

復興知プロジェクト ロボット講習会 in 南相馬

中村啓太

会津大学 復興支援センター

2018/11/17

発表構成

- ❖ プログラムとは
- ❖ Pythonプログラミングの基礎
- ❖ OpenRTM-aistのプログラミングの流れ
- ❖ OpenRTM-aistによる操作
- ❖ EV3の動かすためのプログラム
- ❖ 本日の課題

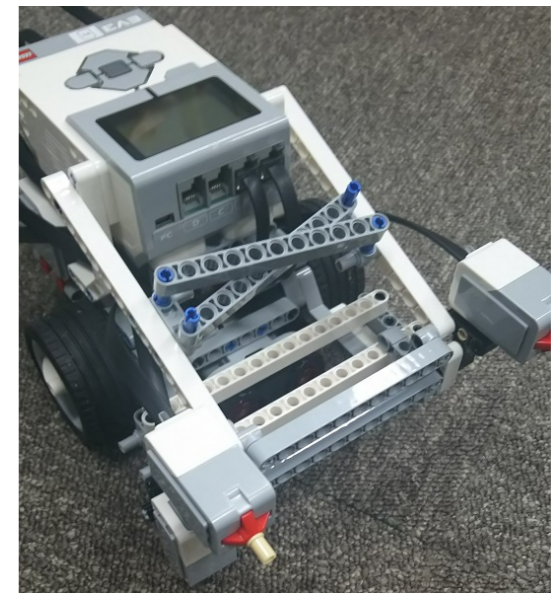
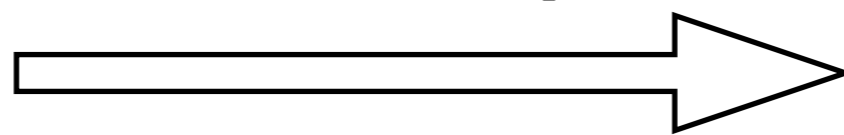
講習会目的

- ❖ プログラミングの基礎を学ぶ (Python)
- ❖ RTミドルウェアを使用して、自由自在にロボットを動かす
 - ❖ 自由自在にロボットを動かす
 - ❖ 仕様に沿ってプログラムを作成する



プログラム

コマンド



EV3

プログラムとは？

- ❖ コンピュータへの命令を記述したもの
 - ❖ コンピュータはこのプログラムに沿って動いている

MS Word

MS Excel

OpenRTM-aist

Windows

mplayer

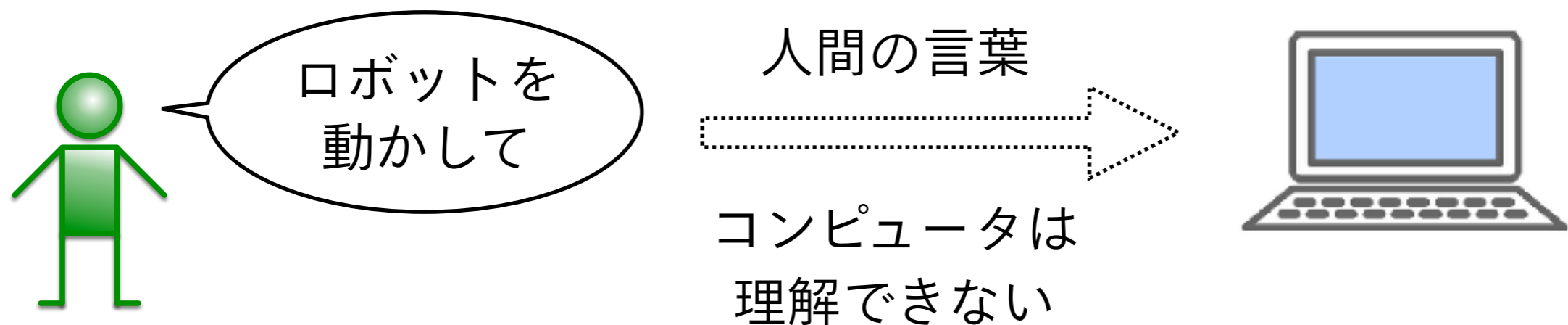
Firefox

OpenRTM-aist

Linux

プログラムの動かし方

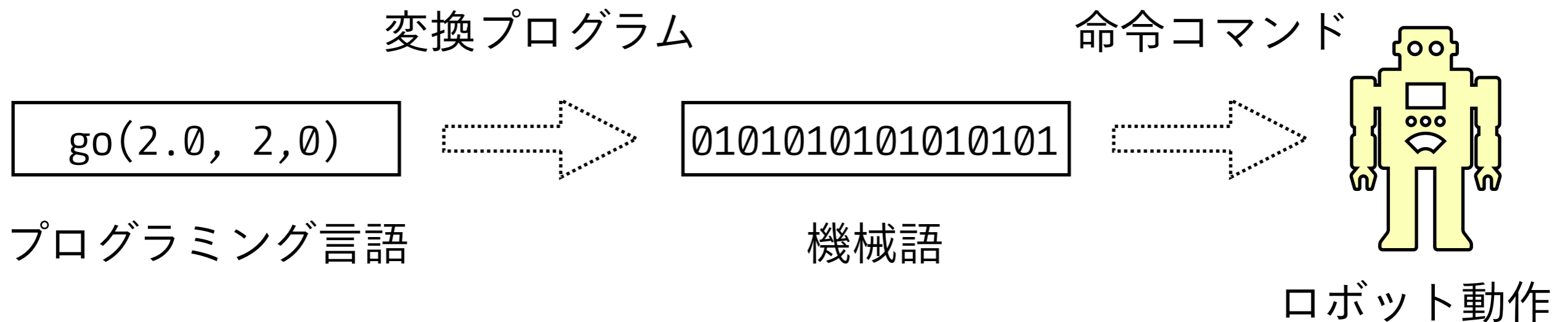
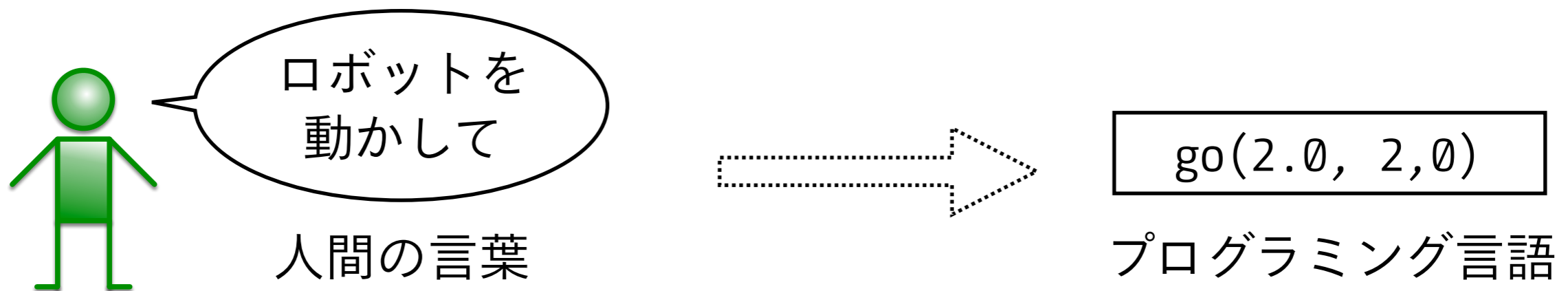
- ❖ プログラムを使えば、コンピュータを動かすことができる
- ❖ コンピュータは機械語(1と0の羅列)しか理解できないので、人間の言葉はコンピュータに命令をするプログラムとならない



人間は「0」と「1」のみで表現する機械語で話すことはできない

プログラミング言語

- ❖ プログラミング言語は機械語に変換することが可能
- ❖ プログラミング言語は人間にもわかりやすい



プログラミング言語の種類

- ❖ プログラミング言語は現在たくさんの種類がある
 - ❖ 各プログラミング言語には独自の特徴がある
- ❖ プログラミング言語の例
 - ❖ C言語：OSや組み込み系でよく使用されている言語
 - ❖ **Python**：AI分野でよく使用され、文法が比較的簡単な言語
 - ❖ Javascript：動的webサイト作成でよく使用されている言語

Pythonの概要

プログラミング言語

- ❖ コードがシンプルで、読みやすさ・書きやすさを重視した設計
 - ❖ 様々なOSに対応 (Windows, macOS, Unix, Linuxなど)
 - ❖ アプリケーション(SNSなど), 人工知能, ロボットなど, 幅広い開発分野で利用



- ❖ **この資料が対象とするバージョンは2.7.x系**
 - ❖ **最新のバージョンは3.7.x系 (2系と3系は互換が無い)**

対話モードの起動・終了方法

- ❖ Pythonでプログラムを実行する方法は2通りある
 1. 対話モードで、直接コードを記述して実行する
 2. ファイルにコードを記述し、読み込ませて実行する
- ❖ 対話モードとは、簡単にPythonプログラムを実行する環境
- ❖ 対話モードの起動は、「スタートメニュー」→「アクセサリ」→「コマンドプロンプト」を起動後、コマンドプロンプトで **python** と入力して「Enter」キーを押す
- ❖ 対話モードの終了は、コマンドプロンプトで **quit()** と入力する

対話モードの起動・終了方法

❖ Pythonでは # 以降はコメントとなる

```
python # 対話モードの起動
```

```
Python 2.7.15 (default, Jun 17 2018, 12:46:58)
```

```
[GCC 4.2.1 Compatible Apple LLVM 9.1.0 (clang-902.0.39.2)]  
on darwin Type "help", "copyright", "credits" or "license"  
for more information.
```

```
>>>
```

```
# 対話モードの終了
```

```
>>> quit()
```

対話モードでの数式実行

- ❖ 対話モードで簡単な数式を入力し実行することで、数式の計算結果が表示される

```
>>> 2 + 2
```

```
4
```

```
>>> 4 / 2
```

```
2
```

```
>>> 2 / 4
```

```
0 # 整数同士の場合，除算結果も整数
```

```
>>> 2.0 / 4
```

```
0.5 # どちらかが小数の場合，除算結果は小数
```

対話モードでの数式実行

❖ 計算の順番は、数学の数式と同様

❖ 丸括弧() → 乗除計算 → 加減計算の順序で行う

```
>>> (1 + 5) / (2 + 3)
```

```
1 # (1 + 5) / (2 + 3) = 6 / 5 = 1
```

```
>>> 1 + 5 / 2 + 3
```

```
6 # 1 + 5 / 2 + 3 = 1 + 2 + 3 = 6
```

❖ 計算順番さえ分かれば、簡易電卓として使用可能

演算子, 変数, 標準入出力, プログラム実行

Pythonの代数演算子

演算子	意味	使用例	結果例
+	加算	$7 + 3$	10
-	減算	$7 - 3$	4
*	乗算	$7 * 3$	21
/	除算	$7 / 3$	2
%	剰余	$7 \% 3$	1
**	累乗	$7 ** 3$	343
//	切り捨て除算	$7.0 // 3$	2.0

Pythonで文字列を扱う

- ❖ シングルクォーテーション(')か、ダブルクォーテーション(")を使用して、入力したい文字列を囲む
- ❖ どちらの記号を使っても問題ないが、前後の囲む記号は統一すること

```
>>> 'xyz'  
'xyz' # 文字列
```

- ❖ 文字列同士は、"+"演算子で繋げることができる

```
>>> 'xyz' + '123'  
'xyz123'
```


標準入出力

- ❖ コマンドプロンプト上に演算結果や文字列を標準出力するには、**print**メソッドを使用

```
>>> print(2 * 3)
```

```
6
```

```
>>> print('abcdef')
```

```
abcdef
```

- ❖ キーボードから標準入力するには、**input**メソッド、もしくは**raw_input**メソッドを使用

- ❖ **input**メソッドは数値入力、**raw_input**メソッドは文字列入力で使用

標準入力の注意点

- ❖ **input**メソッドを使用し、標準入力されたデータが文字列の場合、バージョンによってはエラーが発生
- ❖ **raw_input**メソッドを使用し、標準入力されたデータが数値の場合、エラーが発生

```
>>> input() + 10
```

```
60      # キーボードで数値を入力する
```

```
70
```

```
>>> raw_input() + 'abc'
```

```
xyz     # キーボードで文字列を入力する
```

```
'xyzabc'
```

Pythonにおける基本的データ型

データ型	例
整数 (int)	0, -12, 345, ...
小数 (float)	0.1, 2.3, -4.5, ...
文字列 (string)	'ABC', "123", 'a1b2c3', ...
真偽値 (boolean)	True, False

データ型とキャスト

❖ データ型を変換するキャストメソッドが存在する

❖ 小数から整数に変換した場合、小数点以下の情報が消える

```
>>> float(5)
```

```
5.0      # 整数から小数にキャスト
```

```
>>> int(1.5)
```

```
1        # 小数から整数にキャスト
```

```
>>> str(5)
```

```
'5'      # 整数から文字列にキャスト
```

```
>>> int('15')
```

```
15       # 文字列から整数にキャスト
```

Pythonプログラムの実行

- ❖ ファイルにコードを記述し、読み込ませて実行する
- ❖ テキストエディタを使用して、「**.py**」ファイルを作成
 - ❖ **py**は拡張子
- ❖ 端末上で『**python ファイル名.py**』と入力し実行

```
print('Hello World')
```

hello.py

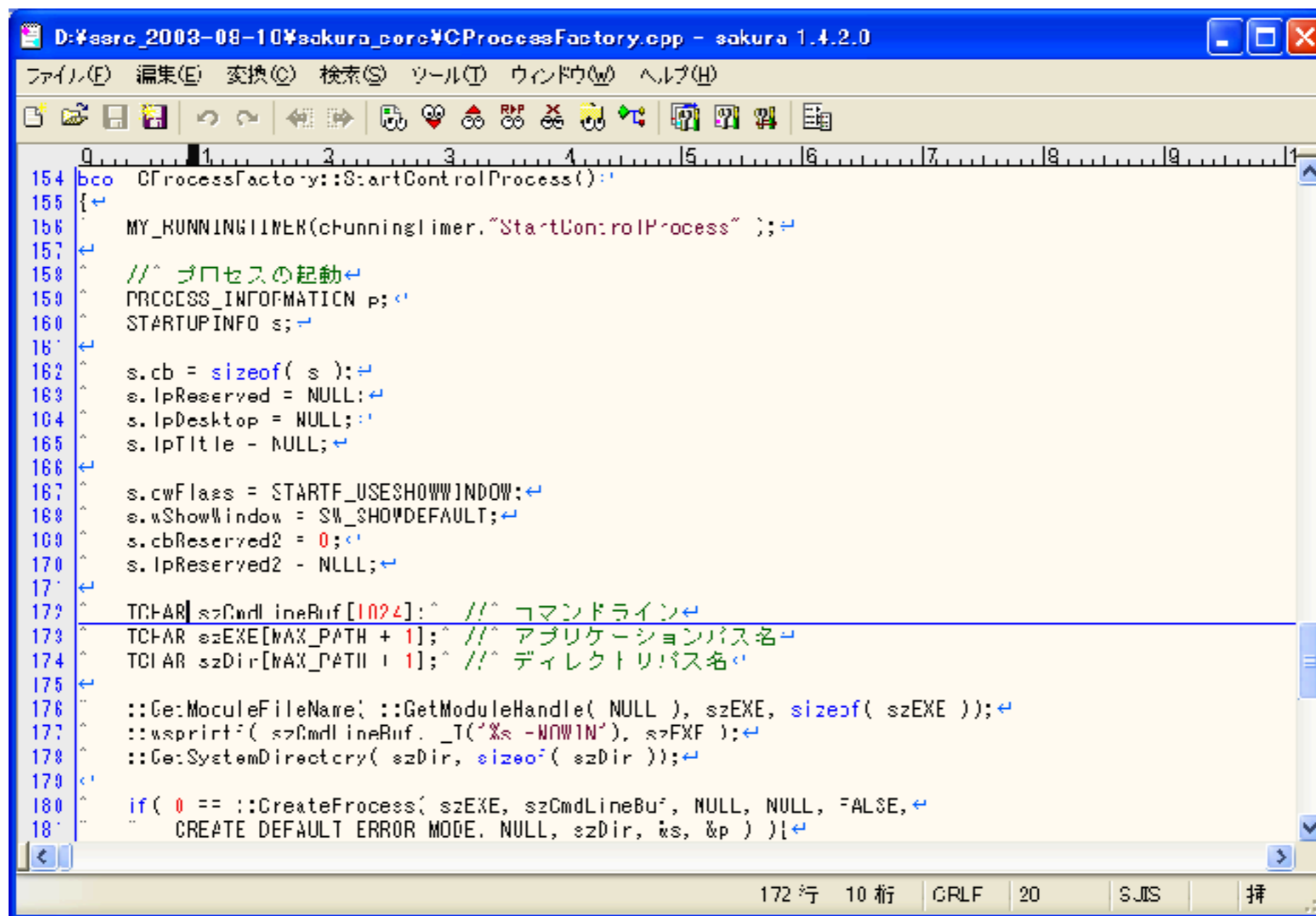
```
python hello.py
```

```
Hello World # 実行結果
```

ファイルを作成して、
端末上で実行する

テキストエディタの導入

- ❖ サクラエディタ : <https://sakura-editor.github.io/>



```
D:\sarc_2003-08-10\sakura_porc\CProcessFactory.cpp - sakura 1.4.2.0
ファイル(F) 編集(E) 変換(C) 検索(S) ツール(T) ウィンドウ(W) ヘルプ(H)
154 bco CProcessFactory::StartControlProcess()
155 {
156     MY_RUNNINGTIMER(cunninglimer, "StartControlProcess");
157
158     // プロセスの起動
159     PROCESS_INFORMATION p;
160     STARTUPINFO s;
161
162     s.cb = sizeof( s );
163     s.lpReserved = NULL;
164     s.lpDesktop = NULL;
165     s.lpTitle = NULL;
166
167     s.dwFlags = STARTF_USESHOWWINDOW;
168     s.wShowWindow = SW_SHOWDEFAULT;
169     s.cbReserved2 = 0;
170     s.lpReserved2 = NULL;
171
172     TCHAR szCmdLineBuf[1024]; // コマンドライン
173     TCHAR szEXE[MAX_PATH + 1]; // アプリケーションパス名
174     TCHAR szDir[MAX_PATH + 1]; // ディレクトリパス名
175
176     ::GetModuleFileName( ::GetModuleHandle( NULL ), szEXE, sizeof( szEXE ) );
177     ::wsprintf( szCmdLineBuf, _T("%s -NOWIN"), szEXE );
178     ::GetSystemDirectory( szDir, sizeof( szDir ) );
179
180     if( 0 == ::CreateProcess( szEXE, szCmdLineBuf, NULL, NULL, FALSE,
181         CREATE_DEFAULT_ERROR_MODE, NULL, szDir, &s, &p ) ) {
```

172行 10桁 CRLF 20 SJS 挿

日本語の取り扱い

- ❖ Python2系で、日本語が含むプログラムを実行する際には、最初の行に文字コードを指定するマジックコメントを記述
- ❖ この講習では、文字コードをUTF-8として扱うため、以下の1行を最初に記述

```
# -*- coding: utf-8 -*-
```

```
# -*- coding: utf-8 -*-  
print('テスト')
```

test.py

```
python test.py  
テスト # 実行結果
```

ファイルを作成して、
端末上で実行する

ここまでのまとめ演習1

- ❖ 100円の商品を5つ, 300円の商品を2つ購入しました
消費税を8%としたときの値段を出力する
プログラム(q1-1.py)を作成せよ
- ❖ 以下のような文字列を出力するプログラム(q1-2.py)を作成せよ
xxxはキーボードから入力した任意の文字が入ることとする

```
Hello, xxx
```


まとめ演習1の解答例

- ❖ 100円の商品を5つ, 300円の商品を2つ購入しました消費税を8%としたときの値段を出力するプログラム(q1-1.py)を作成せよ

```
print(int((100 * 5 + 300 * 2) * 1.08))
```

- ❖ 以下のような文字列を出力するプログラム(q1-2.py)を作成せよ
xxxはキーボードから入力した任意の文字が入ることとする

```
print('Hello, ' + raw_input())
```

変数

- ❖ 数値や文字列に付ける名札のようなもの
- ❖ 変数は自由に付けられるが、以下のルールがある
 - ❖ 英数字、アンダースコア(`_`)のみ使用可能
 - ❖ 数字から始まらないこと (英字、アンダースコアはOK)
 - ❖ 英字の大文字と小文字は区別される (ABCとabcは別の変数)
 - ❖ 予約語 (Pythonで文法で定義されている単語) は使用不可 (if, for, andなど)
 - ❖ 大文字やアンダースコアで始まる変数名は避ける
 - ❖ 規則上問題はないが、一般的には使用されない

変数と初期化

- ❖ 数値や文字列に付ける名札のようなもの
- ❖ 『**変数 = 初期値**』で変数の初期化を行う
- ❖ Pythonは動的型言語のため、初期化によって変数の型が決定
 - ❖ C言語は変数の型名を決めないとエラーになる
- ❖ 『**type(変数)**』で、その変数の型を調べることができる

比較演算子

- ❖ 2つの数字や文字列の比較を行うときに使用する演算子
- ❖ これらの演算子は、加算や減算の代数演算子よりも後に計算する

演算子	数学記号	意味
==	=	等しい
!=	≠	等しくない
<	<	より小さい
>	>	より大きい
<=	≦	以下
>=	≧	以上

論理演算子

- ❖ 2つ以上の条件を組み合わせ、真偽を判定する演算子
 - ❖ not : 論理否定 (Trueの場合 : False, Falseの場合 : True)
 - ❖ and : 論理積 (両方がTrueの場合 : True, それ以外 : False)
 - ❖ or : 論理和 (一方がTrueの場合 : True, それ以外 : False)
 - ❖ not, and, orの順番で演算する

not条件式	結果
not True	False
not False	True

and条件式	結果
True and True	True
True and False	False
False and True	False
False and False	False

or条件式	結果
True or True	True
True or False	True
False or True	True
False or False	False

比較演算子と論理演算子

❖ 比較演算子は真偽の判定結果を返すので、論理演算子と組み合わせて使用可能

❖ $4 < 5$ は真偽値ではないが、式を評価すると真偽値(True)となる

```
>>> (4 < 5) and (5 < 6)
```

```
True      # True and True
```

```
>>> (4 < 5) and (9 < 6)
```

```
False     # True and False
```

```
>>> (1 == 2) or (2 == 2)
```

```
True      # False and True
```

複合演算子

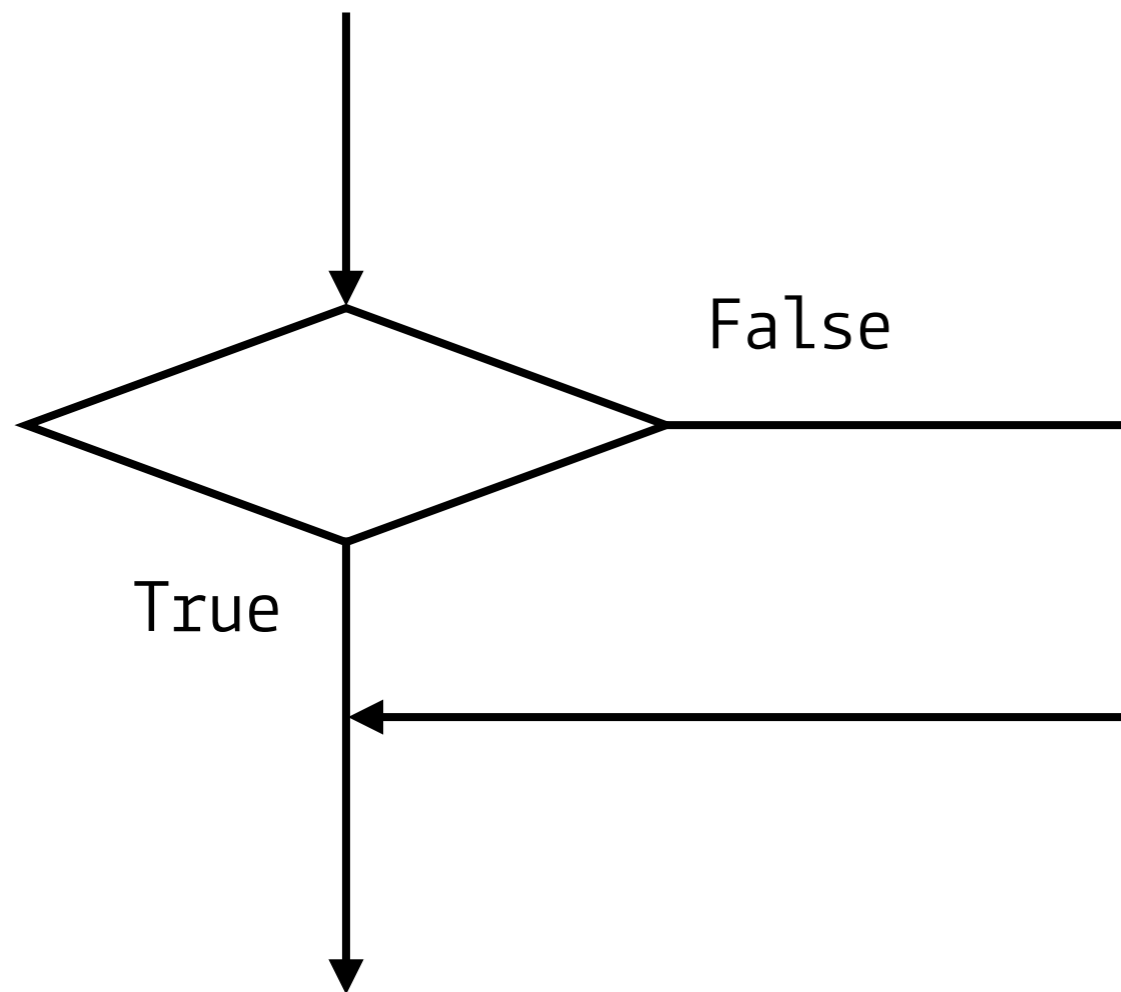
❖ 演算と代入を同時に行う演算子（演算した結果を代入する）

演算子	使用例	意味
<code>+=</code>	<code>i += 1</code>	<code>i = i + 1</code>
<code>-=</code>	<code>i -= 2</code>	<code>i = i - 2</code>
<code>*=</code>	<code>i *= 3</code>	<code>i = i * 3</code>
<code>/=</code>	<code>i /= 4</code>	<code>i = i / 4</code>
<code>%=</code>	<code>i %= 5</code>	<code>i = i % 5</code>
<code>**=</code>	<code>i **= 6</code>	<code>i = i ** 6</code>

条件分岐

条件分岐

- ❖ 論理式を条件式とみなし，if文を利用することで，条件によって処理を分けるプログラムを作成可能
- ❖ 条件式は常にひとつの真偽値(TrueかFalse)に評価される



条件分岐の例：

80点以上ならスコアAと表示

700円以上ならくじを引く

20km以上40km未満なら中距離

コードブロック

- ❖ 条件分岐処理は1行以上のひとまとまりのコードブロックで表現
- ❖ Pythonでは、コードを字下げ（インデント）することでコードブロックを作成可能
 - ❖ 字下げ：行の先頭にいくつかのスペースを入力すること
 - ❖ Pythonは字下げを判断して、プログラムを実行
 - ❖ ブロック化がきちんとしていない場合、プログラムが意図した動作をしない（エラーもなる）
 - ❖ 「Tab」キーを押すことで、指定したスペース数を入力する
 - ❖ 日本語の全角スペースを入力しないように注意

if文

- ❖ ifの条件式がTrueの場合, if文に続くブロック内の処理が実行
- ❖ 条件式がFalseの場合, ブロック内の処理はスキップ

```
if 条件式:  
    # Trueのときの処理
```

```
x = input()  
if x % 2 == 0:  
    print('Even number')
```

【if文の記述方法】

コロン(:)を忘れずに記述する

偶数判定プログラム例

コードブロックの違いによる変化

- ❖ ifの条件式がTrueの場合, if文に続くブロック内の処理が実行
- ❖ 条件式がFalseの場合, ブロック内の処理はスキップ

```
x = 0
if x == 1:
    print('A')
print('AA')
```

```
x = 0
if x == 1:
    print('A')
    print('AA')
```

AA # 実行結果
コードブロック1行と判断

表示無しの実行結果
コードブロック2行と判断

if-else文

- ❖ if-else文は、ifの条件に応じて実行される処理が異なる
- ❖ 条件式がTrueの場合、if文に続くブロック内の処理が実行
- ❖ 条件式がFalseの場合、else文に続くブロック内の処理が実行

```
if 条件式:  
    # Trueのときの処理  
else:  
    # Falseのときの処理
```

【if-else文の記述方法】

コロン(:)を忘れずに記述する

```
x = input()  
if x % 2 == 0:  
    print('Even number')  
else:  
    print('Odd number')
```

偶数奇数判定プログラムの例

if-elif-else文

- ❖ elif文は、複数の条件分岐（if, else以外の処理）で使用する
 - ❖ C言語などの「else if」に相当する
 - ❖ if文や他のelif文に続けて記述する

```
if 条件式1:  
    # 条件式1がTrueのときの処理  
elif 条件式2:  
    # 条件式2がTrueのときの処理  
else:  
    # 条件式1と条件式2が両方Falseのときの処理
```

【if-elif-else文の記述方法】

コロン(:)を忘れずに記述する

ここまでのまとめ演習2

- ❖ 整数を入力して、その値が正の値かどうかを判定するプログラム (q2-1.py) を作成せよ
 - ❖ 5を入力すると「Positive」、
0を入力すると「Not positive」と出力する
- ❖ 西暦（整数）を入力して、その西暦がうるう年かどうかを判定するプログラム (q2-2.py) を作成せよ
 - ❖ 「400で割り切れる年はうるう年である」、
「100で割り切れて、400で割り切れない年はうるう年ではない」、
「4で割り切れて、100で割り切れない年はうるう年である」
 - ❖ 2000を入力すると「Leap year」、
2001を入力すると「Not leap year」と出力する

q2-1.py解答例

```
num = input()
if(num > 0):
    print('Positive')
else:
    print('Not positive')
```

```
python q2-1.py
5                # キーボード入力
Positive         # 出力結果

python q2-1.py
0                # キーボード入力
Not positive    # 出力結果
```


q2-2.py解答例

```
year = input()
if(year % 400 == 0 or (year % 4 == 0 and year % 100 != 0)):
    print('Leap year')
else:
    print('Not leap year')
```

```
python q2-2.py
2020          # キーボード入力
Leap year     # 出力結果

python q2-2.py
2018          # キーボード入力
Not leap year # 出力結果
```

メソッド

メソッドの定義

- ❖ いくつかの処理が機能として1つにまとまっているもの
- ❖ 『**def** **メソッド名**():
』と記述し、処理をブロック内に記述- ❖ メソッド名は自由に命名できるが、どのような処理をするメソッドなのかがわかるように命名する方が良い
- ❖ メソッド名を使用することを『メソッドを呼び出す』といい、『**メソッド名**()』と記述することで呼び出すことができる
- ❖ プログラムの実行がメソッド呼び出しに到達すると、そのメソッドの先頭行に移動して、メソッド内の処理が終わるとメソッド呼び出し元の行に戻り、後続処理を実行

引数ありメソッドの定義

- ❖ メソッドを作成する際に、メソッド定義の丸括弧()内に、変数名を記述することで、引数ありメソッドを作成可能

```
# -*- coding: utf-8 -*-  
  
# Helloを出力するprintHelloメソッド  
def printHello(name):  
    # メソッド内で変数nameを使用可能  
    print('Hello, ' + name)  
  
n = raw_input() # 文字列を入力  
printHello(n)   # メソッドの呼び出し
```

戻り値のあるメソッドの定義

- ❖ メソッドで処理した結果を呼び出し元で使用したい場合,
return文を使用 (この値をメソッドの戻り値と呼ぶ)

```
# -*- coding: utf-8 -*-  
  
# 2つの引数の和を返すgetAdd2メソッド  
def getAdd2(num1, num2):  
    return num1 + num2 # 和を返す  
  
a = input()  
b = input()  
print(getAdd2(a, b)) # メソッドの呼び出し
```

ここまでのまとめ演習3

- ❖ 身長と体重を引数に与え、BMIの値を返すメソッドを作成
 - ❖ プログラム名： `q3.py`
 - ❖ メソッド名： `calc_BMI(height, weight)`
- ❖ 変数 `height` は身長[m]の値を、 `weight` は体重[kg]の値を格納する
- ❖ BMIの値は体重を身長の2乗で割ることで求められる

q3.py解答例

```
# -*- coding: utf-8 -*-  
  
def calc_BMI(height, weight):  
    return weight / (height ** 2)  
  
h = input() # 身長(単位は[m])  
w = input() # 体重(単位は[kg])  
  
print(calc_BMI(h, w))
```

```
python q3.py  
  
1.7          # 身長  
65           # 体重  
22.491349481 # BMI出力結果
```

OpenRTM-aistの 開発の流れ

OpenRTM-aist開発の流れ

コンポーネントの仕様を決める

RTCBuilderで雛形生成

コアロジックを雛形に組み込む

RTMで実行する

コンポーネント名,
データの入出力,
使用言語

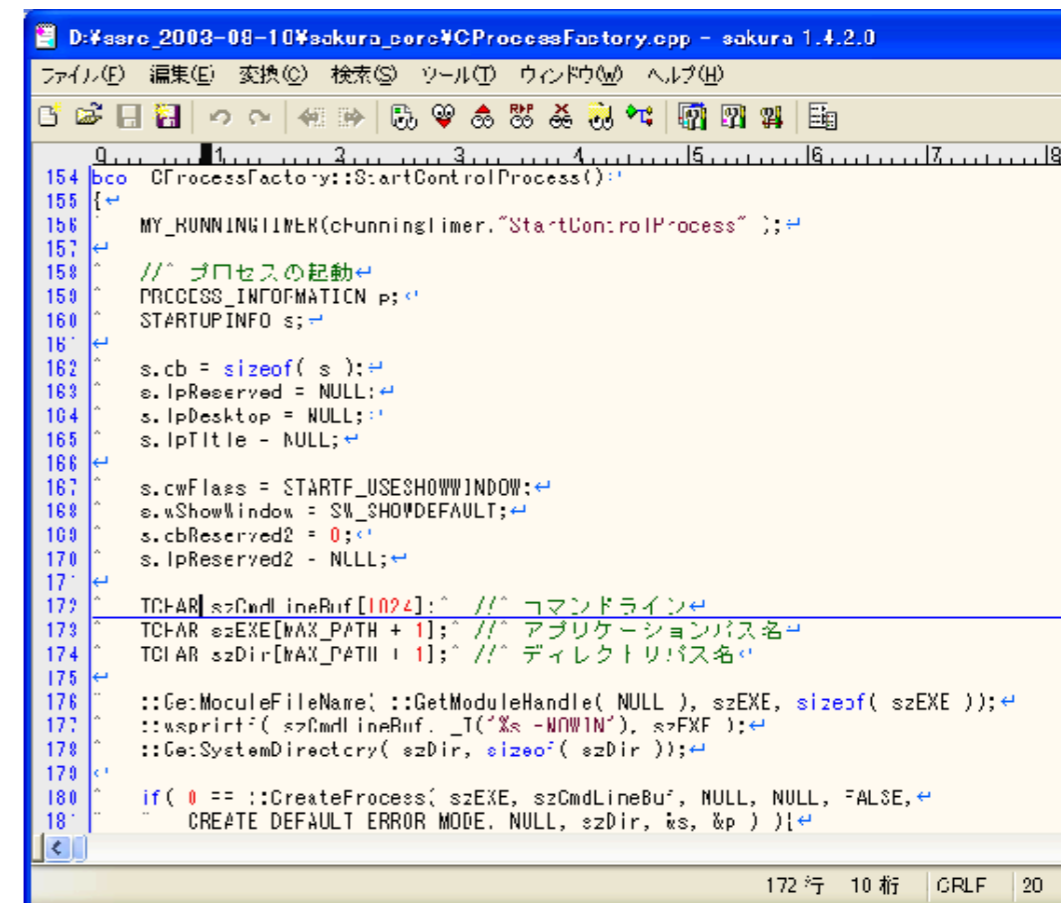
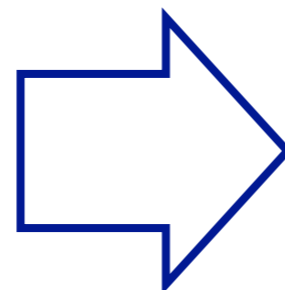
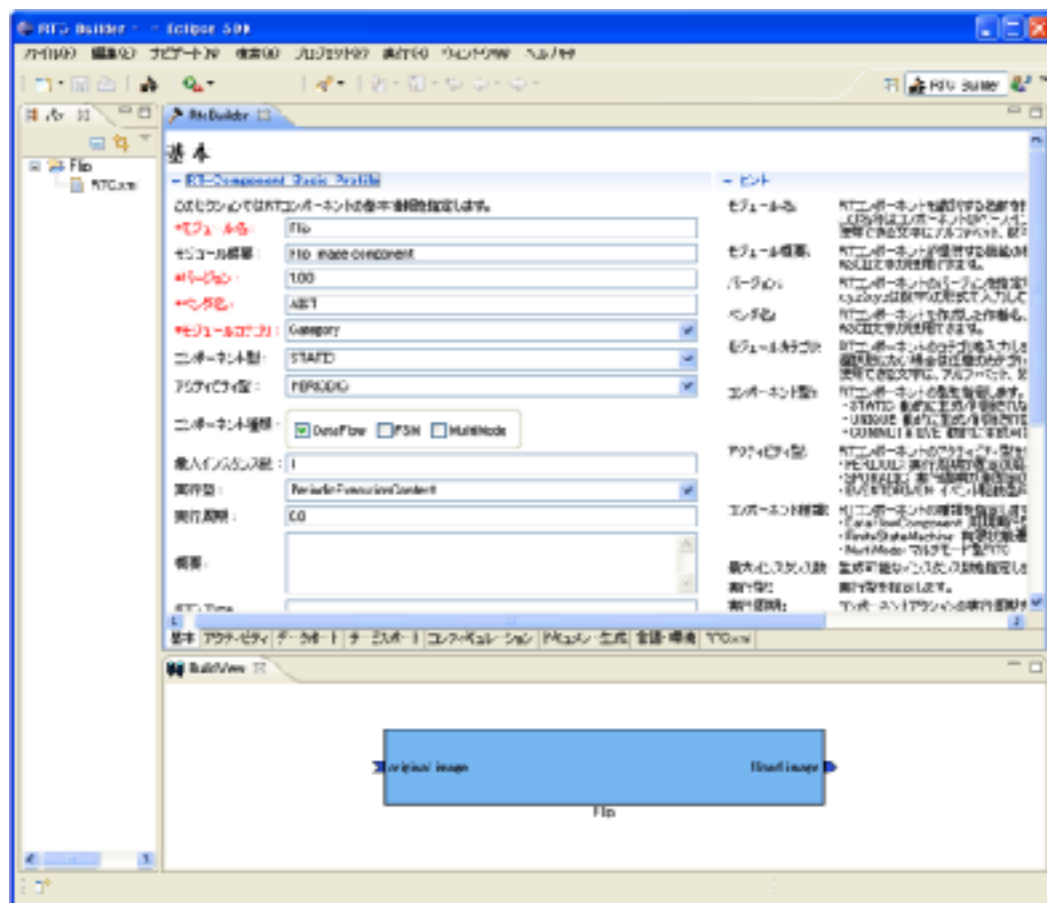
RTC開発者が
プログラムを書き込む

雛形とコアロジック



RTC雛形 + コアロジック = RTコンポーネント

RTC作成例 | この講習会の場合



RTCBuilderで、
コンポーネントを設定し、
Pythonの雛形を生成

テキストエディタで、
Pythonの雛形コードに
ロジックを書き込む

RTコンポーネント作成の仕方

RTC雛形の作成

❖ RTCBuilder

❖ RTCフレームワークを作成するためのソフトウェア

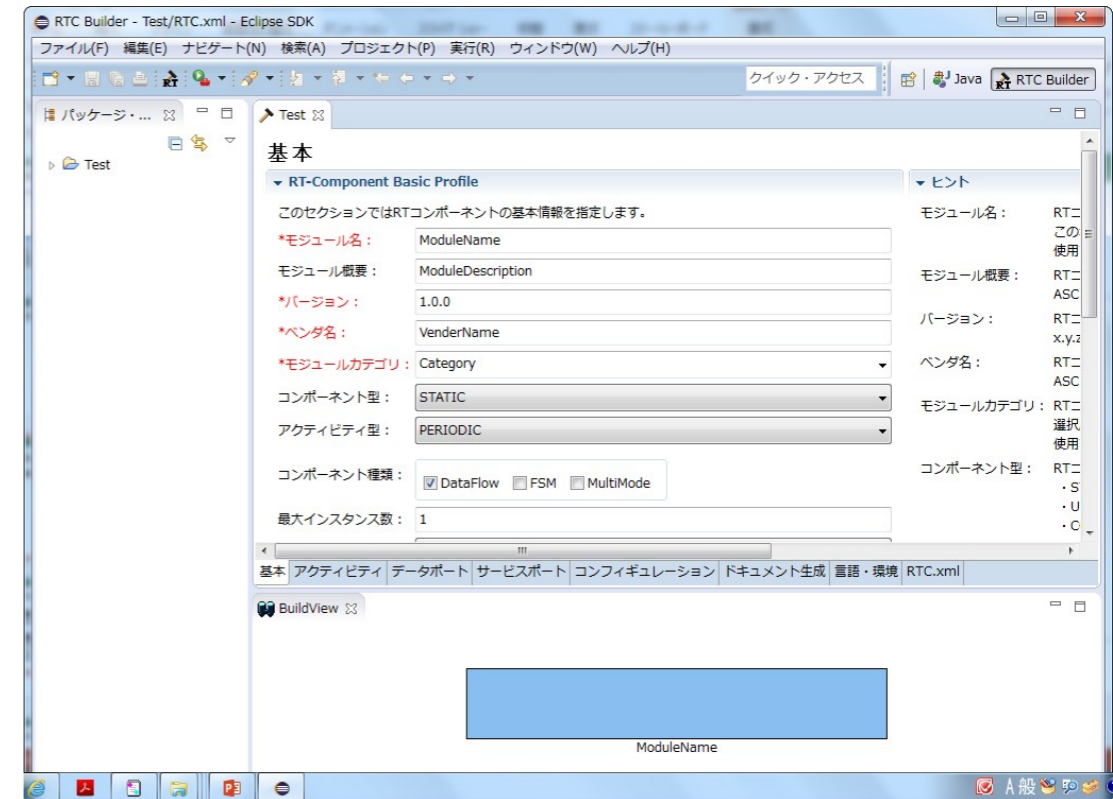
❖ コンポーネント名

❖ 使用コールバックメソッド

❖ データポート名

❖ コンフィギュレーション

❖ 使用言語



RTCBuilder | 基本

❖ コンポーネントの基本情報を入力

Test ☒

基本

▼ RT-Component Basic Profile

このセクションではRTコンポーネントの基本情報を指定します。

*モジュール名:

モジュール概要:

*バージョン:

*ベンダ名:

*モジュールカテゴリ:

コンポーネント型:

アクティビティ型:

コンポーネント種類: DataFlow FSM MultiMode

最大インスタンス数:

▼ ヒント

モジュール名: RT2
この使用

モジュール概要: RT2
ASC

バージョン: RT2
x.y.z

ベンダ名: RT2
ASC

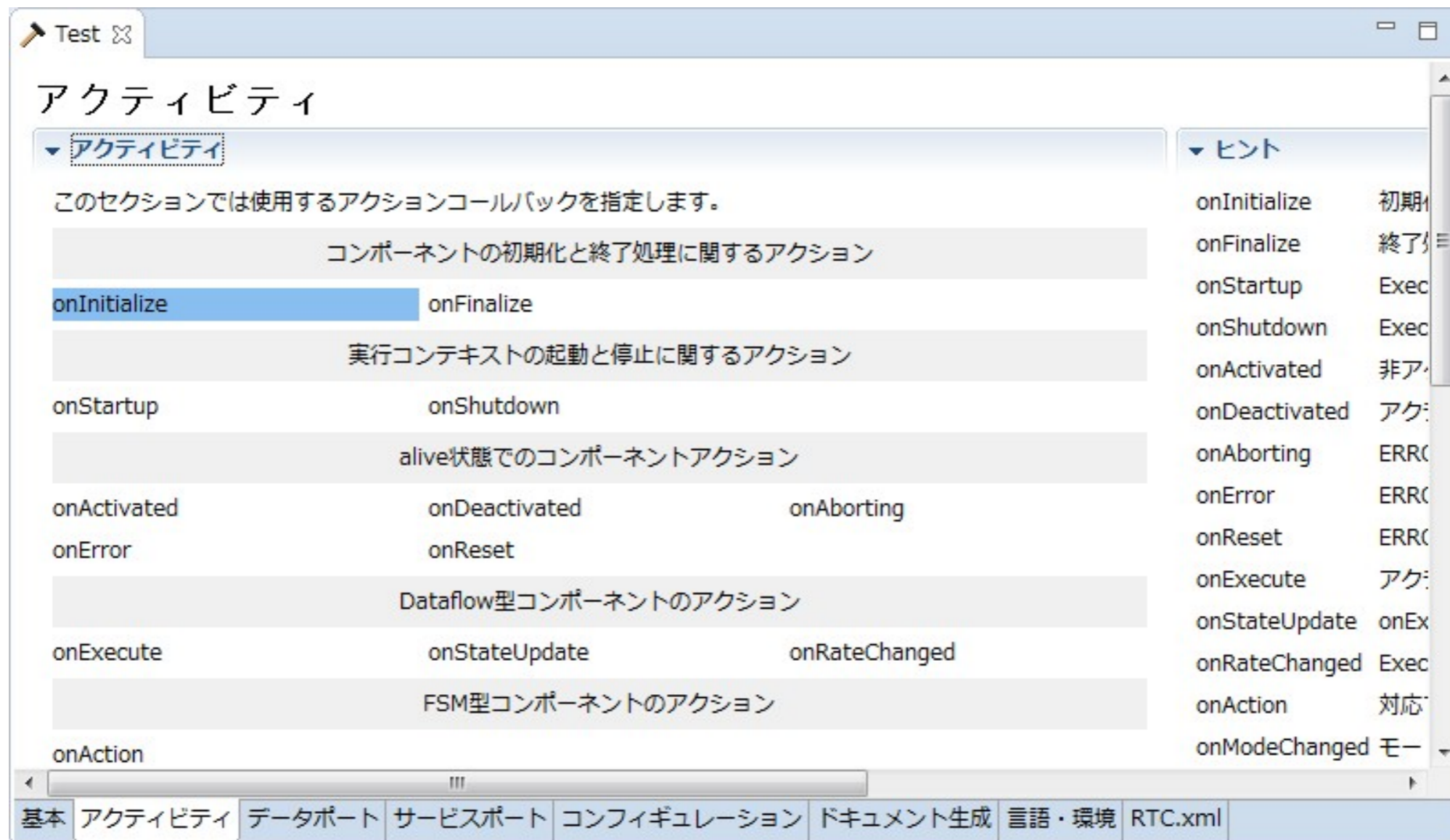
モジュールカテゴリ: RT2
選択使用

コンポーネント型: RT2
・S
・U
・C

基本 | アクティビティ | データポート | サービスポート | コンフィギュレーション | ドキュメント生成 | 言語・環境 | RTC.xml

RTCBuilder | アクティビティ

❖ コンポーネント内で使用するメソッドを選択



プログラムの種類

❖ フロー駆動型

- ❖ プログラム記述通りにプログラムを上から1つずつ実行

❖ イベント駆動型プログラム

- ❖ イベントに対して実行される内容が変化する

- ❖ 「キーボードで入力した」、「マウスホイール動かした」、「マウスをクリックした」などの操作がトリガーとなり発生する事象のこと

❖ **OpenRTM-aistはイベント駆動型プログラミング**

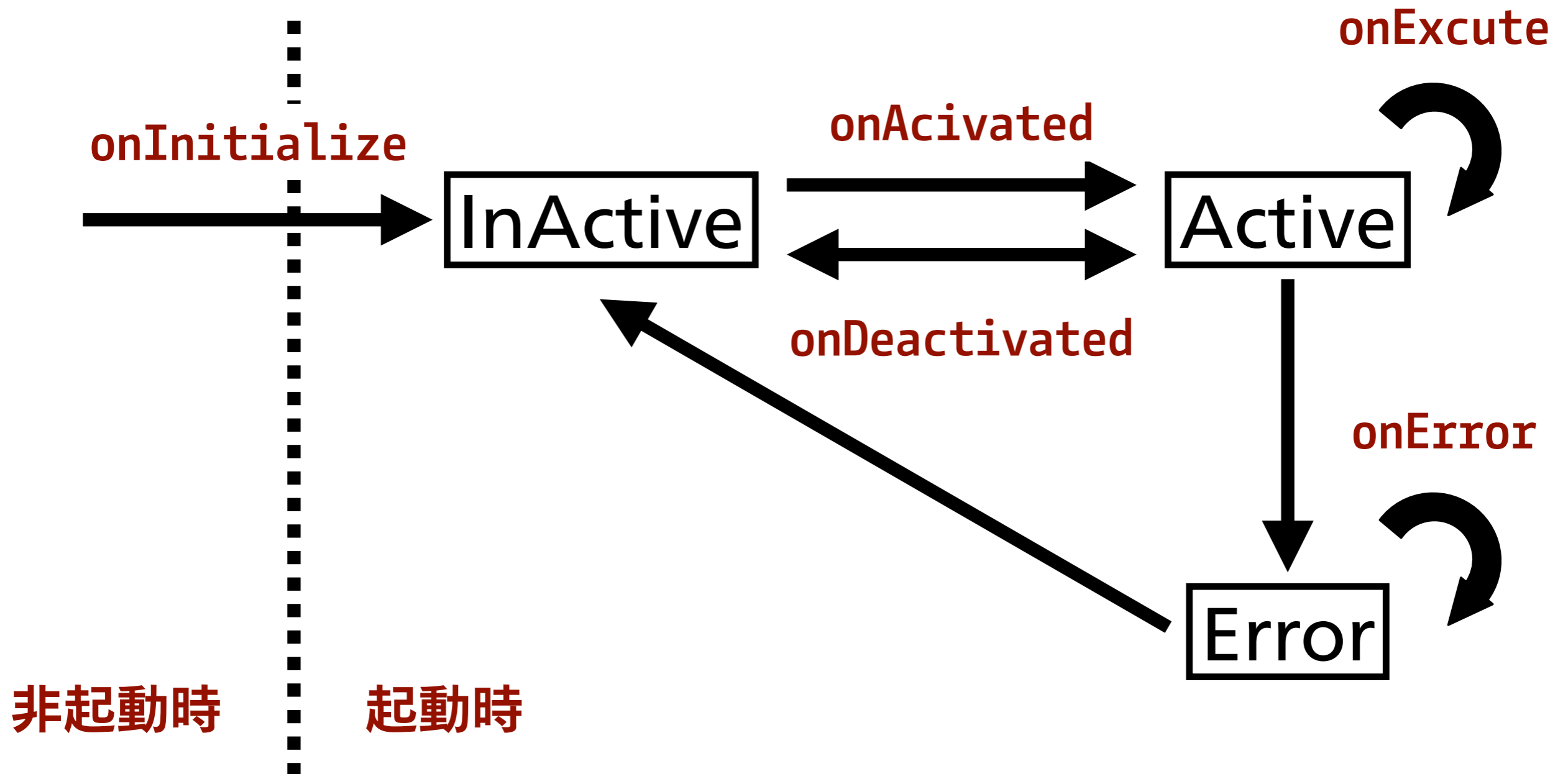
OpenRTM-aistの場合

- ❖ コールバックメソッドによるイベント駆動型プログラム
 - ❖ プログラム中の以下のメソッド内にプログラムを記述

メソッド名	処理内容
onInitialize	コンポーネント起動時に実行し、初期化処理をする
onActivated	コンポーネント実行時に1度だけ実行する
onExcute	コンポーネント実行時に周期的に実行する
onDeactivated	コンポーネント停止時に1度だけ実行する
onError	コンポーネントがERROR状態のときに周期的に実行する

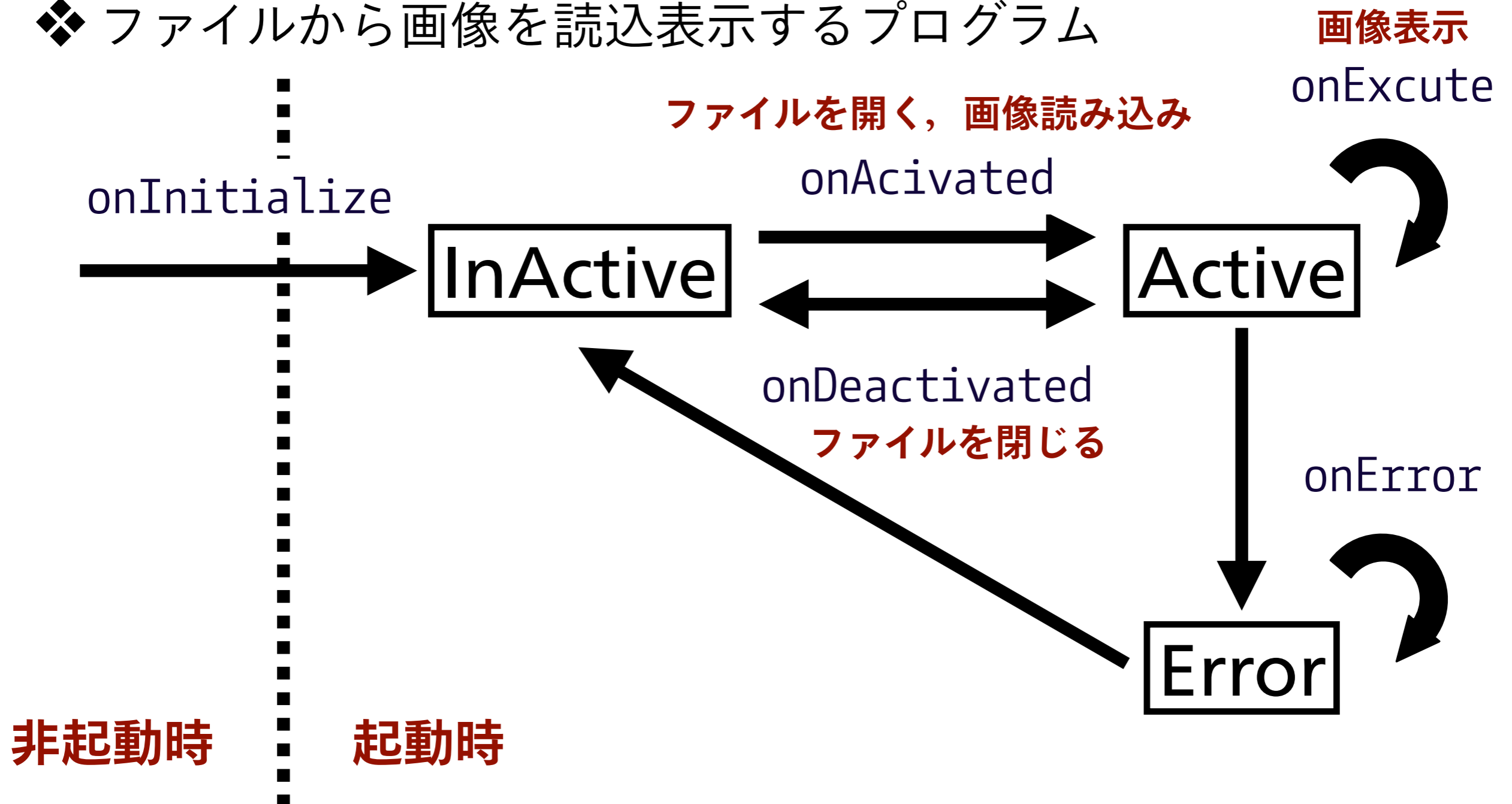
OpenRTM-aistの場合

❖ コールバックメソッドの起動タイミング



OpenRTM-aistの場合

- ❖ ファイルから画像を読み込表示するプログラム

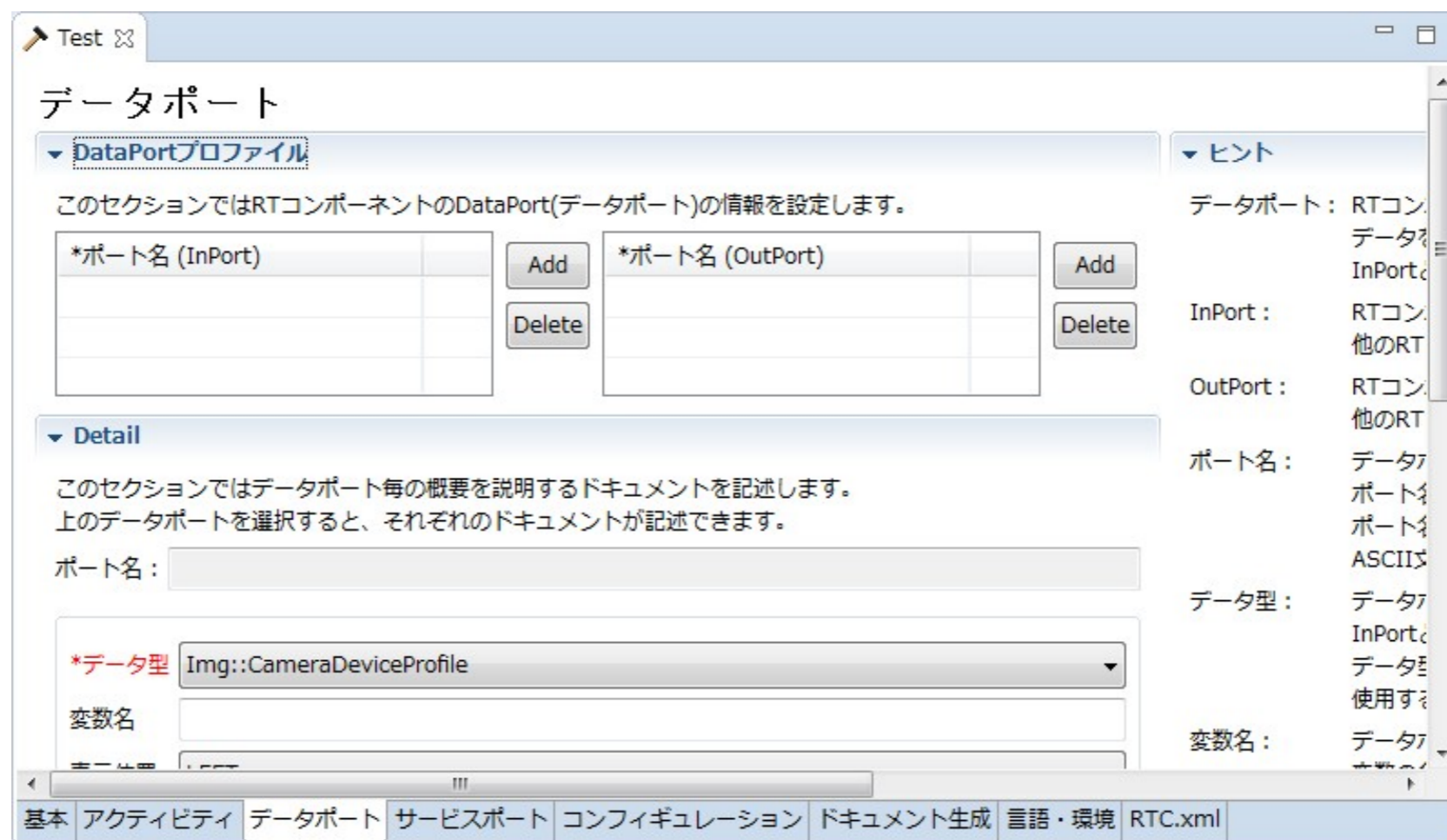


RTCBuilder | データポート

❖ コンポーネントのInPortとOutPortを設定



ポート名, データ型, 変数名



RTCBuilder | データポート

- ❖ データポートで宣言した変数とポート名は Python プログラムでは以下の名前で利用できる

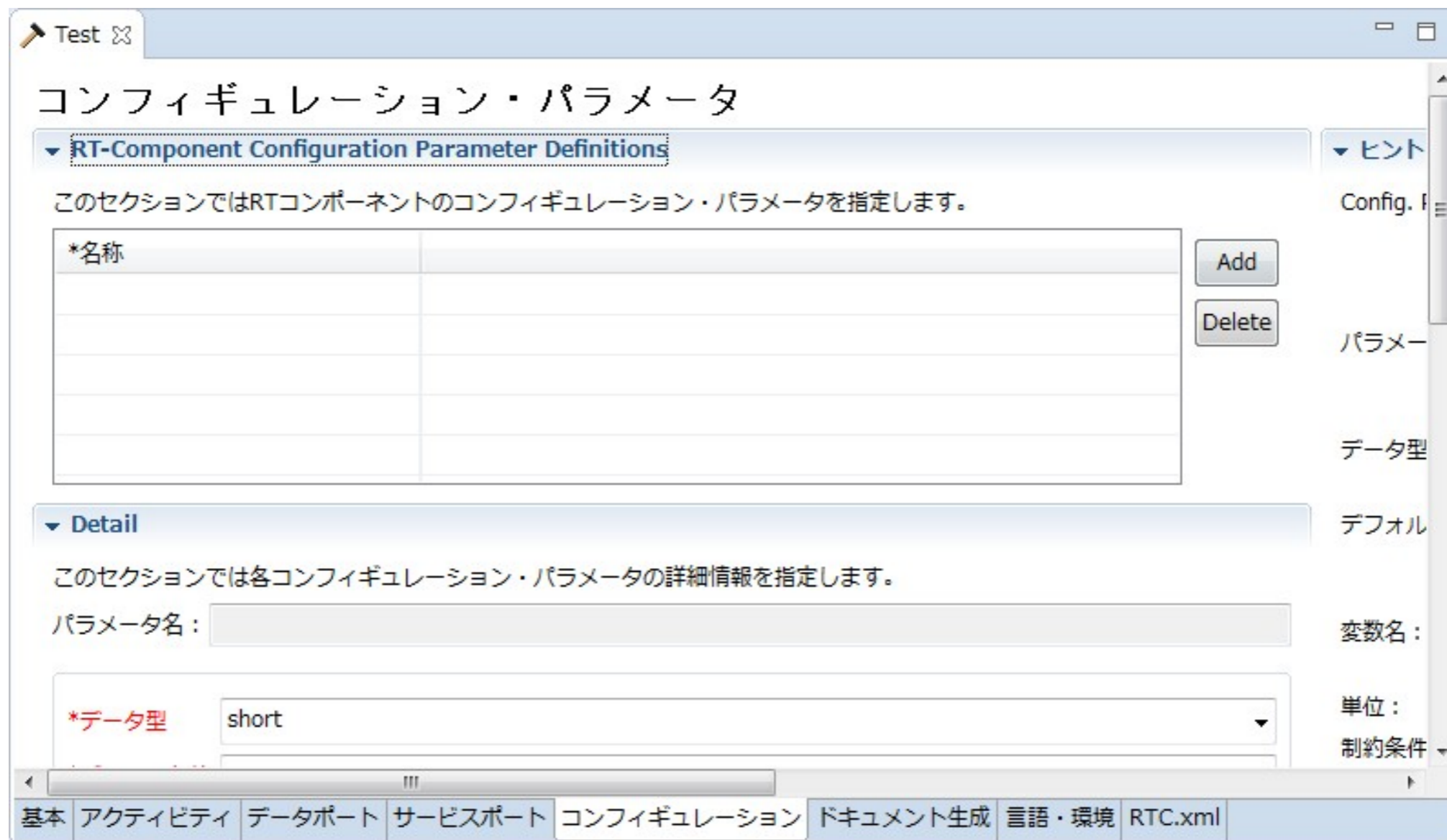
InPortポート名	self._ポート名In
InPort変数名	self._d_変数名
OutPortポート名	self._ポート名Out
OutPort変数名	self._d_変数名

- ❖ 例として以下のようなになる

OutPortポート名 : Vel	self._VelOut
OutPort変数名 : Value	self._d_Value

RTCBuilder | コンフィギュレーション

❖ RTSystemEditorで変更可能な外部パラメータの設定



RTCBuilder | コンフィギュレーション

❖ FlipCompで使ったコンフィギュレーション

The screenshot shows the configuration window for the 'FlipComp0' component. The 'flipMode' property is set to 1. A text overlay indicates that the value is being changed from the external RTSystemEditor.

RTSystemEditor上 (外部) から
コンポーネント内のプログラム内の値を変更している

Type Name	FlipCor
Description	FlipCor
Version	1.0.0

ConfigurationSet : default

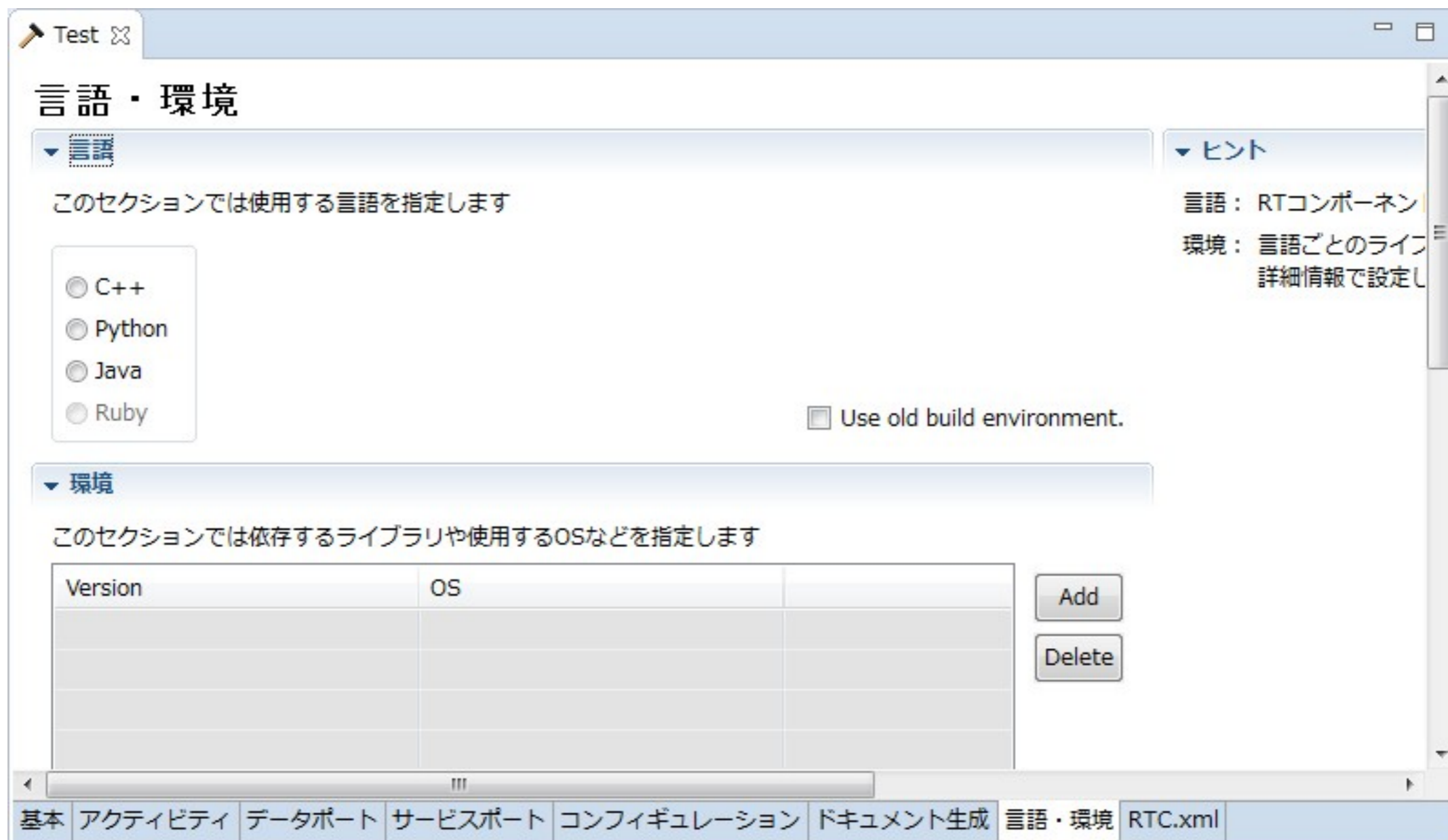
flipMode -1 0 1

Apply

OK キャンセル

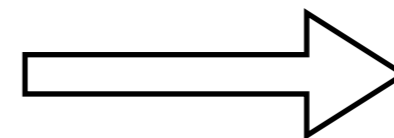
RTCBuilder | 言語・環境

❖ コンポーネントの使用プログラミング言語を設定

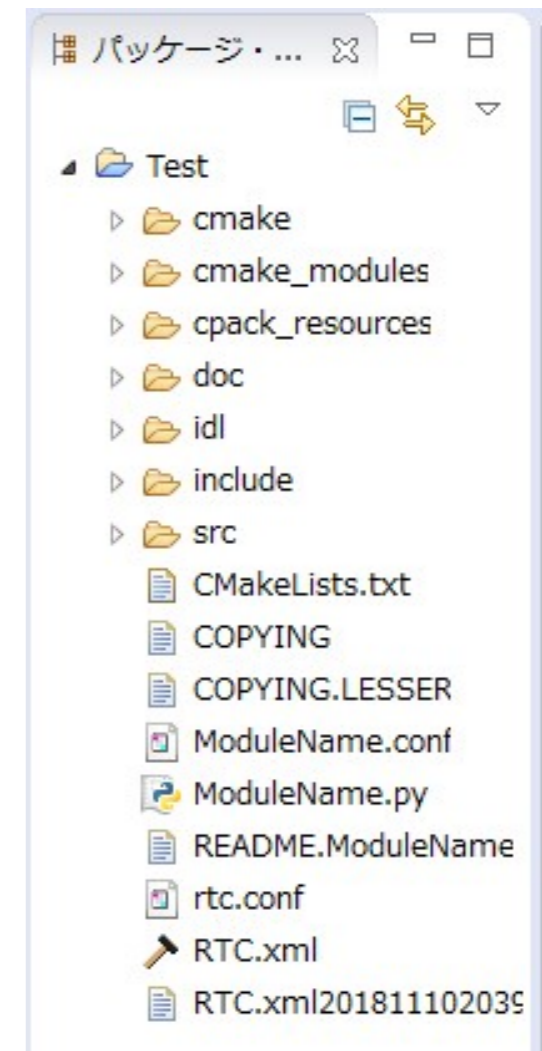


RTCBuilder | 基本

- ❖ 基本タブのコード生成を押すと入力した情報によりRT雛形が生成される

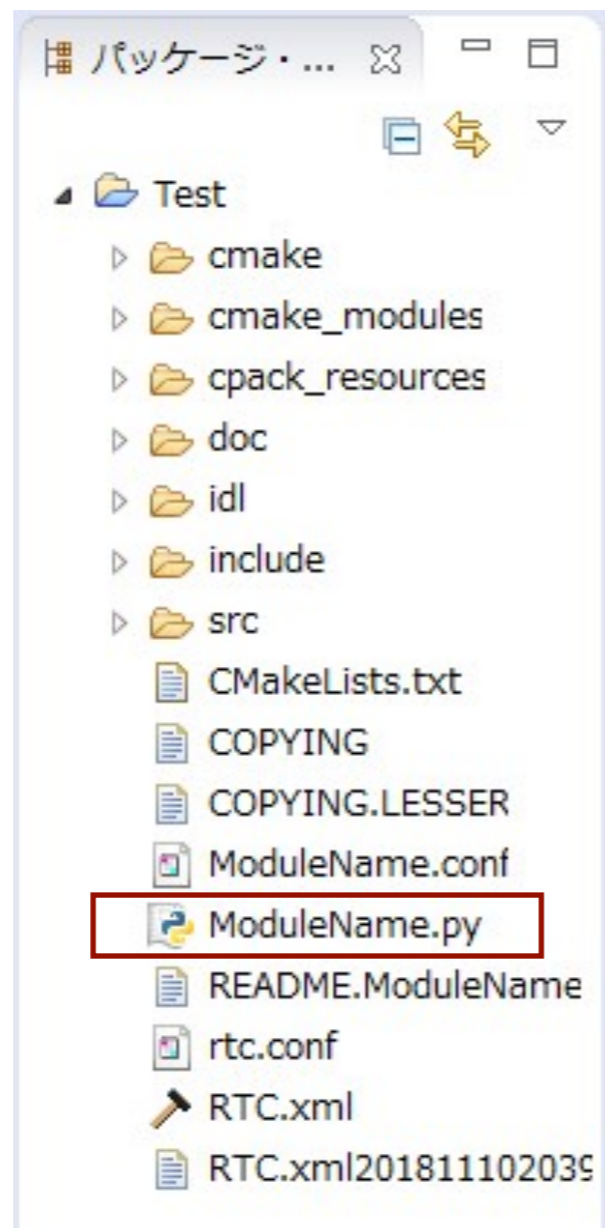


RT雛形生成



Pythonファイルを編集

- ❖ 基本タブのコード生成を押すと入力した情報によりRT雛形が生成される



生成されたPythonファイルを
エディターで開いて編集をする

Pythonファイルの編集場所

- ❖ RTCBuilderで使用を選択したメソッドはコメントが外れているのでその中に処理を追加する

```
def onActivated(self, ec_id):  
    ## この部分にプログラムを追加 ##  
    return RTC.RTC_OK
```

```
def onDeactivated(self, ec_id):  
    ## この部分にプログラムを追加 ##  
    return RTC.RTC_OK
```

```
def onExecute(self, ec_id):  
    ## この部分にプログラムを追加 ##  
    return RTC.RTC_OK
```

他のRTコンポーネントから受取

❖ 例) FlipComp

```
# InPortに値があるか確認
```

```
if self._original_imageIn.isNew():
```

```
    data = self._original_imageIn.read() # 値読み込み
```

❖ `isNew()` : 新しい値を受信しているか確認する

❖ `read()` : 新しい値を読み込む

❖ 読み込んだ値を戻り値として格納する

❖ `read()`の前には、必ず`isNew()`で値を受信しているかを確認する必要がある (値がない状態で`read()`した場合、値が存在しないためエラーになる)

他のRTコンポーネントへの値渡し

❖ 例) FlipComp

```
self._fliped_imageOut.write()
```

❖ write() : OutPortの変数を出力



EV3について

LEGO MINDSTORMS EV3

- ❖ マサチューセッツ工科大学と共同開発されたロボティクス製品
 - ❖ 教育用二輪移動ロボット
 - ❖ ジャイロ，カラー，タッチセンサなど多くのセンサを持つ
 - ❖ Java, C++, Pythonなど多くの言語で動かすことが可能



EV3活用事例

❖ 教育

❖ 企業

❖ 教育研修

❖ 教育機関

❖ プログラミング教育

❖ ロボットコンテスト

❖ ETロボコン

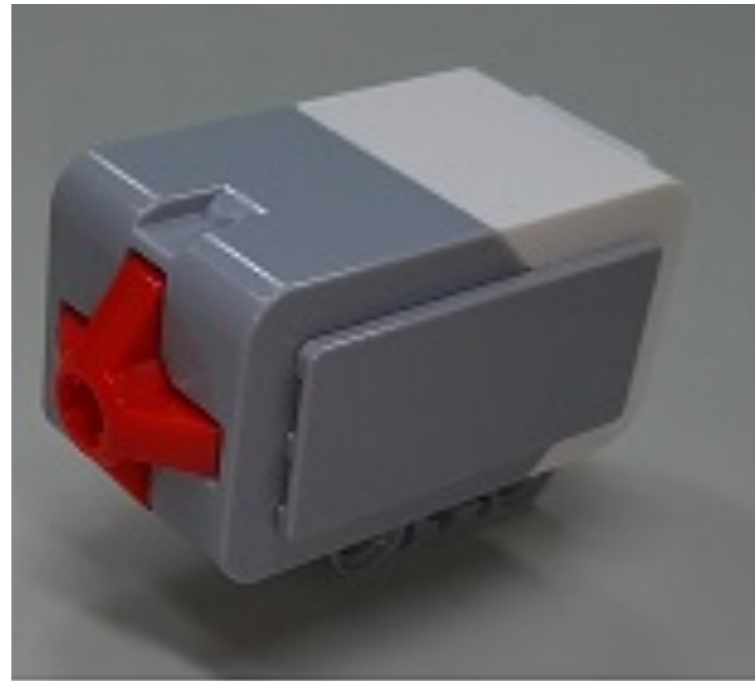
❖ World Robot Olympiad



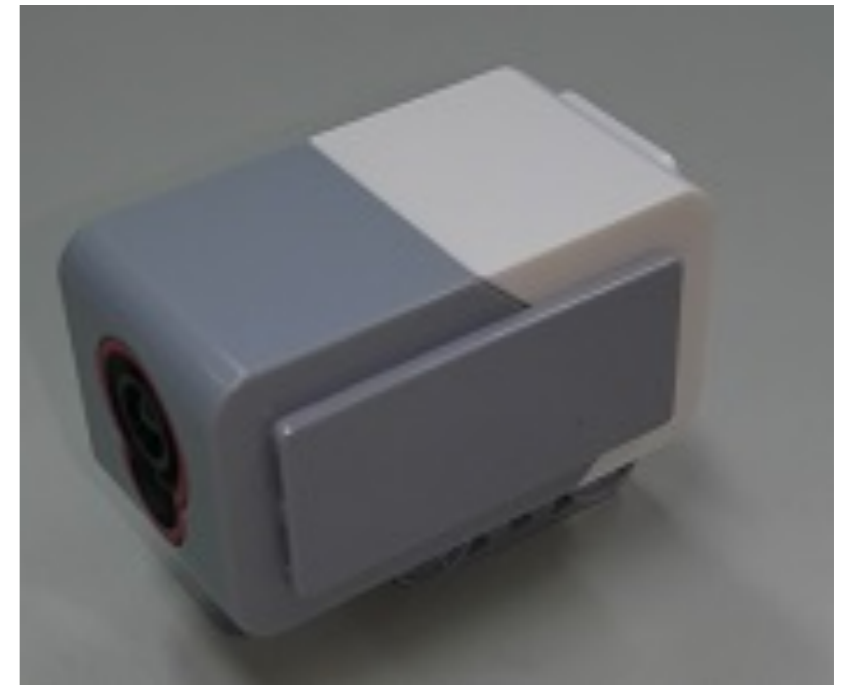
センサ説明



超音波センサ



タッチセンサ



カラーセンサ

超音波センサ

❖ 超音波を発信し, その反射波を読み取るまでに要した時間により距離を測定するセンサ

❖ 性能

❖ 距離計測可能範囲 : 3cmから250cm

❖ 距離計測精度 : $\pm 1\text{cm}$



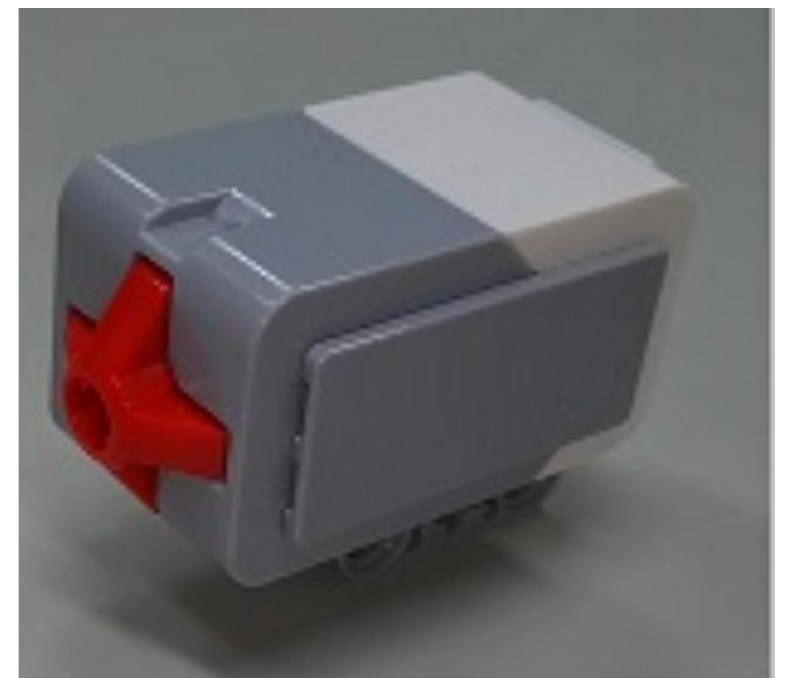
タッチセンサ

❖ 前面にあるボタンが押されたかどうかを検知するセンサ

❖ 性能

❖ スイッチオン：1，オフ：0

❖ スイッチ可動域：約4mm



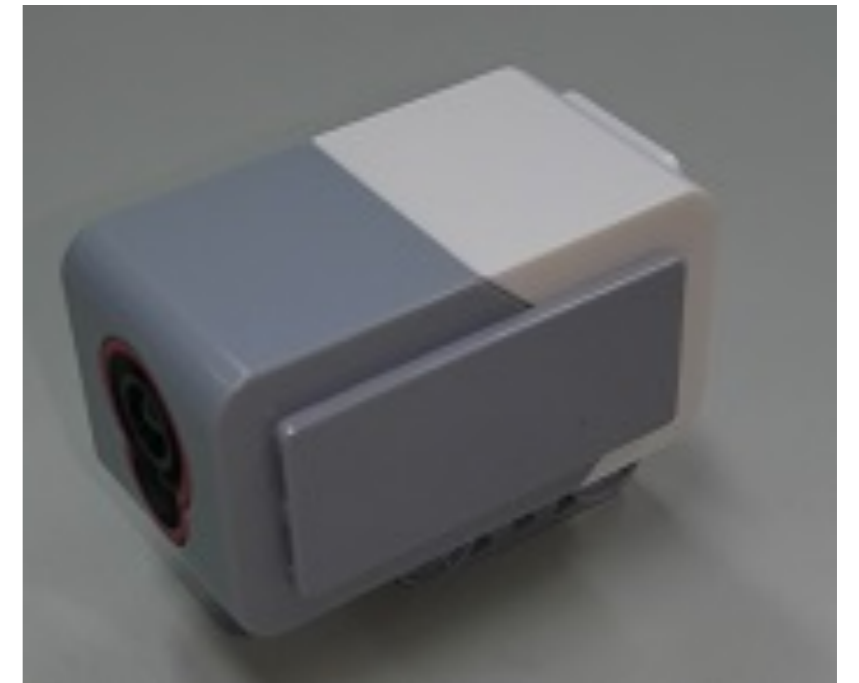
カラーセンサ

❖ 表面の色を検知することが出来るセンサ

❖ 性能

❖ 検出カラー数：

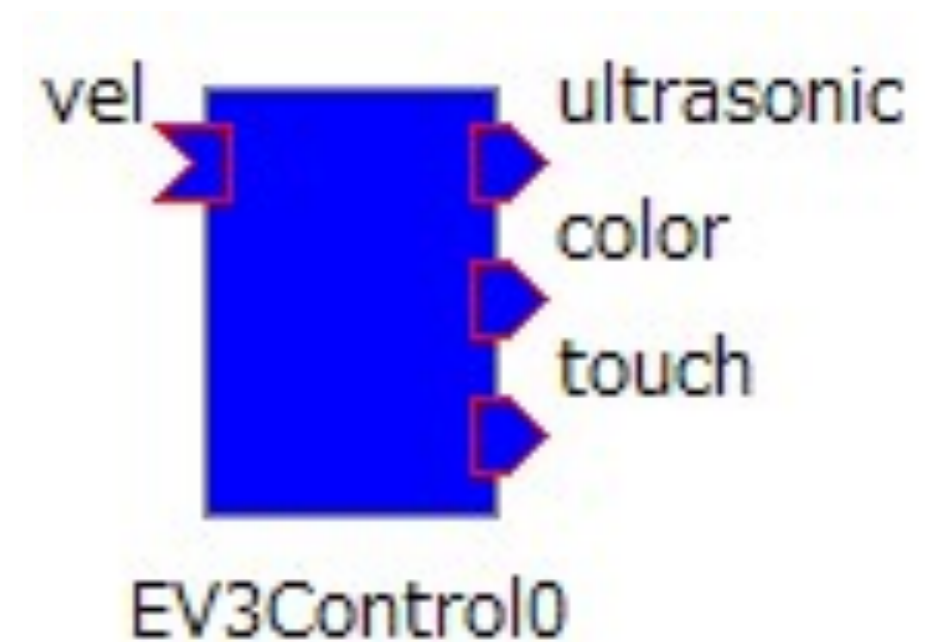
❖ 8色（無色，黒，青，緑，黄，赤，白，茶）



OpenRTM-aistによる操作

EV3制御用コンポーネント概要

- ❖ モーターの制御やセンサーの値を取得
- ❖ InPort
 - ❖ EV3の速度の値を入力
 - ❖ この値によってEV3のモーターが動く
- ❖ OutPort
 - ❖ EV3に接続されたセンサーの値を出力



InPort(vel)

- ❖ 入力vel, データ型 : RTC.TimedVelocity2D
- ❖ 速度の値を受け取る
- ❖ EV3はこの値を使用し, モータを回転させることによって移動
- ❖ 入力する速度はメートル単位で入力
- ❖ 値を入力するとその値でモータを回転し続ける

RTC.TimedVelocity2Dについて

❖ RTC.TimedVelocity2Dは構造体で以下の変数を持つ

型名	変数名	意味
RTC.Velocity2D	data	速度データ
RTC.Time	tm	タイムスタンプ ※今回は使用しません

❖ RTC.Velocity2Dは構造体で以下の変数を持つ

型名	変数名	意味
Double	vx	並進速度（前方）[m/s]
Double	vy	並進速度（横方）[m/s]
Double	va	角速度[rad/s]

RTC.TimedVelocity2Dの使い方

❖ 以下の様に値を代入する

```
self._d_vel.data.vx = 0.04
```

```
self._d_vel.data.vy = 0.01
```

```
self._d_vel.data.va = 0.02
```

OutPort(ultrasonic)

- ❖ 出力ultrasonic, データ型 : RTC.RangeData
- ❖ EV3に接続された超音波センサーの値を出力
- ❖ 障害物までの距離を測定
- ❖ 出力される値はメートル単位で出力

RTC.RangeDataについて

❖ RTC.TimedVelocity2Dは構造体で以下の変数を持つ

型名	変数名	意味
sequence<double> RangeList	ranges	距離の値[m]
RangerGeometry	geometry	スキャンデータが測定されたときのレンジャーの形状（※今回は使用しません）
RangerConfig	config	スキャンデータが測定されたときのレンジャーの設定（※今回は使用しません）
RTC.Time	tm	タイムスタンプ（※今回は使用しません）

❖ 使用する場合、以下の様に使用する

```
self._d_変数名.ranges[0] # double型
```

OutPort(color)

- ❖ 出力color, データ型 : RTC.TimedString
- ❖ EV3に接続されたカラーセンサーの値を出力
- ❖ 出力される値は0~7の値で出力

色	無色	黒	青	緑	黄	赤	白	茶
数値	0	1	2	3	4	5	6	7

RTC.TimedString

❖ RTC.TimedStringは構造体で以下の変数を持つ

型名	変数名	意味
string	data	文字データ
RTC.Time	tm	タイムスタンプ (※今回は使用しません)

❖ 使用する場合、以下の様に使用する

```
if self._d_color.data == "2": # 以下処理を記述
```

OutPort(touch)

- ❖ 出力touch, データ型 : RTC.TimedBooleanSeq
- ❖ EV3に接続されたタッチセンサーの値を出力
- ❖ 値はbool型の配列で, 0番目の要素に左のタッチセンサ, 1番目の要素に右のタッチセンサの値が格納

RTC.TimedBooleanSeq

❖ RTC.TimedBooleanSeqは構造体で以下の変数を持つ

型名	変数名	意味
sequence<boolean>	data	True(1)かFalse(0)が格納されている 配列データ
RTC.Time	tm	タイムスタンプ (※今回は使用しません)

❖ 使用する場合、以下の様に使用する

```
if self._d_touch.data[0] == 1: # 以下処理を記述
```

EV3の制御方法

EV3の制御

❖ EV3用コンポーネントに入力を与えるとEV3が動作する

2つの入力

V_x : 前に進む速度 [m/s]

V_a : 曲がろうとする力 [rad/s]



3つの出力

超音波センサ

カラーセンサ

タッチセンサ

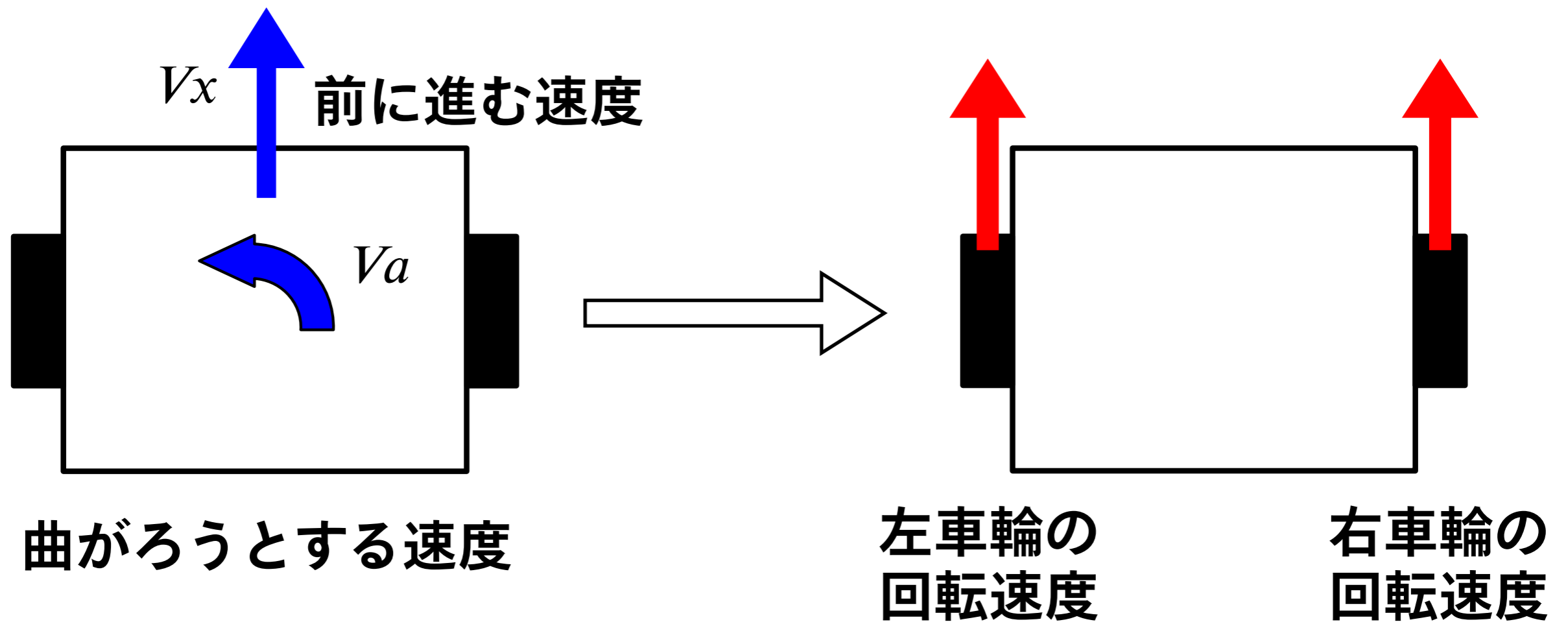
速度から車輪の
回転速度を計算し、
モータを回転させる

左車輪の回転速度
右車輪の回転速度

回転速度により
EV3が動作する



EV3の制御

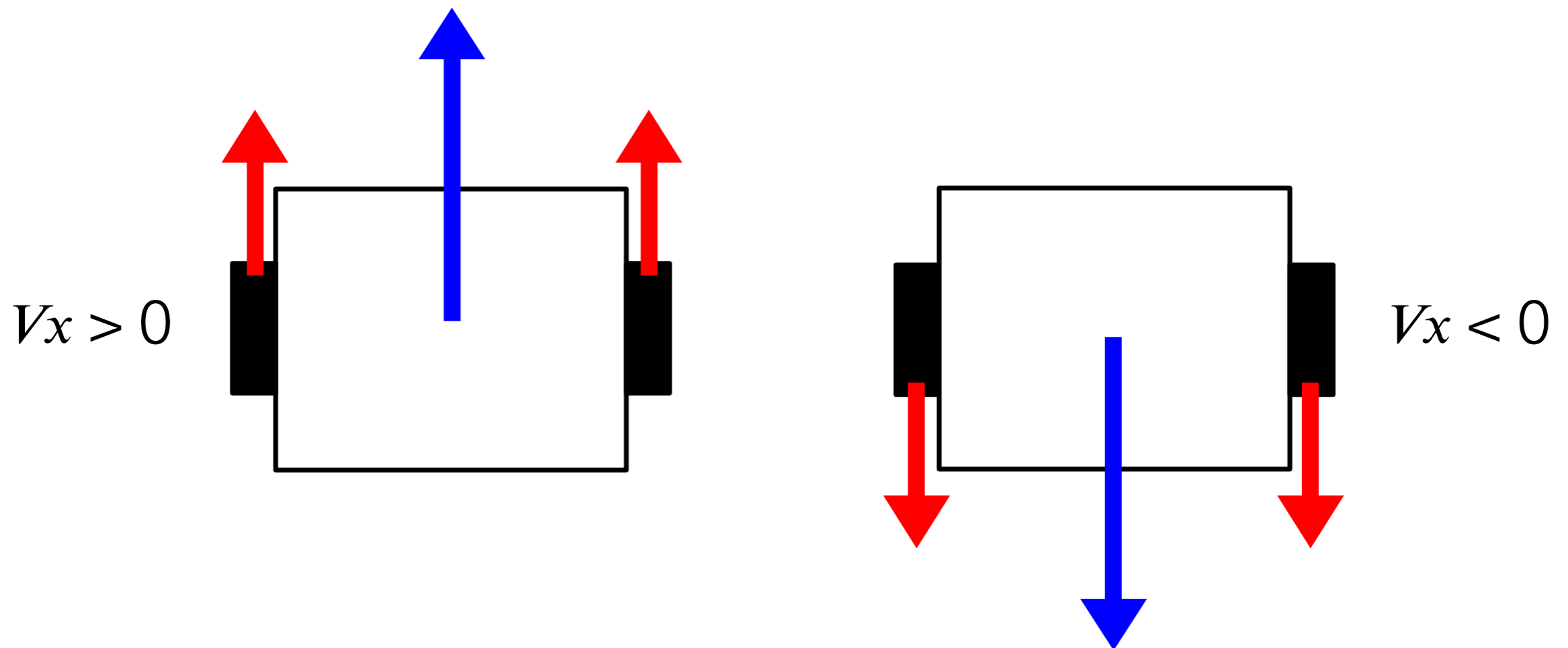


EV3の制御 | $v_a = 0$ のとき

❖ V_x : 前に進む速度

❖ 値がプラスのとき, 両車輪は前に回転 : EV3は前進

❖ 値がマイナスのとき, 両車輪は後ろに回転 : EV3は後退

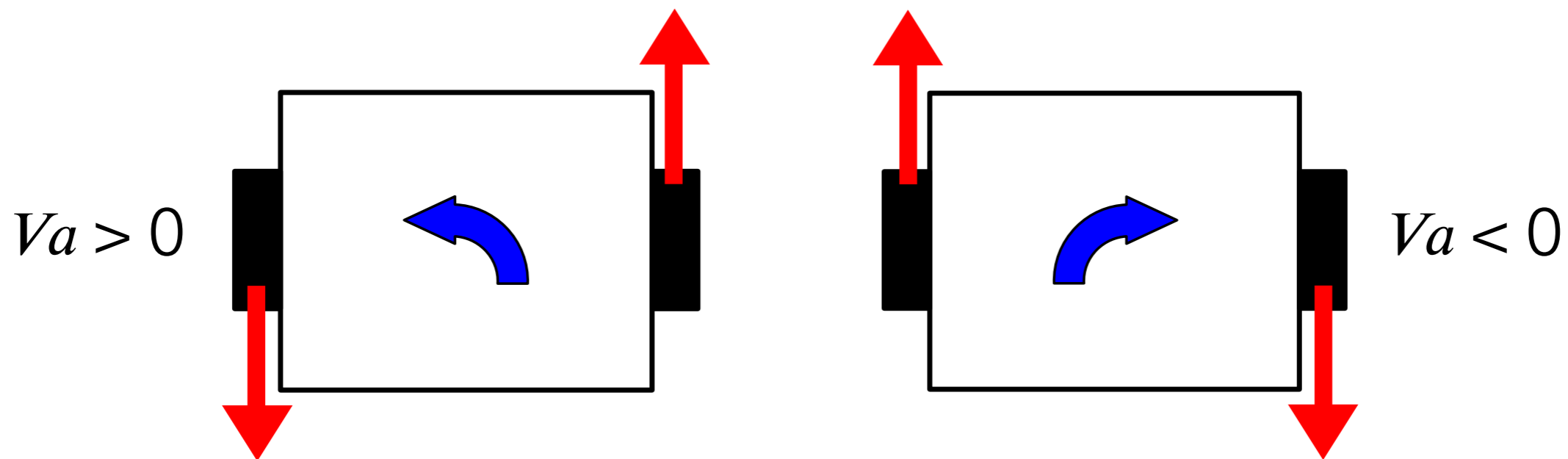


EV3の制御 | $v_x = 0$ のとき

❖ V_a : 曲がろうとする速度

❖ 値がプラスの場合, 右車輪が前に回転,
左車輪が後ろに回転 (EV3はその場で左旋回)

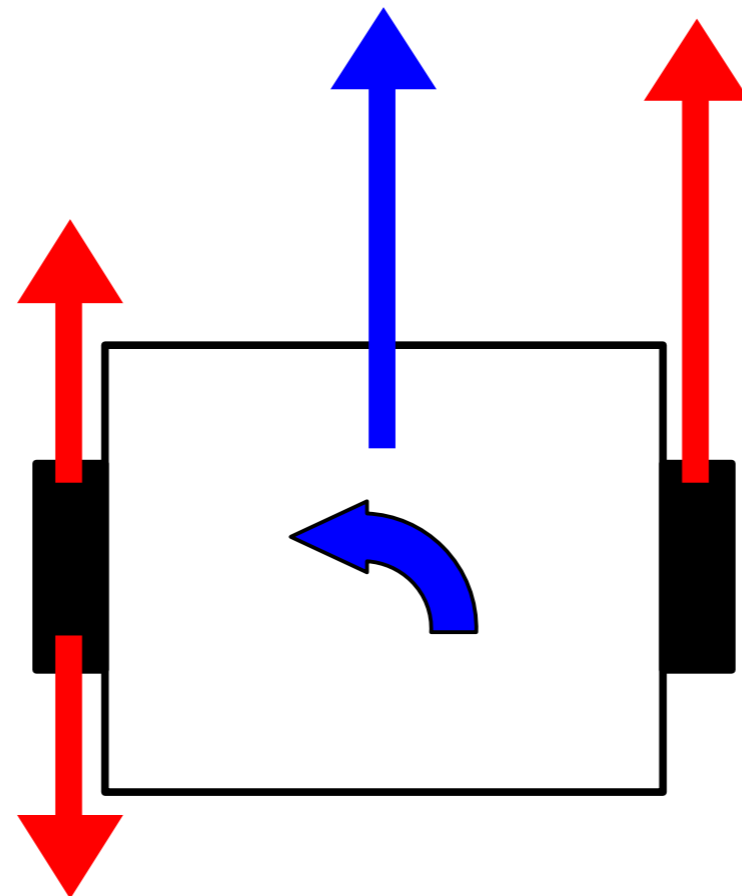
❖ 値がマイナスの場合, 左車輪が前に回転,
右車輪が後ろに回転 (EV3はその場で右旋回)



EV3の制御

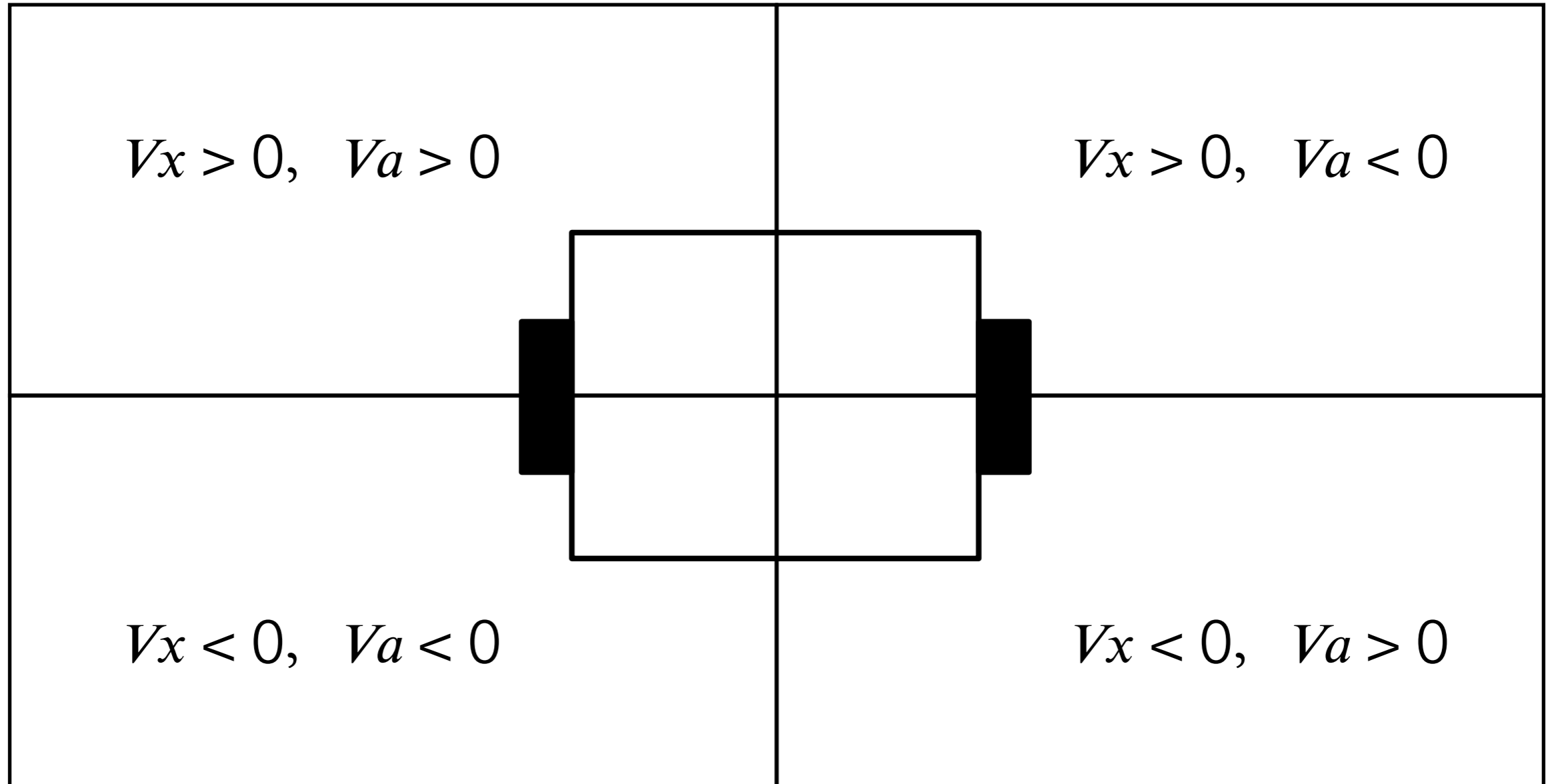
❖ $V_x > 0$, $V_a > 0$ の場合：

❖ 2つの速度から，各車輪の回転速度が求まり，
前進しながら左に旋回する（右車輪の方が回転速度が速い）



$$V_x > 0, V_a > 0$$

EV3の制御 | まとめ



各自で色々な値を入力して、
どのような挙動になるかを確認すること

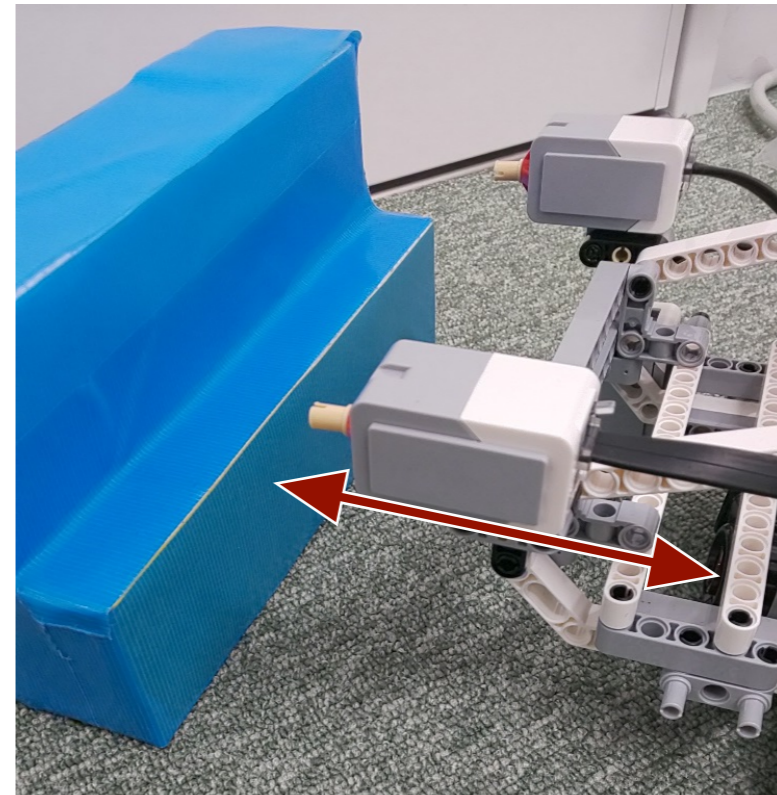
課題について

課題1

- ❖ 超音波センサを使用し、障害物を検知したら旋回するコンポーネントを作成
 - ❖ ※この課題は、実習形式で皆さんと一緒に作成します



超音波センサ

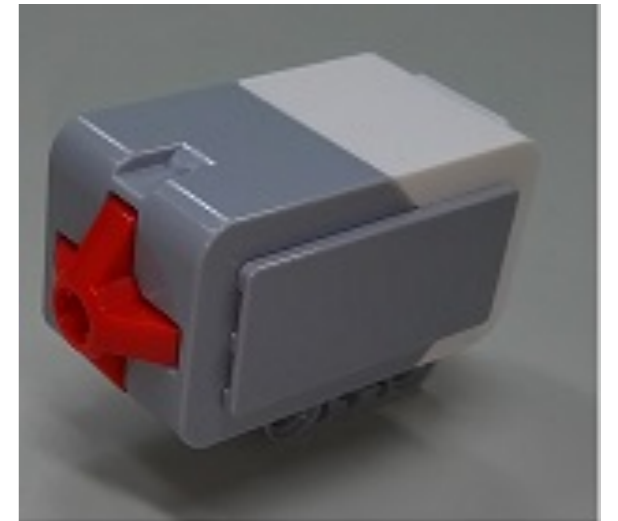


超音波センサの値が一定以下で障害物ありと判断

課題2

❖ 課題2-1

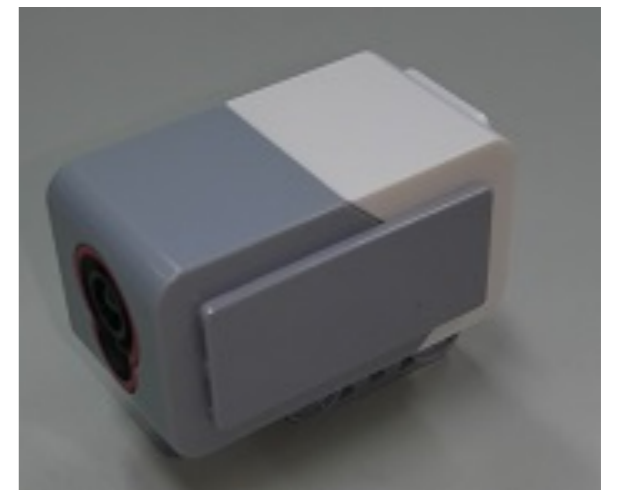
- ❖ タッチセンサを使用してセンサに反応があれば旋回するプログラムを作成



タッチセンサ

❖ 課題2-2

- ❖ カラーセンサを使用してセンサに反応があれば旋回するプログラムを作成

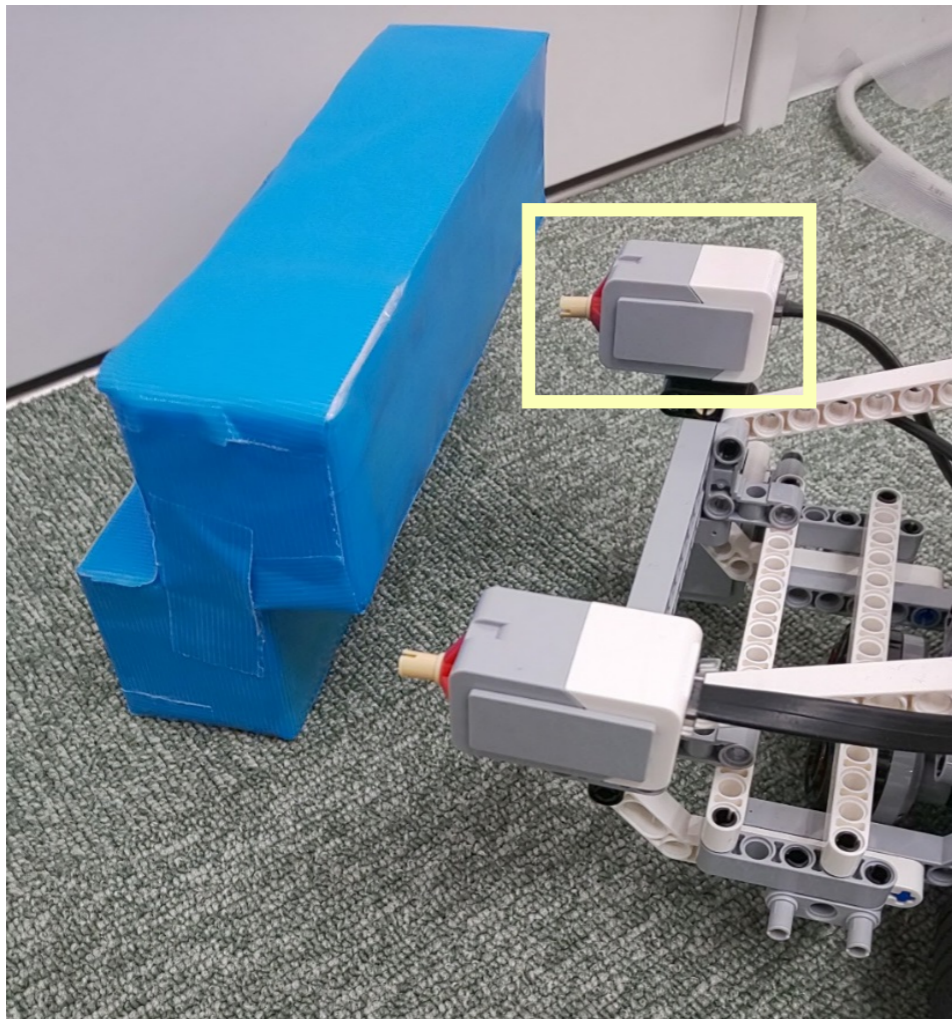


カラーセンサ

- ❖ この課題は手順書を見ながら自分で作成

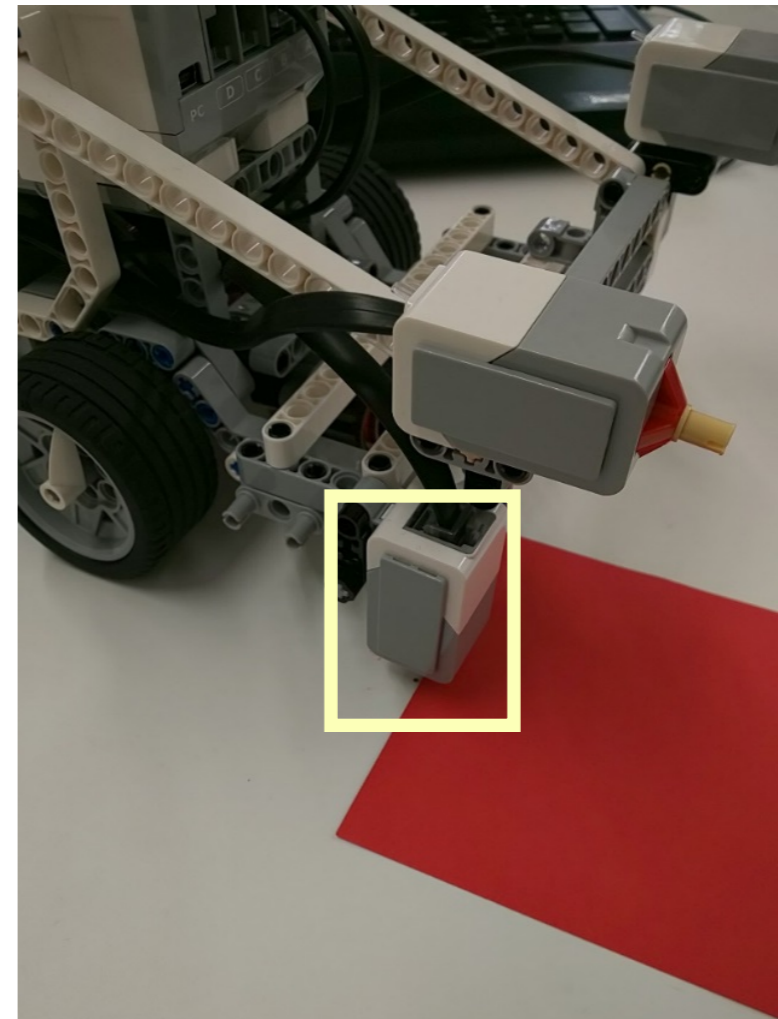
課題2

❖ タッチセンサ



タッチセンサが押されると
障害物があると判定

❖ カラーセンサ



カラーセンサが
通常とは異なる色（ここでは赤）を
検知すると障害物があると判定