せずに進んでしまうことがあります。**モジュール1のクエスト1～3には、パラメータ**の概念という特徴があり、これはゲームを通して主要な概念であることに注目することが重要です。パラメータとは、コマンドの実行方法を定義する値のことです。ゲームで学んだ最初のコマンド - **STEP** – には、スコッティが何歩進むかを指示するパラメータを決める必要があります。

各プログラムは**BEGIN**タイルで開始する。

STEPのようなコマンドには**パラメータが必要**なものがある。

BEGINと**END**タイルの間に、**連続で実行される**コマンドを何個でも置くことができる。

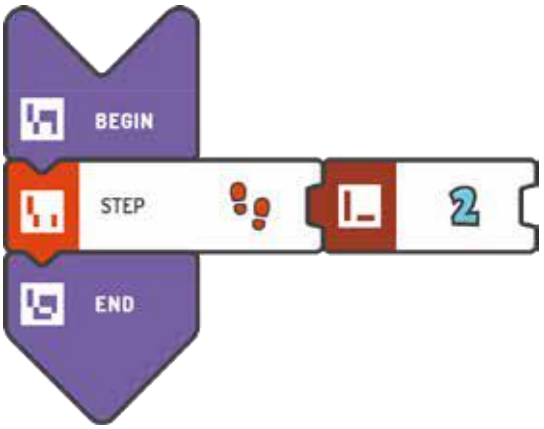


各プログラムは**END**タイルで終了する。

STEP(2)とは、スコッティがスクエアを2個、前へ進むことを意味する;
STEP(3)とは、スコッティがスクエアを3個前へ進むことを意味する、等。

挑戦してみよう:

解答例:

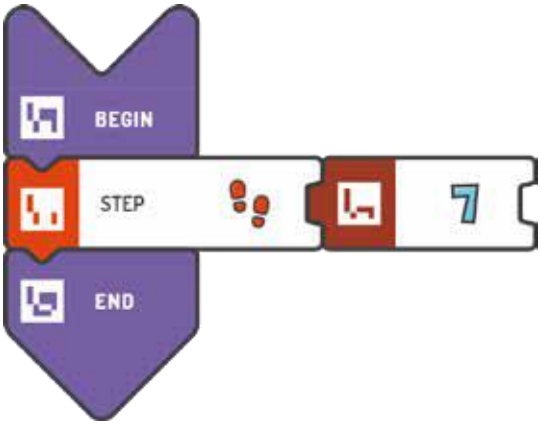


解答例（テキスト形式）：

```
BEGIN
    STEP(2)
END
```

挑戦してみよう:

解答例:



解答例（テキスト形式）：

```
BEGIN
    STEP (7)
END
```

導入:
数字、数を作る、足算と引算の学習

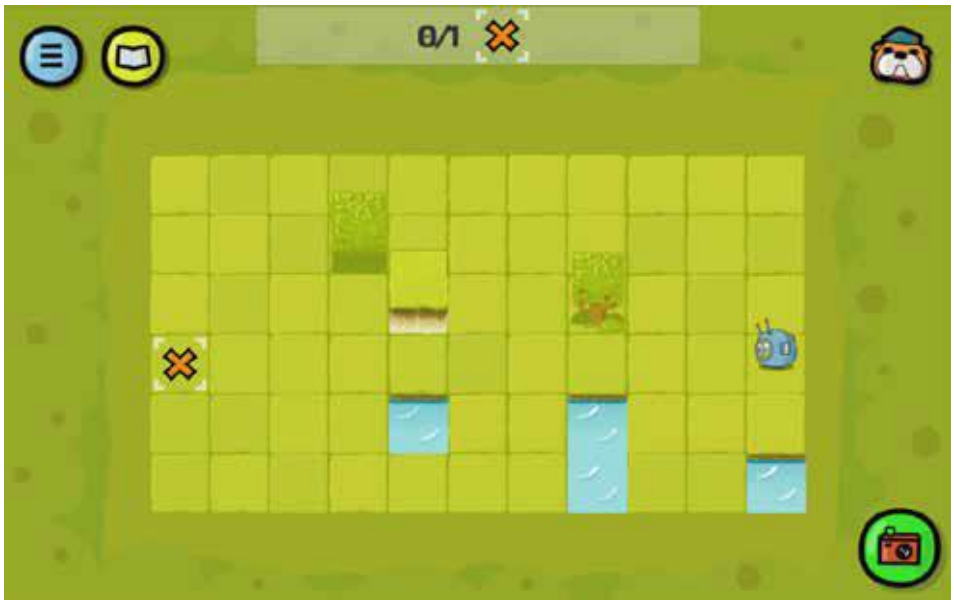
タイルを数字と組み合わせて数を作ります。

PLUSタイルを使う場合、足算をして数を作ることができます。

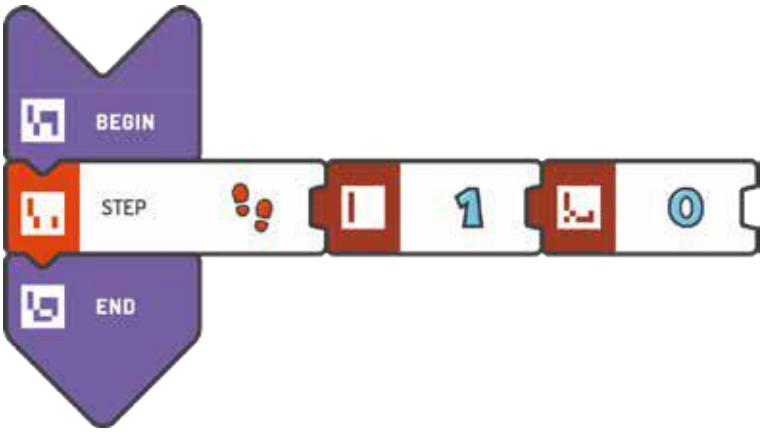
または、**MINUS**タイルを使う場合、引算をします。

数学操作には、もっと大きな数のタイル（数字／数）を含むこともあります。足算や引算を使うこともあります。

挑戦してみよう:



解答例:



解答例（テキスト形式）：

```
BEGIN
    STEP (10)
END
```

導入: 向きを変える

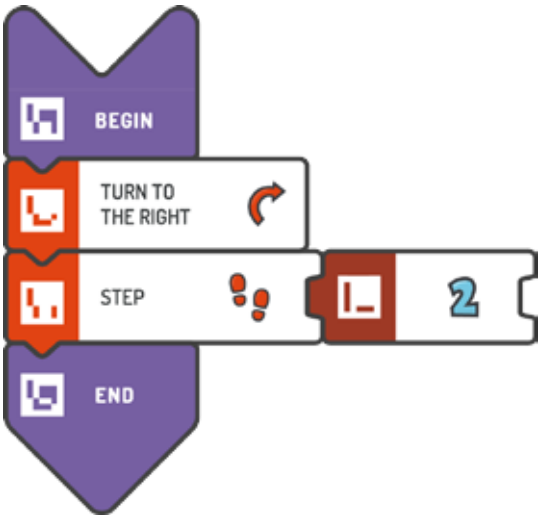
スコッティは方向転換できます。これには、**TURN TO THE LEFT**タイルを使います。

または**TURN TO THE RIGHT**を使います。

スコッティがどのように動くかを説明するために、このクエストでは生徒はスコッティのふりをします。こうすると、スコッティの動き方が簡単に理解できます。

挑戦してみよう:

解答例:

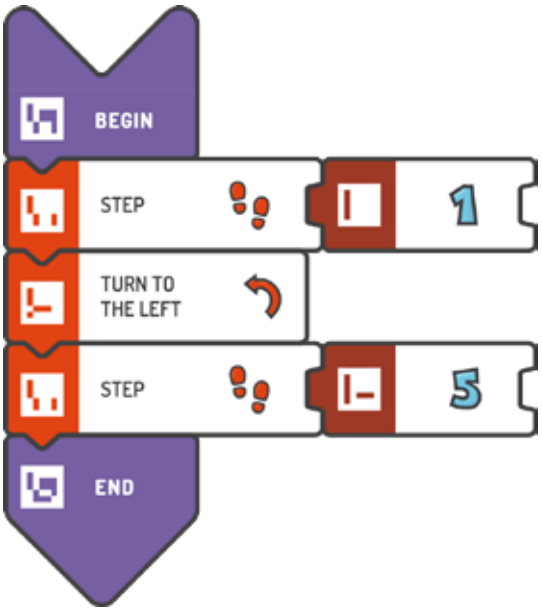


解答例（テキスト形式）：

```
BEGIN
    TURN TO THE
    RIGHT STEP (2)
END
```

挑戦してみよう（スコッティはスクエア「X」すべてを通らないといけないことに注意）：

解答例:



解答例（テキスト形式）：

```
BEGIN
    STEP (1)
    TURN TO THE
    LEFT STEP (5)
END
```

挑戦してみよう（スコッティはスクエア「X」すべてを通らないといけないことに注

解答例:

挑戦してみよう（スコッティはスクエア「X」すべてを通らないといけないこと

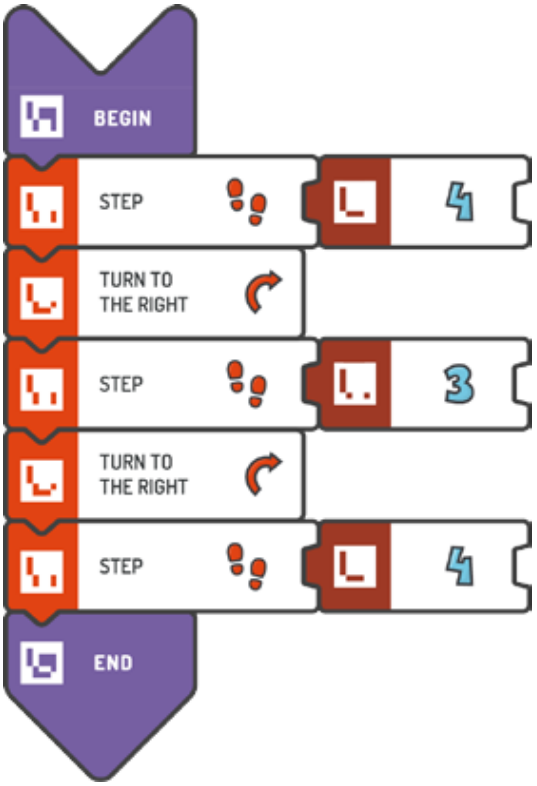
解答例（テキスト形式）：

```
BEGIN
    STEP (2)
    TURN TO THE
    LEFT STEP (2)
    TURN TO THE
    LEFT STEP (2)
END
```

挑戦してみよう:


0/5

解答例:


BEGIN
STEP
4
TURN TO THE RIGHT
STEP
3
TURN TO THE RIGHT
STEP
4
END

挑戦して

解答例（テキスト形式）：

```
BEGIN
    STEP (4)
    TURN TO THE
    RIGHT STEP (3)
    TURN TO THE
    RIGHT STEP (4)
END
```

導入: 後ろ向きに歩く

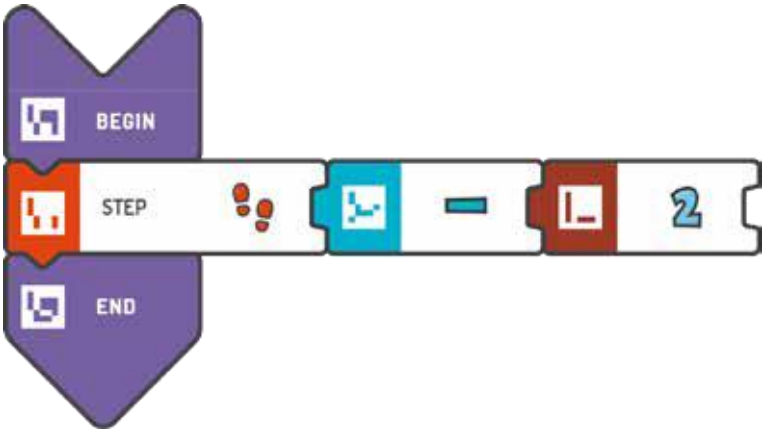
スコッティは後ろ向きに歩くこともできます。これには、**MINUS**タイルを使います。

後ろ向きに歩くことは、スコッティの新しい能力の1つです。このゲームで実行させるには、スコッティの歩数の値の前に**MINUS**タイルを使います。

算数の授業でまだ負数を学んでいない生徒でも、なじみのある正温度と負温度の類似点に気付くと、容易に理解でき、すぐに**MINUS**タイルを使い始めます。

挑戦して

解答例:

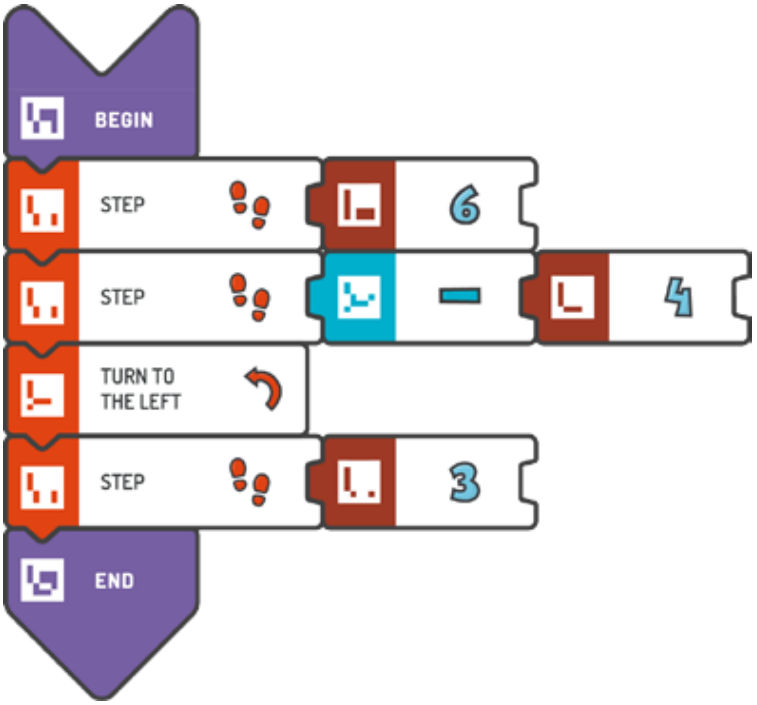


解答例（テキスト形式）：

```
BEGIN
    STEP (-2)
END
```

挑戦してみよう:

解答例:



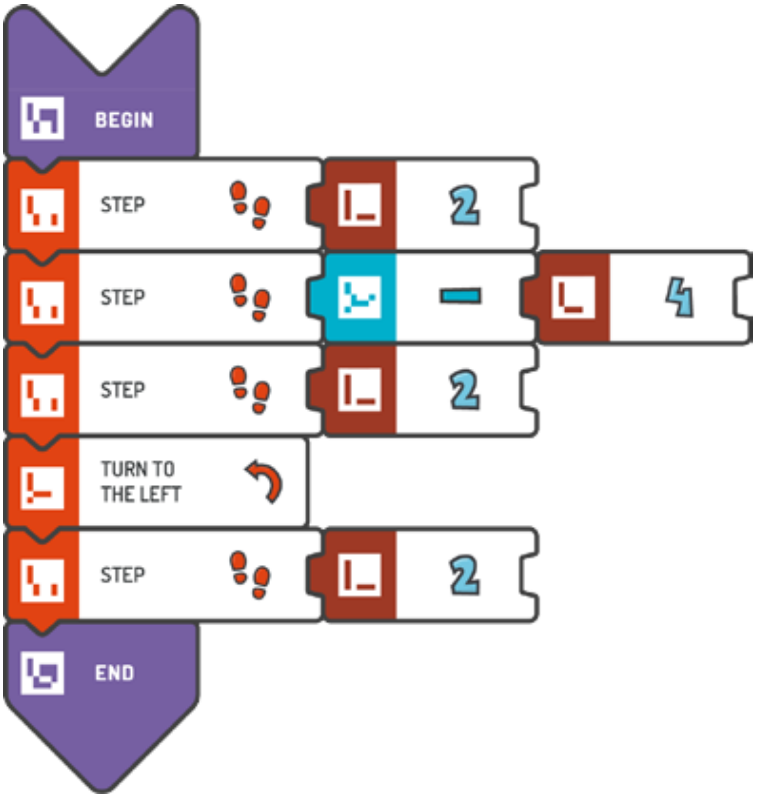
挑戦してみよう:

解答例（テキスト形式）：

```
BEGIN
    STEP (6)
    STEP (-4)
    TURN TO THE
    LEFT STEP (3)
END
```

挑戦してみよう:

解答例:



挑戦して

解答例（テキスト形式）：

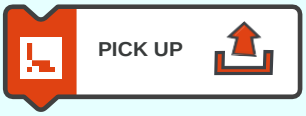
```
BEGIN
    STEP (2)
    STEP (-4)
    STEP (2)
    TURN TO THE
    LEFT STEP (2)
END
```

モジュール2
ヨーロッパ2

スコッティは、周囲と関わり始めます。
地面からカタツムリを全部拾って、ランニングトラックに持っていきましょう。

モジュール導入:
ゲームボードからものを拾う

ゲームボードにあるものを拾うには、**PICK UP**コマンドを使います。

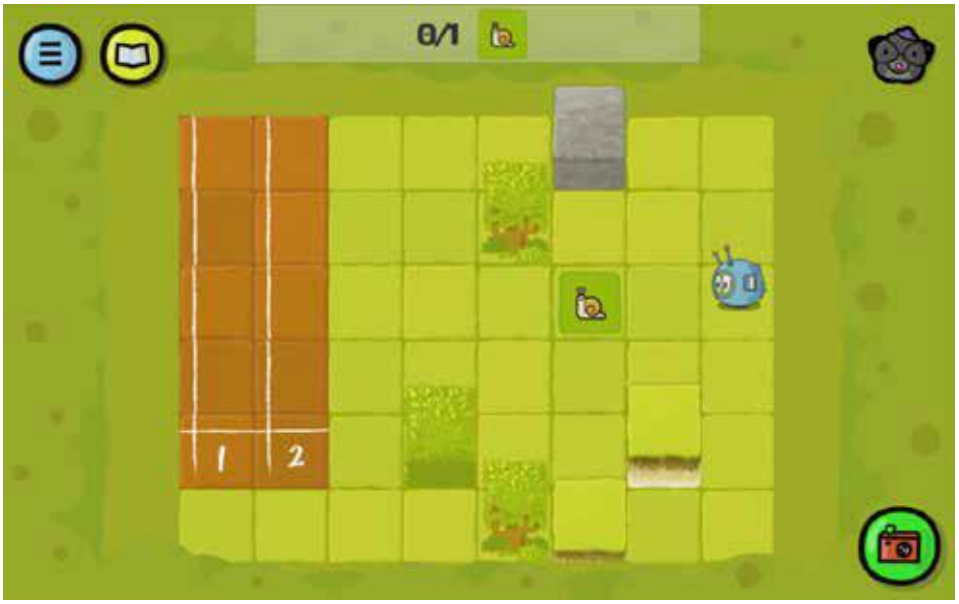


プログラムでは、例えばカタツムリを拾う場合、このコマンドを使います。

注意!

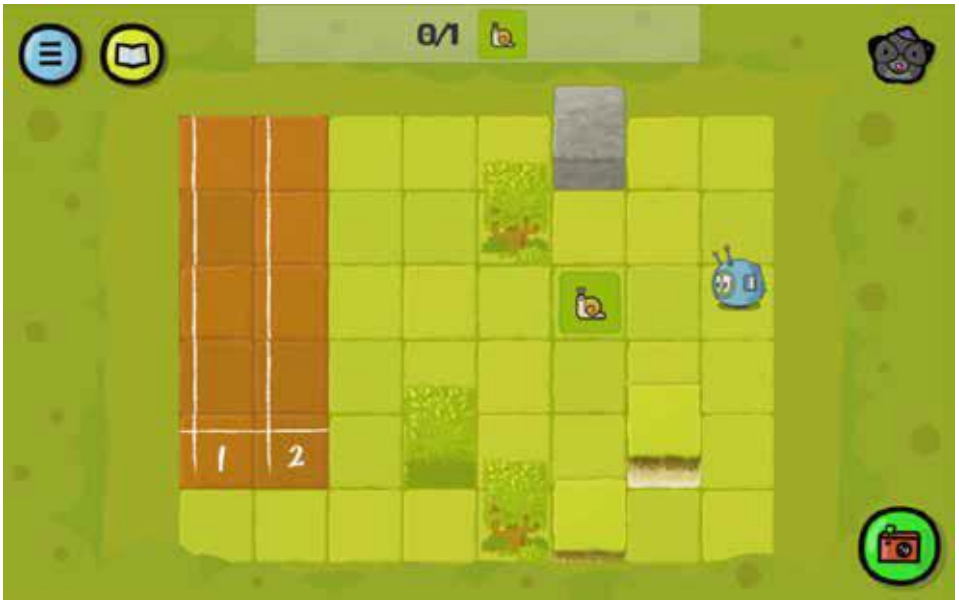
空白のスクエアで**PICK UP**コマンドを使うと、エラーになります。モジュール2で**PLACE**タイルを使う必要はありません。

挑戦してみよう（**スコッティはランニングトラックを完了していなければいけません**）：



解答例:

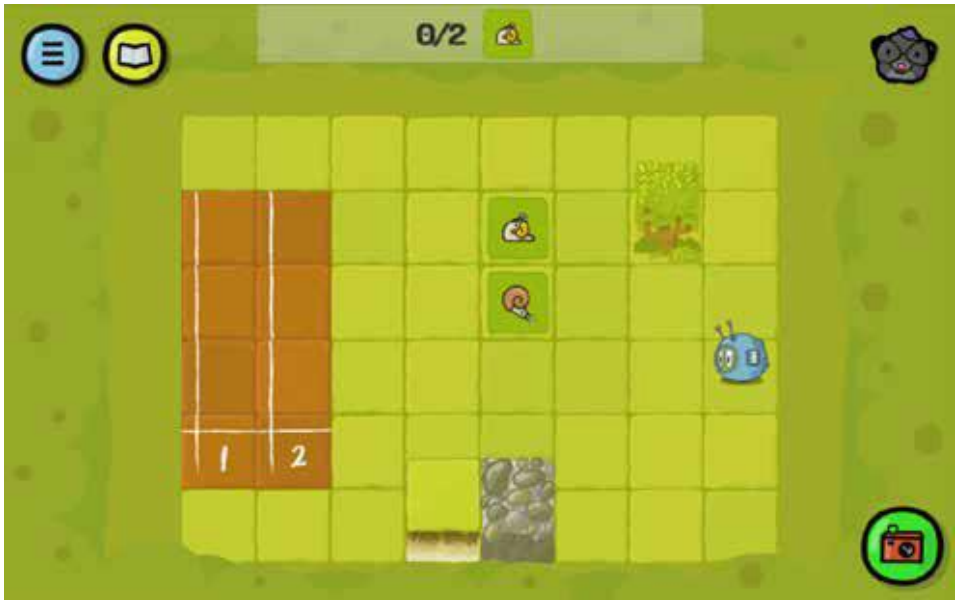
挑戦してみよう（スコッティはランニングトラックを完了していなければいけませ



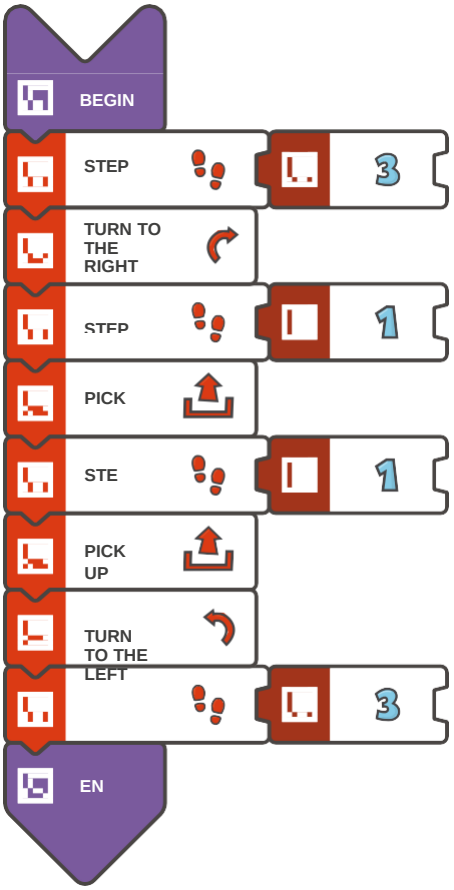
解答例（テキスト形式）：

```
BEGIN
    STEP
    (2)
    PICK
    UP
    STEP
    (4)
END
```

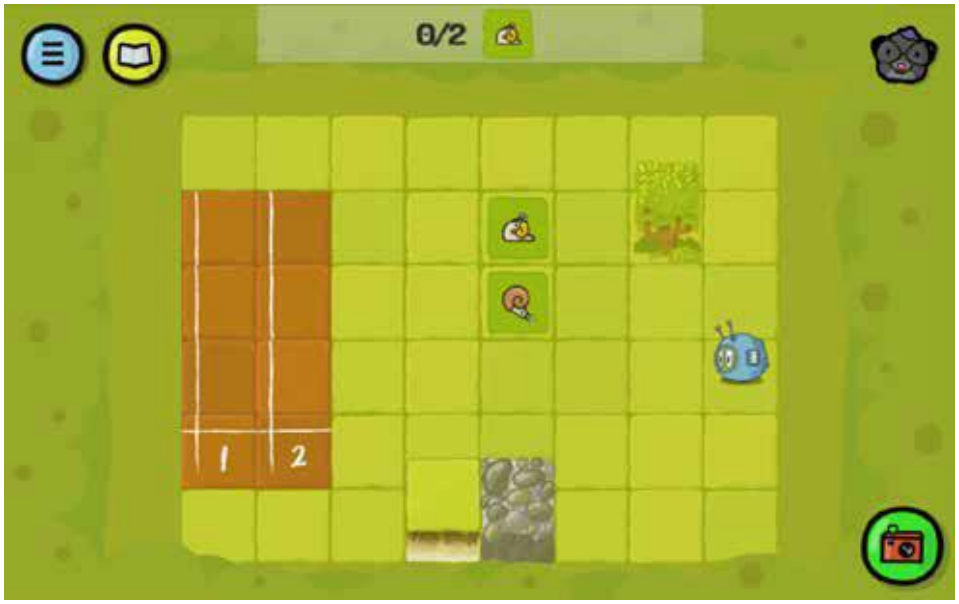
挑戦してみよう（スコッティはランニングトラックを完了していなければいけませ



解答例:



挑戦してみよう（スコッティはランニングトラックを完了していなければいけませ

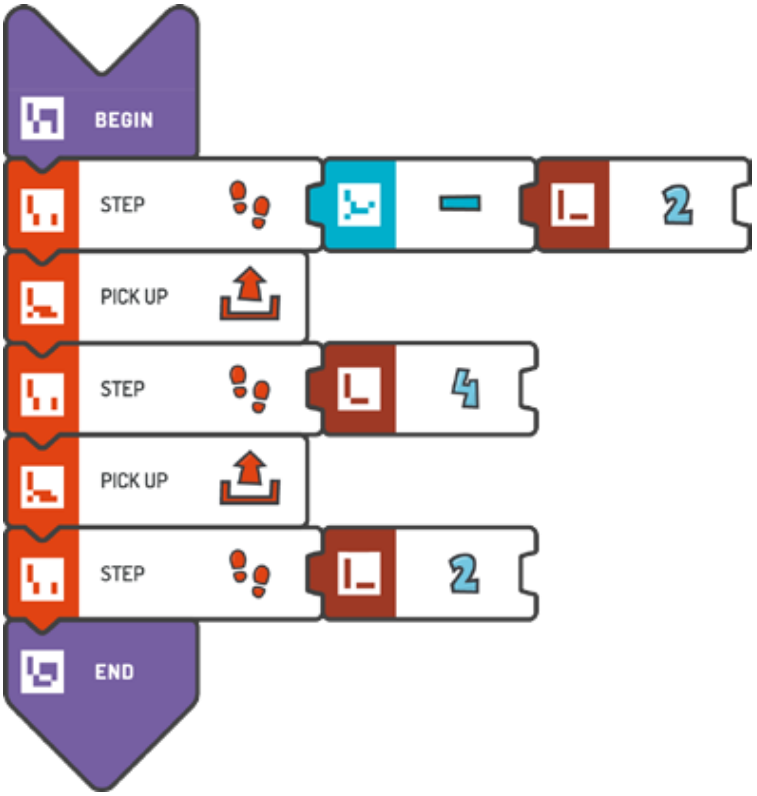


解答例（テキスト形式）：

```
BEGIN
    STEP (3)
    TURN TO THE
    RIGHT STEP (1)
    PICK
    UP
    STEP
    (1)
    PICK
    UP
    TURN TO THE
    LEFT STEP (3)
END
```

挑戦してみよう（スコッティはランニングトラックを完了していなければいけませ

解答例:



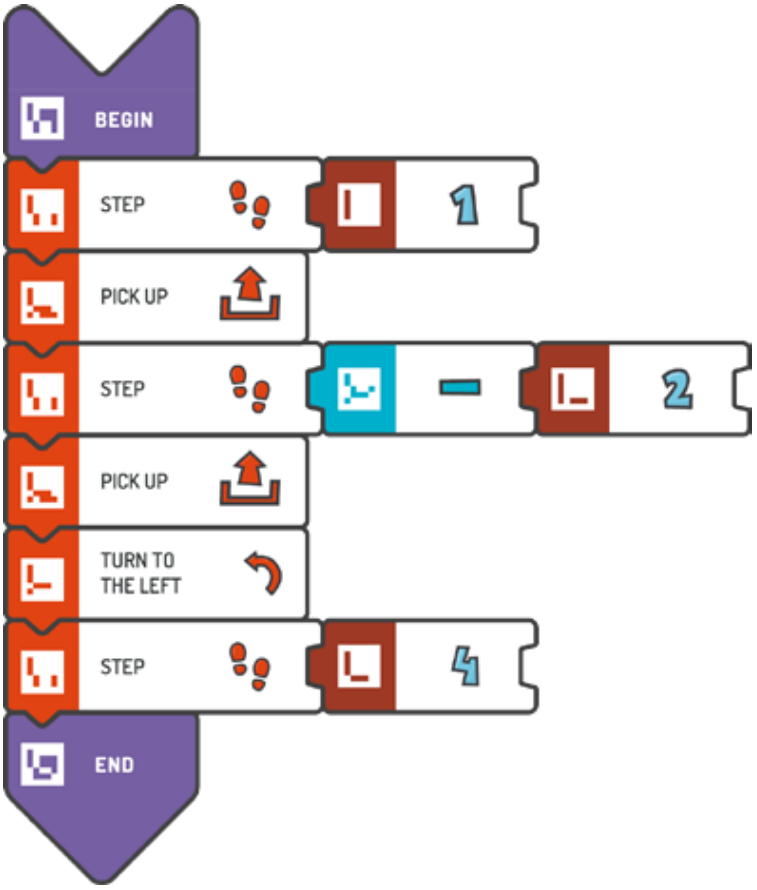
挑戦してみよう（スコッティはランニングトラックを完了していなければいけませ

解答例（テキスト形式）：

```
BEGIN
    STEP (-2)
    PICK UP
    STEP (4)
    PICK UP
    STEP (2)
END
```

挑戦してみよう（スコッティはランニングトラックを完了していなければいけませ

解答例:



挑戦してみよう（スコッティはランニングトラックを完了していなければいけませ

解答例（テキスト形式）：

```
BEGIN
    STEP (1)
    PICK UP
    STEP (-2)
    PICK UP
    TURN TO THE
    LEFT STEP (4)
END
```

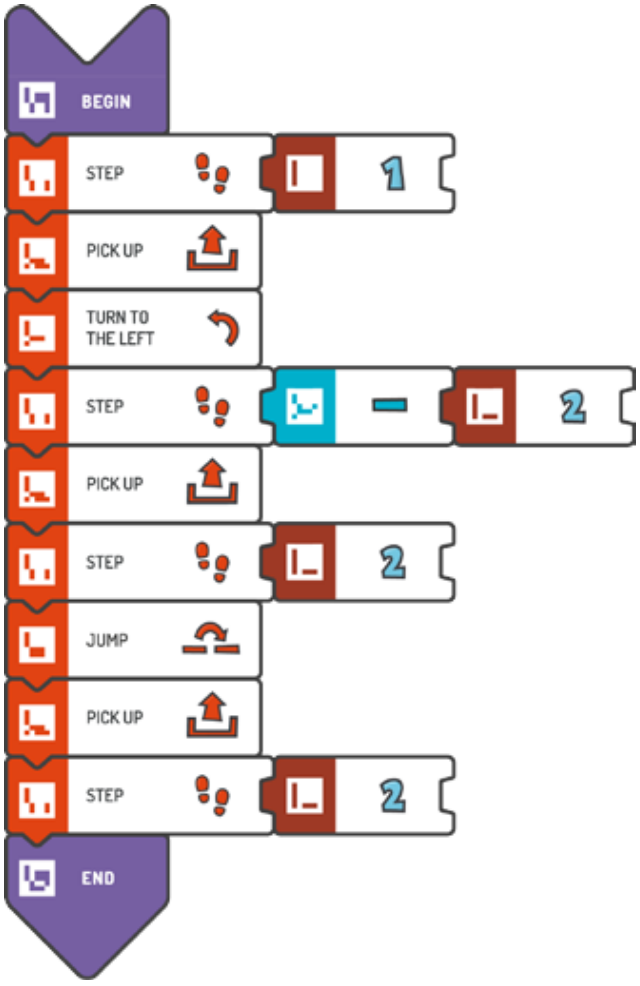
導入: 飛び越える

スコッティは途中で小さな障害物に出くわすことがあります。これを飛び越えるには、赤い**JUMP**タイルを使います。スコッティがジャンプできるのは、前方だけです。

ジャンプするとき、スコッティはスクエア1からスクエア3へと移動し、途中でスクエア2を飛び越えます。

挑戦してみよう（スコッティはランニングトラックを完了していなければいけませ

解答例:



挑戦してみよう（スコッティはランニングトラックを完了していなければいけませ

解答例（テキスト形式）：

```
BEGIN
    STEP (1)
    PICK UP
    TURN TO THE
    LEFT STEP (-2)
    PICK UP
    STEP (2)
    JUMP
    PICK UP
    STEP (2)
END
```

挑戦してみよう（スコッティはランニングトラックを完了していなければいけませ

解答例:

挑戦してみよう（スコッティはランニングトラックを完了していなければいけませ

解答例（テキスト形式）：

```
BEGIN
    STEP (1)
    TURN TO THE
    LEFT STEP (1)
    PICK UP
    JUMP
    PICK UP
    JUMP
    PICK UP
    STEP (2)
END
```

注意!

このクエストでは、ループ・アルゴリズムを使うと解答が容易になりますが、使う必要はありません。

授業中、生徒は直感的な傾向があり、例えば、**REPEAT**など、後のモジュールで導入されるコマンドであっても、このようなタイルを使います。

REPEATや**END IF / REPEAT**タイルは、モジュール1でも使うことができます。ループを構成する正しい方法は、モジュール3の導入で説明します。

挑戦してみよう（スコッティはランニングトラックを完了していなければいけませ

解答例:

挑戦してみよう（スコッティはランニングトラックを完了していなければいけませ

解答例（テキスト形式）：

```
BEGIN
    JUMP
    PICK UP
    TURN TO THE
    LEFT JUMP
    PICK UP
    TURN TO THE
    LEFT JUMP
    PICK UP
    STEP (1)
END
```

注意!

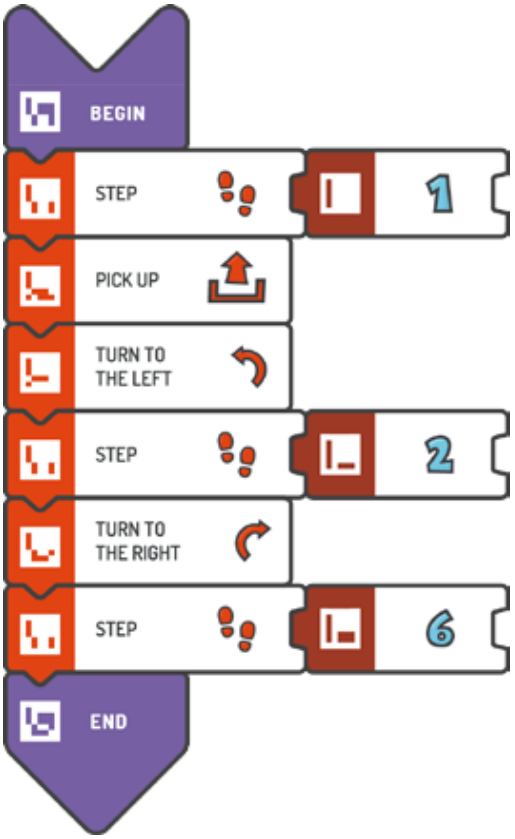
このクエストでは、生徒は、できるだけ短いプログラムでスコッティのための最適なルートを見つけようとします。

スコッティ・ゴー！教育版では、プレイヤーは解答の質に応じて星をもらいます。プログラムが最高の場合、星3つ、解答が少し効率的ではない場合は星2つ。星1つは、クエストは解決したけれども、プログラムが長すぎることを意味します。

星を1つ以上欲しい場合、クエストによっては、例えば**REPEAT**タイルのような特定のタイルを解答に使う必要があります。

挑戦してみよう（スコッティはランニングトラックを完了していなければいけませ

解答例:



挑戦してみよう（スコッティはランニングトラックを完了していなければいけませ

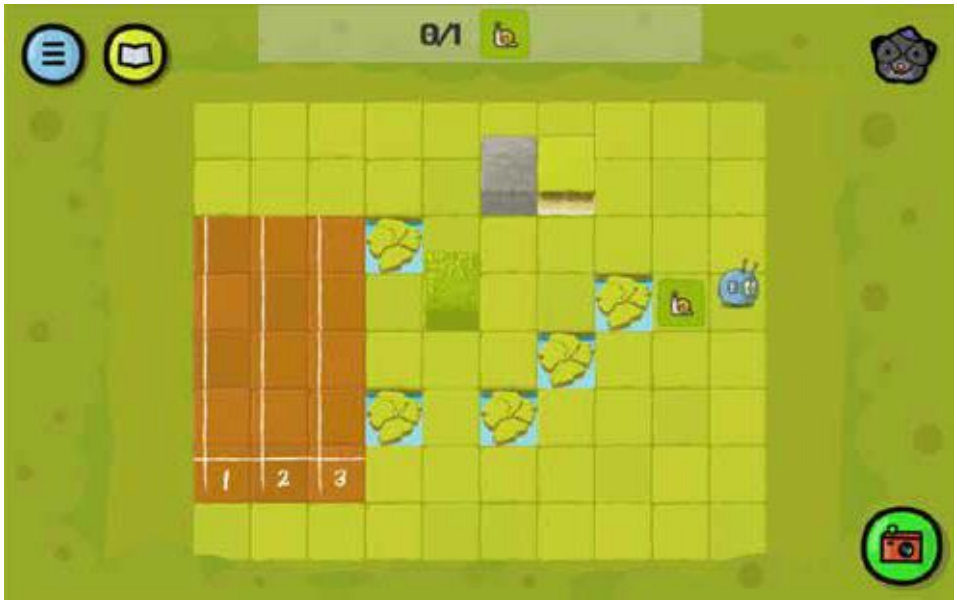
解答例（テキスト形式）：

```
BEGIN
    STEP (1)
    PICK UP
    TURN TO THE
    LEFT STEP (2)
    TURN TO THE
    RIGHT STEP (6)
END
```

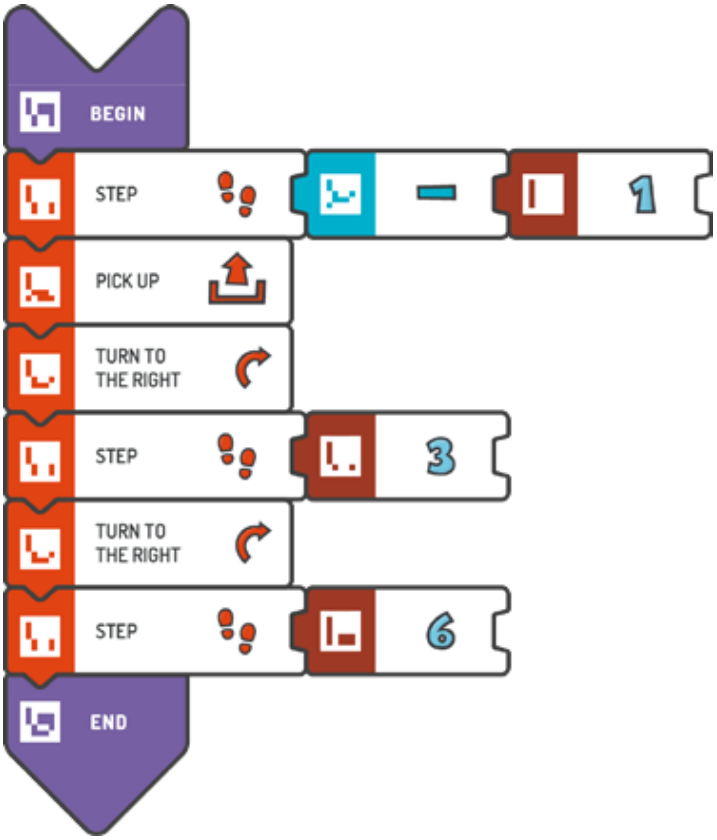
ヒント

このクエストでは、できるだけ短いプログラムで、スコッティのための最適なルートを見つけるよう、生徒にアドバイスしましょう。

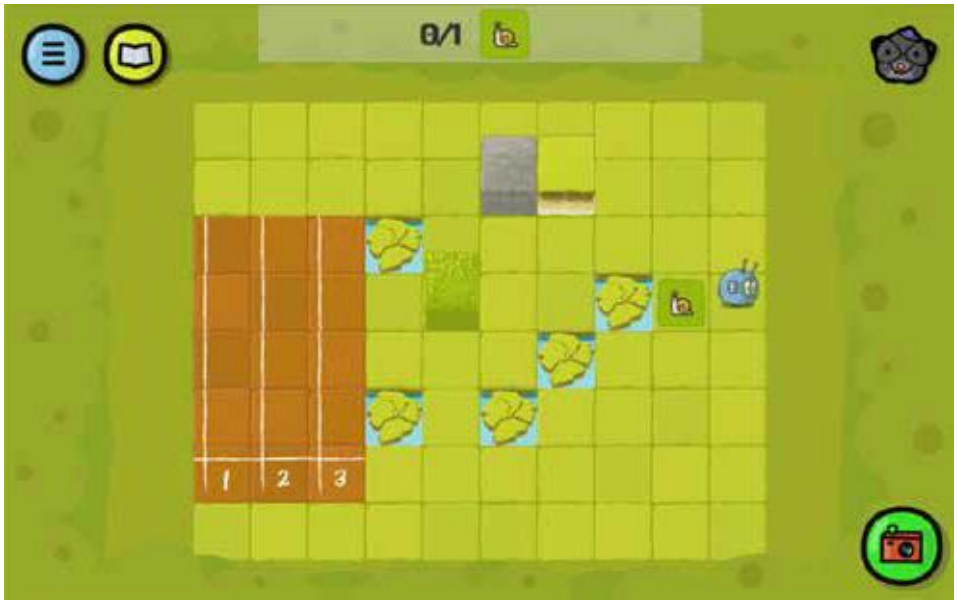
挑戦してみよう（スコッティはランニングトラックを完了していなければいけませ



解答例:



挑戦してみよう（スコッティはランニングトラックを完了していなければいけませ

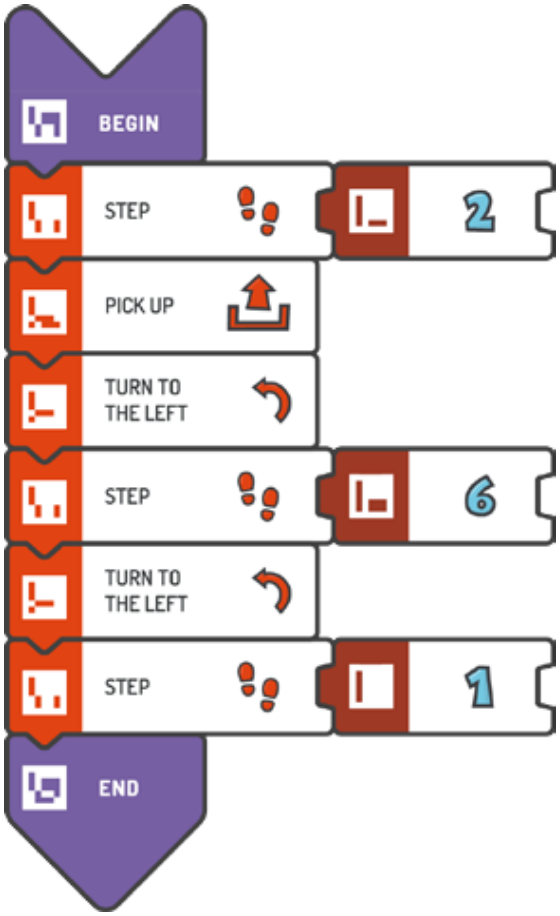


解答例（テキスト形式）：

```
BEGIN
    STEP (-1)
    PICK UP
    TURN TO THE
    RIGHT STEP (3)
    TURN TO THE
    RIGHT STEP (6)
END
```

挑戦してみよう（スコッティはランニングトラックを完了していなければいけませ

解答例:



挑戦してみよう（スコッティはランニングトラックを完了していなければいけませ

解答例（テキスト形式）：

```
BEGIN
    STEP (2)
    PICK UP
    TURN TO THE
    LEFT STEP (6)
    TURN TO THE
    LEFT STEP (1)
END
```

モジュール3
南アメリカ

動作を繰り返します。

ゲームボードからすべての羽を拾いましょう。

モジュール導入:
REPEATループ

モジュール1のクエストをいくつか完了した後、スコッティが何度も行わないといけない動作があることに気付く生徒がいるでしょう。時には、全く同じ順番で、この操作をする必要があります。そのうちのいくつかを自動化する方法を見つけると便利です。最も効率的な解答を見つけて作業を最適化することも学べます。操作を何度も繰り返す簡単な方法を探している場合、**最高のツールはループです。**

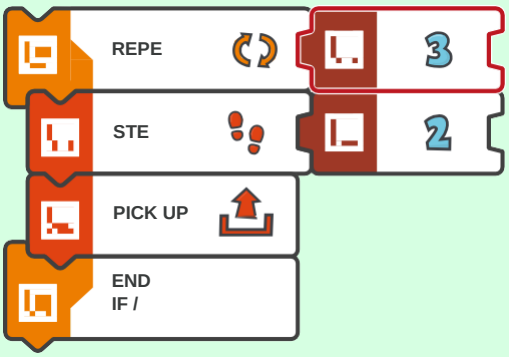
REPEATループは、使用頻度が最も高いループの1つです。スコッティ・ゴー!には、その最も簡単なバージョンがあります。ループを使いたい場合、**プログラムのどの部分を繰り返すか**を特定し、

次に何回繰り返すかを決めます。繰り返すコマンドを特定するプログラムは、2つのタイルで構成されます: **REPEAT**と**ENDループ**または**条件文**。このタイルは、一般的なプログラミング言語で使われる角かっこに相当します。

導入:
最初のループ

ループは、プログラミングで最も基本的で、最も頻繁に使われる概念の1つです。ループを使いこなすことは、重要なアルゴリズム能力でもあります。ループがなければ、多くのデジタル機器は効率よく機能できず、プログラムは長くなりすぎて、効率が悪くなるでしょう。望む目的を達成するまで、ループで一連のコマンドを繰り返すことができます。

このクエストでは、**REPEAT**ループを使うことを覚えておかなくてはいけません。ループは必ず**END IF / REPEAT**タイルで完了します。下の画像で、赤で囲まれたタイルは、ループでの一連のコマンドを繰り返す回数を特定しています:



上のタイルで定義されたループは、以下のようなテキスト形式に書き換えることができます:

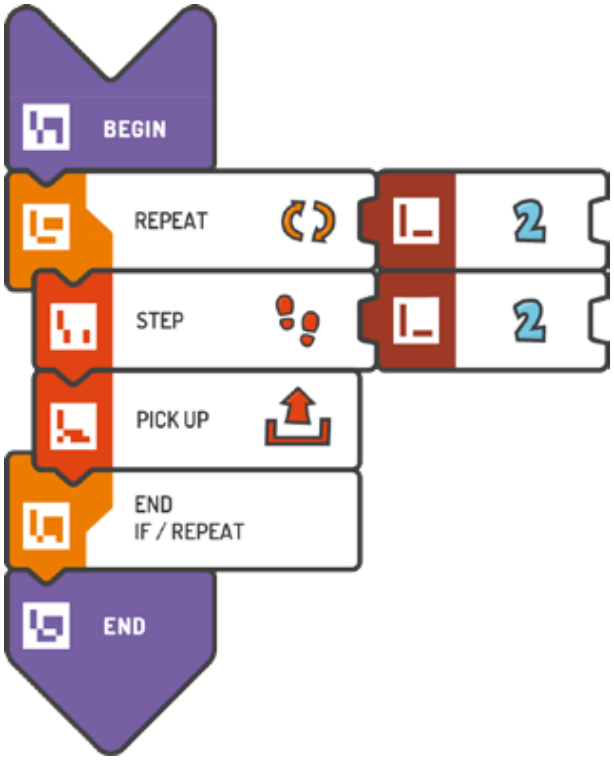
```
REPEAT (3) {
      STEP (2)
      PICK UP
}
```

- **REPEAT**(繰り返す回数){ は、**REPEAT**タイルが使われる場所と繰り返しの回数を示す。

-} は、**END IF / REPEAT**タイルが使われる場所を示す。

挑戦してみよう:

解答例:



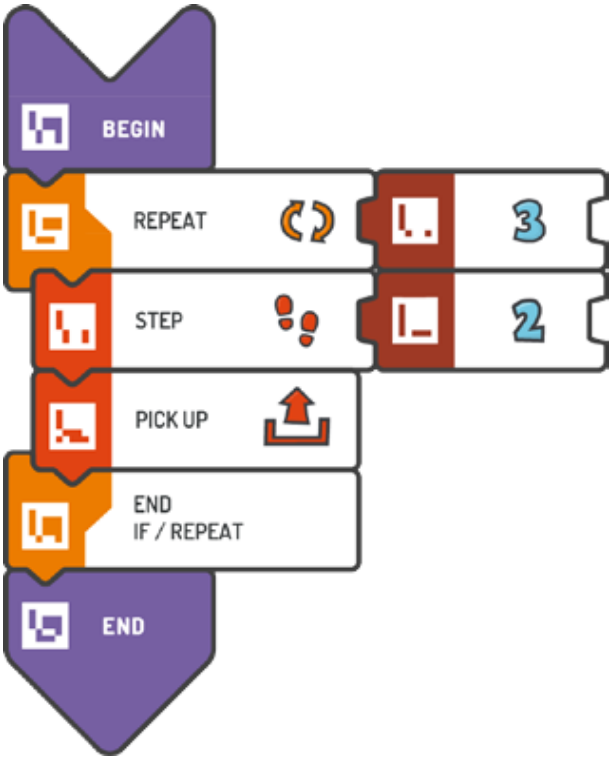
挑戦してみよう:

解答例（テキスト形式）：

```
BEGIN
    REPEAT (2) {
        STEP (2)
        PICK UP
    }
END
```

挑戦してみよう:

解答例:



挑戦してみよう:

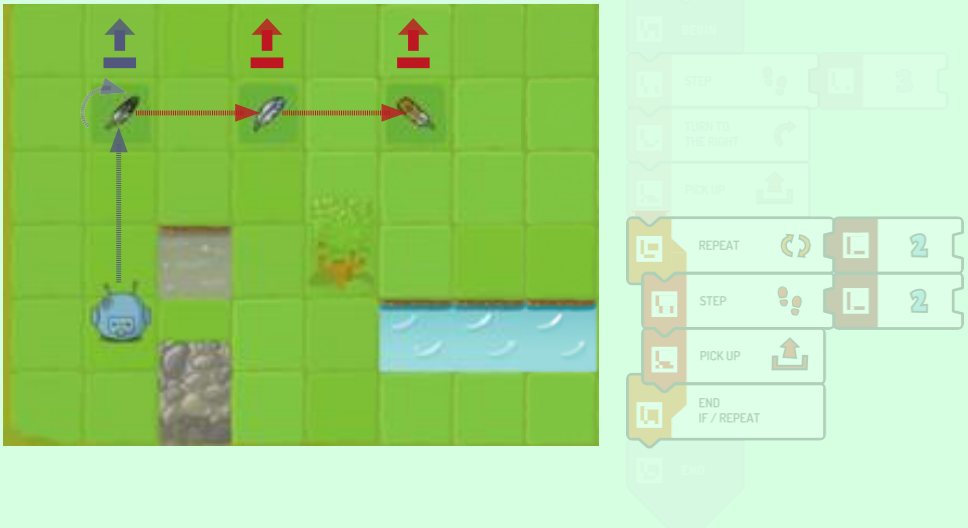
解答例（テキスト形式）：

```
BEGIN
    REPEAT (3) {
        STEP (2)
        PICK UP
    }
END
```

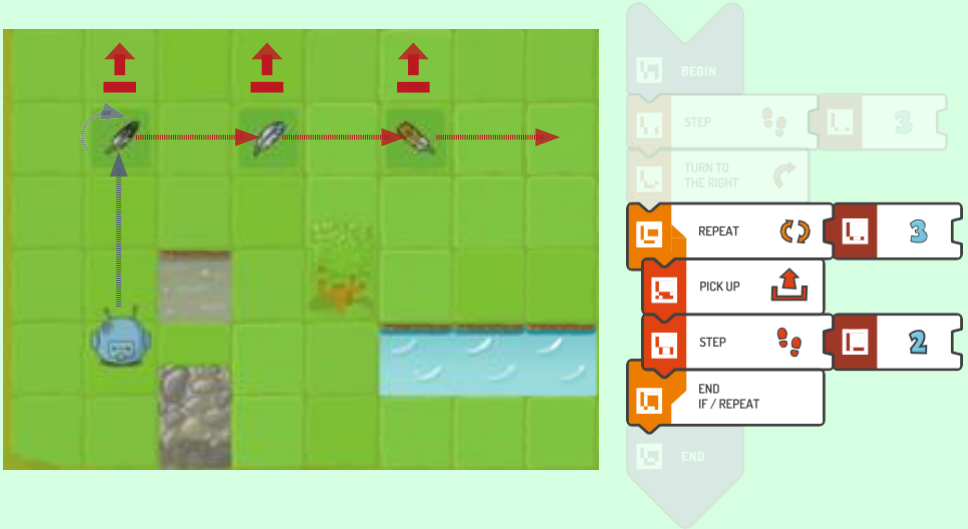
導入:
ループ内やループ外での動作

プログラムのコマンドの中には、場合によっては繰り返しが必要なものと、必要ではないものがあります。繰り返したくないコマンドはループ外に置くように、生徒に教えましょう。ループで極めて重要なのは、出来事の順番であり、プログラムのこの部分を書き始める前にイメージしておかなくてはいけません。操作の順番は、スコッティの動作に影響します。

下の例では、ループ外の最初の羽を拾う場合、ループを2回繰り返さなくてはいけません。



ループ内の最初の羽を拾う場合、ループを3回繰り返さなくてはいけません。



どちらのループにも2つのコマンドがあります。コードの行数が少ないのは、2つの解答のどちらかであるかに注目しましょう。

挑戦してみよう:


0/4

解答例:


BEGIN
STEP
3
PICK UP
TURN TO
THE RIGHT
REPEAT
3
STEP
2
PICK UP
END
IF / REPEAT
END

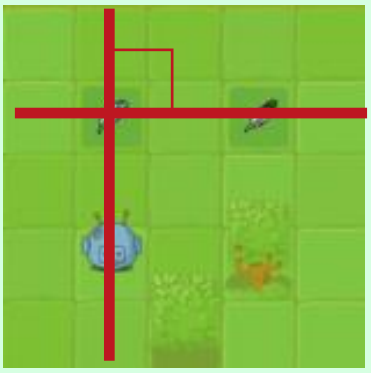
挑戦してみよう:

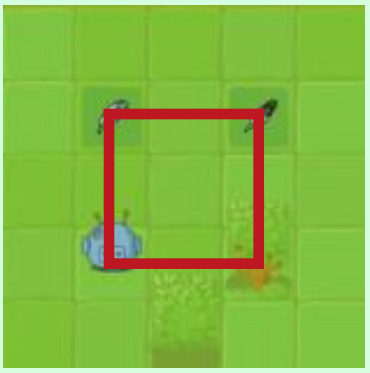
解答例（テキスト形式）：

```
BEGIN
    STEP (3)
    PICK UP
    TURN TO THE
    RIGHT REPEAT (3)
    {
        STEP (2)
        PICK UP
    }
END
```

導入:
数学や物理学の概念を教えるツールとしてのスコッティ・ゴー！教育版

このクエストでは、スコッティが羽を拾いながら通り抜けるルートは、スクエアの一部です。スコッティはこのスクエアに沿って動き、羽はその角に置かれています。正しい角度、平行線、垂線などについて生徒に教え始める絶好の機会です。



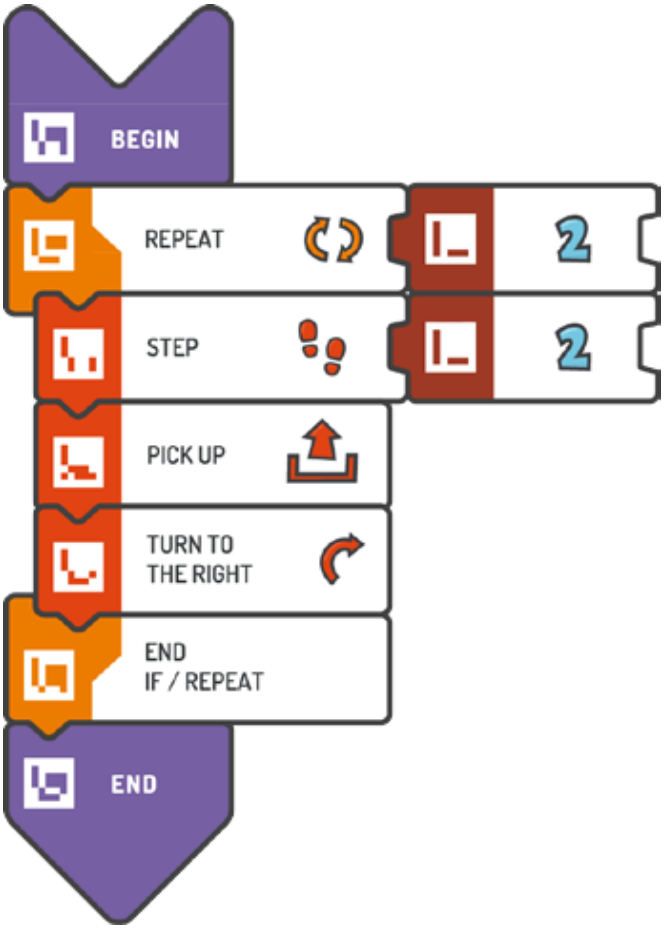


スコッティ・ゴー！教育版は、長方形の格子でプレイします。これは、ベクトル、デカルト座標系、合力などの概念を説明する自然なツールになっています。

挑戦してみよう:


0/2

解答例:


BEGIN
REPEAT
2
STEP
2
PICK UP
TURN TO
THE RIGHT
END
IF / REPEAT
END

挑戦してみよう:

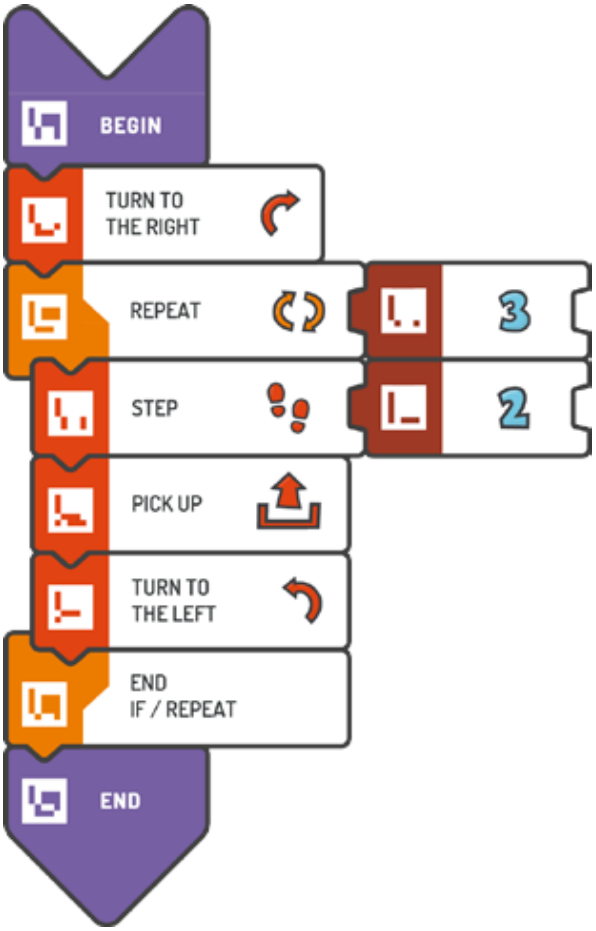
解答例（テキスト形式）：

```
BEGIN
    REPEAT (2) {
        STEP (2)
        PICK UP
        TURN TO THE RIGHT
    }
END
```

挑戦してみよう:


0/3

解答例:


BEGIN
TURN TO THE RIGHT
REPEAT
3
STEP
2
PICK UP
TURN TO THE LEFT
END IF / REPEAT
END

挑戦してみよう:

解答例（テキスト形式）：

```
BEGIN
    TURN TO THE
    RIGHT REPEAT (3)
    {
        STEP (2)
        PICK UP
        TURN TO THE LEFT
    }
END
```

挑戦してみよう:


0/6

解答例:


BEGIN
REPEAT
3
STEP
2
PICK UP
TURN TO
THE LEFT
STEP
2
PICK UP
TURN TO
THE RIGHT
END
IF / REPEAT
END

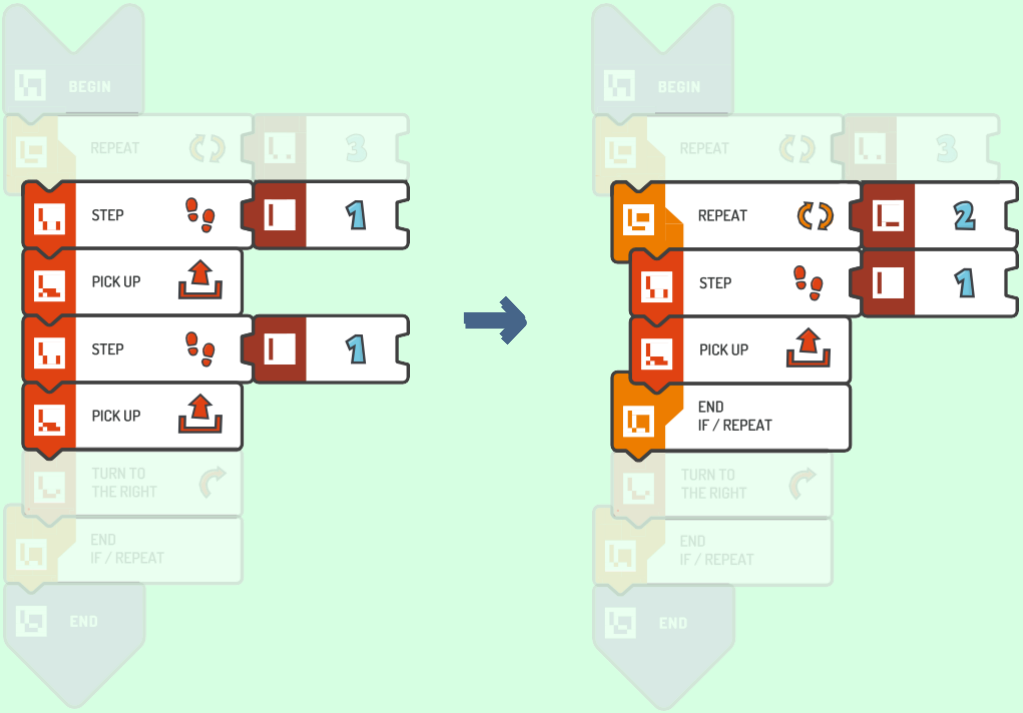
挑戦してみよう:

解答例（テキスト形式）：

```
BEGIN
    REPEAT (3) {
        STEP (2)
        PICK UP
        TURN TO THE LEFT
        STEP (2)
        PICK UP
        TURN TO THE RIGHT
    }
END
```

導入:
入れ子ループ

コードの行数を減らすために、既存の**REPEAT**ループに別のループを追加で入れることができます。このようなループは入れ子ループと呼ばれています。このアルゴリズム的思考は、コードをさらに最適化することにつながり、アルゴリズムの複雑度に影響します。入れ子ループを使いこなせると、アルゴリズム的直感が養われるだけでなく、さらに良くするために、コードを改善する必要性も理解できるようになります。



下記に、これをテキスト形式で示しています：

```
BEGIN
    REPEAT (3) {
        PICK UP
        STEP (1)
        PICK UP
        STEP (1)
        TURN TO THE RIGHT
    }
END
```

```
BEGIN
    REPEAT (3) {
        REPEAT (2) {
            PICK UP
            STEP (1)
        }
        TURN TO THE RIGHT
    }
END
```

挑戦してみよう:


0/6

解答例:


BEGIN
REPEAT
3
REPEAT
2
STEP
2
PICK UP
END
IF / REPEAT
TURN TO
THE RIGHT
END
IF / REPEAT
END

挑戦してみよう:

解答例（テキスト形式）：

```
BEGIN
    REPEAT (3) {
        REPEAT (2) {
            STEP (2)
            PICK UP
        }
        TURN TO THE RIGHT
    }
END
```

挑戦してみよう:

解答例:


BEGIN
REPEAT
2
REPEAT
3
STEP
2
PICK UP
END
IF / REPEAT
TURN TO
THE LEFT
STEP
1
TURN TO
THE LEFT
END
IF / REPEAT
END

挑戦してみよう:

解答例（テキスト形式）：

```
BEGIN
    REPEAT (2) {
        REPEAT (3) {
            STEP (2)
            PICK UP
        }
        TURN TO THE
        LEFT STEP (1)
        TURN TO THE LEFT
    }
END
```

モジュール4
北アメリカ

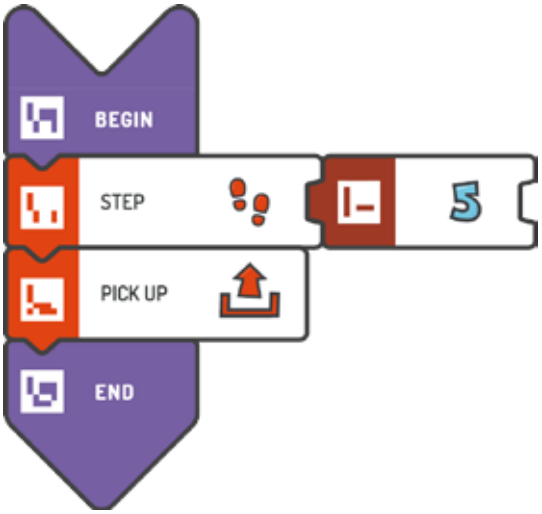
近くの岩には、金がたくさんあります。
スコッティの友達、ビーバーの助けを借りて、スコッティが金を集めるのを手伝いましょう。

導入:
橋としてのビーバー

川を渡るには、スコッティはビーバーの頭を踏んで行かなくてはいけません。生徒はタイルを使う必要があります。タイルで新しい結果を得る方法は学習済みです。

挑戦してみよう:

解答例:



解答例（テキスト形式）：

```
BEGIN
    STEP (5)
    PICK UP
END
```

挑戦してみよう:

解答例:

挑戦してみよう:

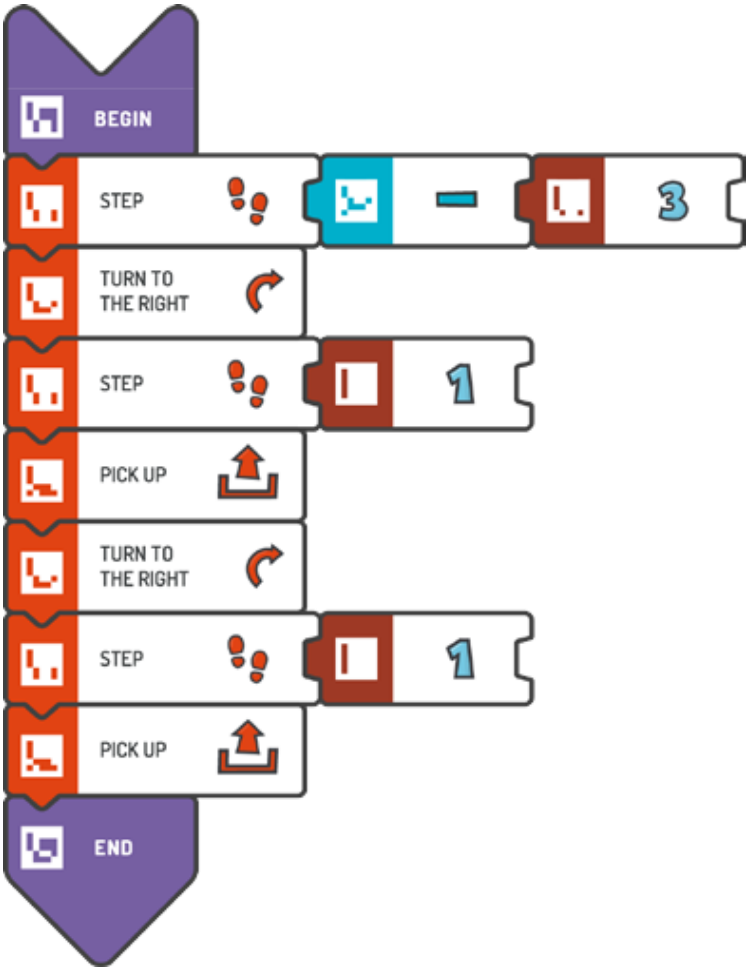
解答例（テキスト形式）：

```
BEGIN
    STEP (5)
    PICK UP
    TURN TO THE
    RIGHT STEP (1)
    PICK UP
END
```

挑戦してみよう:


0/2

解答例:


BEGIN
STEP
3
TURN TO THE RIGHT
STEP
1
PICK UP
TURN TO THE RIGHT
STEP
1
PICK UP
END

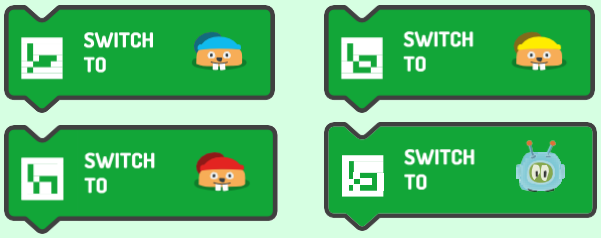
挑戦してみよう:

解答例（テキスト形式）：

```
BEGIN
    STEP (-3)
    TURN TO THE
    RIGHT STEP (1)
    PICK UP
    TURN TO THE
    RIGHT STEP (1)
    PICK UP
END
```

導入:
動いているビーバー

ビーバーは、橋やいかだとして使えます。新しいタイルを使って、ビーバーがどのように泳ぐかをプログラムできます:



注意!

ビーバーは、それぞれ違う色の帽子（**黄色、青、赤**）をかぶっているため、ボックスから正しいタイルを選ぶことが重要です。

挑戦してみよう:


0/2

解答例:


BEGIN
SWITCH TO
STEP
4
SWITCH TO
STEP
5
PICK UP
STEP
1
PICK UP
END

挑戦してみよう:

解答例（テキスト形式）：

```
BEGIN
    SWITCH TO - BLUE
    BEAVER: STEP (4)
    SWITCH TO -
    SCOTTIE: STEP (5)
    PICK
    UP
    STEP
    (1)
    PICK
    UP
END
```

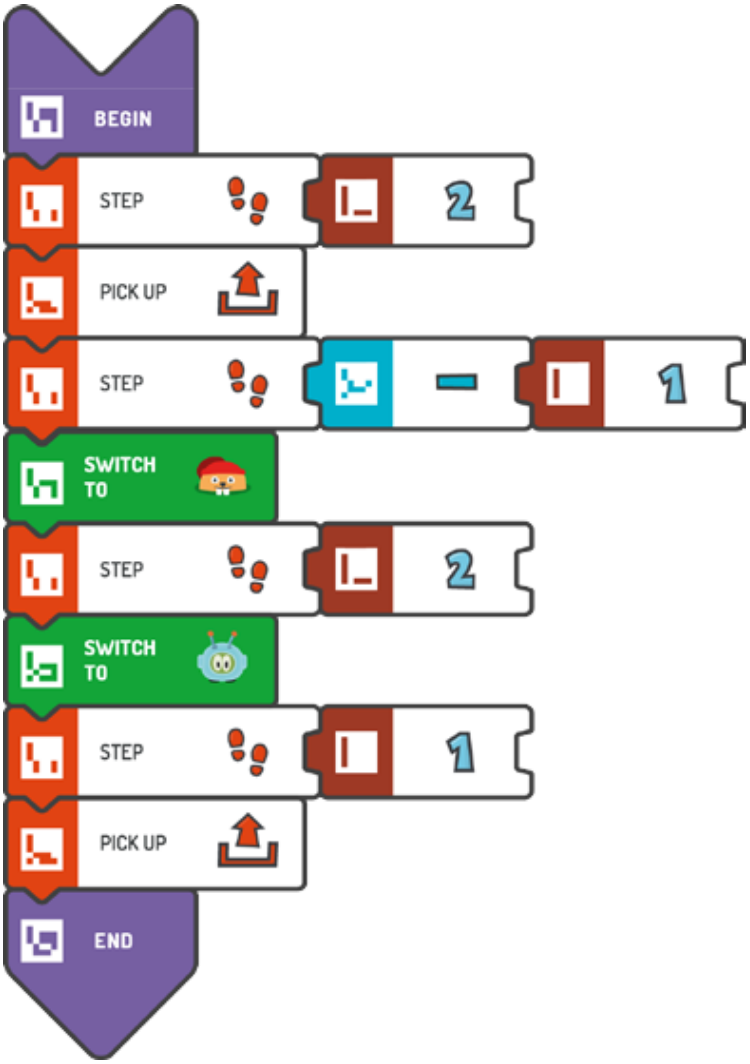
ヒント

もっと素早く、スコッティを目的地に移動させたい場合、ビーバーに乗せて泳がせましょう。できるだけ短いプログラムにするよう、スコッティのために最適なルートを探す絶好のチャンスになります。

挑戦してみよう:


0/2

解答例:


BEGIN
STEP
2
PICK UP
STEP
1
SWITCH TO
STEP
2
SWITCH TO
STEP
1
PICK UP
END

挑戦してみよう:

解答例（テキスト形式）：

```
BEGIN
    STEP (2)
    PICK UP
    STEP (-1)
    SWITCH TO - RED
    BEAVER: STEP (2)
    SWITCH TO -
    SCOTTIE: STEP (1)
    PICK UP
END
```

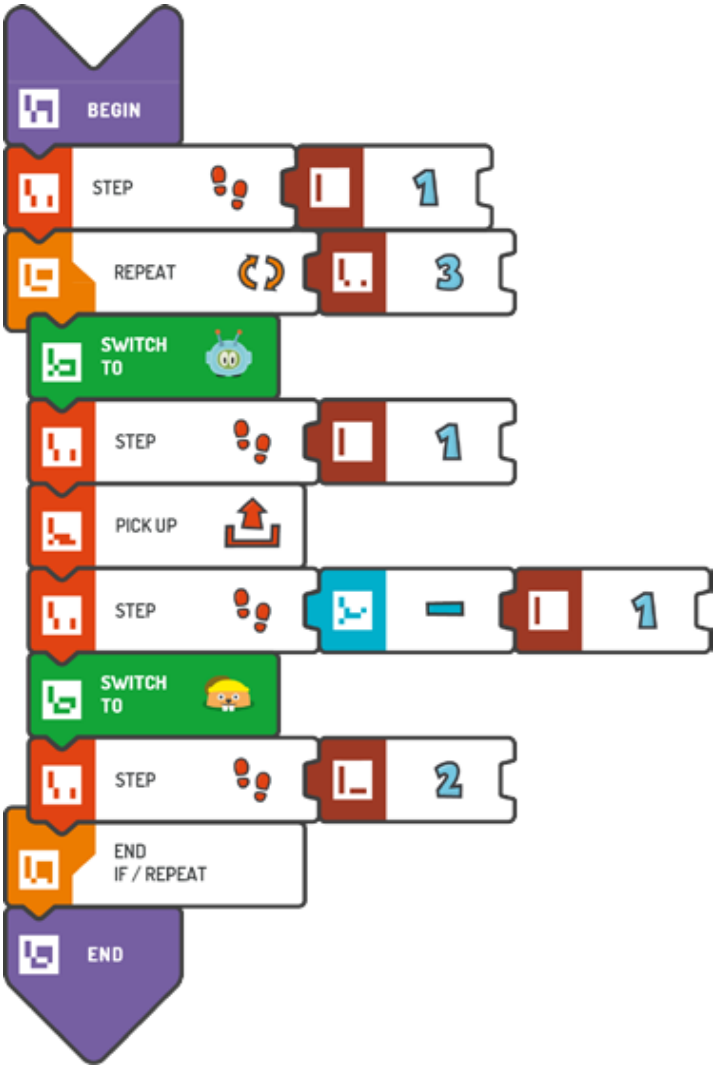
導入: REPEATループやSWITCH TOタイルを一緒に使う

このクエストでは、ゲームキャラクターを順番に動かすために、**REPEAT**ループと**SWITCH TO**タイルを一緒に使います。すでに知っている概念を集約し、コードの個別の部分を組み合わせる能力を養うのに役立ちます。

挑戦してみよう:


0/3

解答例:


BEGIN
STEP
1
REPEAT
3
SWITCH TO
STEP
1
PICK UP
STEP
1
SWITCH TO
STEP
2
END IF / REPEAT
END

挑戦してみよう:

解答例（テキスト形式）：

```
BEGIN
    STEP (1)
    REPEAT (3) {
        SWITCH TO –
        SCOTTIE : STEP (1)
        PICK UP
        STEP (-1)
        SWITCH TO - YELLOW
        BEAVER: STEP (2)
    }
END
```

注意!
ビーバーの向きを変える

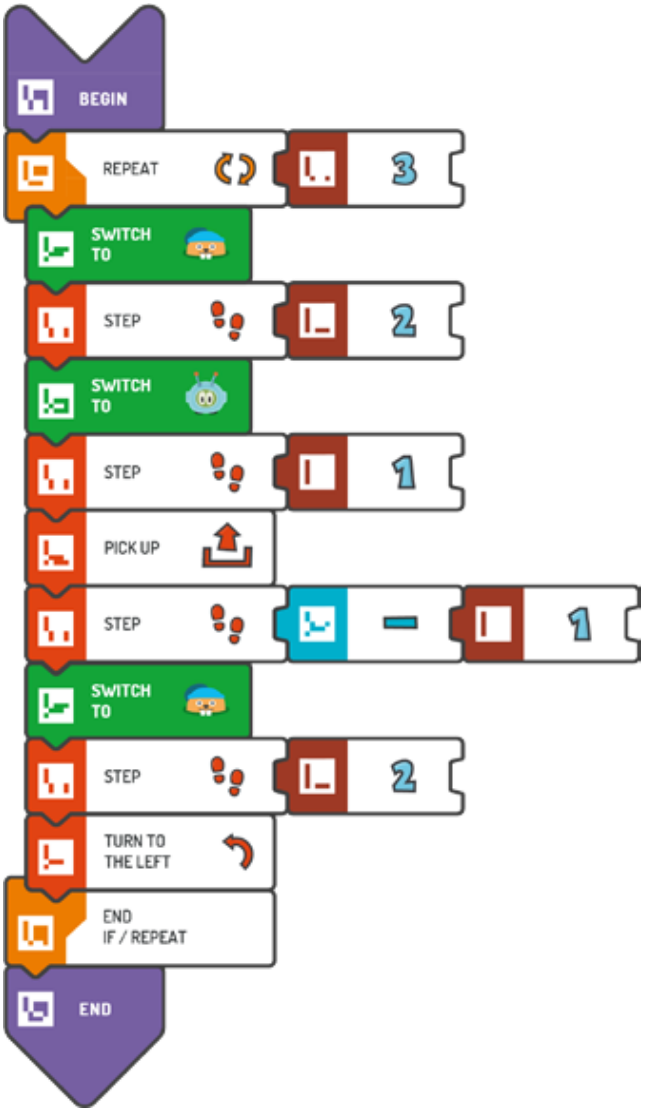
ビーバーも向きを変えます。ビーバーが向きを変えているときにスコッティがビーバーの頭の上で泳いだ場合、スコッティもビーバーと一緒に向きを変えます。.

これで、ビーバーの助けを借りることができます。**REPEAT**ループを使って、
後のクエストを完了し、ビーバーの向きを変えましょう。

挑戦してみよう:


0/3

解答例:


BEGIN
REPEAT
3
SWITCH TO
STEP
2
SWITCH TO
STEP
1
PICK UP
STEP
1
SWITCH TO
STEP
2
TURN TO THE LEFT
END IF / REPEAT
END

挑戦してみよう:

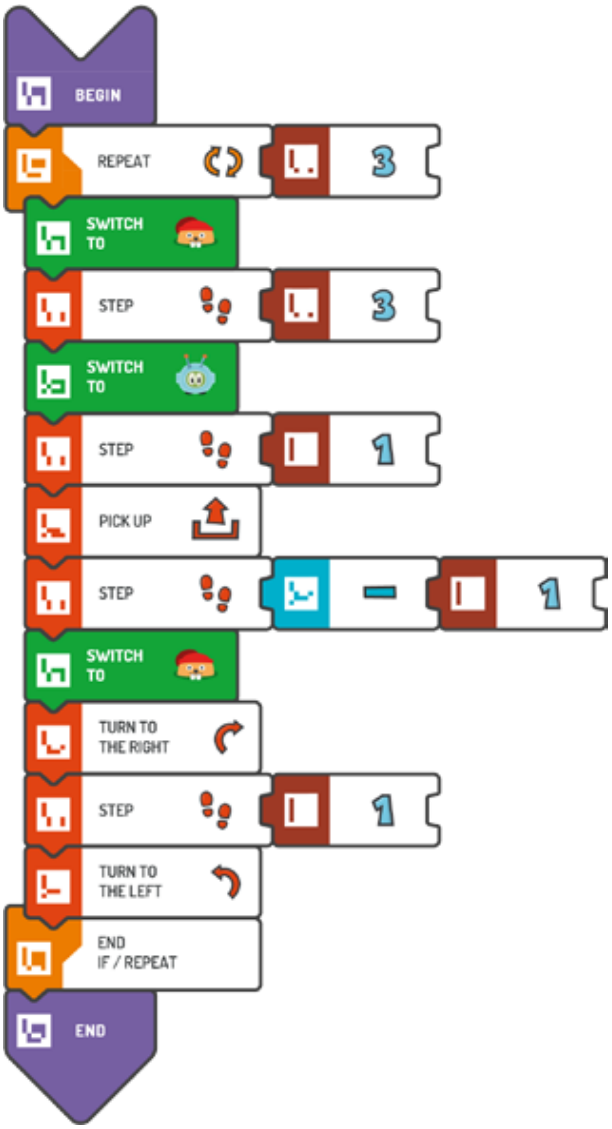
解答例（テキスト形式）：

```
BEGIN
    REPEAT (3) {
        SWITCH TO - BLUE
        BEAVER: STEP (2)
        SWITCH TO -
        SCOTTIE: STEP (1)
        PICK UP
        STEP (-1)
        SWITCH TO - BLUE
        BEAVER: STEP (2)
        TURN TO THE LEFT
    }
END
```

挑戦してみよう:


0/3

解答例:


BEGIN
REPEAT
3
SWITCH TO
STEP
3
SWITCH TO
STEP
1
PICK UP
STEP
1
SWITCH TO
TURN TO THE RIGHT
STEP
1
TURN TO THE LEFT
END IF / REPEAT
END

挑戦してみよう:

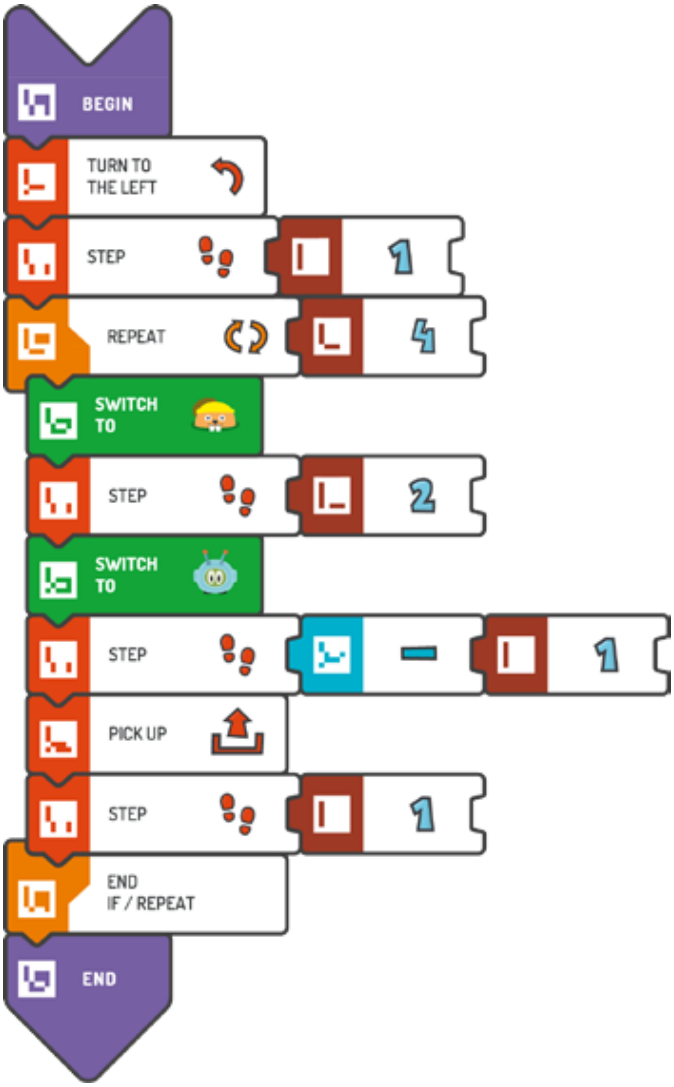
解答例（テキスト形式）：

```
BEGIN
    REPEAT (3) {
        SWITCH TO - RED
        BEAVER: STEP (3)
        SWITCH TO -
        SCOTTIE: STEP (1)
        PICK UP
        STEP (-1)
        SWITCH TO - RED
        BEAVER: TURN TO THE
        RIGHT
        STEP (1)
        TURN TO THE LEFT
    }
END
```

挑戦してみよう:


0/4

解答例:


BEGIN
TURN TO THE LEFT
STEP
1
REPEAT
4
SWITCH TO
STEP
2
SWITCH TO
STEP
1
PICK UP
STEP
1
END IF / REPEAT
END

挑戦してみよう:

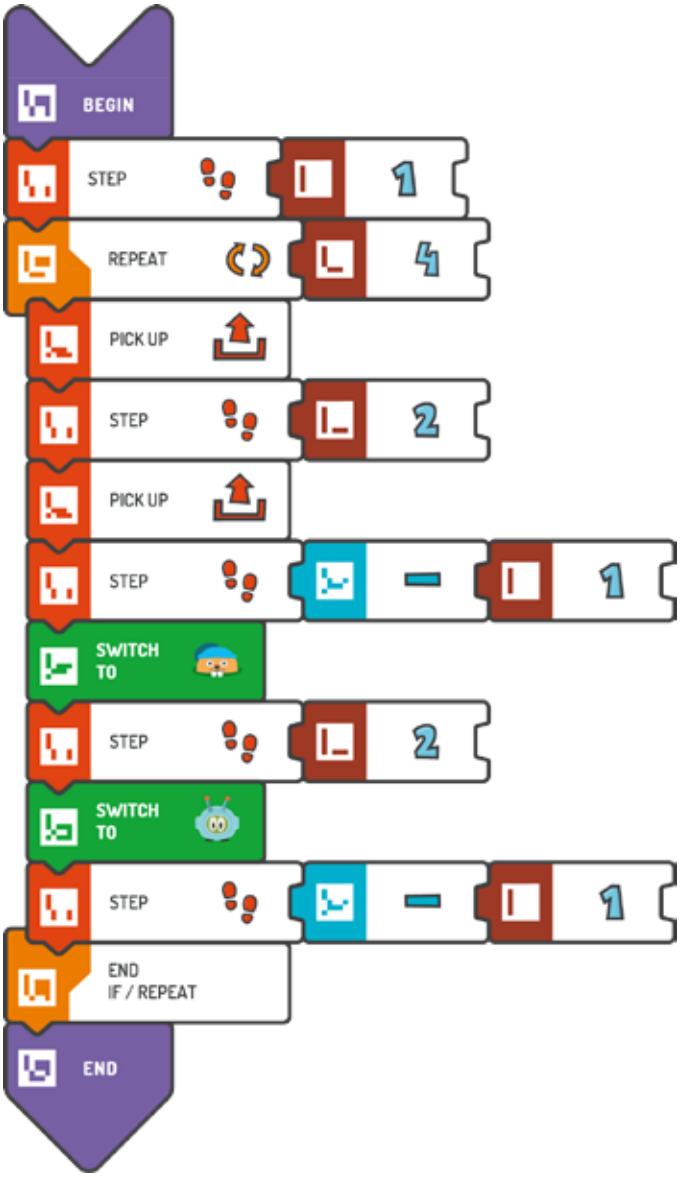
解答例（テキスト形式）：

```
BEGIN
    TURN TO THE
        LEFT STEP (1)
        REPEAT (4) {
            SWITCH TO - YELLOW
            BEAVER: STEP (2)
            SWITCH TO -
            SCOTTIE: STEP (-1)
            PICK UP
            STEP (1)
        }
END
```

挑戦してみよう:


0/8

解答例:


BEGIN
STEP
1
REPEAT
4
PICK UP
STEP
2
PICK UP
STEP
1
SWITCH TO
STEP
2
SWITCH TO
STEP
1
END IF / REPEAT
END

挑戦してみよう:

解答例（テキスト形式）：

```
BEGIN
    STEP (1)
    REPEAT (4) {
        PICK UP
        STEP (2)
        PICK UP
        STEP (-1)
        SWITCH TO - BLUE
        BEAVER: STEP (2)
        SWITCH TO -
        SCOTTIE: STEP (-1)
    }
END
```

モジュール5
南極

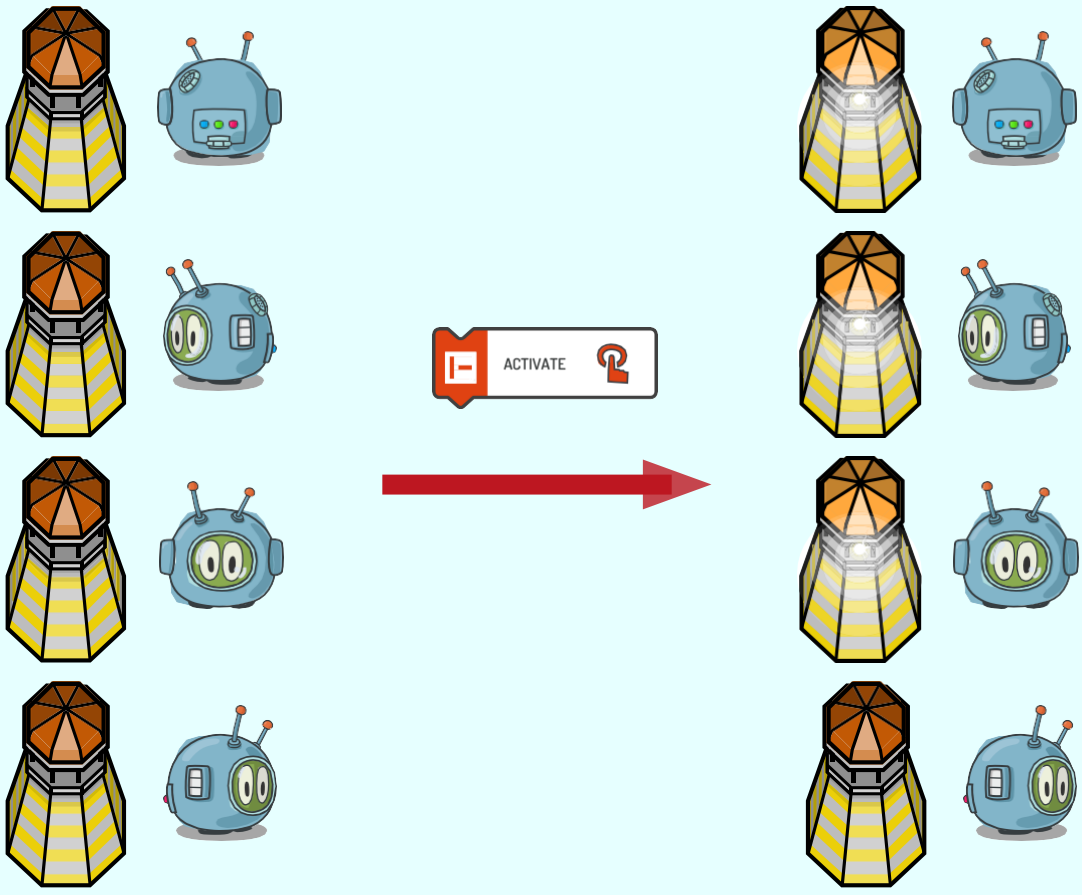
スコッティは船乗りを助けます。このクエストでは、半島の灯台すべてのスイッチを入れて、宇宙船の部品を運んでいる船に港の位置を知らせましょう。

導入 TOモジュール 1/2:
スコッティの周囲との関わり – ACTIVATEと PLACEタイル

このモジュールでは、スコッティはものと関わり続け、**周囲を変え始めます**。

ACTIVATEと**PLACE**タイルが導入されます。

ACTIVATEタイルをプログラムに置いて、スコッティが隣にある灯台に背を向けていなければ、灯台のスイッチを入れさせます。



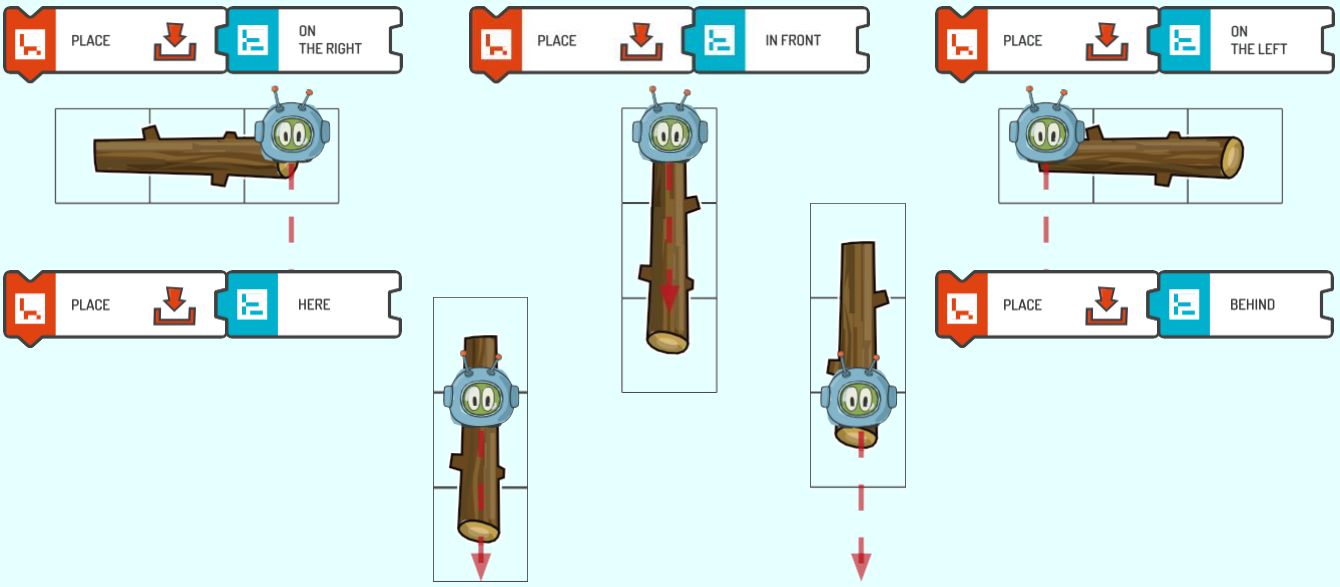
導入 TOモジュール 2/2:
スコッティの周囲との関わり – ACTIVATEと PLACEタイル

このモジュールでは、スコッティはものと関わり続け、**周囲を変え始めます**。

ACTIVATEと**PLACE**タイルが導入されます。

このモジュールでは、スコッティが（**PICK UP**タイルを使って）拾う丸太も取り入れています。

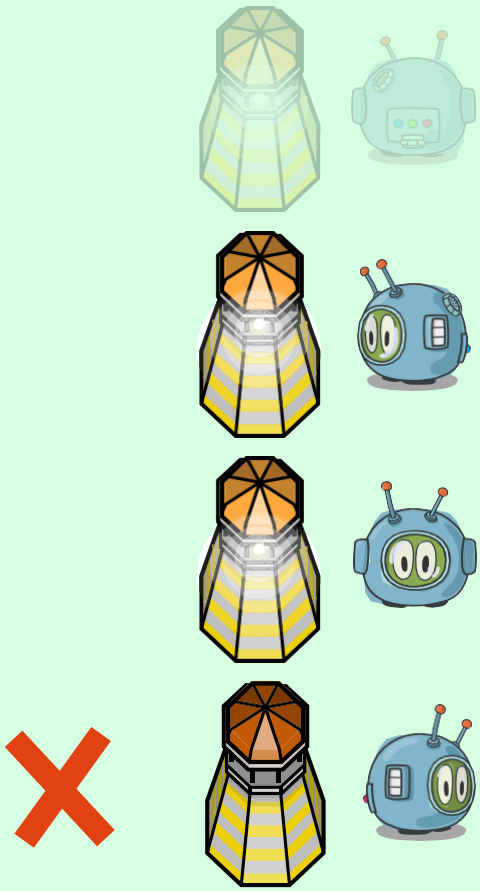
スコッティが丸太を拾うと、**HERE**、**IN FRONT**、**ON THE RIGHT**、**ON THE LEFT**、**BEHIND**タイルを**PLACE**タイルのパラメータとして使って、ゲームボード上の5つの異なる場所に置くことができます。



導入:
ACTIVATEタイル

新しい**ACTIVATE**タイルで灯台のスイッチを入れます。

灯台にスイッチを入れるには、スコッティは灯台の方を向くか、その隣に立たなくてはいけません。スコッティが灯台に背を向けると、**ACTIVATE**タイルが機能しないため、灯台に背を向けてはいけません。



挑戦してみよう:

解答例:

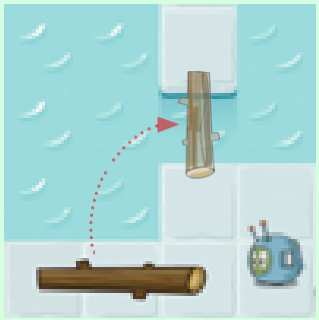
挑戦してみよう:

解答例（テキスト形式）：

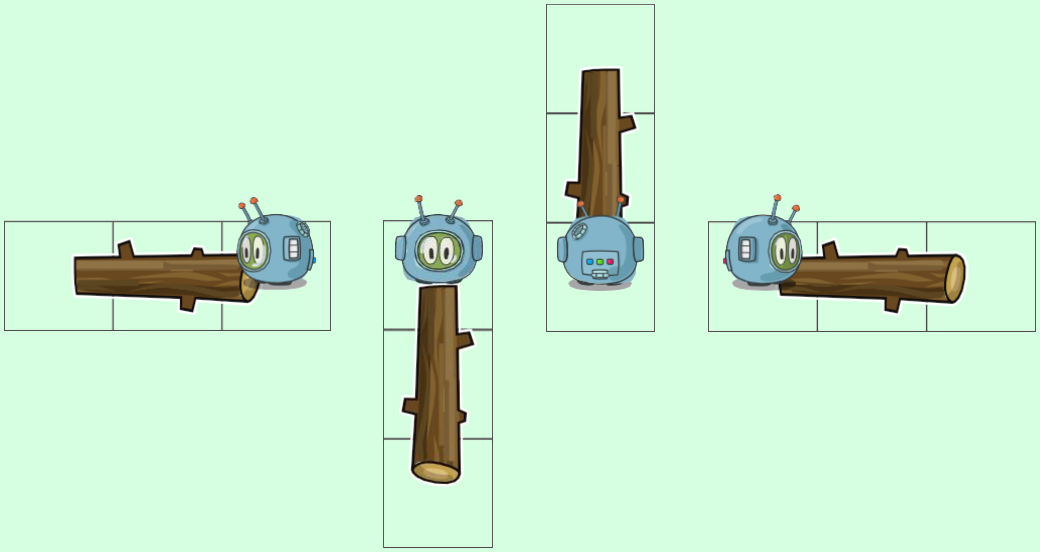
```
BEGIN
    STEP (1)
    TURN TO THE
    RIGHT STEP (4)
    ACTIVATE
END
```

導入:
PLACEタイル

スコッティに渓谷を渡らせるには、丸太を拾い、新しい**PLACE**タイルを使って、正しい位置に丸太を置きます。



渓谷に丸太を置くには、スコッティは渓谷の前のスクエアに丸太を立てなくてはいけません。また、スコッティが渓谷の方を向いている場合、例えば、**PLACE(IN FRONT)**のような**PLACE**タイルに適切なパラメータを選ぶ必要があります。



丸太を拾うには、丸太が架かっている3つのスクエアの**いずれか**に、スコッティが立っていなければいけません。

挑戦してみよう:


0/1

解答例:


BEGIN
STEP
1
PICK UP
TURN TO
THE RIGHT
STEP
1
PLACE
IN FRONT
STEP
3
ACTIVATE
END

挑戦してみよう:

解答例（テキスト形式）：

```
BEGIN
    STEP (1)
    PICK UP
    TURN TO THE
    RIGHT STEP (1)
    PLACE (IN
    FRONT) STEP (3)
    ACTIVATE
END
```

挑戦してみよう:


0/1

解答例:


BEGIN
STEP
3
PICK UP
TURN TO
THE RIGHT
STEP
4
PLACE
IN FRONT
STEP
2
ACTIVATE
END

挑戦して

解答例（テキスト形式）：

```
BEGIN
    STEP (3)
    PICK UP
    TURN TO THE
    RIGHT STEP (4)
    PLACE (IN
    FRONT) STEP (2)
    ACTIVATE
END
```

挑戦してみよう:


0/1

解答例:


BEGIN
STEP
3
PICK UP
TURN TO
THE LEFT
PLACE
IN FRONT
STEP
3
ACTIVATE
END

挑戦してみよう:

解答例（テキスト形式）：

```
BEGIN
    STEP (3)
    PICK UP
    TURN TO THE
    LEFT PLACE (IN
    FRONT) STEP (3)
    ACTIVATE
END
```

注意

後のクエストでは、**REPEAT**タイルを使い、新しいループを作成し、プログラムを最適化し続けるようにしましょう。

挑戦してみよう:


0/1

解答例:


BEGIN
STEP
1
REPEAT
2
STEP
2
PICK UP
PLACE
IN FRONT
END
IF / REPEAT
STEP
2
ACTIVATE
END

挑戦してみよう:

解答例（テキスト形式）：

```
BEGIN
    STEP (1)
    REPEAT (2) {
        STEP (2)
        PICK UP
        PLACE (IN FRONT)
    }
    STEP (2)
    ACTIVAT
EN  E
D
```

挑戦してみよう:


0/4

解答例:


BEGIN
SWITCH TO
STEP
2
SWITCH TO
STEP
1
REPEAT
4
SWITCH TO
STEP
2
ACTIVATE
END IF / REPEAT
END

挑戦してみよう:

解答例（テキスト形式）：

```
BEGIN
    SWITCH TO - YELLOW
    BEAVER: STEP (2)
    SWITCH TO - BLUE
    BEAVER: STEP (1)
    REPEAT (4) {
        SWITCH TO -
        SCOTTIE: STEP (2)
        ACTIVATE
    }
END
```

導入:
ビーバーにスイッチを入れて、流水を動かす

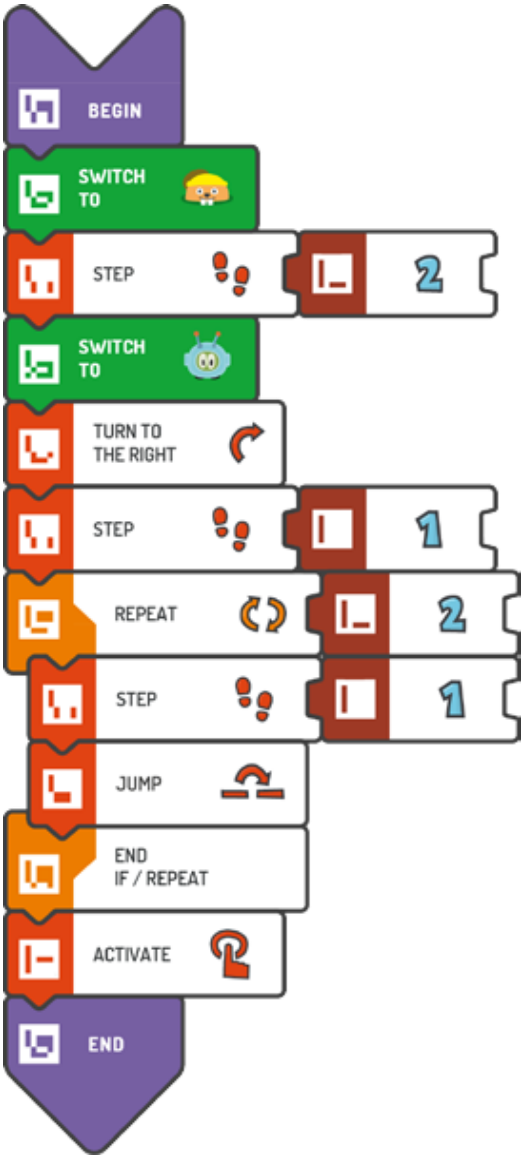
ビーバーをコントロールする際、小さな流氷を動かして、スコッティの新しいルートを作ることができます。このクエストでは、\nalityを使います。

生徒に必ず**REPEAT**ループを使わせましょう。

挑戦してみよう:


0/1

解答例:


BEGIN
SWITCH TO
STEP
2
SWITCH TO
TURN TO THE RIGHT
STEP
1
REPEAT
2
STEP
1
JUMP
END IF / REPEAT
ACTIVATE
END

挑戦してみよう:

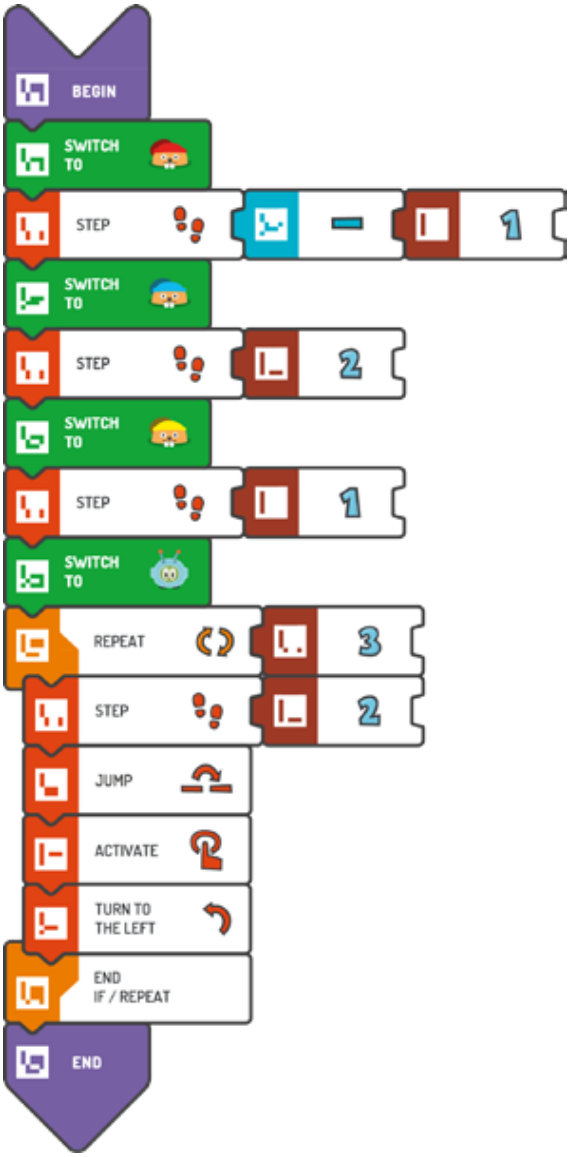
解答例（テキスト形式）：

```
BEGIN
    SWITCH TO - YELLOW
    BEAVER: STEP (2)
    SWITCH TO -
    SCOTTIE: TURN TO
    THE RIGHT STEP (1)
    REPEAT (2) {
        STEP (1)
        JUMP
    }
    ACTIVAT
EN  E
D
```

挑戦してみよう:


0/3

解答例:


BEGIN
SWITCH TO
STEP
1
SWITCH TO
STEP
2
SWITCH TO
STEP
1
SWITCH TO
REPEAT
3
STEP
2
JUMP
ACTIVATE
TURN TO THE LEFT
END IF / REPEAT
END

挑戦してみよう:

解答例（テキスト形式）：

```
BEGIN
    SWITCH TO - RED
    BEAVER: STEP (-1)
    SWITCH TO - BLUE
    BEAVER: STEP (2)
    SWITCH TO - YELLOW
    BEAVER: STEP (1)
    SWITCH TO -
    SCOTTIE: REPEAT
    (3) {
        STEP (2)
        JUMP
        ACTIVAT
        E
        TURN TO THE LEFT
    }
END
```

ヒント:
キャラクターの選択

（できるだけ短く）コードを最適化するプログラム用のキャラクターを選びます。

挑戦してみよう:


0/3

解答例:


BEGIN
SWITCH TO
TURN TO THE RIGHT
STEP
4
SWITCH TO
STEP
5
SWITCH TO
REPEAT
3
STEP
1
JUMP
STEP
1
JUMP
ACTIVATE
TURN TO THE LEFT
END IF / REPEAT
END

挑戦してみよう:

解答例（テキスト形式）：

```
BEGIN
    SWITCH TO - BLUE
    BEAVER: TURN TO THE
    RIGHT
    STEP (4)
    SWITCH TO - RED
    BEAVER: STEP (5)
    SWITCH TO -
    SCOTTIE: REPEAT
    (3) {
        STEP (1)
        JUMP
        STEP (1)
        JUMP
        ACTIVAT
        E
        TURN TO THE LEFT
    }
END
```

挑戦してみよう:


0/3

解答例:


BEGIN
SWITCH TO
TURN TO THE RIGHT
STEP
2
SWITCH TO
STEP
2
SWITCH TO
REPEAT
3
JUMP
JUMP
ACTIVATE
TURN TO THE LEFT
END IF / REPEAT
END

挑戦してみよう:

解答例（テキスト形式）：

```
BEGIN
    SWITCH TO - BLUE
    BEAVER: TURN TO THE
    RIGHT
    STEP (2)
    SWITCH TO - RED
    BEAVER: STEP (2)
    SWITCH TO -
    SCOTTIE: REPEAT
    (3) {
        JUMP
        JUMP
        ACTIVAT
        E
        TURN TO THE LEFT
    }
END
```

モジュール6
アジア

あなたの助けが必要です。とてもしなやかで頑丈な竹は、宇宙船を修理するための完璧な足場になるでしょう。どうか、私のために竹を集めてください。

モジュール導入 1/2:
条件文 - REPEAT WHILE

REPEATループと同様に、**REPEAT WHILE**ループで、スコッティ（または他のキャラクター）にコマンドをループ内で何度も繰り返させることができ、特定の条件が合うまで、繰り返されます。このようにして、**条件付きループ**を作成します。

条件付きループの実例:

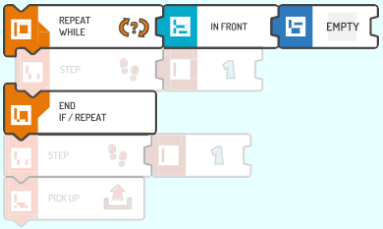
学校から家まで、生徒の親が車で連れて帰るところをイメージしてみましょう。帰宅するまで、どの手順を実行するか、それを何度実行するか、はっきりと決めるのは難しいことがあります。たいていの場合、無意識に**REPEAT WHILE**ループを使います。なぜでしょうか。

1. **「前方に真っ直ぐな道があるか」**という条件を確認します。
2. 条件が合う場合、真っ直ぐ進み、再びポイント（1）へ戻ります。つまり、前方の道が真っ直ぐであるか再確認し始めます。
3. 例えば、前方に曲がり角がある等、条件が合わない場合、ループは壊れています。ポイント（1）からの条件の確認をやめて、ループ外の次のコマンドを実行し始めます。つまり、この場合、正しい方向へ曲がります。

スコッティ・ゴー!の条件付きループ

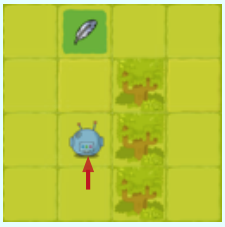
この例では、条件付きループの使い方を見てみましょう。このクエストでは、スコッティは、**STEP**タイルと数字の**1**のタイルの組み合わせだけを使うことができます。こうすると、このループの使い方を説明するのが簡単になります。

1. **「前方に真っ直ぐな道があるか」**という条件を確認します。この条件は合っています。つまり、スコッティの前のスクエアは空です。

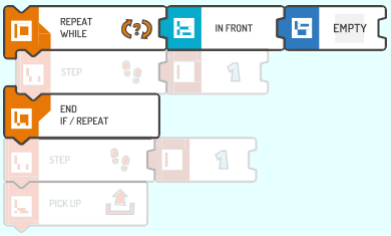


モジュール導入 2/2:
条件文 - REPEAT WHILE

2. スコッティは、ループ外にあるコマンドを実行し始めます。**一歩前に進みます。**
   条件を再確認するためにプログラムは戻ります（ポイント1）。

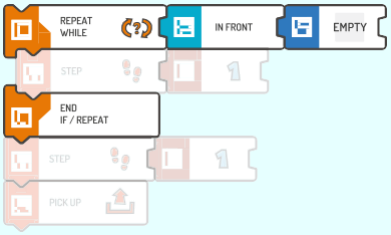



3. スコッティは、「この先通行自由」という条件を再確認します。**条件が合っています。**

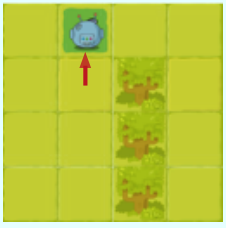



4. スコッティは、ループ内のコマンド、つまり**STEP 1**を再度実行し、その後、ポイント1の条件を確認するために戻ります。


5. スコッティは、「この先通行自由」という条件を再確認します。**今回、条件は合っていません。**

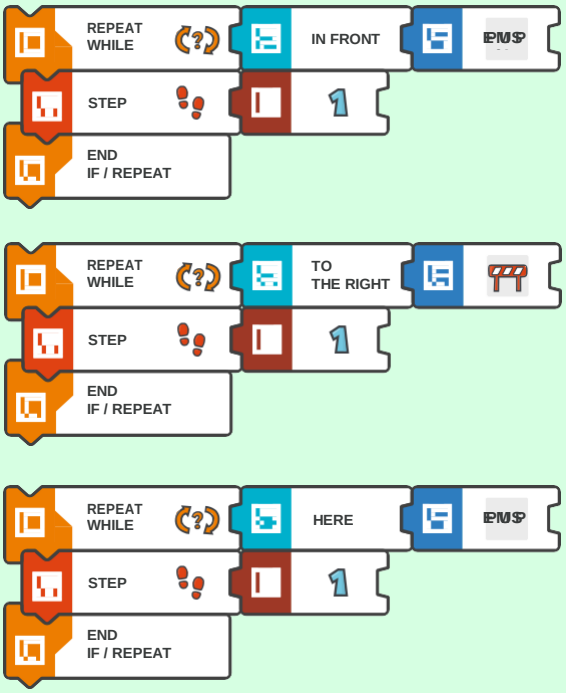
6. **このループは壊れています。**スコッティは、ループ外のプログラムに従い始めます:
   STEP (1)と**PICK UP**。




導入:
REPEAT WHILEタイルを使う

今後のクエストでは、**STEP (1)**タイルと**REPEAT WHILE**ループの組み合わせのみ使います。

REPEAT WHILE (IN FRONT < EMPTY)タイルの組み合わせの代わりに、**REPEAT WHILE(ON THE RIGHT < OBSTACLE)**または**REPEAT WHILE (HERE < EMPTY)**が使えることもあります。



最後の例では、プログラムを1行短くできる可能性があります。

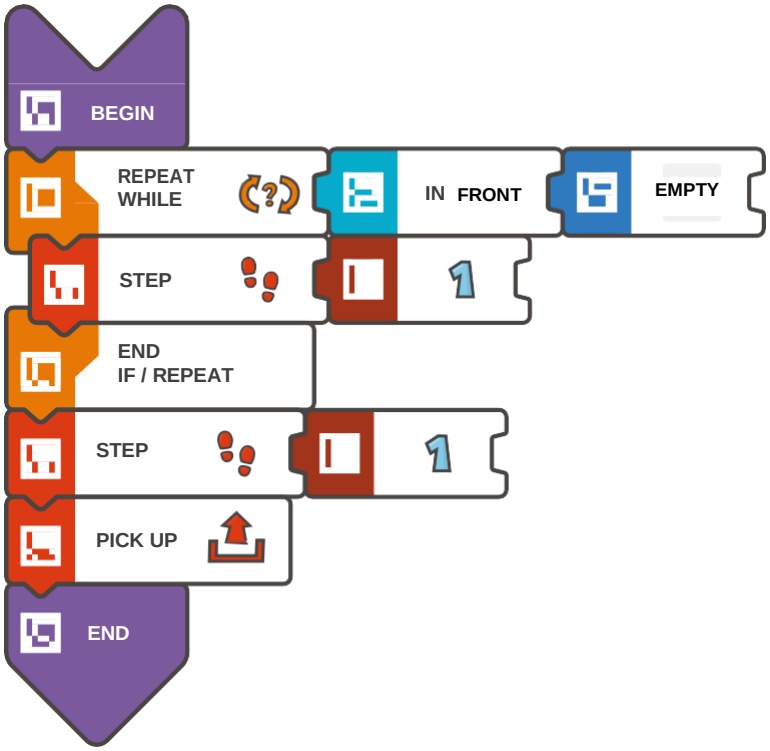
REPEAT WHILEループを導入する際、モジュール導入に記載した「実例」か、または自分で考えた例を使いましょう。

今後の多くのクエストで**STEP (1)**タイルと**REPEAT WHILE**タイルの組み合わせを使う際に一貫性があることは、生徒にループ構造をなじませるのに役立ちます。生徒には**STEP (1)**タイルだけを使わせましょう。

挑戦してみよう:


0/1

解答例:


BEGIN
REPEAT WHILE
IN FRONT
EMPTY
STEP
1
END IF / REPEAT
STEP
1
PICK UP
END

挑戦してみよう:

解答例（テキスト形式）：

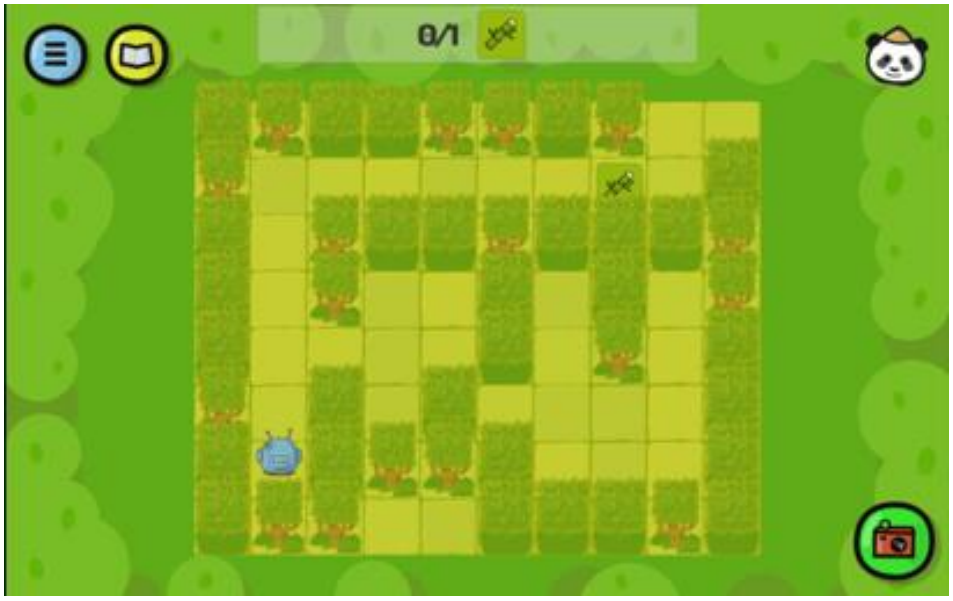
```
BEGIN
    REPEAT WHILE (IN FRONT <
        EMPTY) { STEP (1)
    }
    STEP (1)
    PICK UP
EN
D
```

ヒント

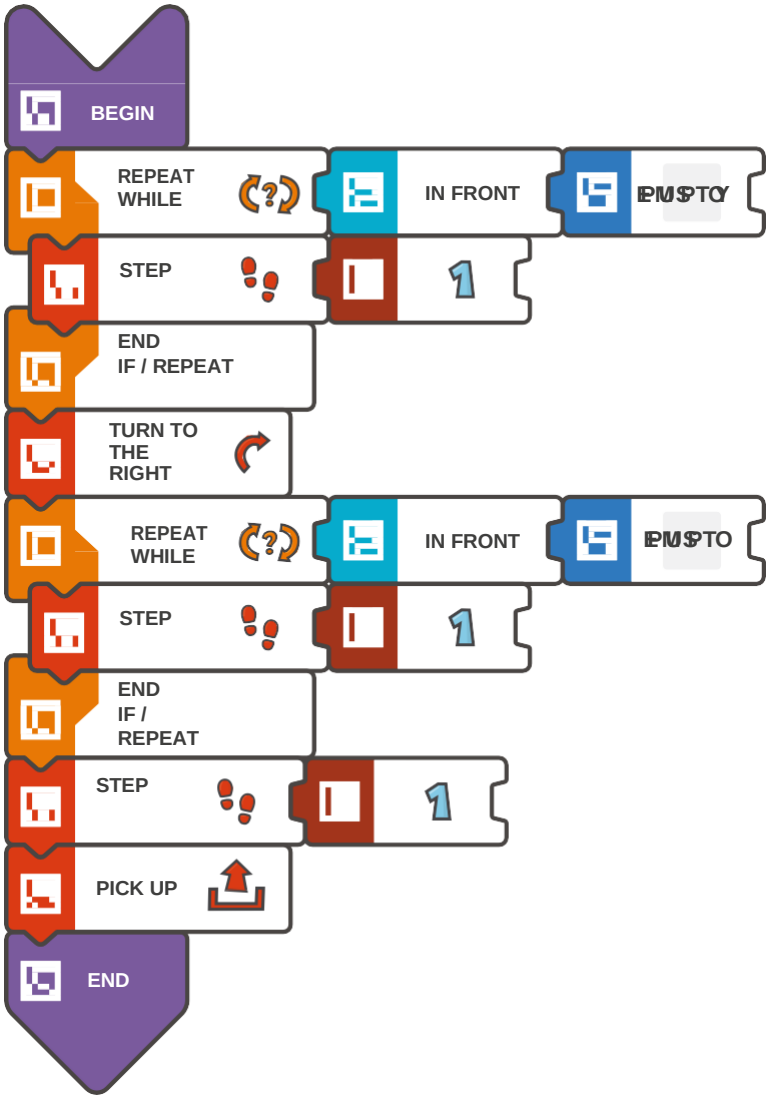
このクエストでは、2つの**REPEAT WHILE**ループを使います。

任意で、このクエストの2番目のループでは、**REPEAT WHILE(IN FRONT < EMPTY)**の組み合わせを使う代わりに、生徒は**REPEAT WHILE (HERE < EMPTY)**の組み合わせを使うことができます。いくらか変更すると、プログラムは1行短くなります。

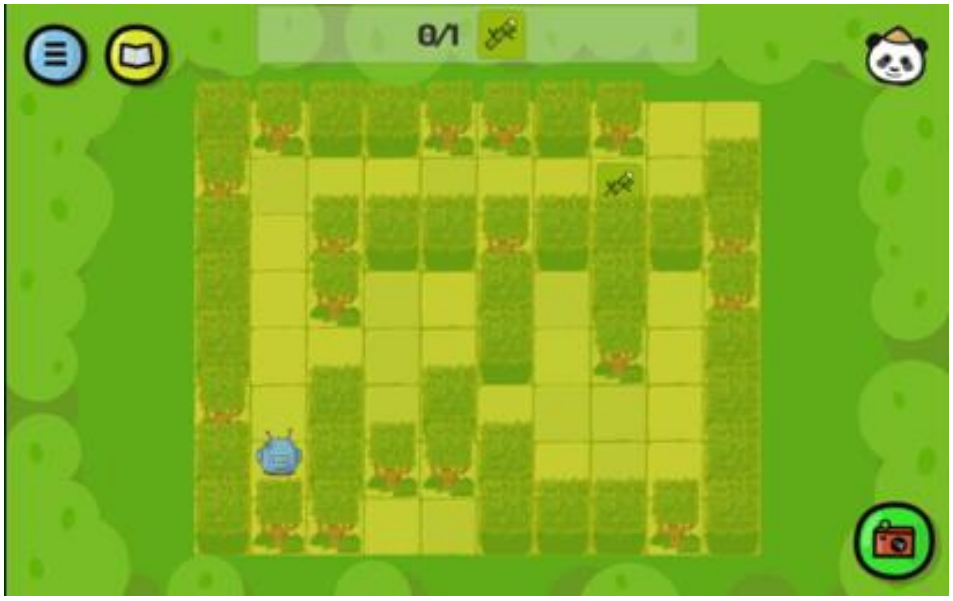
挑戦してみよう:


0/1

解答例:


BEGIN
REPEAT WHILE
IN FRONT
STEP
1
END IF / REPEAT
TURN TO THE RIGHT
REPEAT WHILE
IN FRONT
STEP
1
END IF / REPEAT
STEP
1
PICK UP
END

挑戦してみよう:



解答例（テキスト形式）：

```
BEGIN
    REPEAT WHILE (IN FRONT <
        EMPTY) { STEP (1)
    }
    TURN TO THE RIGHT
    REPEAT WHILE (IN FRONT <
        EMPTY) { STEP (1)
    }
    STEP (1)
    PICK UP
EN
D
```

ヒント

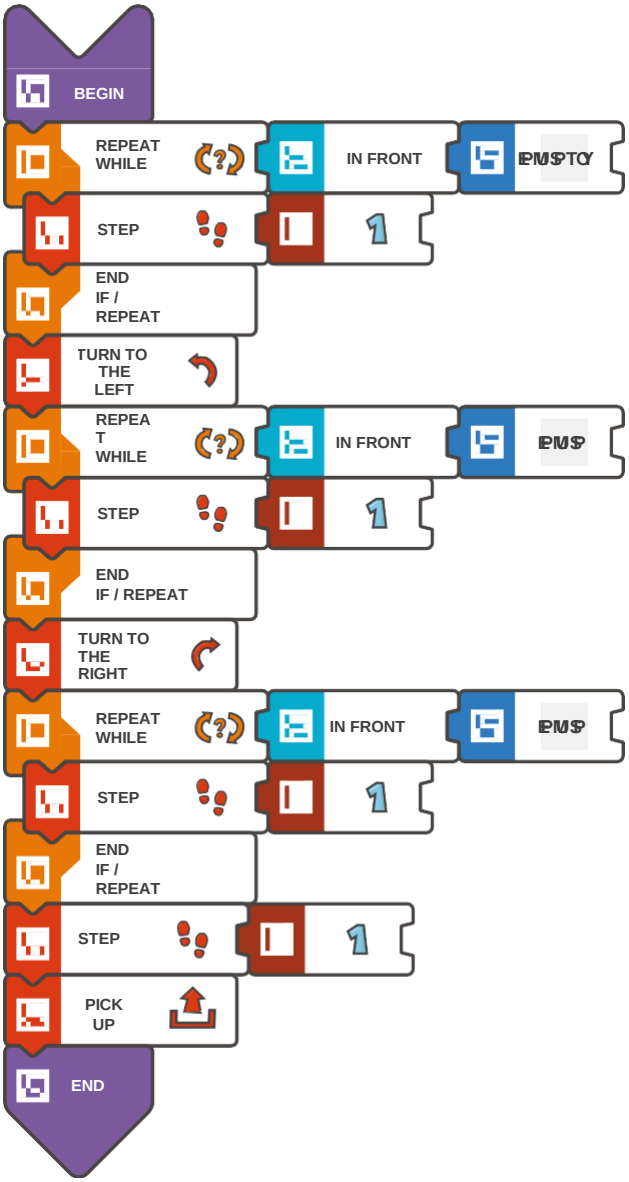
前のクエストで分かるように、**REPEAT WHILE**ループは**REPEAT**ループのさらに高度なバージョンであり、条件（条件文）が合う限り、ループ内の一連のコマンドが実行されます。条件が合わなくなった場合は、ループが壊れています。

このクエストでは、3つの**REPEAT WHILE**ループを使います。

挑戦してみよう:


0/1

解答例:


BEGIN
REPEAT WHILE
IN FRONT
STEP
1
END IF / REPEAT
TURN TO THE LEFT
REPEAT WHILE
IN FRONT
STEP
1
END IF / REPEAT
TURN TO THE RIGHT
REPEAT WHILE
IN FRONT
STEP
1
END IF / REPEAT
STEP
1
PICK UP
END

挑戦してみよう:

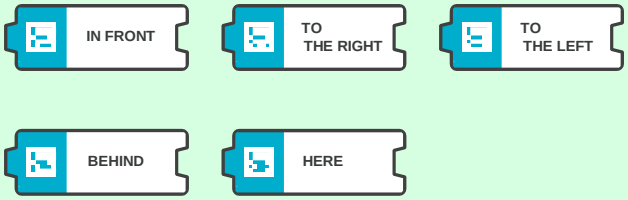
解答例（テキスト形式）：

```
BEGIN
    REPEAT WHILE (IN FRONT <
        EMPTY) { STEP (1)
    }
    TURN TO THE LEFT
    REPEAT WHILE (IN FRONT <
        EMPTY) { STEP (1)
    }
    TURN TO THE RIGHT
    REPEAT WHILE (IN FRONT <
        EMPTY) { STEP (1)
    }
    STEP (1)
    PICK UP
EN
D
```

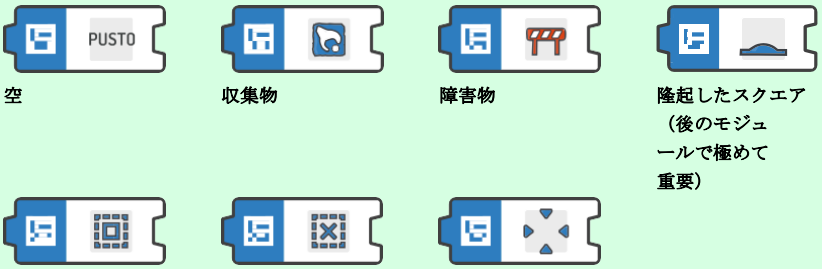
導入:
他の条件付きのREPEAT WHILE

ループ内のコマンドが実行されるように、多くの条件を付けて**REPEAT WHILE**ループを作成できます。

（スコッティの視点を考慮して）どこ?



何?



空

収集物

障害物

隆起したスクエア（後のモジュールで極めて重要）

スクエアのマーク（後のモジュールで極めて重要）

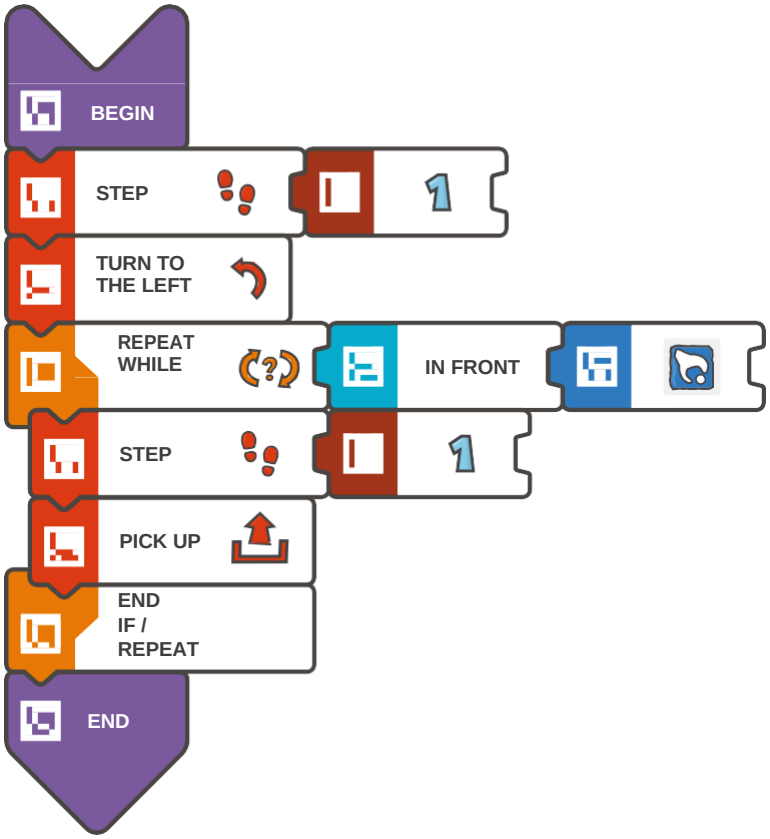
X印（後のモジュールで極めて重要）

アクションスクエア（後のモジュールで極めて重要）

挑戦してみよう:


0/5

解答例:


BEGIN
STEP
1
TURN TO
THE LEFT
REPEAT
WHILE
IN FRONT
STEP
1
PICK UP
END
IF /
REPEAT
END

挑戦してみよう:

解答例（テキスト形式）：

```
BEGIN
    STEP (1)
    TURN TO THE LEFT
    REPEAT WHILE (IN FRONT < COLLECTABLE
        OBJECT) { STEP (1)
        PICK UP
    }
END
```

導入:
入れ子ループ内のREPEAT WHILE

必要であれば、**REPEAT WHILE**ループは何度も使えます。**REPEAT**ループを作成し、それから別の**REPEAT WHILE**ループをその中に置きます。別のループ内のループは、それほど難しくありません。試してみましょう!

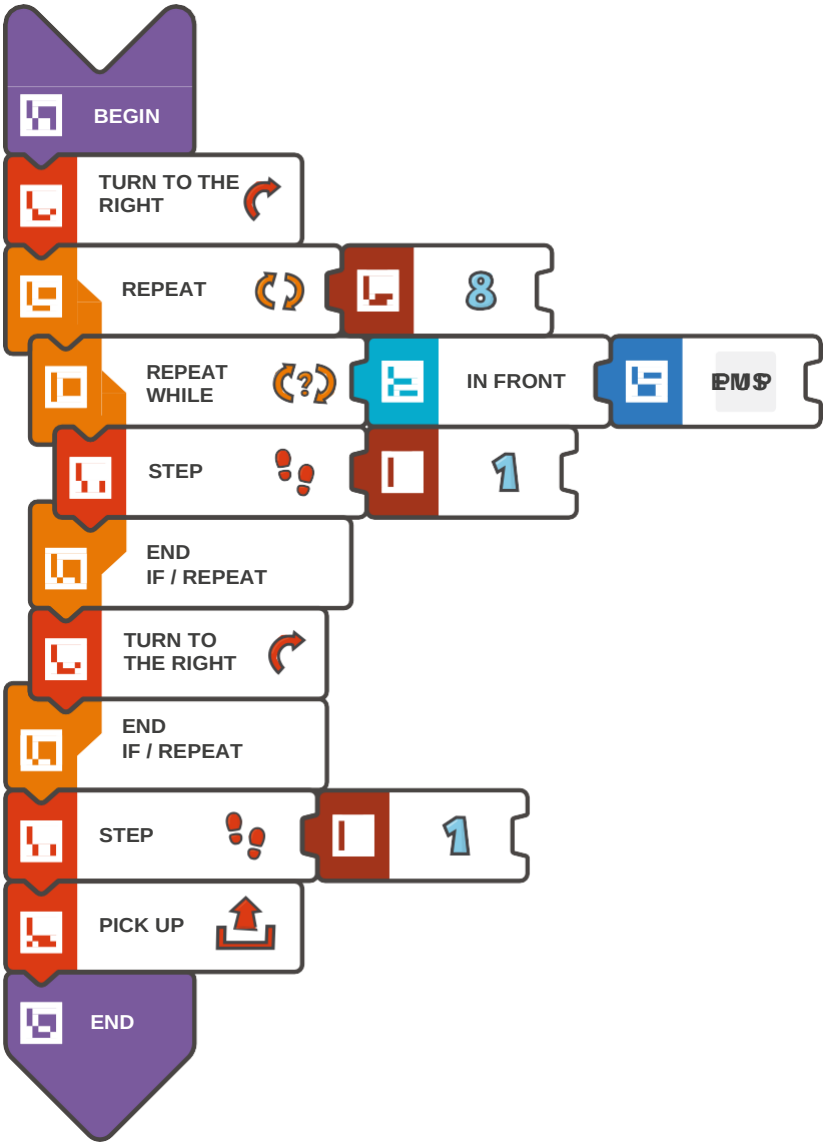
もっと上級の生徒は、以下のような条件文を使って、このクエストを解こうとするかもしれません:

```
IF (ON THE RIGHT <
      EMPTY){ TURN TO
      THE RIGHT
}
```

挑戦してみよう:


0/1

解答例:


BEGIN
TURN TO THE RIGHT
REPEAT
8
REPEAT WHILE
IN FRONT
STEP
1
END IF / REPEAT
TURN TO THE RIGHT
END IF / REPEAT
STEP
1
PICK UP
END

挑戦してみよう:

解答例（テキスト形式）：

```
BEGIN
    TURN TO THE
    RIGHT REPEAT (8)
    {
        REPEAT WHILE (IN FRONT < EMPTY)
            { STEP (1)
        }
        TURN TO THE RIGHT
    }
    STEP (1)
    PICK UP
EN
D
```

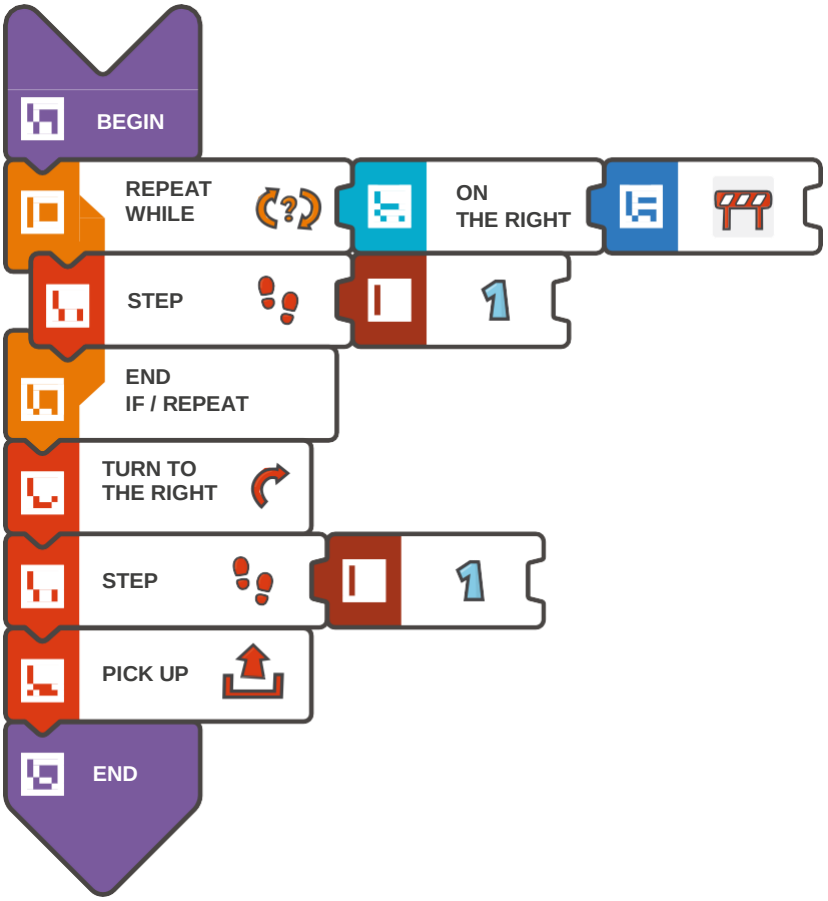
注意!

REPEAT WHILEループは一度だけ使います。

挑戦してみよう:


0/1

解答例:


BEGIN
REPEAT WHILE
ON THE RIGHT
STEP
1
END IF / REPEAT
TURN TO THE RIGHT
STEP
1
PICK UP
END

挑戦してみよう:

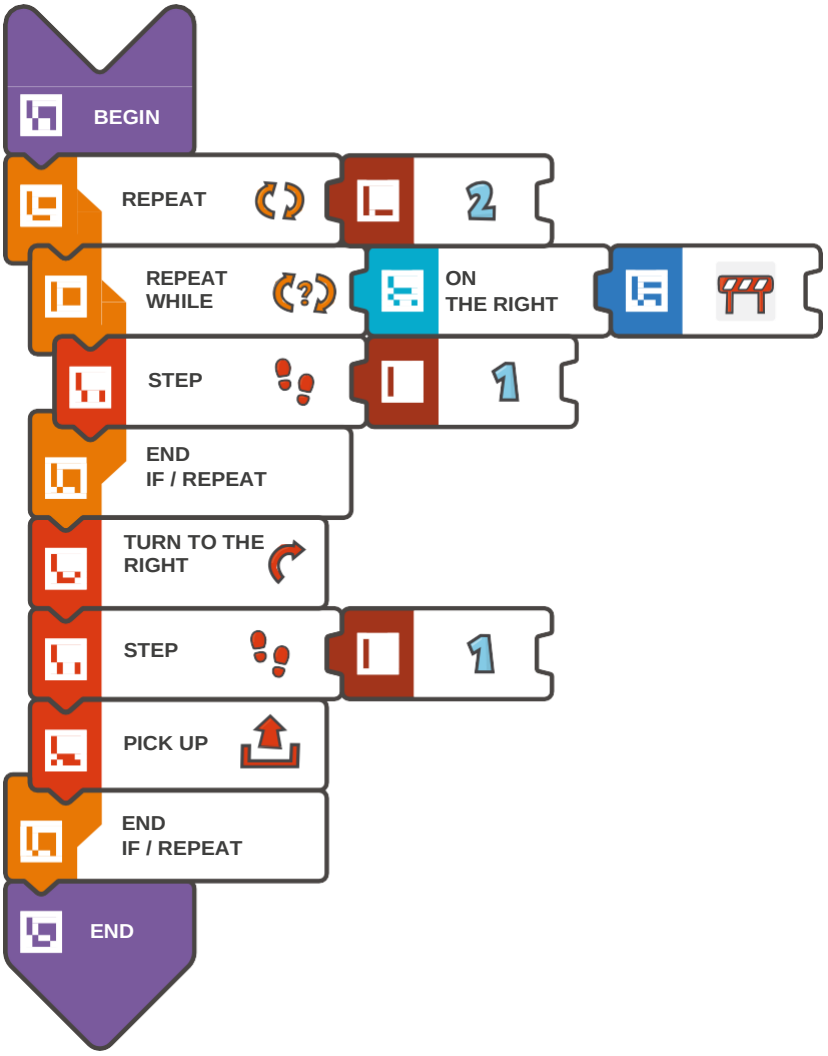
解答例（テキスト形式）：

```
BEGIN
    REPEAT WHILE (ON THE RIGHT <
        OBSTACLE) { STEP (1)
    }
    TURN TO THE
    RIGHT STEP (1)
    PICK UP
EN
D
```

挑戦してみよう:


0/2

解答例:


BEGIN
REPEAT
2
REPEAT WHILE
ON THE RIGHT
STEP
1
END IF / REPEAT
TURN TO THE RIGHT
STEP
1
PICK UP
END IF / REPEAT
END

挑戦してみよう:

解答例（テキスト形式）：

```
BEGIN
    REPEAT (2) {
        REPEAT WHILE (ON THE RIGHT <
            OBSTACLE) { STEP (1)
        }
        TURN TO THE
        RIGHT STEP (1)
        PICK UP
    }
END
```

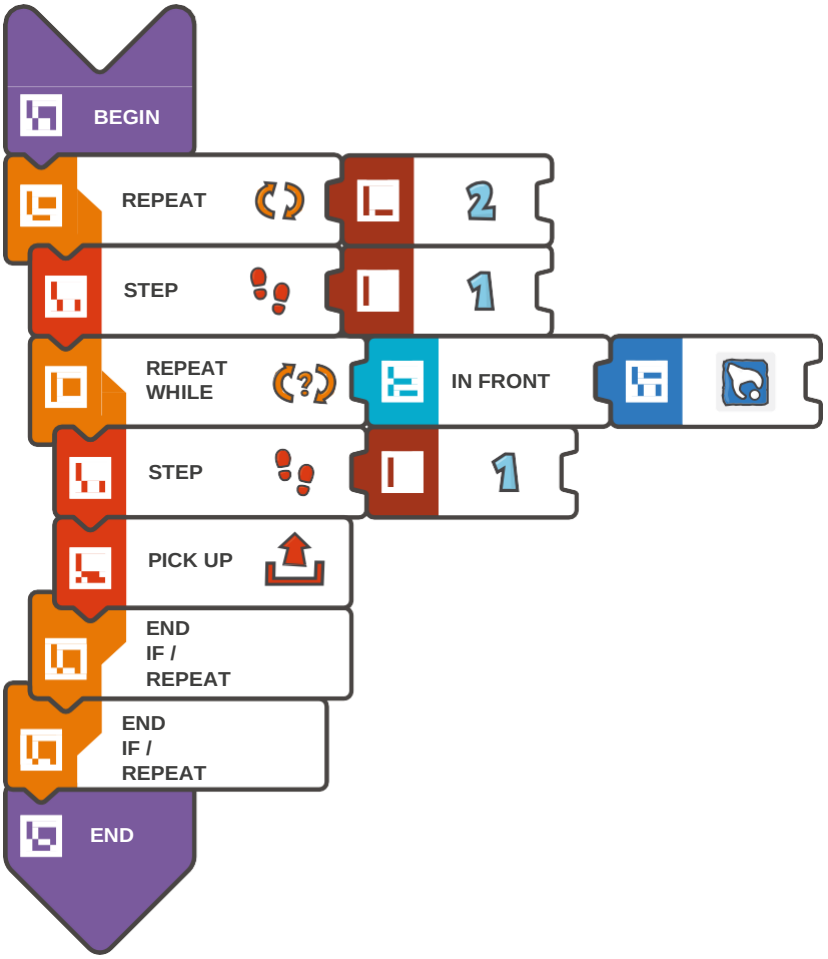
注意!

このクエストでは、**REPEAT WHILE**ループと**REPEAT**ループは、どちらも一度だけ使います。

挑戦してみよう:


0/5

解答例:


BEGIN
REPEAT
2
STEP
1
REPEAT
WHILE
IN FRONT
STEP
1
PICK UP
END
IF /
REPEAT
END
IF /
REPEAT
END

挑戦してみよう:

解答例（テキスト形式）：

```
BEGIN
    REPEAT (2) {
        STEP (1)
        REPEAT WHILE (IN FRONT < COLLECTABLE
            OBJECT) { STEP (1)
            PICK UP
        }
    }
END
```

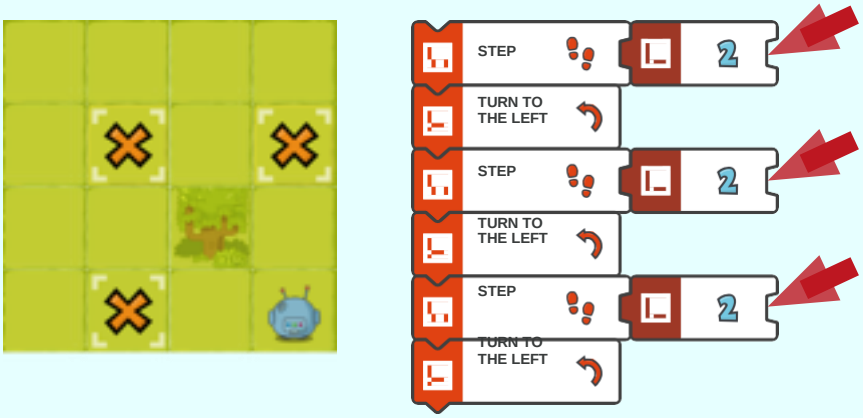
モジュール7
オーストラリア1

このモジュールでは、変数の概念が導入されます。スコッティが村を動き回り、宇宙船の部品を集めることができるように、アスファルトの道を作りましょう。

モジュール導入 1/3:
新しい概念: 変数

変数は容器であると考えることができます。例えば、数のような様々なデータを容器に入れ、後で使うために保存しておきます。コンピュータープログラムは、実行中、現在の内容を調べたり、内容を変更したりするために、このような容器を探します。プログラムが探す場所を知るには（1つのプログラムには複数の変数があると考えると簡単です）、変数に名前が付いていなくてはいけません。

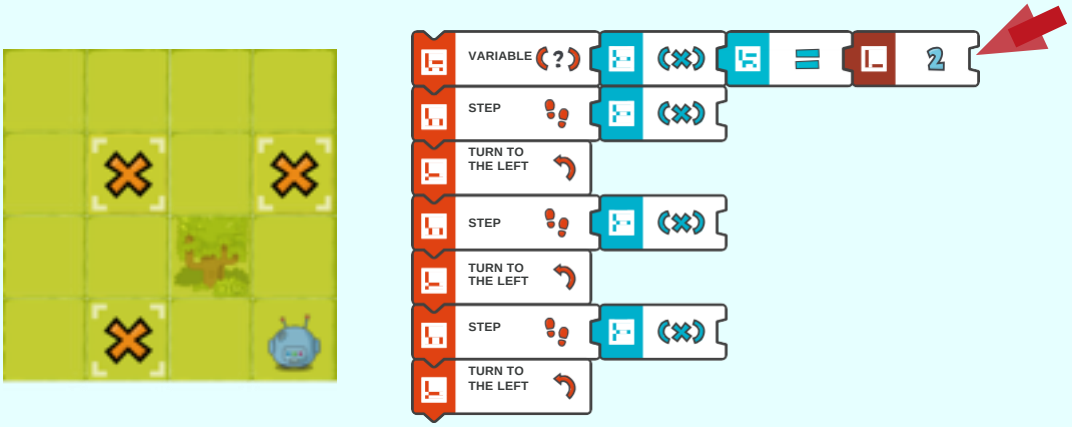
例えば、スコッティの歩数を決めるために、変数を使うことができます。下に示すクエストで、スコッティは2つの指示、2歩前進と**TURN TO THE LEFT**を繰り返し行うと仮定しましょう。

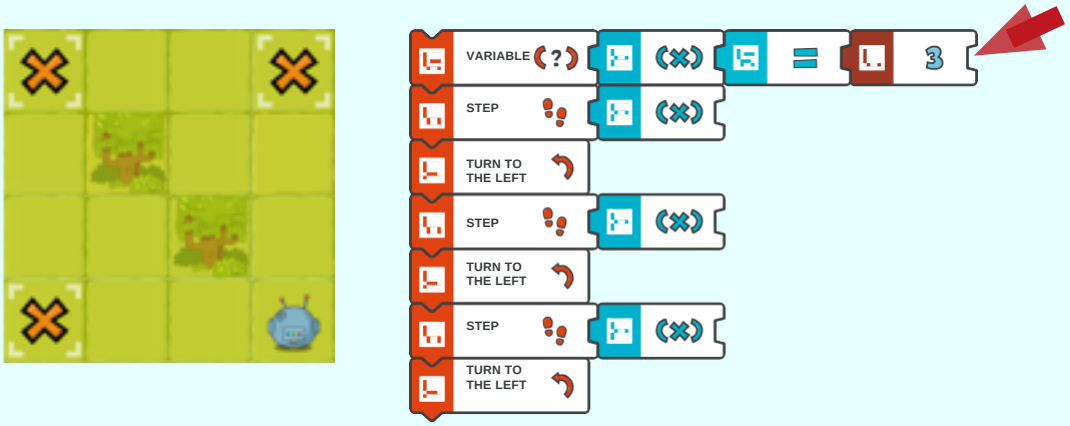


次のクエスト（下の画像参照）では、スコッティの動きはほとんど同じですが、歩数だけ変更しました（3歩前進と**TURN TO THE LEFT**）。このクエストで機能させるには、先に示したプログラムを3回変更する必要があることが明らかです。

モジュール導入 2/3:
新しい概念: 変数

例として示した（下の画像参照）最初のクエストに戻りましょう。ただし、今回は変数を使って、シンプルな解答にしてみましょう。各手順で、数の代わりに変数を使いましょう。プログラムの最初の部分では、等号「＝」を使って、この変数を定義し、そして値（歩数）を2と定義しましょう。





変数が使われた次のクエストを解くには、変数**X**の値が定義されているコードの最初の行の変更だけで十分です。

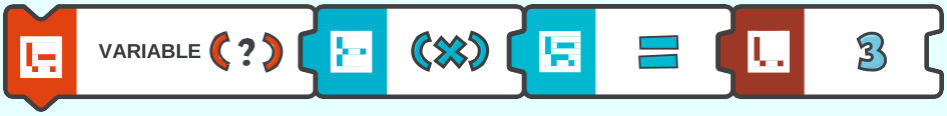
お分かりのように、変数によって、プログラム変更に要する時間を短縮できることがよくあります。

要約: 変数のおかげで、もっと整ったプログラムになり、変更がもっと容易になります。

モジュール導入 3/3:
新しい概念: 変数

変数がどのように機能するのか確かめるには、以下の手順を実行します:

1. 変数に名前を付けて、元々持っていた値を決めて、変数を作成します。

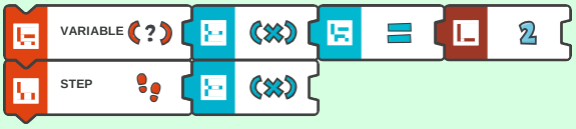


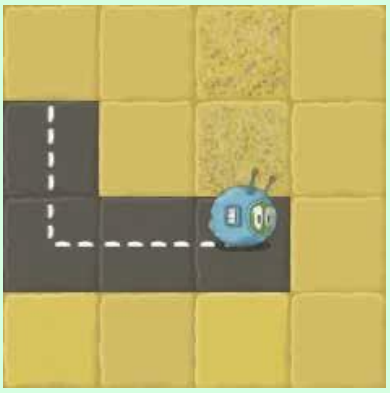
2. プログラムでは、値の代わりに名前で呼び出して、この変数を使いましょう。

導入 - アスファルトの道を作る

スコッティは、砂利を歩きながら、アスファルトの道を作ります。

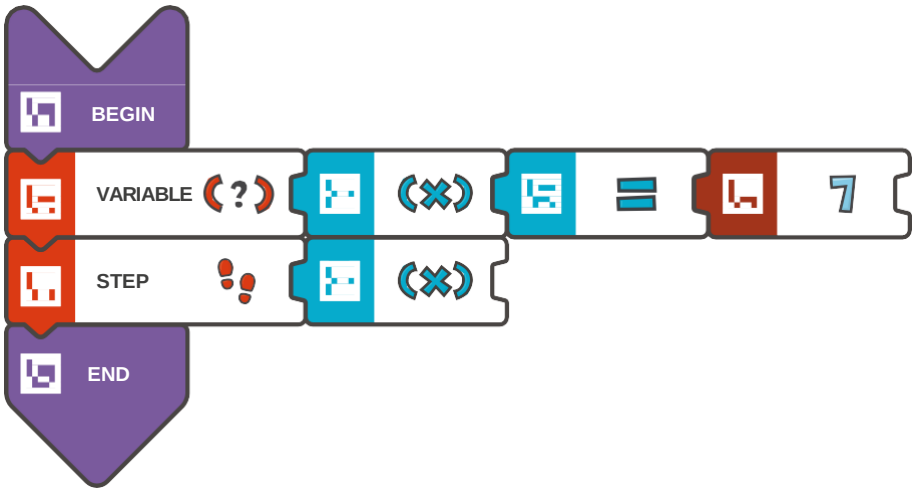
変数**X =** 歩数を作成し、その後、プログラムで呼び出すことを覚えておきましょう。





挑戦してみよう:

解答例:

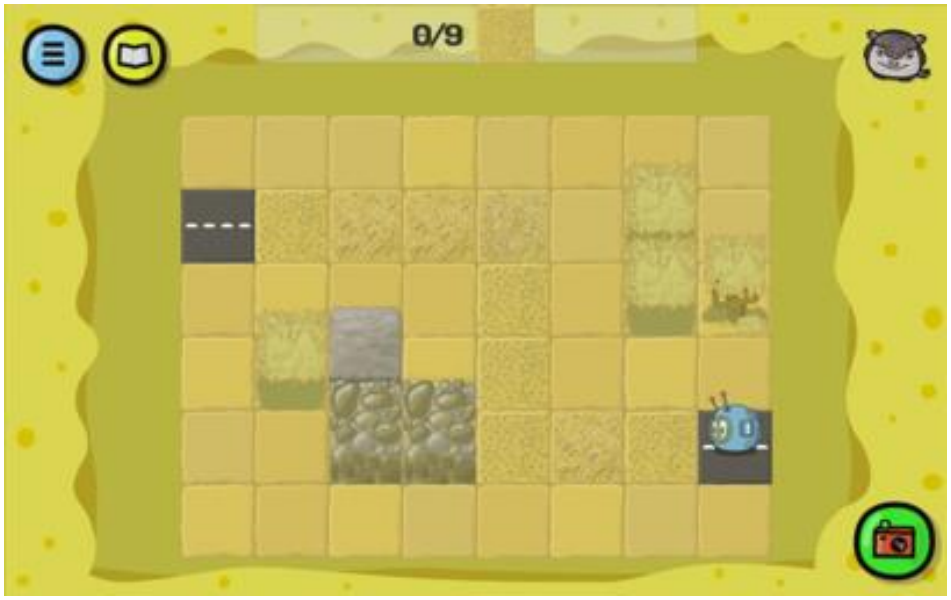


挑戦してみよう:

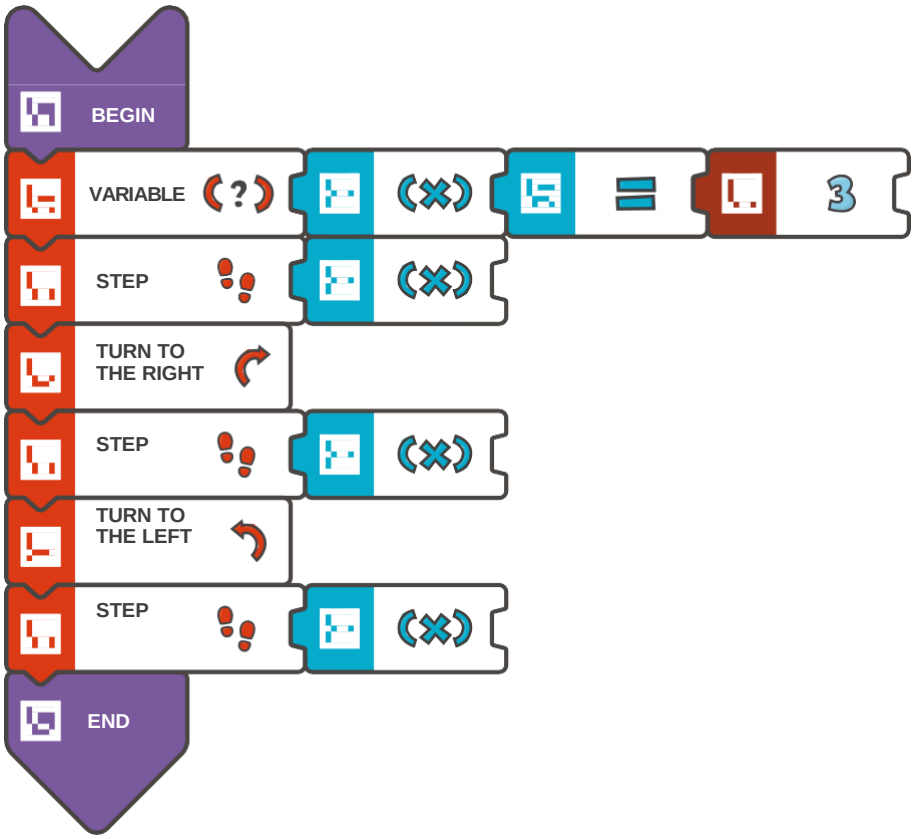
解答例（テキスト形式）：

```
BEGIN
    VARIABLE X=7
    STEP (X)
END
```

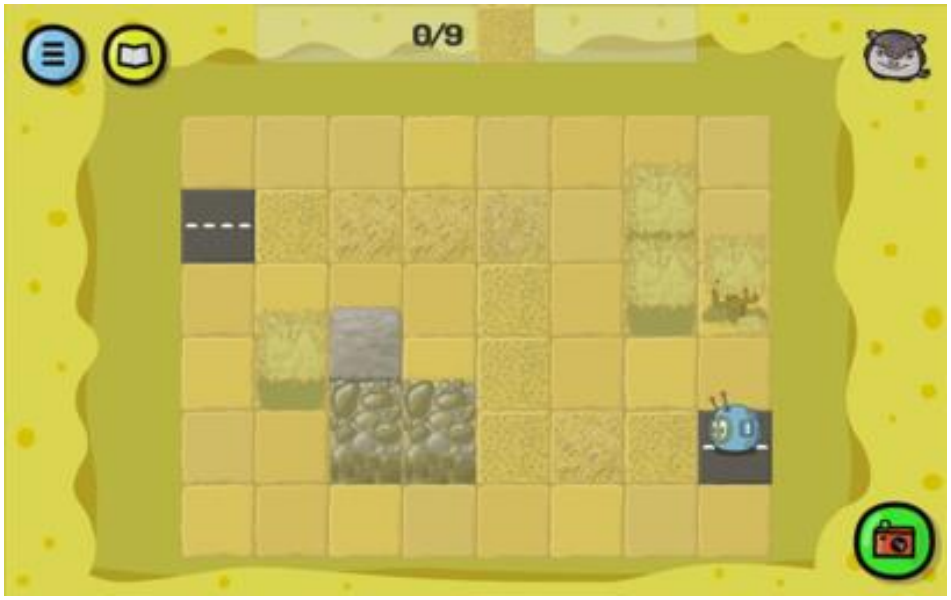
挑戦してみよう:


0/9

解答例:


BEGIN
VARIABLE
STEP
TURN TO THE RIGHT
STEP
TURN TO THE LEFT
STEP
END
3

挑戦してみよう:



解答例（テキスト形式）：

```
BEGIN
    VARIABLE X=3
    STEP (X)
    TURN TO THE
    RIGHT STEP (X)
    TURN TO THE
    LEFT STEP (X)
END
```

注意!

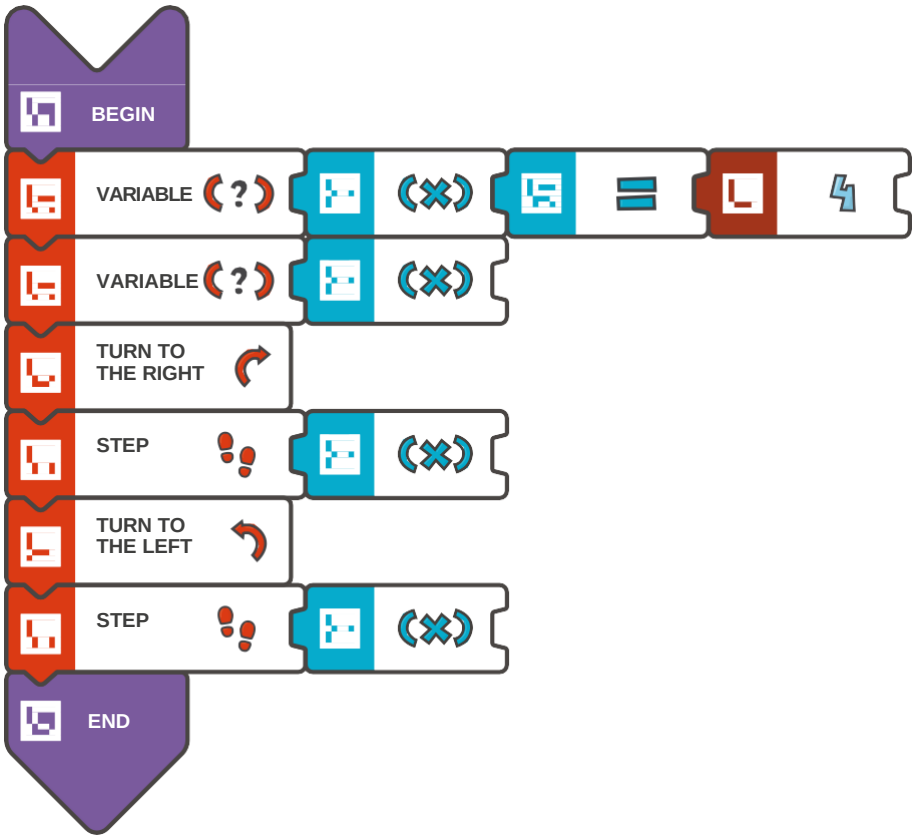
以前のプログラムでこのクエストも解けます。

必要なのは、変数Xの値の変更だけです。

挑戦してみよう:


0/12

解答例:


BEGIN
VARIABLE
VARIABLE
TURN TO THE RIGHT
STEP
TURN TO THE LEFT
STEP
END

挑戦してみよう:

解答例（テキスト形式）：

```
BEGIN
    VARIABLE X=4
    STEP (X)
    TURN TO THE
    RIGHT STEP (X)
    TURN TO THE
    LEFT STEP (X)
END
```

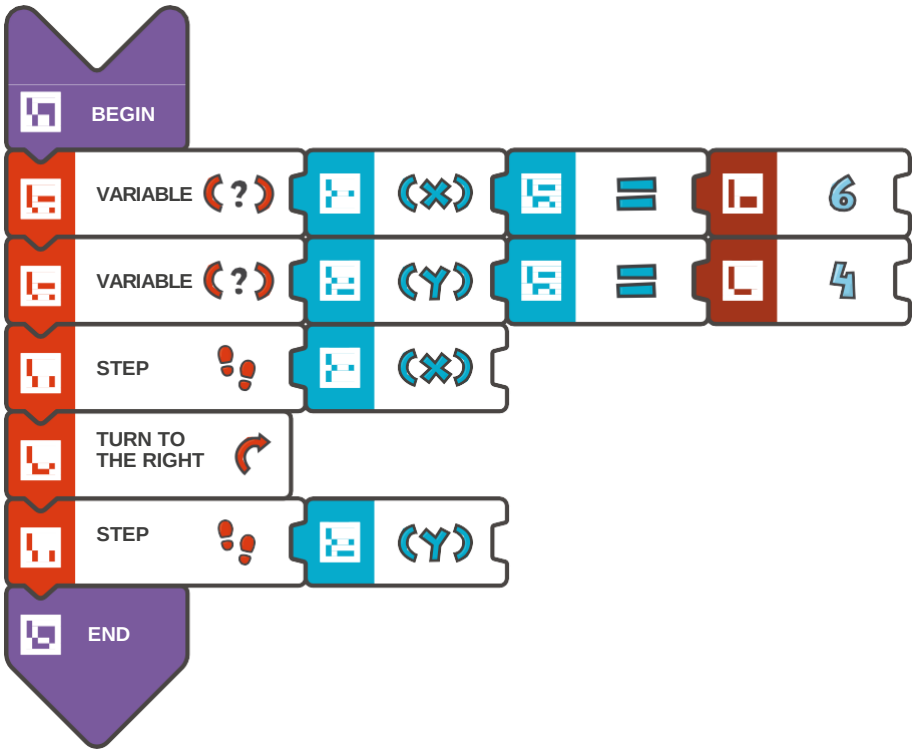
導入:
XとY

スコッティ・ゴー!では、同じプログラムで2つの変数: **X**と**Y**を作成し、使用できます。

挑戦してみよう:


0/10

解答例:


BEGIN
VARIABLE
(x)
=
6
VARIABLE
(y)
=
4
STEP
(x)
TURN TO
THE RIGHT
STEP
(y)
END

挑戦してみよう:

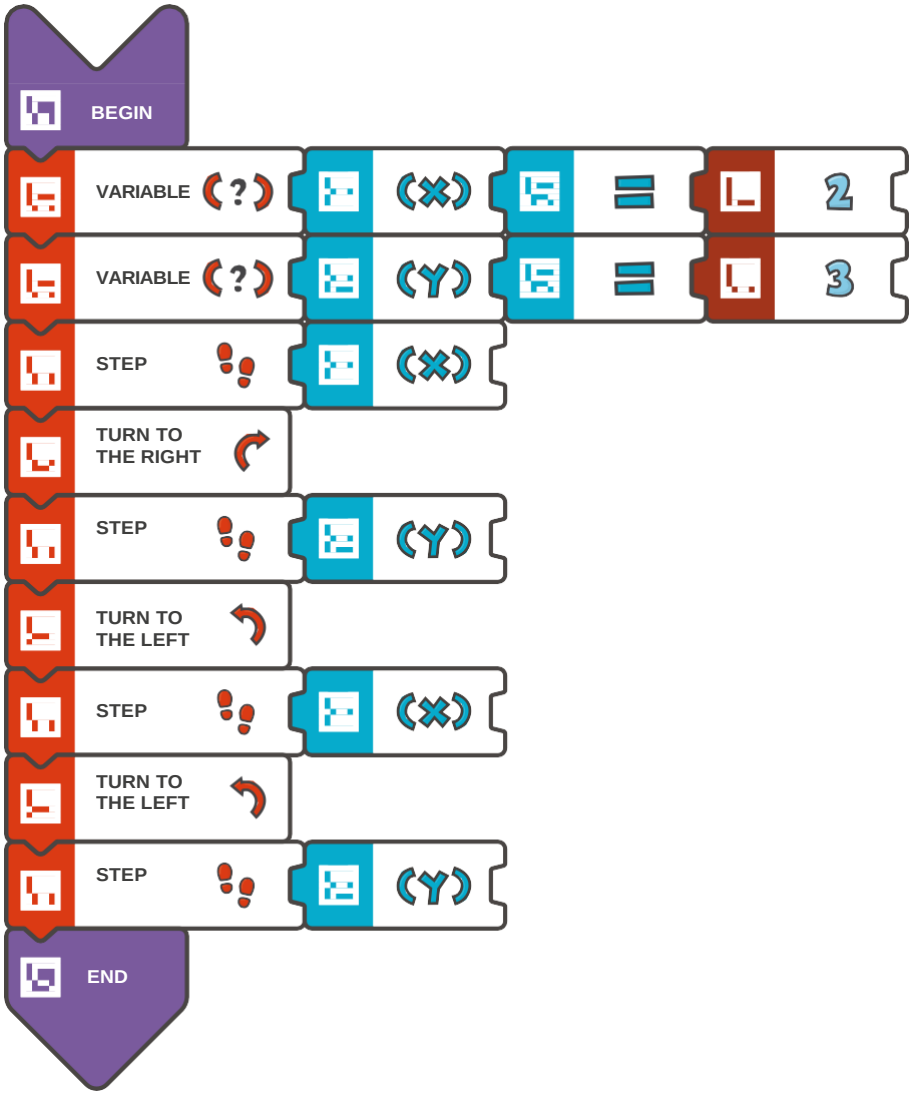
解答例（テキスト形式）：

```
BEGIN
    VARIABLE
    X=6
    VARIABLE
    Y=4 STEP (X)
    TURN TO THE
    RIGHT STEP (Y)
END
```

挑戦してみよう:


0/10

解答例:


BEGIN
VARIABLE
(?)
(x)
=
2
VARIABLE
(?)
(y)
=
3
STEP
(x)
TURN TO
THE RIGHT
STEP
(y)
TURN TO
THE LEFT
STEP
(x)
TURN TO
THE LEFT
STEP
(y)
END

挑戦してみよう:

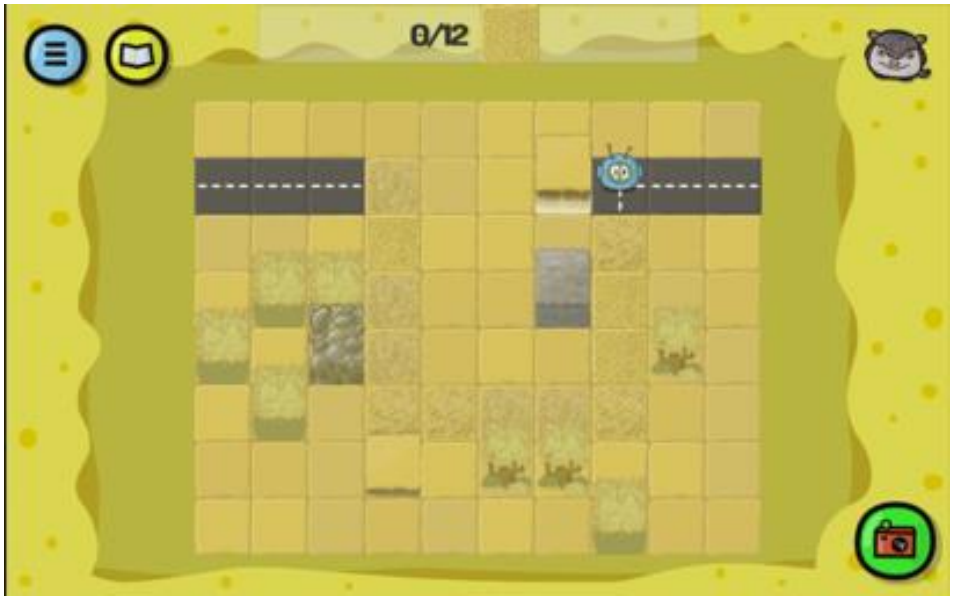
解答例（テキスト形式）：

```
BEGIN
    VARIABLE
    X=2
    VARIABLE
    Y=3 STEP (X)
    TURN TO THE
    RIGHT STEP (Y)
    TURN TO THE
    LEFT STEP (X)
    TURN TO THE
    LEFT STEP (Y)
END
```

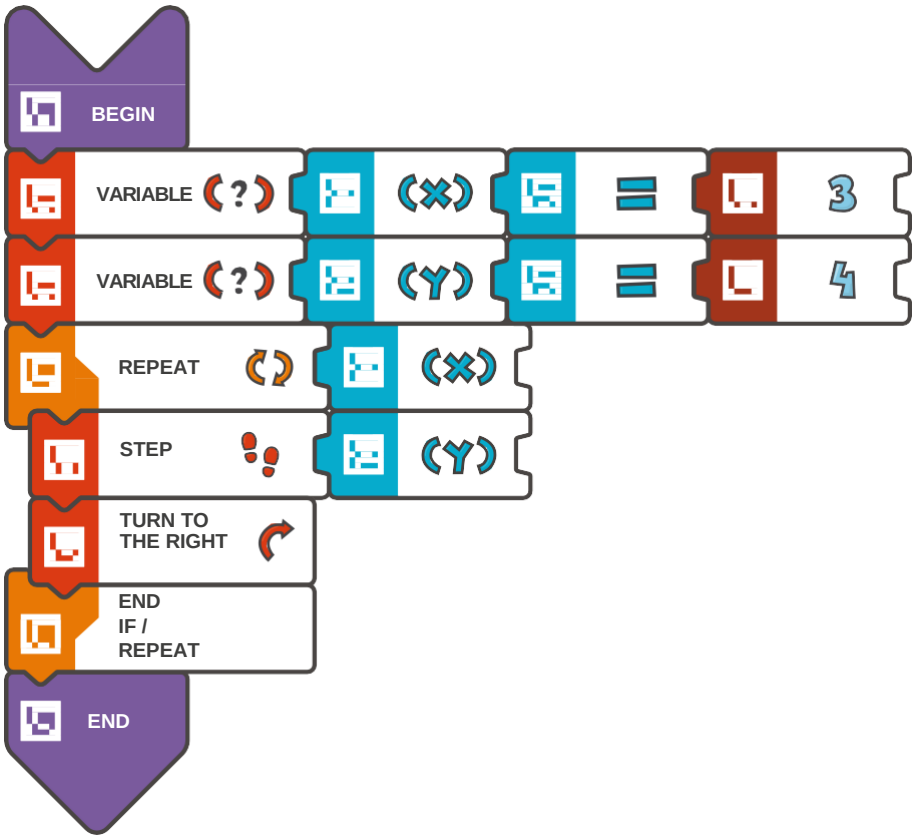
導入:
変数とループ

このクエストでは、2つの変数: **X**と**Y**、**REPEAT**ループを使います。ループを繰り返す回数には変数**X**を、歩数には変数**Y**を使います。

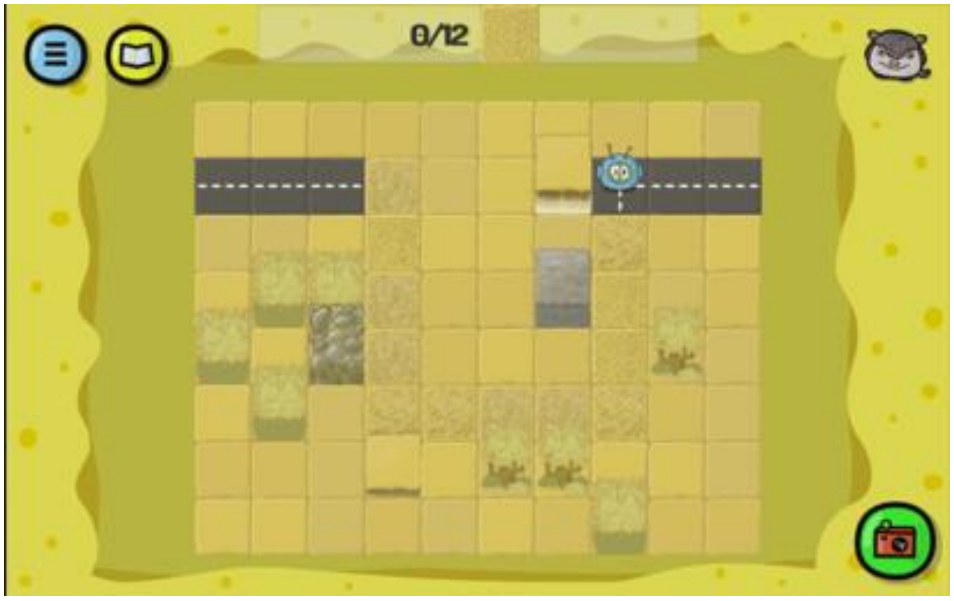
挑戦してみよう:


0/12

解答例:


BEGIN
VARIABLE
(?)
(x)
=
3
VARIABLE
(?)
(y)
=
4
REPEAT
(x)
STEP
(y)
TURN TO
THE RIGHT
END
IF /
REPEAT
END

挑戦してみよう:



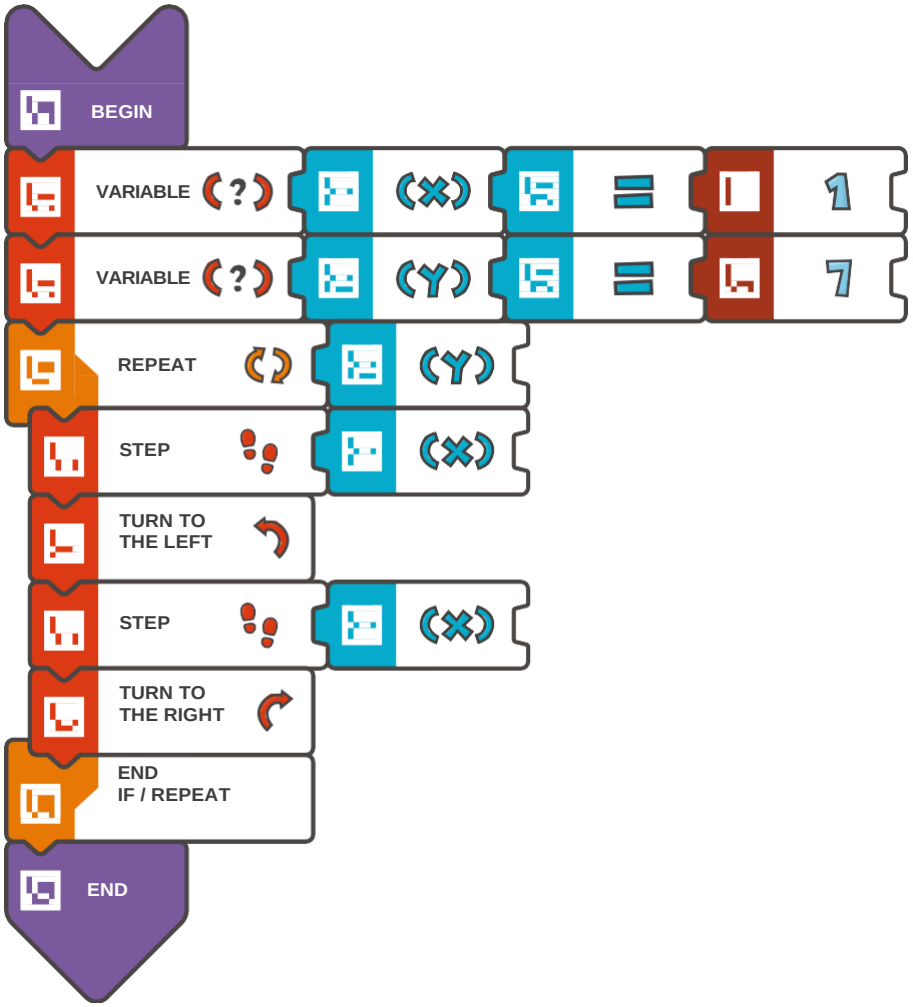
解答例（テキスト形式）：

```
BEGIN
    VARIABLE
    X=3
    VARIABLE
    Y=4   REPEAT
    (X) {
        STEP (Y)
        TURN TO THE RIGHT
    }
END
```

挑戦してみよう

（このクエストでは、プレイヤーは2つの変数を使います。
今回は、その1つがループを繰り返す回数を表します）：

解答例:



挑戦してみよう

（このクエストでは、プレイヤーは2つの変数を使います。

今回は、その1つがループを繰り返す回数を表します）：

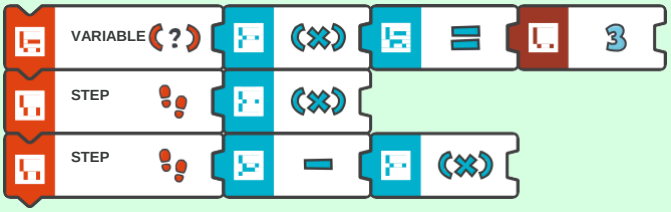
解答例（テキスト形式）：

```
BEGIN
    VARIABLE
    X=1
    VARIABLE
    Y=7   REPEAT
    (Y) {
        STEP (X)
        TURN TO THE LEFT
        STEP (X)
        TURN TO THE RIGHT
    }
END
```

導入:
変数の数学操作

変数の数学操作も行うことができます。

下に示すプログラムの一部では、2度目に呼び出される場合、**MINUS**タイルを変数**X**の前に加えることで、歩数を変数の負の値に変更します。

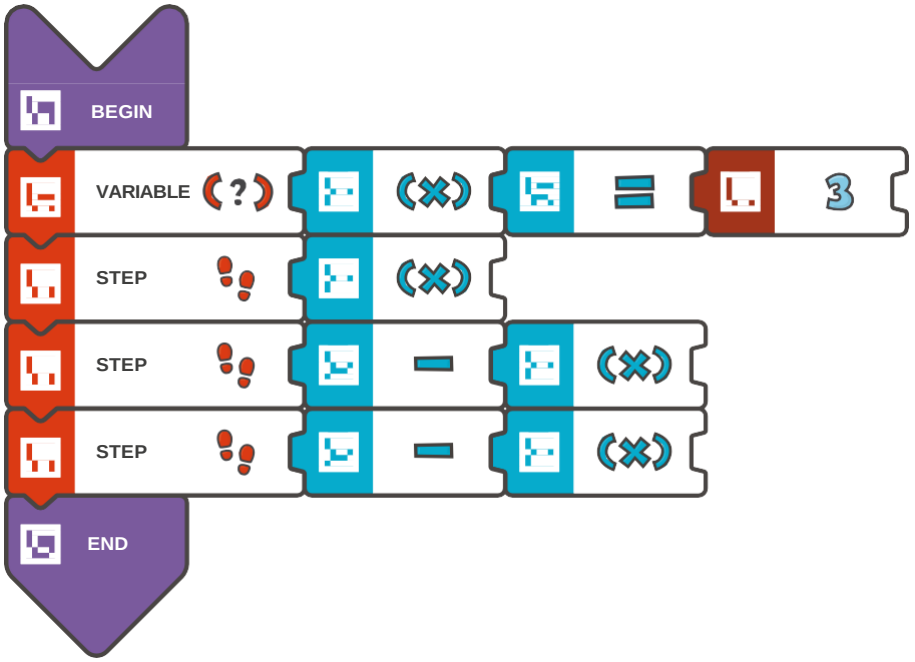


このクエストでは、プレイヤーは数字の3のタイルは1回だけ使えます。

挑戦してみよう

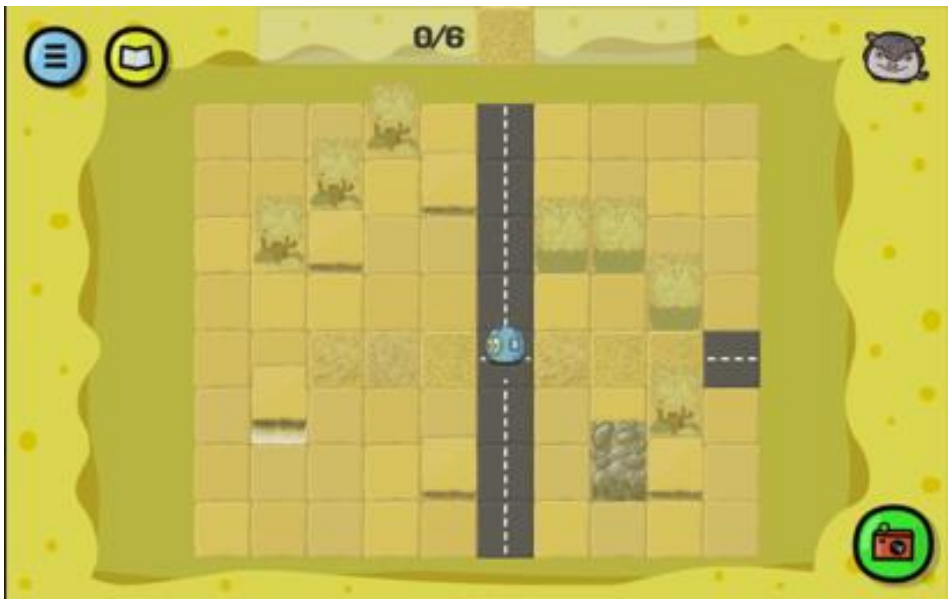
（このクエストでは、プレイヤーは数字の3のタイルは1回だけ使います：

解答例:



挑戦してみよう

（このクエストでは、プレイヤーは数字の3のタイルは1回だけ使います：



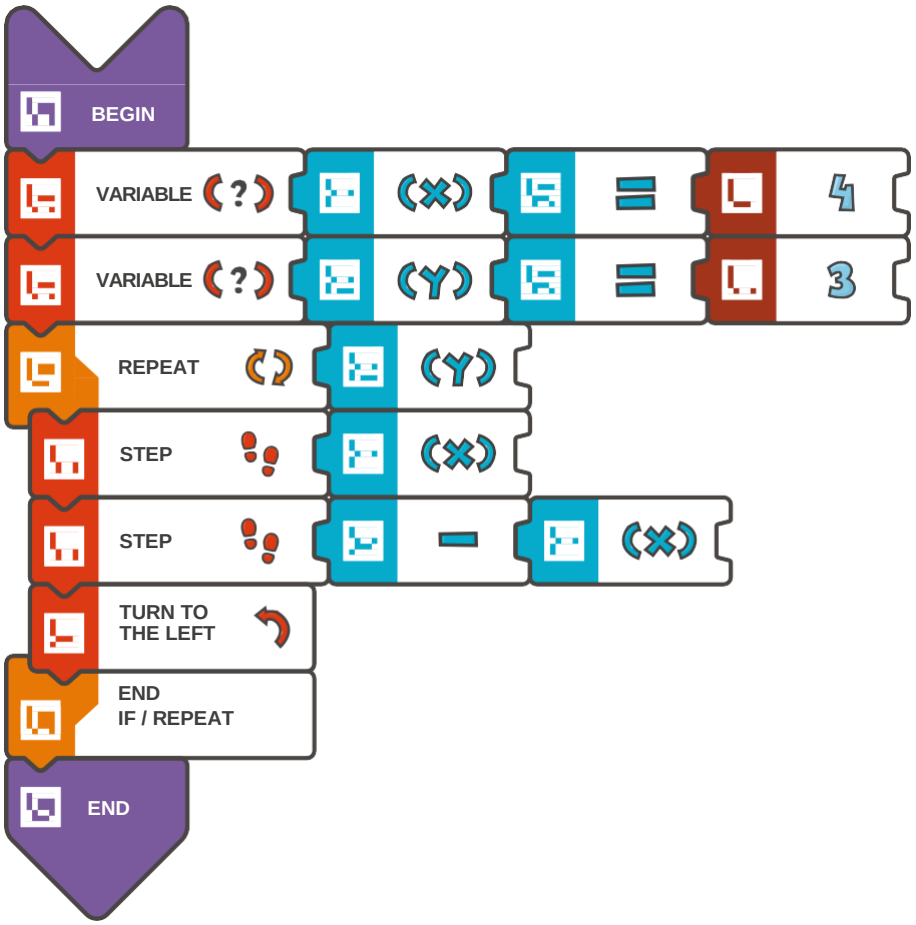
解答例（テキスト形式）：

```
BEGIN
    VARIABLE
    X=3 STEP (X)
    STEP (-X)
    STEP (-X)
END
```

挑戦してみよう

（このクエストでは、プレイヤーは2つの変数を使います: 1つは歩数を、もう1つはループを繰り返す回数を表します）：

解答例:



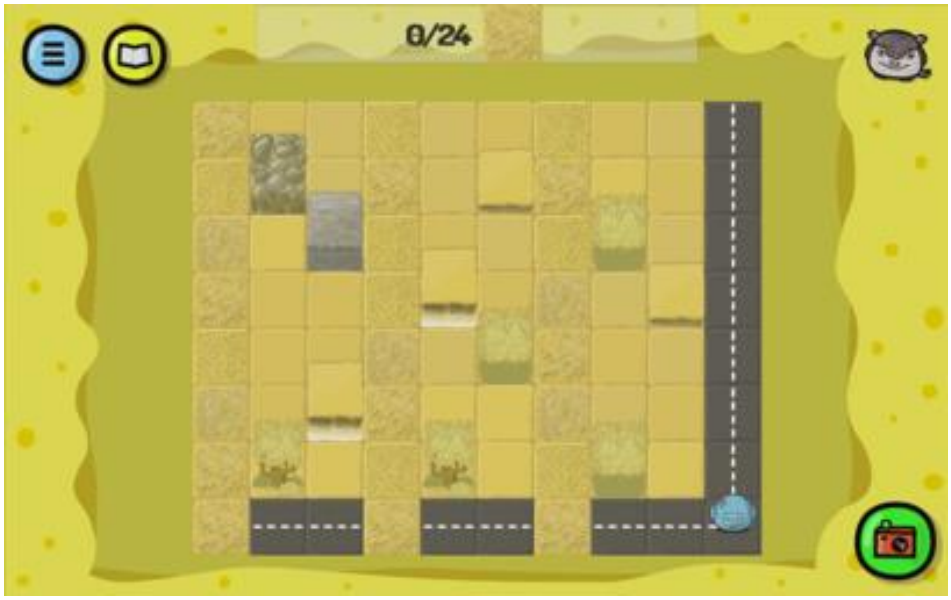
挑戦してみよう

（このクエストでは、プレイヤーは2つの変数を使います: 1つは歩数を、もう1つはループを繰り返す回数を表します）：

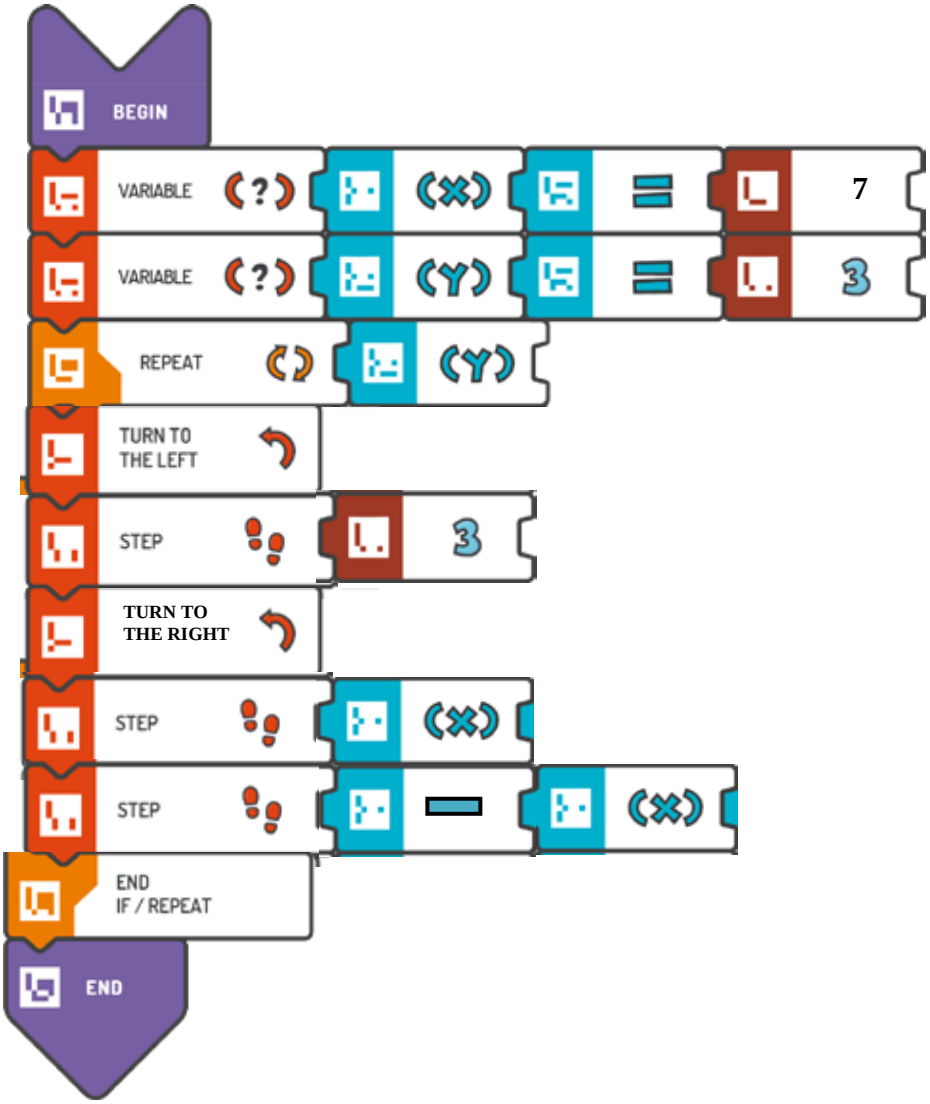
解答例（テキスト形式）：

```
BEGIN
    VARIABLE
    X=4
    VARIABLE
    Y=3   REPEAT
    (Y) {
        STEP (X)
        STEP (-X)
        TURN TO THE LEFT
    }
END
```

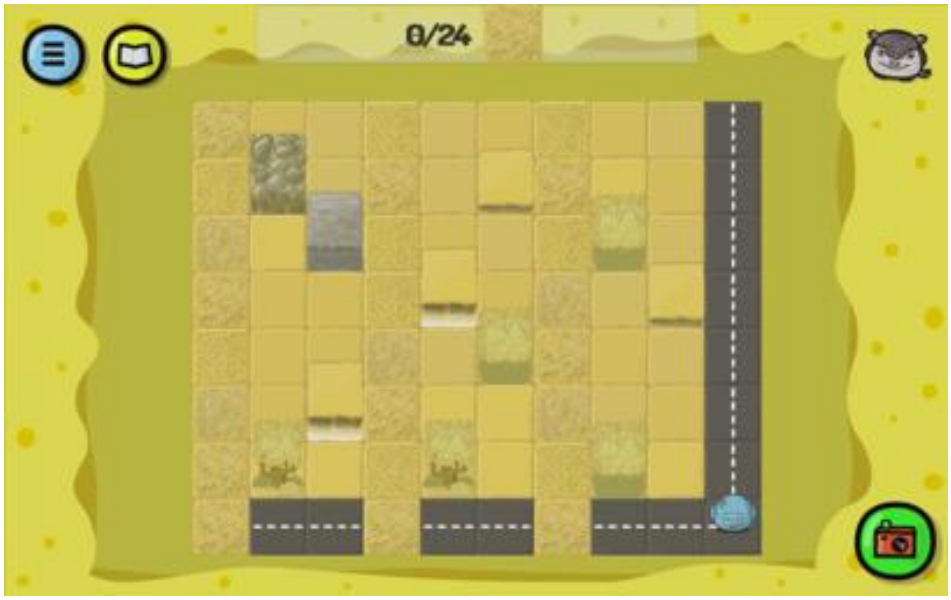
挑戦してみよう:


0/24

解答例:


BEGIN
VARIABLE
(?)
(x)
=
7
VARIABLE
(?)
(y)
=
3
REPEAT
(y)
TURN TO
THE LEFT
STEP
3
TURN TO
THE RIGHT
STEP
(x)
STEP
(x)
END
IF / REPEAT
END

挑戦してみよう:



解答例（テキスト形式）：

```
BEGIN
    VARIABLE
    Y=3
    VARIABLE
    X=7  REPEAT
    (Y) {
        TURN TO THE LEFT
        STEP (3)
        TURN TO THE
        RIGHT STEP (X)
        STEP (-X)
    }
END
```

モジュール8
オーストラリア2

条件文 – IF、ELSE

適切なスクエアに、不足している木を植えましょう。

このモジュールでは、プレイヤーは、条件文のコマンドを使って、各クエストを完了します。

モジュール導入 1/4:
条件文 – IF、ELSE

私達は誰でも、毎日何らかの決定をしています。学校にジャケットを着て行こうか、セーターを着て行こうか？朝ごはんに何個サンドイッチを食べようか？学校にバスで行こうか、地下鉄で行こうか？コンピュータープログラムは何かを常にチェックし、計算し、決定し続けています。プログラムを書く際に、条件文を使おうと決めた場合、プログラムが何をすべきか正確に分かるように、コンピューターにすべてを明確に説明しなくてはいけません。

IF、ELSEコマンドの組み合わせ

IF、ELSEは、調和することが多く、決定の際、最も基本的な論理構造の1つを構成する、2つのコマンドです。条件文は強力なツールであり、そのため、ほとんどのコンピュータープログラムに登場します。

一般的な条件文は、以下の主要な要素から成り立っています:

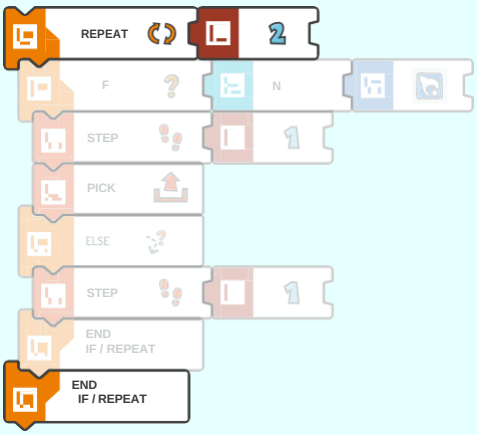
1. 確認する条件、

モジュール導入 2/4:
条件文

2. 条件が合う場合に実行されるコマンド、

3. 条件が合わない場合に実行されるコマンド。

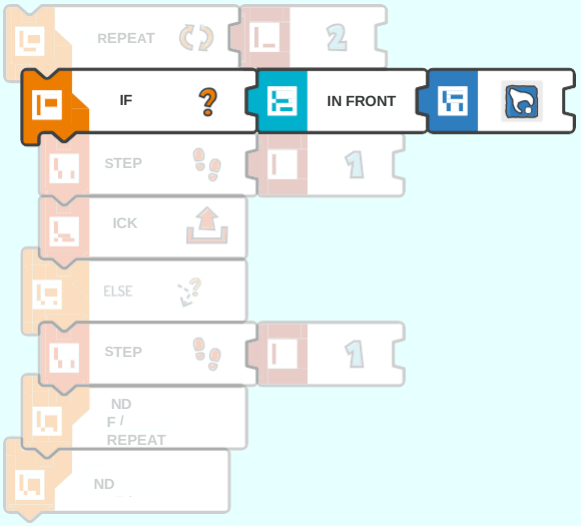
下の例を見てみましょう。条件文の概念をもっと簡単に理解できるよう、**REPEAT(2)ループ内に条件文を置くことで、**条件文を2回使い（呼び出し）ます。



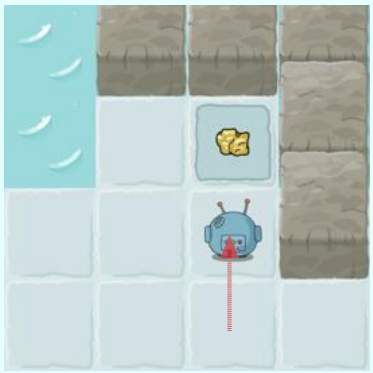
モジュール導入 3/4:
条件文

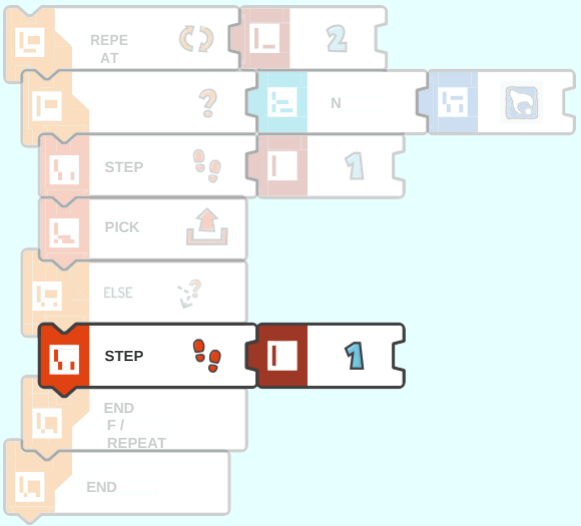
実行中の条件文:

1. スコッティは、初めて条件を確認します: **「前方に収集物があるか？」** 下の例では、この条件は合いません。



2. 条件が合わない場合、**ELSE**指示が実行されます。その結果、スコッティはスクエア1つ、前に進みます - **STEP(1)**。

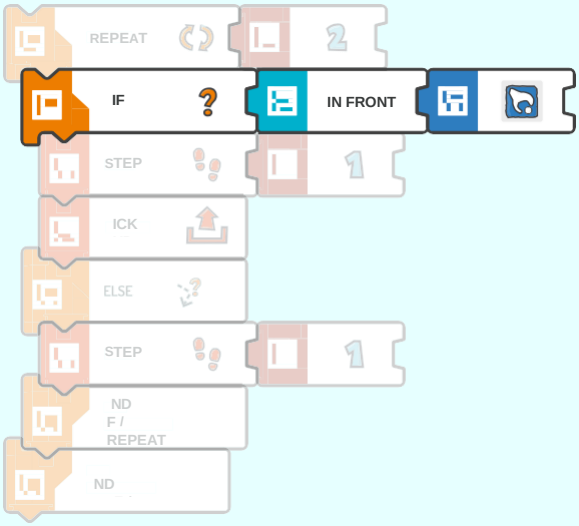




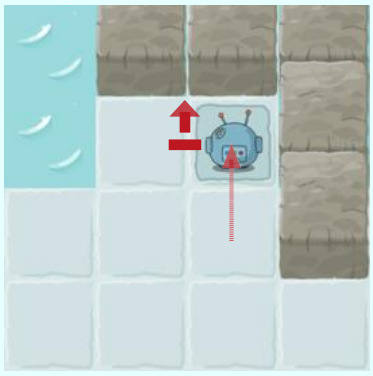
モジュール導入 4/4:
条件文

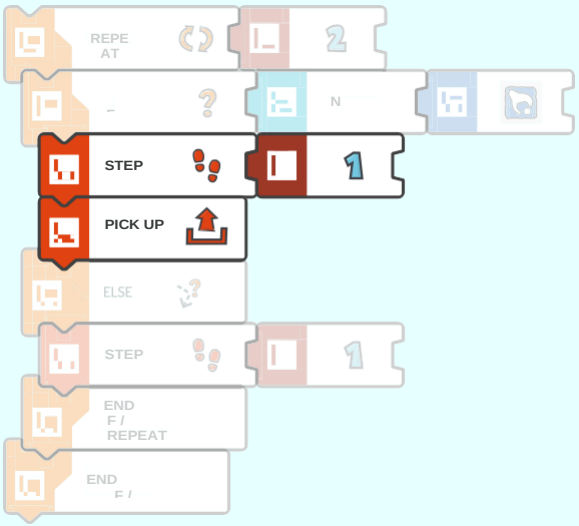
実行中の条件文の続き:

3. REPEAT(2)ループは、**「前方に収集物があるか？」**という条件を再確認するよう要求します。今回は、**この条件は合っています。**



4. 条件が合っているので、スコッティは**IF**行のすぐ下のコマンド、つまり**STEP(1)**と**PICK UP**を実行し始めます。





注意

条件文の最初の条件が合っている場合、他の条件は確認、実行されません。

導入:
アクションスクエア

スコッティの動物仲間の1匹が、美しい果樹園を持っています。友達の手助けのお返しに、スコッティは木を植えてあげる約束をしました。

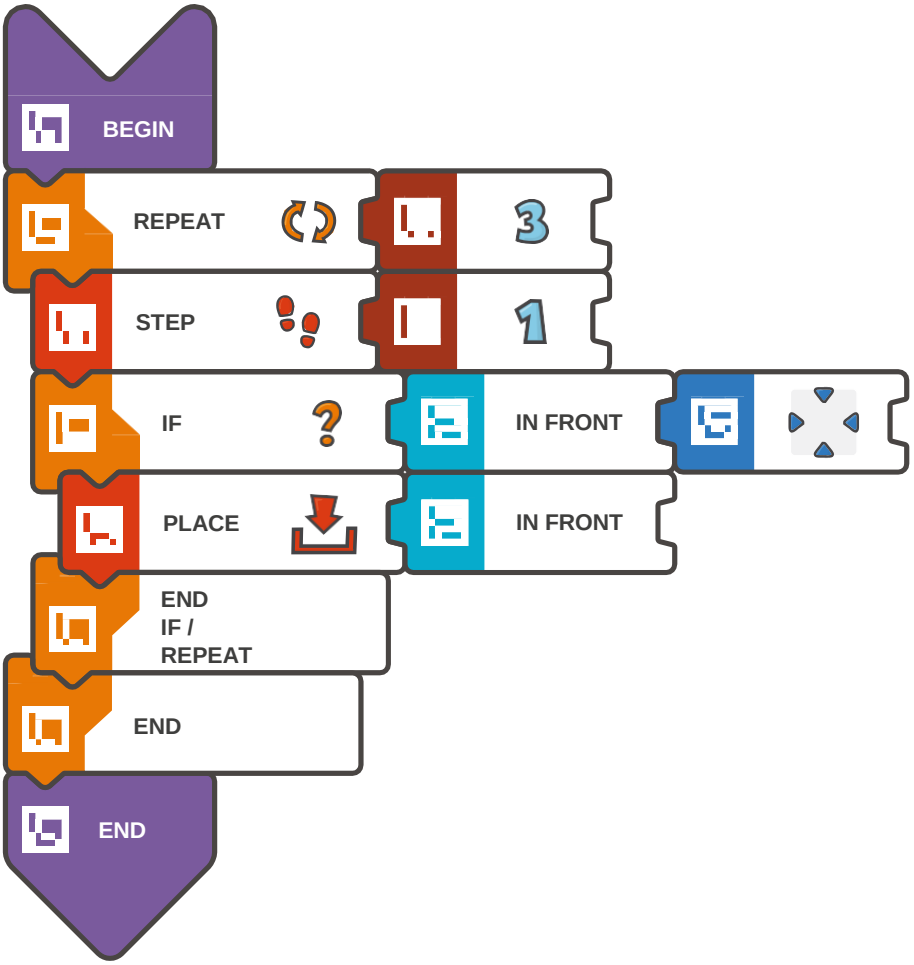
この目的を達成するには、新しいタイルを使って、指定されたスクエアに足りない木を植えます。このクエストは、**アクションスクエア**タイルと関連付けて**IF**条件文を使うことに重点を置いています。確認する条件は、スコッティに対するアクションスクエアの位置です。例えば:

覚えておこう!

END IF / REPEATタイルは、必ず条件文の最後に置きます。

挑戦してみよう（**このクエストでは条件文を使います**）：

解答例:



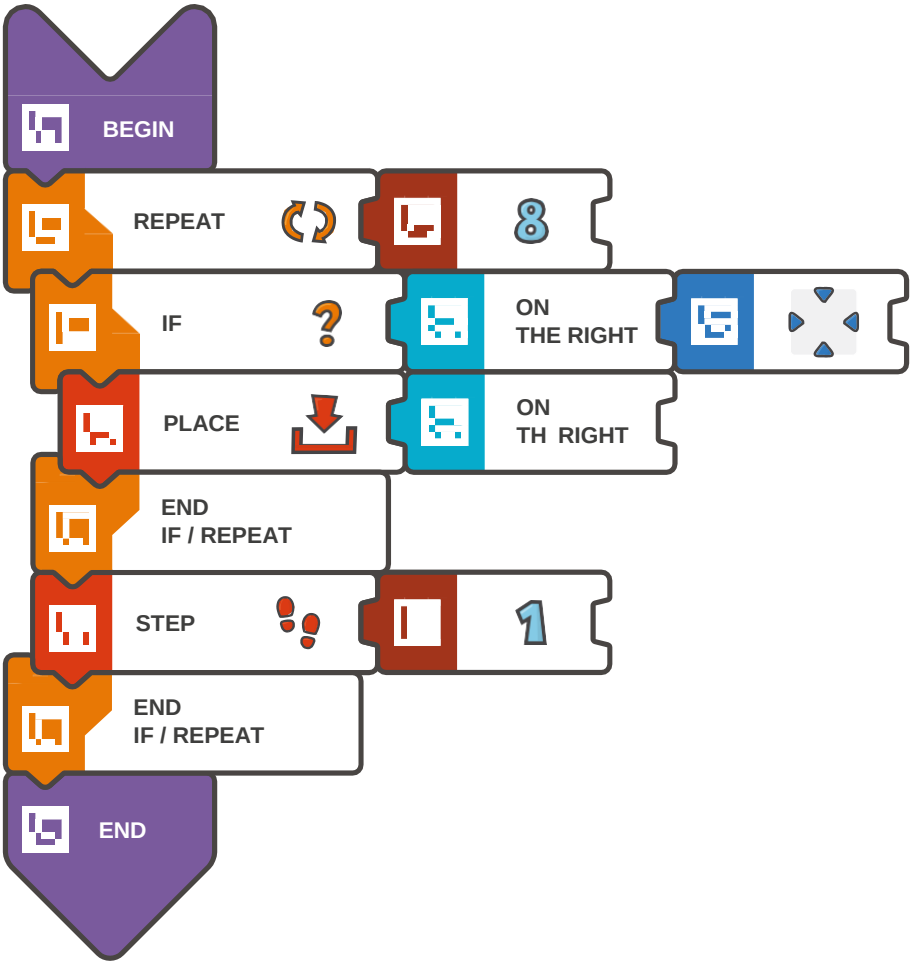
挑戦してみよう（**このクエストでは条件文を使います**）：

解答例（テキスト形式）：

```
BEGIN
    REPEAT (3) {
        STEP (1)
        IF (IN FRONT < ACTION
            SQUARE) { PLACE (IN
            FRONT)
        }
    }
END
```

挑戦してみよう（このクエストでは条件文を使います）：

解答例:



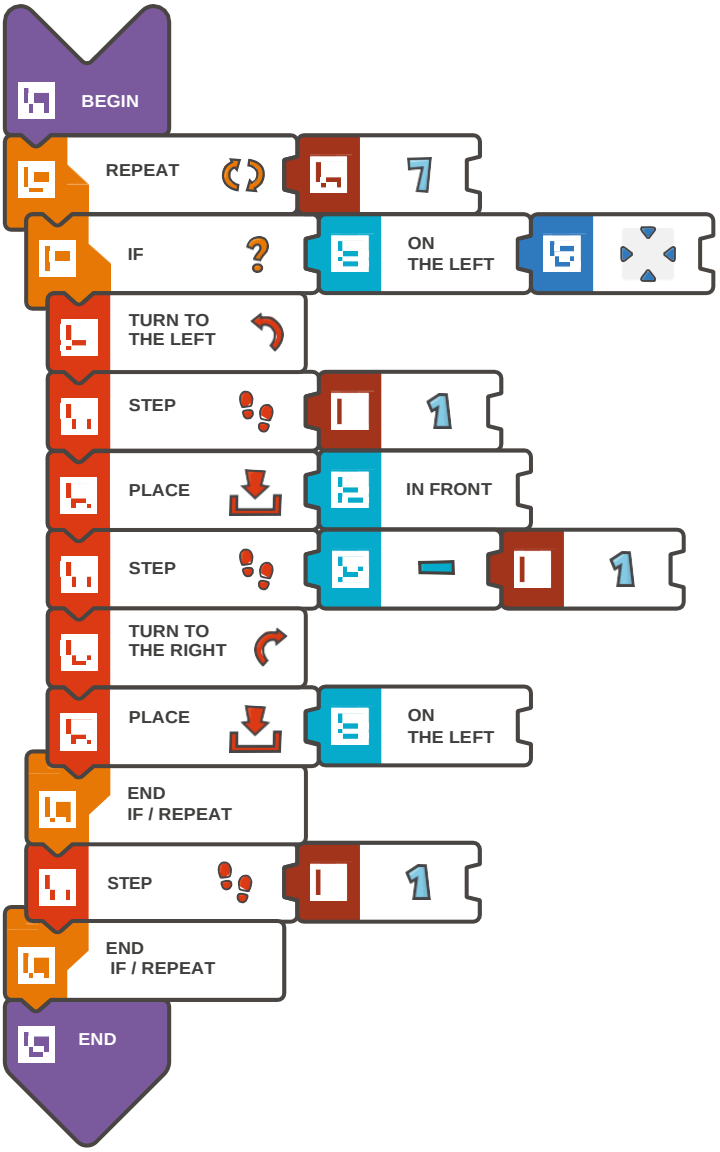
挑戦してみよう（**このクエストでは条件文を使います**）：

解答例（テキスト形式）：

```
BEGIN
    REPEAT (8) {
        IF (ON THE RIGHT < ACTION
            SQUARE) { PLACE (ON THE
            RIGHT)
        }
        STEP (1)
    }
END
```

挑戦してみよう（このクエストでは条件文を使います）：

解答例:



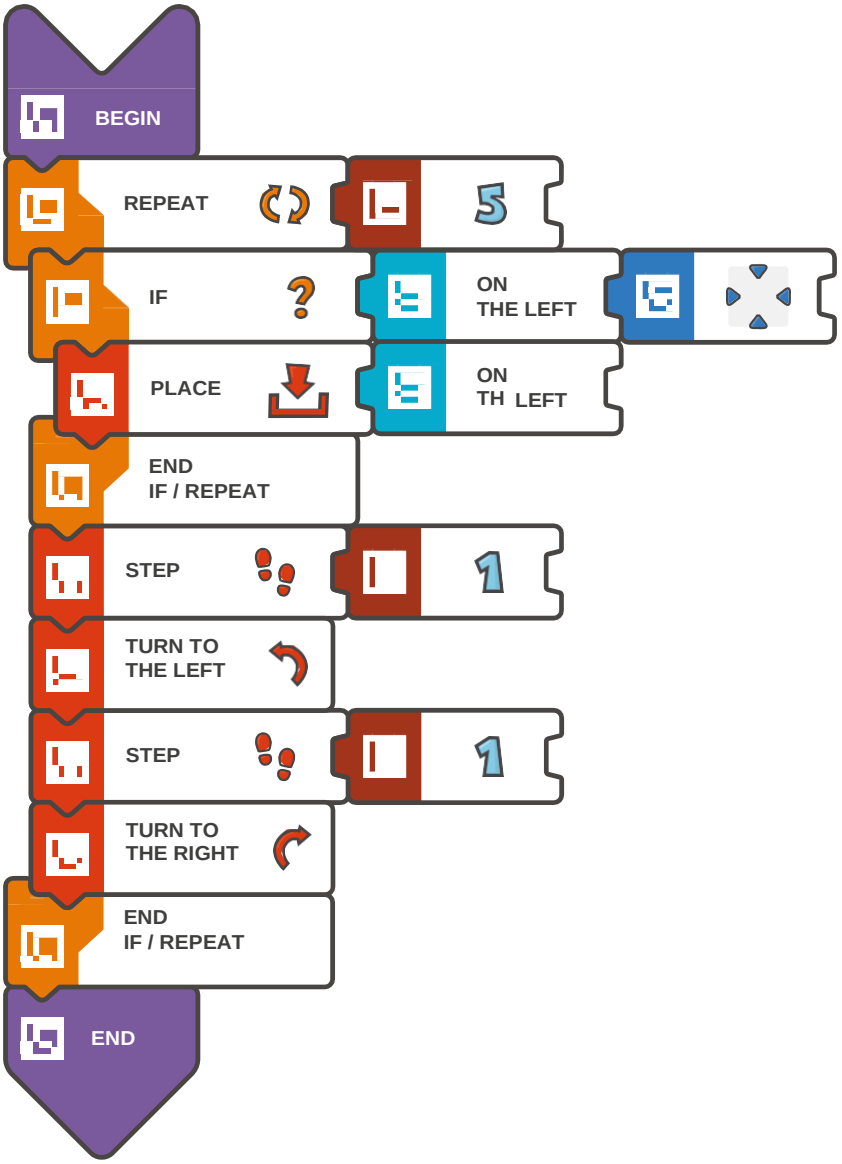
挑戦してみよう（このクエストでは条件文を使います）：

解答例（テキスト形式）：

```
BEGIN
    REPEAT (7) {
        IF (ON THE LEFT < ACTION SQUARE)
            { TURN TO THE LEFT
            STEP (1)
            PLACE (IN
            FRONT) STEP (-1)
            TURN TO THE RIGHT
            PLACE (ON THE
            LEFT)
        }
        STEP (1)
    }
END
```

挑戦してみよう（**このクエストでは条件文を使います**）：

解答例:



挑戦してみよう（このクエストでは条件文を使います）：

解答例（テキスト形式）：

```
BEGIN
    REPEAT (5) {
        IF (ON THE LEFT < ACTION SQUARE)
            { PLACE (ON THE LEFT)
        }
        STEP (1)
        TURN TO THE
        LEFT STEP (1)
        TURN TO THE RIGHT
    }
END
```

ヒント

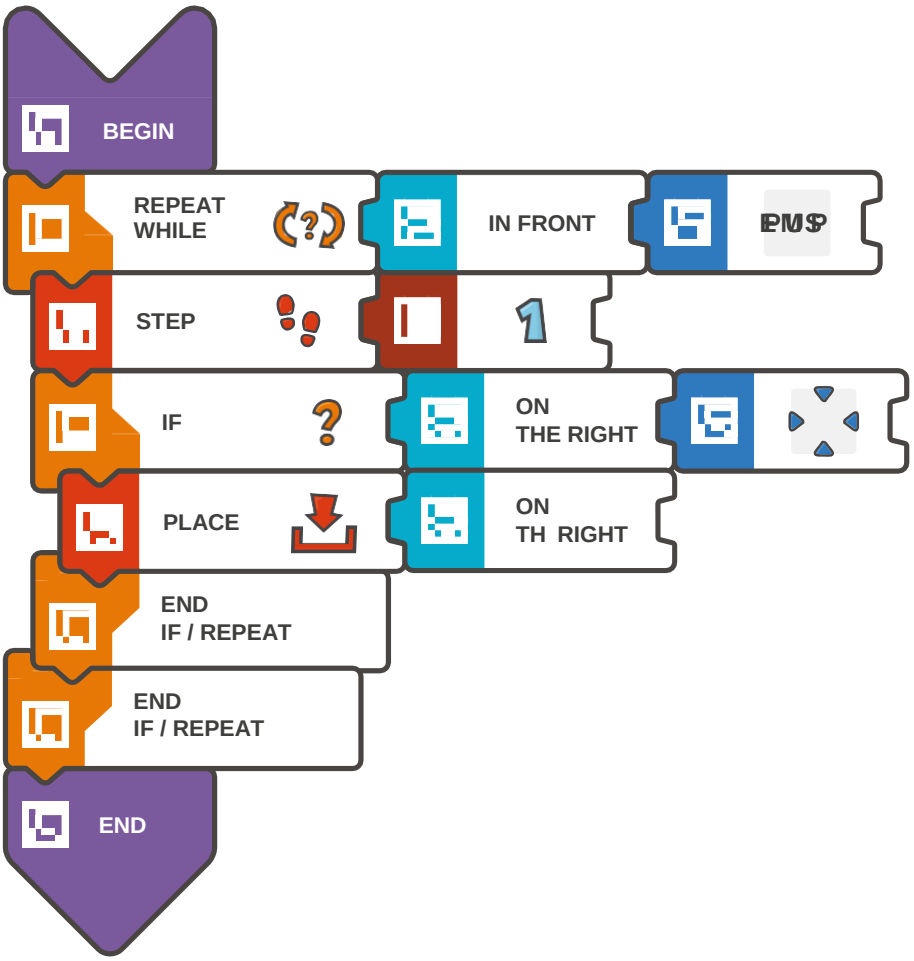
このクエストでは、条件付きループを使って、別の条件を加えるIFの文とつなげる必要があります。

スコッティの道には障害物があるため、通常のREPEATループの代わりにREPEAT WHILEループを使うことができます。

挑戦してみよう

（ゲームボードの端に障害物があります。これを利用して、通常のREPEATループの代わりにREPEAT WHILEループを使いましょう）：

解答例:



挑戦してみよう

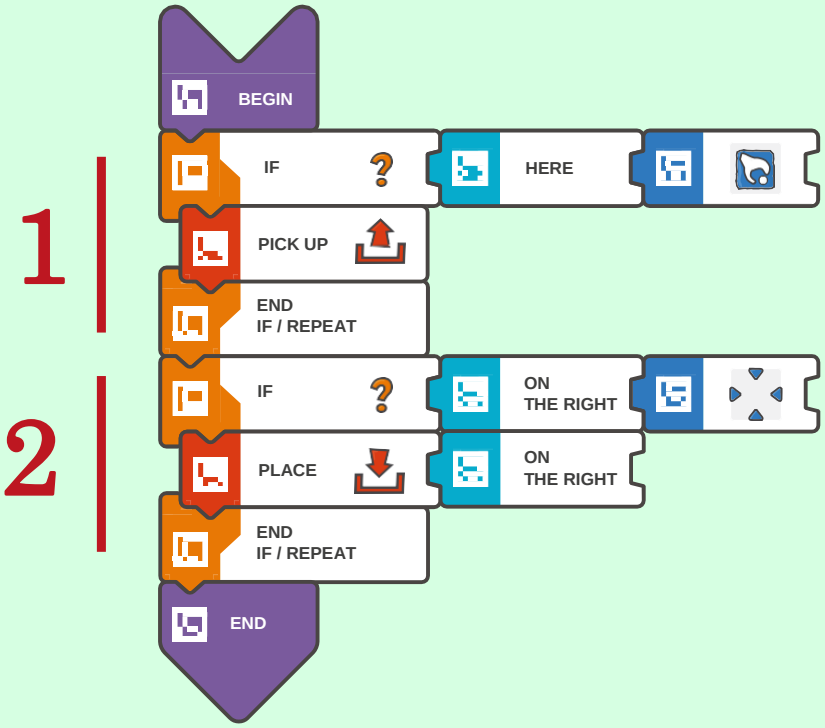
（ゲームボードの端に障害物があります。これを利用して、通常のREPEATループの代わりにREPEAT WHILEループを使いましょう）：

解答例（テキスト形式）：

```
BEGIN
    REPEAT WHILE (IN FRONT <
        EMPTY) { STEP (1)
        IF (ON THE RIGHT < ACTION
            SQUARE) { PLACE (ON THE
            RIGHT)
        }
    }
END
```

導入:
複数の条件文を使う

1つのプログラムで複数の条件文を使うことができます。これは、論理をテストし、改善する優れた方法です。また、一般的なプログラミング言語の構文の重要な要素を練習する良い方法でもあります。それぞれの条件文で、様々な条件を確認できます。例えば:



上のプログラムでは、スコッティは以下の行動をします:

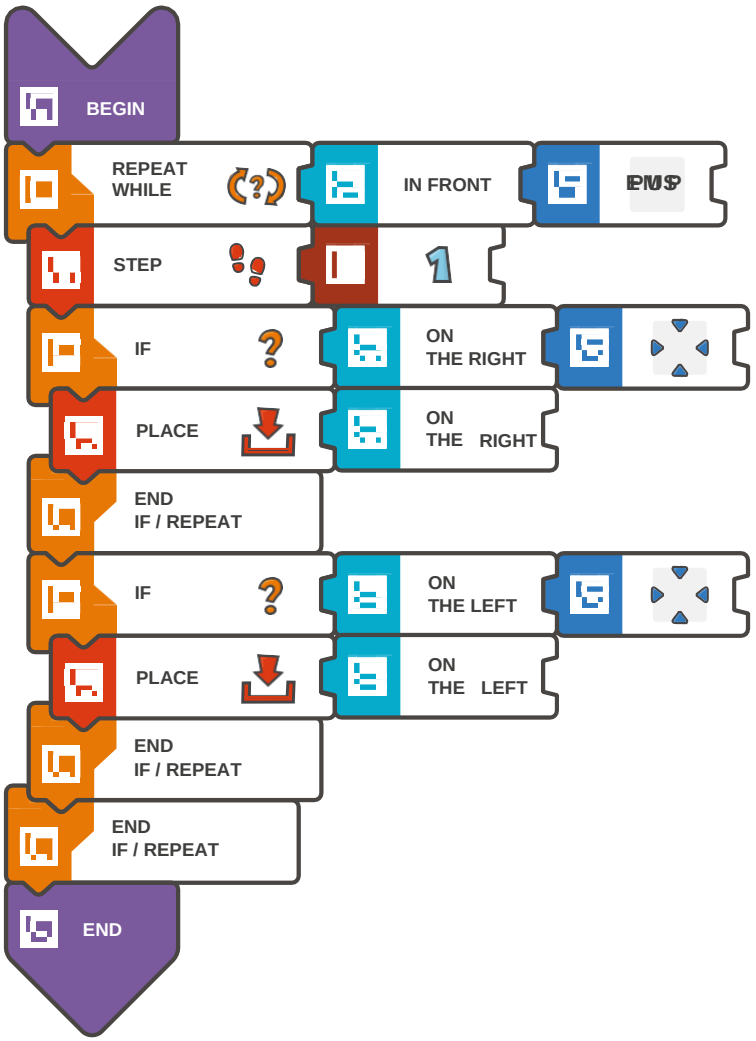
1. **最初に、**今立っているスクエアに収集物があるか確認します:
   - **ある場合、**スコッティは、ものを拾います（そして次の条件の確認に進みます）、
   - **ない場合、**スコッティは、**IF**行のすぐ下のコマンドを飛ばします（そして次の条件の確認に進みます）。

2. **次に、**スコッティは、右側のスクエアにアクションスクエアがあるか確認します:
   - **ある場合、**スコッティは、このスクエアに在庫からものを置きます（そして次の条件の確認に進みます）
   - **ない場合、**スコッティは、**IF**行のすぐ下の2番目のコマンドを飛ばします（さらに、この場合、**END**タイルでプログラムを終了します）。

挑戦してみよう

（このクエストでは、1つ以上の条件文を使いましょう。それぞれの条件文は別の条件を確認します）：

解答例:



挑戦してみよう

（このクエストでは、1つ以上の条件文を使いましょう。それぞれの条件文は別の条件を確認します）：

解答例（テキスト形式）：

```
BEGIN
    REPEAT WHILE (IN FRONT <
        EMPTY) { STEP (1)
        IF (ON THE RIGHT < ACTION
            SQUARE) { PLACE (ON THE
            RIGHT)
        }
        IF (ON THE LEFT < ACTION SQUARE)
            { PLACE (ON THE LEFT)
        }
    }
END
```

挑戦してみよう

（在庫には新しい苗木がもうありません。スコッティは、新しい場所に苗木を植える前に、苗木を拾わなくてはいけません）：

解答例:

挑戦してみよう

（在庫には新しい苗木がもうありません。スコッティは、新しい場所に苗木を植える前に、苗木を拾わなくてはいけません）：

解答例（テキスト形式）：

```
BEGIN
    REPEAT WHILE (IN FRONT <
        EMPTY) { STEP (1)
        IF (ON THE RIGHT < ACTION
            SQUARE) { PLACE (ON THE
            RIGHT)
        }
        IF (ON THE LEFT < COLLECTABLE
            OBJECT) { TURN TO THE LEFT
            STEP (1)
            PICK UP
            STEP (-1)
            TURN TO THE RIGHT
        }
    }
END
```

導入:
隆起したスクエアタイルとHEREタイル

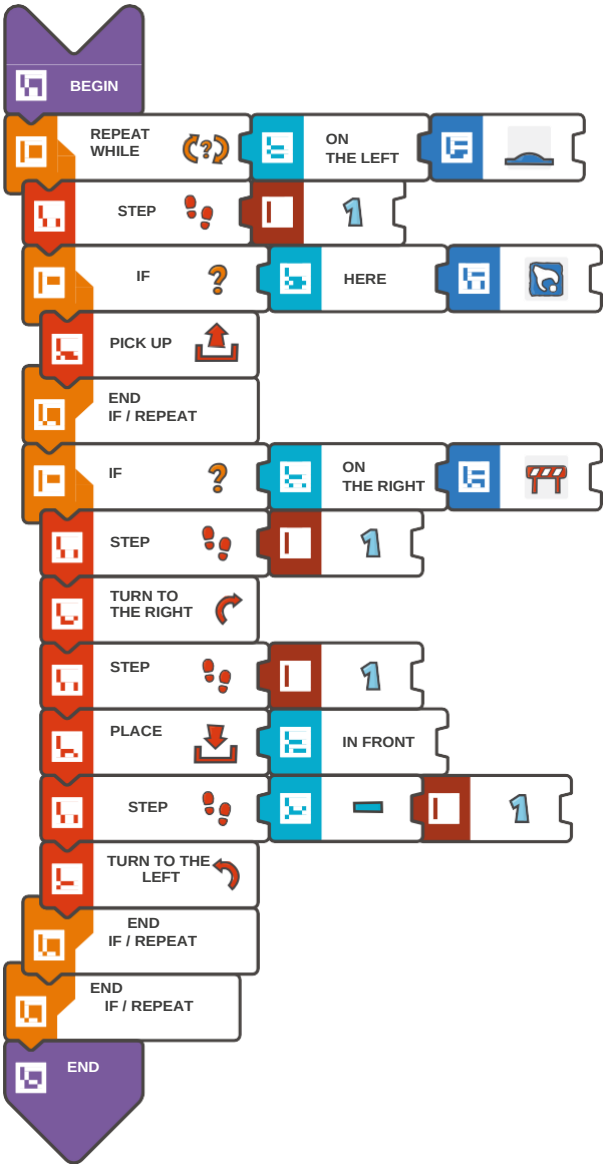
オーストラリアの別の場所にいるスコッティの友達は岩だらけの丘陵地にいます。丘を動き回る方法をスコッティに見せます。**隆起したスクエア**タイル

と**HERE**タイルを使います。

前に学習した条件文と関連付けて、新しいタイルを使います。

挑戦してみよう（このモジュールでは、条件文を使いましょう）：

解答例:



挑戦してみよう（このモジュールでは、条件文を使い

解答例（テキスト形式）：

```
BEGIN
    REPEAT WHILE (ON THE LEFT < ELEVATED
        SQUARE){ STEP (1)
        IF (HERE < COLLECTABLE
            OBJECT) { PICK UP
        }
        IF (ON THE RIGHT <
            OBSTACLE) { STEP (1)
            TURN TO THE
            RIGHT STEP (1)
            PLACE (IN
            FRONT) STEP (-1)
            TURN TO THE LEFT
        }
    }
END
```

モジュール9
北アメリカ2

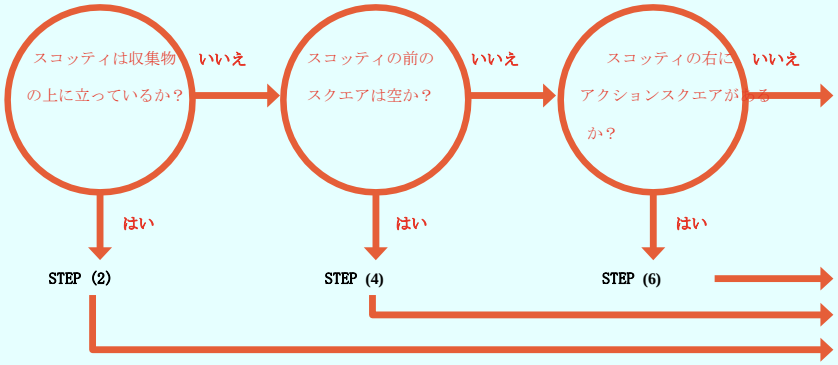
ループと条件文: IF、ELSE IF、ELSE

スコッティの宇宙船はほとんど完成です!けれども、作業員たちが作業場をひどく散らかしたままにしています。スコッティが、床に散らばったネジや道具を集めるのを手伝いましょう。

モジュール導入(A): ELSE IF 1/3

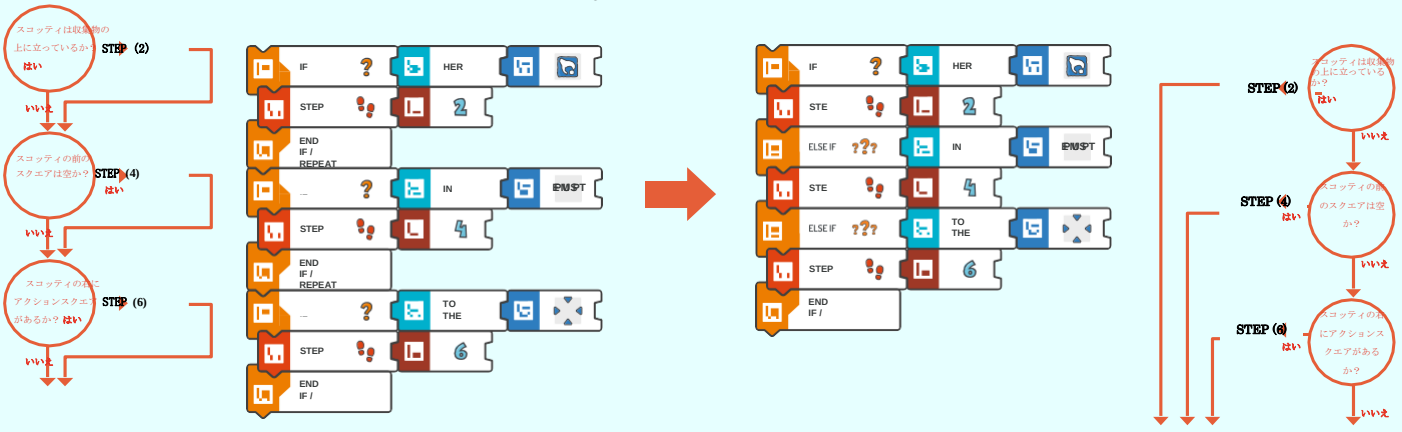
モジュール8で、1つのプログラムで複数の条件文を使う必要がよくあることが分かりました。この必要性によって、結果的に新しいプログラミング構造が発展し、多くの条件のうち1つだけが実行されます。

下の例のように、複数の条件を順番に確認します。



この場合、条件文**ELSE IF**を使う必要があります。

ELSE IFタイルを使うことで、複数の条件文を編集する代わりに、1つの条件文を拡張してプログラムを簡素にすることができます。



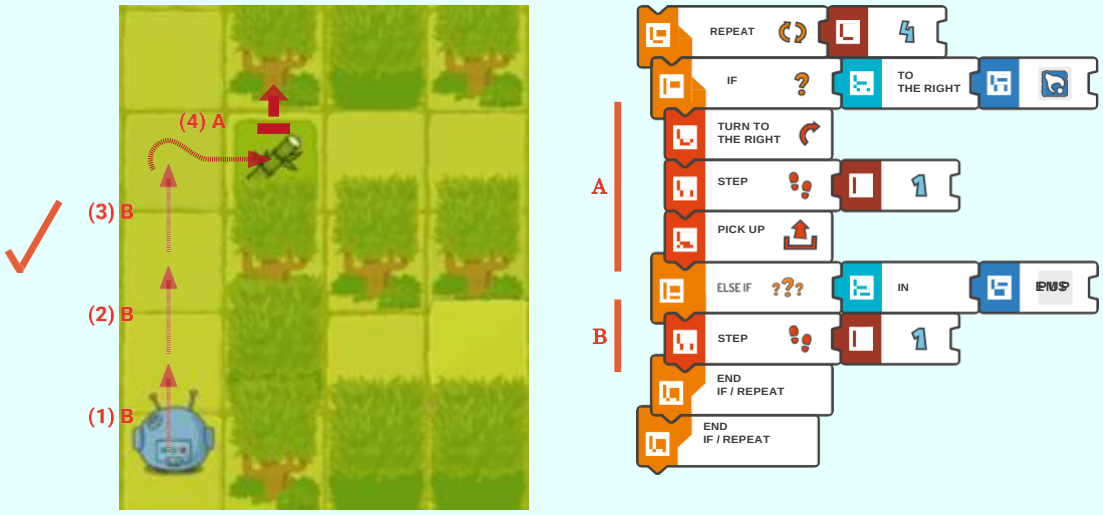
ELSE IFタイルを使用する際、条件の順番が決定的に重要です。

最初の条件が合う（事実である）場合、**2番目や3番目の条件は確認、実行されません。**

最初の条件が合わない場合、プログラムは2番目の条件の確認に進み、条件が合う（事実である）場合、**3番目の条件は確認、実行されません。**

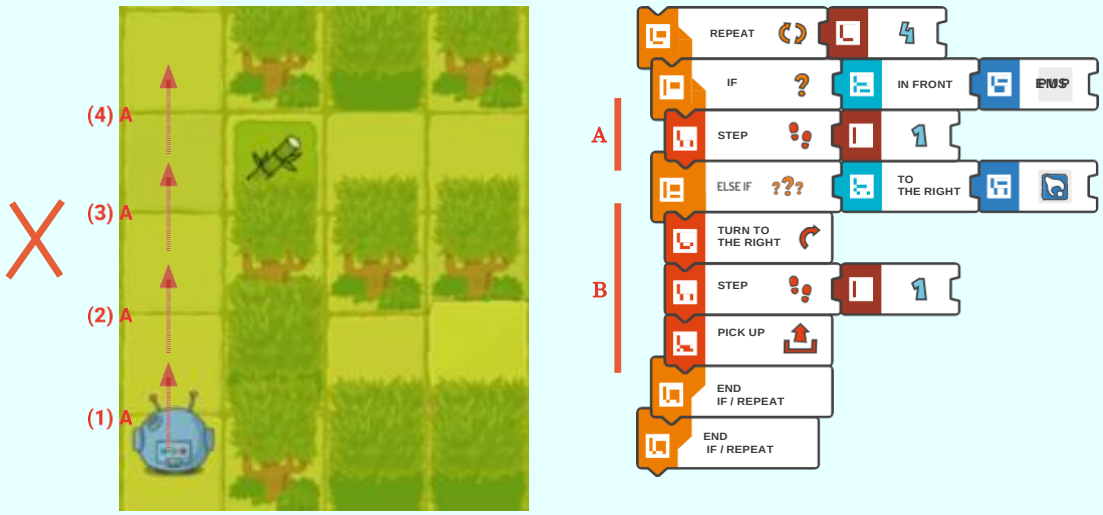
モジュール導入(A): ELSE IF 2/3

例1 - ELSE IF文を使う。スコッティは竹を拾うことになっている: 正しい解答。



正しい解答と誤った解答の違いは、**確認する条件の順番**（および、その条件に対応するコマンド）**のみ**です。**ループの繰り返し（反復）は、1つの条件のみに関連するコマンドを実行する**ことに注意しましょう。

誤った解答。

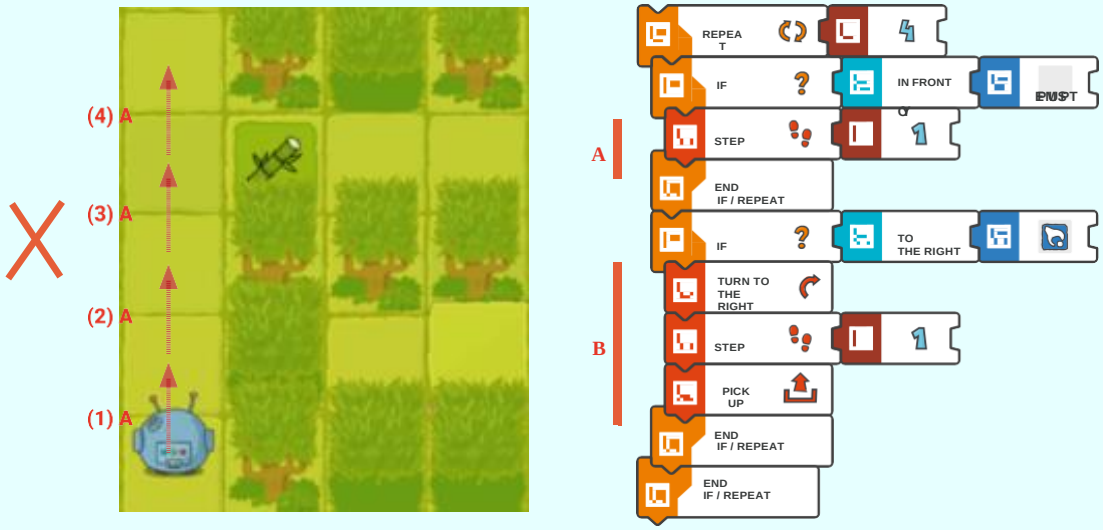


モジュール導入(A): ELSE IF 3/3

例2 - 2つのIF文を使う。**スコッティは竹を拾うことになっている: 正しい解答。**

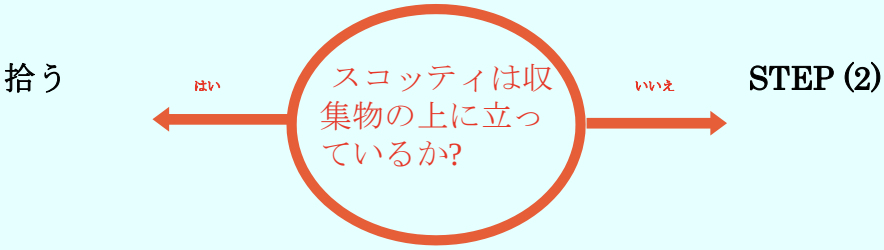
ELSE IFタイルを使用しない正しい解答は**前より1行長く**なります。**ループの4回目の繰り返し(反復)では、両方の条件が合う(事実である)ことに注意しましょう。**もちろん、この場合も、条件の順番は決定的に重要です。

誤った解答。



モジュール導入(B):
ELSEタイル 1/3

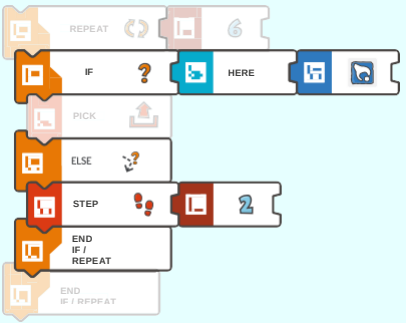
複数の条件文を使うこと以外に、**前の条件が合わない場合のみ、**スコッティに何かさせなければならないことがあります。例えば、**2つの代替動作のみをどのようにコード化するか?**



上に示した構造を作成するには、プログラミングで**ELSE**指示を使います。

スコッティの『既定値』コマンドを描写する効率的な方法です。これは、以前にプログラムで特定した条件が合わない場合に実行するコマンドです。
コマンドが条件文の外にある場合、常に実行されます。これは、特にループでよく見られます。

ループの繰り返し（反復）で、スコッティは常に2歩進みます。



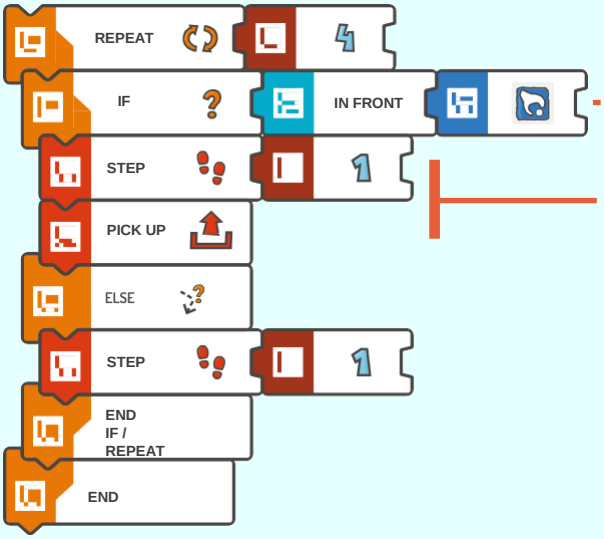
ループの繰り返し（反復）で、スコッティは、収集物の上に立っていない場合のみ、2歩進みます。

一般的なプログラミング言語では、同じような条件文を作成する際、**NO**という言葉を使うこともできます（**スコッティ・ゴー!では使いません**）。

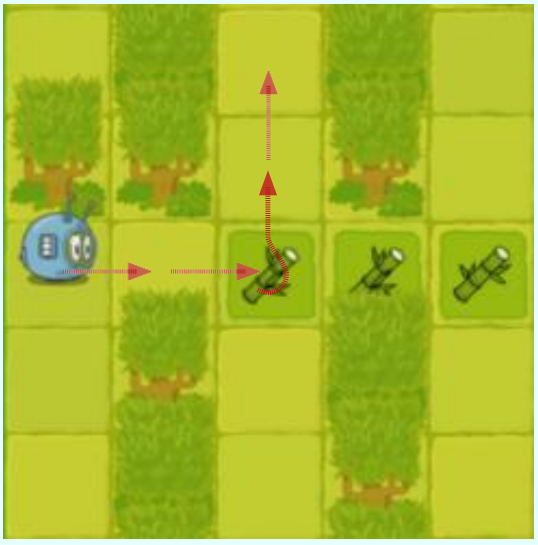
モジュール導入(B):
ELSEタイル 2/3

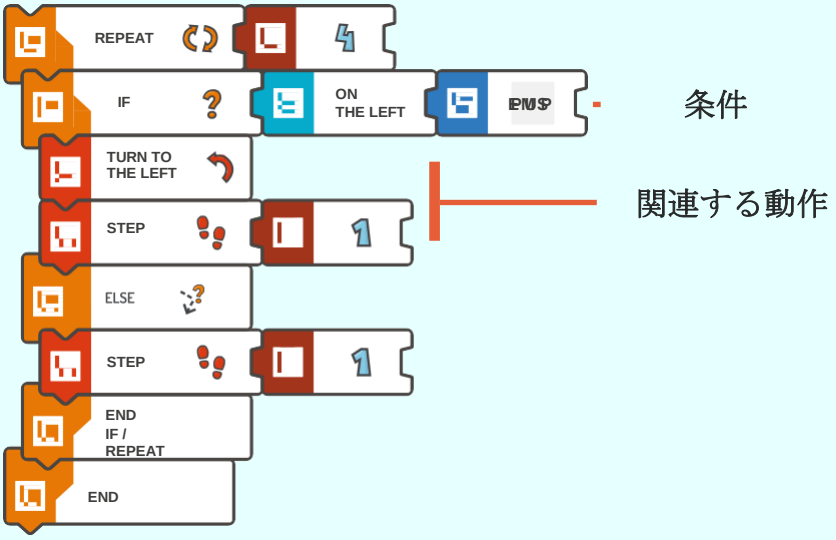
ELSEの文を使う例:

何かがスコッティの動作を要求するまで、スコッティが前進するプログラムを書きましょう。例えば、**スコッティは、前方に収集物があるか、左側に空のスクエアがあれば左に曲がるか等を確認します。**プログラムは（新しい）要求の変更や調整が簡単でなければいけません。



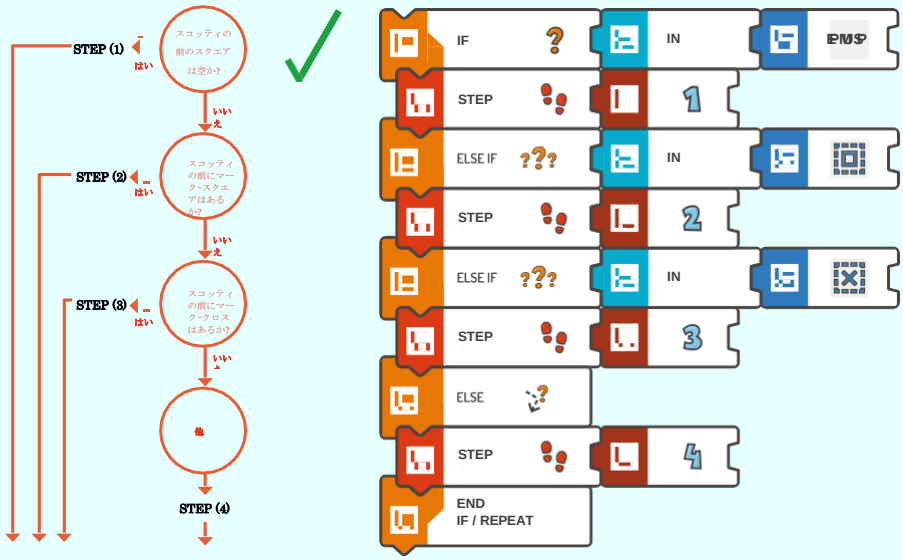
スコッティが前方に歩くための条件は定義する必要がありません。**どの条件にスコッティが反応するか、どのような反応をするかを特定するだけで十分です。**



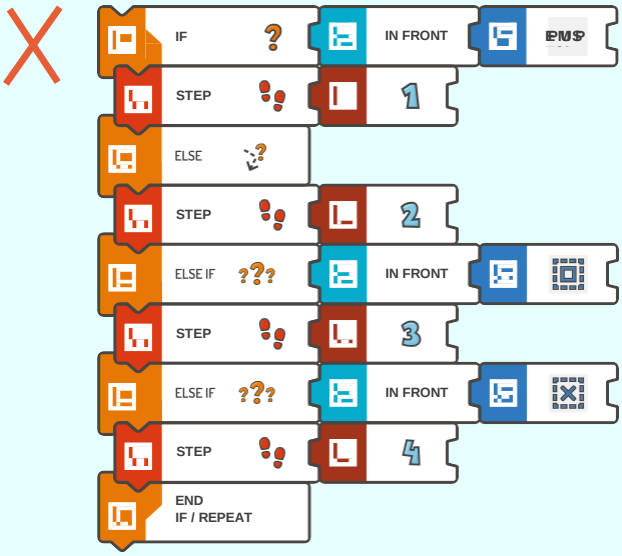


モジュール導入(B):
ELSEタイル 3/3

1つのプログラムで2つの文: **ELSE IF**と**ELSE**を使うことができます。



しかし、**ELSEを使えるのは一度だけで、プログラムの最後に置かなくてはならない**ということを覚えておきましょう。下に示すプログラムの一部は正しくありません。

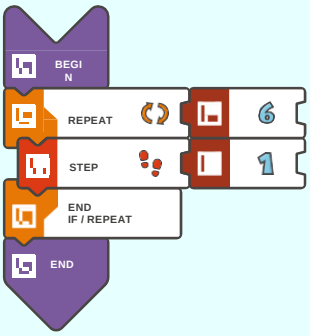


モジュール導入(C):
[オプション] – REPEAT (FOREVER) 1/3

REPEATループのない条件文を含むプログラムでは、一度だけ条件文を確認し、それからプログラムは終了します。しかし、スコッティがすべてのネジと道具を見つけて集めるまで、何度も条件を確認する必要があります。これを実行するには、新しいタイル: **FOREVER**を使います。

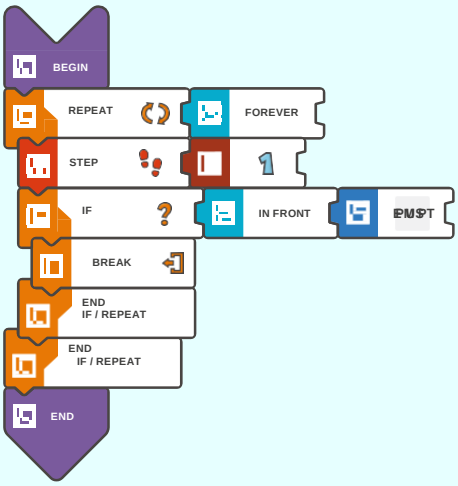
モジュール9のクエストでは、生徒の年齢や経験に応じて、下に示す2つの方法のどちらかで、**REPEAT**ループを使います:

1. REPEAT（特定の回数） - モジュール3で学んだコマンド。ループ内のコマンドを実行する回数の特定が課題です。**下の例では、スコッティは周囲に注意を払わずに1歩ずつ6回進みます。**



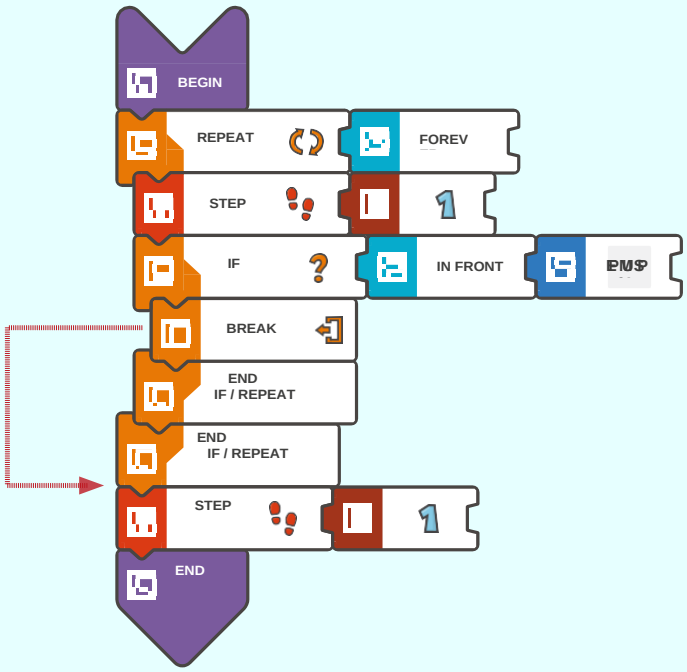
2. BREAKコマンドを使った**REPEAT (FOREVER)**

REPEAT (FOREVER)コマンドは、チュートリアルでは導入されませんが、興味深く、最も万能な解答です。**次のページでは、類似のプログラムを詳細に分析します。**



モジュール導入(C):
[オプション] - REPEAT (FOREVER) 2/3

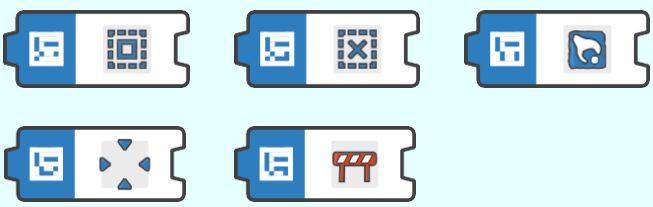
BREAKコマンドを使った**REPEAT (FOREVER)**。



上のプログラムでは、（例えば、ゲームボード上のものに関連した条件文のように、通常、条件文内に置く）**BREAK**コマンドをプログラムが呼び出すまで、**STEP 1**のコマンドを繰り返します。**例では、スコッティは、前方に空のスクエアがある場合、ループの実行を中止します。**

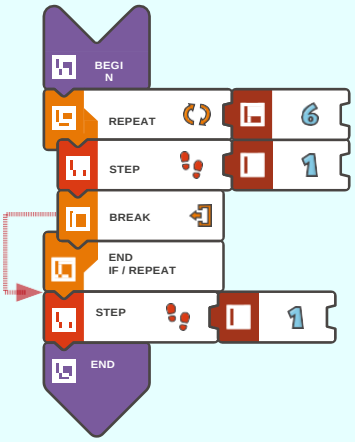
この条件は、戦略的位置に置かれた**BREAK**コマンドを呼び出し、プログラムが正常に機能する上で決定的に重要です。**ループ内であっても、BREAKコマンドの後に置かれているコマンドは、実行されません。**

作業場の床にある、このモジュールには、以下があります: **スクエアのマーク、X印、収集物、アクションスクエア、障害物**。これらは、適切な条件文を作成するための参考として使えます。



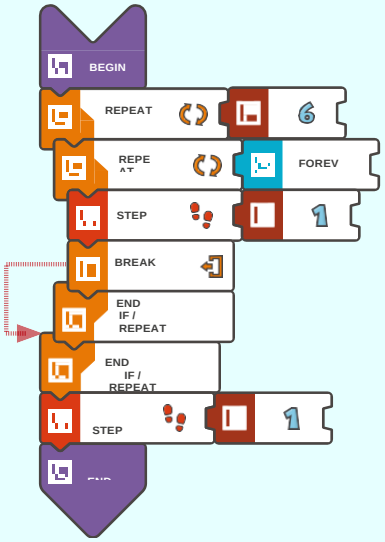
モジュール導入(C):
[オプション] - REPEAT (FOREVER) 3/3

BREAKコマンドを実行すると、プログラムは、**このコマンドを置いたループの外へ飛び出します。**



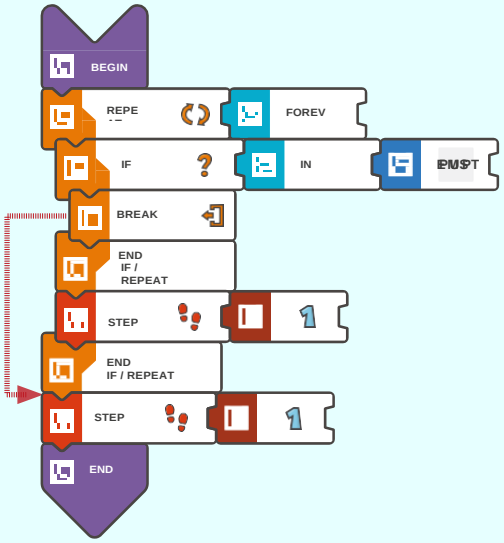
スコッティは2歩進みます。

（1歩はループ内;
もう1歩はループ外）



スコッティは7歩進みます。

（**FOREVER**ループの各反復に対し、
1歩はループ内
- 1回のみ - 6回繰り返す;
ループ外で1歩）



スコッティは、空のスクエアで止まるまで前進します。

（スコッティの前のスクエアが空ではない場合、スコッティは常に1歩進みます;

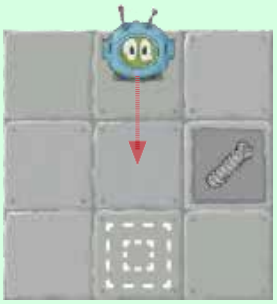
前のスクエアが空の場合、プログラムはループの外へ行き、スコッティに**END**コマンドの前まで最後に1歩進ませます）

導入: ELSE (1/2)

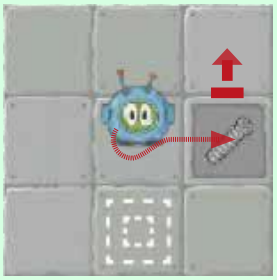
モジュール9のクエストでは、**条件文**と関連付けて**REPEAT**ループを使わなくてはいけません。

このクエストでは、**ELSE**文を使います。

ELSEタイルで、前の条件が合う場合のみ、スコッティが実行する一連のコマンドを定義できます。プログラムの例を見てみましょう。このプログラムでは、**スコッティは、立っている位置とゲームボード上のスクエアのマークの関係に基づいて行動します。**



ループの最初の反復:
スコッティの前には**スクエアのマーク**がありません -> **スコッティは、ELSEとEND IF / REPEATタイルの間にあるコマンドを実行します。**



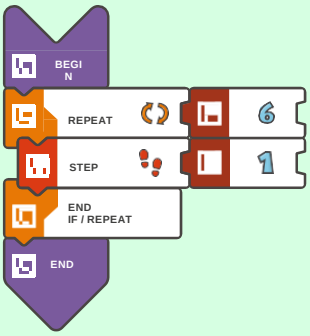
ループの2番目の反復:
スコッティの前に**スクエアのマーク**があります -> **スコッティは、IFとELSEタイルの間にあるコマンドを実行します。**

導入: ELSE (2/2)

> **注意!!**
>
> **このクエストや後のクエストでは、REPEAT (FOREVER)ループが使えます。**

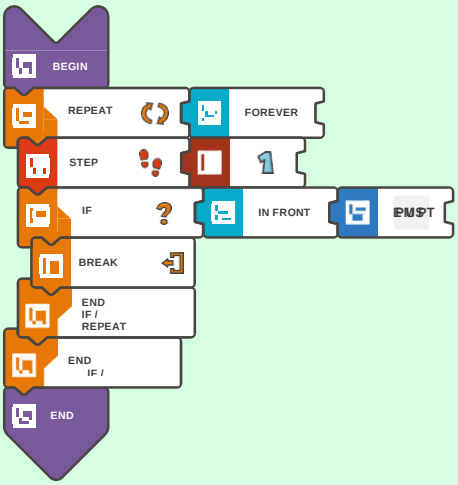
モジュール9のクエストでは、生徒の年齢や経験に応じて、下に示す2つの方法のどちらかで、**REPEAT**ループを使います:

3. 回数が特定されたREPEAT - モジュール3で学習したコマンド。ループ内のコマンドを何度実行するかを特定するのが課題です。下の例では、スコッティは、周囲に注意を払わずに6回、1



歩ずつ進みます。

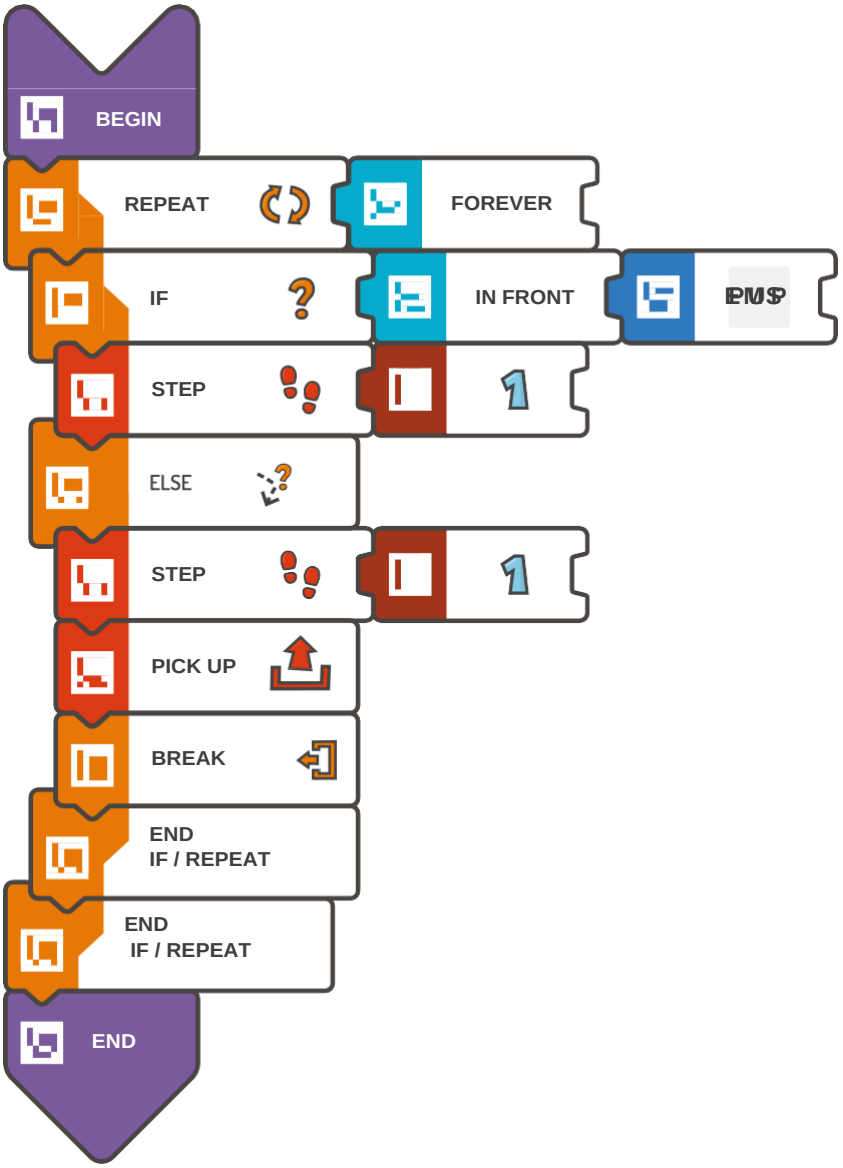
1. BREAKコマンドを使ったREPEAT (FOREVER)



REPEAT (FOREVER)コマンドはチュートリアルでは導入されませんが、興味深く、最も万能な解答です。**モジュール導入(C)に詳細を記載しています。**

挑戦してみよう（**このクエストでは、ELSEタイルを使いましょう**）：

解答例:



挑戦してみよう（このクエストでは、

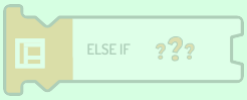
解答例（テキスト形式）：

```
BEGIN
    REPEAT (FOREVER) {
        IF (IN FRONT < EMPTY)
            { STEP (1)
        } ELSE {
            STEP
            (1)
            PICK
            UP
            BREAK
        }
    }
END
```

導入:
ELSE IFとMARKS(1/3)

モジュール9のクエストでは、**条件文**と関連付けて**REPEAT**ループを使わなくてはいけません。

このクエストでは、**ELSE IF**文を使います。



ELSE IFタイルで、最初の条件が合わない場合、**スコッティが確認する追加条件**を定義できます。

プログラムの例を見てみましょう。スコッティは、**条件の1つが合うまで**、1つずつ確認します。

例:

ループの最初の反復:

スコッティの前にスクエアのマークがあります -> **スコッティの実行：**

STEP (1)

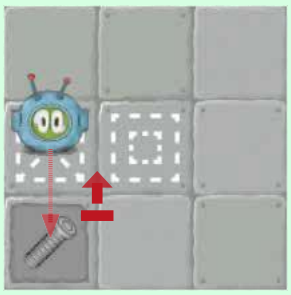
TURN TO THE RIGHT

ループの2番目の反復:

スコッティの前にX印があります -> **スコッティの実行:**

STEP (1)

TURN TO THE LEFT



ループの3番目の反復:

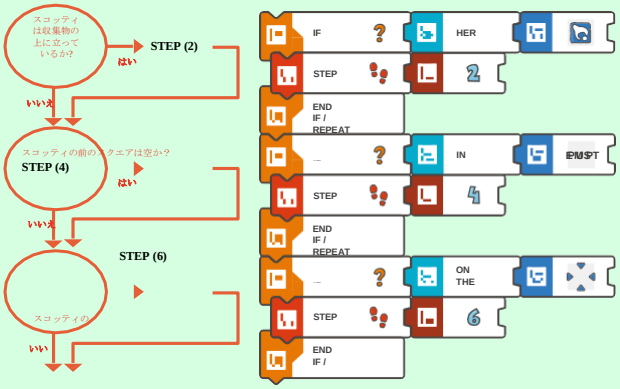
スコッティの前に何もマークがありません -> **スコッティの実行: STEP (1)**

PICK UP

導入:
ELSE IFとMARKS(2/3)

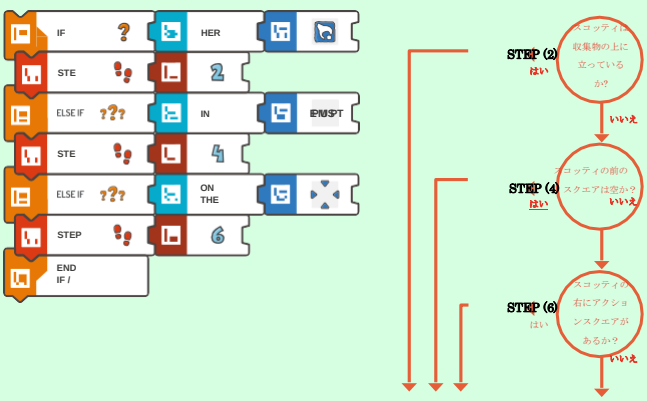
ELSE IFコマンドで、一連の代替コマンドを定義でき、**そのうち1つのコマンドのみ実行されます。**

ELSE IFを使用しない



スコッティは条件を１つずつ確認します。スコッティは2+4+6=12歩も進むことが可能です!

ELSE IFを使用



スコッティは条件を１つずつ確認します。この条件に合うと、他の条件に関連する動作は除外されます。スコッティは2歩、4歩または6歩進みます。

> **覚えておこう**
> **このモジュールのクエストや後のクエストでは、REPEAT (FOREVER)ループが使えます。**

REPEAT (FOREVER)コマンドはチュートリアルでは導入されませんが、興味深く、最も万能な解答です。**モジュール導入(C)に詳細を記載しています。**

導入:
ELSE IFとMARKS(3/3)

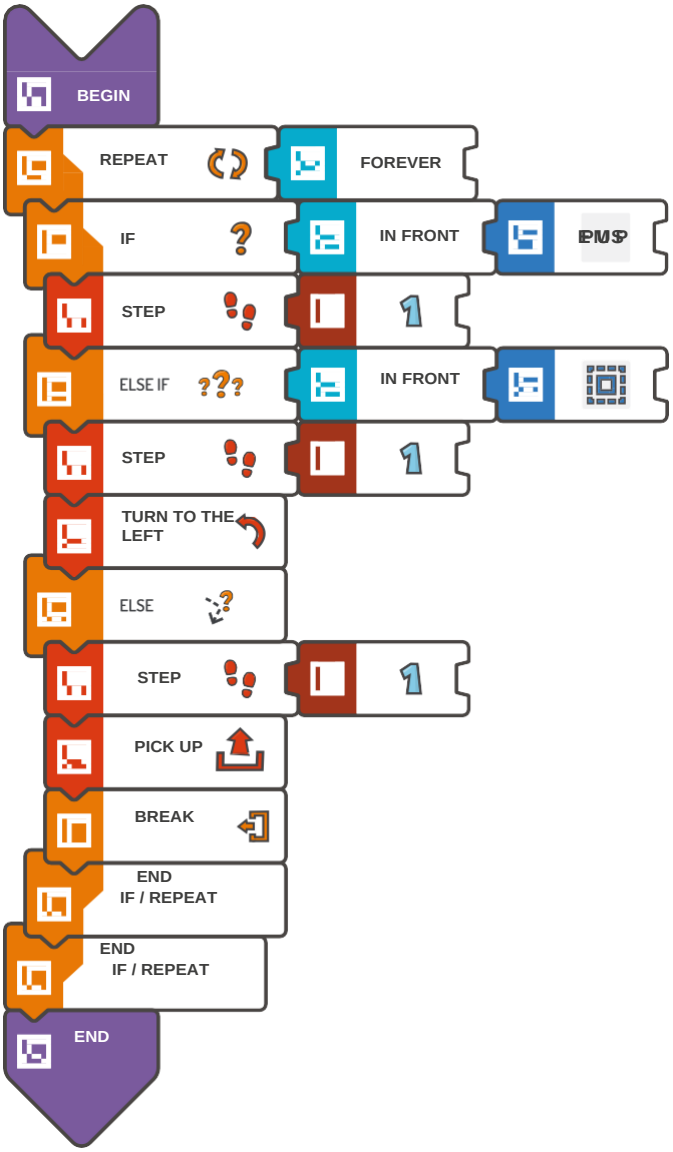
このモジュールでは、**収集物**、**アクションスクエア**、**障害物**のような、すでに知っているもの以外にも、**スコッティは**、作業場の床にある**スクエアのマーク**や**X印と関わることがあります**。

作業員たちは、作業場を整理するのに役立つよう、スクエア（正方形）のマークを位置確認地点として使います。このようなマークは、**条件文IF**や**ELSE IF**を使って、スコッティのためのコマンドを作成する基準点として使えます。

このクエストで条件文を作成するにはタイルを使います。

挑戦してみよう（**このクエストでは、ELSE IFタイルを使いましょう**）：

解答例:



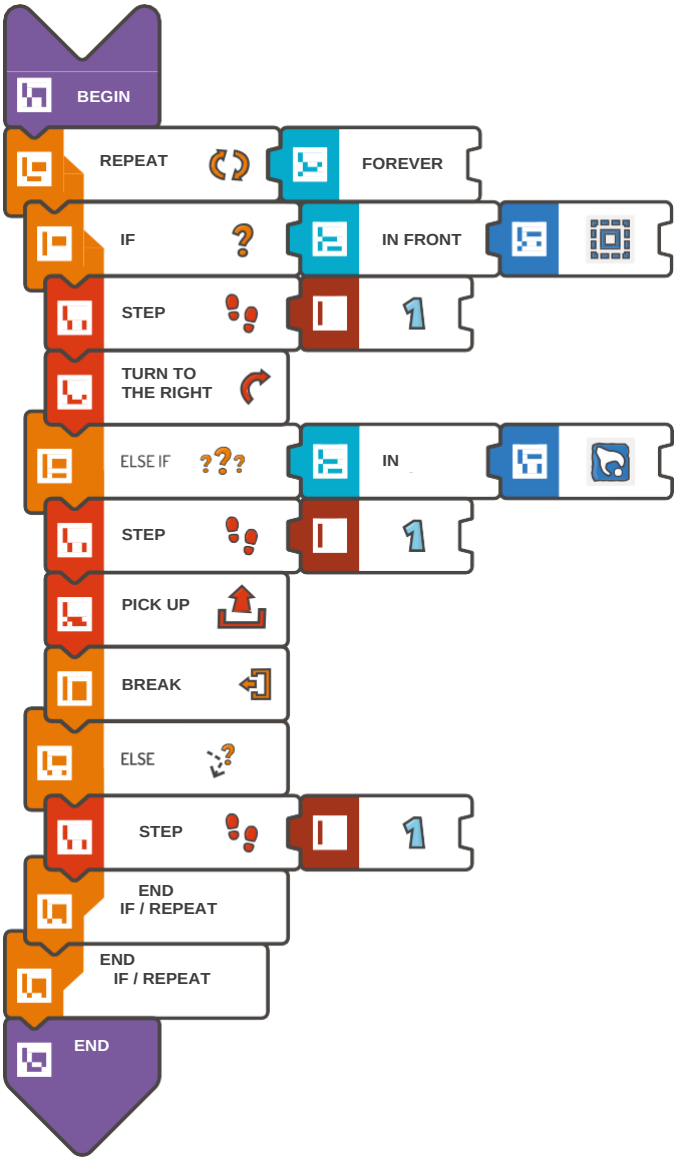
挑戦してみよう（このクエストでは、ELSE IFタイルを使いまし

解答例（テキスト形式）：

```
BEGIN
    REPEAT (FOREVER) {
        IF (IN FRONT < EMPTY)
            { STEP (1)
        } ELSE IF (IN FRONT < MARK -
            SQUARE) { STEP (1)
            TURN TO THE LEFT
        } ELSE {
            STEP
            (1)
            PICK
            UP
            BREAK
        }
    }
END
```

挑戦してみよう（**このクエストでは、複合条件文を使いましょう**）：

解答例:

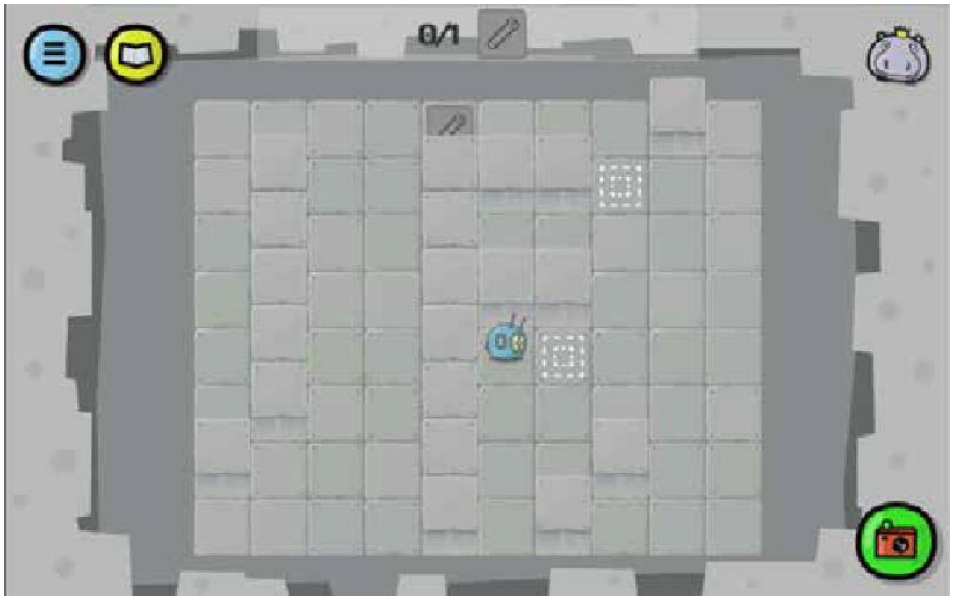


挑戦してみよう（このクエストでは、複合条件文を使いましょう）：

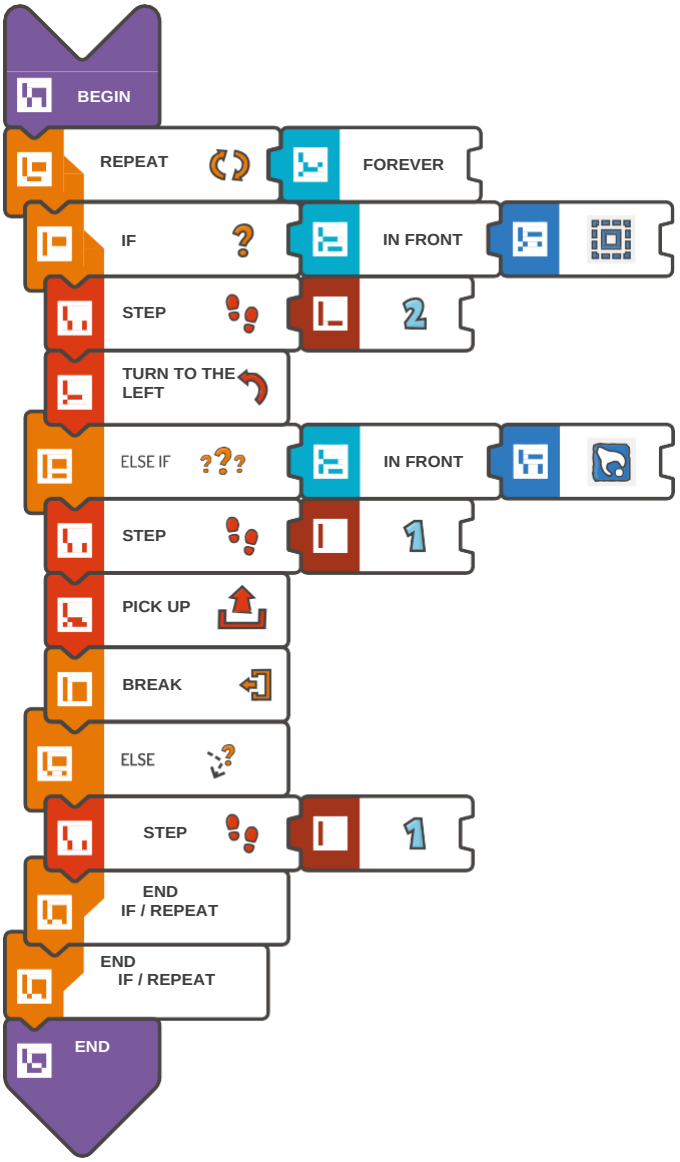
解答例（テキスト形式）：

```
BEGIN
    REPEAT (FOREVER) {
        IF (IN FRONT < MARK -
            SQUARE) { STEP (1)
            TURN TO THE RIGHT
        } ELSE IF (IN FRONT < COLLECTABLE
            OBJECT) { STEP (1)
            PICK UP
            BREAK
        } ELSE {
            STEP (1)
        }
    }
END
```

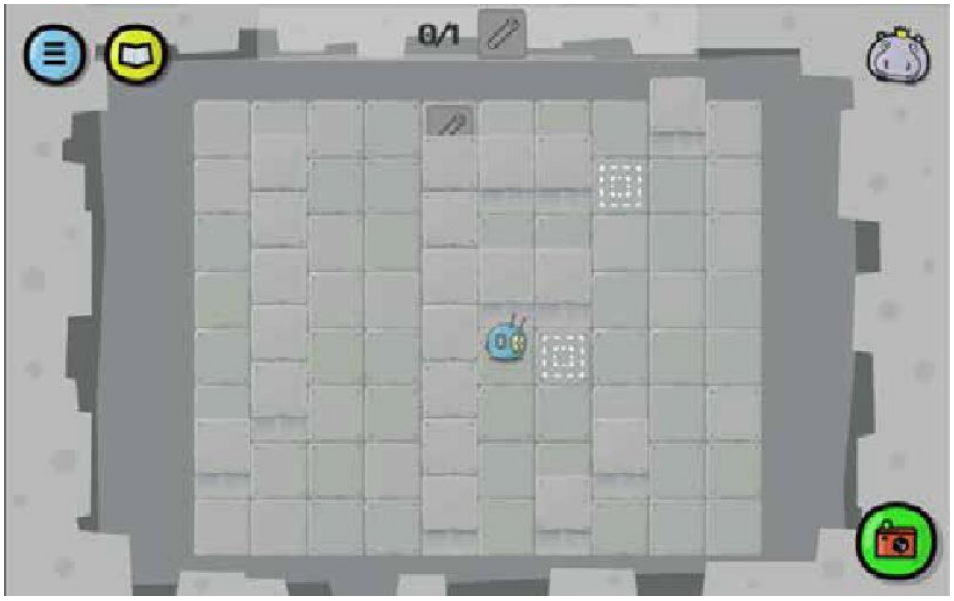
挑戦してみよう（このクエストでは、複合条件文を使いましょ



解答例:



挑戦してみよう（このクエストでは、複合条件文を使いましょ



解答例（テキスト形式）：

```
BEGIN
    REPEAT (FOREVER) {
        IF (IN FRONT < MARK -
            SQUARE) { STEP (2)
            TURN TO THE LEFT
        } ELSE IF (IN FRONT < COLLECTABLE
            OBJECT) { STEP (1)
            PICK UP
            BREAK
        } ELSE {
            STEP (1)
        }
    }
END
```

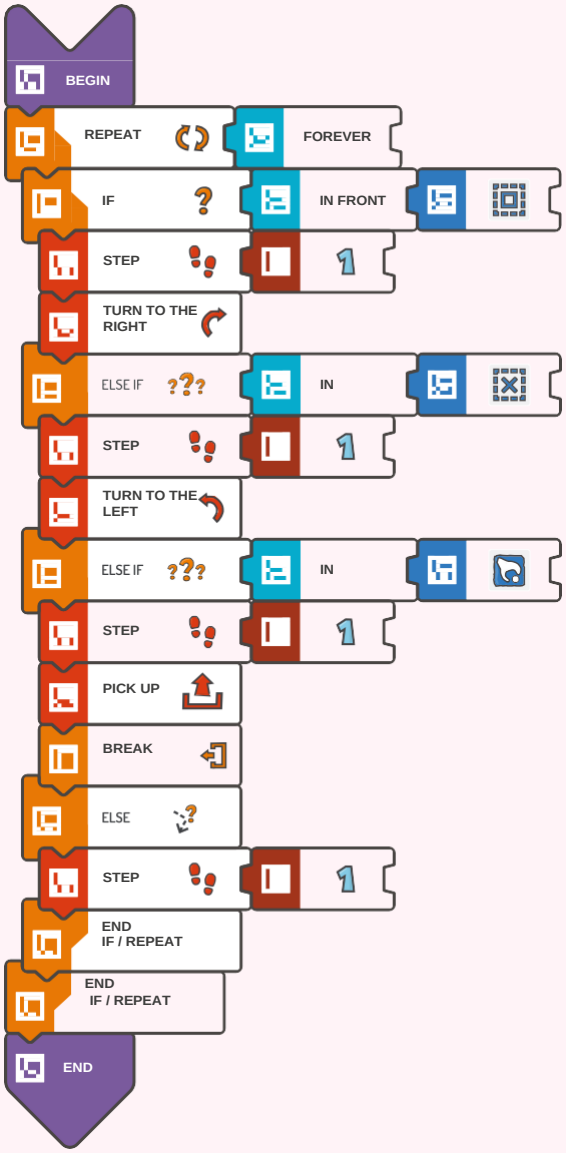
ヒント

コードでの最初の**IF**の後は、複数の**ELSE IF**タイルを使うことができます。

モジュール9 – クエスト5(1/2)

挑戦してみよう（**このクエストでは、条件文を使いましょう**）：

解答例:



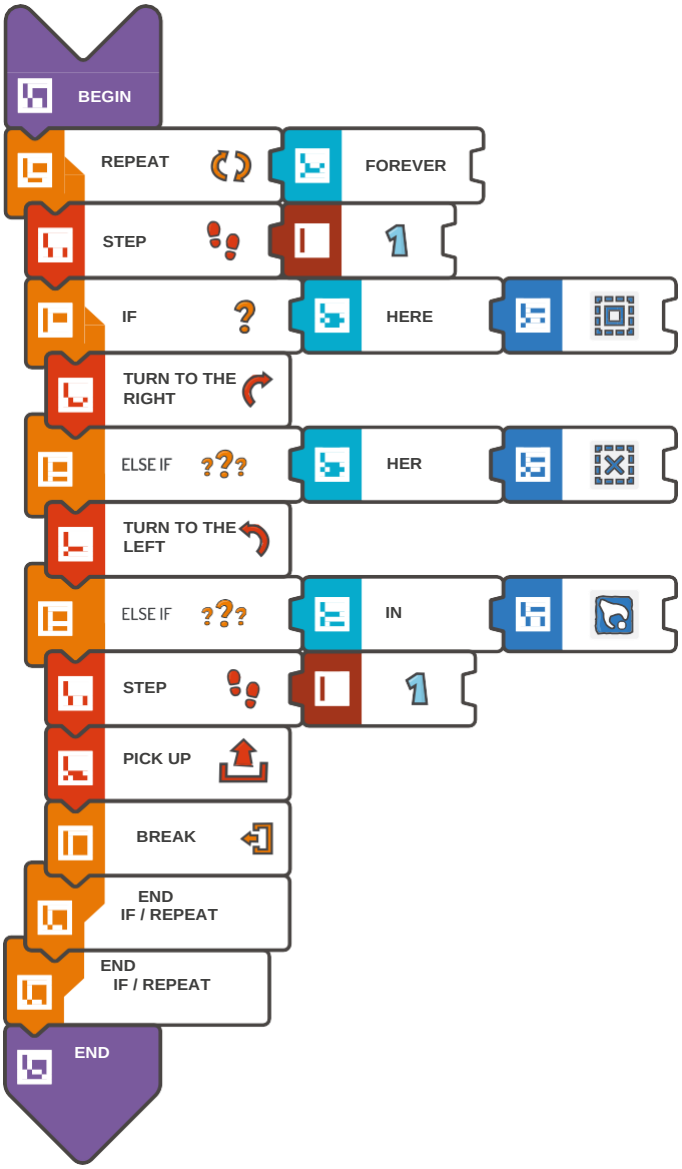
挑戦してみよう（このクエストでは、条件文を使いましょう）：

解答例（テキスト形式）：

```
BEGIN
    REPEAT (FOREVER) {
        IF (IN FRONT < MARK -
            SQUARE) { STEP (1)
            TURN TO THE RIGHT
        } ELSE IF (IN FRONT < MARK -
            CROSS) { STEP (1)
            TURN TO THE LEFT
        } ELSE IF (IN FRONT < COLLECTABLE
            OBJECT) { STEP (1)
            PICK UP
            BREAK
        } ELSE {
            STEP (1)
        }
    }
END
```

挑戦してみよう（このクエストでは、条件文を使いましょう）：

解答例:



挑戦してみよう（このクエストでは、条件文を使いましょう）：

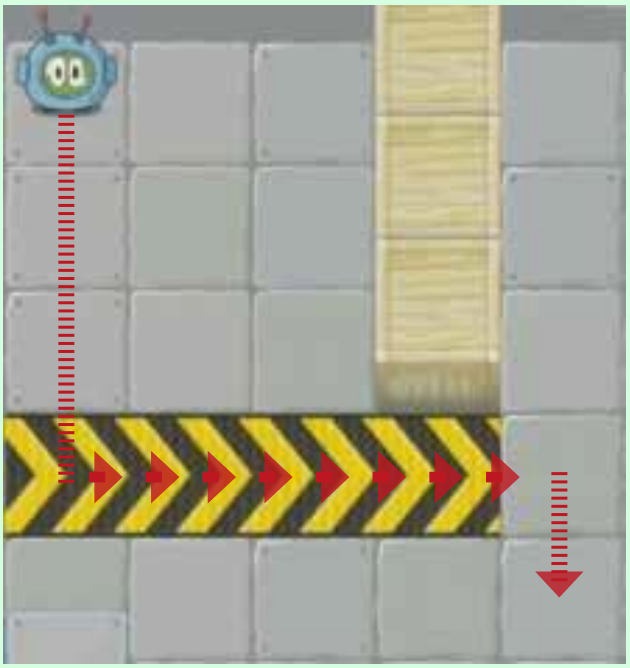
解答例（テキスト形式）：

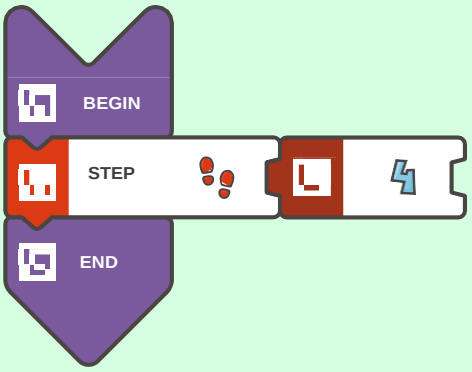
```
BEGIN
    REPEAT
        (FOREVER)
        { STEP (1)
        IF (HERE < MARK -
            SQUARE) { TURN TO
            THE RIGHT
        } ELSE IF
        (HERE < MARK - CROSS)
            { TURN TO THE LEFT
        } ELSE IF
        (IN FRONT < COLLECTABLE
            OBJECT) { STOP(1)
            PICK UP
            BREAK
        }
    }
END
```

導入:
コンベヤーベルト

この作業場には、**コンベヤーベルト**があります。これを使うと、複雑なプログラムを作る必要なしに、スコッティはボード中を動き回ることができます。

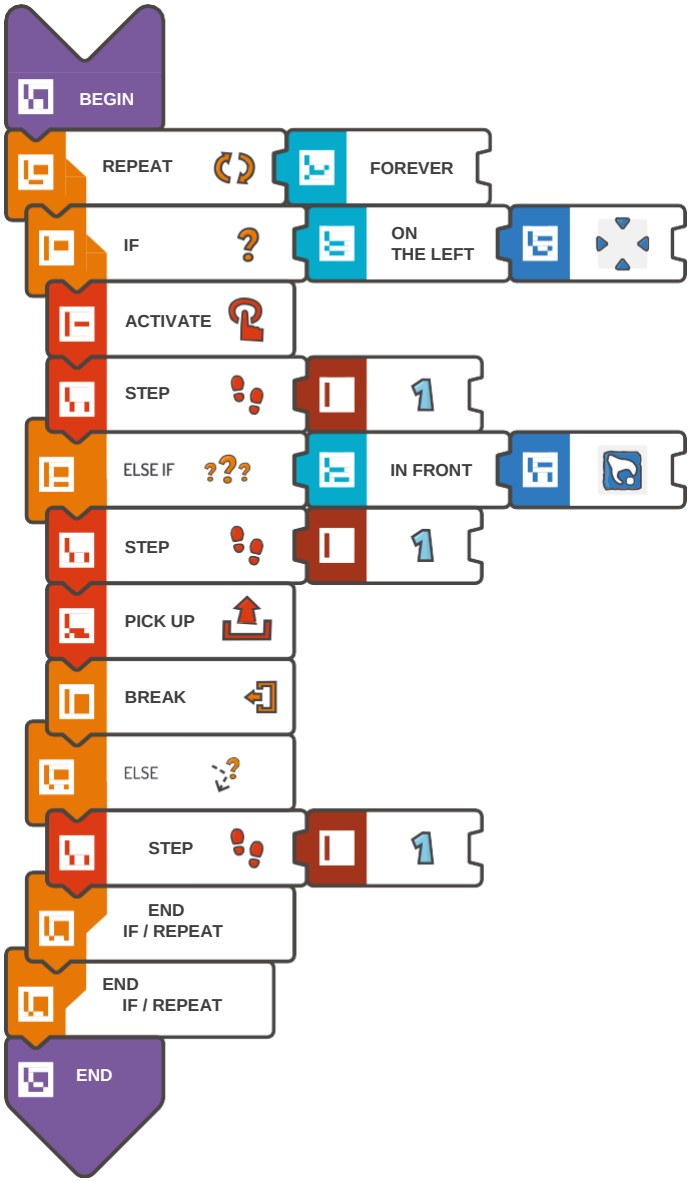
コンベヤーベルトに入ると、**プログラムの実行が一時停止し、スコッティが目的地に着いた後、再開します。**





挑戦してみよう（位置を変更するには、コンベヤーベルトを使いましょう。スイッチを入れるには、**ACTIVATE**タイルをいます）：

解答例:



挑戦してみよう（位置を変更するには、コンベヤーベルトを使いましょう。スイッチを入れるに

解答例（テキスト形式）：

```
BEGIN
    REPEAT (FOREVER) {
        IF (ON THE LEFT < ACTION SQUARE)
            { ACTIVATE
            STEP (1)
        } ELSE IF (IN FRONT < COLLECTABLE
            OBJECT) { STEP (1)
            PICK UP
            BREAK
        } ELSE {
            STEP (1)
        }
    }
END
```

導入:
スイッチ

いくつかのコンベヤーベルトのスイッチが切れています。スイッチを入れて、作業場のスイッチ

を探し、近くに行って、**ACTIVATE**コマンドを使います。

覚えておこう!

ACTIVATEコマンドを実行させるには、スコッティがスイッチに背を向けていてはいけません。

挑戦してみよう（このクエストでは、条件文を使いましょう）：

解答例:

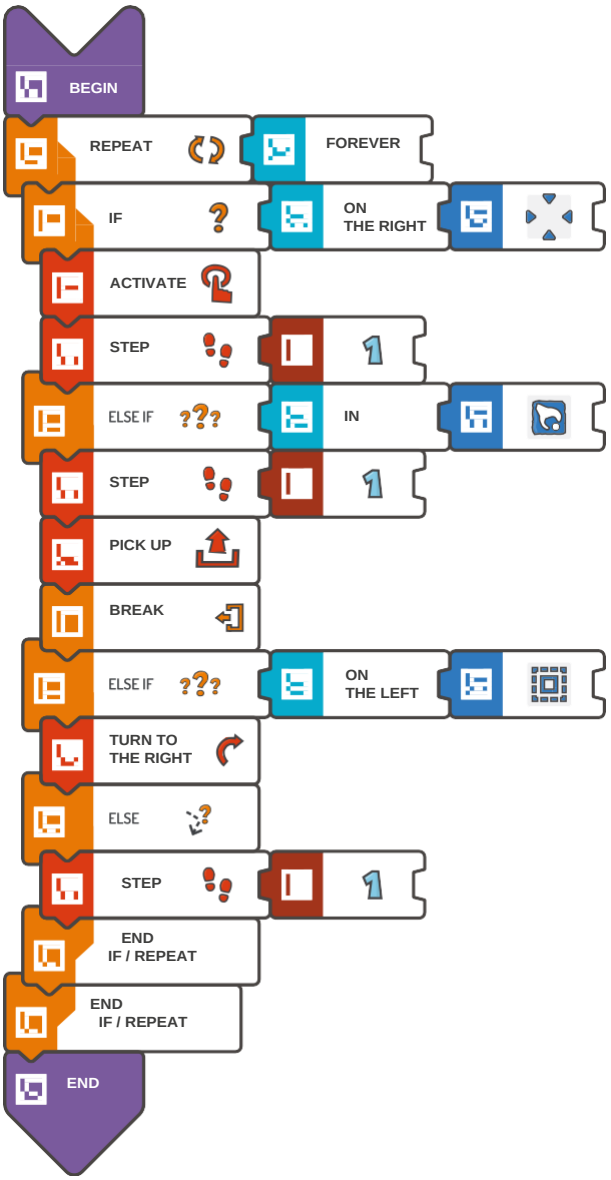
挑戦してみよう（このクエストでは、条件文を使いましょ

解答例（テキスト形式）：

```
BEGIN
    REPEAT (FOREVER) {
        IF (ON THE RIGHT < ACTION
            SQUARE) { ACTIVATE
            STEP (1)
        } ELSE IF (IN FRONT < COLLECTABLE
            OBJECT) { STEP (1)
            PICK UP
        } ELSE IF (IN FRONT <
            OBSTACLE) { BREAK
        } ELSE {
            STEP (1)
        }
    }
END
```

挑戦してみよう（このクエストでは、条件文を使いましょ

解答例:



挑戦してみよう（このクエストでは、条件文を使いまし

解答例（テキスト形式）：

```
BEGIN
    REPEAT (FOREVER) {
        IF (ON THE RIGHT < ACTION
            SQUARE) { ACTIVATE
            STEP (1)
        } ELSE IF (IN FRONT < COLLECTABLE
            OBJECT) { STEP (1)
            PICK UP
            BREAK
        } ELSE IF (ON THE LEFT < MARK -
            SQUARE) { TURN TO THE RIGHT
        } ELSE {
            STEP (1)
        }
    }
END
```

モジュール10
アフリカ

関数の定義。

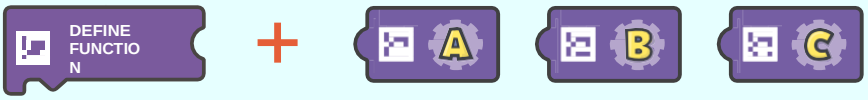
スコッティは宇宙船の修理を終えて、動物仲間と約束した仕事を完了したいと思っています。公園に足りない苗木を植え、小道の位置を示す線を引きましょう。作業の間、必ず関数を使いましょう!

モジュール導入 1/4:
関数

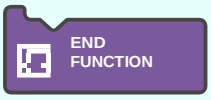
通常、プログラムを書くときは、何度も同じ一連のコマンドを使用します。コマンドを何度も並べるのは時間の無駄であり、変数で使ったのと同じような解答を見つける方がよいことに、すぐに気が付きます。一連のコマンドを特別な容器に入れて、必要な時に呼び出しましょう。こうすることで、プログラムをもっとシンプルで短く、明確にできます。

この容器は**関数**（サブプログラム）と呼ばれ、これを使うには、特定の一連のタイルを使ってアプリケーションで定義し、名前を付けます。

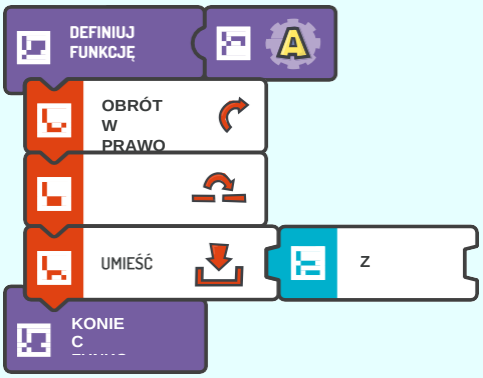
関数を作成するには、**DEFINE FUNCTION**タイルを使用し、この関数に**名前**を付ける必要があります: **A**、**B**または**C**（スコッティ・ゴー!では、1度に3つの関数まで定義できます）：



END FUNCTIONタイルも必要です。



DEFINE FUNCTIONタイルと**END FUNCTION**タイルの間に、プログラムで繰り返し



使うコマンド列を置きます。例えば:

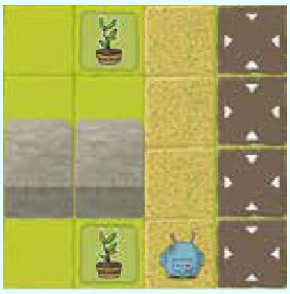
関数をスキャンすると、関数名（**A**、**B**または**C**）の付いた**CALL FUNCTION**タイルを使ってプログラムで使用できます。この場合:

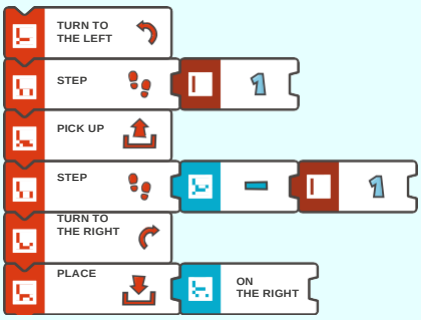
モジュール導入 2/4:
関数

例

下に示すクエストでは、**スコッティは苗木を集め、**（土が用意されている）指定された場所に**植えることになっています。お分かりのように、このタスクは、それぞれを2回行わなければいけません。**この例で、関数の作成や使用について復習しましょう。

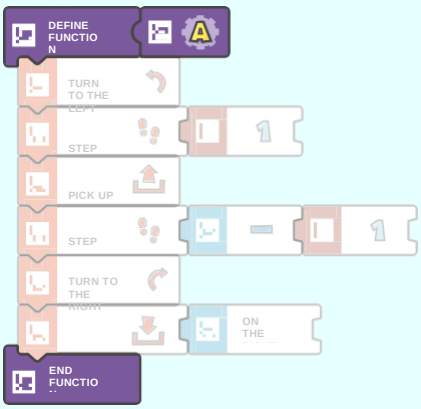
1. まず、苗木を集めて、指定した場所に植えることに関与する**指示列を並べます。**



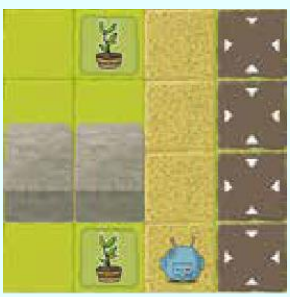


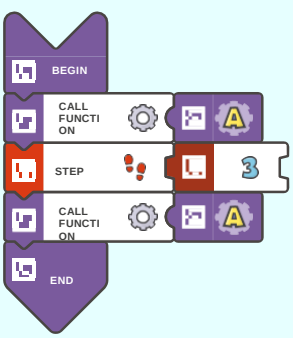
2. それから、前にアセンブルした一連の指示を使って、**関数を定義します。**

A、BまたはCという名前を付けましょう。



3. 必要な場合には**名前で関数を呼び出せるプログラムを作成します。**





モジュール導入 3/4:
関数[追加]

関数の中に別の関数を入れることができます。

関数を使うことで、非常に複雑なプログラムを作成できます:

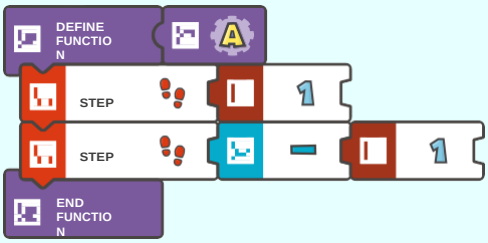
1. まず、使用するタイルが少なくなります（関数のおかげで、コードはもっと短く、管理しやすくなります）。

2. 次に、スコッティが行うタスクにもっと焦点をあてることができます。**関数の概念を取り入れると、抽象度を上げます。これは、靴ひもを結ぶというようなタスクを身に付ける場合などに、日常的に起こっています。**

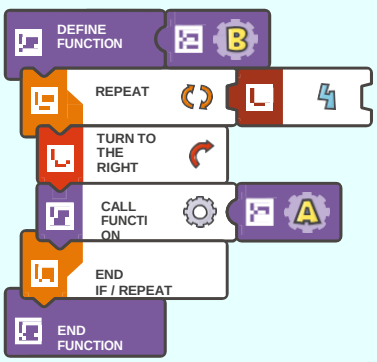
例えば、「靴ひもを結ぶ」方法を身に付ける過程を考慮すると、正確な順番で靴ひもを巧みに扱うことを含む、一連の複雑な動作で成り立っているにも関わらず、1つのタスクとしてとらえています。

以下の場合、スコッティはどの道を通るでしょうか:

関数Aを呼び出す?



関数Bを呼び出す?

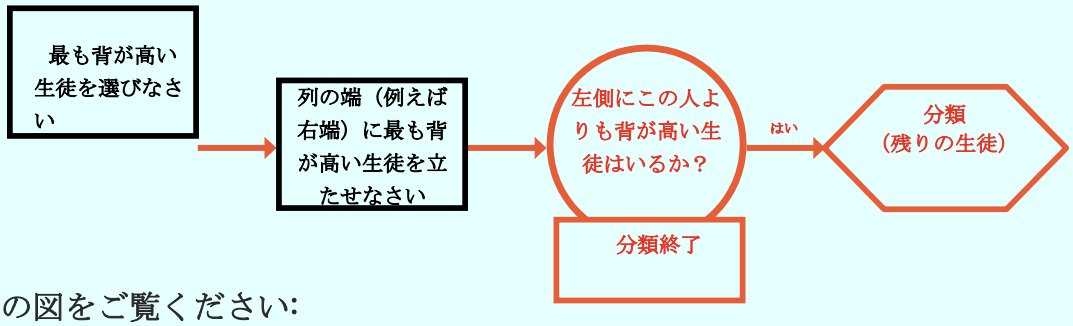


関数Cを呼び出す?

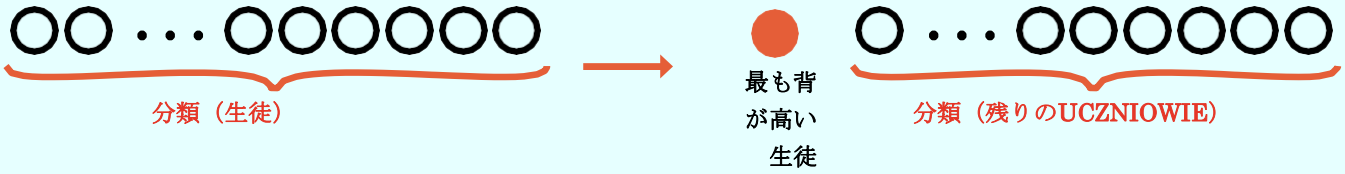
モジュール導入 4/4:
関数[追加]

関数は関数自身を呼び出すことができます!

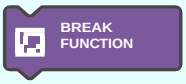
これを**再帰関数**と呼びます。再帰関数は、例えば、アルゴリズムの分類に使われます。つまり、大きさやその他のカギとなる物の順番に、一連の値を付けるために使うアルゴリズムです（例えば、本をタイトルで分類したい場合など）。生徒を身長で分類するために使うアルゴリズムを見てみましょう:



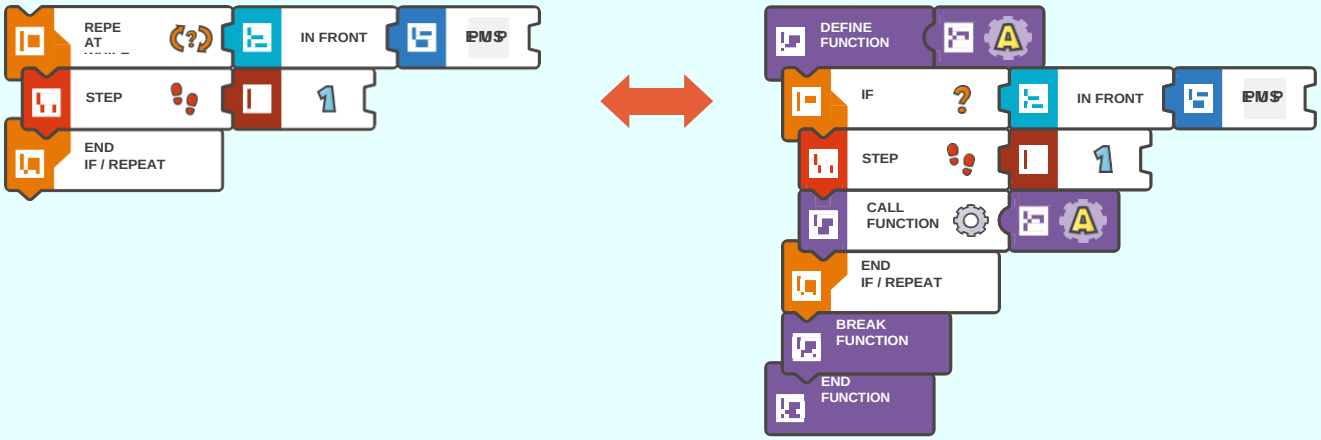
またはこの図をご覧ください:



再帰関数の重要な要素とは、条件が事実である場合に関数を破壊する条件文です。これがないと、関数は決して破壊されず、エラーが生じます（あるいは、メモリーがなくなるまで、コンピューターをブロックします）。関数を停止するには、**BREAK FUNCTION**タイルが必要です。



再帰関数を使うと、自分の条件付きループを作成できます:

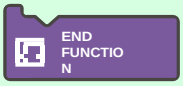


導入:
最初の関数、隆起したスクエア(1/2)

このクエストでは、**CALL FUNCTION**タイルを**A**タイル（Aは関数名を表します）と一緒に使って、独自の関数Aを呼び出します。

しかし、まずは、関数が呼び出された際に実行されるよう、コマンド列を作成してこの関数を定義する必要があります。**DEFINE FUNCTION**タイルを**A**タイル（Aは関数名を表します）や、

最後に置かれた**END FUNCTION**タイルと一緒に使います。



例えば:

テキスト形式では:

```
DEFINE
        FUNCTION A
        { STEP (2)
}
```

覚えておこう!

ステップ1 – 関数を記録する（3つの関数: A、B、Cまで）

ステップ2 - 関数をスキャンする（3つの関数: A、B、Cまで）

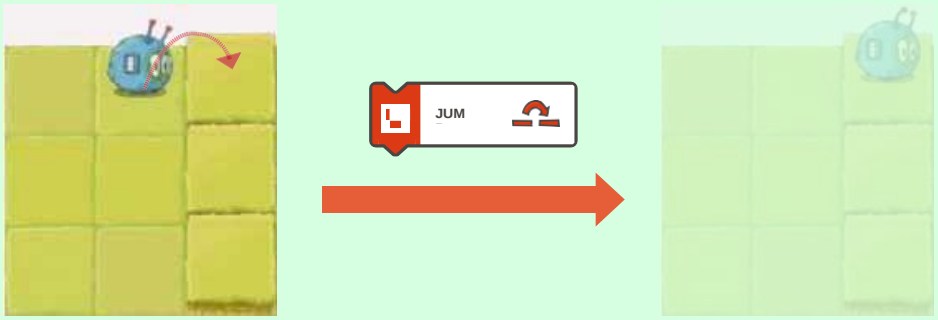
ステップ3 - 関数を呼び出すプログラムを書く

ステップ4 - プログラムをスキャンして実行する

モジュール10の導入で関数についてさらに学びましょう。

導入:
最初の関数、隆起したスクエア(2/2)

このモジュールのいくつかのクエストでは、できるだけ少ない行で、スコッティに隆起したスクエアを飛び越えさせて、クエストを解決したいと思うかもしれません。

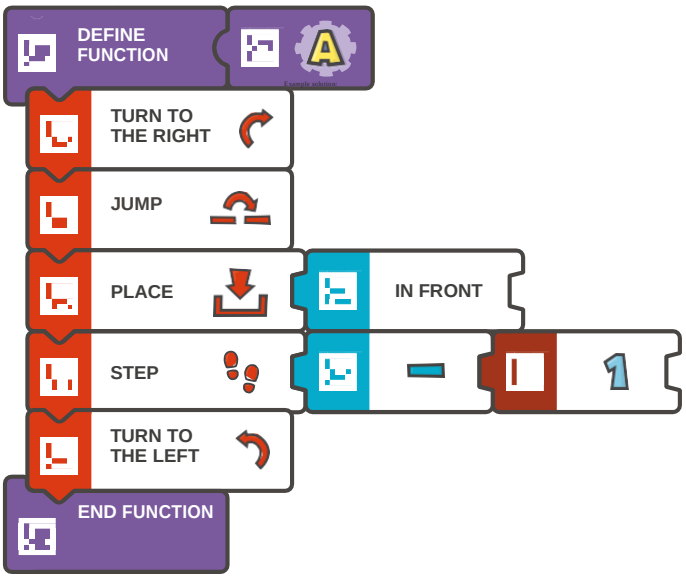


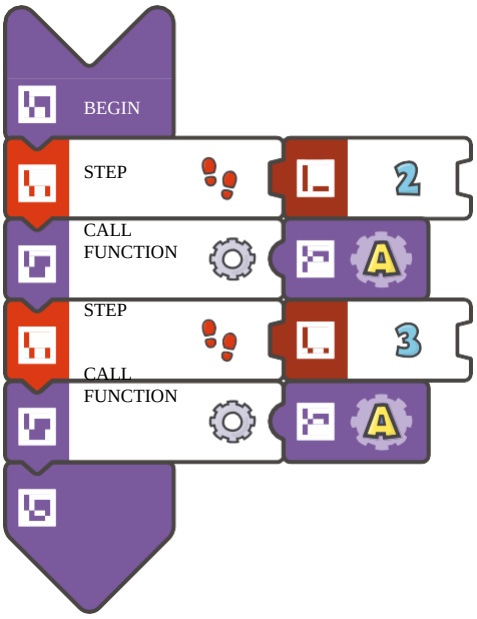
隆起したスクエアにたどり着く唯一の方法は、**JUMP**タイルを使うことです。降りるには、**JUMP**または**STEP**タイルを使います。

ELAVATED SQUAREタイルを使って、コードに隆起したスクエアを入れることもできます。

挑戦してみよう（**独自の関数を作成して呼び出し、アクションスクエアに苗木を植えましょう**）：

解答例:





挑戦してみよう（独自の関数を作成して呼び出し、アクションスクエアに苗木を植えましょ

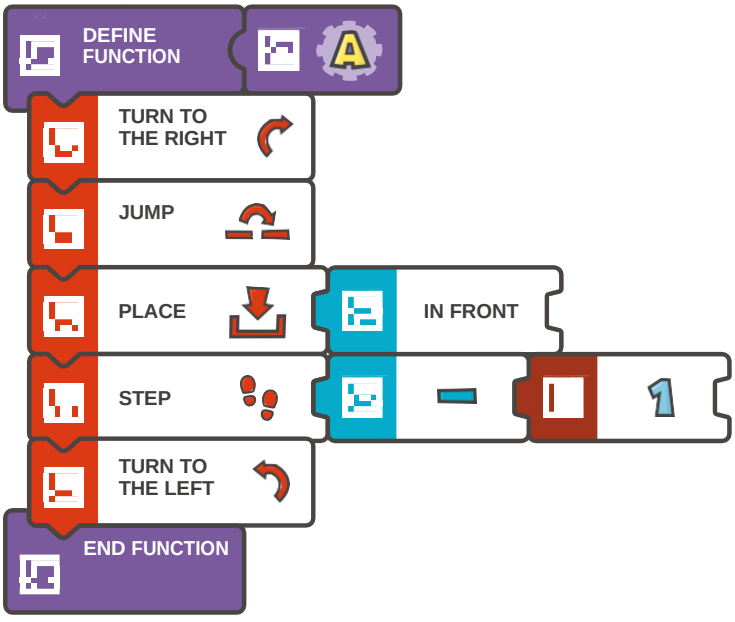
解答例（テキスト形式）：

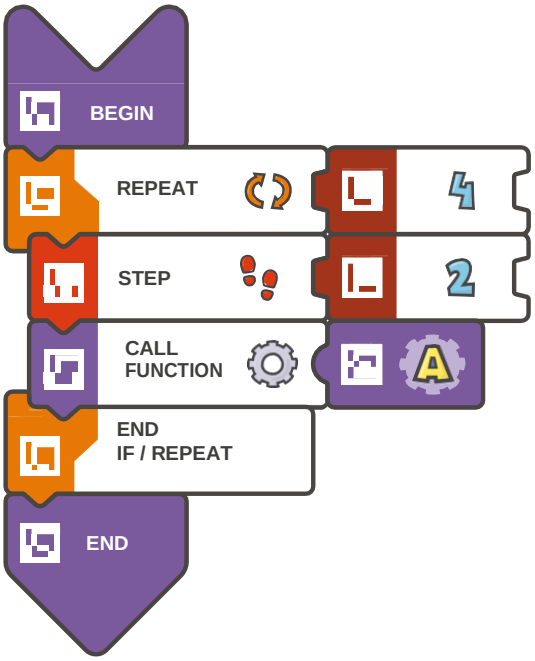
```
DEFINE FUNCTION A
        { TURN TO THE
        RIGHT JUMP
        PLACE (IN FRONT)
        STEP (-1)
        TURN TO THE LEFT
}

BEGIN
        STEP (2)
        CALL FUNCTION
        A STEP (3)
        CALL FUNCTION A
END
```

挑戦してみよう

（REPEATループを使って独自の関数を4回呼び出しましょう）：

解答例:





挑戦してみよう

（REPEATループを使って独自の関数を4回呼び出しましょ

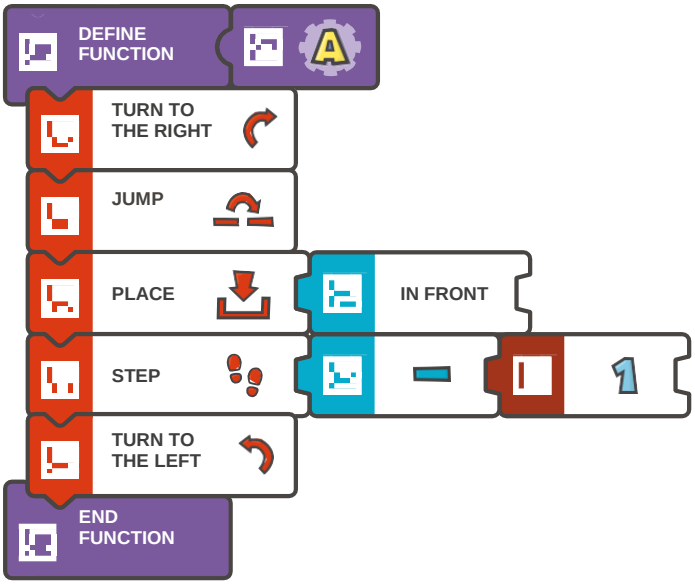
解答例（テキスト形式）：

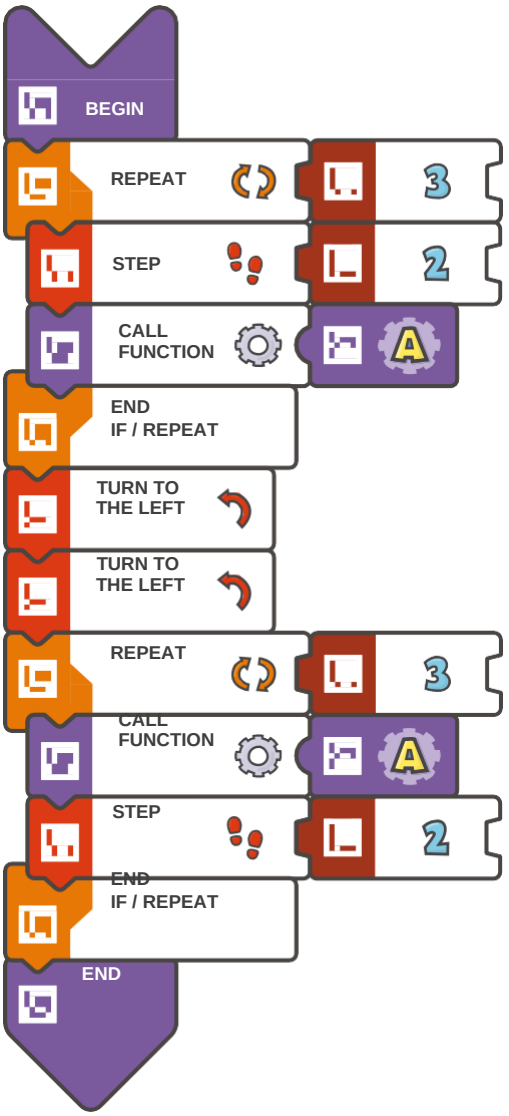
```
DEFINE FUNCTION A
        { TURN TO THE
        RIGHT JUMP
        PLACE (IN FRONT)
        STEP (-1)
        TURN TO THE LEFT
}

BEGIN
        REPEAT (4) {
                STEP (2)
                CALL FUNCTION A
        }
END
```

挑戦してみよう:


0/6
6

解答例:


DEFINE FUNCTION
A
TURN TO THE RIGHT
JUMP
PLACE
IN FRONT
STEP
1
TURN TO THE LEFT
END FUNCTION


BEGIN
REPEAT
3
STEP
2
CALL FUNCTION
A
END IF / REPEAT
TURN TO THE LEFT
TURN TO THE LEFT
REPEAT
3
CALL FUNCTION
A
STEP
2
END IF / REPEAT
END

挑戦してみよう:

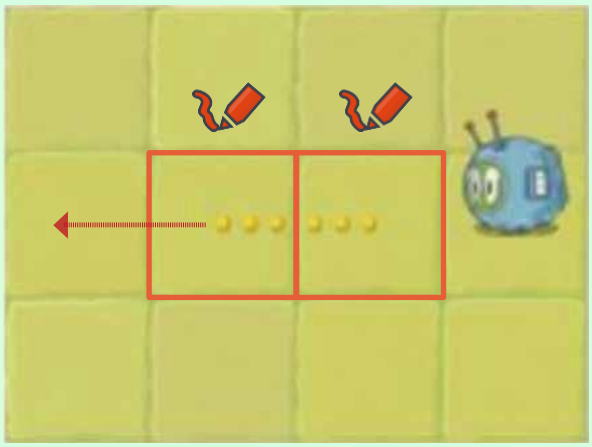
解答例（テキスト形式）：

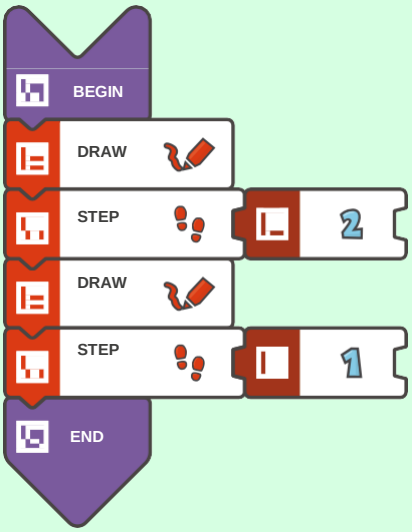
```
DEFINE FUNCTION A
        { TURN TO THE
        RIGHT JUMP
        PLACE (IN
        FRONT) STEP (-1)
        TURN TO THE
        LEFT
}

BEGIN
        REPEAT (3) {
                STEP (2)
                CALL FUNCTION A
        }
        TURN   TO   THE
        LEFT   TURN   TO
        THE          LEFT
        REPEAT (3) {
                CALL FUNCTION A
                STEP (2)
        }
END
```

導入:
線を引く

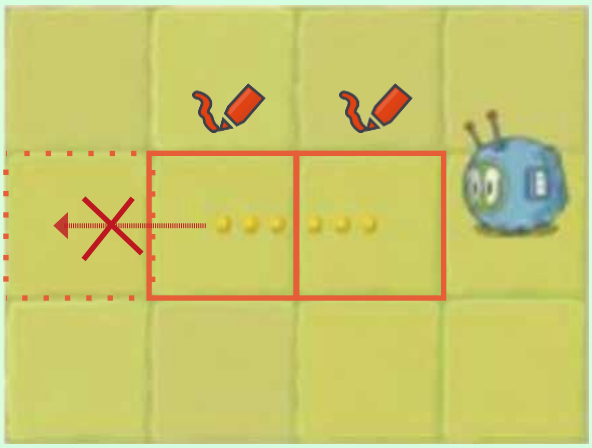
公園に小道を作る計画があります。この小道のルートの印となる線を引くタスクがスコッティに与えられています。

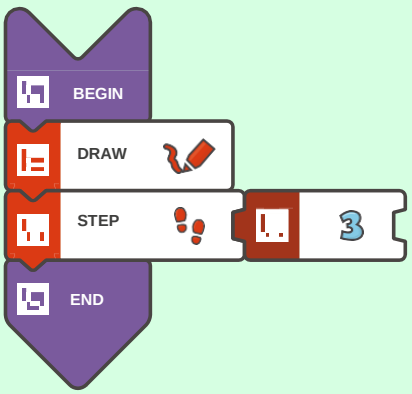
このタスクを完了するには、**DRAW**タイルを使います。**線を引き始めるときも、引き終わるときも、このタイルを使います。**線を引くモードになっていれば、**スコッティは入るスクエア、または通り抜けるスクエアに線を残します。**





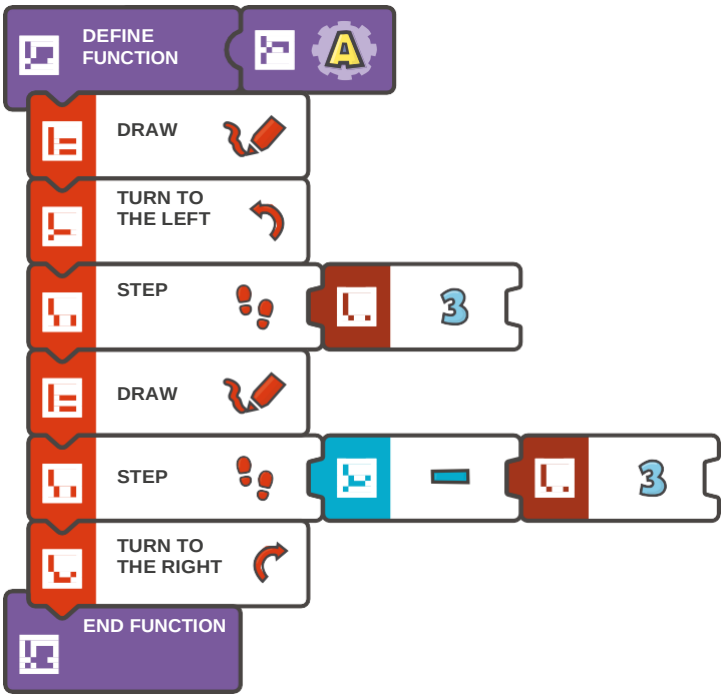
ドットなしでスクエアに線を引くと、エラーとみなされます。

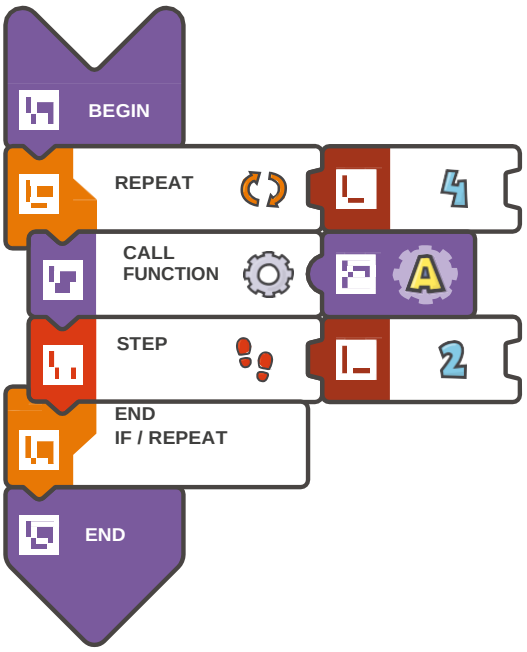




挑戦してみよう（**独自の関数Aを作成し、その中にDRAWタイルを入れましょう**）：

解答例:





挑戦してみよう（**独自の関数Aを作成し、その中にDRAWタイルを入れましょ**

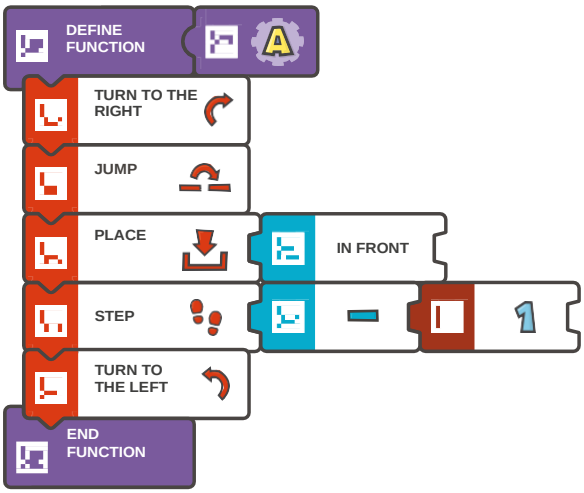
解答例（テキスト形式）：

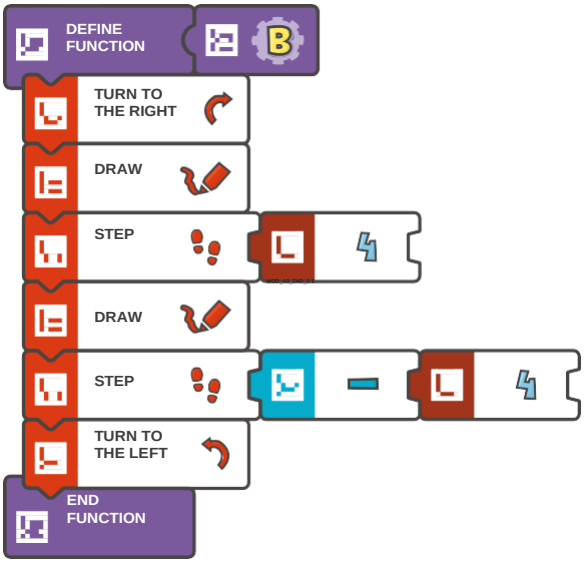
```
DEFINE FUNCTION A
        { DRAW
        TURN TO THE
        LEFT STEP (3)
        DRAW
        STEP (-3)
        TURN TO THE RIGHT
}

BEGIN
        REPEAT (4) {
                CALL FUNCTION A
                STEP (2)
        }
END
```

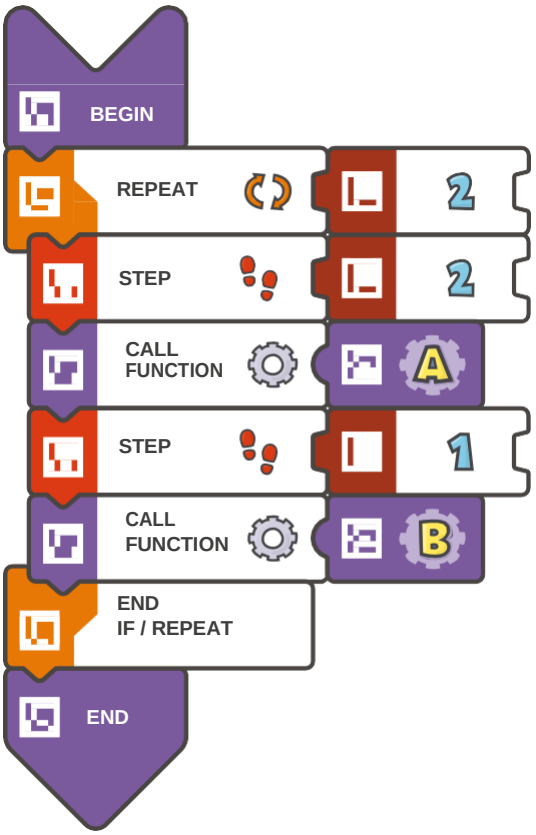
挑戦してみよう

（2つの独自の関数A、Bを作成し、コードで使いましょう）：

解答例







挑戦してみよう

（2つの独自の関数A、Bを作成し、コードで使いましょう）：

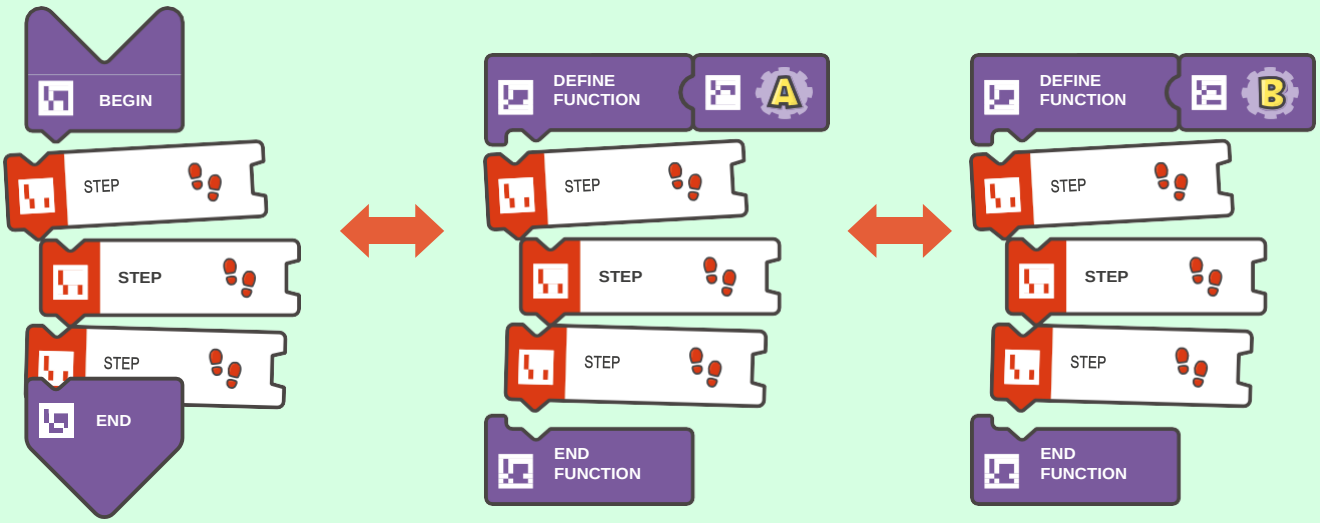
解答例（テキスト形式）：

```
DEFINE FUNCTION A
    { TURN TO THE
    RIGHT JUMP
    PLACE (IN
    FRONT) STEP (-1)
    TURN TO THE
    LEFT
}

DEFINE FUNCTION B
    { TURN TO THE
    RIGHT DRAW
    STEP (4)
    DRAW
    STEP (-4)
    TURN TO THE
    LEFT
}
```

```
BEGIN
    REPEAT (2) {
        STEP (2)
        CALL FUNCTION A
        STEP (1)
        CALL FUNCTION B
    }
END
```

導入
タイルを何度も使う

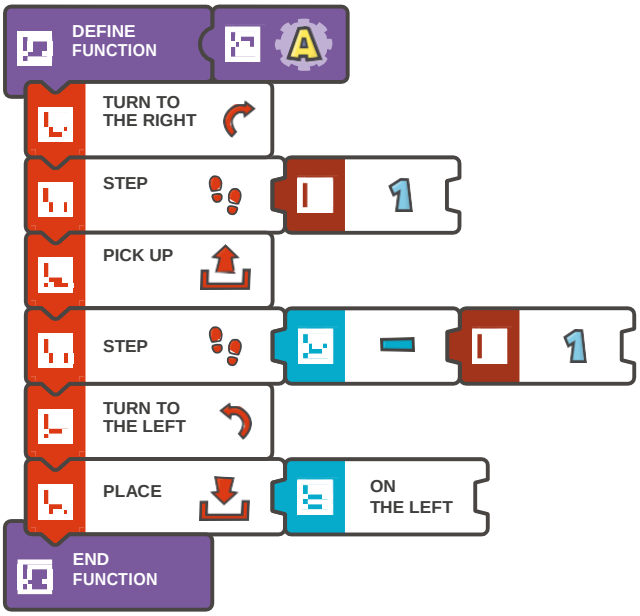
セットのタイル数は限られていますが、プログラムで様々な関数を定義する際、同じタイルが使えます。**つまり、タイルと関数、プログラムをスキャンする際に、差し替えることができます。**

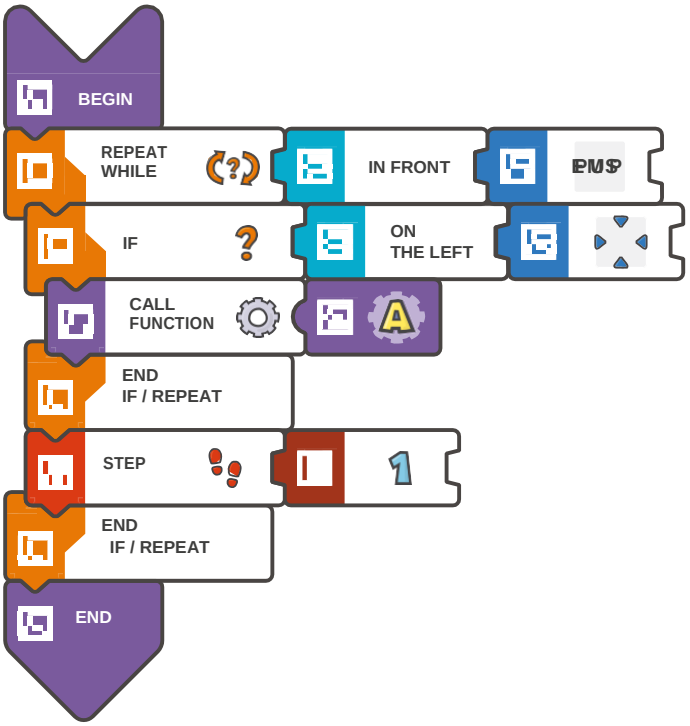


挑戦してみよう

（このクエストでは、**REPEAT WHILEタイルとIFタイルを使いましょう**）：

解答例:





挑戦してみよう

（このクエストでは、**REPEAT WHILEタイルとIFタ**

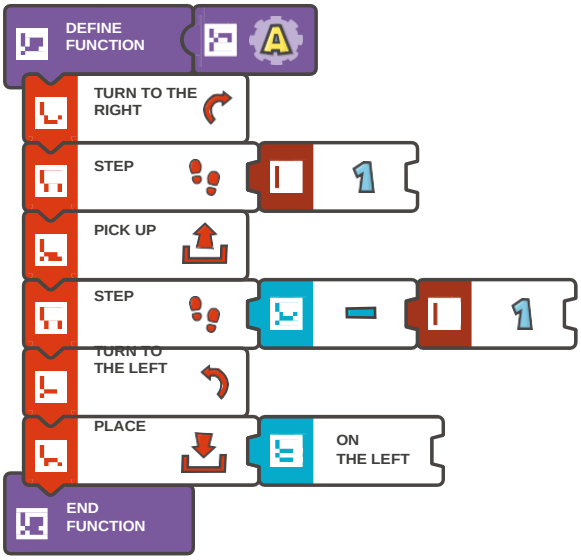
解答例（テキスト形式）：

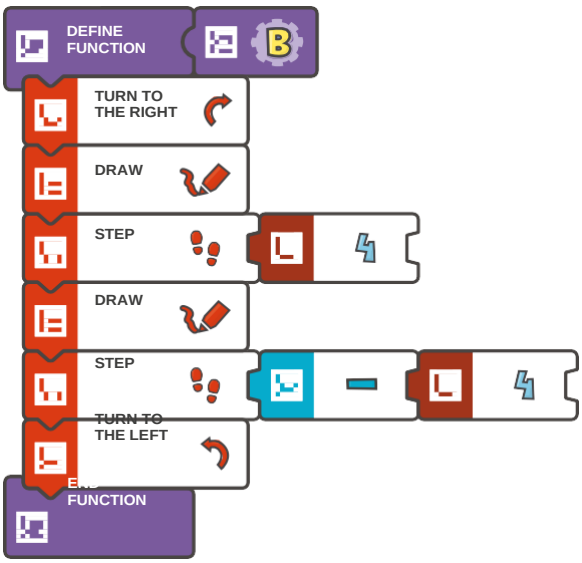
```
DEFINE FUNCTION A
      { TURN TO THE
      RIGHT STEP (1)
      PICK UP
      STEP (-1)
      TURN TO THE LEFT
      PLACE (ON THE
      LEFT)
}

BEGIN
      REPEAT WHILE (IN FRONT < EMPTY) {
            IF (ON THE LEFT < ACTION SQUARE)
                  { CALL FUNCTION A
            }
            STEP (1)
      }
END
```

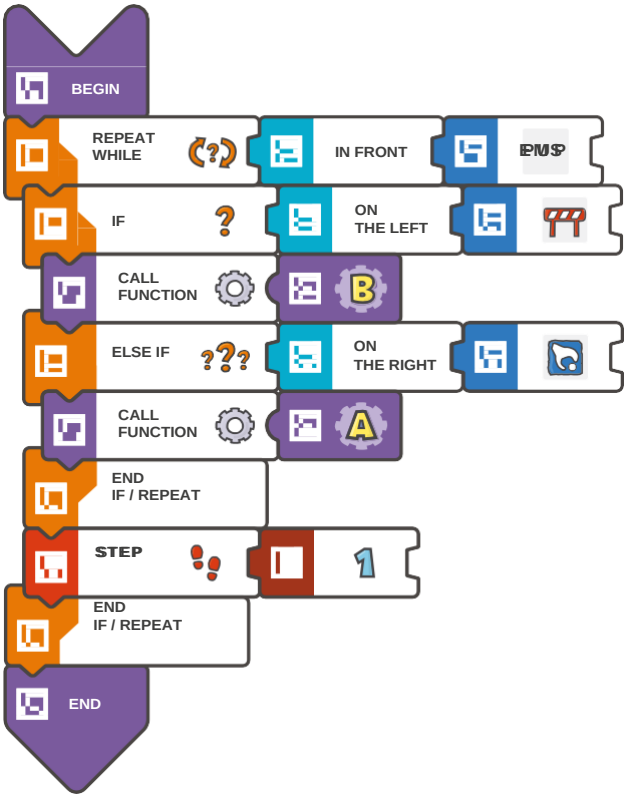
挑戦してみよう:


0/3
0/12

解答例:


DEFINE FUNCTION
A
TURN TO THE RIGHT
STEP
1
PICK UP
STEP
1
TURN TO THE LEFT
PLACE
ON THE LEFT
END FUNCTION


DEFINE FUNCTION
B
TURN TO THE RIGHT
DRAW
STEP
4
DRAW
STEP
4
TURN TO THE LEFT
FUNCTION


BEGIN
REPEAT WHILE
IN FRONT
IF
ON THE LEFT
CALL FUNCTION
B
ELSE IF
ON THE RIGHT
CALL FUNCTION
A
END IF / REPEAT
STEP
1
END IF / REPEAT
END

挑戦してみよう:

解答例（テキスト形式）：

```
DEFINE FUNCTION A
        { TURN TO THE
        RIGHT STEP (1)
        PICK UP
        STEP (-1)
        TURN TO THE
        LEFT PLACE (ON
        THE LEFT)
}

DEFINE FUNCTION B
        { TURN TO THE
        RIGHT DRAW
        STEP (4)
        DRAW
        STEP (-4)
        TURN TO THE
        LEFT
}
```

```
BEGIN
        REPEAT WHILE (IN FRONT <
                EMPTY) { IF (ON THE LEFT <
                OBSTACLE) {
                        CALL FUNCTION B
                } ELSE IF
                (ON THE RIGHT < COLLECTABLE
                        OBJECT) { CALL FUNCTION A
                }
                STEP (1)
        }
END
```

導入:
関数の管理

スコッティ・ゴー!では、3つまで関数（**A**、**B**、**C**）を定義して、1つのプログラムで使うことができます。関数をスキャンすると、画面左端に、そのアイコンが現れます。

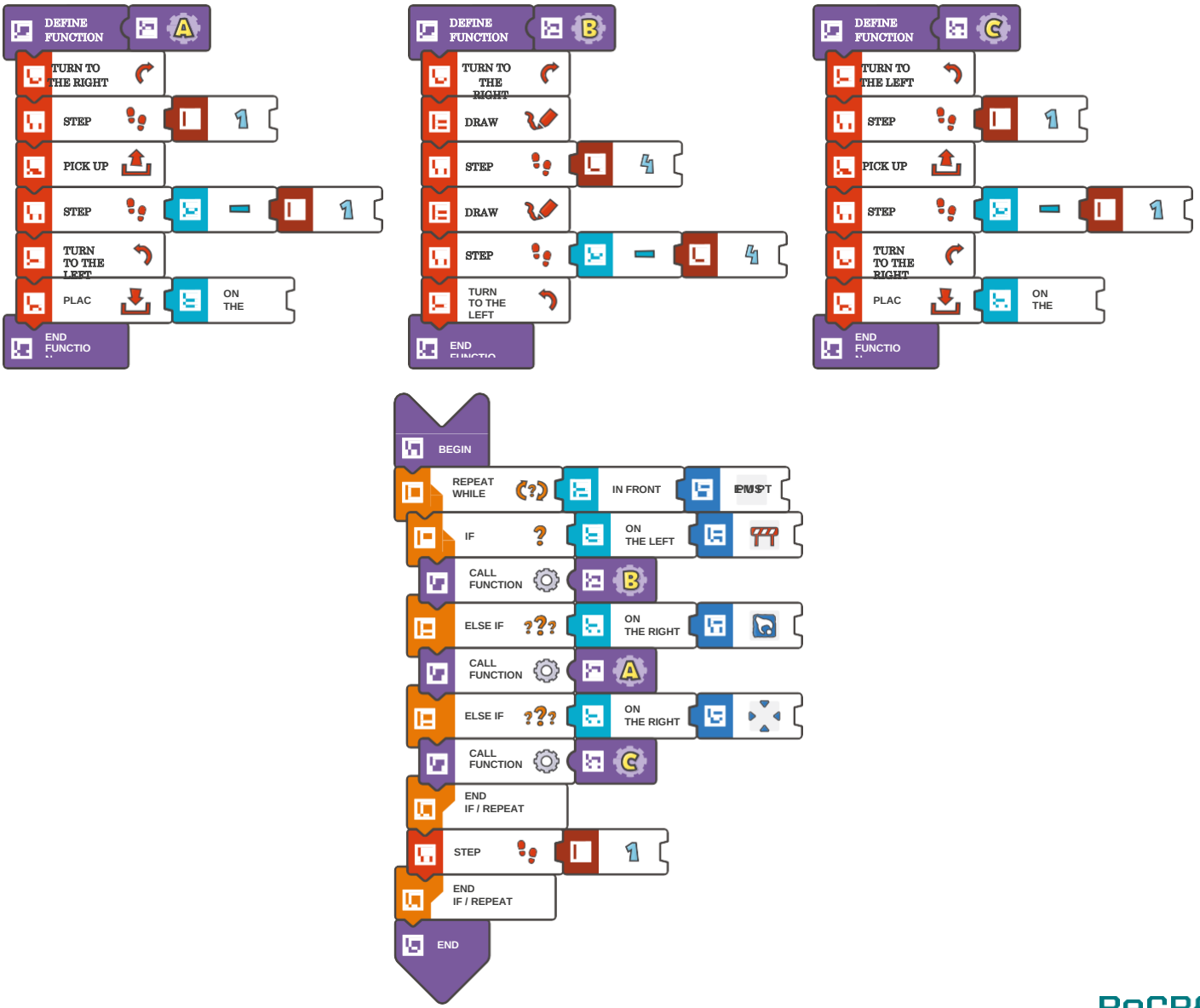
まだ定義されていない関数は灰色になって使えません。定義済みの関数名をタップして、画面に定義を表示します。

定義済みの関数は、機器のメモリーからいつでも削除できます。名前（例えばA）の付いた関数の新しい定義をスキャンすると、以前メモリーに保存した定義と入れ替わります。

挑戦してみよう:


0/3
0/12

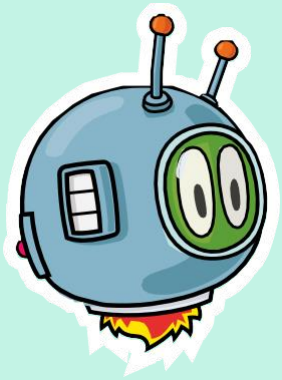
解答例:


DEFINE FUNCTION A
TURN TO THE RIGHT
STEP 1
PICK UP
STEP 1
TURN TO THE LEFT
PLAC
ON THE
END FUNCTIO
DEFINE FUNCTION B
TURN TO THE RIGHT
DRAW
STEP 4
DRAW
STEP 4
TURN TO THE LEFT
END FUNCTIO
DEFINE FUNCTION C
TURN TO THE LEFT
STEP 1
PICK UP
STEP 1
TURN TO THE RIGHT
PLAC
ON THE
END FUNCTIO
BEGIN
REPEAT WHILE
IN FRONT
IF
ON THE LEFT
CALL FUNCTION B
ELSE IF
ON THE RIGHT
CALL FUNCTION A
ELSE IF
ON THE RIGHT
CALL FUNCTION C
END IF / REPEAT
STEP 1
END IF / REPEAT
END

挑戦してみよう:

解答例（テキスト形式）：

```
DEFINE FUNCTION A
        { TURN TO THE
        RIGHT STEP (1)
        PICK UP
        STEP (-1)
        TURN TO THE LEFT
        PLACE (ON THE LEFT)
}

DEFINE FUNCTION B
        { TURN TO THE
        RIGHT DRAW
        STEP (4)
        DRAW
        STEP (-4)
        TURN TO THE LEFT
}

DEFINE FUNCTION C
        { TURN TO THE
        LEFT STEP (1)
        PICK UP
        STEP (-1)
        TURN TO THE RIGHT
        PLACE (ON THE RIGHT)
}

BEGIN
        REPEAT WHILE (IN FRONT < EMPTY)
                { IF (ON THE LEFT <
                OBSTACLE ) {
                        CALL FUNCTION B
                } ELSE IF
                (ON THE RIGHT < COLLECTABLE
                        OBJECT) { CALL FUNCTION A
                } ELSE IF
                (ON THE RIGHT < ACTION SQUARE)
                        { CALL FUNCTION C
                }
                STEP (1)
        }
END
```



ありがとう
スコッティ・ゴーで楽しい時間を過ごしてください!

記載した解答の中には、もっと短くできるものがあります。

スコッティ・ゴー!の楽しさは、まだ始まったばかりです。モジュールでは、すべての文、コマンド、関数が使えることを覚えていてください。

この教師用指導書について何かご意見、ご感想、ご質問がある場合、件名に「**教師用指導書**」（Teachers' Guide）と記載の上、下記の宛先まで、Eメールでご連絡ください。

contact@scottiego.com