aibo さんと ビジュアルプログラミング まなんじゃぉ～ パート2

ご注意

この資料はＳＯＮＹさんのアイボ（ERS-1000）を簡単にプログラミング体験できる「aiboビジュアルプログラミング」の使い方やサンプルなどをアイボオーナ（ハピラキ）が自身のマニュアルのために勝手に作成したものです。この内容についての保証、お問い合わせ、配布、販売などはご遠慮願います。

作成 ２０２２年１０月 時点のものです。

LESSON 4

おもちゃと遊んでみましょう つづき

LESSON3 の音声と画像認識のプログラムどうでしたでしょうか？　きっとイメージ通りに、ナカナカ動作してくれなかった・・や、なんとなくできているかも・・や、正しく判断ができていないのでは？　と思ったオーナさんもいたかと思います。

思った通りにならないと、何が起こっているのだろうか・・・と考えてしまいます。プログラムをつくって、思った通りに動かないとき、いろいろな手段をつかって確認するのが一般的です。

例えば

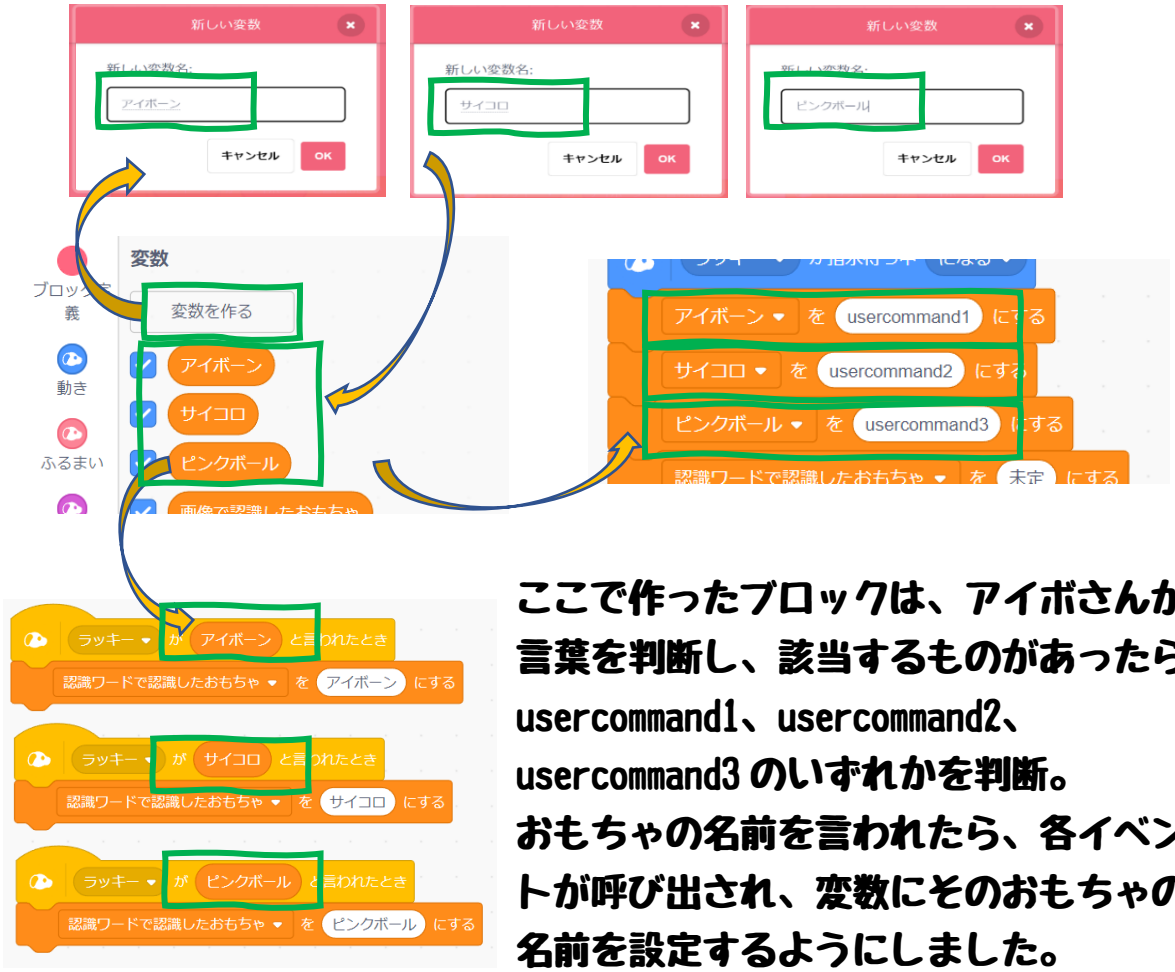
- 設定されている変数の値を確認
- 取得したデータの内容を確認
- 計算した結果や実行結果を確認
- 一時的に値を設定してみて動作を確認

これらを解決するために、一般的な確認方法に「デバッグする」や「トレースする」あるいは「ログ取り」などと呼ばれる手法で確認します。このテキストでも、何度か確認したことがある【変数】をクリックすると、現在の値が表示されることを学んできましたが、その手法も一つとなります。

プログラムを作っていくと必ず経験するので、このLESSON4では、その方法を少しだけ確認してみましょう。

■認識ワードの確認

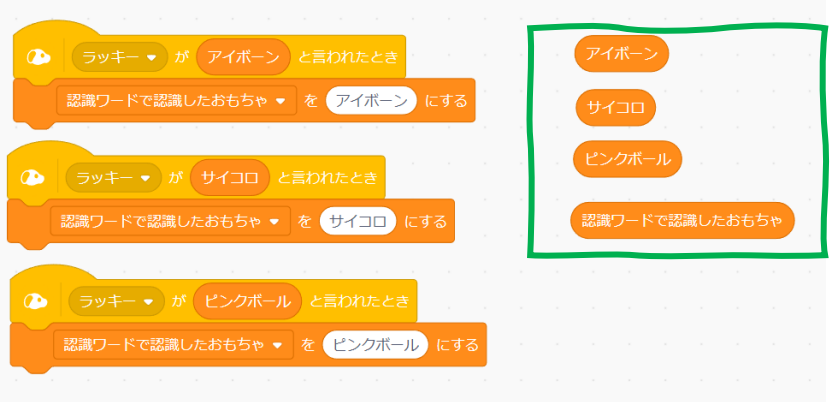
LESSON3で音声の認識ワードしたワードを保存する変数【アイボーン】【サイコロ】【ピンクボール】そして認識できた時に何が言われたかを保存する【認識ワードで認識したおもちゃ】を作って次のようなブロックを作りました。



ここで作ったブロックは、アイボさんが言葉を判断し、該当するものがあったらusercommand1、usercommand2、usercommand3 のいずれかを判断。
おもちゃの名前を言われたら、各イベントが呼び出され、変数にそのおもちゃの名前を設定するようにしました。

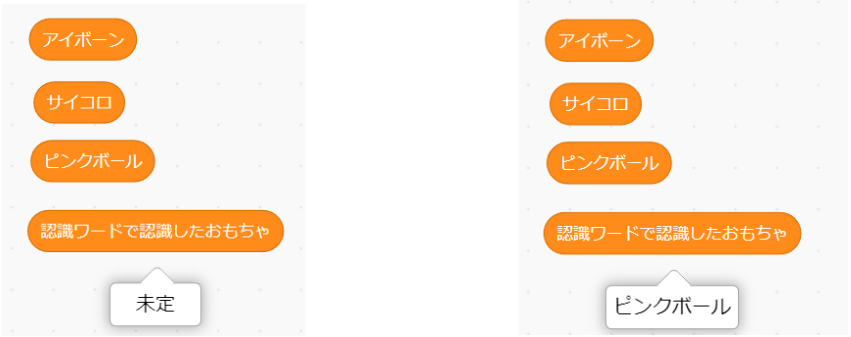
実際に実行したとき変数【認識ワードで認識したおもちゃ】がどのような値が保存されているかを確認してみましょう。

このテキストを見ていただいているオーナさんはすでに何度か実行していますが、以下のように変数を置き、何が設定されているかをクリックすることで、現在の値が確認できます。

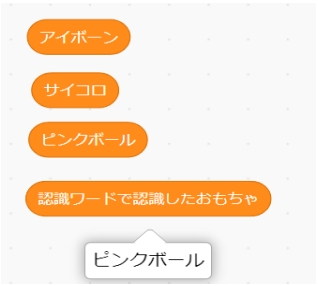


変数【アイボーン】【サイコロ】【ピンクボール】をクリックすると設定した値 usercommand1、usercommand2、usercommand3 が設定されていることが分かりますね。

次に変数【認識ワードで認識したおもちゃ】を確認すると、すでに認識ワードで認識されていれば、その値が設定されているかと思います。ＬＥＳＳＯＮ３のプログラムを実行した状況で、まだ認識ワードが認識できていなければ【未定】のままかと思います。



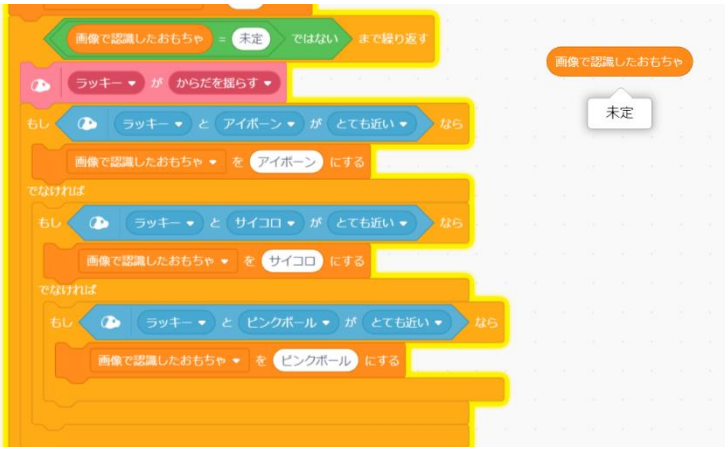
せっかくなので「アイボーン」「サイコロ」「ピンクボール」を言ってみて、この変数がそれぞれ変わっているか確認してみましょう。変わっていれば、正しく認識できていることとなります。



もし、どのキーワードを言っても値が変わらなければ、WEB登録の内容、変数を作った時の「usercommand1」などの文字が全角になっていたり空白が入っていたり、いずれも何か間違っている場所がある可能性あります。

このように実行時の途中の状況を確認することで、正しくプログラムが動作しているかを確認することができます。

■画像認識の確認



同じように、画像についても確認してみましょう。変数【画像で認識したおもちゃ】を横において実行直後にリックしてみてください。最初は、【未定】が表示されているかと思います。

「アイボーン」「さいころ」「ピンクボール」を見せてあげながら、同じように、変数の値がどのようになっているかを確認してみましょう。

どうでしょうか？　うまく変数に設定されましたでしょうか？
アイボさんの鼻先にじっくり見せてあげても「未定」のままの場合は、正しく認識できていない可能性があります。もう一度プログラムを確認してみてください。

LESSON3のプログラムは、この認識ワードで認識できた内容を変数【認識ワードで認識したおもちゃ】と画像で認識したときに保存した内容を変数【画像で認識したおもちゃ】の内容を比較して、その結果で喜んだり悲しんだりするようにしています。

このように、プログラムが正しく動作しているかを確認する方法と

して、途中の状態や値を確認することで、思った通りの動作をしているか、タイミングがあっているかなどを確認しながら、作っていくことで、より確実なプログラムの作成ができるようになります。

このような作業を先ほど説明した「デバッグ」や「バグ取り」などと呼びます。実際にはもっともっと複雑に確認する手段や開発用のツールなどがありますが、ビジュアルプログラミングでは、この方法でしか現状確認できませんが、是非、活用していきましょう。

今回のＬＥＳＳＯＮ４では、プログラミングの方法について学習しておりますので、もう一つ、新しいブロックも一緒に覚えてみましょう。

● ブロック定義をマスターしましょう

ブロックに【ブロック定義】と呼ばれているものがありますが、
【ブロックを作る】ボタン以外ブロックがありませんねぇ・・・

「ブロックを作る」を押してみると、難しい言葉がいっぱい並んでいる何やら新しいブロックを作るようなものがでてきました。

この時点で　チンプンカンプン・・・ですよね・・
きっとこの先も何度も？？が出てきそうな内容ですが、今後のＬＥＳＳＯＮで順次説明しながら使っていきたいと思いますが、つぎのようなことを考えたいと思います。

プログラムを作っていくと、同じような処理や複雑なプログラムを作っていく必要がありますが、ビジュアルプログラミングで作っていくと、膨大なブロックが並んでいくことになりますね。

こんな時に、ある同じものを繰り返し作っていくのではなく、一つ

にまとめてしまいたいときなどがあります。こんな時には、ブロック定義を作っておくことで、長～ぃ プログラミングを分割して見やすくしたり、同じ処理を共通化することができます。

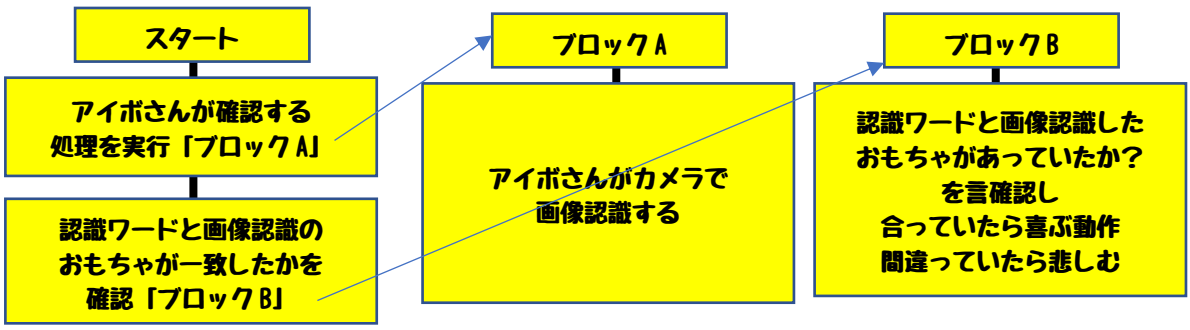
今回は、まずは大きな流れを理解するために次のような流れを作って、ブロック定義を使って長いプログラムを見やすくしたいと思います。

【事前準備】

① 認識ワードを作成
「ピンクボール」「アイボーン」「サイコロ」をそれぞれ
usercommand1, usercommand2, usercommand3 で登録

② 認識したおもちゃが何であったかを保存する
変数「認識ワードで認識したおもちゃ」
変数「画像で認識したおもちゃ」を作る

おもちゃをチェックして確認ができたら、あっているかを表現する



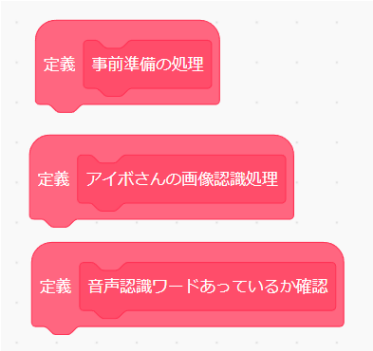
定義ブロックで次のようなブロックを作ってみましょう。
「ブロックを作る」をクリックして作成してみましょう。

【事前準備の処理】という新しいブロックを作ります。

【アイボさんの画像認識処理】という新しいブロックを作ります。

【音声認識ワードとあっているか確認】という新しいブロックを作ります。

新しいブロックが作られました。



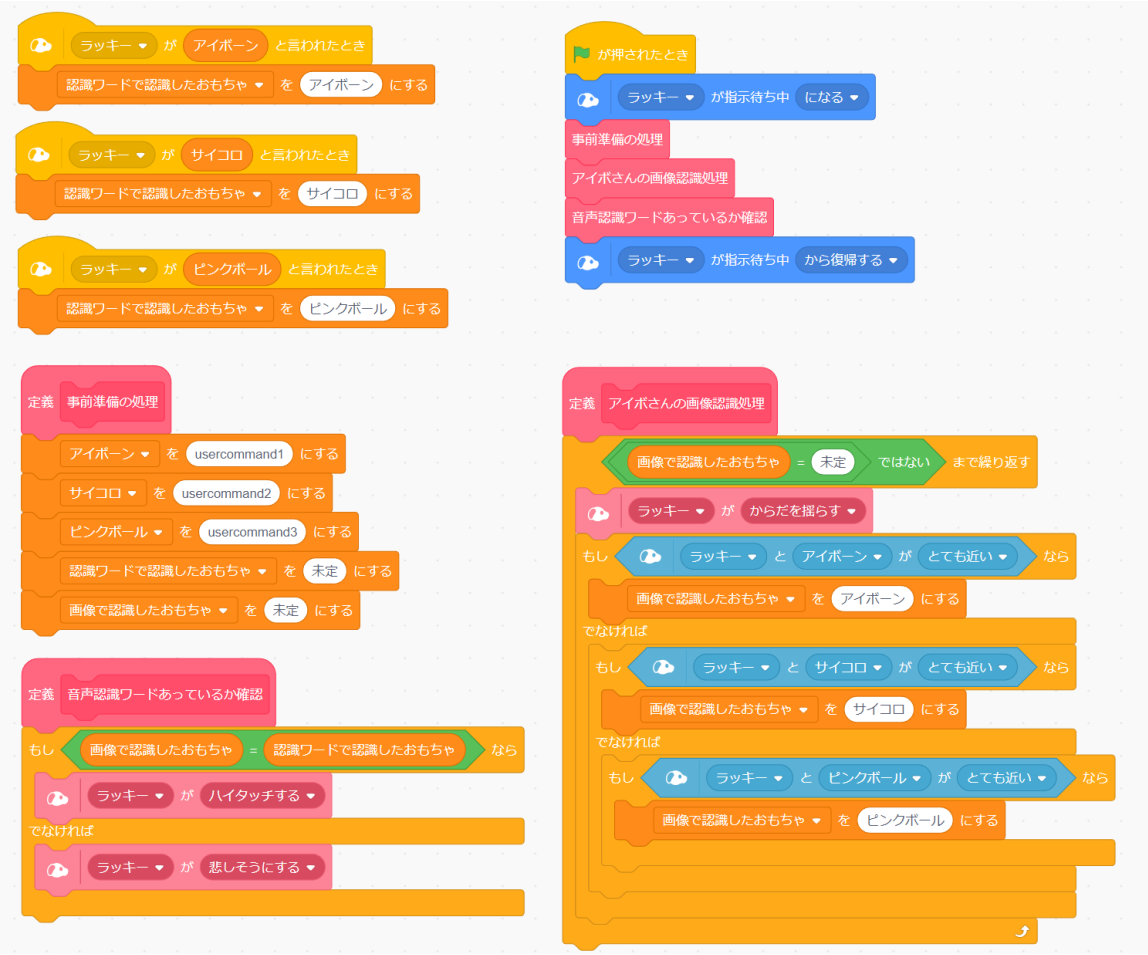
同時にこのようなブロックが自動に画面に表示されたかと思います。

新しいブロックの定義を作ると、自動的に定義というブロックが作られます。この時点でなにがなんだか・・・・になるかと思いますので、先ほどフローチャートの

事前準備
「ブロックA」＝「アイボさんの音声認識処理」
「ブロックB」＝「音声認識ワードとあっている確認」

の定義だと思ってください。

なぜこのようなブロックを作ったのかを説明するために、ＬＥＳＳＯＮ３のプログラムを次のように変更してみました。



今までは、ブロックを積み重ねてきましたが、だいぶ雰囲気がかわってきましたね。よ～く見てみると、ＬＥＳＳＯＮ３のプログラムが分割されているようになっているのが確認できるかと思います。このような分割したり共通化するようなことを、ほかのプログラミング言語などでよく使われる言葉で、関数定義やサブルーチンなどと呼ばれています。

では、もう一度、ひとつひとつ確認してみましょう。

ここは何度も確認しているイベントブロックですね。認識ワードを認識されたときに、スタートとなるイベントブロックです。

ここに先ほど作成した新しいブロックがならんでいます。

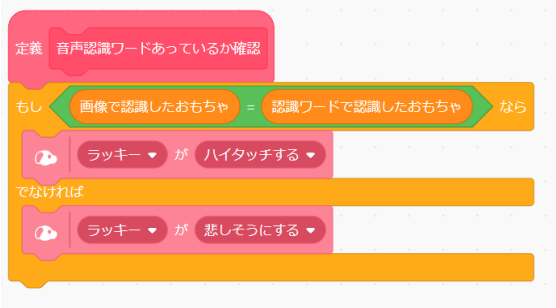
①事前準備の処理

②アイボさんの画像認識処理

③音声認識ワードあっているか確認

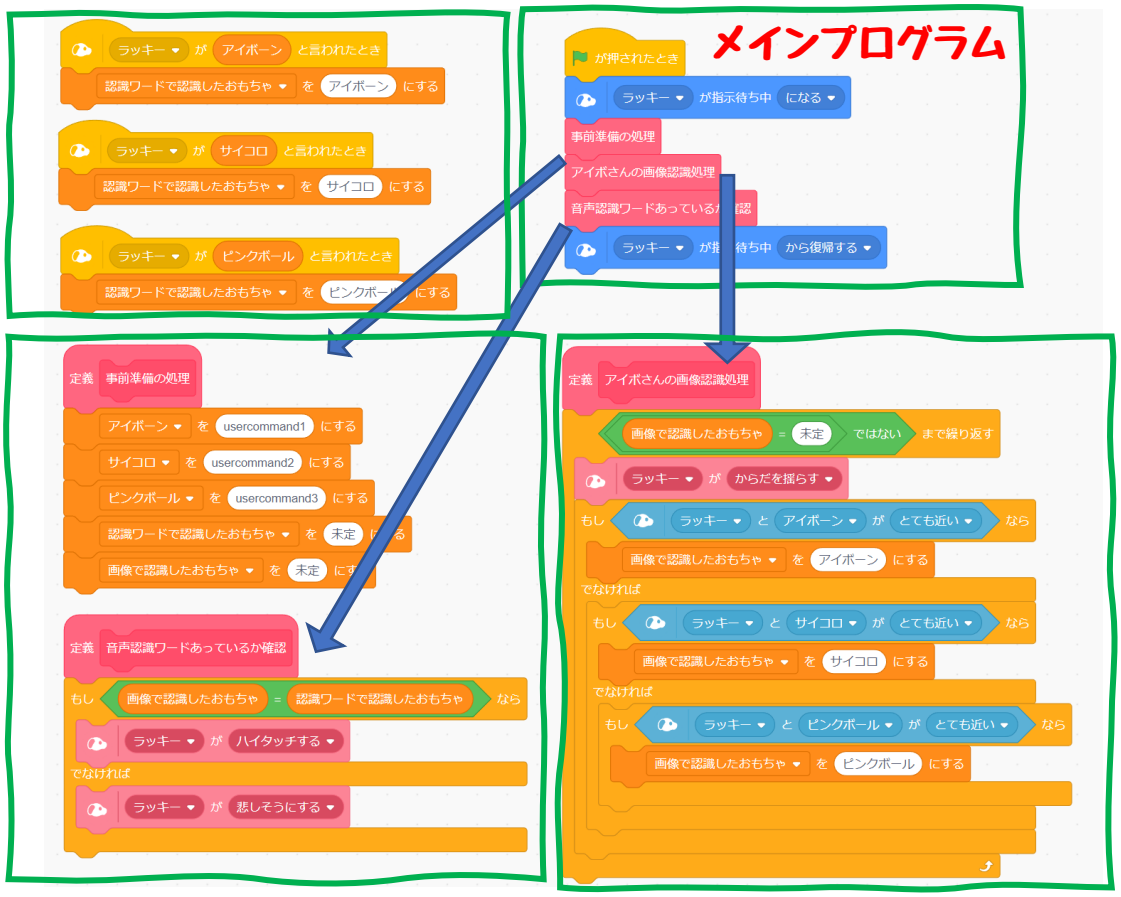
先ほどブロック定義で作成した「事前準備の処理」と準備に必要なブロックを並べています。先ほどの①が実行されるときにこの部分に制御が移り処理され、元の①の場所に戻ります。

先ほどブロック定義で作成した「アイボさんの画像認識処理」とおもちゃを見て判断するブロックを並べています。先ほどの②が実行されるときにこの部分に制御が移り処理され、元の②の場所に戻ります。



先ほどブロック定義で作成した「音声認識ワードあっているか買確認」と準備に必要なブロックを並べています。先ほどの③が実行されるときにこの部分に制御が移り処理され、元の③の場所に戻ります。

いかがでしょうか？　つまり定義されたブロックはそれぞれの同じ名前で自動に作られた【定義 OOOOOO】という先頭となるブロックが実行されるようになることで、長～ぃ　プログラムから解放され、それぞれのブロックが定義されたブロック群が処理されるように見やすくなってきたかと思います。全体を見渡すとこんな感じで動作することとなります。



LESSON4のおさらいをしておきましょう。

① プログラムを作っていくと正しく動作しているか、確認がひつようとなることがあります。とくにうまく動作できない場合、何が起こっているのか確認が必要となります。
② 確認方法はいろいろな方法があります。ビジュアルプログラミングで、変数を使っている場合は、その内容を確認することできちんと処理されているかを確認することができます。
③ これらの確認方法は一般的には「デバック」などと呼ばれています。
④ LESSON3のプログラムのプログラムを使いながら、認識ワードで認識した内容、画像で認識したときの変数の内容を確認してみました。
⑤ 長いプログラムを分割し繰り返し使うものをまとめたり、見やするための手法として【定義】ブロックを使って、LESSON3のプログラムを分割してみました。
⑥ 定義プログラムで分割することで、それぞれ纏まった単位で理解できるようになり、また、見やすくすることができました。

今回は複雑なプログラミング時に必ずと言っていいほど体感する途中経過の状況を確認する方法としてデバックという手法、また、長いプログラムをわかりやすく分割する方法として定義ブロックを使ってみました。是非、いろいろと活用して見やすいプログラムを作ってみてくださいね。

おまけ

画像を処理するプログラム応用として、第１回のビジュアルプログラミングコンテストの時に作成したアイボーンとサイコロを使った、「ビーチフラッグゲーム」を載せておきます。工夫次第で簡単なプログラミングでゲームを作ることができるかと思います。
YouTube で動画もアップしていますので参考にしてください。
https://happy-lucky.fun/aibo/2021/07/31/visulprgraming007/

【遊び方】

① サイコロの上にアイボーンを載せてビーチフラックに見立てます。
② アイボさんを５０ｃｍ程度話して、反対側に向けます。
③ プログラムを実行されると、アイボさんは座って待ちます。
④ 頭を撫でてあげるとスタート
⑤ 180 度回転して、サイコロとアイボーンを探しに行きます。
⑥ ①のビーチフラッグもどきの場所に近づいたらヘディングで倒します。
⑦ お友達のアイボさんがいたら一緒にやって、二人で勝負したり、時間を測定したりしてあそぶことができます。