初等・中等教育における
プログラミングの指導

帝塚山学院大学
喜家村 奨

本日の内容

- 初等教育から中等教育へのプログラミング指導について（指導に利用するプログラミング言語を中心に）
- スクラッチとMakeCodeブロックのイベント処理について（作成した拡張ブロックについて）
- スクラッチとMakeCodeブロック（および他のプログラミング言語）との相違点
- JavaScript（およびPython）利用における留意点

※本日の内容は2022/05に調べた内容です。

中学技術・家庭科の教科書のプログラミング言語の傾向

- 多くの教科書で、プログラミング言語を複数紹介している。
- 紹介されているプログラミング言語は、日本語入力型言語（なでしこ、ドリトル）、スクラッチ、JavaScript
- どの言語を使って、授業を実施するかは、教員よってまちまちであり、生徒がどのプログラミング言語を経験するかもまちまちであることが予想される。

このような状況の中、高校のプログラミング教育における言語選択を、どう考えればよいか？

高等学校情報科「情報Ⅰ」教員研修用教材

- 本編には、Pythonというプログラミング言語が使われている。制御のところではmicro:bitが紹介されている。
- 以下の4種類の言語も例示されている。

ドリトル以外は
実際のアプリケーション開発に
使われているテキスト型の
プログラミング言語

第3章 他プログラミング言語版

- JavaScript版（PDF:7.9MB）
- VBA版（PDF:6.3MB）
- ドリトル版（PDF:5.8MB）
- swift版（PDF:9.8MB）

プログラミング言語の学習ステップ

小学生
初心者

いきなりテキスト型のプログラミング言語に
移行するのは、ハードルが高い

スクラッチなどの
ブロック型
プログラミング言語

中学校
スクラッチ、
日本語表記型プロ
グラミング言語
ドリトル、なでしこ

MakeCode
ブロック型＋
（JavaScript or
Python）

Python
C++
C#
PHP
Ruby
Java
JavaScript

人工知能、データ分析

組み込み系

ゲーム、モバイル

Web系

基幹系

テキスト型に移行してしまえば
他のテキスト型言語は比較的容易に習得
できる。

ブロック型、日本語入力型からいかにスムーズにテキスト型に移行できるかが課題

MakeCodeエディター

- ブロック型、テキスト型（JavaScript、Python）を切り替えて、プログラミングが可能

https://makecode.microbit.org/#editor

スクラッチとmicro:bitの
イベント処理について

イベント処理の方法

- イベントを処理する方法としては、大きく分けて以下の2つの方法がある
  - イベント駆動型：システムがイベントを検知して知らせてくれる。ユーザは発生したイベントに対応した処理をイベントハンドラに書く
  - イベント検知型：自分のプログラム内でイベントの発生を検知する処理を書く

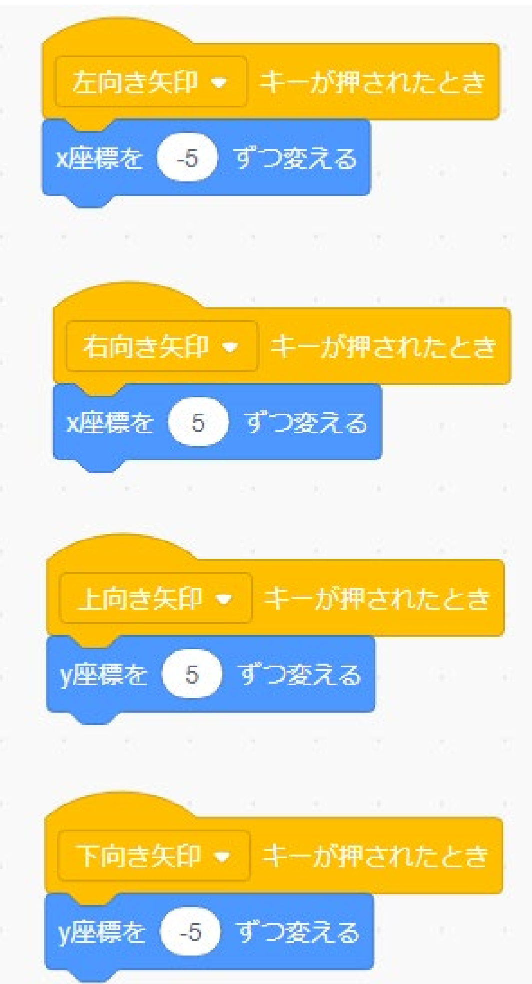
※イベント検知型とは、あまり言いませんが、ここでは、区別のために、そう呼んでいます。

イベント処理の例

- 以下のURLをクリックしてプログラムを実行してみてください。どちらのロケットも矢印キーで操作できますが、処理の仕方が異なります。
- https://scratch.mit.edu/projects/520990142
- 左のロケット：イベント駆動型プログラム
  - Scratchのシステムがイベント検知して知らせてくれる
- 右のロケット：イベント検知型プログラム
  - 自分のプログラム内でイベントの発生を検知する処理を書く

イベント駆動型のプログラム

- 右のプログラムは、イベントブロック「〇〇が押されたとき」を使い、キーが押されたらロケットの座標を動かすイベント駆動型のプログラム
- イベント駆動型のプログラムのメリットは
  - イベントの検知処理を自分で書く必要がない。
  - そのイベントに関する処理だけをひと固まりにかける
  - コードが比較的短くなる
- イベント駆動型のプログラムのデメリットは
  - 複数のイベントを処理する場合、それぞれの処理が並列で動くので、並列処理を意識したプログラミングをする必要がある
  - イベントの発生を無視できない



イベント検知型のプログラム

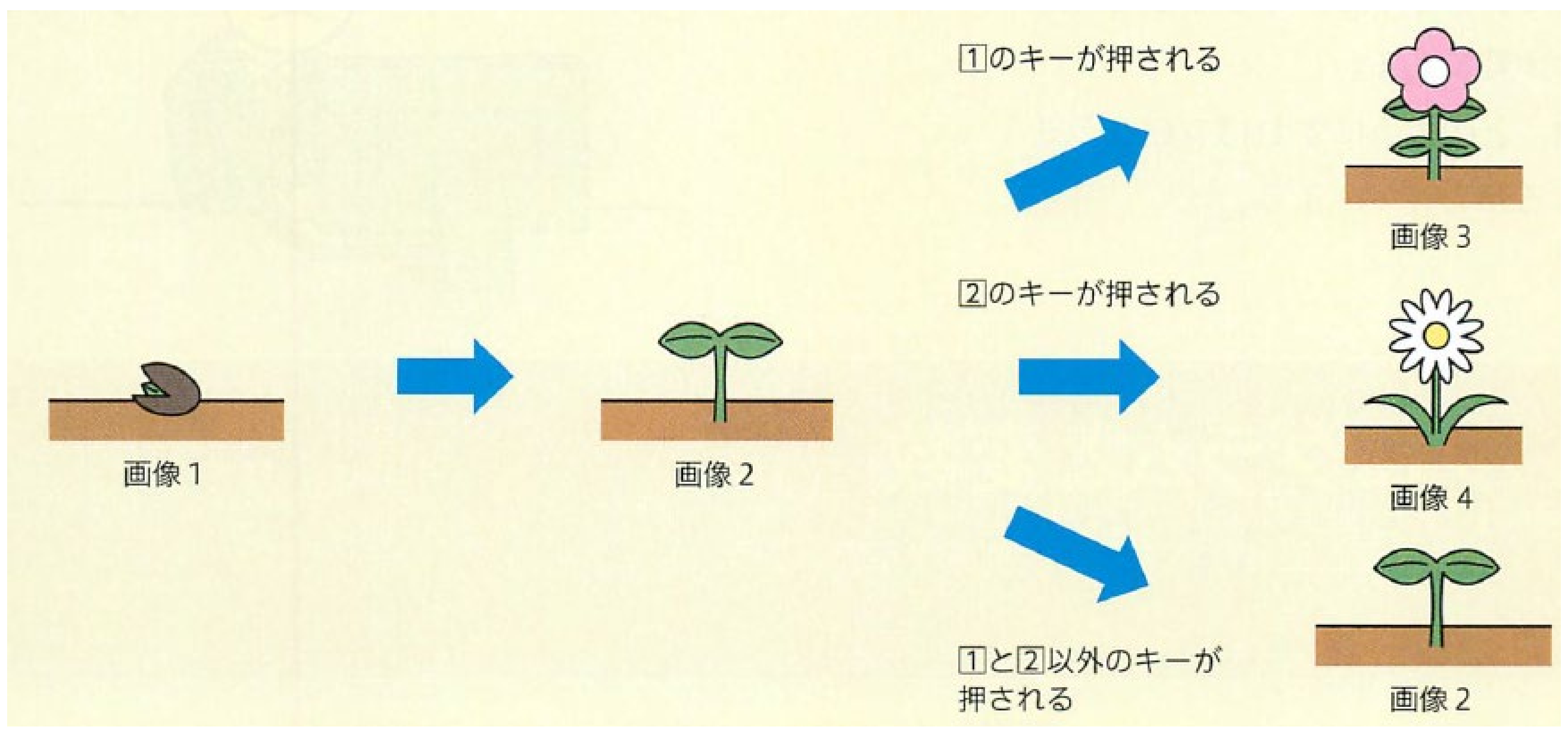
- 「キーが押された」を使い、自分のプログラム内で、キーが押されたかを繰り返し確認し、もし、キーが押されたら、ロケットの座標を動かす。
- イベント検知型のプログラムのメリットは
  - 複数のイベント処理を逐次的に記述できるので、プログラム全体の見通しがいい。
- イベント検知型のプログラムのデメリットは
  - 自分の処理の中でずっとイベントの発生を検知しないといけないので、CPU時間を浪費する。
  - コードが長くなりがち

どちらのイベント処理方法を使うか

- 一概にどちらがいいとは言いにくのですが、並列処理を意識したイベント処理をちゃんと書くのは難しい（排他制御のためにセマフォを使うとか、イベントキューを使うとか）。
- 次に示す例のように、ある状態から、発生するイベントによって処理をかえるようなとき（自動販売機などの順序機械のプログラミング）は、イベント検知型のほうが理解しやすいように思う。例で説明します。

スクラッチのイベント処理プログラムの例

- 以下のような、キー入力を待ち、入力されたキーの応じて、処理をするプログラムを考える。



花が咲く処理のフローチャート

画像1を表示

画像2を表示

キーが押されるまで待つ

入力されたキーは？

1のキー

2のキー

1,2以外

画像3を表示

画像4を表示

画像2を表示

スクラッチのプログラム

```
[旗] が押されたとき
コスチュームを 画像1 ▾ にする
1 秒待つ
コスチュームを 画像2 ▾ にする
どれかの ▾ キーが押された まで待つ
もし 1 ▾ キーが押された なら          ⑥
  コスチュームを 画像3 ▾ にする
でなければ
  もし 2 ▾ キーが押された なら
    コスチュームを 画像4 ▾ にする      ⑦
  でなければ
    コスチュームを 画像2 ▾ にする      ⑧
```

1のキーが押される → 画像 3

2のキーが押される → 画像 4

1と2以外のキーが押される → 画像 2

フローチャートで表現された順序機械の実装

- フローチャートは設計図であり、その手順通り、プログラムが書けることが望ましい。
- 例で示したようなプログラムで、イベント駆動型にすると（イベントハンドラを使うと）、同期処理のために共有変数を使ったり、イベントの発生後の処理を条件で制限したりする必要がある。

micro:bitでの実装

- 先ほどのプログラムをmicro:bitで実装することを考える。1または2のキーを押す代わりにmicro:bitのAボタンとBボタンを使ったとしても、MakeCodeのブロックでは、イベントの入力を待つブロックはイベントハンドラのみであり、処理が並列になる。
- ボタンを押されたかを検知するプログラムを書くこともできるが、スクラッチのプログラムより煩雑になる ⇒

ボタンの入力待ちのための拡張ブロックを作成

MakeCodeのボタンの入力を待つブロック

作成したボタン入力待ち拡張ブロック

- micro:bit A、Bボタンが任意のパターンで押されるまで待つブロック。

変数 a を いずれか のボタンが押されるまで待つ にする

ボタンが押されるまで、
このブロックでウエイトする

A
B
A+B
AかB
AかA+B
BかA+B
AかBかA+B
✔ いずれか

A,B,A+Bボタンを
任意の組み合わせで待つ

ボタン入力待ち拡張ブロックの利用

- 拡張ブロックを利用すると、先ほどのスクラッチのプログラムと同様に、逐次処理プログラムとしてmicro:bitでも実装できる

どれかの キーが押された まで待つ
もし 1 キーが押された なら
コスチュームを 画像3 にする
でなければ

変数 a を いずれか のボタンが押されるまで待つ にする
もし a = 1 なら
LED画面に表示

拡張ブロックを用いた逐次処理プログラム

変数 a を いずれか のボタンが押されるまで待つ にする

もし a = 1 なら

LED画面に表示

その他の作成した拡張ブロック

- スクラッチの〇〇と聞いて待つブロックの置き換え用にmicro:bitのLEDにメッセージを表示して、その後、数字を選択するブロックも作成した

仕様

- メッセージを表示し、その後、0～9までの数値が入力（選択）されるまで、このブロックで処理を継続
- 数値の入力はAボタンを押すと0、1、2・・・9と順番に選択でき、Bボタンで確定

スクラッチの〇〇と聞いて待つブロック

データはいくつですか？ と聞いて待つ

作成した〇〇と聞いて数字入力を待つブロック

変数 data を 〇〇と聞いて待つ "data=" にする

初等・中等教育での
プログラミング指導の留意点

初等教育での
プログラミング指導の留意点

- 自動販売機などの順序機械のプログラミングでは、設計図であるフローチャートや状態遷移図にそって、プログラミングできるようにイベント駆動型のプログラミングを避けたほうがよい。
- 並列処理プログラムは、処理が複雑になり、デバッグなども含め、指導が難しい。

中等教育での
プログラミング指導の留意点

帝塚山学院大学

- ビジュアル言語を利用した場合
  - テキスト型プログラミングへのステップアップをスムーズにできるように工夫する。
- テキスト型プログラミング言語を利用した場合
  - 学生のテキスト型言語に対する理解度は、学生によって異なる ⇒ Makecodeのようなブロック型⇔テキスト型タイプの開発環境で、それぞれの学生の習得度に合わせてプログラミング言語を切り合えられることは有効
- 日本語入力型のプログラミング言語を利用する場合、高校への接続性を配慮することが望ましい。
- 通信プログラムについて
  - 通信プログラムでは、処理が並列になるため、同期や排他制御、イベントキューなどの概念を少し一緒に教えてもいいのではないかと思う？？

スクラッチのプログラミング指導における留意点（MakeCodeおよび、その他のプログラミング言語と比較して）

配列の添え字について

- スクラッチの配列（リスト）の添え字は1から始まる。MakeCodeも、その他のテキスト型言語も配列の添え字は0から始まる。

ループの繰り返し条件について

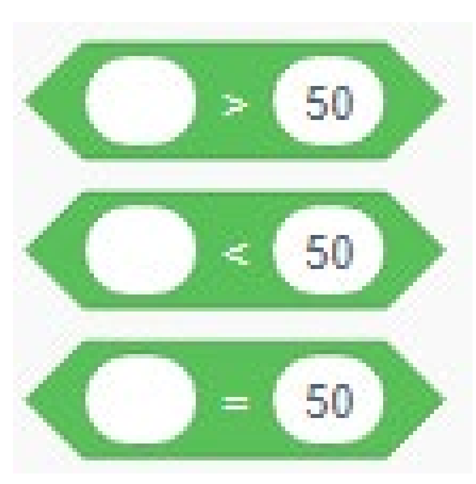
- スクラッチでは、ループブロックの繰り返し条件の真偽が通常と逆で、条件が成り立つまで繰り返すことになっている。

スクラッチの繰り返しブロック

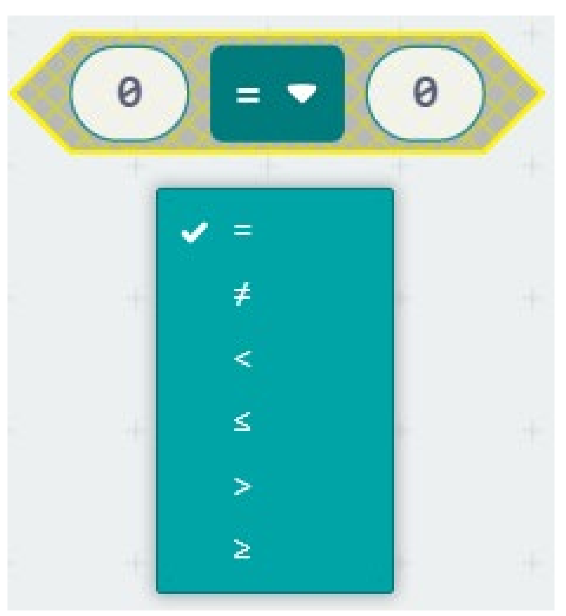
MakeCodeの繰り返しブロック

比較演算子について

- スクラッチでは、比較演算子が＞、＜、＝しか準備されていない。
- 以上（≧）、以下（≦）がないと、条件を読み替えないといけないときがある。
（例えば：80点以上などと書けない）



スクラッチの比較演算子ブロック

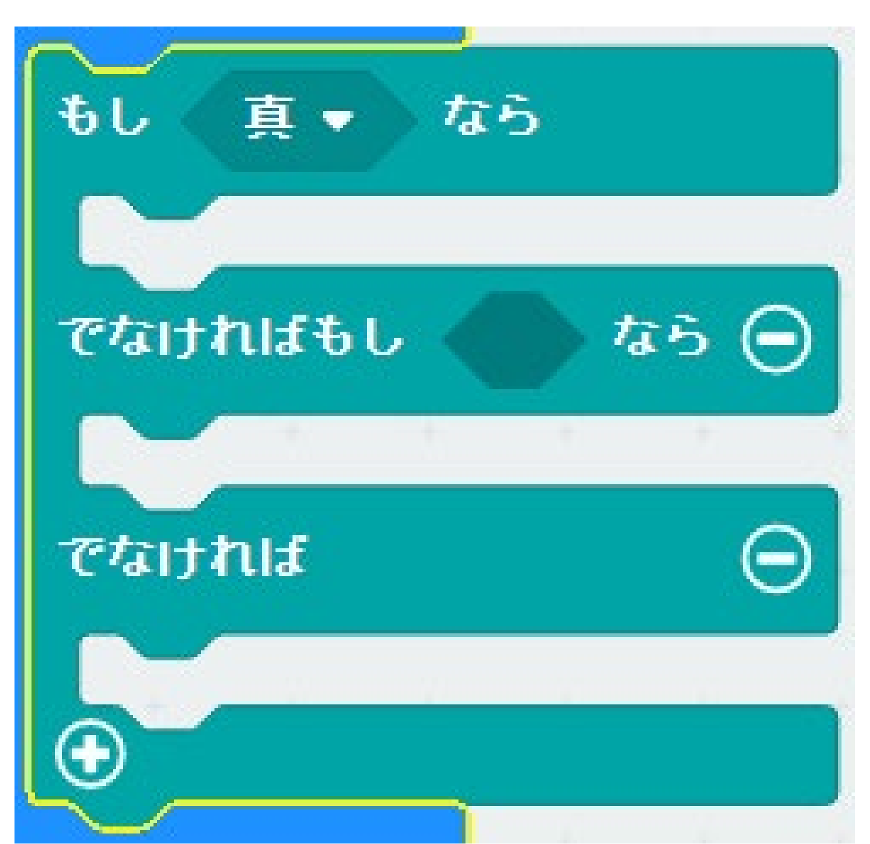


MakeCodeの比較演算子ブロック

条件分岐について

- スクラッチのIF文には、else if節が用意されていない。

スクラッチの条件分岐ブロック



MakeCodeの条件分岐ブロック
＋を押すとelse if節を増やしていける

else-ifの使用例

- else-if節は以下のようなときに使える。
- 試験の点数が80点以上の人は“優”、60点以上の人は“可”、それ以外の人は“不可”となるとき、点数を評価するプログラムは

点数≧80

NO

YES

成績は優

点数≧60

NO

YES

成績は可

成績は不可

スクラッチで書くと

MakeCodeのブロックでは

ずっと
もし 点数 ≥ 80 なら
文字列を表示 "Very Good!"
でなければもし 点数 ≥ 60 なら
文字列を表示 "Good!"
でなければ
文字列を表示 "Bad!"

関数の引数と戻り値について

- スクラッチでは、関数の引数は使えるが戻り値は使えない。
- 引数が使えるので、関数を一般化することは可能だが、戻り値がないと、関数の結果をグローバル変数に代入するか、直接、出力（ステージに表示等）するしかない。

関数の一般化の例

正N角形を描く 4

関数の結果をグローバル変数に代入

関数の結果を画面に出力

補足：MakeCodeブロックでの引数の操作の注意点

- MakeCodeでプログラミングしていると引数と同じ名前の変数が知らない内に作られてしまうときがあります。下のプログラムは正常に動作しません。

最初だけ
変数 配列 を この要素の配列 1 2 3 にする
数を表示 呼び出し 合計を求める 配列

関数 合計を求める list
変数 合計 を 0 にする
変数 カウンター を0～ 配列の長さ list − 1 に変えてくりかえす
変数 合計 を list の カウンター 番目の値 だけ増やす
戻る 合計

MakeCodeのブロックでの引数の操作の注意点

- 必ず関数の定義のところからドラッグする

関数 合計を求める list
変数 合計 を 0 にする
変数 カウンター を0 配列の長さ list − 1 に変えてくりかえす
変数 合計 を list の カウンター 番目の値 だけ増やす
戻る 合計

応用：
ブロック型からJavaScript、Python言語への移行における留意点

MakeCodeでは関数内のローカル変数を使用できない

- 下のプログラムはブロック化すると、グローバル変数に書き換えられる

```
function add (a: number, b: number) {
    let ans = a + b
    return ans
}
basic.showNumber(add(1, 3))
```

一度ブロック化するとグローバル変数に勝手に置き換えられる

```
r, b: number) {
    ans = a + b
    return ans
}
let ans = 0
basic.showNumber(add(1, 3))
```

関数の引数について

- JavaScriptでは、引数の渡し方に「値渡し」と「参照渡し」があり、「参照渡し」の場合、関数内で引数の値を変更するとそれが、呼び出し側の実引数に影響する

仮引数と実引数

- 仮引数：関数を定義するときに宣言する引数
- 実引数：関数を呼び出すときに渡す引数

```
function 数字型の引数 (a: number) {
    let y = a + 1
    a = 5
    return y
}
```

仮引数

```
let a1 = 1
let a2 = "hello"
let a3 = [1,2,3]
basic.showNumber(数字型の引数(a1))
```

実引数

JavaScriptのデータ型と引数の渡し方の関係

- JavaScriptのデータ型には、基本（プリミティブ）型と参照型がある
- 基本型：数値、文字列、真理値など
- 参照型：配列など

- データタイプの基本型は、「値渡し」され、参照型は「参照渡し」される

引数が基本型の場合の例

- 引数が基本型の場合は、「値渡し」なので、関数内で仮引数に代入しても実引数に影響はない

```
function 文字列型の引数 (txt: string) {
    let x = txt + "Wd"
    txt = "bye"
    return x
}
function 数字型の引数 (a: number) {
    let y = a + 1
    a = 5
    return y
}
```

```
let a1 = 1
let a2 = "hello"
let a3 = [1,2,3]
basic.showNumber(数字型の引数(a1))
basic.showString(文字列型の引数(a2))
basic.showNumber(配列の引数(a3))
basic.showIcon(IconNames.Heart)
basic.showNumber(a1)
basic.showString(a2)
basic.showNumber(a3[0])
```

a1は1まま
a2はhelloのまま

※このプログラムをMakeCodeで打ち込んでも一度、ブロックボタン押すと、
代入している引数はグローバル変数に
無理やり置き換えられます。

引数が参照型の場合の例

- 配列は「参照渡し」なので、以下のように引数で渡した配列の内容を関数内で操作できる

```
function 配列の引数(list: any[]) {
    let z = list[0] + 1
    list[0] = 9
    return z
}
```

```
let a1 = 1
let a2 = "hello"
let a3 = [1,2,3]
basic.showNumber(数字型の引数(a1))
basic.showString(文字列型の引数(a2))
basic.showNumber(配列の引数(a3))
basic.showIcon(IconNames.Heart)
basic.showNumber(a1)
basic.showString(a2)
basic.showNumber(a3[0])
```

a3[0]は9になる

※このプログラムをMakeCodeで打ち込んで、ブロックボタンを押しても、ブロックに変換されません。

配列の要素の入れ替え

- 配列が引数で渡せると以下のような、任意の配列の要素を入れ替える関数が実装できる。

```
function 入れ替える (list: any[], i: number, j: number) {
    let temp = list[i]
    list[i] = list[j]
    list[j] = temp
}
let 配列 = [1, 2, 3]
basic.showIcon(IconNames.Heart)
入れ替える(配列, 0, 2)
for (let カウンター = 0; カウンター <= 2; カウンター++) {
    basic.showNumber(配列[カウンター])
}
```

ブロック型でも関数内での配列操作は可能？

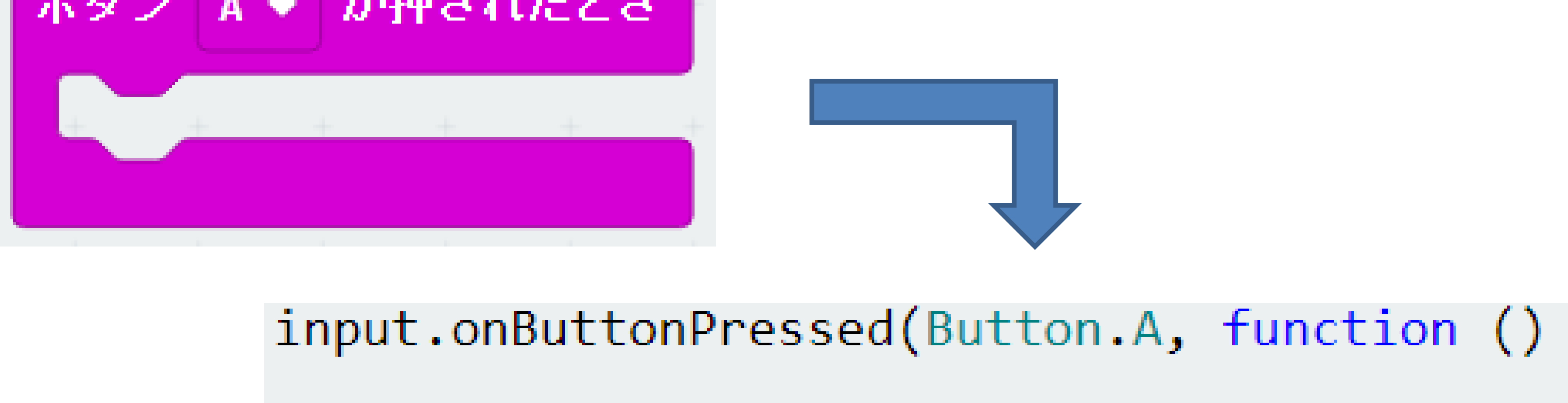
- ブロック型でも、引数の配列への代入は可能だが、ブロック型に変換できない場合もあるので（前述）、基本、ブロック型では、引数に対する代入は禁止されているようである

最初だけ
変数 配列 を この要素の配列 1 2 3 にする
アイコンを表示
呼び出し 入れ替える 配列 0 2
変数 カウンター を0～ 2 に変えてくりかえす
数を表示 配列 の カウンター 番目の値

関数 入れ替える list i j
変数 temp を list の i 番目の値 にする
list の i 番目の値を list の j 番目の値 にする
list の j 番目の値を temp にする

いきなり高階関数は難解では？

- MakeCodeでは、入力ブロックなどで、無名関数、および高階関数が利用されている。高階関数の概念は少し高度すぎるように思う。




```
input.onButtonPressed(Button.A, function () {

})
```

micro:bitのPythonの環境

- Python Editorを利用する
- Muなどのエディタを使ってmicroPythonでプログラミングする
- 変換機能を利用してJavaScript同様にMakeCode内でプログラミングする

MakeCodeのPythonについて

- タプルなどのPython固有のデータ型や記述は使えない。あくまで、Makecodeのブロックおよび、JavaScriptとの互換性を考慮した実装になっている。
- Pythonでは、{}ではなく、インデントで、プログラムの構造を表現するので、その点は配慮が必要

ご清聴ありがとうございました。

本研究はJSPS科研費JP20K02528の助成を受けています.