プログラミング言語
Python による
計測・制御入門

伊藤　陽介

暫定版

２０２０年１２月

鳴門教育大学

ま　え　が　き

みなさんはコンピュータというとどのような形を想像するでしょうか？　ほとんどの人はパソコンやスマートフォンというでしょう。しかし，コンピュータは大きくわけて２種類の使い方がなされています。パソコンのように文書を作成したりインターネットを閲覧したりするような様々な情報を処理するコンピュータと，機械の中に部品として組み込まれて温度や音を読み取り，モータやヒータなどを制御するコンピュータがあります。

身の回りにある機械をあげてみましょう。テレビ，スマートフォン，自動車，エアコン，電子炊飯器，洗濯機，電子レンジなどたくさんの種類があります。例えば，電子炊飯器は「はじめちょろちょろ中パッパ，赤子泣いてもふた取るな。」といわれるような微妙な温度加減を行って，おいしいご飯を炊いています。単純な動作だけではなく，まわりの状態に応じて適切な働き方をさせるためには，コンピュータが必要です。また，ロボットにコンピュータを内蔵すると，自分で考えて行動できるようになります。

複雑な手順をコンピュータにさせるためには，それを表すための言葉を使います。その言葉のことをプログラムといいます。パソコンの文書作成用ソフトウェアやゲームなどもすべてプログラムで作られています。プログラムはプログラミング言語と呼ばれる言語で記述します。

プログラミング言語を学ぶときに，実際にそれが自分の考えているとおりに動作するかどうか確かめる必要があります。パソコンには，本体にディスプレイやキーボード，マウス，スピーカが付いていますが，自分自身を移動させることはできません。プログラムの動作結果のほとんどはディスプレイに表示された絵や文字として反応があるだけです。しかし，コンピュータを組み込んだロボットは，モータなどを使って移動することができます。例えば，「迷路を通り抜ける。」というプログラムを実行すると，実際に迷路を探索しながら抜け出すような動作をロボットが行うと面白いと思いませんか？　自分だけのロボットを作り，そして自分の思い通りにロボットを動かすための魂ともいえる計測・制御用プログラミング言語を学びましょう。

学校で学ぶプログラミング言語には，アイコンやブロックを組み合わせてプログラムを作るビジュアル型のものがよく利用されています。簡単な処理であればビジュアル型プログラミング言語でも記述はできますが，少し複雑な処理になると手に負えなくなります。そのため，プログラミングを生業とするプログラマは，文字を組み合わせてプログラムを作ることが一般的です。このテキストでは，様々な工夫ができるように文字を組み合わせてプログラムを作成するPythonと呼ばれるプログラミング言語を土台として，ロボットを動作させるための拡張機能を読み込んで利用する方法を取り上げています。

それでは，楽しいプログラミングの世界に入りましょう。

２０２０年１２月

国立大学法人　鳴門教育大学大学院
教　　授　　伊　藤　　陽　介
博士（工学）　（いとう　ようすけ）

本テキストの使い方

本テキストは，主に中学生がロボットを動かすためのプログラミング言語を学ぶことができるように作られています。

第１章では，ロボットの頭脳となるコンピュータを内蔵したEV3ブロックの使い方とPythonを使ってプログラミングためのソフトウェアの設定方法について説明しています。

第２章から第４章までは，EV3ブロックを単体で使って基本的なプログラムの作り方について説明でしています。

第５章から第９章までは，車輪移動型ロボットを使っていろいろなプログラムを簡単なものから順番に説明しています。各章を順番に学習することで基本的なロボットの動きをPythonのプログラムとして作ることができるようになります。プログラム例を実際に入力して，ロボットを動かしてみるとプログラムに書かれた命令の意味がよりわかるようになります。

第１０章は，ラインをたどる「ライントレース・ロボット」について説明しています。簡単そうに見えて奥の深い「ライントレース」に取り組んでください。この章の学習が終わったら，ロボットの形を工夫してより速くライントレースできるロボットを作成してみましょう。

チャレンジでは，問題を解決するプログラムを自分で作ることで，今まで学んだことがわかっているかどうか確認できるようにしています。

このテキストに掲載されているプログラムは，次の環境で動作を確認しています。

パソコンのＯＳ	: Windows 10 Pro (64bit版)
プログラム開発環境	: Microsoft Visual Studio Code 1.44.2
ev3devのバージョン	: ev3dev-stretch-ev3-generic-2020-04-10
Pythonのバージョン	: Python 3.5.3
ロボットのキット	: 教育版レゴ マインドストーム EV3 基本セット 45544
EV3ブロックのファームウェアのバージョン	: 1.10

プログラミング言語 Python による
計測・制御入門

目　　次

第１章　ロボット教材

１．１　コンピュータを内蔵したブロック

LEGO 製 Mindstorms EV3（レゴ製マインドストーム・イーブイスリー）は，ロボットを作ることのできる様々なブロックとロボットの頭脳となるコンピュータを内蔵した EV3 ブロックを含んでいます。EV3 ブロック各部の機能を図１－１，図１－２に示します。

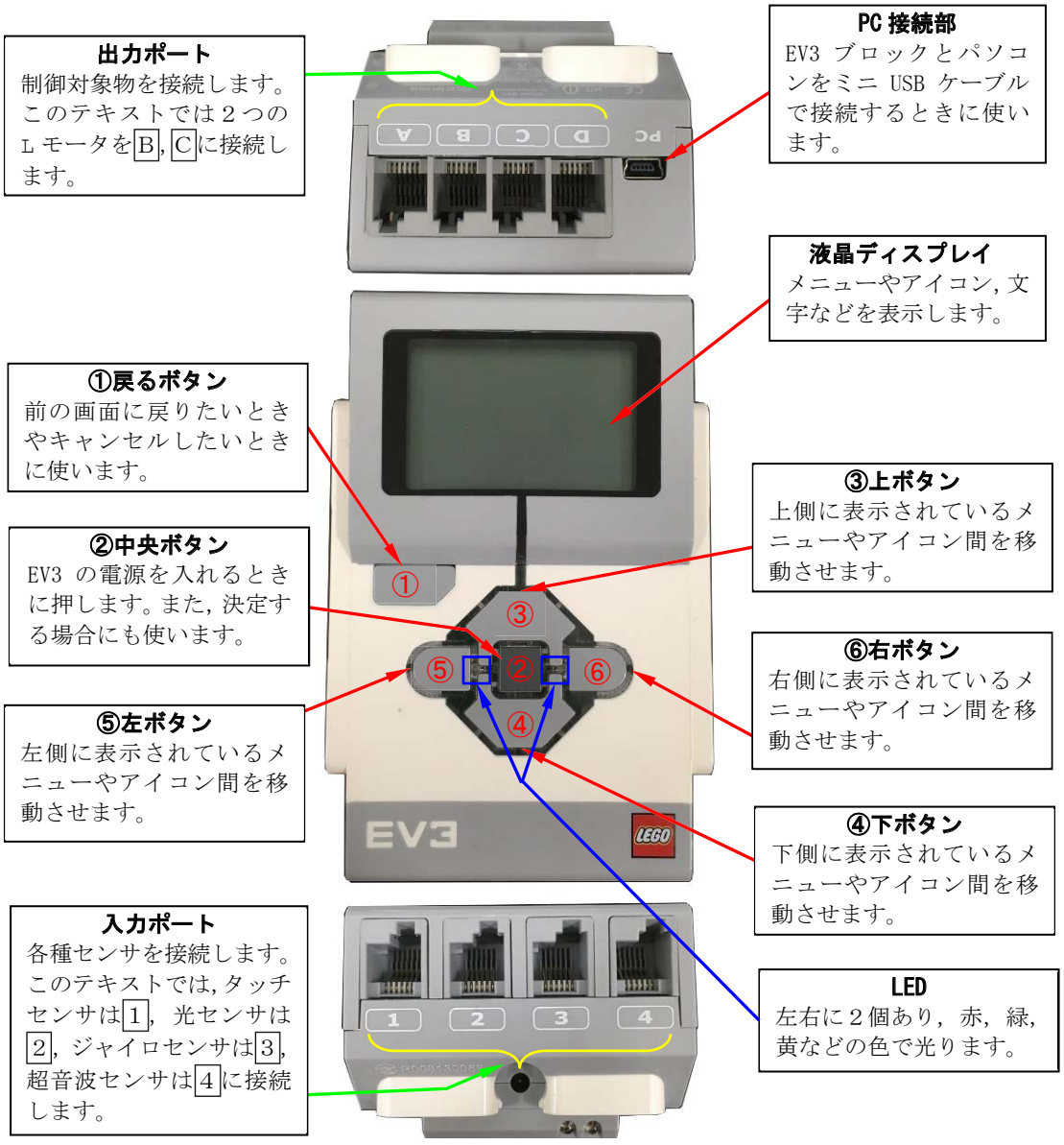


図１－１　EV3 ブロックの使い方

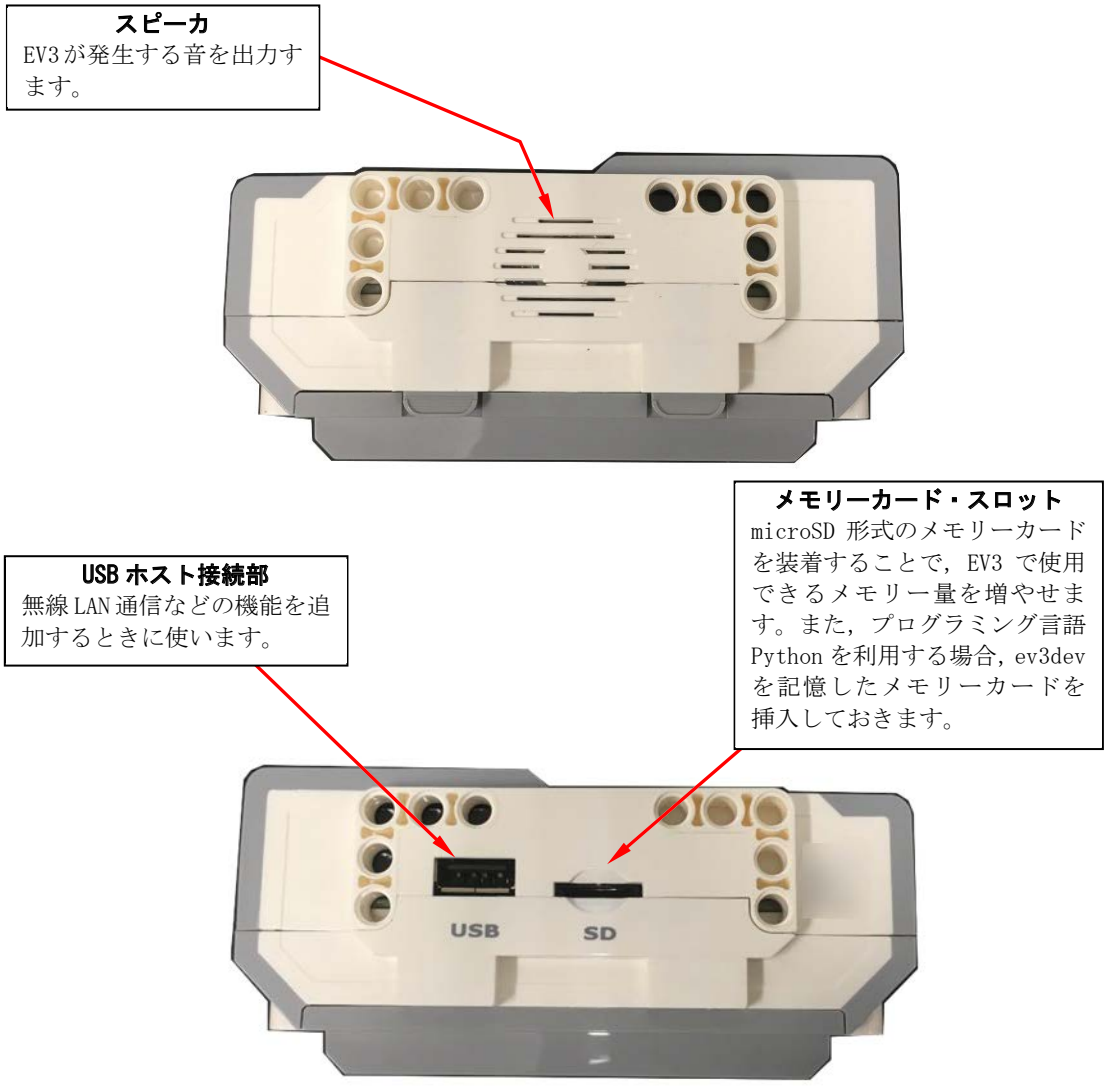


図１－２　EV3 ブロックの使い方

１．２　使用するロボット

このテキストでは，まず，図１－１に示す EV3 ブロックを単体で使用して，基本的な情報処理の流れとプログラミングについて学習します。

また，EV3 ブロックの入力ポートと出力ポートには，タッチセンサや超音波センサ，カラーセンサ，モータなどの部品を接続して利用します。

さらに，モータの制御には，図１－３に示す車輪移動型ロボットを実際に動作させて，プログラムを使って計測や制御を行う情報処理について学習します。車輪移動型ロボットは，Mindstorms EV3 に同梱されている組み立て説明図を見て組み立てます。EV3 ブロックと他のブロックを組み合わせて様々な形のロボット作ることができますが，まず車輪移動型ロボットを使ってプログラミングします。

図１－３　車輪移動型ロボット

１．３　プログラミング言語と制作環境

ロボットを思い通りに動かすためには，EV3 ブロックに内蔵されたコンピュータに動きを手順として指示しなければなりません。この手順のことをプログラムといいます。ここでは，ロボットが外部の状態を知ったり，モータを動かしたりできる計測・制御用に機能を拡張したプログラミング言語 Pytnon を使います。

Python によるプログラミングは，図１－４のような制作環境で行います。
プログラミングするパソコンには，プログラミング支援ソフトウェアをインストールしておきます。本テキストでは，Microsoft 社が提供している Visual Studio Code (VS Code)を利用します。VS Code のインストール方法と初期設定の仕方は，付録 A で説明しています。
EV3 ブロックは，ev3dev と呼ばれるソフトウェアを書き込んだメモリーカードを装着します。このメモリーカードの作り方は，付録 B で説明しています。

パソコンと EV3 ブロックはつぎの３通りの方法で接続できます。
①USB ケーブルによる有線接続は，パソコンと EV3 ブロックを USB ケーブルで接続します。
②Bluetooth による無線通信は，Bluetooth 機能を内蔵したパソコンを使って接続します。
③無線 LAN による無線通信は，無線 LAN 機能を内蔵したパソコンと USB ホスト接続部に無線 LAN 用 USB ドングルを装着した EV3 ブロックを使って接続します。
それぞれの接続方法は，付録 C で説明しています。

EV3 ブロックを個別に名前（ホスト名と呼びます。）を付けて識別する方法は，付録 D で説明しています。

まず，VS Code を起動し，プログラムを記述したファイルを保存するフォルダを作成します。そのプログラムを EV3 ブロックにダウンロードし，実行します。一旦 EV3 にダウンロードしたプログラムは，ボタン操作でも実行できます。

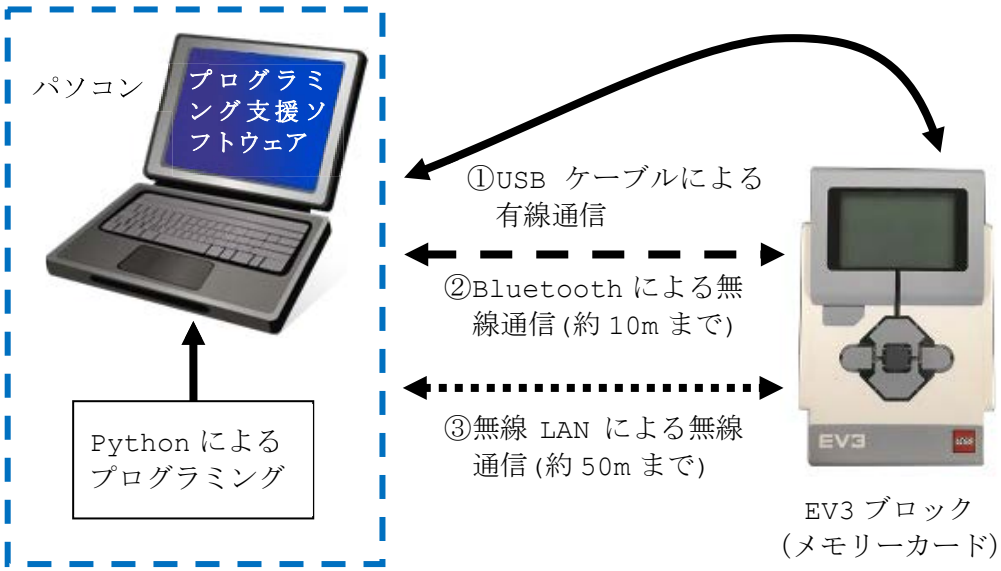


図１－４　Python によるプログラミング環境

第２章　初めてのプログラム

このテキストでは，Python と呼ばれるプログラミング言語を使います。この章では，EV3 ブロックのみを使って動作するプログラミングについて説明します。

２．１　プログラムの入力

それでは，さっそく EV3 ブロックに英語であいさつしてもらうプログラム（prog2-1.py）を入力し，実行してみましょう。

初めてプログラムを入力するときに注意する点は，次のとおりです。

※　プログラムを入力するときに注意しなければならないこと　※

①半角の英字，数字，記号を使います。英字の大文字と小文字は区別されます。

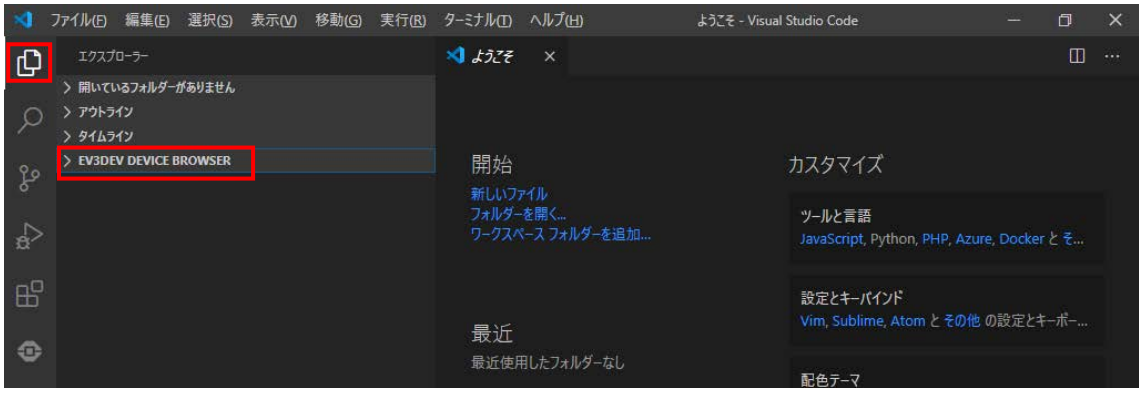
②全角の日本語や記号は特別な場合しか使うことができません。

③空白も半角を使います。全角の空白と区別できませんので特に注意しましょう。

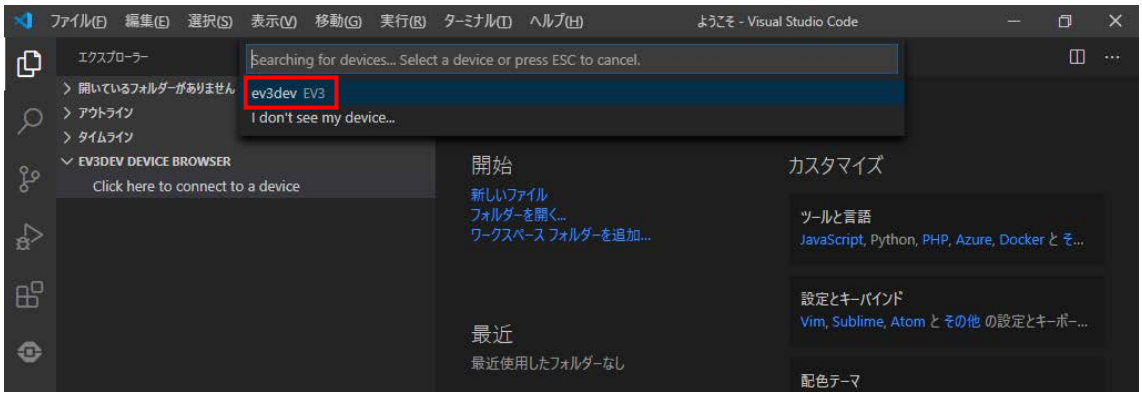
パソコンと EV3 ブロックを通信接続してから，VS Code を起動します。

VS Code の表示画面の左上にあるエクスプローラーアイコン  を左クリックします。

「EV3DEV DEVICE BROWSER」を左クリックし，「Click here to connect to a device」を表示させます。



「ev3dev EV3」を左クリックします。ここで，「ev3dev」は，EV3 に設定した名称（ホスト名）であり，「EV3」は接続方法が表示されていますので，異なる場合があります。

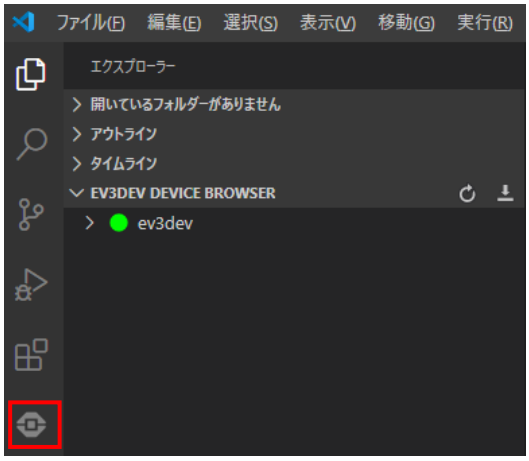


パソコンと EV3 ブロックが正常に接続されると「ev3dev」の左側のマークが●（緑色の丸）になります。

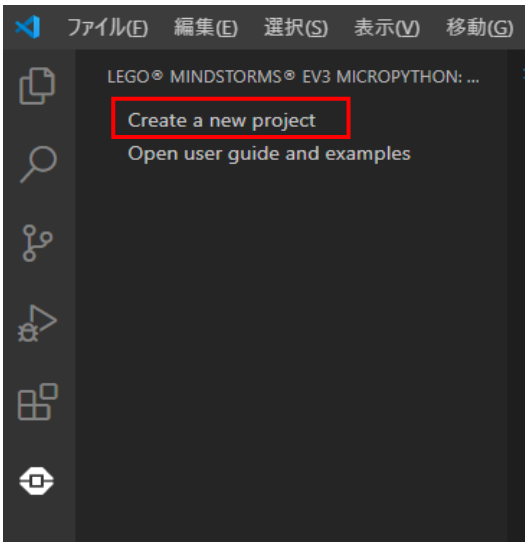
「ev3dev」の左側の「>」を左クリックすると，EV3 の電池の電圧やファイルが表示されます。

つぎに，Python のプログラムの作成方法について説明します。

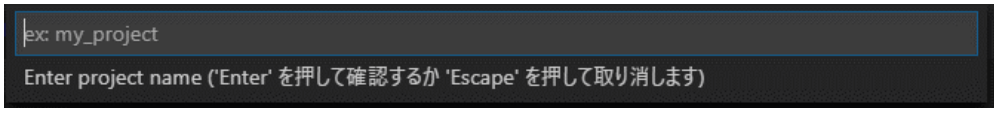
VS Code の表示画面の左側（アクティビティバーと呼びます。）に表示されている EV3 用のアイコン  を左クリックします。



「Create a new project」を左クリックします。



「ex: my_project」と表示されている枠の中に，新しく作成するフォルダ名となる文字列として，例えば「ev3project」を入力します。フォルダ名は，「半角の英数字と下線のみ」とし，日本語は利用できません。

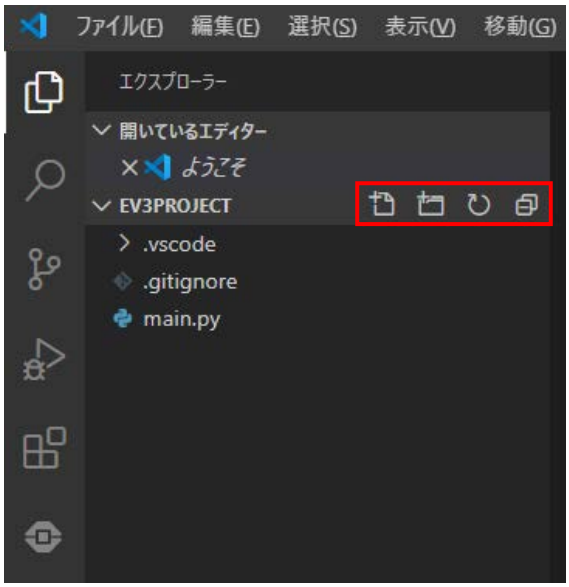


「エンターキー」を押すと，新しく作成するフォルダの場所を設定するダイアログが表示されるので，適切なフォルダの場所を選択し，「Select Folder」ボタンを左クリックし

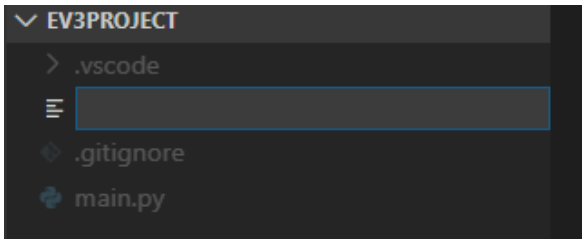
ます。

「EV3PROJECT」というプロジェクトが作成され，その中に，フォルダ「.vscode」，ファイル「.gitignore」「.gitignore」「main.py」が自動的に作成されますが，変更しないようにしてください。また，「.py」は，このファイルの内容が Python のプログラムであることを示す拡張子なので，消さないように注意しましょう。

「EV3PROJECT」と表示されているところに，マウスカーソルを重ねると，下図の赤枠のアイコンが表示されます。一番左側にある  を左クリックします。



枠内に，新たに作成するプログラムのファイル名「prog2-1.py」を入力します。ファイル名は「半角の英数字と下線のみ」とし，日本語は利用できません。

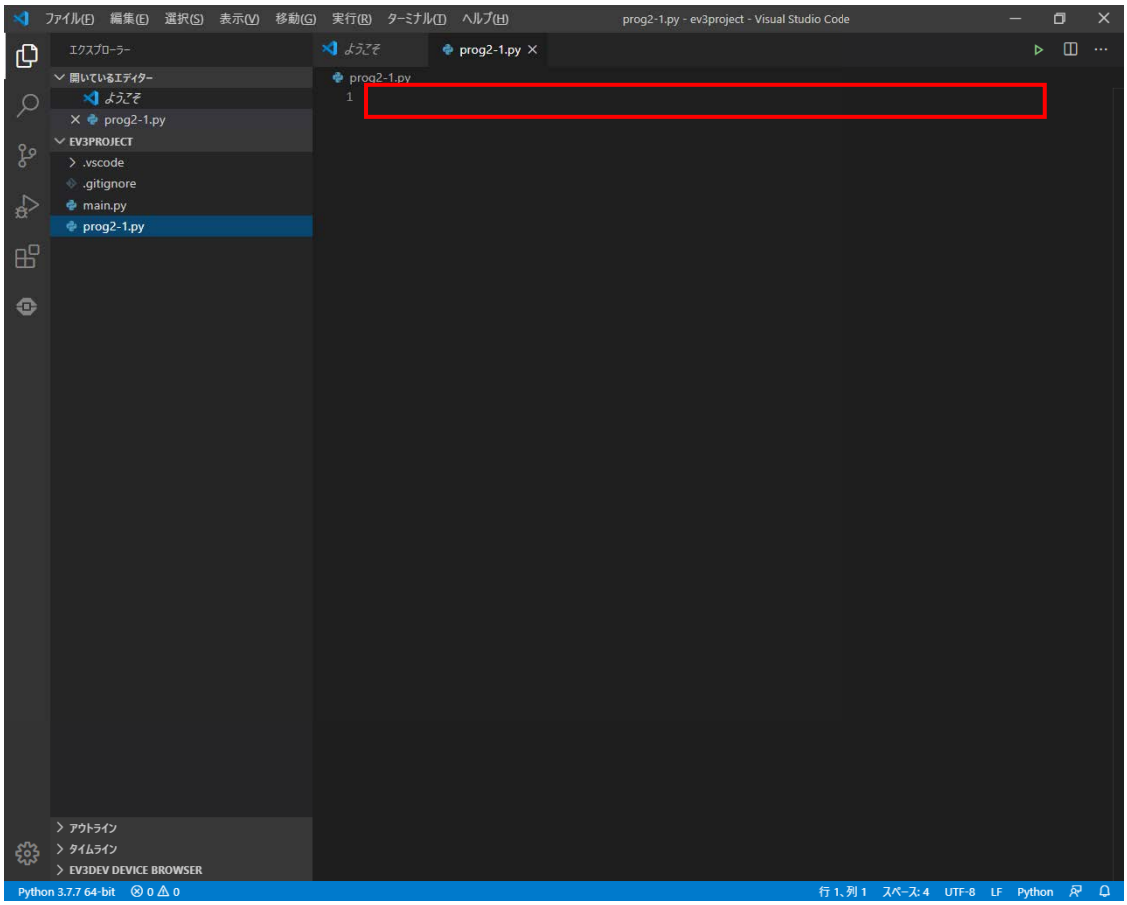


ファイル名を入力した後,「エンターキー」を押すと,「main.py」の下に「prog2-1.py」が追加され,「prog2-1.py」を編集するためのウィンドウ枠が表示されます。

「｜」で表示されているところから，以下のプログラムを入力します。ここで，␣は半角空白を示しています。

入力している途中で，候補となる語句が表示されるので，適切な語句を上下矢印キーで選択しエンターキーを押すと素早く確実に入力できます。

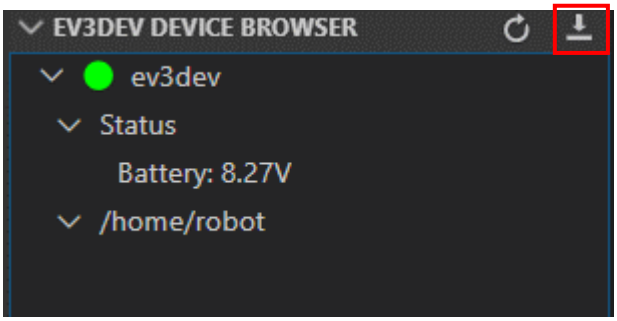
```
#!/usr/bin/env␣python3
from␣ev3dev2.sound␣import␣Sound
spk=Sound()
spk.speak('Hello.␣Nice␣to␣meet␣you.')
```



入力すると，キーワードや文字列に自動的に色分けされます。

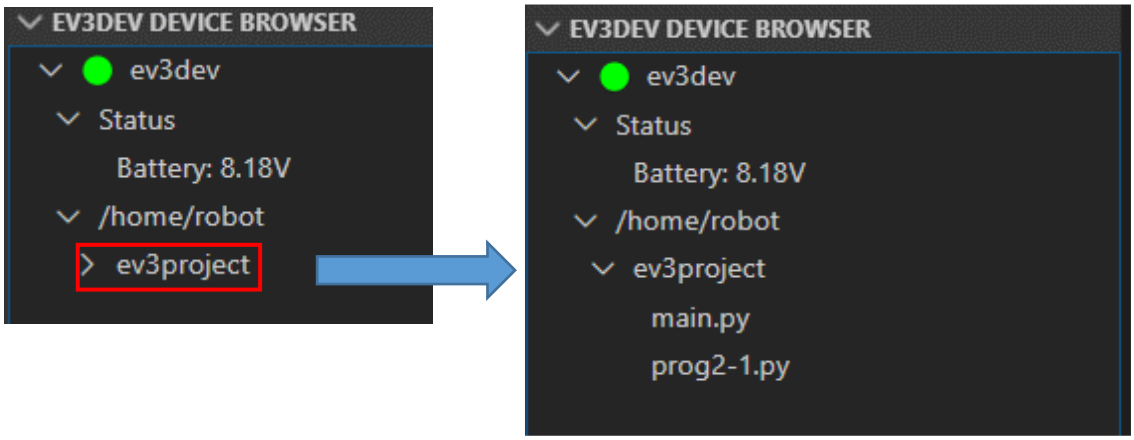
```
prog2-1.py > ...
1   #!/usr/bin/env python3
2   from ev3dev2.sound import Sound
3   spk=Sound()
4   spk.speak('Hello. Nice to meet you.')
```

「EV3DEV DEVICE BROWSER」と表示されているところにマウスカーソルを重ねると，下図の赤枠のアイコンが表示されます。一番右側にある  を左クリックします。



「ev3project」フォルダがパソコン側に保存されて，EV3 側にダウンロードされます。

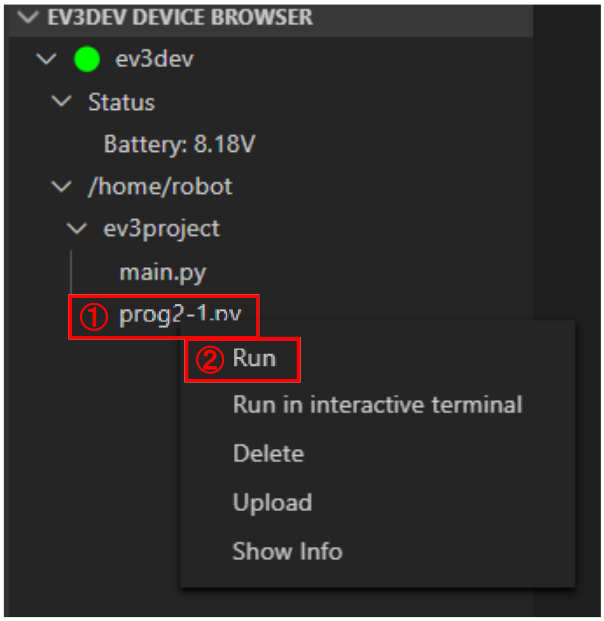
「ev3project」を左クリックすると，ファイル名の一覧が表示されます。



プログラムの実行は，つぎの手順で行います。

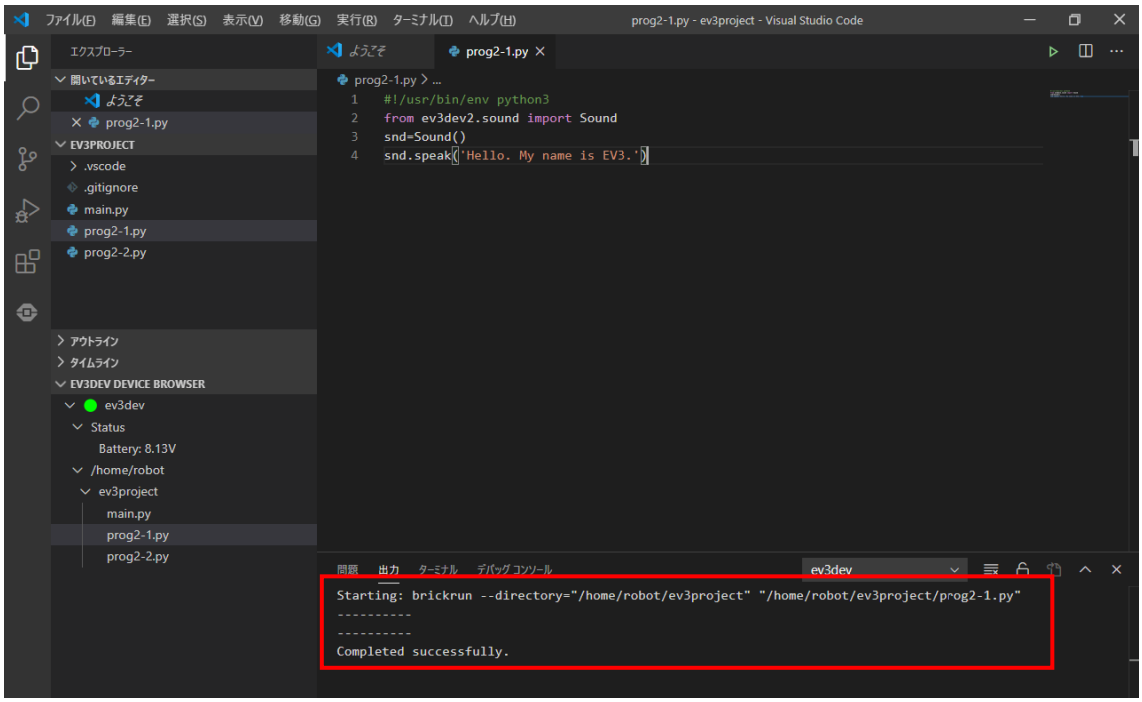
① 「prog2-1.py」を右クリックします。

② 表示されたメニューの中から「run」を左クリックします。



プログラムの実行が EV3 ブロック側で開始されるまで，約 10 秒程度必要です。「prog2-1.py」を実行すると，EV3 ブロックからあいさつが英語で発音されます。

プログラムの実行が正常に終わると，VS Code の表示画面の右下側の枠の最後に「Completed successfully.」と表示されます。



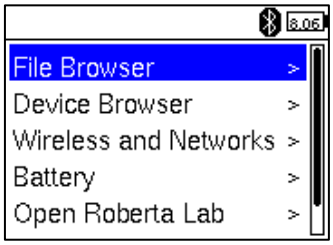
また，プログラムを実行しているときは，VS Code の右上に，

が表示されます。プログラムの実行を中断したときは，  を左クリックします。

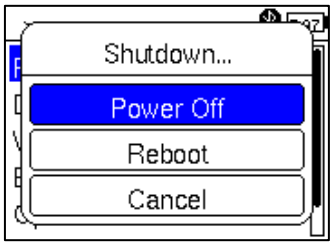
また，EV3 ブロック本体の「戻るボタン」を押しても，プログラムを終了させることができます。

一旦，EV3 ブロックにプログラムのファイルをダウンロードしておくと，EV3 の液晶ディスプレイに表示されているメニューから実行することもできます。

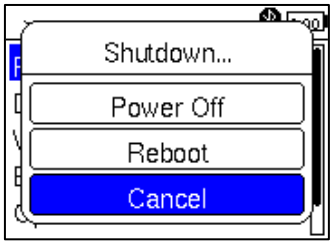
まず，下に示すメニューを表示させます。



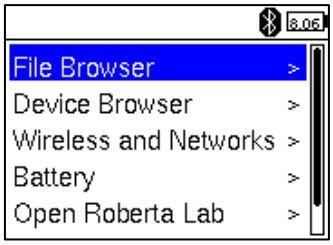
もし，このメニューが表示されていないときは，「戻るボタン」を何度か押して，下に示すメニューを表示させます。



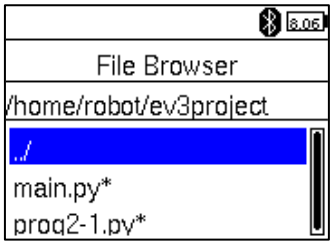
「Cancel」を「上ボタン」と「下ボタン」を使って選択し，「中央ボタン」を押します。



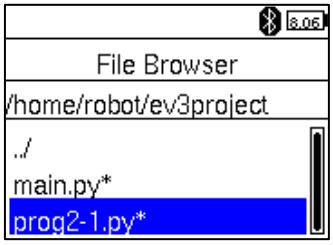
「File Browser」を「上ボタン」と「下ボタン」を使って選択し，「中央ボタン」を押します。



「上ボタン」と「下ボタン」を使って，実行するファイル名「prog2-1.py」を選択します。ファイル名の右に表示されている「*」は実行可能であることを示しています。



「中央ボタン」を押すと，「prog2-1.py」の実行が始まります。液晶ディスプレイがクリアされ，約 10 秒後に，EV3 ブロックからあいさつが英語で発音されます。



prog2-1.py を説明するために，１行ごとに行番号をつけたリストを示します。

【 prog2-1.py 】

```
1 #!/usr/bin/env python3
2 from ev3dev2.sound import Sound
3 spk=Sound()
4 spk.speak('Hello. Nice to meet you.')
```

prog2-1.py は，EV3 ブロックに英語であいさつするプログラムです。

このプログラム例には，プログラムを作成する上で必ず使う命令がいくつか入っています。それでは，プログラムの意味を説明していきます。

1 行目　#!/usr/bin/env python3

このファイルを Python のプログラムとして実行するために必要で，すべてのプログラムに必ず書いておかなければなりません。

2 行目　from ev3dev2.sound import Sound

音を処理するためのプログラムを準備します。

ev3dev2.sound は，EV3 ブロックに対して，音を処理するための様々なプログラムを集めたものです。これを Python ではモジュールと呼んでいます。

このモジュールの中から，Sound という設計図（クラスと呼びます。）を読み込みます。

「sound」と「Sound」はよく似ていますが，異なっていますので注意しましょう。

3 行目　spk=Sound()

音を処理するためクラスである Sound の実体となる spk を作ります。この実体をインスタンスと呼びます。Python によるプログラムでは，設計図（クラス）に基づいて具体化した実体（インスタンス）を作る必要があります。

4 行目　spk.speak('Hello. Nice to meet you.')

Sound クラスの中にある speak という処理（メソッドと呼びます）を実行して，

Hello. Nice to meet you.

を英語で発音します。

発音させる文字列は，'と'で囲みます。

【チャレンジ２－１】

prog2-1.py の 4 行目を修正して，次の英文を発音させてみましょう。

Rome is not made in a day. There is no royal road to learning.

【チャレンジ２－２】

prog2-1.py の 4 行目を修正して，発音させたい英文に変えてみましょう。

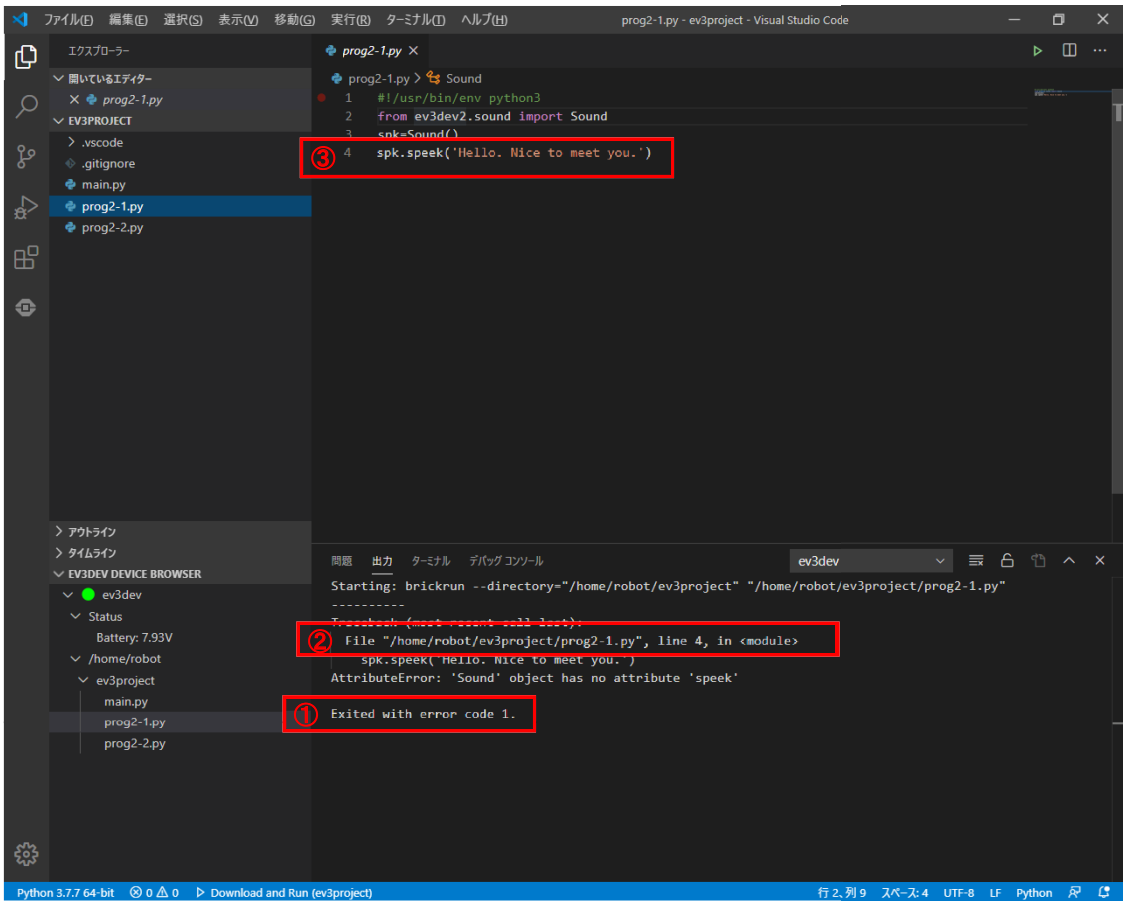
２．２　プログラムがうまく実行できないとき

例えば，「prog2-1.py」を実行したときに，うまくあいさつが英語で発音されなかったとします。

①の枠に「Exited with error code 1」というエラーを示す文字が表示されています。

エラーの場所は，②に示されている「line 4」のように，「４行目」にあります。

③に示すようにプログラムの４行目の「speek」のスペルに誤りがあり，エラーになっていることがわかります。「speek」を「speak」に修正し，再度ダウンロードして実行すると，うまくあいさつが英語で発音されます。



このように，プログラムがうまく実行できないときは，どのあたりにエラーがあるかを line ○と表示されるので，その付近を修正します。

２．３　絵を描くプログラム

EV3 ブロックの液晶ディスプレイに，図２－１に示すような顔の絵を描くプログラムを実行してみましょう。

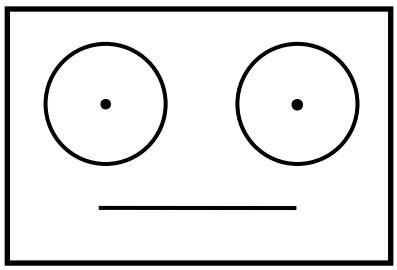


図２－１　顔のような絵

VS Code を使って新しく prog2-2.py という名前のファイルを作成し，つぎのプログラムを入力しましょう。なお，行番号は入力する必要はありません。

とくに，8 行目と 9 行目にある「fill_color」の「fill」と「color」の間にある文字は下線 _ です。半角空白とまちがわないよう入力しましょう。

【 prog2-2.py 】

```
 1 #!/usr/bin/env python3
 2 from time import sleep
 3 from ev3dev2.display import Display
 4 lcd=Display()
 5 lcd.clear()
 6 lcd.point(False,45,45)
 7 lcd.point(False,135,45)
 8 lcd.circle(False,45,45,25,fill_color=None)
 9 lcd.circle(False,135,45,25,fill_color=None)
10 lcd.line(False,45,100,135,100,width=1)
11 lcd.update()
12 sleep(10)
```

入力すると，つぎのように色がつきますので確認してください。色が異なっていると誤りがあります。

prog2-2.py > ...

```
1   #!/usr/bin/env python3
2   from time import sleep
3   from ev3dev2.display import Display
4   lcd=Display()
5   lcd.clear()
6   lcd.point(False,45,45)
7   lcd.point(False,135,45)
8   lcd.circle(False,45,45,25,fill_color=None)
9   lcd.circle(False,135,45,25,fill_color=None)
10  lcd.line(False,45,100,135,100,width=1)
11  lcd.update()
12  sleep(10)
```

正しく入力できていることが確認できたら，EV3 ブロックにプログラムをダウンロードしましょう。

prog2-2.py を実行すると，EV3 ブロックの液晶ディスプレイに，図２－２に示すような絵が 10 秒間表示されます。

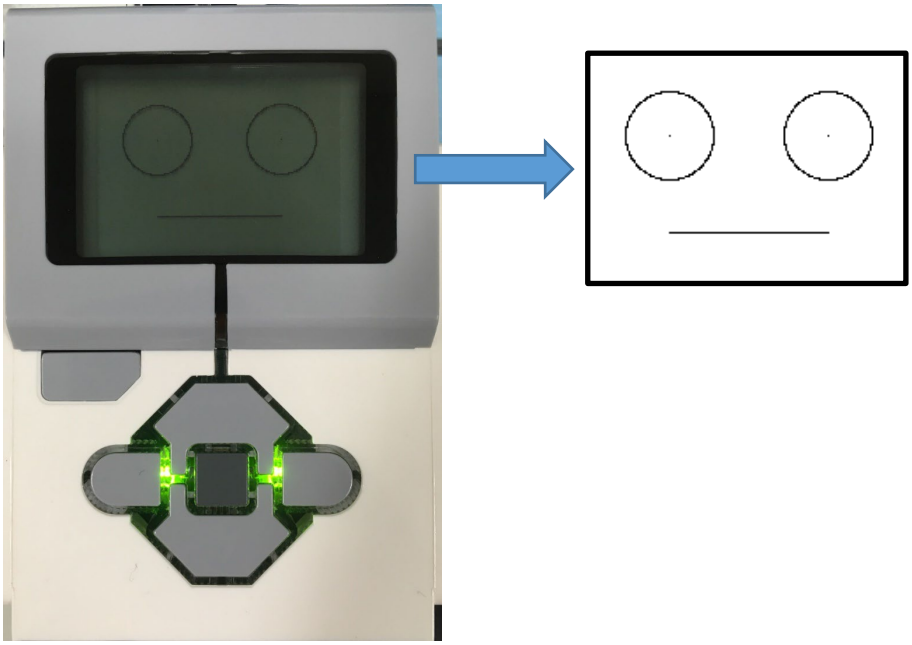


図２－２　液晶ディスプレイに表示された顔の絵

それでは，プログラムの意味を説明していきます。

1 行目　#!/usr/bin/env python3

このファイルを Python のプログラムとして実行するために必要で，すべてのプログラムに必ず書いておかなければなりません。

2 行目　from time import sleep

プログラムの実行が終了すると，EV3 ブロックの液晶ディスプレイは自動的に元の表示状態に戻ります。プログラムによって描いた内容が消えないように，時間待ちする必要がありますので，時間を処理するためのプログラムを準備します。

time は，時間に対して処理するための様々なプログラムを集めたモジュールです。このモジュールの中から，指定した時間だけ待つ機能をもつ sleep というプログラム（関数と呼びます。）を読み込みます。

3 行目　from ev3dev2.display import Display

液晶ディスプレイに対する処理を行うためのプログラムを準備します。

ev3dev2.display は，EV3 ブロックの液晶ディスプレイに対して処理するための様々なプログラムを集めたモジュールです。このモジュールの中から，Display という設計図（クラスと呼びます。）を読み込みます。

「display」と「Display」はよく似ていますが，異なった意味ですので注意しましょう。

4 行目　lcd=Display()

液晶ディスプレイに対する処理するためクラスである Display の実体となる lcd を作ります。この実体をインスタンスと呼びます。Python によるプログラムでは，設計図（クラス）に基づいて具体化した実体（インスタンス）を作る必要があります。

5 行目　lcd.clear()

Display クラスの中にある clear という処理（メソッドと呼びます）を実行して，液晶ディスプレイの表示内容を消します。

EV3 ブロックの液晶ディスプレイは，図２－３のような座標軸となっています。

X 座標の値は，0 から 177 まで，Y 座標の値は 0 から 127 までです。原点(0, 0)は，液晶ディスプレイの左上の隅になります。X 座標と Y 座標の値を組み合わせて位置を指定し，点や線，円などを描きます。

とくに，Y 座標の表し方が，数学で学習した座標と異なり，上から下が「プラス」になっていますので注意してください。

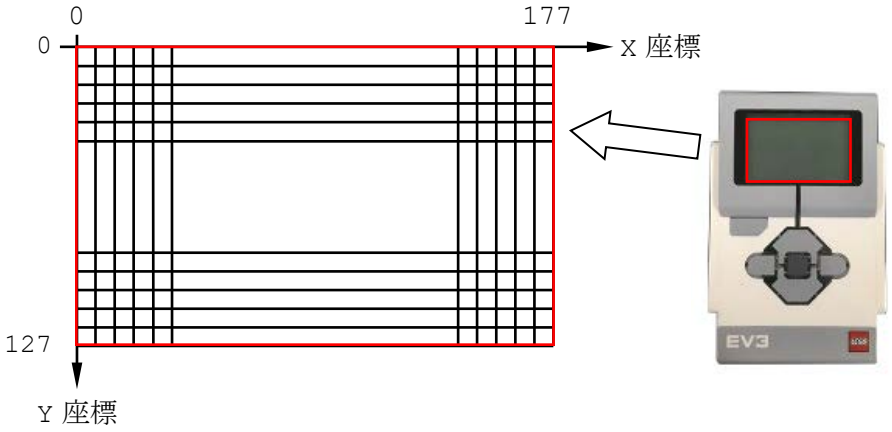


図２－３　液晶ディスプレイの座標

6 行目　`lcd.point(False,45,45)`

まず，左側の目を示す点を描きます。

図２－４に示すように，左側の目を表す点の X 座標の値を 45，Y 座標の値を 45 としましょう。

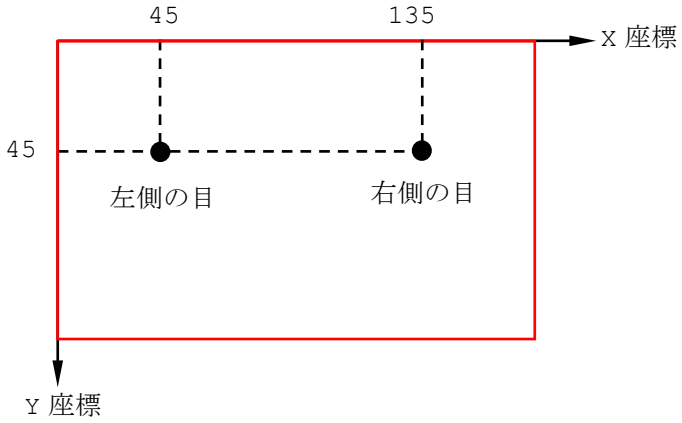


図２－４　左目と右目を表す点の座標

Display クラスの中にある point というメソッドを実行して，液晶ディスプレイに点を描きます。

point に続く丸カッコ(　)の中に書いた一番最初の「False」は，現在液晶ディスプレイに表示されている内容を消去しないという意味で，追加して表示したいときに指定します。

次の数字は，描きたい点のＸ座標の値で，今回は左側の目のＸ座標の値「45」を指定しています。

その次の数字は，描きたい点のＹ座標の値で，今回は左側の目のＹ座標の値「45」を指定しています。

7行目　lcd.point(False,135,45)

つぎに，右側の目を示す点を描きます。

図２－４に示したように，右側の目を表す点のＸ座標を 135，Ｙ座標を 45 としましょう。

6行目と同じようにpointメソッドを使って点を描きます。

pointに続く丸カッコ(　)の中に，消去しないという意味の「False」を書き，右側の目のＸ座標の値「135」を指定し，Ｙ座標の値「45」を指定しています。

左側と右側の目のＹ座標の値が「45」で同じなので，横に並んで表示されます。

8行目　lcd.circle(False,45,45,25,fill_color=None)

左側の目を示す枠を描きます。

図２－４に示すように，左側の目を表す枠を表す円の半径を 25，その中心点のＸ座標の値を 45，Ｙ座標の値を 45 としましょう。

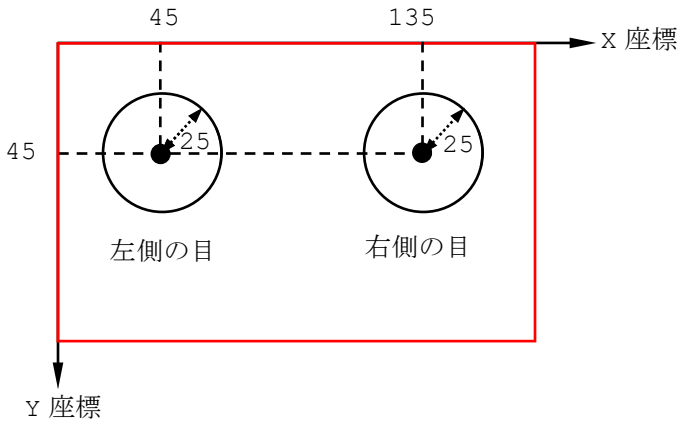


図２－５　左目と右目の枠を表す円の半径と中心点

Displayクラスの中にあるcircleというメソッドを実行して，液晶ディスプレイに円を描きます。

circleに続く丸カッコ( )の中に書いた一番最初の「False」は，現在液晶ディスプレイに表示されている内容を消去しないという意味で，追加して表示したいときに指定しま

す。

次の数字は，描きたい円の中心を表す X 座標の値で，今回は左側の目の中心である X 座標の値「45」を指定しています。

その次の数字は，描きたい円の中心を表す Y 座標の値で，今回は左側の目の中心である Y 座標の値「45」を指定しています。

さらに，その次の数字は，描きたい円の半径を表す値で，今回は，目の枠を示す値「25」を指定しています。

最後の文字は，円の内部をどのように塗りつぶすか指定し，今回は何も塗りつぶさないように「fill_color=None」を指定しています。

```
9 行目　lcd.circle(False,135,45,25,fill_color=None)
```

つぎに，右側の目を示す枠を描きます。

図２－５に示したように，左側の目を表す枠を表す円の半径を 25，その中心点の X 座標の値を 45，Y 座標の値を 45 としましょう。

8 行目と同じように circle メソッドを使って円を描きます。

circle に続く丸カッコ( )の中に，消去しないという意味の「False」を書き，右側の目の中心である X 座標の値「135」を指定し，Y 座標の値「45」を指定しています。さらに，円の半径を「25」で指定し，何も塗りつぶさないように「fill_color=None」を指定しています。

```
10 行目　lcd.line(False,45,100,135,100,width=1)
```

最後に，口を表す直線を描きます。

図２－６に示すように，口を表す直線の始点（直線の始まりを示す点）の X 座標の値を 45，Y 座標の値を 100 とし，終点（直線の終わりを示す点）の X 座標の値を 135，Y 座標の値を 100 としましょう。

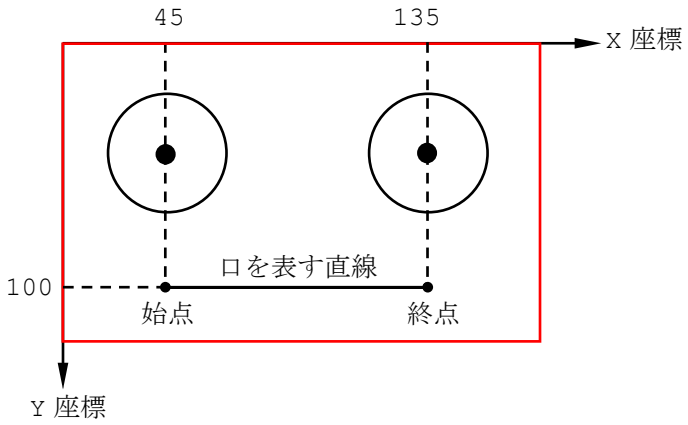


図２－６　口を表す線の位置

Display クラスの中にある line というメソッドを実行して，液晶ディスプレイに直線を描きます。

line に続く丸カッコ( )の中に書いた一番最初の「False」は，現在液晶ディスプレイに表示されている内容を消去しないという意味で，追加して表示したいときに指定します。

次の２つの数字は，描きたい直線の始点を表す X 座標と Y 座標の値で，今回は「45」と「100」を指定しています。

その次の２つの数字は,描きたい直線の終点を表す X 座標と Y 座標の値で,今回は「135」と「100」を指定しています。

最後の文字は，直線の幅を指定し，今回は幅 1 点分として「width=1」を指定しています。

11 行目　lcd.update()

点や円，直線などを描いく処理は内部で行われているだけで，液晶ディスプレイの表示に反映されません。そのため，Display クラスの中にある update メソッドを実行して，液晶ディスプレイの表示内容を更新します。ここで，顔の絵が表示されます。

12 行目　sleep(10)

プログラムの実行が終了すると，自動的に液晶ディスプレイの表示が元の状態に戻ります。これでは，表示した顔の絵がすぐに消えてしまいます。

そのため，sleep という関数と使って待つ処理を行います。

sleep に続く丸カッコ( )の中に書いた数字は，待つ時間を「秒」の単位で指定しています。このプログラムでは，10 秒間待つことを指定しています。

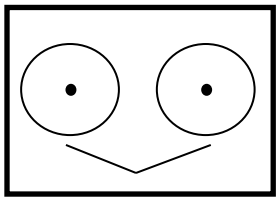
10 秒間経過すると,顔の絵から自動的に液晶ディスプレイの表示が元の状態に戻ります。

【チャレンジ２－３】

prog2-2.py の 8 行目と 9 行目の circle メソッドで指定している円の半径「25」を左目と右目で変えてみましょう。思い通りの目になりましたか？

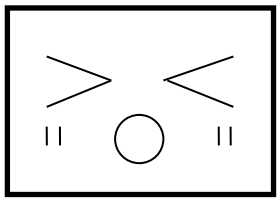
【チャレンジ２－４】

prog2-2.py を工夫して，下図のような顔を液晶ディスプレイに表示してみましょう。



【チャレンジ２－５】

prog2-2.py を工夫して，下図のような顔を液晶ディスプレイに表示してみましょう。



２．４　画像を表示してみよう

EV3 ブロックには表２－１のような画像ファイルがあらかじめ組み込まれています。この画像ファイルは，パス名とファイル名を組み合わせて指定します。「パス」は，画像ファイルの保存している場所を示しています。画像は，png という形式のファイルです。

表２－１　組み込まれている画像の一覧

上段：パス名，下段：ファイル名		画像の内容
/usr/share/images/ev3dev/mono/information/		
accept.png backward.png decline.png forward.png left.png no_go.png question_mark.p	right.png stop_1.png stop_2.png thumbs_down.png thumbs_up.png warning.png	お知らせなどに関する画像
/usr/share/images/ev3dev/mono/eyes/		
angry.png awake.png black_eye.png bottom_left.png bottom_right.png crazy_1.png crazy_2.png disappointed.png dizzy.png down.png evil.png hurt.png knocked_out.png love.png middle_left.png	middle_right.png neutral.png nuclear.png pinched_left.png pinched_middle.p pinched_right.pn sleeping.png tear.png tired_left.png tired_middle.png tired_right.png toxic.png up.png winking.png	目で表情を表した画像
/usr/share/images/ev3dev/mono/lego/		
ev3.png	ev3_icon.png	EV3 のロゴ画像

さっそく，prog2-3.py を入力して実行してみましょう。EV3 ブロックの液晶ディスプ

レイにウィンクする目の画像が 10 秒間表示されます。

【 prog2-3.py 】

```
 1 #!/usr/bin/env python3
 2 from time import sleep
 3 from ev3dev2.display import Display
 4 from PIL import Image
 5 lcd=Display()
 6 lcd.clear()
 7 img=Image.open('/usr/share/images/ev3dev/mono/eyes/winking.png')
 8 lcd.image.paste(img,(0,0))
 9 lcd.update()
10 sleep(10)
```

画像ファイルを取り扱うところについて説明します。

4 行目　from PIL import Image

画像を処理するためのプログラムを準備します。

PIL は，画像を処理するための様々なプログラムを集めたモジュールです。このモジュールの中から，Image という設計図（クラスと呼びます。）を読み込みます。

PIL というモジュールは，ev3dev 用に限らずパソコンなどで使われている python でも利用できます。

7 行目　img=Image.open('/usr/share/images/ev3dev/mono/eyes/winking.png')

画像ファイルを使うための準備（オープンと呼びます。）します。この準備には，Image クラスにある open というメソッドを使い，丸カッコ( )の中に，パス名とファイル名をつないだ文字列で，画像ファイルを指定します。

ここでは，「/usr/share/images/ev3dev/mono/eyes/」というパスにある「winking.png」という画像ファイルを指定しています。

オープンした画像は，img という名前で使います。

8 行目　lcd.image.paste(img,(0,0))

Display の中にある image というクラスにある paste という処理（メソッドと呼びます）を実行して，液晶ディスプレイに画像を配置します。

paste の丸カッコ( )の最初に，オープンした画像を指定し，その画像を液晶ディスプレイのどの位置から表示するか座標で指定します。

ここでは，7 行目でオープンした画像である img を，原点(0, 0)から配置します。

【チャレンジ２－６】

表２－１に示した画像ファイルを使って，EV3 ブロックに 5 秒ごとに３種類の表情を示す目の画像を表示してみましょう。

２．５　プログラムのファイルを消す方法

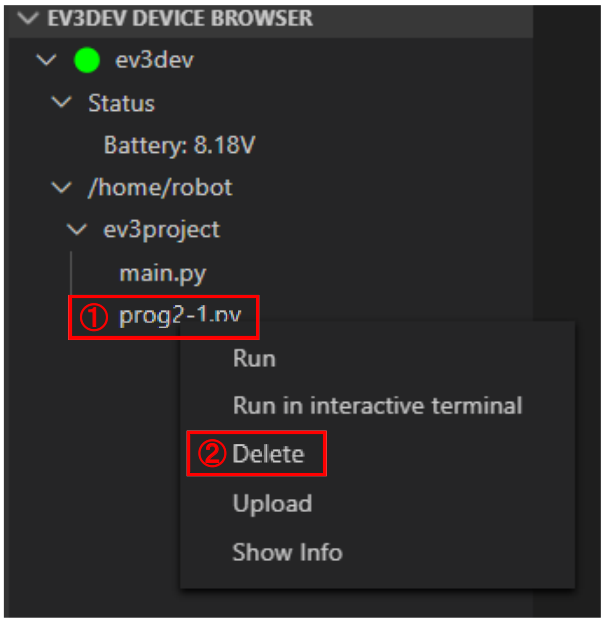
プログラムのファイルは，パソコン側と EV3 ブロック側の両方にあります。

パソコン側のファイルを消しても，EV3 グロック側のファイルは自動的に消えません。

EV3 側のファイルは，VS Code を使ってパソコンと EV3 ブロックを接続した後，つぎの手順で消します。

① 「EV3DEVICE BROWSER」の枠の中からフォルダ名を選択し，消したいファイル名を左クリックします。

② 表示されたメニューの中から「Delete」を左クリックします。



また，「EV3DEVICE BROWSER」の枠の中からフォルダ名を右クリックして表示されるメニューの中から「Delete」を左クリックすると，そのフォルダの中にあったファイルのすべてが消されます。

第３章　初めてのセンサ

まわりの状態を知るための仕組み（計測機能）をもつものを「センサ」といいます。温度や明るさ，音など様々な状態を知るセンサがありますが，この章では，何かに触れたことを知るためのセンサである「タッチセンサ」について学びます。

３．１　タッチセンサの仕組み

ここでは，EV3 ブロック用タッチセンサ（図３－１）を使います。タッチセンサは，小さな押しボタンスイッチで接触を検知します。赤色のスイッチ部分を押したり離したりすることで触れている状態を知ります。

タッチセンサは,EV3 ブロックと専用のケーブルを使って図３－２のように接続します。４つある入力ポート（図１－１）のうち，一番左端の入力ポート１番に接続できているか確認しましょう。

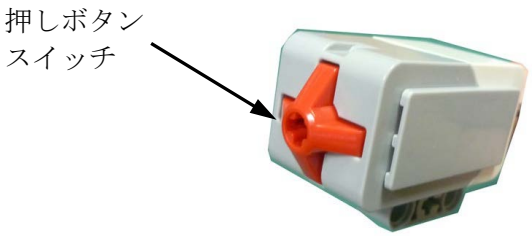


図３－１　EV3 ブロック用タッチセンサ

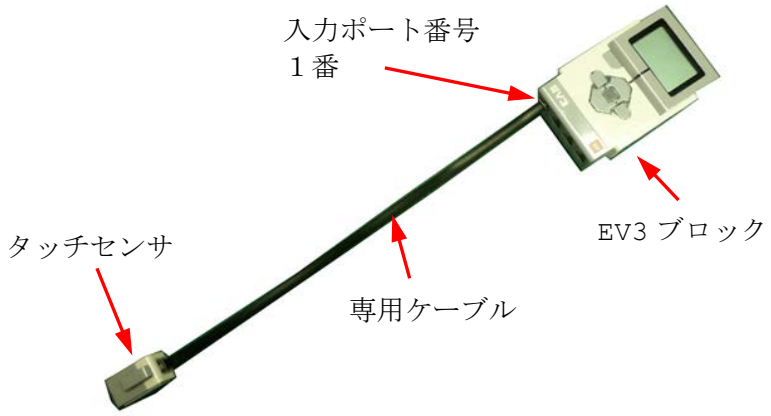


図３－２　EV3 ブロックとタッチセンサの接続

３．２　触れたことを感知するプログラム

何かに触れたことを感知するプログラム（prog3-1.py）を作りましょう。

このプログラムは，第２章で作ったプログラム「prog2-2.py」に，赤字の部分が追加になっていますので,VS Code で新しく「prog3-1.py」というファイルを作った後,「prog2-2.py」を表示します。

つぎに,「選択」メニューから「すべて選択」を行い,「編集」メニューから「コピー」を行います。この操作は，コントロールキーを押しながら「A」キーを押し,「C」キーを押しても同じです。キーによる操作の方が素早いので慣れましょう。

「prog3-1.py」を表示します。「編集」メニューから「貼り付け」を行います。この操作は，コントロールキーを押しながら「V」キーを押しても同じです。

【 prog3-1.py 】

```
 1 #!/usr/bin/env python3
 2 from time import sleep
 3 from ev3dev2.display import Display
 4 from ev3dev2.sensor.lego import TouchSensor
 5 ts=TouchSensor()
 6 lcd=Display()
 7 lcd.clear()
 8 lcd.point(False,45,45)
 9 lcd.point(False,135,45)
10 lcd.circle(False,45,45,25,fill_color=None)
11 lcd.circle(False,135,45,25,fill_color=None)
12 lcd.update()
13 ts.wait_for_pressed()
14 lcd.line(False,45,100,135,100,width=1)
15 lcd.update()
16 sleep(10)
```

プログラムを入力した後，EV3 ブロックにダウンロードし，実行しましょう。

NXT ブロックの液晶ディスプレイに２つの「目」が表示されます。タッチセンサのスイッチを押すと，「口」を表す直線が表示されて顔が完成し，10 秒後に元の画面に戻ります。

prog3-1.py は，液晶ディスプレイに表示された未完成な顔を，タッチセンサを押すことによって，完成させるプログラムです。それでは新たに加わったプログラムを細かく説明していきます。

1 行目から 3 行目までのプログラム

「２．２　絵を描くプログラム」で説明した「prog2-2.py」の 1 行目から 3 行目と同じです。

4 行目　from ev3dev2.sensor.lego import TouchSensor

タッチセンサに対する処理を行うためのプログラムを準備します。

ev3dev2.sensor.lego は，EV3 ブロックで利用できるセンサに対して処理するための様々なプログラムを集めたモジュールです。このモジュールの中から，タッチセンサに対する TouchSensor という設計図（クラスと呼びます。）を読み込みます。

5 行目　ts=TouchSensor()

タッチセンサを処理するためクラスである TouchSensor の実体（インスタンス）として ts を作ります。

6 行目から 11 行目までのプログラム

「２．２ 絵を描くプログラム」の説明で紹介したように「点」と「丸」で表現される２つの「目」を描きます。「prog2-2.py」の 4 行目から 9 行目と同じです。

12 行目　lcd.update()

Display クラスの中にある update メソッドを実行して，両目を描いたところで液晶ディスプレイの表示内容を更新します。ここで，両目の絵が表示されます。

13 行目　ts.wait_for_pressed()

TouchSensor クラスの中にある wait_for_pressed メソッドを実行します。このメソッドは，タッチセンサのスイッチが押されるまで待つ処理を行います。タッチセンサのスイッチが押されると，次の命令に実行が移ります。

14 行目から 16 行目までのプログラム

「２．２ 絵を描くプログラム」の説明で紹介したように「直線」で表現される「口」を描き，液晶ディスプレイの表示内容を更新し，10 秒間待ちます。

「prog2-2.py」の 10 行目から 12 行目と同じです。

３．３　タッチセンサの使い方

prog3-1.py で表示される顔を，まず，左目だけ表示して，タッチセンサのスイッチを押すと，右目が表示され，そのスイッチを離すと，「口」が描かれるようにしてみましょう。

この動作をするプログラムを prog3-2.py に示します。

タッチセンサのスイッチを押すまで待つメソッドは，wait_for_pressed でしたが，スイッチを離すまで待つメソッドは，17 行目に赤字で示すように wait_for_released です。

このプログラムでは，処理内容のまとまりがよくわかるように，9 行目，14 行目，18 行目にそれぞれ「空白行」を入れています。「空白行」はプログラムの実行には無関係ですが，処理のまとまりがよくわかるようになるので，よく使われます。

【 prog3-2.py 】

```
 1 #!/usr/bin/env python3
 2 from time import sleep
 3 from ev3dev2.display import Display
 4 from ev3dev2.sensor.lego import TouchSensor
 5 ts=TouchSensor()
 6 lcd=Display()
 7 lcd.clear()
 8
 9 lcd.point(False,45,45)
10 lcd.circle(False,45,45,25,fill_color=None)
11 lcd.update()
12 ts.wait_for_pressed()
13
14 lcd.point(False,135,45)
15 lcd.circle(False,135,45,25,fill_color=None)
16 lcd.update()
17 ts.wait_for_released()
18
19 lcd.line(False,45,100,135,100,width=1)
20 lcd.update()
21 sleep(10)
```

【チャレンジ３－１】

prog3-1.py において，顔が表示された後，さらに，タッチセンサの押しボタンを押すと，「ハロー」とあいさつするようにプログラムを修正してみましょう。

第４章　サウンド

EV3 ブロックはスピーカを内蔵しています。このスピーカを使って発音したり，簡単な音楽を演奏したりできます。また，スピーカを使うことでプログラムがどのような動作を行っているのかを音で知らせることができます。

４．１　ブザーを鳴らそう

EV3 ブロックにブザーのような音を鳴らせてみましょう。

prog4-1.py は，ブザーを１回鳴らすプログラムです。

【 prog4-1.py 】

```
1 #!/usr/bin/env python3
2 from ev3dev2.sound import Sound
3 spk=Sound()
4 spk.beep()
```

1 行目　#!/usr/bin/env python3

このファイルを Python のプログラムとして実行します。

2 行目　from ev3dev2.sound import Sound

音を処理するためのプログラムを準備します。

ev3dev2.sound は，EV3 ブロックを使って音を処理するための様々なプログラムを集めたモジュールです。このモジュールの中から Sound という設計図（クラスと呼びます。）を読み込みます。

3 行目　spk=Sound()

音を処理するためクラスである Sound の実体（インスタンス）となる spk を作ります。

4 行目　spk.beep()

Sound クラスの中にある beep という処理（メソッドと呼びます）を実行して，ブザーを１回鳴らします。

【チャレンジ４－１】

prog4-1.py を修正して，ブザーを３回鳴らすプログラムにしてみましょう。

４．２　音楽の演奏

音楽は，音の高さや音量，鳴らす長さで決められる単音と休みの長さを組み合わせてできています。

EV3 ブロックでは，おおよそ８オクターブの範囲で音を鳴らすことができます。音の高さ（周波数）を指定する数値と音階の対応を表４－１に示します。ここでの数値は，プログラムを入力しやすくするように整数にしていますので，やや音階がずれることがあります。その場合は，小数点以下３桁程度までのより正確な値を指定します。

聴きやすい音楽を作るときは，音の高さが 440 で指定される「ラ」を基準とします。

表４－１　音の高さ（周波数）を指定する数値

オクターブ	1	2	3	4	5	6	7	8
ド	33	65	131	262	523	1047	2093	4186
ド＃	35	69	139	277	554	1109	2217	4435
レ	37	73	147	294	587	1175	2349	4699
レ＃	39	78	156	311	622	1245	2489	4978
ミ	41	82	165	330	659	1319	2637	5274
ファ	44	87	175	349	698	1397	2794	5588
ファ＃	46	92	185	370	740	1480	2960	5920
ソ	49	98	196	392	784	1568	3136	6272
ソ＃	52	104	208	415	831	1661	3322	6645
ラ	55	110	220	440	880	1760	3520	7040
ラ＃	58	117	233	466	932	1865	3729	7459
シ	62	123	247	494	988	1976	3951	

prog4-2.py は，単音の音の大きさを設定した後，「ド，レ，ミ，ファ，ソ，ラ，シ」と，0.4 秒間ずつ鳴らし，各音の間で 0.1 秒間休むプログラムです。プログラム中の「# ド」などは，注釈（コメントとも呼びます。）です。プログラムを分かりやすくするために書いてあるものなので，入力しなくてもかまいません。

【 prog4-2.py 】

```
 1 #!/usr/bin/env python3
 2 from ev3dev2.sound import Sound
 3 spk=Sound()
 4 spk.set_volume(50, 'Beep')
 5 spk.play_tone(262,0.4,0.1)    # ド
 6 spk.play_tone(294,0.4,0.1)    # レ
 7 spk.play_tone(330,0.4,0.1)    # ミ
 8 spk.play_tone(349,0.4,0.1)    # ファ
 9 spk.play_tone(392,0.4,0.1)    # ソ
10 spk.play_tone(440,0.4,0.1)    # ラ
11 spk.play_tone(494,0.4,0.1)    # シ
```

単音を鳴らすためのメソッドとして play_tone と set_volume があります。これを組み合わせていろいろな音楽を演奏するプログラムを作ってみましょう。

それでは，新たに出てきた命令を説明します。

4 行目　spk.set_volume(50, 'Beep')

Sound クラスの中にある set_volume という処理（メソッドと呼びます）を実行して，単音の大きさを設定しています。

単音の大きさは，set_volume に続く丸カッコ( )の中に，0 から 100 の数値で設定し，その次に'Beep'という文字列を指定します。100 が一番大きな音になります。この設定は，プログラムを終了しても，元に戻されませんので注意してください。

5 行目　spk.play_tone(262,0.4,0.1)　　# ド

Sound クラスの中にある play_tone という処理（メソッドと呼びます）を実行して，単音を鳴らします。

単音は，play_tone に続く丸カッコ( )の中に，音の高さを示す周波数として 262，鳴らす時間として 0.4 秒，次に鳴らすまで休む時間を 0.1 秒と指定しています。音の高さは周波数という数値で指定します。この数値が大きいほど高い音となり，小さいほど低い音になります。鳴らす時間と休む時間の単位は「秒」で指定します。

また，「#」から行末までは，注釈（コメント）を意味しています。注釈はプログラムの実行には関係ありませんが，プログラムの内容や意味を人間にわかりやすくするためによく使われます。

同様に，6 行目から 11 行目でも play_tone メソッドを使って単音を鳴らしています。

つぎに，音の大きさを変えてみましょう。

prog4-3.py は，「ラ」を 0.4 秒間ずつ鳴らし，音と音の間で 0.1 秒間休みながら次第

に音が大きくなっていくプログラムです。

【 prog4-3.py 】

```
1 #!/usr/bin/env python3
2 from ev3dev2.sound import Sound
3 spk=Sound()
4 spk.set_volume(10, 'Beep')
5 spk.play_tone(440,0.4,0.1)
6 spk.set_volume(50, 'Beep')
7 spk.play_tone(440,0.4,0.1)
8 spk.set_volume(90, 'Beep')
9 spk.play_tone(440,0.4,0.1)
```

prog4-4 は，童謡「きらきら星」の曲の一部です。プログラムを実行して演奏させてみましょう。

11 行目は，休みで音を出さないように，音の高さ（周波数）を 0 にしています。

【 prog4-4.py 】

```
1 #!/usr/bin/env python3
2 from ev3dev2.sound import Sound
3 spk=Sound()
4 spk.set_volume(50, 'Beep')
5 spk.play_tone(262,0.4,0.1)
6 spk.play_tone(262,0.4,0.1)
7 spk.play_tone(392,0.4,0.1)
8 spk.play_tone(392,0.4,0.1)
9 spk.play_tone(440,0.4,0.1)
10 spk.play_tone(440,0.4,0.1)
11 spk.play_tone(392,0.4,0.1)
12 spk.play_tone(0,0.4,0.1)
13 spk.play_tone(349,0.4,0.1)
14 spk.play_tone(349,0.4,0.1)
15 spk.play_tone(330,0.4,0.1)
16 spk.play_tone(330,0.4,0.1)
17 spk.play_tone(294,0.4,0.1)
18 spk.play_tone(294,0.4,0.1)
19 spk.play_tone(262,0.4,0.1)
```

prog4-4.py の音の高さをよりわかりやすくすることができます。音の高さを表す周波数を「音階」で書いたプログラムを prog4-5.py に示します。

6 行目から 11 行目に「音階」を表すために，変数を利用しています。

音階に対応したわかりやすい名前の変数を使って，音階を表す周波数を代入します。代入は，変数名の次に「=」を書き，続けて数式を書きます。数式は数値のみでもかまいません。

例えば，音階の「ド」を示す場合，

do=262

とします。これ以降，do と書くと，「262」を意味することになります。同様に，使用する音階の周波数を変数に代入していきます。

12 行目では，音を出さないという意味で，「nn=0」としています。

さらに，14 行目と 15 行目で，発音する時間と休む時間もそれぞれ変数 st と sr に代入しています。

【 prog4-5.py 】

```
 1 #!/usr/bin/env python3
 2 from ev3dev2.sound import Sound
 3 spk=Sound()
 4 spk.set_volume(50, 'Beep')
 5
 6 do=262
 7 re=294
 8 mi=330
 9 fa=349
10 so=392
11 la=440
12 nn=0
13
14 st=0.4
15 sr=0.1
16
17 spk.play_tone(do,st,sr)
18 spk.play_tone(do,st,sr)
19 spk.play_tone(so,st,sr)
20 spk.play_tone(so,st,sr)
21 spk.play_tone(la,st,sr)
22 spk.play_tone(la,st,sr)
23 spk.play_tone(so,st,sr)
24 spk.play_tone(nn,st,sr)
25 spk.play_tone(fa,st,sr)
```

```
26 spk.play_tone(fa,st,sr)
27 spk.play_tone(mi,st,sr)
28 spk.play_tone(mi,st,sr)
29 spk.play_tone(re,st,sr)
30 spk.play_tone(re,st,sr)
31 spk.play_tone(do,st,sr)
```

テンポを速めたいときは，14 行目と 15 行目を

st=0.3

sr=0.05

に変更して，実行します。このように，変数を使うと play_tone で指定した値をそれぞれ修正する必要はなくなります。

また，単なる数値では，何を表しているかわからなくなりますが，変数に適切な名前を付けることで意味を示すこともできるようになります。

prog4-6.py は，「さくらさくら」の曲を演奏するプログラムです。楽譜に合わせて周波数や鳴らす時間，休みの時間が少しずつ異なることが分かります。

16 行目から 19 行目に示すように，代入文を並べて書きたいときは，セミコロン「;」で区切ります。

【 prog4-6.py 】

```
 1 #!/usr/bin/env python3
 2 from ev3dev2.sound import Sound
 3 spk=Sound()
 4 spk.set_volume(50, 'Beep')
 5
 6 sil=247
 7 do=262
 8 re=294
 9 mi=330
10 fa=349
11 so=392
12 la=440
13 si=494
14 doh=523
15
16 sth=0.2; srh=0.05
17 st1=0.4; sr1=0.1
```

```
18 st2=0.8; sr2=0.2
19 st3=1.2; sr3=0.3
20
21 spk.play_tone(la,st1,sr1)
22 spk.play_tone(la,st1,sr1)
23 spk.play_tone(si,st2,sr2)
24
25 spk.play_tone(la,st1,sr1)
26 spk.play_tone(la,st1,sr1)
27 spk.play_tone(si,st2,sr2)
28
29 spk.play_tone(la,st1,sr1)
30 spk.play_tone(si,st1,sr1)
31 spk.play_tone(doh,st1,sr1)
32 spk.play_tone(si,st1,sr1)
33 spk.play_tone(la,st1,sr1)
34 spk.play_tone(si,sth,srh)
35 spk.play_tone(la,sth,srh)
36 spk.play_tone(fa,st2,sr2)
37
38 spk.play_tone(mi,st1,sr1)
39 spk.play_tone(do,st1,sr1)
40 spk.play_tone(mi,st1,sr1)
41 spk.play_tone(fa,st1,sr1)
42 spk.play_tone(mi,st1,sr1)
43 spk.play_tone(mi,sth,srh)
44 spk.play_tone(do,sth,srh)
45 spk.play_tone(sil,st2,sr2)
46
47 spk.play_tone(la,st1,sr1)
48 spk.play_tone(si,st1,sr1)
49 spk.play_tone(doh,st1,sr1)
50 spk.play_tone(si,st1,sr1)
51 spk.play_tone(la,st1,sr1)
52 spk.play_tone(si,sth,srh)
53 spk.play_tone(la,sth,srh)
54 spk.play_tone(fa,st2,sr2)
55
56 spk.play_tone(mi,st1,sr1)
57 spk.play_tone(do,st1,sr1)
58 spk.play_tone(mi,st1,sr1)
59 spk.play_tone(fa,st1,sr1)
60 spk.play_tone(mi,st1,sr1)
```

```
61 spk.play_tone(mi,sth,srh)
62 spk.play_tone(do,sth,srh)
63 spk.play_tone(sil,st2,sr2)
64
65 spk.play_tone(la,st1,sr1)
66 spk.play_tone(la,st1,sr1)
67 spk.play_tone(si,st2,sr2)
68
69 spk.play_tone(la,st1,sr1)
70 spk.play_tone(la,st1,sr1)
71 spk.play_tone(si,st2,sr2)
72
73 spk.play_tone(mi,st1,sr1)
74 spk.play_tone(fa,st1,sr1)
75 spk.play_tone(si,sth,srh)
76 spk.play_tone(la,sth,srh)
77 spk.play_tone(fa,st1,sr1)
78 spk.play_tone(mi,st3,sr3)
```

【チャレンジ４－２】

prog4-5.py の童謡「きらきら星」の続きのメロディを入力して，曲を完成させましょう。

＜ヒント＞

ドドソソララソ　ファファミミレレド　ソソファファミミレ　ソソファファミミレ
ドドソソララソ　ファファミミレレド

４．３　組み込みサウンドを再生してみよう

EV3 ブロックには表４－１のようなサウンドがあらかじめ組み込まれています。このサウンドは，パス名とファイル名を組み合わせて指定します。「パス」は，ファイルの保存している場所を示しています。サウンドは，wav という形式のファイルです。

表４－２　組み込みサウンドの一覧

上段：パス名，下段：ファイル名		サウンドの内容
/usr/share/sounds/ev3dev/animals/		
cat_purr.wav dog_bark_1.wav dog_bark_2.wav dog_growl.wav dog_sniff.wav dog_whine.wav elephant_call.wav	insect_buzz_1.wav insect_buzz_2.wav insect_chirp.wav snake_hiss.wav snake_rattle.wav t-rex_roar.wav	動物などの鳴く音
/usr/share/sounds/ev3dev/system/		
click.wav confirm.wav general_alert.wav	overpower.wav ready.wav	システムに関係する音
/usr/share/sounds/ev3dev/information/		
activate.wav analyze.wav backwards.wav color.wav detected.wav down.wav error.wav error_alarm.wav flashing.wav forward.wav	left.wav object.wav right.wav searching.wav start.wav stop.wav touch.wav turn.wav up.wav	お知らせなどに関する音
/usr/share/sounds/ev3dev/communication/		
bravo.wav ev3.wav fantastic.wav game_over.wav	lego.wav mindstorms.wav morning.wav no.wav	あいさつなどに関する音

go.wav good.wav good_job.wav goodbye.wav hello.wav hi.wav	okay.wav okey-dokey.wav sorry.wav thank_you.wav yes.wav	
/usr/share/sounds/ev3dev/mechanical/		
air_release.wav airbrake.wav backing_alert.wav horn_1.wav horn_2.wav laser.wav	motor_idle.wav motor_start.wav motor_stop.wav ratchet.wav sonar.wav tick_tack.wav	機械が発する音
/usr/share/sounds/ev3dev/numbers/		
zero.wav one.wav two.wav three.wav four.wav five.wav	six.wav seven.wav eight.wav nine.wav ten.wav	数字の音
/usr/share/sounds/ev3dev/colors/		
black.wav blue.wav brown.wav green.wav	red.wav white.wav yellow.wav	色の名前の音
/usr/share/sounds/ev3dev/expressions/		
boing.wav boo.wav cheering.wav crunching.wav crying.wav fanfare.wav kung_fu.wav laughing_1.wav	laughing_2.wav magic_wand.wav ouch.wav shouting.wav smack.wav sneezing.wav snoring.wav uh-oh.wav	表情などを表す音

さっそく，prog4-7.py を入力して実行し，サウンドを再生してみましょう。犬の鳴き声がした後，虫の鳴き声が聞こえてきます。

【 prog4-7.py 】

```
1 #!/usr/bin/env python3
2 from ev3dev2.sound import Sound
3 spk=Sound()
4 spath='/usr/share/sounds/ev3dev/animals/'
5 spk.play_file(spath+'dog_bark_1.wav')
6 spk.play_file(spath+'insect_chirp.wav')
```

4 行目　path='/usr/share/sounds/ev3dev/animals/'

サウンドのファイルが保存されている場所を示すパス名を，spath という変数名に文字列で代入しています。

5 行目　spk.play_file(path+'dog_bark_1.wav')

Sound クラスの中にある play_file という処理（メソッドと呼びます）を実行して，サウンド・ファイル再生します。play_file に続く丸カッコ( )の中に，パス名とファイル名を文字列で指定します。

ここでは, 変数 spath に代入された文字列「/usr/share/sounds/ev3dev/animals/」というパスに保存されている犬の鳴き声のサウンド・ファイル「dog_bark_1.wav」をプラス記号「+」でくっつけて一つの文字列にしています。

「dog_bark_1.wav」が再生されると，犬の鳴き声がスピーカから聞こえてきます。

6 行目　spk.play_file(path+'insect_chirp.wav')

変数 spath に代入された文字列「/usr/share/sounds/ev3dev/animals/」というパスに保存されている虫の鳴き声のサウンド・ファイル「insect_chirp.wav」をプラス記号「+」でくっつけて一つの文字列にしています。

「insect_chirp.wav」が再生されると，虫の鳴き声がスピーカから聞こえてきます。

【チャレンジ４－３】

表４－２に示した組み込みサウンドを使って, ファンファーレを鳴らした後,「素晴らしい」と英語で発音するプログラムを作ってみましょう。

ヒント：　ファンファーレのファイル名は「fanfare.wav」です。
「素晴らしい」の英語のファイル名は「fantastic.wav」です。

第５章　モータ

この章からは，図１－３に示した車輪移動型ロボットを動かします。まず，EV3 ブロックの正面から見て向かって左側のモータを出力ポート B に接続し，右側のモータを出力ポート C に接続していることを確認してください。

EV3 ブロックには，L モータと M モータという２種類のモータが利用できます。ここで取り扱うモータは「L モータ」のみです。

５．１　ロボットの前後移動

いよいよロボットを動かすところまでやってきました。ロボットを動かすためには，モータを駆動させる必要があります。早速，プログラム（prog5-1.py）を入力しましょう。

【 prog5-1.py 】

```
 1 #!/usr/bin/env python3
 2 from time import sleep
 3 from ev3dev2.motor import LargeMotor,OUTPUT_B,OUTPUT_C
 4 Lmt=LargeMotor(OUTPUT_B)
 5 Rmt=LargeMotor(OUTPUT_C)
 6 Lmt.on(30)
 7 Rmt.on(30)
 8 sleep(4)
 9 Lmt.on(-30)
10 Rmt.on(-30)
11 sleep(4)
12 Lmt.off()
13 Rmt.off()
```

prog5-1.py は,ロボットを 4 秒間前進させ,その後 4 秒間後退させるプログラムです。それでは，新たに加わったプログラムを説明していきます。

3 行目　`from ev3dev2.motor import LargeMotor,OUTPUT_B,OUTPUT_C`

EV3 ブロックに接続できるモータと出力ポートの準備をします。

ev3dev2.motor は，EV3 ブロックに接続できるモータを制御するための様々なプログラムを集めたモジュールです。このモジュールの中から，L モータに対して制御する

LargeMotor という設計図（クラスと呼びます。）と出力ポート B, C に対する OUTPUT_B, OUTPUT_C という設計図（クラスと呼びます。）を読み込みます。

なお，出力ポートとして A や D を使うときは，OUTPUT_A，OUTPUT_D と書きます。

4 行目　Lmt=LargeMotor(OUTPUT_B)

モータを制御するためクラスである LargeMotor の実体（インスタンス）を出力ポート B に接続された左側の車輪を回転させるモータとして Lmt を作ります。

OUTPUT_B は，出力ポート B を示しています。

なお，出力ポートとして A, C, D も利用でき，それぞれ INPUT_A, INPUT_C, INPUT_D, 書きます。利用する出力ポートは，あらかじめ ev3dev2.motor から読み込んでおく必要があります。

5 行目　Rmt=LargeMotor(OUTPUT_C)

モータを制御するためクラスである LargeMotor の実体（インスタンス）を出力ポート C に接続された右側の車輪を回転させるモータとして Rmt を作ります。

OUTPUT_C は，出力ポート C を示しています。

6 行目　Lmt.on(30)

LargeMotor クラスの中にある on メソッドを実行して，左側のモータを回転させます。

on に続く丸カッコ( )の中に書いた数値は，回転するスピードを示し「30」を設定しています。このスピードは，-100～100 の間として，正の時は「正回転」，負の時は「逆回転」を示し，最も速いスピードを 100 または-100 として，その割合で指定します。

車輪移動型ロボットの場合，正回転は前進，逆回転は後進となります。

7 行目　Rmt.on(30)

LargeMotor クラスの中にある on メソッドを実行して，右側のモータを回転させます。

on に続く丸カッコ( )の中に書いた数値は，回転するスピードを示し「30」を設定しています。このスピードは，-100～100 の間として，正の時は「正回転」，負の時は「逆回転」を示しています。スピードを 0 に設定するとモータは停止します。

この時点で，左側と右側のモータが同じスピードで正回転するので，ロボットはまっすぐ前進を開始します。

8 行目　sleep(4)

sleep に続く丸カッコ( )の中に書いた数字は，待つ時間を「秒」の単位で指定しています。このプログラムでは，4 秒間待つことを示し，その間ロボットは前進を続けます。

9 行目　Lmt.on(-30)

LargeMotor クラスの中にある on メソッドを実行して,左側のモータを回転させます。

on に続く丸カッコ(　)の中に書いた数値は，回転するスピードを示し「-30」を設定していますので，逆回転します。

10 行目　Rmt.on(-30)

LargeMotor クラスの中にある on メソッドを実行して,右側のモータを回転させます。

on に続く丸カッコ(　)の中に書いた数値は，回転するスピードを示し「-30」を設定していますので，逆回転します。

この時点で,左側と右側のモータが同じスピードで逆回転するので,ロボットは後退を開始します。

11 行目　sleep(4)

sleep に続く丸カッコ(　)の中に書いた数字は，待つ時間を「秒」の単位で指定しています。このプログラムでは，4 秒間待つことを示し，その間ロボットは後退を続けます。

12 行目　Lmt.off()

LargeMotor クラスの中にある off メソッドを実行して,左側のモータを停止させます。

13 行目　Rmt.off()

LargeMotor クラスの中にある off メソッドを実行して,右側のモータを停止させます。

この時点で，左側と右側のモータが停止するので，ロボットの移動は止まります。

prog5-1.py のプログラムでは，左右のモータを個別に回転させたり，停止させたりしていましたから，厳密には同時に回転させたり，停止させたりできていませんでした。プログラムの実行速度は速いため，ロボットの動きを見てもわからなかっただけです。

それでは,左右の両方のモータを同時に回転させたり，停止させたりする方法について紹介します。

prog5-2.py を入力して実行して，prog5-1.py を実行したときのロボットの動きと比較してみましょう。

ロボットの動きは同じように見えるかもしれませんが，prog5-2.py では，両方のモータに対して制御をしています。

【 prog5-2.py 】

```
1 #!/usr/bin/env python3
2 from time import sleep
3 from ev3dev2.motor import MoveTank,OUTPUT_B,OUTPUT_C
4 rbt=MoveTank(OUTPUT_B,OUTPUT_C)
5 rbt.on(30,30)
6 sleep(4)
7 rbt.on(-30,-30)
8 sleep(4)
9 rbt.off()
```

２つのモータをセットにして制御するプログラムについて説明します。

3 行目　from ev3dev2.motor import MoveTank,OUTPUT_B,OUTPUT_C

EV3 ブロックに接続できるモータの組み合わせと出力ポートの準備をします。

ev3dev2.motor は，EV3 ブロックに接続できるモータを制御するための様々なプログラムを集めたモジュールです。このモジュールの中から，２つのモータを組み合わせて制御する MoveTank という設計図(クラスと呼びます。)と出力ポート B,C に対する OUTPUT_B, OUTPUT_C という設計図（クラスと呼びます。）を読み込みます。

4 行目　rbt=MoveTank(OUTPUT_B,OUTPUT_C)

２つのモータを組み合わせて制御するクラスである MoveTank の実体（インスタンス）を，出力ポート B に接続された左側の車輪を回転させるモータと出力ポート C に接続された右側の車輪を回転させるモータを組み合わせ rbt を作ります。

5 行目　rbt.on(30,30)

MoveTank クラスの中にある on メソッドを実行して，左右のモータを回転させます。

on に続く丸カッコ( )の中に書いた最初の数値は左側のモータを回転させるスピードを設定し，次の数値は右側のモータを回転させるスピードを設定しています。

両方とも「30」を設定していますので，ロボットは前進を開始します。

7 行目　rbt.on(-30,-30)

MoveTank クラスの中にある on メソッドを実行して，左右のモータを回転させます。

on に続く丸カッコ( )の中に書いた最初の数値は左側のモータを回転させるスピードを設定し，次の数値は右側のモータを回転させるスピードを設定しています。

両方とも「-30」を設定していますので，ロボットは後退を開始します。

9 行目　rbt.off()

MoveTank クラスの中にある off メソッドを実行して，左右のモータを停止させます。この時点で，両方のモータが停止するので，ロボットの移動は止まります。

【チャレンジ５－１】

prog5-2.py の前進や後退時のスピードの値をいろいろ変えて，車輪移動型ロボットの移動する速度を毎秒 cm の単位で計測しましょう。

５．２　ロボットの方向転換

前進と後退，それに移動する速度の変更までができるようになりました。次に，ロボットの移動方向を変える方向転換に挑戦しましょう。車輪移動型ロボットは，自動車と違いハンドルで車輪の角度を変えることによって移動方向を変えることができません。どのようにして方向転換するプログラムを作ればよいのでしょうか？

ロボットの左右の車輪は，それぞれ独立に回転スピードと回転方向を設定できます。左右の車輪の回転方向を変えることで，方向転換を実現します。この方法は，ブルドーザーなどのキャタピラーで移動する機械でも使われています。

ここでは，方向転換する方法として図５－１と図５－２の２種類を説明します。両図とも，車輪移動型ロボットを上から見た図になっています。矢印は車輪が進む方向を示し，◎は方向転換するときの中心となる位置を示しています。

図５－１は，片方の車輪だけを回転させて，もう一方の車輪を停止していますから，ロボットの本体は，停止した車輪を中心として，回転する車輪の方向に転換します。このテキストでは，このような方向転換のことを「カーブ」と呼ぶことにします。

図５－２は，両方の車輪を互いに逆方向に回転させていますから，ロボットの本体は，両方車輪を結ぶ線の真ん中を中心として，前進する車輪の方向に転換します。このテキストでは，このような方向転換のことを「ターン」と呼ぶことにします。「ターン」は，本体を移動することなく方向だけ転換する場合に便利です。

また，左右の車輪の回転方向とスピードを変化させることで「カーブ」と「ターン」の中間のような方向転換もできます。

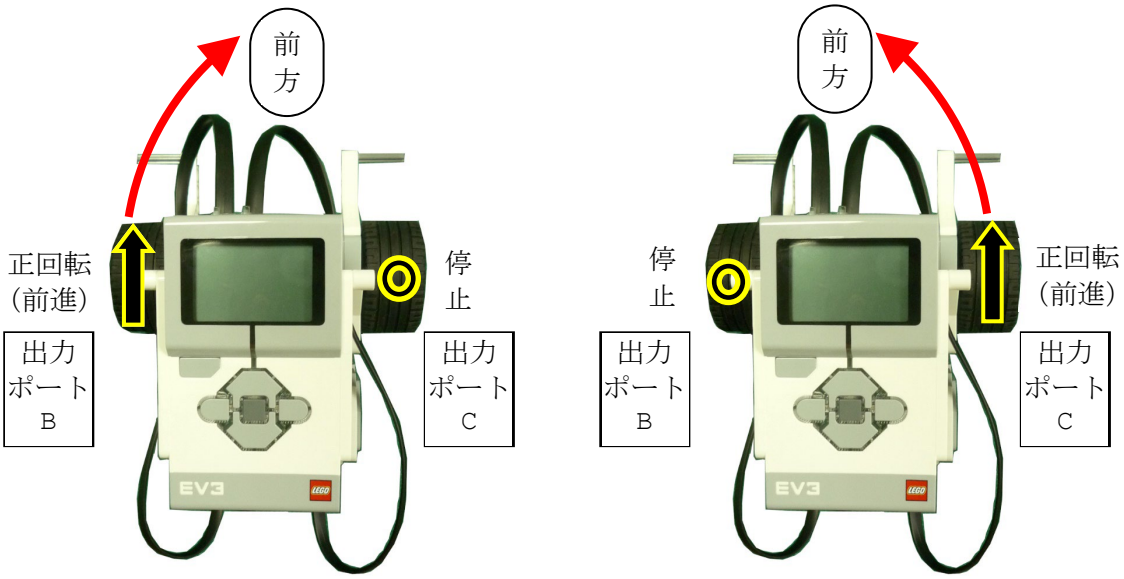


図５－１　「カーブ」で方向転換するときの車輪の回転方向と方向転換の中心位置（◎）。

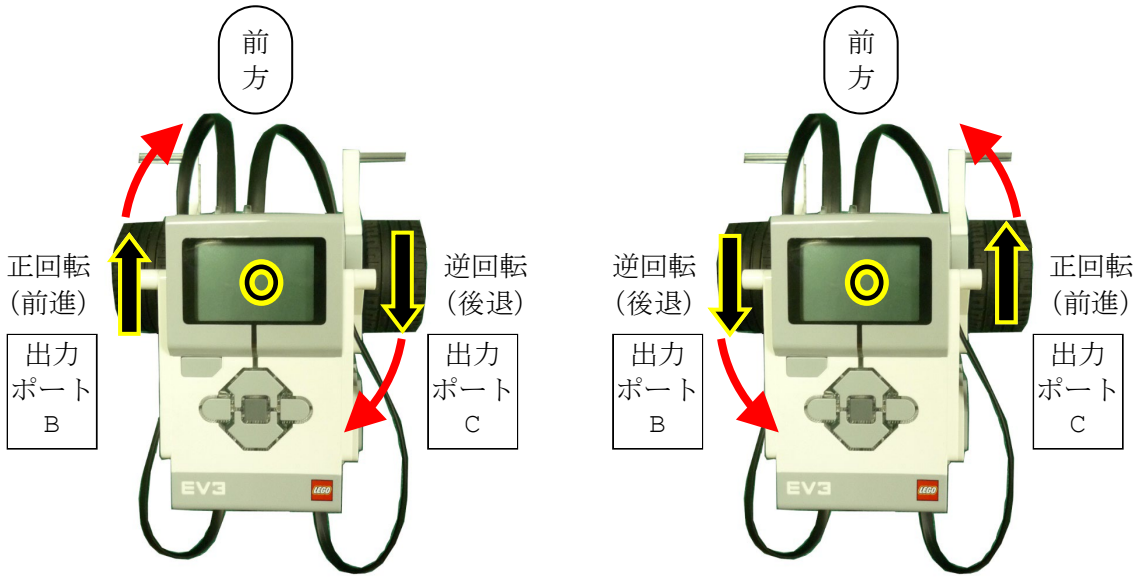


図５－２　「ターン」で方向転換するときの車輪の回転方向と方向転換の中心位置（◎）。

それでは，4 秒間右にカーブした後，4 秒間左にカーブして停止するプログラム prog5-3.py を入力し，ロボットの動きを確認しましょう。

【 prog5-3.py 】

```
1 #!/usr/bin/env python3
2 from time import sleep
3 from ev3dev2.motor import MoveTank,OUTPUT_B,OUTPUT_C
4 rbt=MoveTank(OUTPUT_B,OUTPUT_C)
5 rbt.on(30,0)
6 sleep(4)
7 rbt.on(0,30)
8 sleep(4)
9 rbt.off()
```

5 行目　rbt.on(30,0)

on に続く丸カッコ( )の中に書いた最初の数値は，出力ポート B につないでいる左側のモータを回転させるスピードを設定し,次の数値は出力ポート C につないでいる右側のモータを回転させるスピードを設定しています。

丸カッコ( )内の最初の数値は「30」なので左側のモータがスピード 30 で正回転します。次の数値は「0」なので，右側のモータは停止します。そのためロボットは，右にカーブを開始します。

7 行目　rbt.on(0,30)

丸カッコ( )内の最初の数値は「0」なので左側のモータは停止します。次の数値は「30」なので右側のモータがスピード 30 で正回転します。そのためロボットは，左にカーブを開始します。

次に，4 秒間右にターンした後，4 秒間左にターンして停止するプログラム prog5-4.py を入力し，ロボットの動きを確認しましょう。

【 prog5-4.py 】

```
1 #!/usr/bin/env python3
2 from time import sleep
3 from ev3dev2.motor import MoveTank,OUTPUT_B,OUTPUT_C
4 rbt=MoveTank(OUTPUT_B,OUTPUT_C)
5 rbt.on(30,-30)
6 sleep(4)
7 rbt.on(-30,30)
8 sleep(4)
9 rbt.off()
```

5 行目　rbt.on(30,-30)

on に続く丸カッコ( )の中に書いた最初の数値は，出力ポート B につないでいる左側のモータを回転させるスピードを設定し,次の数値は出力ポート C につないでいる右側のモータを回転させるスピードを設定しています。

丸カッコ( )内の最初の数値は「30」なので左側のモータがスピード 30 で正回転します。次の数値は「-30」なので，右側のモータがスピード 30 で逆回転します。。そのためロボットは，右にターンを開始します。

7 行目　rbt.on(-30,30)

丸カッコ( )内の最初の数値は「-30」なので左側のモータがスピード 30 で逆回転します。次の数値は「30」なので，右側のモータがスピード 30 で正回転します。そのためロボットは，左にターンを開始します。

モータを回転させてから時間待ちする処理はよく利用します。そのため，モータを指定した時間だけ回転させる on_for_seconds というメソッドがあります。このメソッドを使うと sleep を使わなくてもよいので，time モジュールも読み込む必要がなくなります。prog5-4.py を on_for_seconds を使ったプログラムに修正した prog5-5.py を示します。prog5-4.py とほぼ同じ動きをロボットがすることを確認しましょう。

【 prog5-5.py 】

```
1 #!/usr/bin/env python3
2 from ev3dev2.motor import MoveTank,OUTPUT_B,OUTPUT_C
3 rbt=MoveTank(OUTPUT_B,OUTPUT_C)
4 rbt.on_for_seconds(30,-30,4)
5 rbt.on_for_seconds(-30,30,4)
6 rbt.off()
```

4 行目　rbt.on_for_seconds(30,-30,4)

MoveTank クラスの中にある on_for_seconds メソッドを実行して，左右のモータを指定した時間だけ回転させます。

on_for_seconds に続く丸カッコ( )の中に書いた最初の数値は，出力ポート B につないでいる左側のモータを回転させるスピードを設定し，次の数値は出力ポート C につないでいる右側のモータを回転させるスピードを設定しています。最後の数値はモータを動作させる時間を秒の単位で指定します。

丸カッコ( )内の最初の数値は「30」なので左側のモータがスピード 30 で正回転します。次の数値は「-30」なので，右側のモータがスピード 30 で逆回転します。最後に 4 を指定しているので，ロボットは，4 秒間右にターンします。

【チャレンジ５－２】

2 秒間右にカーブした後，2 秒間まっすぐ前進し，2 秒間左にカーブするようにロボットを動かしてみましょう。

【チャレンジ５－３】

右にターンさせて，ロボットを３回転させるように動かしてみましょう。

５．３　ロボットを正確に移動させよう

ロボットを前進させたり，後退させたり，方向転換ができるようになりました。モータを動かすために「時間」を使っていました。そのため，スピードの設定を変えると，それに合わせて「時間」も適切に変えないと，同じ動きをさせることができませんでした。

EV3 ブロックに接続できるモータのうち「L モータ」と呼ばれるモータは，軸がどれだけ回転したかを知るためのセンサ（ロータリーエンコーダと呼びます。）を内蔵しています。

そのため，「時間」の代わりに「回転数」を指定してモータを動かすことができます。「回転数」は，スピードと関係ありません。

それでは，ロボットを時計回りに 360°ターンさせるプログラム prog5-6.py を入力して実行してみましょう。

【 prog5-6.py 】

```
1 #!/usr/bin/env python3
2 from ev3dev2.motor import MoveTank,OUTPUT_B,OUTPUT_C
3 rbt=MoveTank(OUTPUT_B,OUTPUT_C)
4 rbt.on_for_rotations(30,-30,2.2)
5 rbt.off()
```

4 行目　rbt.on_for_rotations(30,-30,2.2)

MoveTank クラスの中にある on_for_rotations メソッドを実行して，左右のモータを指定した回転数だけ軸を回転させます。

on_for_rotations に続く丸カッコ（　）の中に書いた最初の数値は，出力ポート B につないでいる左側のモータを回転させるスピードを設定し，次の数値は出力ポート C につないでいる右側のモータを回転させるスピードを設定しています。最後の数値はモータの軸を回転数を指定します。

丸カッコ（　）内の最初の数値は「30」なので左側のモータがスピード 30 で正回転します。次の数値は「-30」なので，右側のモータがスピード 30 で逆回転します。最後に「2.2」を指定しているので，ロボットは，両方のモータの軸が互いに逆方向に 2.2 回転し，時計回りに 360°ターンします。

なお，ロボットのタイヤの状態や設置する素材によって，設定した回転数を修正する必要があります。

【チャレンジ５－４】

prog5-6.py で，スピードを「30」から「20」に変えても，同じように時計回りに360°ターンすることを確かめてみましょう。

【チャレンジ５－５】

このロボットのタイヤの直径は，56mm です。300mm だけ前進するために，タイヤを何回転させればよいでしょうか？

左右のタイヤを指定しただけ回転させて 300mm だけ前進するプログラムを作って確かめてみましょう。

第６章　分かりやすいプログラムと繰り返し処理

６．１　変数を使ってみよう

「４．２ 音楽の演奏」で紹介したプログラム（prog4-5.py）でも少し説明しましたが，数値を変数に代入しておくと，その数値の意味がよく分かるようになります。また，プログラム中に何度も入力しなければならない同じ意味の数値を変数に置きかえると，それぞれの数値を変更しなくても，変数に代入する値を変更するだけですみます。

※　変数を使うときに注意しなければならないこと　※

① 変数名で利用できる文字は，a～z，A～Z，0～9，_（下線）です。
② 変数名の一番最初の文字に 0～9 は使えません。
③ 変数名に含まれる英字の大文字と小文字は区別されます。
④ 次の単語は，変数名として利用できません。

```
False None True and as assert break class continue def
del elif else except finally for from global if import
in is lambda nonlocal not or pass raise return try while
with yield
```

prog5-5.py は，4 秒間右にターンした後，4 秒間左にターンして停止するプログラムでした。このプログラムで，モータのスピードとターンする時間をそれぞれ，変数 sp と wt を使って修正したプログラム prog6-1.py を入力してみましょう。

【 prog6-1.py 】

```
1 #!/usr/bin/env python3
2 from ev3dev2.motor import MoveTank,OUTPUT_B,OUTPUT_C
3 rbt=MoveTank(OUTPUT_B,OUTPUT_C)
4 sp=30
5 wt=4
6 rbt.on_for_seconds(sp,-sp,wt)
7 rbt.on_for_seconds(-sp,sp,wt)
8 rbt.off()
```

それでは，変数を使っているところを説明します。

4 行目　sp=30

モータのスピードを示す「30」を変数 sp に代入しています。

5 行目　wt=4

ターンする時間を示す「4」を変数 wt に代入しています。

7 行目　rbt.on_for_seconds(sp,-sp,wt)

on_for_seconds に続く丸カッコ( )の中に書いた最初の数値は，出力ポート B につないでいる左側のモータを回転させるスピードを設定し，次の数値は出力ポート C につないでいる右側のモータを回転させるスピードを設定しています。最後の数値はモータを動作させる時間を秒の単位で指定します。

丸カッコ( )内に，「sp」，「-sp」，「wt」と書いていますので，左側のモータがスピード「sp」で正回転し，右側のモータがスピード「sp」で逆回転する動作を wt 秒間行います。

変数の前に，マイナス記号を付けると「-1」を掛け算したことになります。

sp には 30 が代入されているので，左側のモータはスピード 30 で正回転し，右側のモータはスピード 30 で逆回転します。さらに，wt に 4 が代入されているので，ロボットは，4 秒間右にターンします。

8 行目　rbt.on_for_seconds(-sp,sp,wt)

丸カッコ( )内に，「-sp」，「sp」，「wt」と書いていますので，左側のモータがスピード「sp」で逆回転し，右側のモータがスピード「sp」で正回転する動作を wt 秒間行います。

変数の前に，マイナス記号を付けると「-1」を掛け算したことになります。

sp には 30 が代入されているので，左側のモータはスピード 30 で逆回転し，右側のモータはスピード 30 で正回転します。さらに，wt に 4 が代入されているので，ロボットは，4 秒間左にターンします。

【チャレンジ６－１】

prog6-1.py で代入している値を変えて，いまより遅く動くロボットにしてみましょう。

６．２　同じ動きの繰り返し

例えば，時計回りに四角形を描く動きをロボットにさせたいときどのような手順を考えればよいでしょうか?

図６－１のように四角形を描くためには,「①前進する動き」と「②右に 90° 曲がる」という２種類の動きを４回実行しなければなりません。

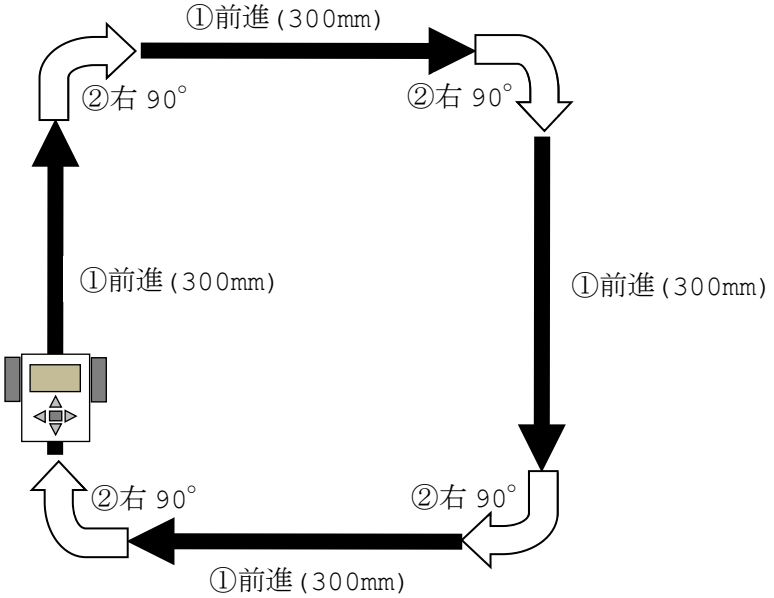


図６－１　時計回りに四角形を描くように移動する手順

まず，右に 90° ターンさせるために必要なロボットのタイヤの回転数を決めましょう。プログラム(prog6-2.py)を入力して，実行し，ロボットが約 90° 右にターンするように 6 行目の変数 rt に代入している数値「0.54」を修正しましょう。この値は，ロボットのタイヤと床面の状態で変化します。

【 prog6-2.py 】

```
1 #!/usr/bin/env python3
2 from ev3dev2.motor import MoveTank,OUTPUT_B,OUTPUT_C
3 rbt=MoveTank(OUTPUT_B,OUTPUT_C)
4 sp=20
5 rt=0.54
6 rbt.on_for_rotations(sp,-sp,rt)
7 rbt.off()
```

つぎに，四角形を描く動きをさせるプログラム（prog6-3.py）を入力しましょう。

5 行目の変数 rt に代入している数値「0.54」は，右に約 90° ターンするために必要なタイヤの回転数に修正します。

また，6 行目から 8 行目で 300mm だけ前進するために必要なタイヤの回転数を計算しています。まず，変数 len に前進する距離を代入します。変数 d はロボットのタイヤの直径を代入します。タイヤの周りの長さは，円周率×直径ですから，タイヤが 1 回転すると「円周率×直径」だけ前進します。そのため，変数 rs に代入する回転数は，前進する距離を「円周率×直径」で割って求めます。

数式をプログラムで書くとき，かけ算「×」は，アスタリスク「*」，割り算「÷」は，スラッシュ「/」で記述します。計算する順序を変えるときは，丸カッコ( )で囲みます。

【 prog6-3.py 】

```
 1 #!/usr/bin/env python3
 2 from ev3dev2.motor import MoveTank,OUTPUT_B,OUTPUT_C
 3 rbt=MoveTank(OUTPUT_B,OUTPUT_C)
 4 sp=20                  # スピード(%)
 5 rt=0.54                # 90° ターンに必要な回転数
 6 len=300                # 前進する距離(mm)
 7 d=56                   # タイヤの直径(mm)
 8 rs=len/(3.14*d)        # 前進に必要な回転数
 9
10 rbt.on_for_rotations(sp,sp,rs)   # 前進
11 rbt.on_for_rotations(sp,-sp,rt)  # 右ターン
12
13 rbt.on_for_rotations(sp,sp,rs)   # 前進
14 rbt.on_for_rotations(sp,-sp,rt)  # 右ターン
15
16 rbt.on_for_rotations(sp,sp,rs)   # 前進
17 rbt.on_for_rotations(sp,-sp,rt)  # 右ターン
18
19 rbt.on_for_rotations(sp,sp,rs)   # 前進
20 rbt.on_for_rotations(sp,-sp,rt)  # 右ターン
21
22 rbt.off()
```

prog6-3.py では，10 行目から 20 行目まで，「①前進」と「②右にターン」という制御を行うプログラムを 4 回繰り返して書いていることに気がつきましたか？ このように，同じプログラムを繰り返して書くことは，手間がかかり入力ミスも増えますし，プログラムが必要以上に長くなってしまいます。

プログラムを繰り返し実行するための文として，Python では，for 文があります。for 文を使って繰り返しを行うプログラム（prog6-4.py）を入力しましょう。

Python によるプログラムの記述では，半角の空白を使って段付け（インデントとも呼びます。）した行をひとまとまりとして扱います。このテキストでは，半角の空白を４つ入れて段付けします。ここでは，半角の空白がわかりやすいように，␣と表記してあります。

【 prog6-4.py 】

```
 1 #!/usr/bin/env python3
 2 from ev3dev2.motor import MoveTank,OUTPUT_B,OUTPUT_C
 3 rbt=MoveTank(OUTPUT_B,OUTPUT_C)
 4 sp=20                  # スピード(%)
 5 rt=0.54                # 90°ターンに必要な回転数
 6 len=300                # 前進する距離(mm)
 7 d=56                   # タイヤの直径(mm)
 8 rs=len/(3.14*d)        # 前進に必要な回転数
 9
10 for i in range(4):
11 ␣␣␣␣rbt.on_for_rotations(sp,sp,rs)      # 前進
12 ␣␣␣␣rbt.on_for_rotations(sp,-sp,rt)     # 右ターン
13
14 rbt.off()
```

prog6-3.py と比べて，プログラムの長さが短くなっていることが分かるでしょう。それでは，新たに加わったところを説明していきます。

8 行目　for i in range(4):
9 行目　␣␣␣␣rbt.on_for_rotations(sp,sp,rs)
10 行目　␣␣␣␣rbt.on_for_rotations(sp,-sp,rt)

for 文と range という関数を使って，繰り返し処理を行います。繰り返す回数は range 関数の次の丸カッコ( )の中に書きます。

また，for 文の最後に「:」を忘れないようにしましょう。

range 関数の丸カッコ内に「4」と書いていますから，段付けされた 9 行目と 10 行目までをひとまとまりとして，4 回繰り返し実行します。

for 文と range 関数の詳しい説明は８章でします。なお，for 文の次に書いている i は変数名です。変数 i の値は，0 から始まり繰り返し毎に 1 つずつ増えていきますが，このプログラムでは利用していません。

【チャレンジ６－２】

prog6-3.py は，時計回りに四角形を描くプログラムです。このプログラムを修正して反時計回りに四角形を描くようなプログラムを考えましょう。

【チャレンジ６－３】

三角形を描くような動きをさせるにはどのようなプログラムにすればよいでしょうか？

６．３　入れ子を使った繰り返し

みなさんは「入れ子」という言葉を知っていますか？ 台所で使うザルは，大きいザルに少し小さいザルを重ねて置くことができるようになっていて，このような特徴をもつ道具を入れ子といいます。同じようなものに，ロシアの伝統的なおもちゃのマトリョーシカがあり，かわいい人形の中に，少し小さい人形が入っています。

ロボットの動きにも，ある動作の中に別の動作を入れ子のように組み合わせることができます。図６－２のように大きな四角形の中に小さな四角形の動きを反時計回りにロボットにさせるには，どのようにすればよいでしょうか？

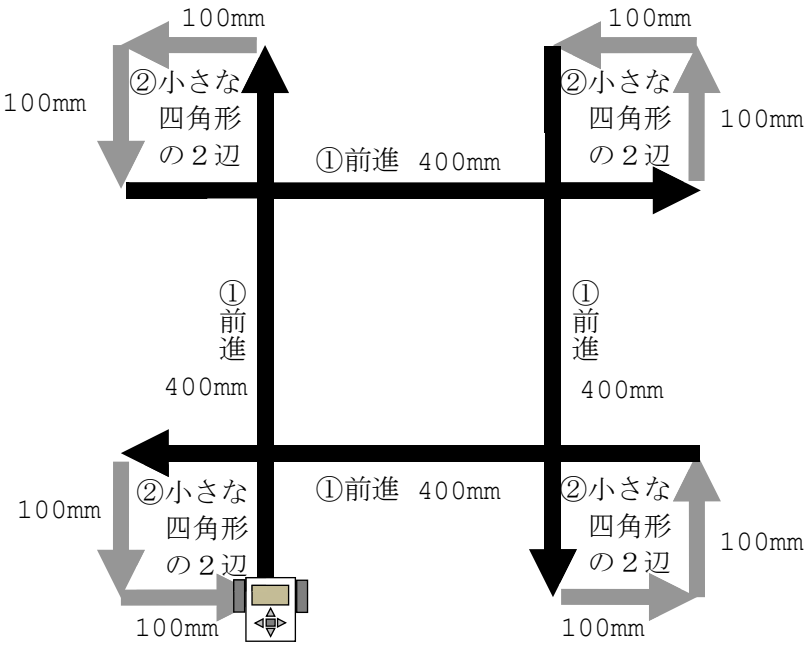


図６－２　大きな四角形の中に小さな四角形を含む動き

図６－２に示す動きをさせるためには，大きな四角形を描くための①直進と左ターン，②小さな四角形の２辺を描くための直進と左ターンを２回行う動作，を４回実行すればよいことがわかります。

大きな四角形の動きを実現するために４回行う for 文の中に，②の動きを行う for 文を入れ込むことによって，このような動きをするプログラムを作成できます。prog6-5.py を見てみましょう。

6 目の変数 rt に代入している数値「0.54」は，左に約 90° ターンするために必要なモータの回転数に修正します。

また，7 行目から 11 行目で，大きな四角形と小さな四角形を前進するために必要なタイ

ヤの回転数をそれぞれ計算し，変数 rsa と rsb に代入しています。

正確な円周率の値は，2 行目で，math というモジュールを読み込んでおくと，math.pi と書くことができます。

【 prog6-5.py 】

```
 1 #!/usr/bin/env python3
 2 import math
 3 from ev3dev2.motor import MoveTank,OUTPUT_B,OUTPUT_C
 4 rbt=MoveTank(OUTPUT_B,OUTPUT_C)
 5 sp=20                  # スピード(%)
 6 rt=0.54                # 90° ターンに必要な回転数
 7 lena=400               # 大きな四角形で前進する距離(mm)
 8 lenb=100               # 小さな四角形で前進する距離(mm)
 9 tire=math.pi*56        # タイヤの円周(mm)
10 rsa=lena/tire          # 大きな四角形の前進に必要な回転数
11 rsb=lenb/tire          # 小さな四角形の前進に必要な回転数
12
13 for i in range(4):
14 ␣␣␣␣rbt.on_for_rotations(sp,sp,rsa)     # 前進
15 ␣␣␣␣rbt.on_for_rotations(-sp,sp,rt)     # 左にターン
16
17 ␣␣␣␣for j in range(2):
18 ␣␣␣␣␣␣␣␣rbt.on_for_rotations(sp,sp,rsb) # 前進
19 ␣␣␣␣␣␣␣␣rbt.on_for_rotations(-sp,sp,rt) # 左にターン
20
21 rbt.off()
```

9 行目　for i in range(4):

①の大きな四角形の４辺を描くための繰り返し処理です。

range 関数の丸カッコ内に「4」と書いていますから，段付けされた 10 行目から 13 行目までをひとまとまりとして，4 回繰り返し実行します。

なお，for 文の次に書いている i は変数名です。変数 i の値は，0 から始まり繰り返し毎に 1 つずつ増えていきますが，このプログラムでは利用していません。

10 行目　␣␣␣␣rbt.on_for_rotations(sp,sp,rsa)

11 行目　␣␣␣␣rbt.on_for_rotations(-sp,sp,rt)

大きな四角形を描くように前進を実行し，左に 90° ターンするプログラムです。

13 行目　␣␣␣␣for j in range(2):

②の小さな四角形の２辺を描くための繰り返し処理です。

range 関数の丸カッコ内に「2」と書いていますから，二段階に段付けされた 14 行目と 15 行目までをひとまとまりとして，2 回繰り返し実行します。

なお，for 文の次に書いている j は変数名です。変数 j の値は，0 から始まり繰り返し毎に 1 つずつ増えていきますが，このプログラムでは利用していません。

9 行目の for 文で使った i と同じにすることもできますが,通常異なる変数名にします。

```
14 行目　　rbt.on_for_rotations(sp,sp,rsb)
15 行目　　rbt.on_for_rotations(-sp,sp,rt)
```

小さな四角形を描くように前進を実行し，左に 90°ターンするプログラムです。

このプログラムでは，9 行目の for 文で 4 回繰り返し処理が実行され，さらにその繰り返し処理ごとに，13 行目の for 文で 2 回繰り返し処理が実行されます。このように for 文の中に for 文が含まれているプログラムを「入れ子構造」をもつと言います。

【チャレンジ６－４】

図６－３に示すように，ジグザグに動きながら四角形を描く動きをするプログラムを作りましょう。

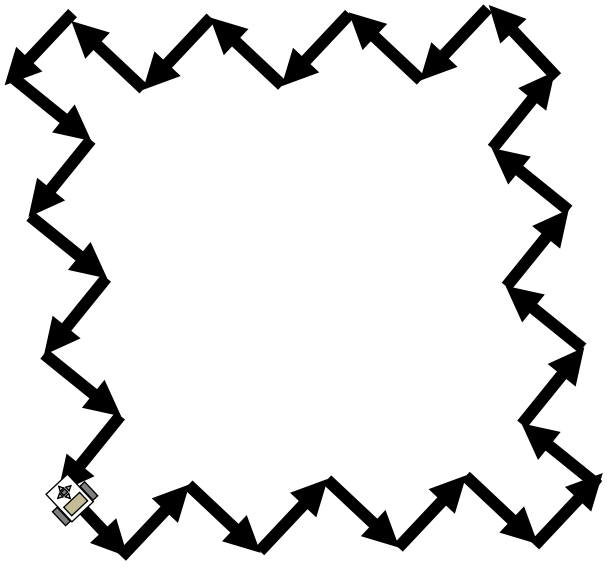


図６－３　ジグザグに動きながら四角形を描く動き

６．４　コメントを使ってプログラムを説明

プログラムを分かりやすくするために，プログラム中にロボットの動きや命令の意味を説明する注釈（コメント）を書きます。コメントには日本語を利用できます。コメントはプログラムの実行にまったく影響しませんが，適切な内容を書いておくことで，プログラムを読みやすくできます。

また，一部のプログラムを無効にしたいときに，その部分をコメントにしてしまうという使い方もよくします。

最後までを注釈にするときは，「#」から行の最後までがコメントになります。

このテキストで紹介しているプログラムの１行目の「#」はコメントではなく，python を指定する意味なので，コメントは 2 行目以降に書くことができます。

【例】

```
sp=10   # 行末までがコメントです。
```

いくつかの行にまたがってコメントをするときには，「'」一重引用符（シングルクォーテーション）または「"」二重引用符（ダブルクォーテーション）３つで囲みます。なお，段付けは同じにしておきます。

【例】

```
'''
この行からコメントです。
途中の行もコメントです。
この行までコメントです。
'''
```

【例】

```
"""
この行からコメントです。
途中の行もコメントです。
この行までコメントです。
"""
```

ロボットの動きをうまくコメントとしてプログラムに書いておくと，後でプログラムを見直すときに参考になります。

prog6-5.py にさらにコメントを追加すると，prog6-6.py のようになります。prog6-6.py を入力しましょう。コメント部分は自分の分かりやすいように工夫して入力しましょう。

【 prog6-6.py 】

```
#!/usr/bin/env python3
'''
  大きな四角形に小さな四角形を組み合わせた動作をするプログラム
      作成日：2020 年 5 月 6 日
      作成者：鳴門　渦美
'''
import math
from ev3dev2.motor import MoveTank,OUTPUT_B,OUTPUT_C
rbt=MoveTank(OUTPUT_B,OUTPUT_C)
sp=20                  # スピード(%)
rt=0.54                # 90° ターンに必要な回転数
lena=400               # 大きな四角形で前進する距離(mm)
lenb=100               # 小さな四角形で前進する距離(mm)
tire=math.pi*56        # タイヤの円周(mm)
rsa=lena/tire          # 大きな四角形の前進に必要な回転数
rsb=lenb/tire          # 小さな四角形の前進に必要な回転数

for i in range(4):
    rbt.on_for_rotations(sp,sp,rsa)      # 前進
    rbt.on_for_rotations(-sp,sp,rt)     # 左にターン

    for j in range(2):
        rbt.on_for_rotations(sp,sp,rsb)   # 前進
        rbt.on_for_rotations(-sp,sp,rt) # 左にターン

rbt.off()
```

第７章　少し進んだプログラム

７．１　変数の使い方（うずまきを描くロボット）

より動きに変化のあるロボットを作るためには，プログラムを実行している途中で，動きを決めている値を変える必要があります。このような値を入れておくところとして，変数を使います。変数とは，数値を入れておける箱のようなものと考えればよいでしょう。

例えば，sleep(4)というプログラムでは，「4」という数値を実行中に変更することはできません。プログラム中に書かれた数値は，実行中に変更できませんが，変数は何回でもその値を変更できます。

図７－１のようにうずまき状に四角を描いていく動きをするロボットを考えましょう。うずまき状ですから，前進する時間が少しずつ長くなっていく必要があります。前進する時間に変数を使って，段々と長くなるようにプログラムを作ります。

図７－１の動きをするプログラムを prog7-1.py に示します。

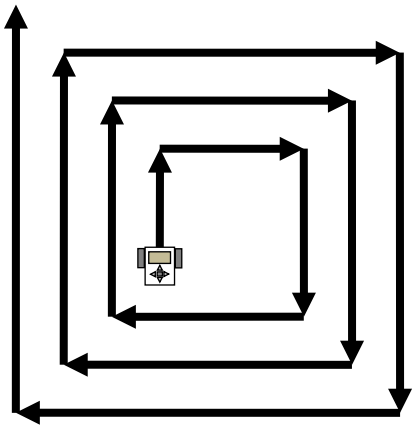


図７－１　うずまき状の四角形を描きながら移動

【 prog7-1.py 】

```
 1 #!/usr/bin/env python3
 2 from ev3dev2.motor import MoveTank,OUTPUT_B,OUTPUT_C
 3 rbt=MoveTank(OUTPUT_B,OUTPUT_C)
 4 sp=20              # スピード(%)
 5 wt=1.1             # 90°ターンする時間(秒)
 6 ws=1               # 前進する時間(秒)
 7
 8 for i in range(10):
 9 ␣␣␣␣rbt.on_for_seconds(sp,sp,ws)       # ws 秒間前進
10 ␣␣␣␣rbt.on_for_seconds(sp,-sp,wt)      # wt 秒間右にターン
11 ␣␣␣␣ws += 0.5                          # 前進する時間を増加
12
13 rbt.off()
```

それでは新たに加わったプログラムの書き方について説明します。

5 行目　wt=1.1

90°ターンする時間 1.1 秒を変数 wt に代入しています。この値は，状態によって変わります。

6 行目　ws=1

一番最初に前進する時間として 1 秒を変数 ws に代入しています。変数の最初に代入する値を初期値と呼びます。この様子を表７－１に示します。

8 行目　for i in range(10):

9 行目　␣␣␣␣rbt.on_for_seconds(sp,sp,ws)

10 行目　␣␣␣␣rbt.on_for_seconds(sp,-sp,wt)

うずまき状に四角形を描くようにするためには，前進する時間を段々と長くする必要があります。

for 文の次にある 9 行目から 11 行目までは,半角の空白４文字で段付けしておきます。

ここでは，ws 秒だけ前進(9 行目)と右に 90°ターン(10 行目)を 10 回実行する繰り返し処理を 8 行目から 11 行目に書いています。

11 行目　␣␣␣␣ws += 0.5

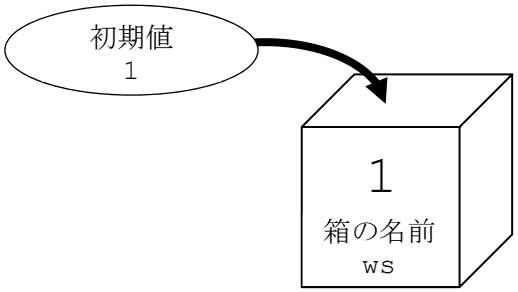
前進する時間を段々と長くするために，変数 ws の値を増加させていきます。

ws += 0.5 を実行するたびに ws という変数の値に 0.5 が加えられています。この処理の様子を表７－１に詳しく説明しています。

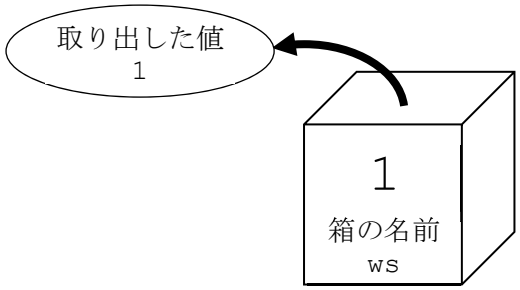
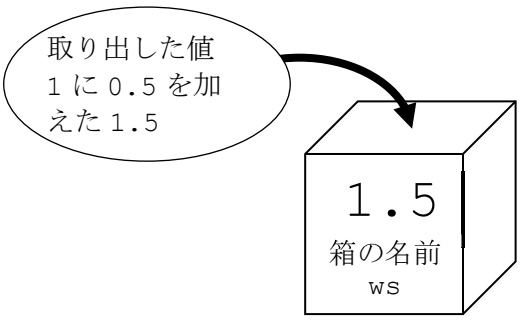
ws の値は，はじめは 1 だったものが，1.5，2，2.5，3，･･･と 0.5 ずつ 10 回加えられていきますから，最後には 6 になります。

変数 ws で指定された前進する時間がだんだんと長くなることになります。左に 90° ターンするたびに前進する時間が長くなり，うずまきを描くようにロボットが移動します。

表７－１　prog7-1.py における変数の値の変化

<table>
<tr><th>行番号</th><th colspan="2">内　　容</th><th>意　　味</th></tr>
<tr><td>6</td><td colspan="2">変数は，数値を入れる「箱」と考えます。箱の名前は「変数名」に対応します。
最初に入れる値のことを「初期値」といい，変数に値を入れることを「代入」と呼びます。</td><td> 
</td></tr>
<tr><td rowspan="2">11</td><td rowspan="2">ws += 0.5 の意味

ws + 0.5
↓
ws

ws の値は，0.5 だけ増加します。</td><td>ws の箱から値 1 を取り出します。</td><td> 
</td></tr>
<tr><td>取り出した値 1 に 0.5 を加えた値 1.5 を ws の箱に再び入れます。</td><td> 
</td></tr>
</table>

【チャレンジ７－１】

渦巻状の三角形を描くような動きをさせるにはどのようなプログラムにすればよいでしょうか？

７．２　数式の書き方

変数が使えるようになると，様々な計算もできるようになります。Python で利用できる数式の書き方を表７－１にまとめました。

表７－１　数式の書き方

記号	意　　味	例	例　の　意　味
+	加算	a + b	a に b を加える
-	減算	a - b	a から b を引く
-	符号変換	-a	a の符号を反対にする
*	乗算（×）	a * b	a と b をかける
/	除算（÷）	a / b	a を b で割る
//	整数除算（÷）	a // b	a を b で割り，整数化する。
%	剰余	a % b	a を b で割った余り
**	累乗	a ** 3	a を 3 乗する

数学と同じように，計算の種類によって優先順位が付けられています。その順番を変えたいときは，丸カッコ ( ) で囲みます。丸カッコの中にさらに丸カッコを書くように使ってもかまいません。

例えば，

$$x = \frac{c}{a \times b}$$

という計算式を

```
x = c / a * b
```

のように書くと，「c を a で割った値に b をかける」ことになりますから

```
x = c / (a * b)
```

のように，分母は丸カッコを使って書きます。

つぎに，代入と数式をまとめた書き方を表７－２に示します。

表７－２　代入と数式をまとめた書き方

記号	意　　味	例	例　の　意　味
+=	加算代入	a += b	a = a + b と同じ
-=	減算代入	a -= b	a = a - b と同じ
*=	乗算代入	a *= b	a = a * b と同じ
/=	除算代入	a /= b	a = a / b と同じ
//=	整数除算代入	a //= b	a = a // b と同じ
%=	剰余代入	a %= b	a = a % b と同じ
**=	累乗代入	a **= 3	a = a ** 3 と同じ

さらに，円周率πの値は，

```
import math
```

として数学関係のモジュールを読み込んだ後に

```
math.pi
```

と書くことで利用できます。

また，x の平方根は，

```
math.sqrt(x)
```

と書き，数式の中に書くことができます。

例えば，直角三角形の直角をはさむ２辺の長さが変数 a と b にそれぞれ入っているとき，斜辺の長さ c は，

```
c = math.sqrt(a ** 2 + b ** 2)
```

で求められます。

７．３　乱数の使い方

ロボットが虫のようにランダムな動きをしたら面白いと思いませんか。ここでは，このような動きを乱数を使って作ります。例えば，サイコロをふったときの目の数のように，予測できない値を乱数といいます。

これまでのプログラムに含まれる数値や変数の値は，あらかじめ決まった値しかとりませんでした。予測できない値（乱数）を得るためには，random モジュールを使います。このモジュールのうち，指定された範囲の乱数を返す uniform 関数を使って，動きが予測できない虫のようなロボットを作ります。

prog7-2.py を実行すると，予測できない前進や左ターンを繰り返します。同じ動きを繰り返すことがないことを確認しましょう。

【 prog7-2.py 】

```
#!/usr/bin/env python3
from random import uniform
from ev3dev2.motor import MoveTank,OUTPUT_B,OUTPUT_C
rbt=MoveTank(OUTPUT_B,OUTPUT_C)
sp=30                                    # スピード(%)

while True:
    ws=uniform(0.1,0.6)                  # 前進する時間(秒)
    wt=uniform(0.1,0.4)                  # ターンする時間(秒)
    rbt.on_for_seconds(sp,sp,ws)         # ws 秒間前進
    rbt.on_for_seconds(-sp,sp,wt)        # wt 秒間左にターン
```

このプログラムを実行するとロボットは動き続け終了しません。
プログラムを終了させるときは，VS Code の右上に表示されている

のなかから，  を左クリックします。

また，EV3 ブロック本体の「戻るボタン」を押しても，プログラムを終了させることができます。

prog7-2.py は，ランダムな時間だけ前進したりターンしたりするプログラムです。それでは新たに加わったプログラムを詳しく説明していきます。

2 行目　`from random import uniform`

乱数を処理するためのプログラムを準備します。
random は，乱数を処理するための様々なプログラムを集めたモジュールです。このモジュールの中から，uniform という関数を読み込みます。

7 行目　`while True:`

while 文を使って繰り返し処理をします。while の条件式がいつも True（真）なので，段付けされた 6 行目から 11 行目までのプログラムを無限に繰り返し実行します。
そのため，プログラムを終了させるためには，VS Code から指示したり，EV3 ブロック本体の「戻るボタン」を押したりする必要があります。
while 文に書く「True」の最初の文字が大文字の「T」であることに注意しましょう。

8 行目　`␣␣␣␣ws=uniform(0.1,0.6)`

変数 ws に直進する時間を uniform 関数を使って得られた乱数の値を代入しています。uniform 関数は，丸カッコ( )の最初の数値と次の数値の間の値をランダムに返します。
ここでは，0.1 から 0.6 までのどれかの値が変数 ws に代入されます。そのため，前進する時間は 0.1 秒から 0.6 秒までの値となります。

9 行目　`␣␣␣␣wt=uniform(0.1,0.4)`

変数 wt にターンする時間を uniform 関数を使って得られた乱数の値を代入しています。uniform 関数は，丸カッコ( )の最初の数値と次の数値の間の値をランダムに返します。
ここでは，0.1 から 0.4 までのどれかの値が変数 ws に代入されます。そのため，ター

ンする時間は 0.1 秒から 0.4 秒までの値となります。

【チャレンジ７－２】

乱数を使って前進, 後進, 右ターン, 左ターンをいろいろと繰り返すプログラムを作ってみましょう。

７．４　LED の使い方

図１－１に示したように EV3 ブロックには，2 個の LED（発光ダイオード）があります。各 LED は，消灯させたり，赤，緑，茶，オレンジ，黄の 5 色から一つを選んで光らせたりすることができます。

プログラム prog8-3.py は，実行開始後，一旦左右の LED を消灯し，「赤」，「緑」で 3 秒間光らせ，「茶」，「オレンジ」で 3 秒間光らせ，最後に「黄」，「黒」で 3 秒間光らせます。「黒」で光らせることは，「消灯」と同じ結果になります。

【 prog7-3.py 】

```
 1 #!/usr/bin/env python3
 2 from time import sleep
 3 from ev3dev2.led import Leds
 4 led = Leds()
 5
 6 led.all_off()          # LED を消灯
 7 sleep(2)
 8
 9 led.set_color("LEFT",  "RED")
10 led.set_color("RIGHT", "GREEN")
11 sleep(3)
12
13 led.set_color("LEFT",  "AMBER")
14 led.set_color("RIGHT", "ORANGE")
15 sleep(3)
16
17 led.set_color("LEFT",  "YELLOW")
18 led.set_color("RIGHT", "BLACK")
19 sleep(3)
20
21 led.all_off()          # LED を消灯
22 sleep(2)
```

それでは，LED に関係するところを説明します。

3 行目　from ev3dev2.led import Leds

LED に対する処理を行うためのプログラムを準備します。

ev3dev2.led は，EV3 ブロックの LED に対して処理するための様々なプログラムを集めたモジュールです。このモジュールの中から，Leds という設計図（クラスと呼びます。）

を読み込みます。

4 行目　led=Leds()

LED に対する処理するためクラスである Leds の実体となる led を作ります。この実体をインスタンスと呼びます。Python によるプログラムでは，設計図（クラス）に基づいて具体化した実体（インスタンス）を作る必要があります。

6 行目　led.all_off()

プログラムを実行している状況を示すように左右の LED が緑色で点滅します。そのため，Leds クラスの中にある all_off という処理（メソッドと呼びます）を実行して，左右の LED を消灯します。

9 行目　led.set_color("LEFT",　"RED")

Leds クラスの中にある set_color という処理（メソッドと呼びます）を実行して，左の LED を「赤」で点灯します。

set_color の丸カッコ(　)の次が，"LEFT"のとき，左側の LED を，"RIGHT"のとき，右側の LED を対象とし，次の文字列で色を指定します。

文字列と色は次の通りです。

文字列	色
"RED"	赤
"GREEN"	緑
"AMBER"	茶
"ORANGE"	オレンジ
"YELLOW"	黄
"BLACK"	黒（消灯）

【チャレンジ７－３】

左右の LED を赤色と緑色で，それぞれ 0.5 秒ごとに交互に点滅するプログラムを作ってみましょう。

第８章　処理の順番を変えるプログラム

今まで，繰り返し処理を行うために for 文や while 文を紹介しました。ここでは処理の順番を変えるプログラムで使う「制御文」について説明をします。

８．１　if 文

ある条件が成り立つときに実行したい処理があるときに使う文です。

身近な例をあげると交差点の信号による動作があります。交差点の信号が緑色のとき進みますが，赤色のときには止まります。この例を if 文で表現すると図８－１のようになります。

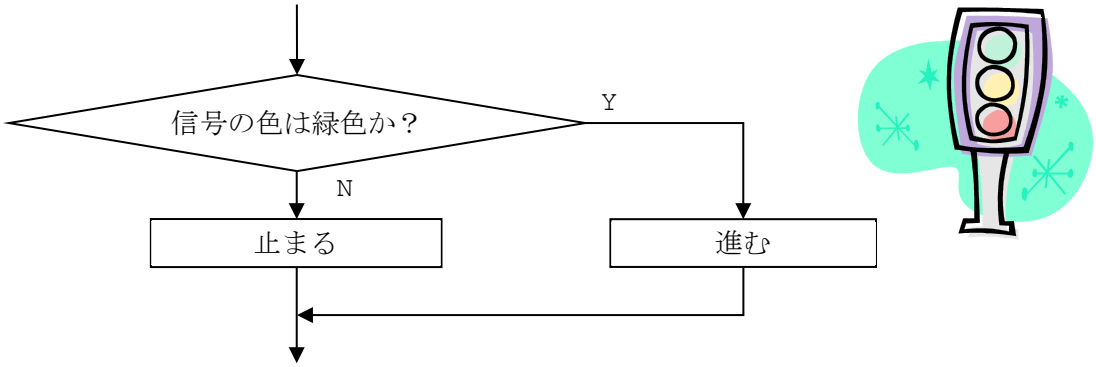


図８－１　交差点の信号による動作

このような図のことを「フローチャート」といいます。フローチャートは，矢印を使って順番に実行する内容を示します。実行する内容は長方形の枠の中に書きます。条件によって実行する内容が変わる場合は，ひし形を使ってその中に条件を書きます。ひし形の横と下にある「Y」と「N」は，それぞれ「YES」，「NO」を意味します。「Y」は，ひし形の中に書いた条件「信号の色が緑色である」が正しいとき，実行する方向を示します。「N」は，条件が正しくないとき，実行する方向を示します。

条件が正しい（成り立つ）ときを「真(True)」，正しくない（成り立たない）ときを「偽(False)」ともいいます。

図８－１のフローチャートを参考にして，信号が緑色ならば「Go」，赤色ならば「Stop」

としゃべるプログラムを作るとprog8-1.pyのようになります。このプログラムでは，信号は緑色になっているとしています。

【 prog8-1.py 】

```
1 #!/usr/bin/env python3
2 from ev3dev2.sound import Sound
3 spk=Sound()
4 signal = 'Green'
5 if signal == 'Green':
6 ␣␣␣␣spk.speak('Go.')
7 else:
8 ␣␣␣␣spk.speak('Stop.')
```

4行目　signal = 'Green'

現在の信号の状態を示すための変数signalに，緑色を意味する文字列「Green」を代入しています。

5行目　if signal == 'Green':

if文を使って条件によって実行する手順を変えます。

条件（プログラミング言語では「条件式」と呼びます。）は，if文とコロン「:」の間に書きます。

ここでは，「信号の色は緑色か？」という条件を「signal == 'Green'」という条件式で書いています。変数signalの値が，文字列「Green」と同じかという意味です。数学と異なり，条件式で同じかどうかを意味する場合「==」のように「=」を二つ続けて並べることに注意しましょう。

6行目　␣␣␣␣spk.speak('Go.')

5行目のif文の条件式が正しいときに実行する処理です。信号の色が緑色のとき，「信号の色は緑色か？」の条件式は正しいと判断されるので，この処理が実行され，「Go」と発音されます。

このようにif文の条件式が正しいとき，条件式は「真（True)」または「成り立つ」といいます。

7行目　else:

5行目のif文の条件式が正しくないときに実行する処理を書くために必要な文です。

8 行目　␣␣␣␣spk.speak('Stop.')

5 行目の if 文の条件式が正しくないときに実行する処理です。例えば，信号の色が赤色のとき,「信号の色は緑色か？」の条件は正しくないと判断され, この処理が実行され,「Stop」と発音されます。

このように if 文の条件式が正しくないとき，条件は「偽（False)」または「成り立たない」といいます。

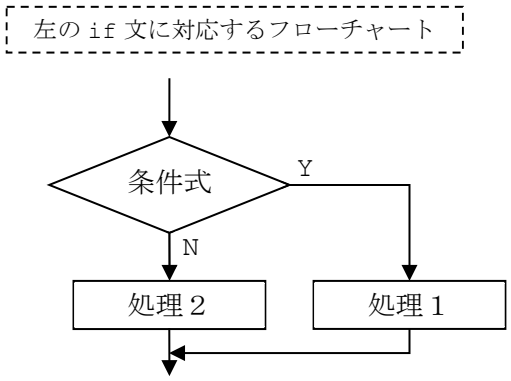
プログラム prog8-1.py の 4 行目を

signal = 'Red'

に修正して，実行して，if 文の条件式が成り立たない場合の処理結果を確かめましょう。

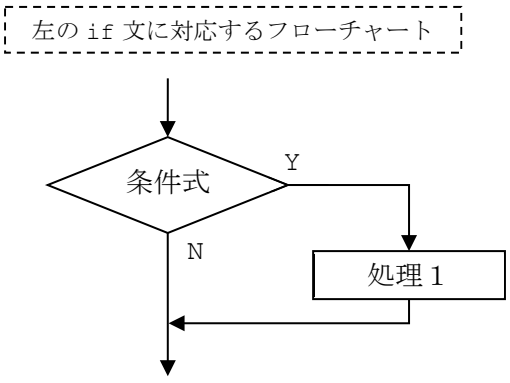
if 文の条件が正しい（Y，True，成り立つ）のとき命令１を実行し，正しくない（N，False，成り立たない）のとき命令２を実行する if 文は，次のように書きます。

```
if 条件式:
␣␣␣␣処理１
else:
␣␣␣␣処理２
```



とくに，実行する処理２がないときは，else 以降を省略することができ，

```
if 条件式:
␣␣␣␣処理１
```



と書くこともできます。

さらに，処理１が１行だけのときは，

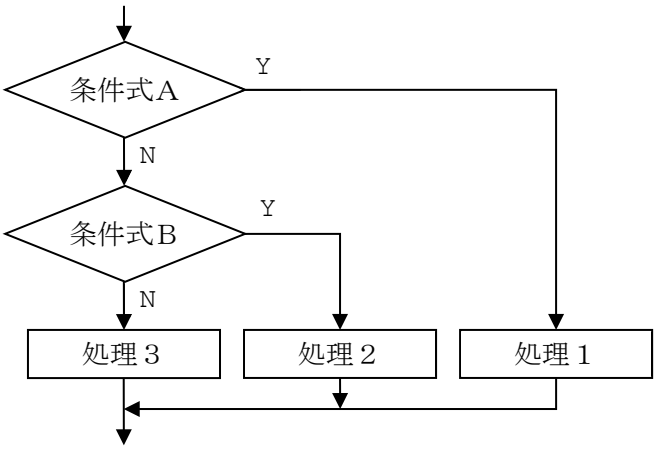
```
if 条件式: 処理１
```

と短く書くこともできます。

また，２つの条件式Ａと条件式Ｂがあり３通りに分けたいときは，if 文を次のように書きます。

```
if 条件式A:
␣␣␣␣処理１
elif 条件式B:
␣␣␣␣処理２
else:
␣␣␣␣処理３
```

左の if 文に対応するフローチャート



この場合，条件式Aが正しい（Y，True，成り立つ）のとき，処理１を実行します。条件式Aが正しくない（N，False，成り立たない）であり，条件式Bが正しい（Y，True，成り立つ）のとき，命令２を実行し，条件式Bが正しくない（N，False，成り立たない）のとき命令３を実行します。

prog8-2.py を実行すると，2 秒間前進した後，右または左に 0.5 秒間ターンする動きを繰り返します。この動きを実現するために乱数を用います。ここで使う乱数は，0 から 1 の値をとり，その値が 0.5 より小さいときはロボットが右にターンし，そうでないときは左にターンするように制御します。

【 prog8-2.py 】

```
 1 #!/usr/bin/env python3
 2 from random import uniform
 3 from ev3dev2.motor import MoveTank,OUTPUT_B,OUTPUT_C
 4 rbt=MoveTank(OUTPUT_B,OUTPUT_C)
 5 sp=20              # スピード(%)
 6 wt=0.5             # ターンする時間(秒)
 7
 8 while True:
 9 ␣␣␣␣rbt.on_for_seconds(sp,sp,2)           # 2 秒間前進
10 ␣␣␣␣if uniform(0,1) < 0.5:
11 ␣␣␣␣␣␣␣␣rbt.on_for_seconds(sp,-sp,wt)     # wt 秒間右にターン
12 ␣␣␣␣else:
13 ␣␣␣␣␣␣␣␣rbt.on_for_seconds(-sp,sp,wt)     # wt 秒間左にターン
```

8 行目　while True:

9 行目から 13 行目にかけて段付けした範囲の処理を無限に繰り返します。

10 行目　␣␣␣␣if uniform(0,1) < 0.5:

if 文の条件式「uniform(0,1) < 0.5」は，「uniform(0,1)で得られた乱数（0 から 1 の間の数）の値が 0.5 より小さいか？」という条件になっています。

２つの値の比較には，表８－１に示す記号を使います。

11 行目　␣␣␣␣␣␣␣␣rbt.on_for_seconds(sp,-sp,wt)

10 行目の if 文の条件式が正しいときに実行し，wt 秒間右にターンします。

段付けする空白の数が２倍にふえていることに注意しましょう。

12 行目　␣␣␣␣else:

10 行目の if 文と組み合わせて書いています。

13 行目　␣␣␣␣␣␣␣␣rbt.on_for_seconds(-sp,sp,wt)

10 行目の if 文の条件式が正しくないときに実行し，wt 秒間左にターンします。

段付けする空白の数が 11 行目と同じであることに注意しましょう。

【チャレンジ８－１】

prog8-2.py では，ターンして方向転換していました。カーブして方向転換するプログラムに変更しましょう。

８．２　条件式の使い方

if 文などで使われる条件式を表８－１にまとめました。２つの値が等しいことや異なることを条件式として使えます。また，２つの値の大小関係も判断することができます。

表８－１　条件の書き方

記号	意　　味	例	例　の　意　味
==	等しい	a == b	a と b が等しいか？
!=	異なる	a != b	a と b が異なるか？
<	小さい	a < b	a が b より小さいか？
<=	小さいか等しい	a <= b	a が b より小さいか，または，等しいか？
>	大きい	a > b	a が b より大きいか？
>=	大きいか等しい	a >= b	a が b より大きいか，または，等しいか？

また，条件式Ａと条件式Ｂの関係を組み合わせて，１つの条件式にまとめることもできます。

条件式Ａと条件式Ｂの両方ともが正しい（Y，True，成り立つ）ときだけ，正しい（Y，True，成り立つ）と判断するための条件式は，

「条件式Ａ and 条件式Ｂ」

と書きます。読み方は「条件Ａ　かつ　条件Ｂ」といいます。

例えば，変数 a の値が 5 から 10 の間にあるかどうか判断する条件式は，図８－２のように

「条件式Ａ　5 <= a」（a は 5 以上か？）

「条件式Ｂ　a <= 10」（a は 10 以下か？）

を組み合わせて，

「5 <= a and a <= 10」

とします。なお，Python では「5 <= a <= 10」と書くこともできます。

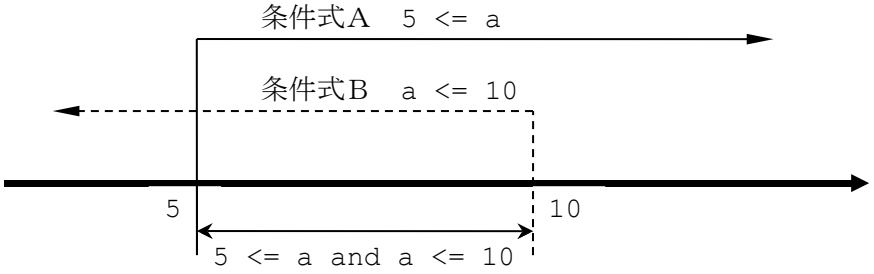


図８－２　数直線を使った条件式「5 <= a and a <= 10」の考え方

条件式A，または，条件式Bが正しい（Y，True，成り立つ）ときに，正しい（Y，True，成り立つ）とする判断するための条件式は，

「条件式A or 条件式B」

と書きます。読み方は「条件式A　または　条件式B」といいます。

例えば，変数 a の値が 1 か 5 かどうかを判断する条件式は，

「条件式A　a == 1」（a は 1 と等しいか？）

「条件式B　a == 5」（a は 5 と等しいか？）

を組み合わせて，

「a == 1 or a == 5」

とします。

８．３　while文

ある条件が成り立っている間，命令を繰り返し実行するための制御文として，while 文があります。

while 文は，次のように書きます。

```
while 条件式:
    処理
次の処理
```

while 文のフローチャートは図８－３のようになり，条件が成り立つ（Y）の間，{　} に書かれた命令を繰り返し，条件が成り立たない（N）となると次の命令を実行します。

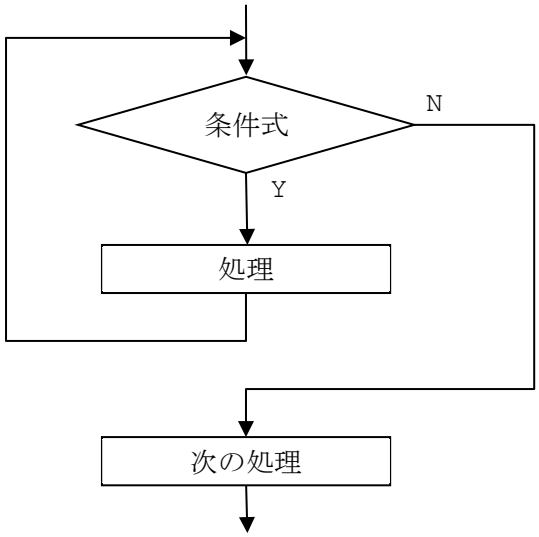


図８－３　while 文のフローチャート

無限に処理を繰り返したいときは,while 文の条件式を True とし次のように書きます。

```
while True:
    処理
```

とくに，プログラムを終了させたくないときには，プログラムの最後に

```
while True:
    pass
```

と書きます。ここで，pass 文は何も処理しないという意味ですが，pass 文を省略することはできません。

約 10 秒間様々な前進と右ターンを繰り返した後，停止するプログラムを while 文を使って作ると prog8-3.py のようになります。

【 prog8-3.py 】

```
 1 #!/usr/bin/env python3
 2 from random import uniform
 3 from ev3dev2.motor import MoveTank,OUTPUT_B,OUTPUT_C
 4 rbt=MoveTank(OUTPUT_B,OUTPUT_C)
 5 sp=20            # スピード(%)
 6 t=0              # 経過時間(秒)
 7
 8 while t < 10:
 9 ␣␣␣␣wt=uniform(0.5,1)
10 ␣␣␣␣rbt.on_for_seconds(sp,sp,wt)      # 前進
11 ␣␣␣␣t += wt
12 ␣␣␣␣rbt.on_for_seconds(sp,-sp,wt)     # 右にターン
13 ␣␣␣␣t += wt
14
15 rbt.off()
```

6 行目　t=0

経過時間を記憶しておくために変数 t を使います。最初に t を 0 に初期化しておきます。

8 行目　while t < 10:

変数 t の値が 10 以下の間，段付けした 9 行目から 13 行目までの処理を繰り返し実行します。t は経過時間を表しているので，その時間が 10 秒以下のときに前進と右にターンを繰り返すことになります。

経過時間が 10 秒を過ぎると条件式 t < 10 が成り立たたなくなり，9 行目から 13 行目の処理は行わず 14 行目に処理が移ります。

9 行目　␣␣␣␣wt=uniform(0.5,1)

前進や右にターンする時間を示す変数 wt に，uniform 関数を使って 0.5 から 1 までの間に設定します。

８．４　for 文

ある処理を繰り返し実行するための制御文として for 文があります。

for 文は，次のように書きます。

```
for 変数 in オブジェクト:
    処理
```

for 文は，オブジェクトから１つずつ値を取り出し，それを変数に入れて，処理を行います。

オブジェクトには，様々なものがありますが，すでに紹介した range 関数について説明します。例えば，range(5)とすると，そのオブジェクトは，リストと呼ばれ

[0, 1, 2, 3, 4]

となります。カギカッコで囲み，カンマで区切った要素が 1 つずつ変数に入れられて処理されます。変数名を i とすると

```
for i in range(5):
    処理
```

のとき，

変数 i の値が 0 で１回目の処理を行い，

変数 i の値が 1 で２回目の処理を行い，

変数 i の値が 2 で３回目の処理を行い，

変数 i の値が 3 で４回目の処理を行い，

変数 i の値が 4 で５回目の処理を行います。

この結果として処理を５回行うことになります。

また，range 関数を

range(a, b, c)

というように使うと，a から始まり，その値に c を加えて，b を超えない範囲までの値のリストが作成できます。

例えば，range(0, 20, 5)とすると

[0, 5, 10, 15]

というリストを作成できます。最後の「20」は含まれませんので注意してください。

次のプログラムは，EV3 ブロックの液晶ディスプレイに格子を描きます。

【 prog8-4.py 】

```
 1 #!/usr/bin/env python3
 2 from time import sleep
 3 from ev3dev2.display import Display
 4 lcd=Display()
 5 lcd.clear()
 6
 7 for x in range(0, 178, 25):
 8 ␣␣␣␣lcd.line(False,x,0,x,127,width=1)
 9
10 for y in range(0, 128, 25):
11 ␣␣␣␣lcd.line(False,0,y,177,y,width=1)
12
13 lcd.update()
14 sleep(10)
```

7 行目　for x in range(0, 178, 25):

この for 文では，変数 x の値が 0 から 25 ずつ増加していき，175 まで変化しながら，8 行目の線を描く処理を繰り返し実行します。

8 行目　␣␣␣␣lcd.line(False,x,0,x,127,width=1)

線幅 1 の垂直線を描きます。

10 行目　for y in range(0, 128, 25):

この for 文では，変数 y の値が 0 から 25 ずつ増加していき，125 まで変化しながら，11 行目の線を描く処理を繰り返し実行します。

11 行目　␣␣␣␣lcd.line(False,0,y,177,y,width=1)

線幅 1 の水平線を描きます。

８．５　break 文

while 文や for 文の途中で，繰り返しを終了し，次の処理に実行を移したいときは，break 文を使います。通常，if 文と一緒に使って，ある条件を満たしたら繰り返しの途中で中断するような処理にします。

プログラム prog8-5.py は，動作時間を無限にして，入力ポート１番に接続したタッチセンサがオンになると終了し停止するように prog8-3.py を変更したものです。

【 prog8-5.py 】

```
 1 #!/usr/bin/env python3
 2 from random import uniform
 3 from ev3dev2.sensor.lego import TouchSensor
 4 from ev3dev2.motor import MoveTank,OUTPUT_B,OUTPUT_C
 5
 6 ts=TouchSensor()
 7 rbt=MoveTank(OUTPUT_B,OUTPUT_C)
 8 sp=20            # スピード(%)
 9
10 while True:
11 ␣␣␣␣if ts.is_pressed == True: break
12 ␣␣␣␣wt=uniform(0.5,1)
13 ␣␣␣␣rbt.on_for_seconds(sp,sp,wt)      # 前進
14 ␣␣␣␣rbt.on_for_seconds(sp,-sp,wt)     # 右にターン
15
16 rbt.off()
```

11 行目　␣␣␣␣if ts.is_pressed == True: break

タッチセンサがオンの状態のとき，is_pressed の値は，「True」となり，オフのときは，「False」となります。

この if 文によって，タッチセンサがオンになると break 文を実行し，while 文の繰り返しを抜け出し，16 行目に実行が移りモータが停止します。

８．６　continue 文

while 文や for 文の途中で，繰り返し処理をスキップしたいときに continue 文を使います。通常，if 文と一緒に使って，ある条件を満たしたら繰り返し処理をスキップするような処理にします。

プログラム prog8-6.py は，動作時間を無限にして，入力ポート１番に接続したタッチセンサがオンのときだけ停止するように prog8-3.py を変更したものです。

【 prog8-6.py 】

```
 1 #!/usr/bin/env python3
 2 from random import uniform
 3 from ev3dev2.sensor.lego import TouchSensor
 4 from ev3dev2.motor import MoveTank,OUTPUT_B,OUTPUT_C
 5
 6 ts=TouchSensor()
 7 rbt=MoveTank(OUTPUT_B,OUTPUT_C)
 8 sp=20            # スピード(%)
 9
10 while True:
11 ␣␣␣␣if ts.is_pressed == True: continue
12 ␣␣␣␣wt=uniform(0.5,1)
13 ␣␣␣␣rbt.on_for_seconds(sp,sp,wt)      # 前進
14 ␣␣␣␣rbt.on_for_seconds(sp,-sp,wt)     # 右にターン
```

11 行目　␣␣␣␣if ts.is_pressed == True: continue

タッチセンサがオンの状態のとき，is_pressed の値は，「True」となり，オフのときは，「False」となります。

この if 文によって，タッチセンサがオンになると continue 文を実行し，12 行目から 14 行目をスキップします。

そのため，タッチセンサをオンにしている間，ロボットは停止した状態となり，オフになると再び動き出します。

第９章　いろいろなセンサ

EV3 ブロックの入力ポートには，第３章で説明したタッチセンサに加えて，カラーセンサ，超音波センサ，ジャイロセンサなどを接続して利用できます。この章では，いろいろなセンサの使い方を説明します。

９．１　タッチセンサ

タッチセンサ（図３－１）を使うと，何かに触れたら動きをかえるロボットを作ることができます。タッチセンサを図１－３に示した車輪移動型ロボットの前方に取り付け，入力ポート１番に専用ケーブルで接続します。接続した様子を図９－１に示します。

タッチセンサは，車輪移動型ロボットの組み立て説明図に掲載されているように取り付けてください。

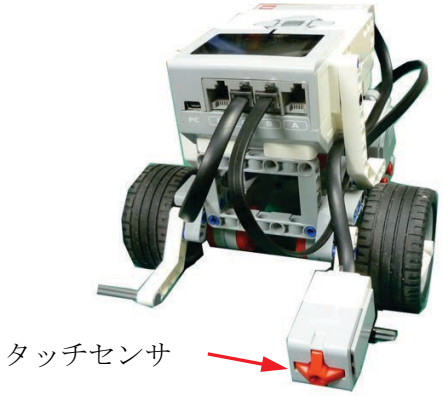
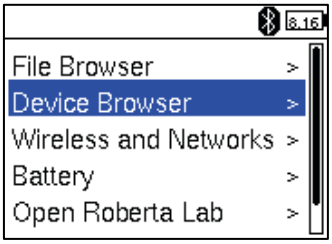


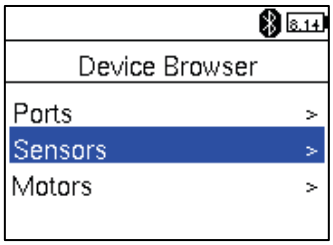
図９－１　タッチセンサを前方に取り付けた車輪移動型ロボット

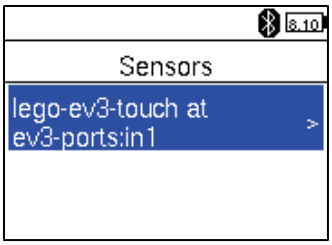
プログラムを作成する前にタッチセンサが正常の動作しているか確認しましょう。タッチセンサの接続ができたら，EV3 ブロックの基本メニュー画面を表示します。



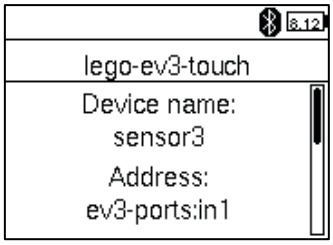
基本メニュー画面でない場合，何度か「戻るボタン」を押して「Shudown メニュー画面」を表示し「Cancel」を選択します。

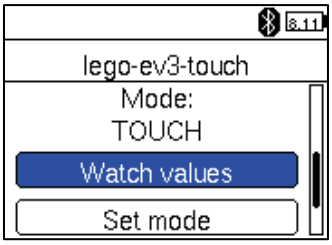
「Sensors」を選択し，「lego-ev3-touch at ev3-ports:in1」を選択します。



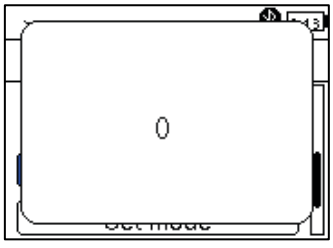


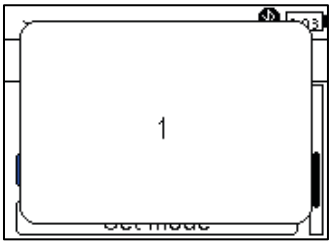
上下ボタンで画面をスクロールさせ，「Watch values」を選択します。





タッチセンサの押しボタンが押されていない状態では「0」を，押されている状態では「1」が表示されます。





元の基本メニュー状態に戻りたいときは「戻るボタン」を押していきます。

ここでは，ロボットが何かにぶつかるまで前進し続けるプログラム prog9-1.py を示します。

【 prog9-1.py 】

```
 1 #!/usr/bin/env python3
 2 from ev3dev2.sensor.lego import TouchSensor
 3 from ev3dev2.motor import MoveTank,OUTPUT_B,OUTPUT_C
 4
 5 ts=TouchSensor()
 6 rbt=MoveTank(OUTPUT_B,OUTPUT_C)
 7 sp=30                              # スピード(%)
 8
 9 rbt.on(sp,sp)                     # 前進開始
10 while True:
11 ␣␣␣␣if ts.is_pressed == True: break
12
13 rbt.off()
```

2 行目　from ev3dev2.sensor.lego import TouchSensor

EV3 ブロックで利用できるセンサに対して処理するためモジュール ev3dev2.sensor.lego の中から，タッチセンサに対する TouchSensor という設計図（クラスと呼びます。）を読み込みます。

5 行目　ts=TouchSensor()

タッチセンサを処理するためクラスである TouchSensor の実体（インスタンス）として ts を作ります。

9 行目 rbt.on(sp,sp)

左右のモータのスピードを変数 SP に代入している値(30)で正回転させ，ロボットを前進させます。

10 行目　while True:

11 行目 ␣␣␣␣if ts.is_pressed == True: break

while 文の条件式として True を書いていますので，段付けされた 11 行目を繰り返し無限に実行します。

11 行目の if 文で，条件式「ts.is_pressed == True」が成り立つと，break 文によって，while 文による繰り返し処理を抜け出し 12 行目に実行が移ります。

タッチセンサに何も触れていないとき，ts.is_pressed は，True になり，何も触れら

れていないとき，False となります。そのため，条件式「ts.is_pressed == True」は，「タッチセンサに何か触れているか？」という意味になります。

すでに 9 行目の rbt.on(sp,sp)が実行されロボットは前進を開始し，その状態が保たれていますから，while 文による繰り返し処理の間も，ロボットは前進し続けます。

14 行目 rbt.off()

12 行目の break 文によって 13 行目に実行が移り，モータは停止します。

prog9-1.py では，壁などの障害物が前方の取り付けられたタッチセンサにぶつかるとその場で止まるプログラムでした。次に，壁などの障害物に当たると後退して止まるというプログラムを作るにはどうすればよいでしょうか？

prog9-1.py の 14 行目の「rbt.off()」の前に，1 秒間後退する処理を追加したプログラムは，prog9-2.py のようになります。

【 prog9-2.py 】

```
 1 #!/usr/bin/env python3
 2 from ev3dev2.sensor.lego import TouchSensor
 3 from ev3dev2.motor import MoveTank,OUTPUT_B,OUTPUT_C
 4
 5 ts=TouchSensor()
 6 rbt=MoveTank(OUTPUT_B,OUTPUT_C)
 7 sp=30                              # スピード(%)
 8
 9 rbt.on(sp,sp)                     # 前進開始
10 while True:
11 ␣␣␣␣if ts.is_pressed == True: break
12
13 rbt.on_for_seconds(-sp,-sp,1)    # 1 秒間後退
14 rbt.off()
```

13 行目 rbt.on_for_seconds(-sp,-sp,1)

prog9-1.py に，タッチセンサに何か触れてロボットが停止する前に，1 秒間後退する処理を追加しています。

【チャレンジ９－１】

図９－２のフローチャートとプログラム prog9-2.py を参考にして，while 文と if 文を組み合わせて，障害物を避けて前進するプログラムを作ってみましょう。

また，カーブやターンの違いをうまく使って，スムーズに障害物を避けられるように工夫しましょう。

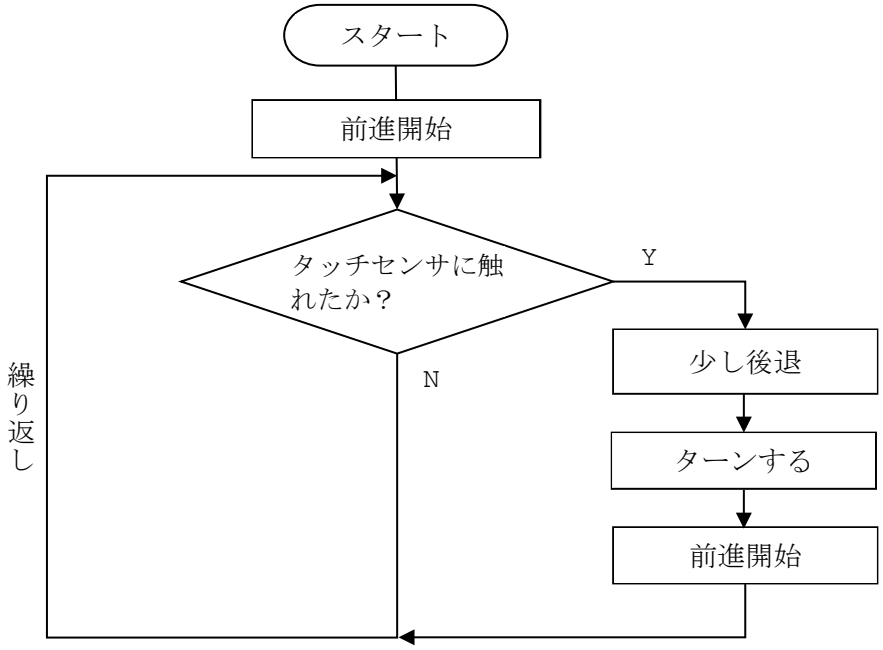


図９－２　障害物を避けて前進する動きのフローチャート

９．２　超音波センサ

タッチセンサは，何かに接触しないと状況を知ることができませんでした。

コウモリのように超音波を使って接触しなくても，物体までの距離を測る超音波センサを使ってみましょう。

ここで使う超音波センサは，図９－３に示すように「超音波を送信するスピーカ」と「超音波を受信するマイク」で構成されています。EV3 ブロックで利用できる超音波センサには，スピーカとマイクが並んで配置されています。ここで使っている超音波は，人間の耳では聞こえない高い周波数の音です。「スピーカ」から発せられた超音波が，物体に当たり反射し，元に戻ってきた超音波を「マイク」で受信するまでの時間差を測り，その時間差から往復距離を求めます（図９－４）。

図９－３　超音波センサ

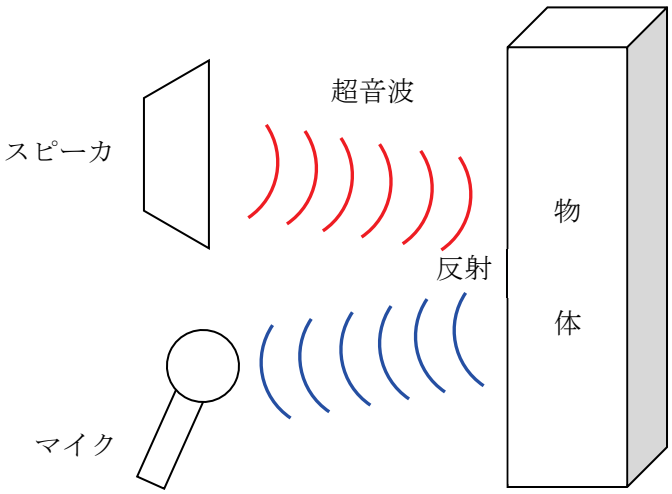


図９－４　超音波を使った距離計測

それでは，超音波センサを使って，何かに近づいたら逃げるロボットを作りましょう。車輪移動型ロボットの組み立て説明図を参考にして，超音波センサを図１－３に示した車輪移動型ロボットの前方に取り付け，入力ポート４番に専用ケーブルで接続します。接続した様子を図９－５に示します。

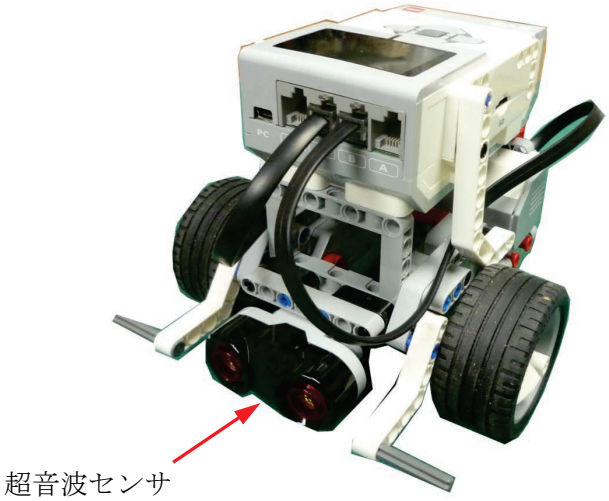
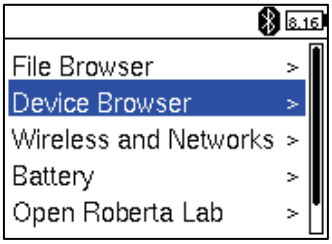


図９－５　超音波センサを取り付けた車輪移動型ロボット

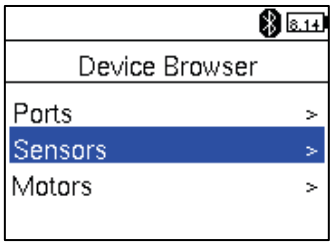
プログラムを作成する前に超音波センサが正常に動作しているか確認しましょう。

超音波センサの接続ができたら，EV3 ブロックの基本メニュー画面を表示します。

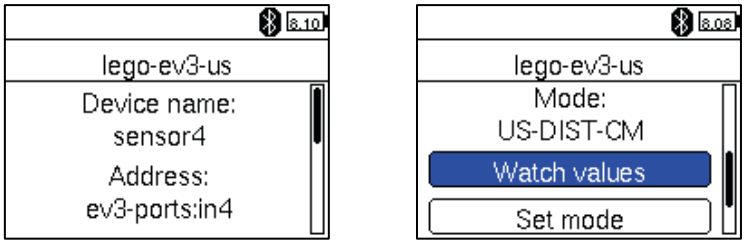


基本メニュー画面でない場合，何度か「戻るボタン」を押して「Shudown メニュー画面」を表示し「Cancel」を選択します。

「Sensors」を選択し，「lego-ev3-us at ev3-ports:in4」を選択します。

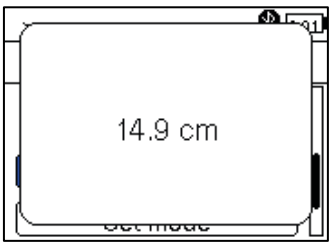


上下ボタンで画面をスクロールさせ，「Watch values」を選択します。



超音波センサから対象物体までの距離が cm の単位で表示されます。超音波センサのスピーカとマイクに付いている赤色の発光ダイオードも光ります。

この超音波センサで計測できる距離は 3cm から 250cm で，その精度は約 1cm とされています。



元の基本メニュー状態に戻りたいときは「戻るボタン」を押していきます。

それでは，超音波センサで測った距離を液晶ディスプレイに表示するプログラム prog9-3.py を入力して実行してみましょう。

【 prog9-3.py 】

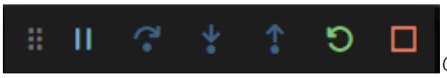
```
 1 #!/usr/bin/env python3
 2 from time import sleep
 3 import os
 4 os.system('setfont Lat15-VGA16')
 5
 6 from ev3dev2.sensor.lego import UltrasonicSensor
 7
 8 us=UltrasonicSensor()
 9 while True:
10 ␣␣␣␣print(us.distance_centimeters)
11 ␣␣␣␣sleep(1)
```

EV3 ブロックの液晶ディスプレイの上から，1 秒ごとに物体までの距離が表示されていきます。物体までの距離が変わると表示される数値も変わります。

```
157.20000000000002
255.0
255.0
37.2
19.1
19.1
255.0
```

このプログラムは無限に繰り返す処理になっています。

終了させるときは，VS Code の右上に表示されている



の中から，  を左クリックします。

また，EV3 ブロック本体の「戻るボタン」を押しても，プログラムを終了させることができます。

それでは，プログラムについて説明しましょう。

2 行目　from time import sleep

時間に対して処理するための様々なプログラムを集めたモジュール time の中から，指定した時間だけ待つ機能をもつ sleep というプログラムを読み込みます。

3 行目　import os

4 行目　os.system('setfont Lat15-VGA16')

ev3dev というオペレーティングシステム(OS)で様々な機能を利用するためのモジュールである os を読み込んでいます。

さらに，os モジュールの中にある system メソッドを使って，液晶ディスプレイを「コンソール」と呼ばれる使い方をする際に利用する書体（文字の形と大きさ）を設定しています。この設定はプログラムを終了しても継続しています。

6 行目　from ev3dev2.sensor.lego import UltrasonicSensor

EV3 ブロックで利用できるセンサに対して処理するためモジュール ev3dev2.sensor.lego の中から，超音波センサに対する UltrasonicSensor という設計図（クラスと呼びます。）を読み込みます。超音波のことを英語で「Ultrasonic」といいます。

8 行目　us=UltrasonicSensor()

超音波センサを処理するためクラスである UltrasonicSensor の実体(インスタンス)として us を作ります。

7 行目　while True:

while 文の条件式として True を書いていますので，段付けされた 8 行目と 9 行目を繰り返し無限に実行します。

9 行目　␣␣␣␣print(us.distance_centimeters)

print 関数は，丸カッコ( )の中に書いた値を，液晶ディプレイをコンソールとして使って表示します。値を表示した後，改行します。

超音波センサで測った距離(cm)は,us.distance_centimeters に入っていますので,液晶ディプレイに，その距離が次々と表示されていきます。

10 行目　sleep(1)

sleep 関数を使って 1 秒間待ちます。

超音波センサの使い方が分かったところで，ロボットが壁などの手前 35cm まで前進し続けるプログラム prog9-4.py を示します。

【 prog9-4.py 】

```
 1 #!/usr/bin/env python3
 2 from ev3dev2.sensor.lego import UltrasonicSensor
 3 from ev3dev2.motor import MoveTank,OUTPUT_B,OUTPUT_C
 4
 5 us=UltrasonicSensor()
 6 rbt=MoveTank(OUTPUT_B,OUTPUT_C)
 7 sp=30                              # スピード(%)
 8
 9 rbt.on(sp,sp)                     # 前進開始
10 while True:
11 ␣␣␣␣if us.distance_centimeters < 35: break
12
13 rbt.off()
```

9 行目 rbt.on(sp,sp)

左右のモータのスピードを変数 SP に代入している値(30)で正回転させ，ロボットを前進させます。

10 行目 while True:

11 行目 ␣␣␣␣if us.distance_centimeters < 35: break

while 文の条件式として True を書いていますので，段付けされた 11 行目を繰り返し無限に実行します。

11 行目の if 文で，条件式「us.distance_centimeters < 35」が成り立つと，break 文によって，while 文による繰り返し処理を抜け出し 12 行目に実行が移ります。

us.distance_centimeters には，超音波センサで測った距離が cm の単位で記録されています。その距離が 35 より小さいとき，この条件式は「成り立つこと」となり，break 文が実行され，12 行目に処理が移ります。この条件式が「成り立っていない」とき，再び 11 行目の if 文を実行します。

すでに 9 行目の rbt.on(sp,sp)が実行されロボットは前進を開始し，その状態が保たれていますから，while 文による繰り返し処理の間も，ロボットは前進し続けます。

13 行目 rbt.off()

11 行目の break 文によって 12 行目に実行が移り，モータは停止します。

【チャレンジ９－２】

図９－６のフローチャートとプログラム prog9-4.py を参考にして，while 文と if 文を組み合わせて，障害物を避けて前進するプログラムを作ってみましょう。

また，カーブやターンの違いをうまく使って，スムーズに障害物を避けられるように工夫しましょう。

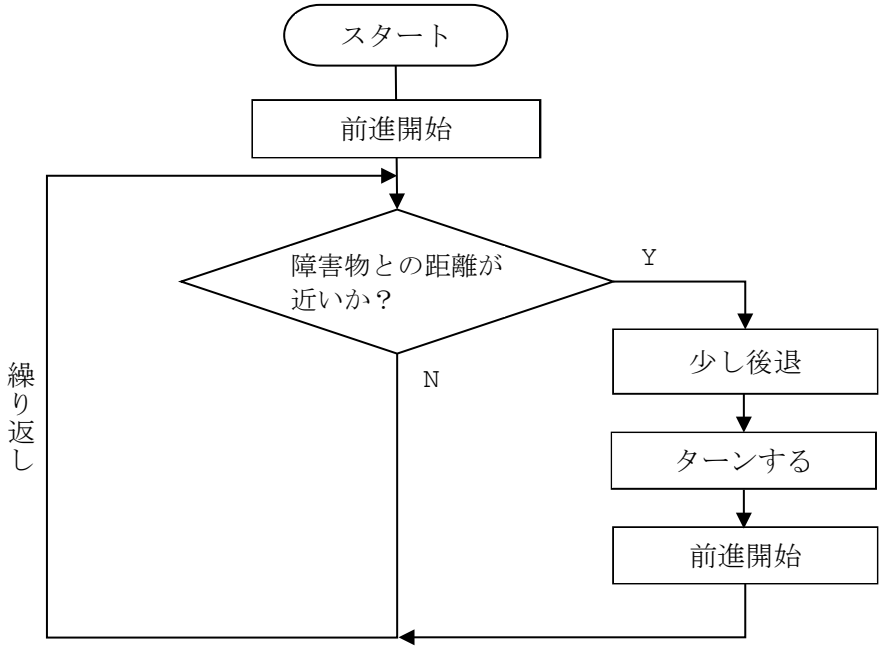


図９－６　障害物を避けて前進する動きのフローチャート

prog9-4.py では，超音波センサと壁などの障害物までの距離が 35cm 以下になるとロボットが停止するプログラムでした。次に，人や壁などの障害物にある程度近づくと後退して逃げるというプログラムを作るにはどうすればよいでしょうか？

このような動きをするロボットのプログラムは，prog9-4.py の 11 行目に書いている if 文を「もし，障害物までの距離が 35cm より大きいとき，前進し，さもなければ，後退する。」のように変更しなければなりません。変更したプログラムはつぎのようになります。

【 prog9-5.py 】

```
 1 #!/usr/bin/env python3
 2 from ev3dev2.sensor.lego import UltrasonicSensor
 3 from ev3dev2.motor import MoveTank,OUTPUT_B,OUTPUT_C
 4
 5 us=UltrasonicSensor()
 6 rbt=MoveTank(OUTPUT_B,OUTPUT_C)
 7 sp=30                              # スピード(%)
 8
 9 while True:
10 ␣␣␣␣if us.distance_centimeters > 35:
11 ␣␣␣␣␣␣␣␣rbt.on(sp,sp)              # 前進開始
12 ␣␣␣␣else:
13 ␣␣␣␣␣␣␣␣rbt.on(-sp,-sp)           # 後退開始
```

```
 9 行目  while True:
10 行目  ␣␣␣if us.distance_centimeters > 35:
11 行目  ␣␣␣␣␣␣␣␣rbt.on(sp,sp)              # 前進開始
12 行目  ␣␣␣␣else:
13 行目  ␣␣␣␣␣␣␣␣rbt.on(-sp,-sp)           # 後退開始
```

while 文の条件式として True を書いていますので，段付けされた 10 行目から 13 行目までを繰り返し無限に実行します。

10 行目の if 文で，条件式「us.distance_centimeters > 35」が成り立つと，二重に段付けされた 11 行目を実行し，前進を開始します。さもなければ，12 行目の else 文のつぎから二重に段付けされた 13 行目を実行し，後退を開始します。

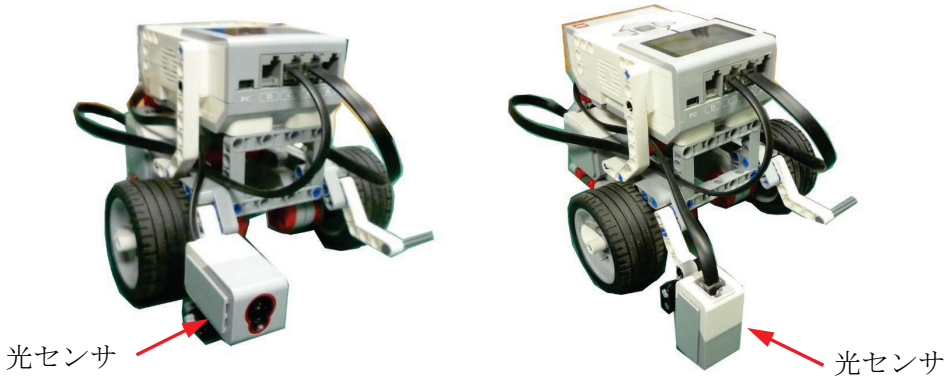
９．３　光センサ

ロボットに眼をつけて，外の光に反応したり，線の上をたどっていくロボットを作ってみましょう。眼のように明るさや色を検知するためのセンサを光センサと呼びます。

ここで使う光センサは，図９－７に示すように光を発する「発光素子」と，明るさを検知する「受光素子」で構成されています。NXT ブロック用光センサは，発光素子と受光素子が並んで配置されています。発光素子を利用しているときは，明るく光ります。受光素子にその光が当たると，その強さに応じて電流が流れ，明るさを検知できます。

図９－７　光センサ

光センサを使用して，周囲の光や色によって動きをかえるロボットを作りましょう。車輪移動型ロボットの組み立て説明図を参考にして，光センサを図１－３の車輪移動型ロボットの前方に取り付け，入力ポート２番に専用ケーブルで接続します。接続した様子を図９－８に示します。



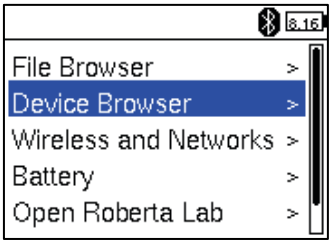
（ａ）水平に取り付けた例　　（ｂ）下向きに取り付けた例

図９－８　光センサを取り付けた車輪移動型ロボット

光センサの接続ができたら，プログラムを作成する前に光センサが，次の２通りの方法で正常に動作しているか確認しましょう。

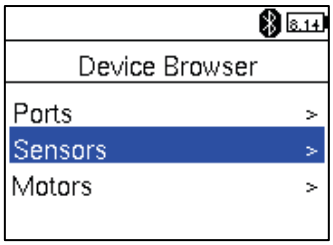
（1）周囲の明るさを検知できるかを確認します。

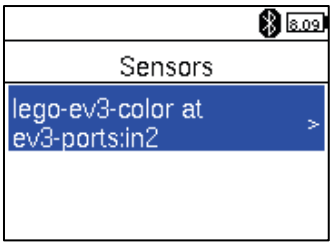
EV3 ブロックの基本メニュー画面を表示します。



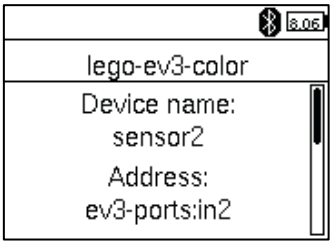
基本メニュー画面でない場合，何度か「戻るボタン」を押して「Shudown メニュー画面」を表示し「Cancel」を選択します。

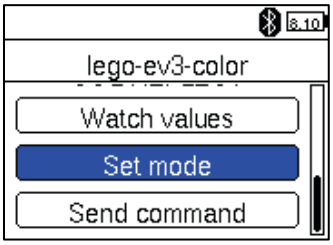
「Sensors」を選択し，「lego-ev3-color at ev3-ports:in2」を選択します。



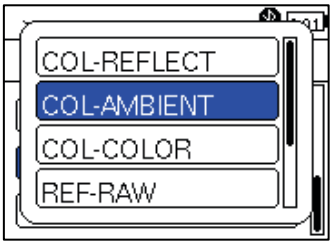


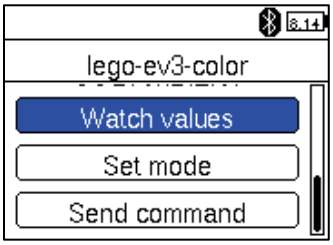
上下ボタンで画面をスクロールさせ，「Set mode」を選択します。



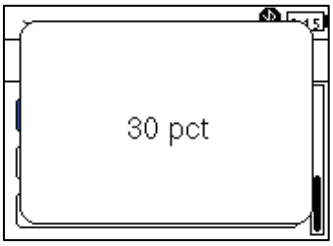


「COL-AMBIENT」を選択し，「Watch values」を選択します。





光センサで計測された明るさが数値(%)で表示されます。「pct」はパーセントを省略して表示しています。



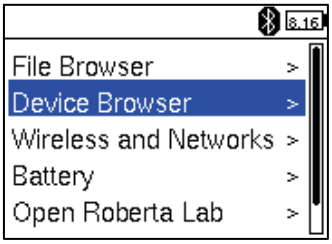
この光センサが計測している状態に応じて「0%（最も暗いとき)」から「100%（最も明るいとき)」で表示されます。

この状態では,「発光素子」が暗く青色に光ります。

元の基本メニュー状態に戻りたいときは「戻るボタン」を押していきます。

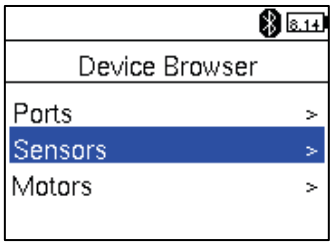
（2）光を発行し，その反射の明るさを検知できるかを確認します。

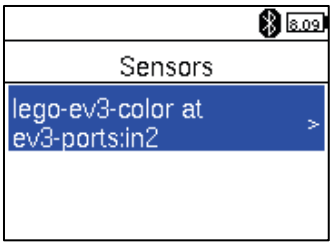
EV3 ブロックの基本メニュー画面を表示します。



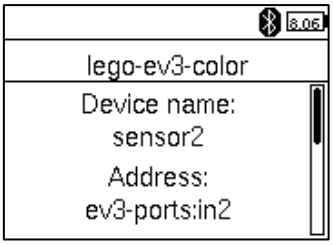
基本メニュー画面でない場合，何度か「戻るボタン」を押して「Shudown メニュー画面」を表示し「Cancel」を選択します。

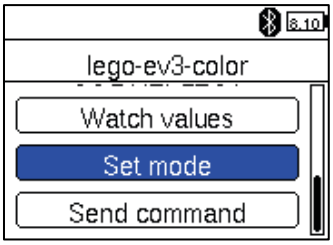
「Sensors」を選択し，「lego-ev3-color at ev3-ports:in2」を選択します。



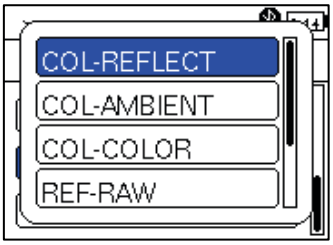


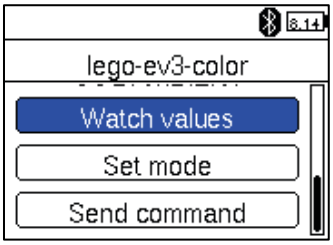
上下ボタンで画面をスクロールさせ，「Set mode」を選択します。



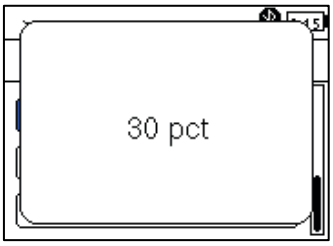


「COL-REFLECT」を選択し，「Watch values」を選択します。





光センサで計測された明るさが数値(%)で表示されます。「pct」はパーセントを省略して表示しています。



この光センサが計測している状態に応じて，対象物から反射してきた光の強さが「0%（黒色などの対象物で最も暗いとき）」から「100%（白色や銀色などの対象物で最も明るいとき）」で表示されます。

この状態では，「発光素子」が明るく赤色に光り，この赤色の反射の強さを計測しています。

元の基本メニュー状態に戻りたいときは「戻るボタン」を押していきます。

９．４　周辺の明るさを検知

周囲の明るさによって動きを変えるロボットを作ってみましょう。図９－９のように光センサは，受光素子に当たる光の強さを検知します。

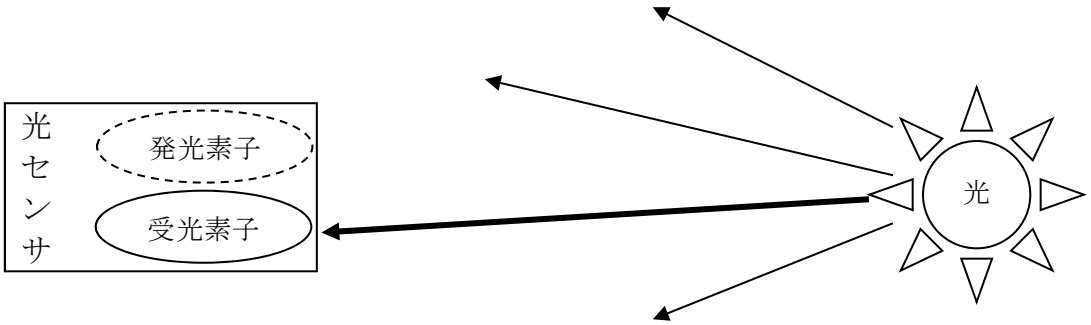


図９－９　受光素子に当たる光の強さを検知

この機能を利用して，図９－８（ａ）に示した水平に光センサを取り付けた車輪移動型ロボットを使って，虫のように明るいところに寄ってくるようなロボットを作ってみましょう。例えば，点灯した懐中電灯をロボットに向けると前進するような動きをします。

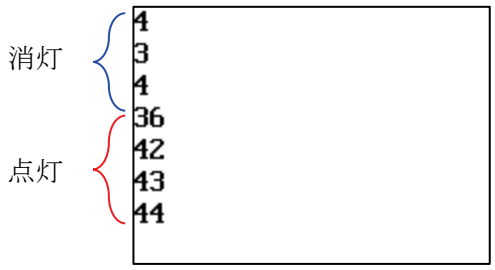
まず，光センサで測った明るさを液晶ディスプレイに表示するプログラム prog9-6.py を入力して実行してみましょう。

【 prog9-6.py 】

```
 1 #!/usr/bin/env python3
 2 from time import sleep
 3 import os
 4 os.system('setfont Lat15-VGA16')
 5
 6 from ev3dev2.sensor.lego import ColorSensor
 7
 8 cs=ColorSensor()
 9 while True:
10 ____print(cs.ambient_light_intensity)
11 ____sleep(1)
```

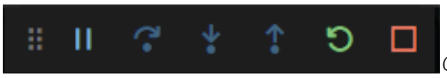
EV3 ブロックの液晶ディスプレイの上から,1 秒ごとに光センサで計測された明るさが%で表示されます。明るさが変わると表示される数値も変わります。

次に示す表示例では，懐中電灯を消灯しているときは，4 程度の明るさの値（%）で，点灯すると 40 程度の明るさの値（%）になっています。



このプログラムは無限に繰り返す処理になっています。

終了させるときは，VS Code の右上に表示されている

 の中から，  を左クリックします。

また，EV3 ブロック本体の「戻るボタン」を押しても，プログラムを終了させることができます。

それでは，プログラムについて説明しましょう。

2 行目　`from time import sleep`

時間に対して処理するための様々なプログラムを集めたモジュール time の中から，指定した時間だけ待つ機能をもつ sleep というプログラムを読み込みます。

3 行目　`import os`

4 行目　`os.system('setfont Lat15-VGA16')`

ev3dev というオペレーティングシステム(OS)で様々な機能を利用するためのモジュールである os を読み込んでいます。

さらに，os モジュールの中にある system メソッドを使って，液晶ディスプレイを「コンソール」と呼ばれる使い方をする際に利用する書体（文字の形と大きさ）を設定しています。この設定はプログラムを終了しても継続しています。

6 行目　`from ev3dev2.sensor.lego import ColorSensor`

EV3 ブロックで利用できるセンサに対して処理するためモジュール ev3dev2.sensor.lego の中から，光センサに対する ColorSensor という設計図（クラスと呼びます。）を読み込みます。

8 行目　cs=ColorSensor()

光センサを処理するためクラスである ColorSensor の実体（インスタンス）として cs を作ります。

7 行目　while True:

while 文の条件式として True を書いていますので，段付けされた 8 行目と 9 行目を繰り返し無限に実行します。

9 行目　␣␣␣␣print(cs.ambient_light_intensity)

print 関数は，丸カッコ( )の中に書いた値を，液晶ディプレイをコンソールとして使って表示します。値を表示した後，改行します。

光センサで測った明るさ(%)は，cs.ambient_light_intensity に入っていますので，液晶ディプレイに，明るさの値が次々と表示されていきます。

10 行目　sleep(1)

sleep 関数を使って 1 秒間待ちます。

表示例のように，懐中電灯を消灯しているときは，明るさ（%）の値が 4 程度で，点灯しているとき 40 程度の値の場合，懐中電灯の光があたっているかどうかを判定するための境目となる明るさは，平均とします。この例では，(4 + 40)÷2 = 22 となります。このように境目となる値のことを「しきい値」と呼びます。

prog9-7.py は，光センサで計測された明るさが 22%より大きくなる（明るくなる）と前進し，そうでなければ後退するように作られています。

【 prog9-7.py 】

```
 1 #!/usr/bin/env python3
 2 from ev3dev2.sensor.lego import ColorSensor
 3 from ev3dev2.motor import MoveTank,OUTPUT_B,OUTPUT_C
 4
 5 cs=ColorSensor()
 6 rbt=MoveTank(OUTPUT_B,OUTPUT_C)
 7 sp=30                              # スピード(%)
 8
 9 while True:
10 ␣␣␣␣if cs.ambient_light_intensity > 22:
11 ␣␣␣␣␣␣␣␣rbt.on(sp,sp)             # 前進開始
12 ␣␣␣␣else:
13 ␣␣␣␣␣␣␣␣rbt.on(-sp,-sp)           # 後退開始
```

9 行目　while True:
10 行目　␣␣␣if cs.ambient_light_intensity > 22:
11 行目　␣␣␣␣␣␣␣␣rbt.on(sp,sp)　　# 前進開始
12 行目　␣␣␣␣else:
13 行目　␣␣␣␣␣␣␣␣rbt.on(-sp,-sp)　　# 後退開始

while 文の条件式として True を書いていますので，段付けされた 10 行目から 13 行目までを繰り返し無限に実行します。

10 行目の if 文に書いている条件式「cs.ambient_light_intensity > 22」の，cs.ambient_light_intensity は，光センサで計測した明るさを%で表した値が入っています。計測できる最も暗いときは 0，最も明るいときは 100 になります。

条件式「cs.ambient_light_intensity > 22」は，明るさが 22%より大きいとき（懐中電灯の光があたっている状態）成り立ちます。

if 文の条件式が成り立つとき，二重に段付けされた 11 行目を実行し，前進を開始します。さもなければ，懐中電灯の光はあたっていない状態となり，12 行目の else 文のつぎから二重に段付けされた 13 行目を実行し，後退を開始します。

【チャレンジ９－３】

ロボットの光センサに懐中電灯の光をあてているときは，ランダムに左または右にターンし，さもなければ停止しつづけるプログラムを作ってみましょう。

９．５　同じセンサを複数使うとき

EV3 本体に，同じセンサを複数接続して使いたいときは，それぞれのセンサからの入力値を別々に取り扱う必要があります。

例えば，２個のタッチセンサを入力ポート１番と２番に接続した場合の使い方について説明します。

```
・・・・・
from ev3dev2.sensor import INPUT_1,INPUT_2
from ev3dev2.sensor.lego import TouchSensor
ts1=TouchSensor(INPUT_1)
ts2=TouchSensor(INPUT_2)
・・・・・
```

これまでは，タッチセンサを処理するためクラスである TouchSensor の実体（インスタンス）について作るとき，入力ポートについて何も指定していませんでした。そのため，

`from ev3dev2.sensor import INPUT_1,INPUT_2`

というプログラムを書いて，EV3 ブロックで使う入力ポートの準備をする必要があります。

ev3dev2.sensor は，EV3 ブロックの入力ポートに対して処理するための様々なプログラムを集めたモジュールです。このモジュールの中から，入力ポート１番に対する INPUT_1 という設計図（クラスと呼びます。）と入力ポート２番に対する INPUT_2 という設計図（クラスと呼びます。）を読み込みます。

なお，入力ポートとして３番を使うときは，INPUT_3 と書きます。同様に入力ポート４４番を使うときは，INPUT_4 と書きます。

つぎに，タッチセンサに対する処理を行うためのプログラムを準備するために

`from ev3dev2.sensor.lego import TouchSensor`

と書きます。ev3dev2.sensor.lego は，EV3 ブロックで利用できるセンサに対して処理するための様々なプログラムを集めたモジュールです。このモジュールの中から，タッチセンサに対する TouchSensor という設計図（クラスと呼びます。）を読み込みます。

さらに，タッチセンサを処理するためクラスである TouchSensor の実体（インスタンス）を入力ポート１番に接続されたセンサとして ts1 を作るため

`ts1=TouchSensor(INPUT_1)`

と書きます。ここで，INPUT_1 は入力ポート１番を示しています。

同様に，タッチセンサを処理するためクラスである TouchSensor の実体（インスタンス）を入力ポート２番に接続されたセンサとして ts2 を作るため

```
ts1=TouchSensor(INPUT_2)
```
と書きます。ここで，INPUT_2 は入力ポート２番を示しています。

ts1 と ts2 を利用することで，それぞれのタッチセンサの状態を個別に読み取ることができるようになります。

なお，タッチセンサ以外の超音波センサや光センサについても同様なプログラムで利用できます。

第１０章　ライントレース・ロボット

ラインをたどっていくロボットを作ってみましょう。このような動きをするロボットのことを「ライントレース・ロボット」といいます。この章では,「ライントレース・ロボット」の作り方を説明します。

１０．１　反射した光を検出

図１０－１に示すように小判状の黒いラインを印刷したテスト用紙を使って，ラインをたどって動くロボットを作ってみましょう。

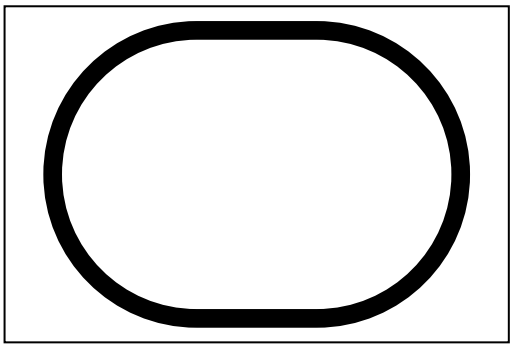


図１０－１　小判状の黒いラインを印刷したテスト用紙

テスト用紙に印刷された黒いラインを検知するために光センサを下向きに取り付けます（図９－８（ｂ））。

光センサは発光素子と受光素子を使って，テスト用紙の黒いラインと白い部分を識別します。発光素子が出した光がテスト用紙に当たり，反射した光の強さを受光素子で検知し，その変化からテスト用紙の色を識別します。

図１０－２を見てみましょう。この図の➔は，発光素子の発した光を示し，⇢は，テスト用紙から反射した光を示しています。図１０－２（ａ）に示すように，黒いラインの上では黒色の性質から，発光素子の光が吸収されてしまい反射する光の量が少なくなります。そのため，受光素子が感知する光は弱くなり，光センサの値は小さくなります。一方，図１０－２（ｂ）に示すように，白い部分の上では白色の性質から，発光素子の光は強く反射されます。その結果，黒いラインの場合と比較して，受光素子が感知する光は強くなり，光センサの値は大きくなります。

この性質を利用して，光センサを使って，黒いラインの上か白い部分の上かを判断します。

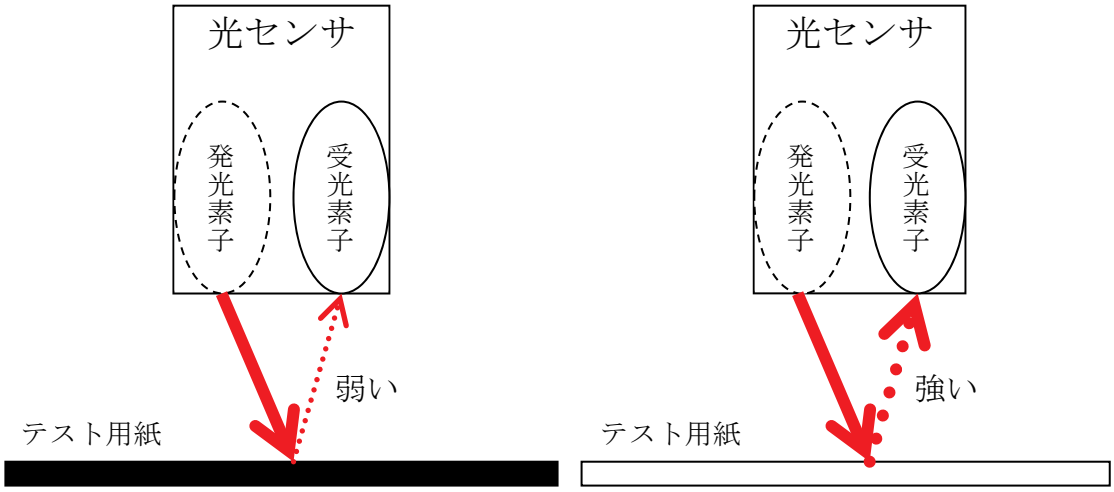


図１０－２　反射する光の強さの変化

１０．２　ラインをたどる方法

図１０－１のテスト用紙に描かれた黒いラインに沿って動くライントレース・ロボットのプログラムを考えましょう。ライントレース・ロボットは，ロボットがラインの上にあることをいつも検知しながら，ラインからはずれると移動する方向を修正しながら，ラインの片側をなぞるように動きます。

ここでは，テスト用紙のラインに沿って時計回りにロボットが移動していくプログラムを作ります（図１０－３）。始めにロボットはライン上にあるとします。ロボットが前進していくとラインから離れ，はずれてしまうことがあります。そのときは，ラインに戻すためにロボットの方向を変えて，ラインのあるところに移動させなければなりません。

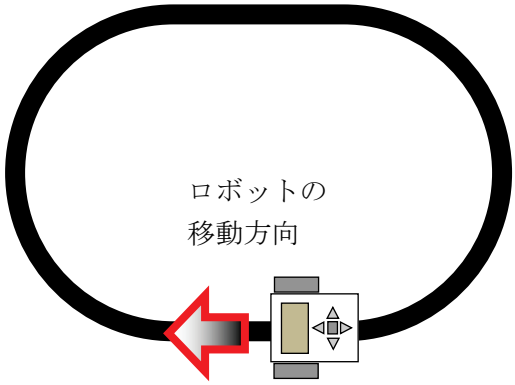


図１０－３　時計回りにライントレース

ラインをたどっていく方法のひとつとして，「ラインの左側をなぞっていく」考え方があります。最初に，ロボットは，ライン上にあるとします。図１０－４のように，ロボットがライン上にあるときは，ラインの左側をなぞるために「左にカーブ」させます。その後，ロボットが白い部分にきたら逆に「右にカーブ」させます。この動作を繰り返すことで，図１０－４のようにロボットは，ラインの左側をジグザグになぞっていくように移動し，結局ラインをたどることになります。

このようにしてライントレースするロボットの動きをフローチャートで表すと図１０－５のようになります。

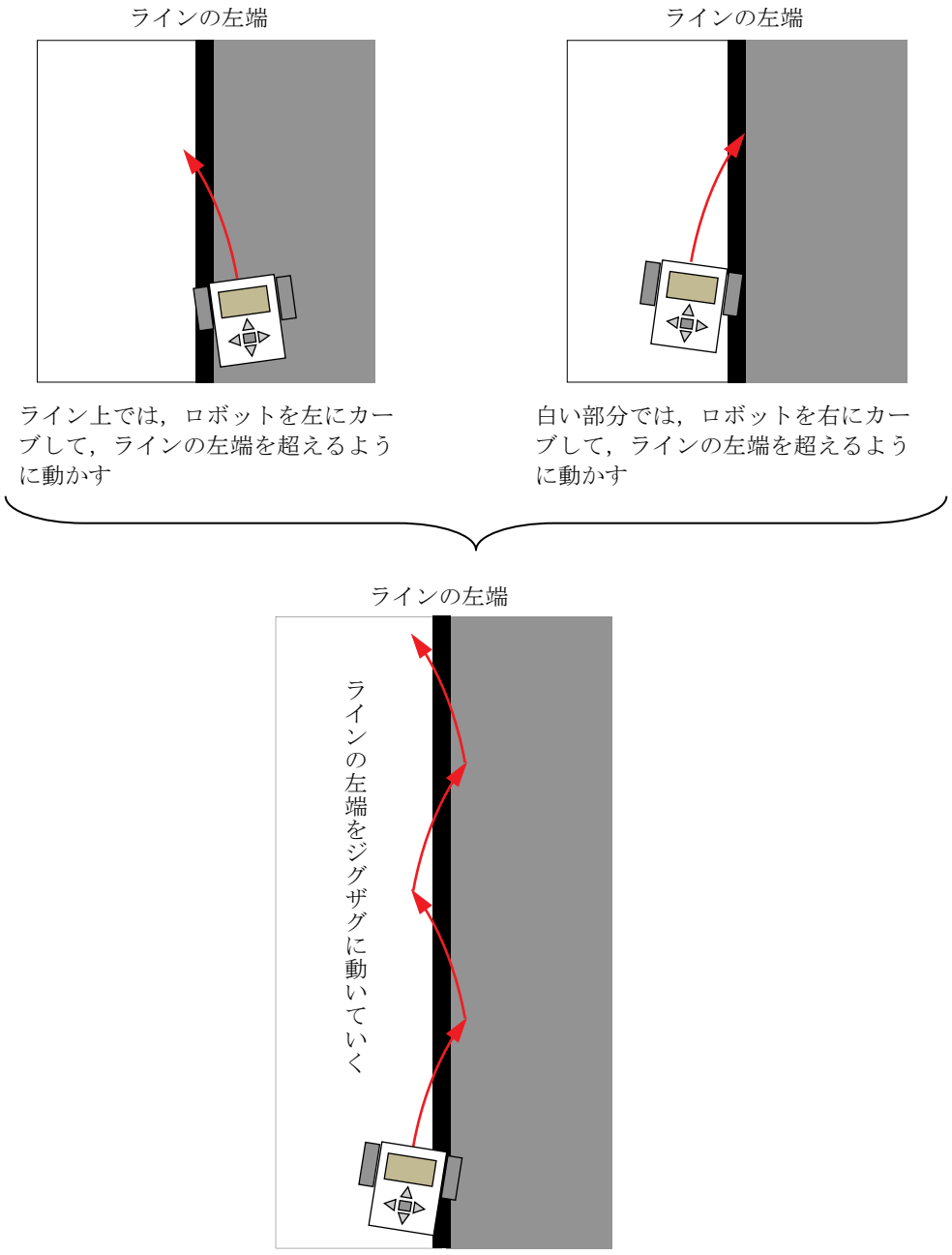


図１０－４　ラインの左端をなぞっていくライントレースの考え方

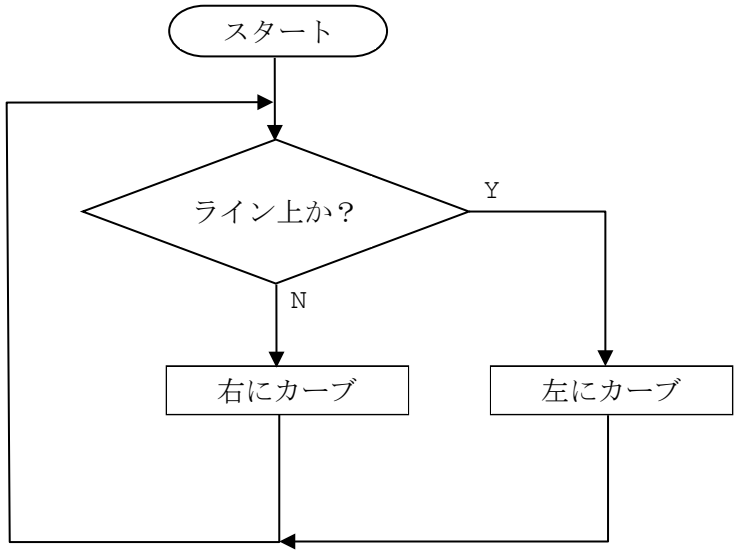


図１０－５　ラインの左端をなぞるライントレースのフローチャート

ロボットがラインの上にあるのか，それとも，ラインからはずれているのか判別するために光センサを使います。図１０－２を使って説明したように，ラインの上にあるときは，ラインが黒いことから反射した光が弱くなり光センサの値が小さくなります。逆に，ラインからはずれているときは，テスト用紙の色が白いことから反射した光が強くなり光センサの値が大きくなります。

光センサの値によって，ライン上にあるかどうか判断することができます。しかし，それを判断するための境目の値はどのように決めればよいのでしょうか？ 部屋の明るさやテスト用紙への照明方法，人の影などによって光センサの値は，かなり変化します。そこで，ライン上と白い部分での光センサの値を実際に実験して決めます。

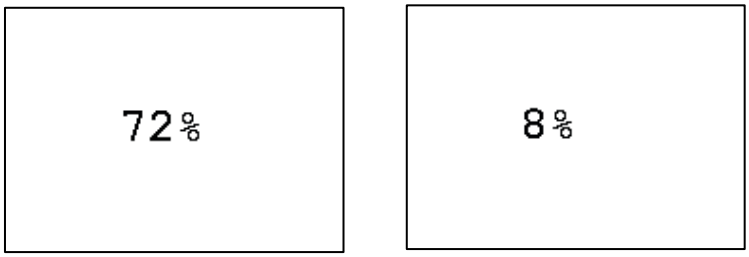
まず, 光センサで測った明るさを液晶ディスプレイに表示するプログラム prog10-1.py を入力して実行してみましょう。

【 prog10-1.py 】

```
 1 #!/usr/bin/env python3
 2 from time import sleep
 3 from ev3dev2.display import Display
 4 from ev3dev2.sensor.lego import ColorSensor
 5
 6 lcd=Display()
 7 cs=ColorSensor()
 8 while True:
 9 ␣␣␣␣x=cs.reflected_light_intensity
10 ␣␣␣␣lcd.text_pixels(str(x)+'%', True, 50, 40, font='courB24')
11 ␣␣␣␣sleep(0.5)
```

EV3 ブロックの液晶ディスプレイの上から，0.5 秒ごとに光センサで計測された明るさが%で表示されます。明るさが変わると表示される数値も変わります。

次に示す表示例では，白い部分に光センサがあるときは，70 程度の明るさの値（%）で，ライン上に光センサがあると 8 程度の明るさの値（%）になっています。



白い部分　　ライン上

光センサで計測した反射光の強さの値を液晶ディスプレイの文字を表示する部分について説明します。

9 行目　␣␣␣␣x=cs.reflected_light_intensity

cs.reflected_light_intensity は，光センサで反射光を計測した明るさを%で表した値が入っています。計測できる最も暗いときは 0，最も明るいときは 100 になります。

得られた明るさの値は，変数 x に代入して，10 行目で使っています。

10 行目 ␣␣␣␣lcd.text_pixels(str(x)+'%', True, 50, 40, font='courB24')

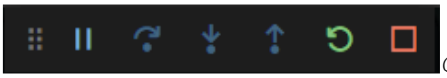
Display クラスの中にある text_pixels というメソッドを実行して，液晶ディスプレイに文字列を描きます。

text_pixels に続く丸カッコ( )の中に書いた一番最初の「str(x)+'%'」は，描きたい文字列です。str 関数は，数値である x を文字列に変換する処理を行います。また，文字列と文字列を連結する場合，「+」を使います。

次の「True」は，現在液晶ディスプレイに表示されている内容を消去するという意味です。次の数字は，描きたい文字列の左上角の X 座標の値で，今回は X 座標の値として，「50」を指定しています。その次の数字は，描きたい文字列の左上角の Y 座標の値で，今回は Y 座標の値として，「45」を指定しています。最後の，「font='courB24'」は，使用する書体（文字の形や大きさ）を指定しています。

また，このプログラムは無限に繰り返す処理になっています。

終了させるときは，VS Code の右上に表示されている

 の中から，  を左クリックします。

また，EV3 ブロック本体の「戻るボタン」を押しても，プログラムを終了させることができます。

この実験の結果を表１０－１にまとめてみましょう。適切な境目の値（しきい値と呼びます。）を決めるためには，３回ぐらい実験して光センサの値を読んで，平均した値をしきい値として使います。

表１０－１　ライン上と白い部分の光センサの値

回	ライン上	白い部分
1		
2		
3		
平均		

次に，ライン上と白い部分のしきい値を，

（ライン上の光センサの平均値　＋　白い部分の光センサの平均値）÷２

で求め，小数第１位を四捨五入して整数値にしましょう。

しきい値＝

このしきい値は，ちょうど，２つの値の中間となります。しきい値よりも小さいとき，ライン上にあり，逆に大きいとき，ラインからはずれ白い部分にあることがわかります。

図１０－５のフローチャートを参考にして，プログラムを prog10-2.py に示します。ここでは，例としてしき値を 55 としています。この値は，実験の結果から求めたものに修正しましょう。

【 prog10-2.py 】

```
 1 #!/usr/bin/env python3
 2 from ev3dev2.sensor.lego import ColorSensor
 3 from ev3dev2.motor import MoveTank,OUTPUT_B,OUTPUT_C
 4
 5 cs=ColorSensor()
 6 th=55                          # しきい値(%)
 7
 8 rbt=MoveTank(OUTPUT_B,OUTPUT_C)
 9 sp=20                          # スピード(%)
10
11 while True:
12 ␣␣␣␣if cs.reflected_light_intensity < th:
13 ␣␣␣␣␣␣␣␣rbt.on(0,sp)          # ライン上は，左カーブ開始
14 ␣␣␣␣else:
15 ␣␣␣␣␣␣␣␣rbt.on(sp,0)          # 白い部分は，右カーブ開始
```

6 行目　th=55

白い部分とライン上を判別するためのしきい値として 55 を変数 th に代入しています。しきい値は実験して得られた値にしましょう。

9 行目　sp=20

モータのスピード 20 を変数 sp に代入しています。あまりモータを速く回転させるとうまくラインをトレースできなくなるので，実験しながらちょうどよい値にしましょう。

11 行目　while True:
12 行目　␣␣␣if cs.reflected_light_intensity < th:
13 行目　␣␣␣␣␣␣␣␣rbt.on(0,sp)　　# ライン上は，左カーブ開始
14 行目　␣␣␣␣else:
15 行目　␣␣␣␣␣␣␣␣rbt.on(sp,0)　　# 白い部分は，右カーブ開始

while 文の条件式として True を書いていますので，段付けされた 12 行目から 15 行目までを繰り返し無限に実行します。

12 行目の if 文に書いている条件式「cs.reflected_light_intensity < th」の，cs.reflected_light_intensity は，光センサで計測した反射光の明るさを％で表し

た値が入っています。計測できる最も暗いときは 0，最も明るいときは 100 になります。

条件式「cs.reflected_light_intensity < th」は，明るさが変数 th に記憶されている値より小さいとき成り立ちます。この状態は，ライン上に光センサがありますから二重に段付けされた 13 行目を実行し，左にカーブを開始します。

条件式「cs.reflected_light_intensity < th」が成り立たないときは，白い部分に光センサがありますから，ライン上に戻るように 14 行目の else 文のつぎから二重に段付けされた 15 行目を実行し，右にカーブを開始します。

【チャレンジ１０－１】

prog10-2.py では，モータのスピードが 20 でやや遅くなっています。モータのスピードをもう少し速くして，ライントレースさせてみましょう。そのときに，どんな問題が起きるでしょうか？

【チャレンジ１０－２】

prog10-2.py では，ラインの左端をなぞるようにロボットが移動します。ラインの右端をなぞるようにロボットを移動させるためには，どのようにプログラムを修正すればよいでしょうか？

図１０－５のフローチャートによるライントレースは，ロボットが常にラインの境界部分を境にしてカーブして移動するので，「ジグザグ」と進み，あまり速くライントレースすることができません。例えば，モータのパワー値を 20 から 40 にしても，ロボットの移動速度は速くなりますがジグザグする動作は変わりません。何かよいアイディアはないでしょうか？

prog10-2.py では，左右にカーブしながらライントレースしていました。ラインがゆるい曲線であれば，もっとゆるい角度でターンすれば，なめらかに動作するでしょう。カーブする角度をゆるくするためには，停止していた片方のモータを遅い速度で回転させる方法があります。

例えば，左にカーブを開始する場合

```
rbt.on(0,20)
```

としていたところを

```
rbt.on(10,30)
```

に変更すると，左にカーブしていく角度がゆるくなります。

prog10-2.py を修正して，カーブする角度をゆるくしたプログラム prog10-3.py を入力しましょう。

【 prog10-3.py 】

```
 1 #!/usr/bin/env python3
 2 from ev3dev2.sensor.lego import ColorSensor
 3 from ev3dev2.motor import MoveTank,OUTPUT_B,OUTPUT_C
 4
 5 cs=ColorSensor()
 6 th=55                          # しきい値(%)
 7
 8 rbt=MoveTank(OUTPUT_B,OUTPUT_C)
 9 sp1=10                         # スピード(%)
10 sp2=30
11
12 while True:
13 ____if cs.reflected_light_intensity < th:
14 ________rbt.on(sp1,sp2)        # ライン上は，ゆるく左カーブ開始
15 ____else:
16 ________rbt.on(sp2,sp1)        # 白い部分は，ゆるく右カーブ開始
```

9 行目　sp1=10

10 行目　sp2=30

ゆるくカーブするときのモータのスピードを変数 sp1 と sp2 に代入しています。

14 行目　____rbt.on(sp1,sp2)

左側の車輪を回転させるモータのスピードを sp1 に，右側の車輪を回転させるモータのスピードを sp2 に設定しています。sp1 に 10，sp2 に 30 がそれぞれ代入されていますから，左にゆるくカーブを始めます。

16 行目　____rbt.on(sp2,sp1)

14 行目とは逆に，左側の車輪を回転させるモータのスピードを sp2 に，右側の車輪を回転させるモータのスピードを sp1 に設定しています。sp1 に 10，sp2 に 30 がそれぞれ代入されていますから，右にゆるくカーブを始めます。

【チャレンジ１０－３】

prog10-3.py では，ラインの左端をなぞるようにロボットが移動します。ラインの右端をなぞるようにロボットを移動させるためには，どのようにプログラムを修正すればよいでしょうか？

もっと速くラインをたどる方法はないのでしょうか？ 次は，ロボットの過去の状態も考慮してみます。光センサがライン上にあるときは，できるだけ前進させて，白い部分にあるときは，できるだけ速くライン上に戻すように考えてみましょう。

そのために，図１０－６のようにロボットの過去の状態（①）と現在の状態（②）に応じた４通りのパターンで考えます。

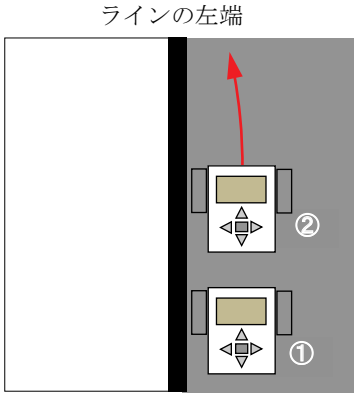
まず，ライン上からライン上に移動する【パターン１】では，すでにロボットはライン上にあるので，とてもゆるく左にカーブさせてできるだけ前進させます。直進してしまうとラインの右端を超えることがあるので，とてもゆるく左にカーブさせます。

ライン上から白い部分に入った【パターン２】では，ライン上にあったロボットが少しだけ，ラインを外れたことになるので，ゆるく右にカーブさせて，ライン上に戻すようにします。

白い部分からライン上に入った【パターン３】では，ライン上に入ったロボットの向きを少し修正するために，ゆるく左にカーブさせて，ライン上を移動するようにします。

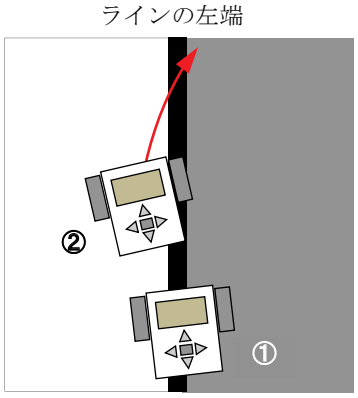
白い部分から白い部分を移動する【パターン４】では，ロボットがラインを外れて移動しているので，できるだけ速くライン上に戻るように右にカーブさせています。

【パターン１】①ライン上　⇒　②ライン上



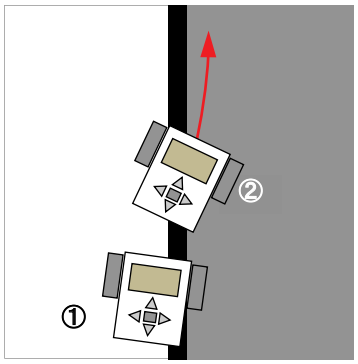
とてもゆるく左にカーブさせ，できるだけ前進させる

【パターン２】①ライン上　⇒　②白い部分



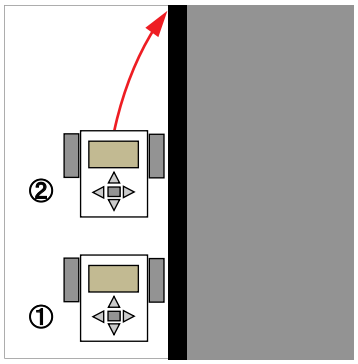
ゆるく右カーブさせ，ライン上に戻るようにさせる

【パターン３】①白い部分　⇒　②ライン上



ゆるく左にカーブさせ，ライン上を前進させる

【パターン４】①白い部分　⇒　②白い部分



右にカーブさせ，できるだけ速くライン上に戻るようにさせる

図１０－６　ロボットの過去の状態（①）と現在の状態（②）を考慮したライントレース

この考え方をフローチャートに描いた図を図１０－７に示します。

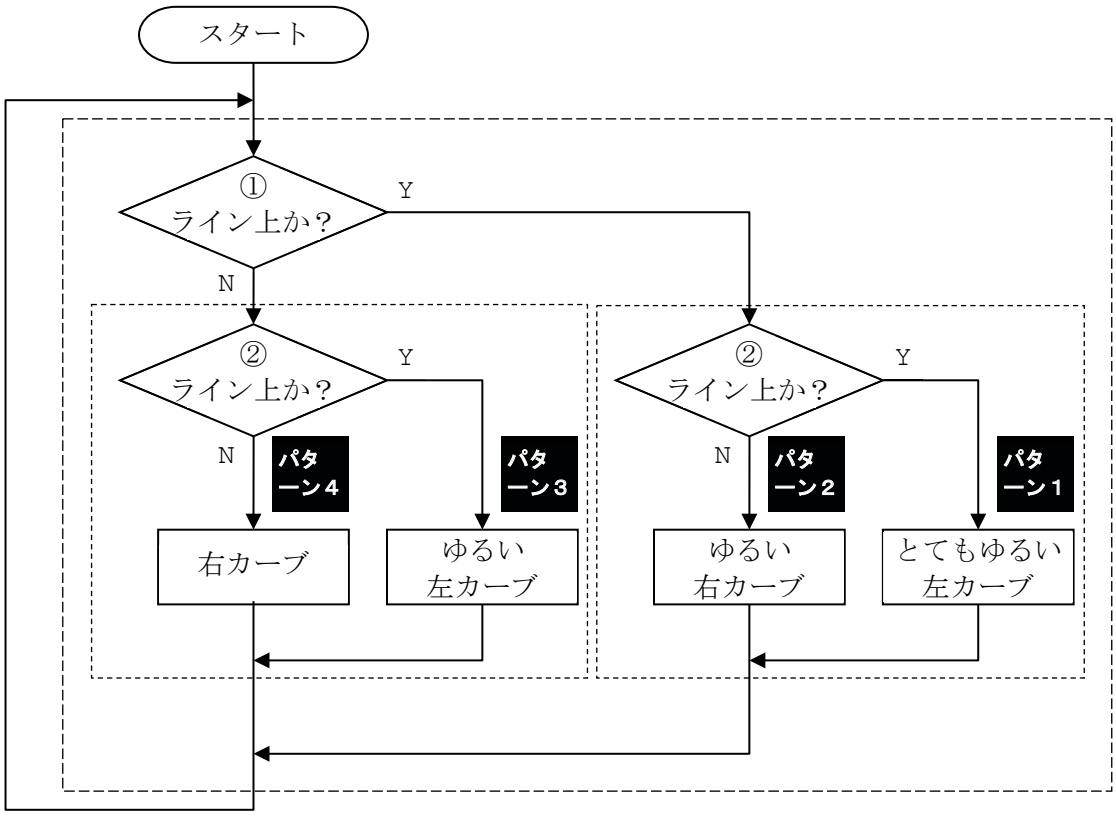


図１０－７　ロボットの過去の状態（①）と現在の状態（②）を考慮したラインとレースのフローチャート

それでは，このフローチャートに従った動きをするためのプログラム prog10-4.py を入力しましょう。このプログラムでは，if 文の中にさらに if 文を入れて，４通りのパターンを判断しています。

【 prog10-4.py 】

```
1 #!/usr/bin/env python3
2 from ev3dev2.sensor.lego import ColorSensor
3 from ev3dev2.motor import MoveTank,OUTPUT_B,OUTPUT_C
4
5 cs=ColorSensor()
6 th=55                  # しきい値(%)
7
8 rbt=MoveTank(OUTPUT_B,OUTPUT_C)
```

```
 9 sp1=30                  # スピード(%)
10 sp2=15
11 sp3=10
12
13 s1=1                   # 最初はライン上
14
15 while True:
16 ␣␣␣␣if cs.reflected_light_intensity < th:
17 ␣␣␣␣␣␣␣␣s2=1           # ライン上
18 ␣␣␣␣else:
19 ␣␣␣␣␣␣␣␣s2=0           # 白い部分
20
21 ␣␣␣␣if s1==1:
22 ␣␣␣␣␣␣␣␣if s2==1:      # ライン上→ライン上：とてもゆるい左カーブ
23 ␣␣␣␣␣␣␣␣␣␣␣␣rbt.on(sp1,sp2)
24 ␣␣␣␣␣␣␣␣else:         # ライン上→白い部分：ゆるい右カーブ
25 ␣␣␣␣␣␣␣␣␣␣␣␣rbt.on(sp3,sp1)
26 ␣␣␣␣else:
27 ␣␣␣␣␣␣␣␣if s2==1:      # 白い部分→ライン上：ゆるい左カーブ
28 ␣␣␣␣␣␣␣␣␣␣␣␣rbt.on(sp1,sp3)
29 ␣␣␣␣␣␣␣␣else:         # 白い部分→白い部分：右カーブ
30 ␣␣␣␣␣␣␣␣␣␣␣␣rbt.on(0,sp2)
31
32 ␣␣␣␣s1 = s2;
```

状態のパターンを判別するために変数 s1 と s2 を使っています。変数の値が 1 の場合「ライン上」，0 の場合「白い部分」にあることを意味しています。s1 は過去の状態（①）を示し，s2 は現在の状態（②）を示します。

13 行目　s1=1

最初にロボットはライン上あることにしているので，「ライン上」を意味する「1」を s1 に代入しています。

15 行目～32 行目　while True:

ライントレース処理の全体を無限に実行するための while 文です。

16 行目～19 行目

␣␣␣␣if cs.reflected_light_intensity < th:　……　else:　……

現在の状態を示す光センサの値 cs.reflected_light_intensity が th より小さいとき，「ライン上」にあるため，変数 s2 に「1」を代入しています。さもなければ，「白い

部分」にあるため，変数 s2 に「0」を代入しています。

21 行目～30 行目　␣␣␣␣if s1==1:　……　else:　……

このif 文で「パターン１と２」,「パターン３と４」の判断します。

22 行目～25 行目　␣␣␣␣␣␣␣␣if s2==1:　……　else:　……

このif 文で,「パターン１」と「パターン２」を判断し，図１０－６の処理を行います。

27 行目～30 行目　␣␣␣␣␣␣␣␣if s2==1:　……　else:　……

このif 文で,「パターン３」と「パターン４」を判断し，図１０－６の処理を行います。

32 行目　␣␣␣␣s1 = s2;

現在の状態を示す変数 s2 の値を，過去の状態を示す変数 s1 に代入して，状態を変化させます。

【チャレンジ１０－４】

prog10-4.py では，ラインの左端をなぞるようにロボットが移動します。ラインの右端をなぞるようにロボットを移動させるためには，どのようにプログラムを修正すればよいでしょうか？

１０．３　ラインの両端を使ってたどる方法

ここまでは，ラインの片方の端だけをなぞってラインをトレースする方法でした。誤ってなぞっている端と異なる方を検知すると，ライントレースする方向が逆転してしまうという問題もありました。

ラインの幅が太いときは，図１０－８のようにライントレースを速めるために，できるだけライン上のロボットを前進させるようにし，ラインを外れたときだけ，元のラインに戻るような動きにすることを考えてみましょう。

この例では，ライン上ではいつも前進するようにします（①）。ラインから外れて白い部分になったら，ロボットをターンさせてラインを探索します（②）。ラインを見つけたら再び前進します（③）。再度，ラインから外れて白い部分になったら，ロボットターンさせてラインを探索します（④）。ラインが見つかったら前進に戻ります（⑤）。この動きを繰り返すことで，ラインの両端を使ってすばやくライントレースができるはずです。

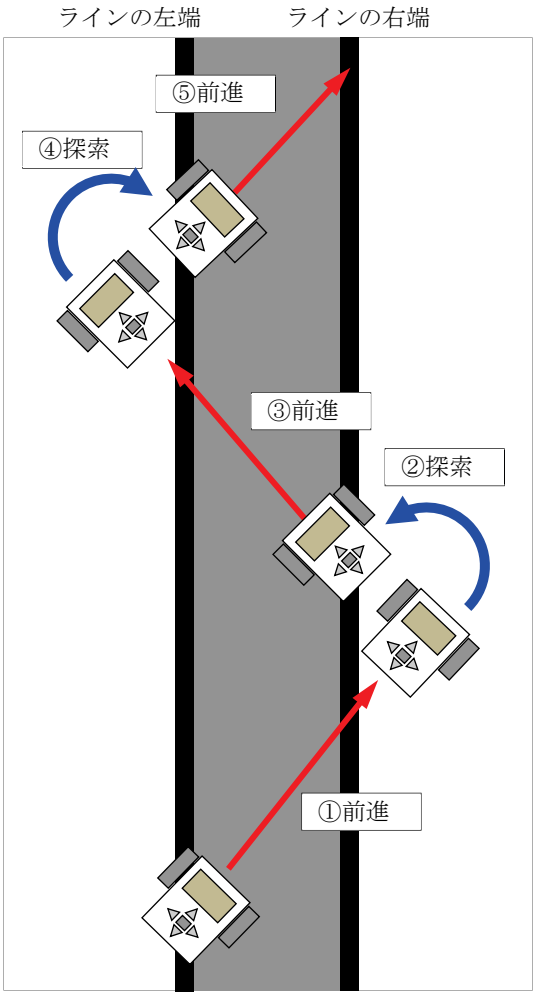


図１０－８　ライン上を前進していくライントレースの考え方

ところが，図１０－８で示した②と④のラインの探索処理は，どちら側にターンすればよいかわかりません。光センサが１つしかないので，ラインの左端と右端を区別できないからです。

そこで，図１０－９のようなラインの探索方法を考えてみましょう。まず最初にある一定時間だけ「左ターン」しながらラインを探索します。この時間内でラインが見つかった場合を図１０－９（ａ）に示します。この状態でラインを見つけることができなかったら，今度は，ターンする時間を増やして「右ターン」しながらラインを探索します。ターンする時間を増やすことで，探索する範囲を広げています。この時間内でラインが見つかった場合を図１０－９（ｂ）に示します。

それでも，ラインが見つからないときは，さらにターンする時間を増やして再び「左ターン」しながらラインを探索します。この時間内でラインが見つかった場合を図１０－９（ｃ）に示します。まだ，ラインが見つからないときは，さらにターンする時間を増やして再び「右ターン」しながらラインを探索します。この時間内でラインが見つかった場合を図１０－９（ｄ）に示します。

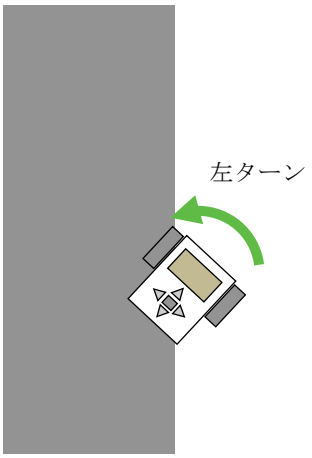
このように，ある一定時間だけ「右ターン」と「左ターン」を交互に行いながら，ラインを探索していくことで，ラインの左端でも右端でも見つけることができます。

光センサが１つのときは，ラインの探索は難しくなりますが，プログラムの工夫にやりがいはあります。

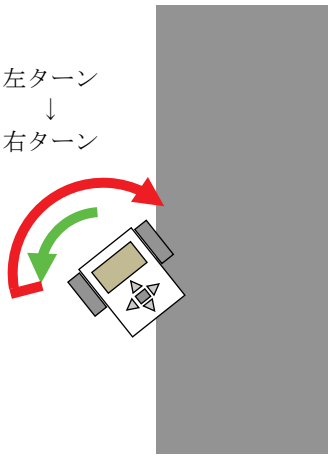
ライン上を前進するライントレース・ロボットのフローチャートを図１０－１０に示します。ロボットは，最初ライン上にあるので，前進します（①）。はじめに探索するターンの方向を一旦「左」にしておきます。「右」にしてもライントレースはできますが，ここでは左にしています。

つぎに，ロボットがラインを外れて白い部分に来ているかどうかを判断します（②）。もし，白い部分になったら，ラインの探索を始めます（③）。探索時間を 0 に初期化しておいてから，探索するターンの方向に従って，「左ターン」または「右ターン」を開始します（④）。探索時間を増加させた後，探索時間内で，光センサを使ってライン上にあるかどうかを検知します。探索時間内でライン上になれば，前進します（⑤）。探索時間を超えたら，探索するターンの方向を「左ターン」ならば「右ターン」，「右ターン」ならば「左ターン」のように交換します（⑥）。所定の時間内でラインを探索できなかったかどうか判断し（⑦），探索できなかったら，少し後退し再び探索のためのターンを開始します。

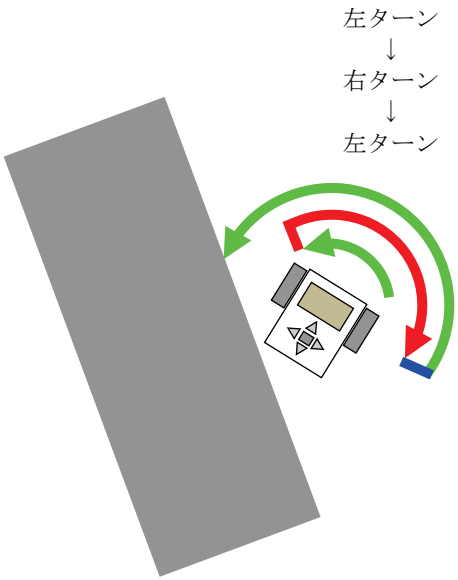
まっすぐなライン上を右端⇒左端⇒右端⇒左端をたどりながら前進する場合，探索のためのターンの方向が，ちょうど交互になり，無駄な探索のためのターンが減ります。



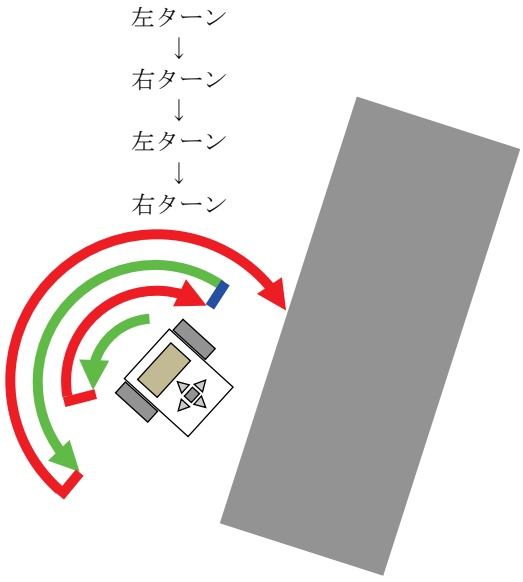
（ａ）左ターンのみでラインを見つける場合



（ｂ）左ターンでラインが見つからないので，途中で右ターンしてラインを見つける場合



（ｃ）左ターンでラインが見つからず，途中で右ターンに変更しても，まだ見つからないときは，再度，探索範囲を広げて左ターンして，ラインを見つける場合



（ｄ）左ターンでラインが見つからず，途中で右ターンに変更しても，まだ見つからないときは，再度，探索範囲を広げて左ターンする。それでもラインが見つからないときはさらに探索範囲を広げて右ターンしてラインを見つける場合

図１０－９　白い部分からラインを探索する方法

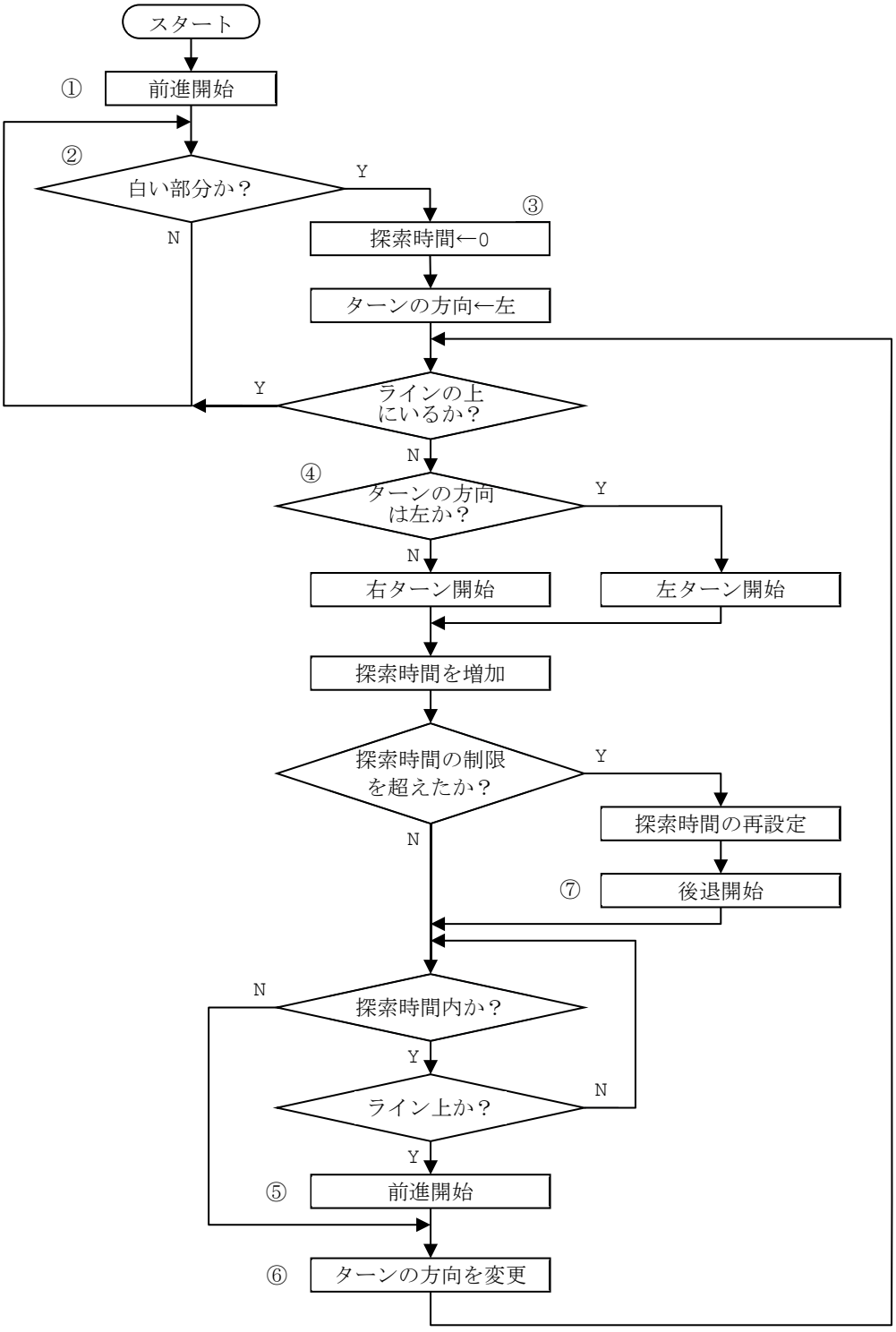


図１０－１０　ライン上を前進するライントレースのフローチャート

それでは，このフローチャートに従った動きをするためのプログラム prog10-5.py よう。このプログラムは繰り返しや判断が何重にもなっているので，段付けするためのスペー

スは２個にしています。

EV3 ブロックは，内部に時計をもっていて，その時刻を知ることができます。この時刻は実際の時刻とは異なり，EV3 ブロックがオンになってからカウントされています。time()は，EV3 の時刻を示しています。EV の時刻を使って探索のためのターンする時間を制限することができます。

【 prog10-5.py 】

```
#!/usr/bin/env python3
from time import time
from ev3dev2.sensor.lego import ColorSensor
from ev3dev2.motor import MoveTank,OUTPUT_B,OUTPUT_C

cs=ColorSensor()
th=55                          # しきい値(%)

rbt=MoveTank(OUTPUT_B,OUTPUT_C)
sp=30                          # スピード(%)

rbt.on(sp,sp)                  # 最初はライン上にあるので直進

while True:
    if cs.reflected_light_intensity >= th:
        ti=0.0                      # ラインから外れて白い部分に来た
        online=False
        dir=1                       # 一番はじめは左ターンして探索
        while online==False:
            if dir==1:
                rbt.on(-sp,sp)      # 左ターンしてラインを探索
            else:
                rbt.on(sp,-sp)      # 右ターンしてラインを探索

            ti += 0.1;              # 探索時間を長くしていく
            if ti > 1.2:            # 探索時間内にラインを
                ti=0.2              # 見つけられなかった場合
                rbt.on(-sp,-sp)     # 少し後退する。

            ct=time();              # 現在時刻
            while time()-ct < ti: # 探索時間内で繰り返し
                if cs.reflected_light_intensity < th:
                    rbt.on(sp,sp)   # ライン上は，直進
                    online=True
```

```
35  ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣break          # 繰り返しを抜け出す
36
37  ␣␣␣␣␣␣␣␣␣␣␣␣dir = -dir              # ターンの方向を交換
```

変数 online は，探索してラインが見つかったら True を，見つからなかったら False を代入して使います。また，変数 dir は，ラインを探査するためにターンする方向を示し，1 ならば左ターン，-1 ならば右ターンとしています。左右のターンの交換は，「1 ⇒ -1」，「-1 ⇒ 1」という計算が必要になりますから，dir の「プラス」と「マイナス」を互いに変えるため dir に「-1」をかけた値を再度 dir に代入します。

2 行目　from time import time

time は，時間に対して処理するための様々なプログラムを集めたモジュールです。このモジュールの中から，EV3 ブロック内部の時刻を返す time というプログラム（関数と呼びます。）を読み込みます。

12 行目　rbt.on(sp,sp)

最初にロボットはライン上にありますから前進させます。

14 行目～37 行目　while True:

この while 文は無限に実行するようになっています。この中に，ラインを外れたときの処理を書いています。

15 行目␣␣␣␣if cs.reflected_light_intensity >= th:

現在の状態を示す光センサの値 cs.reflected_light_intensity が th 以上かどうか判断し，「白い部分」にいる処理を，16 行目～17 行目に行います。

16 行目　␣␣␣␣␣␣␣␣ti=0.0

一旦，探索する時間を 0.0 秒にしておきます。

17 行目　␣␣␣␣␣␣␣␣online=False

白い部分にいるので「ライン上」であることを示す変数 online に False を代入します。

18 行目　␣␣␣␣␣␣␣␣dir=1

最初は左ターンから探索するため，その方向を示す変数 dir に 1 を代入します。

19 行目～37 行目　␣␣␣␣␣␣␣␣while online==False:

「ライン上」であることを示す変数 online が False の間，すなわち，ロボットがライ

ンから外れ，白い部分にいる間，実行します。

20 行目～23 行目　␣␣␣␣␣␣␣␣␣␣␣␣if dir==1: …… else: ……

dir の値が 1 ならば，左ターンを開始し，さもなければ，右ターンを開始します。

25 行目　␣␣␣␣␣␣␣␣␣␣␣␣ti += 0.1;

探索する時間を設定している変数 ti の値を 0.1 秒増加させます。増加させる時間は，実験してちょうどよい値に変更しましょう。図１０－９で説明したように，探索するための時間は，段々と増えていきます。

26 行目～28 行目　␣␣␣␣␣␣␣␣␣␣␣␣if ti > 1.2:……

探索時間が 1.2 秒より大きくなったとき，ラインを見つけられなかったと判断し，0.2 秒間後退します。探索時間の制限や後退する時間は，実験してちょうどよい値に変更しましょう。

30 行目　␣␣␣␣␣␣␣␣␣␣␣␣ct=time()

探索開始時点の EV3 ブロック内の時刻を ct に入れます。この値は，31 行目の while 文の条件の中で使います。

31 行目　␣␣␣␣␣␣␣␣␣␣␣␣while time()-ct < ti:

この while 文を使って，決められた探索時間（ti）中にラインを見つけます。while 文の条件式 time()-ct<ti は，現在の EV3 ブロック内の時刻 time()から探索を開始したときの EV3 ブロックの時刻を引き算し，探索に要している時間 time()-ct を求め，その値が決められた探索時間（ti）よりも小さい間，探索をします。もし，決められた探索時間を過ぎてしまったら，ラインを見つけていなくても，この while 文を終わり 37 行目に実行が移ります。

32 行目　␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣if cs.reflected_light_intensity < th:

現在の状態を示す光センサの値 cs.reflected_light_intensity が th より小さいかどうか判断し，「ライン上」にいる処理を，33 行目～35 行目で行います。

33 行目　␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣rbt.on(sp,sp)　　# ライン上は，直進

ラインが見つかると，前進を開始します。

34 行目　␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣online=True

ラインが見つかったことを示す True を変数 online に入れます。

35 行目　␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣break

このbreak文によって，一番内側の繰り返し処理（ここでは，31行目のwhile文による32行目～34行目の繰り返し処理）を中断し，37行目に実行が移ります。

このように，ある条件を満たしていたら，繰り返しをやめたいときにbreak文を使います。

37 行目　␣␣␣␣␣␣␣␣␣␣␣␣dir = -dir

変数dirの値が1のとき，-1となり，逆に，-1のとき1になります。この計算結果を使って，探索するターンの方向の左右を交換しています。

【チャレンジ１０－５】

prog10-5.pyでは，まっすぐな太いラインの左端と右端で交互にターンしながら前進していくと，かなり速くライントレースできます。しかし，小判型のテスト用紙のようなラインでは，曲線部における探索のためのターンに無駄な動きが目立ちます。このような無駄を減らし，いろいろな形のラインでももっと速くライントレースするためには，どのような工夫をしたらよいでしょうか？

＜ヒント＞

「左ターンでラインを見つけた回数をカウントする変数na」と「右ターンでラインを見つけた回数をカウントする変数nb」を使って，探索を開始するときの方向を決める方法があります。この方法を使うときは，na+nbが指定した回数（例えば12）を超えたら，naとnbを0にして状態をたまにリセットするとよいでしょう。

チャレンジ１０－５のプログラム例をprog10-6.pyに示します。網掛け部分によく注意してプログラムを読みましょう。また，このプログラムからフローチャートを書いてみましょう。

【 prog10-6.py 】

```
1 #!/usr/bin/env python3
2 from time import time
3 from ev3dev2.sensor.lego import ColorSensor
4 from ev3dev2.motor import MoveTank,OUTPUT_B,OUTPUT_C
5
6 cs=ColorSensor()
7 th=55                          # しきい値(%)
8
```

```
rbt=MoveTank(OUTPUT_B,OUTPUT_C)
sp=30                          # スピード(%)

na = 0          # 左ターンでライン探査できた回数
nb = 0          # 右ターンでライン探査できた回数

rbt.on(sp,sp)  # 最初はライン上にあるので直進

while True:
  if cs.reflected_light_intensity >= th:
    ti=0.0        # ラインから外れて白い部分に来た
    online=False

    if na > nb:  # 左右のターンでライン探査できた回数
      dir= 1     # を比較して，探査する方向を決める
    else:
      dir=-1

    if na + nb > 12: # ライン探査カウンタのリセット
      na=0
      nb=0

    while online==False:
      if dir==1:
        rbt.on(-sp,sp)  # 左ターンしてラインを探索
      else:
        rbt.on(sp,-sp)  # 右ターンしてラインを探索

      ti += 0.1;          # 探索時間を長くしていく
      if ti > 1.2:        # 探索時間内にラインを見つけられなかった場合
        ti=0.2
        rbt.on(-sp,-sp)  # 少し後退する。

      ct=time();               # 現在時刻
      while time()-ct < ti:  # 探索時間
        if cs.reflected_light_intensity < th:
          rbt.on(sp,sp)    # ライン上は，直進
          online=True
          if dir==1:
            na += 1      # 左ターンでライン探査できた回数を 1 増加
          else:
            nb += 1      # 右ターンでライン探査できた回数を 1 増加
          break
```

```
52
53  ␣␣␣␣␣␣dir = -dir              # ターンの方向を交換
```

【チャレンジ１０－６】

大きな白い厚紙の上に黒のテープなどを使って，いろいろな太さで直線や曲線のラインを作って，「ライントレース・ロボット」を使ったタイムレースをしてみましょう。

付　録

※本付録に掲載した全てのWebページは引用であり，その引用元はURL等として記載しています。

※本付録に掲載した全てのソフトウェアのインストール時の表示画面等は引用であり，その引用元は当該ソフトウェアの名称等として記載しています。

付録 A

Visual Studio Code (VS Code)のインストールと初期設定の仕方

Visual Studio Code (VS Code)のインストールと初期設定の仕方について，Windows パソコンを利用する場合について説明します。

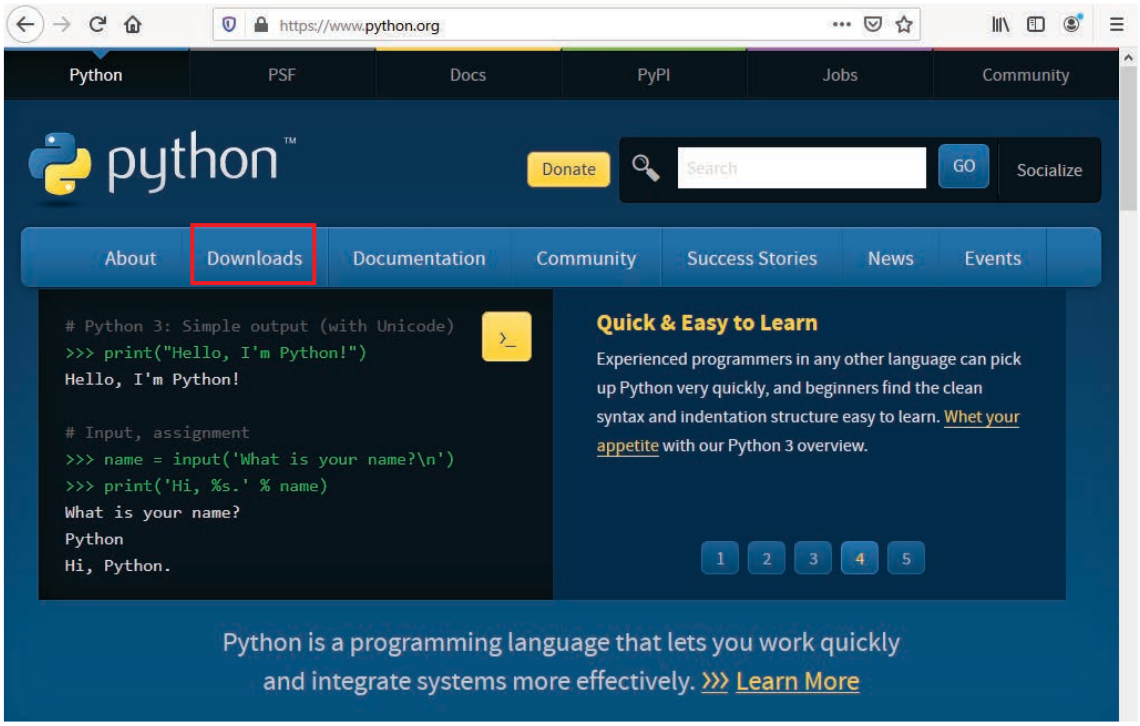
【ステップ１】Windows パソコン側に Python のインストール

EV3 で動作する OS（イメージファイル名：ev3dev-stretch-ev3-generic-2020-04-10.img）にインストールされている Python のバージョンは，3.5.3 のため，パソコン側も同じバージョンにして，文法チェックの精度を高めます。

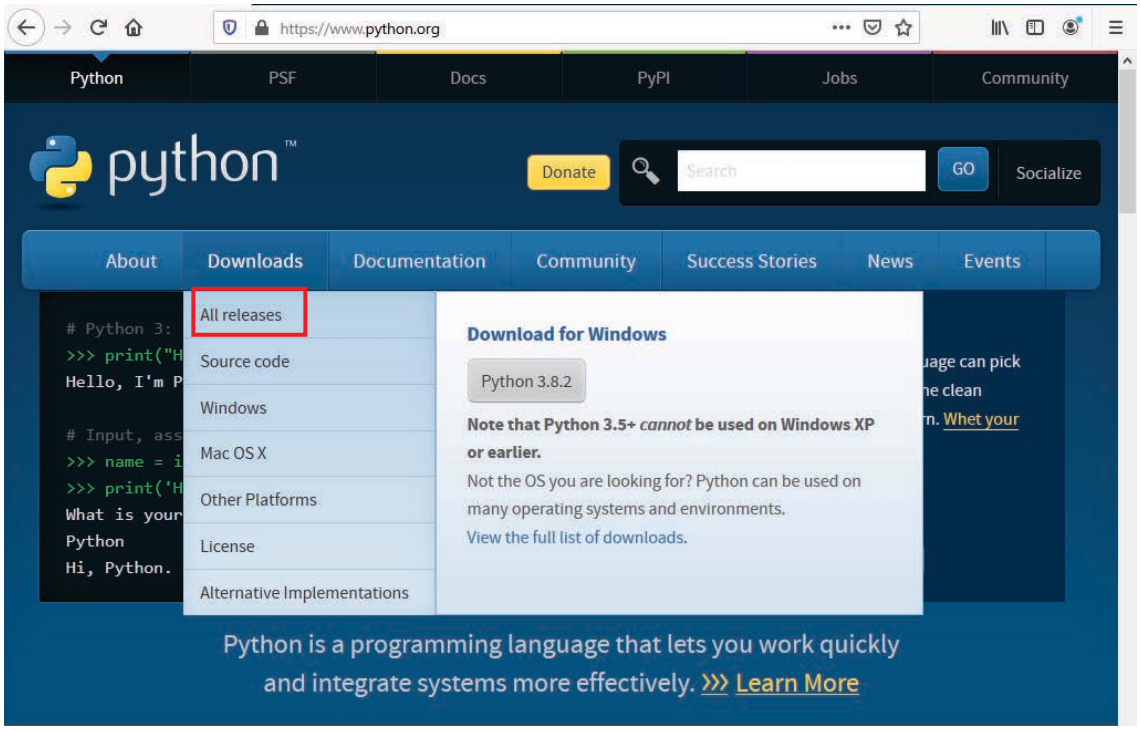
Windows パソコンは，「Windows 10 (64bit)」とします。インストールは，管理者権限のあるアカウントでサインインして行います。

Web ブラウザで
https://www.python.org/
を開きます。

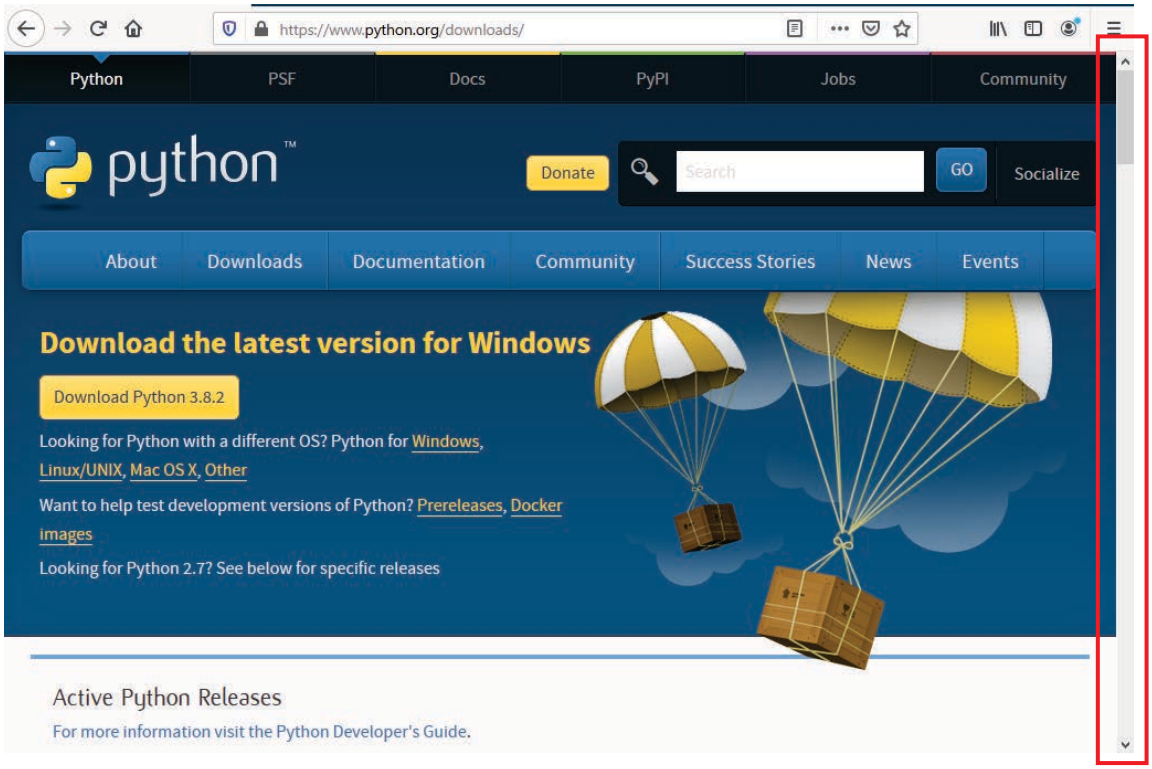
「Downloads」にマウスカーソルを合わせます。



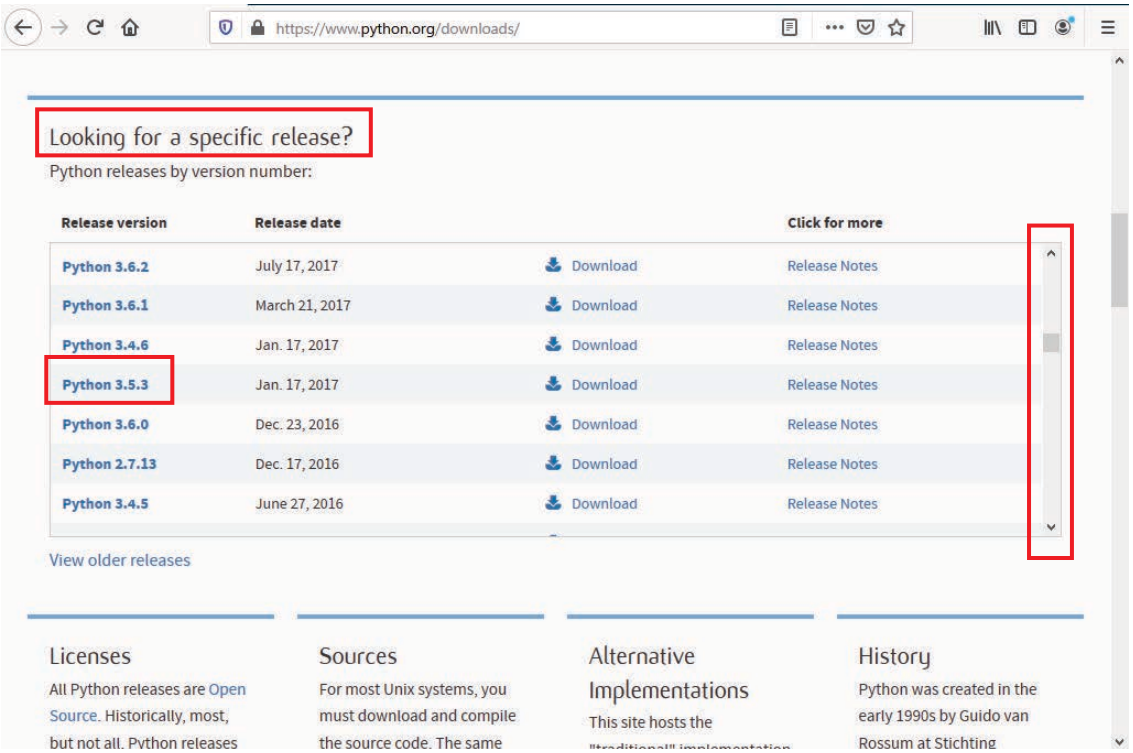
「All releases」を左クリックします。



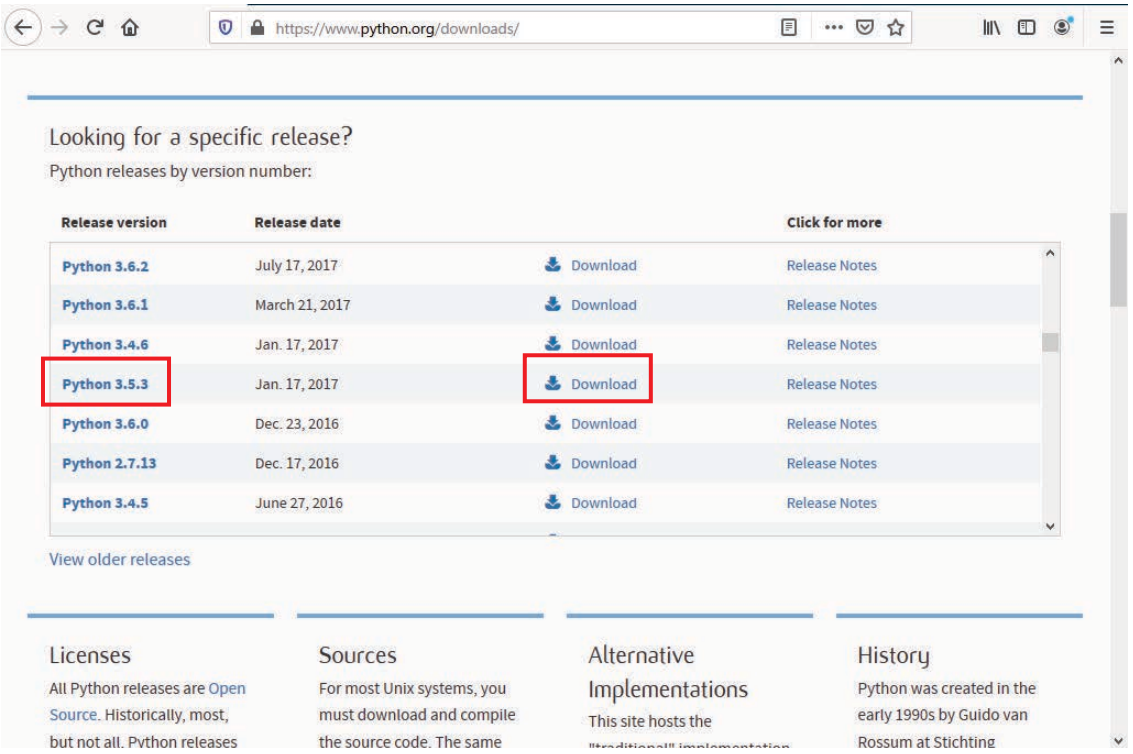
画面を下にスクロールさせ，「Looking for a specific release?」を表示させます。



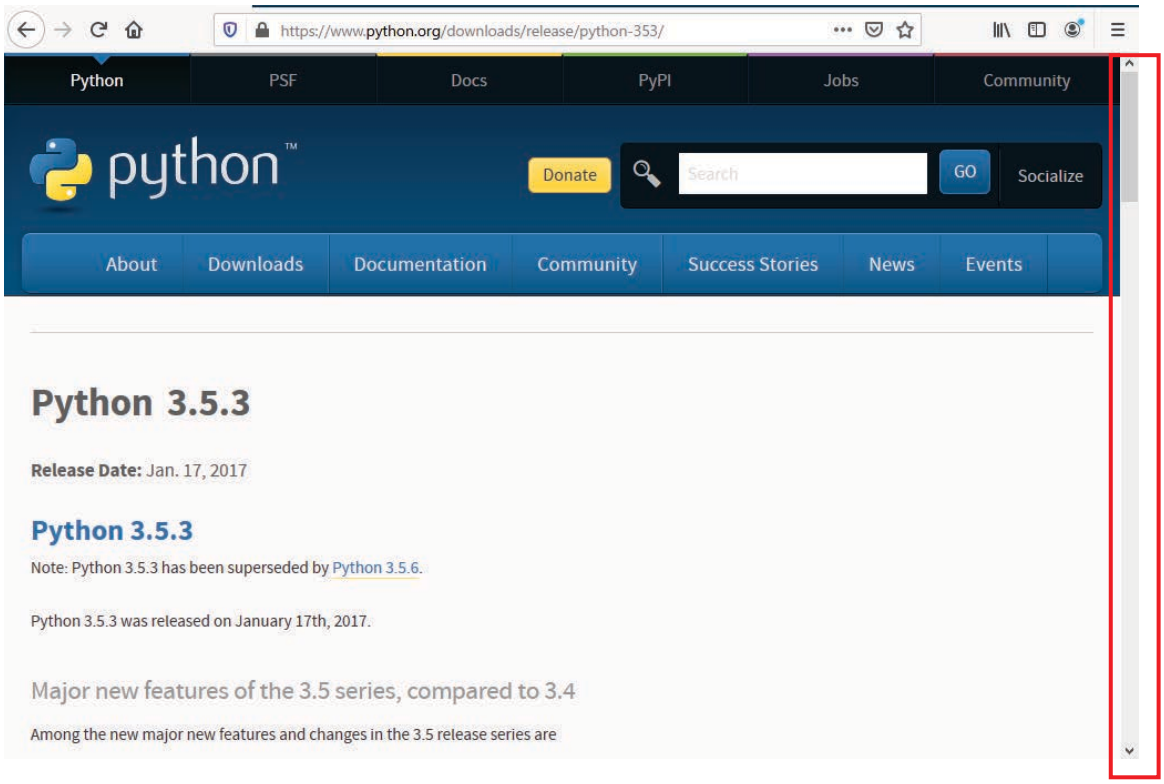
「Looking for a specific release?」の中にあるスクロールバーを使って,Release version が「Python 3.5.3」を表示させます。



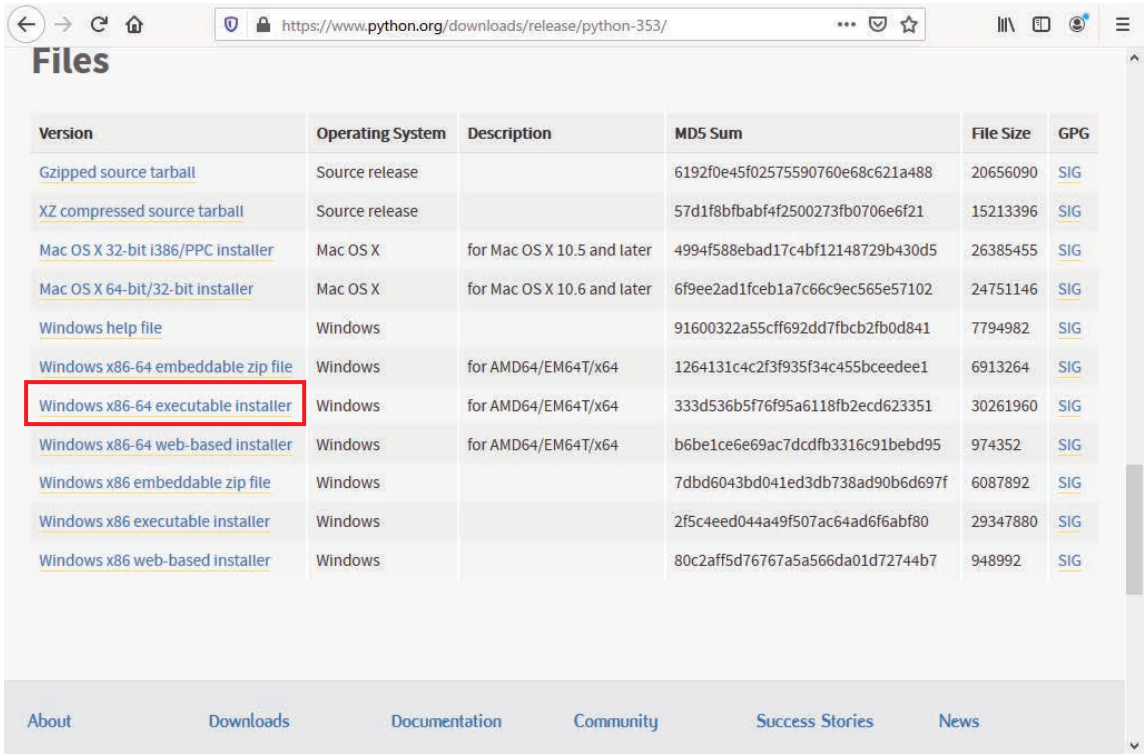
「Python 3.5.3」の「Download」を左クリックします。



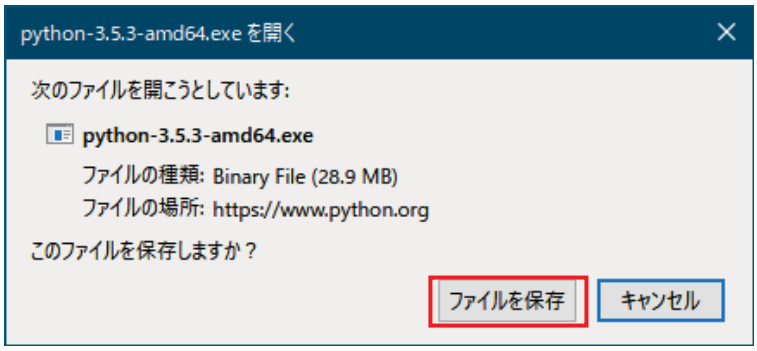
画面を下にスクロールさせ，「Files」を表示させます。



「Windows x86-64 executable installer」を左クリックします。



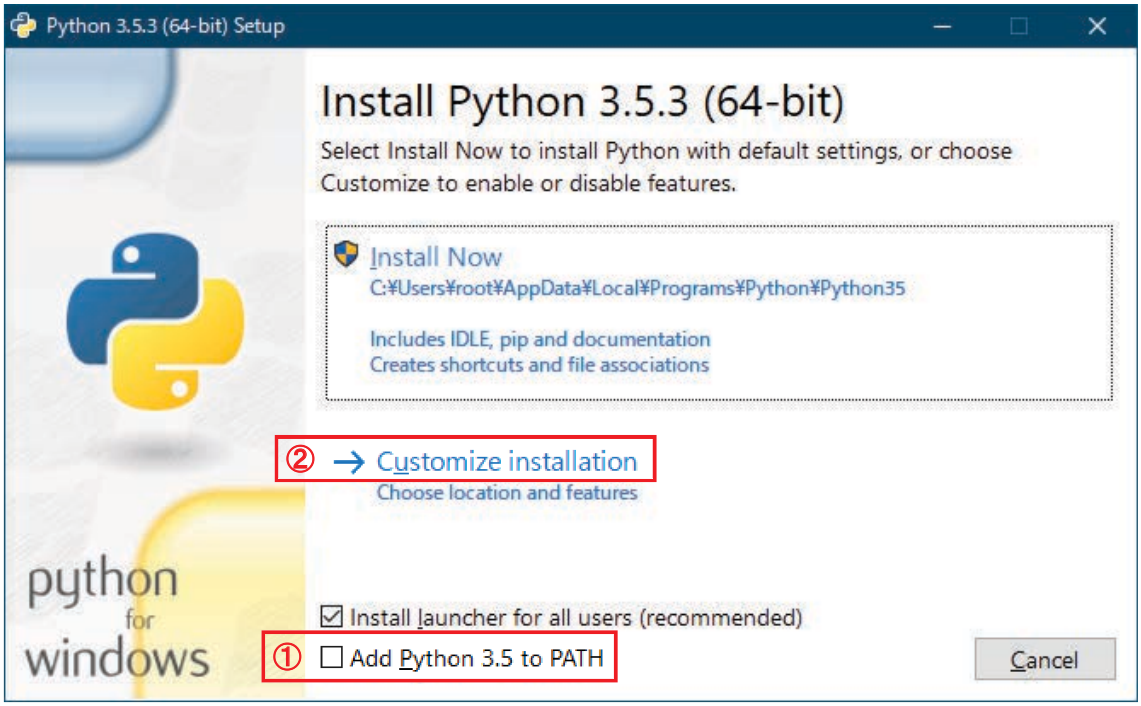
「ファイルを保存」を左クリックして，「python-3.5.3-amd64.exe」をパソコン側にダウンロードします。



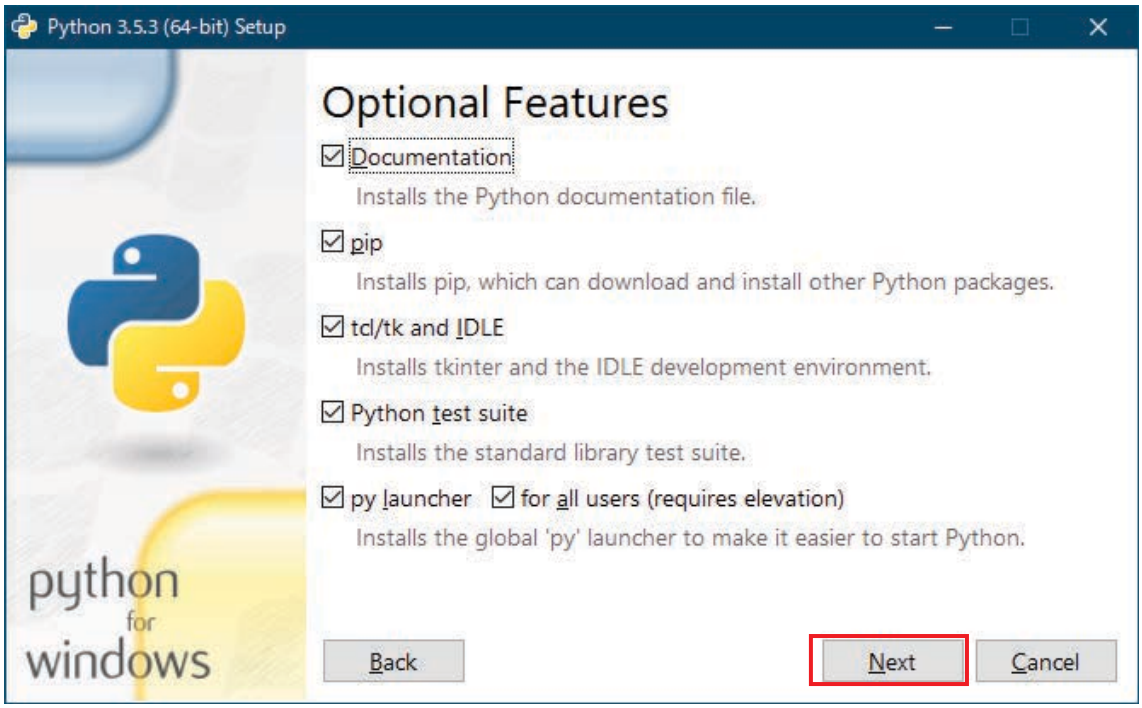
「python-3.5.3-amd64.exe」（Windows において，「ファイル名拡張子」を表示するようにチェックを入れておくと，.exe が表示されます。）というファイル名がダウンロードされます。

「python-3.5.3-amd64.exe」を右クリックして，「管理者として実行」を左クリックします。

①「Add Python 3.5 to PATH」の□にチェックを入れます。
②「Customize installation」を左クリックします。



「Next」を左クリックします。

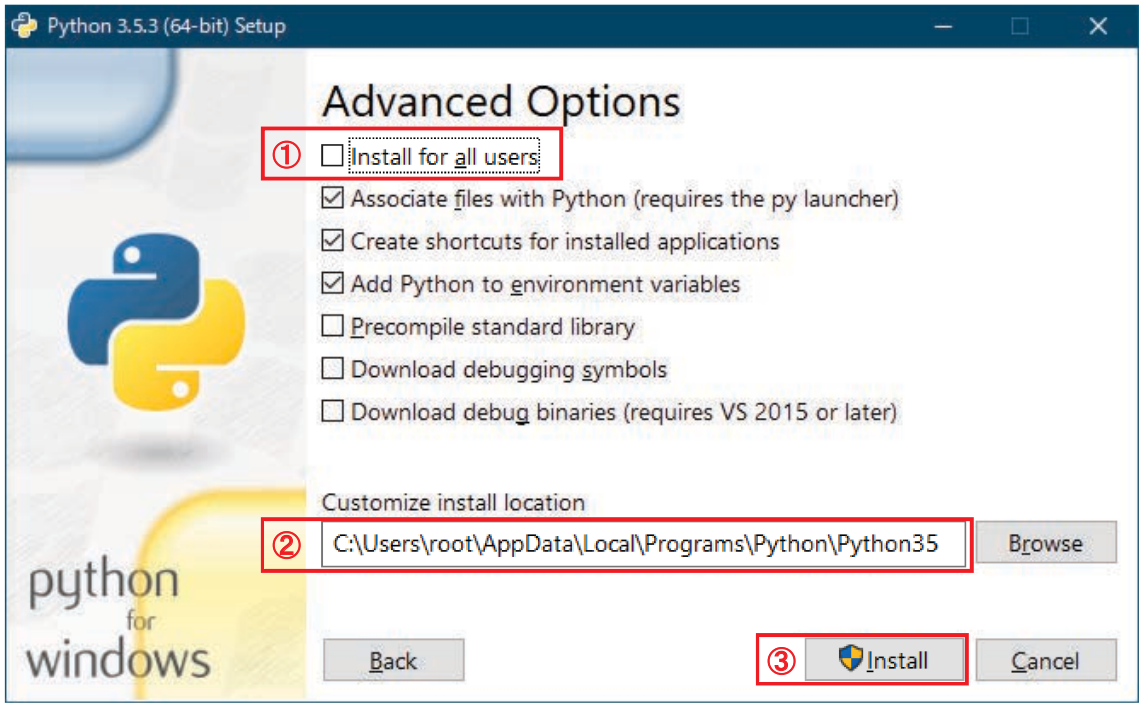


①「Install for all users」の□にチェックを入れます。

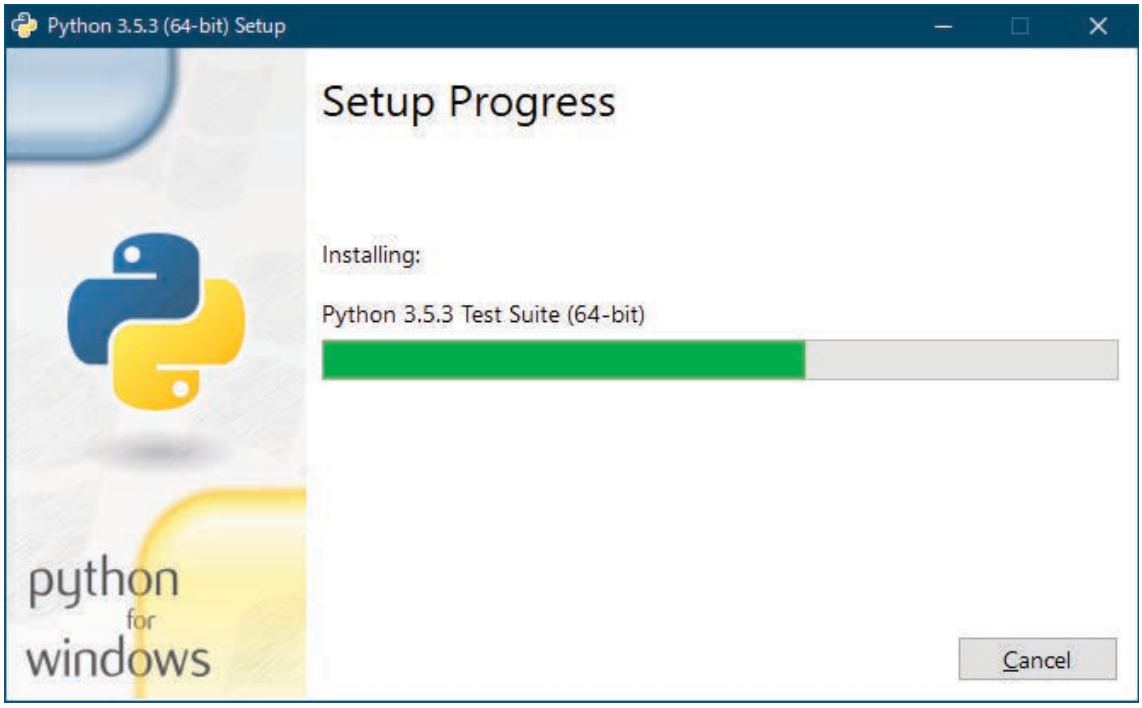
②枠内を C:Program Files¥Python35 に修正します。

※　「¥」は，バックスラッシュで表示されます。

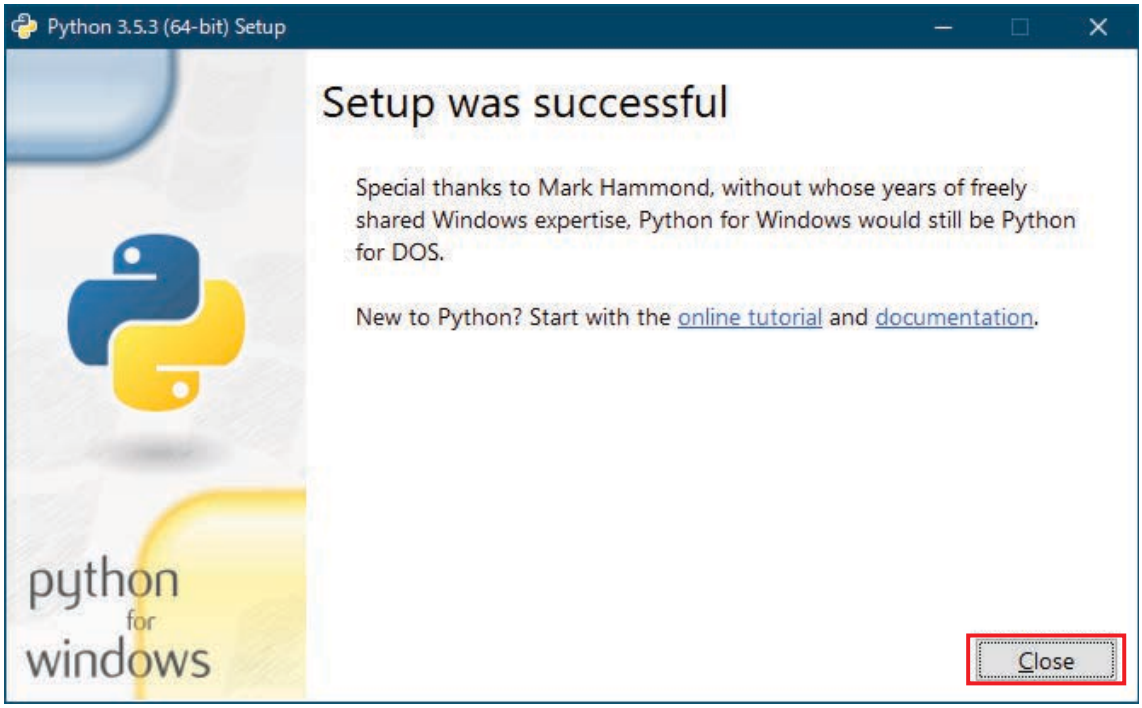
③「Install」を左クリックします。



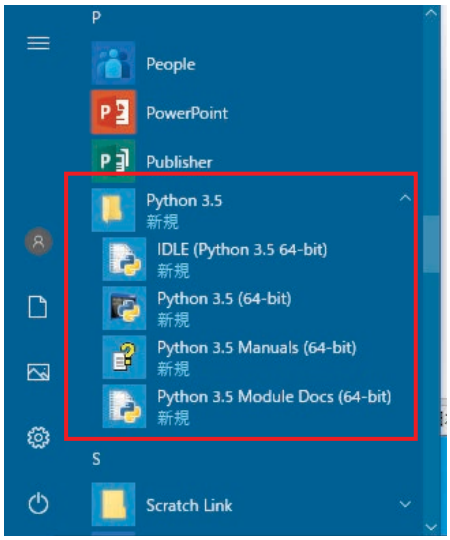
インストールが始まります。



インストールが完了すると，以下の画面が表示されます。
「Close」を左クリックして，終了します。



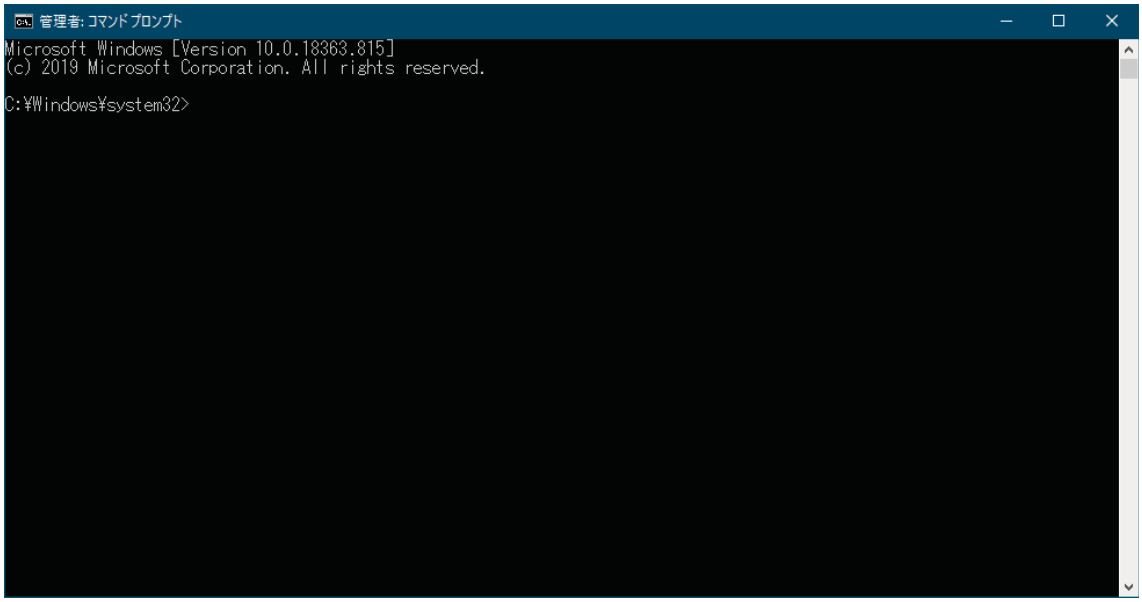
スタートメニューに「Python 3.5」フォルダが追加され，メニューのあることを確認します。



次に，Python 用ライブラリをインストールします。

スタートメニューの「Windows システムツール」の「コマンドプロンプト」を右クリックし，「その他」を右クリックし「管理者として実行」を左クリックして，コマンドプロンプトを管理者として実行します。

「管理者：コマンドプロンプト」が起動します。

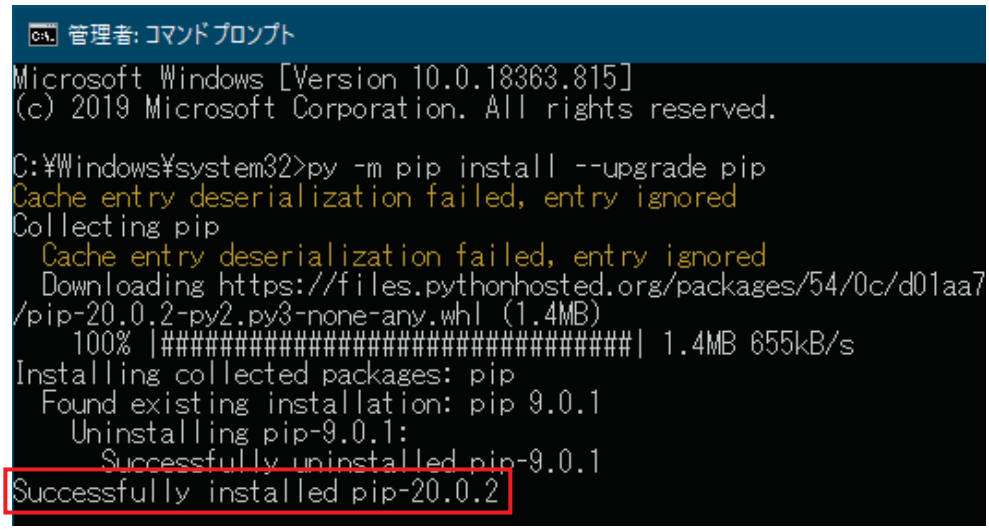


ライブラリ管理用コマンド pip を更新するため

```
py -m pip install --upgrade pip
```

と入力します。

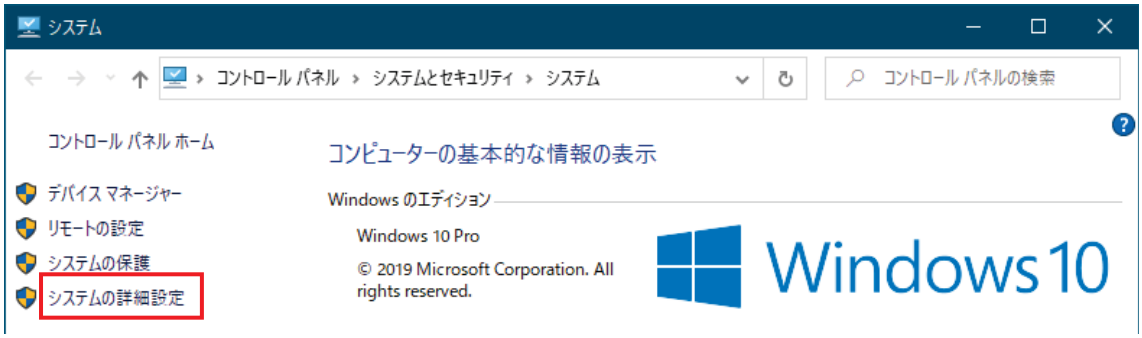
「Successfully installed pip-20.0.2」と表示されていれば，更新に成功しています。



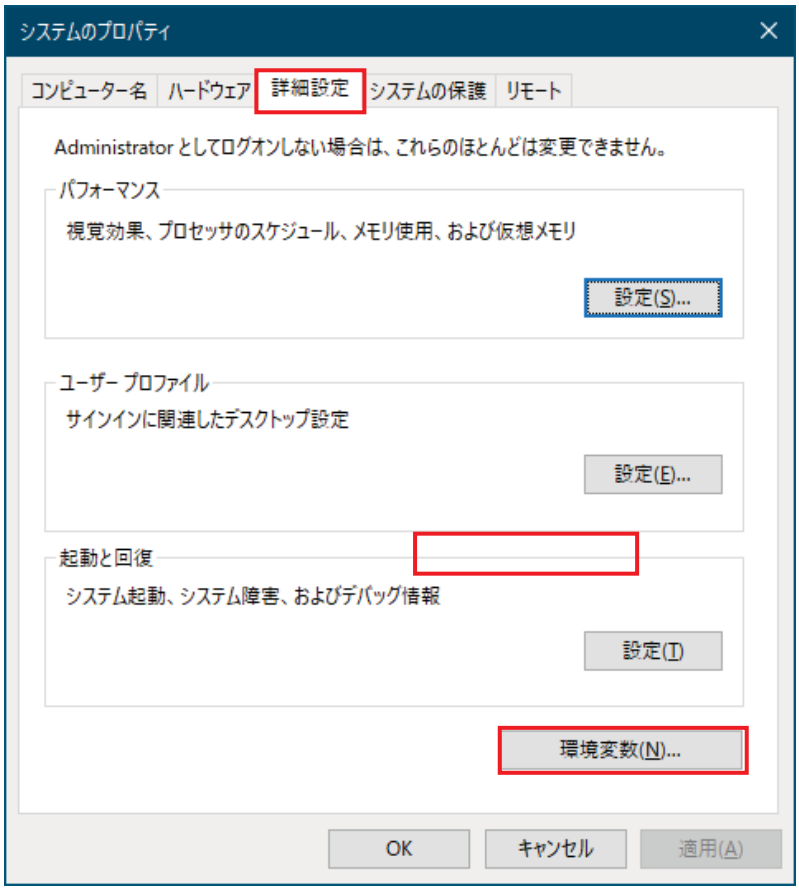
更新が失敗した場合，以下の手順でプロキシーサーバの設定をします。

スタートメニューの「Windows システムツール」のコントロールパネルを開き，「システムのセキュリティ」を開き「システム」を開きます。

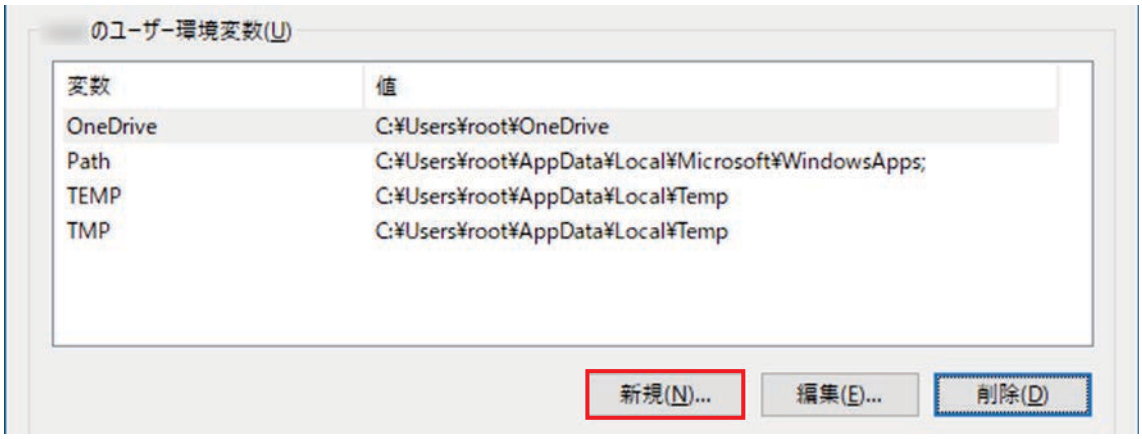
「システムの詳細設定」を左クリックします。



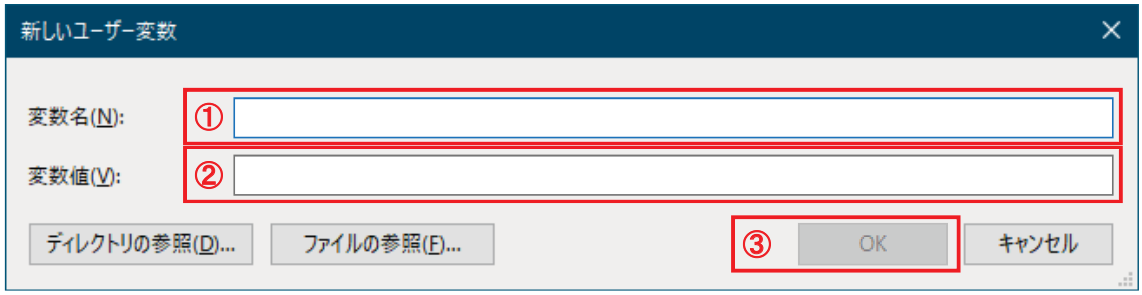
「システムのプロパティ」が表示されるので,「詳細設定」タブを左クリックし,「環境変数」ボタンを左クリックします。



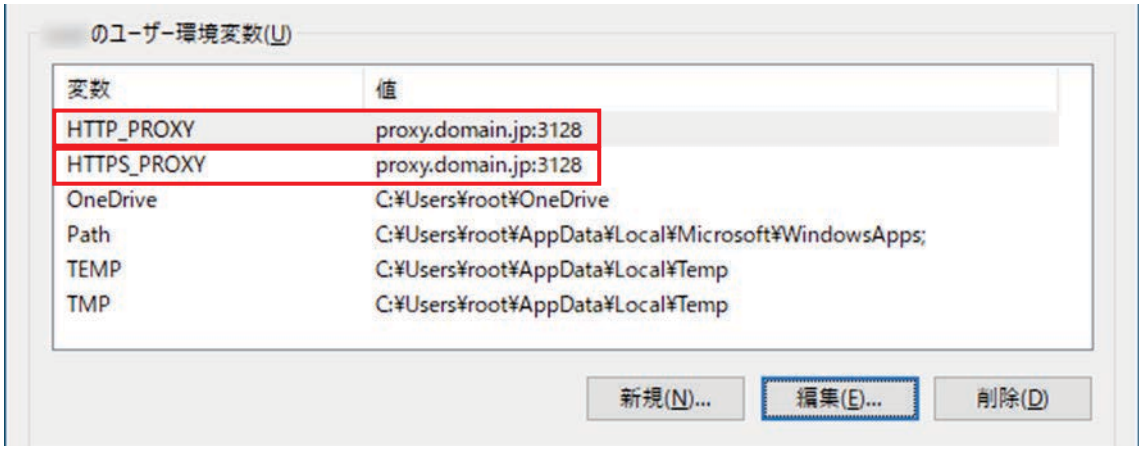
「新規」ボタンを左クリックします。



①「変数名」の枠内に，HTTP_PROXY を入力します。
②「変数値」の枠内に，プロキシーサーバの名称と必要に応じてポート番号を入力します。
　例えば，proxy.domain.jp:3128 にように入力します。
③「OK」ボタンを左クリックします。



同様な手順で，変数名 HTTPS_PROXY も追加します。



環境変数の設定が完了したら，一旦サインアウトして，サインインします。

pip の更新が完了後，setuptools をインストールするため

pip install -U setuptools

と入力します。

```
C:¥Windows¥system32>pip install -U setuptools
Collecting setuptools
  Using cached setuptools-46.1.3-py3-none-any.whl (582 kB)
Installing collected packages: setuptools
  Attempting uninstall: setuptools
    Found existing installation: setuptools 28.8.0
    Uninstalling setuptools-28.8.0:
      Successfully uninstalled setuptools-28.8.0
Successfully installed setuptools-46.1.3
```

画像処理関係のライブラリをインストールするため,

pip install pillow

と入力します。

Python のソースコードの品質検査用ライブラリをインストールするため,

pip install pylint

と入力します。

ev3 用 Python の入力補完機能用ライブラリをインストールするため

pip install python-ev3dev2

と入力します。

インストール済みのライブラリの一覧を表示するため

pip list

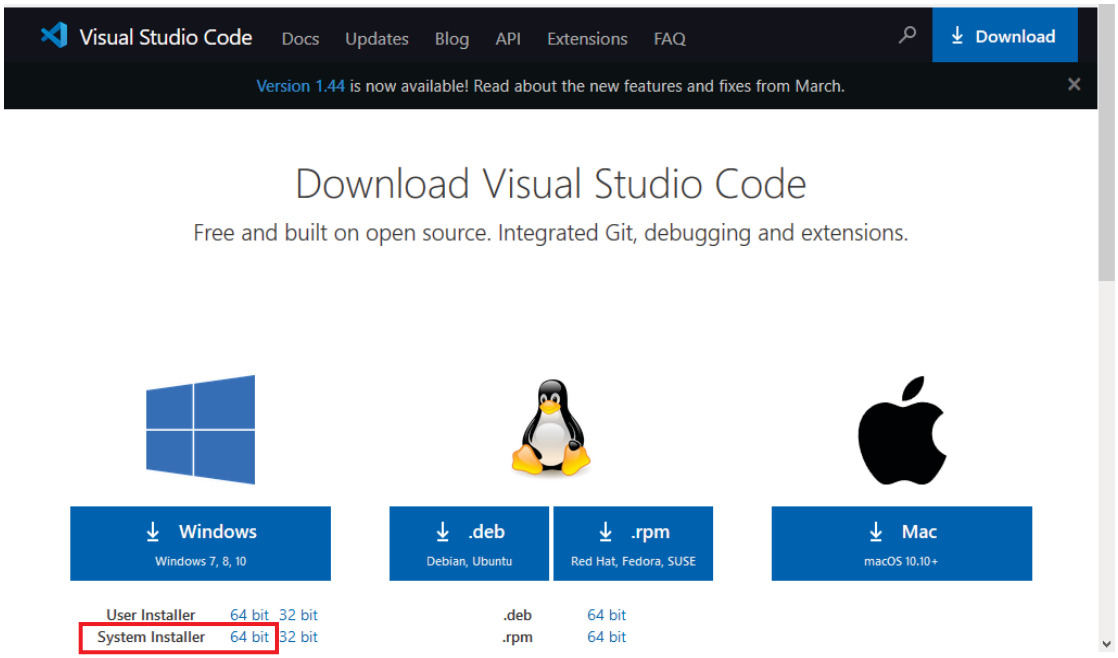
と入力します。インストールしたライブラリが表示されていることを確認します。

コマンドプロンプトを終了します。

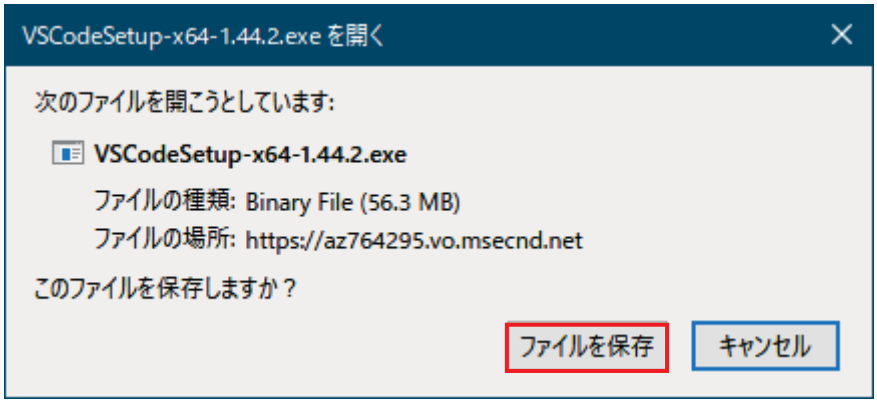
【ステップ２】Windows パソコン側に Visual Studio Code をインストール

Visual Studio Code（以下，VS Code）は，プログラムを作成するエディタです。拡張機能を追加することで，ファイルやフォルダ管理，プログラムの文法チェック，EV3 との通信等ができるようになります。

Web ブラウザで
https://code.visualstudio.com/download
を開きます。
「Windows」の下にある「System Installer」の「64 bit」を左クリックします。

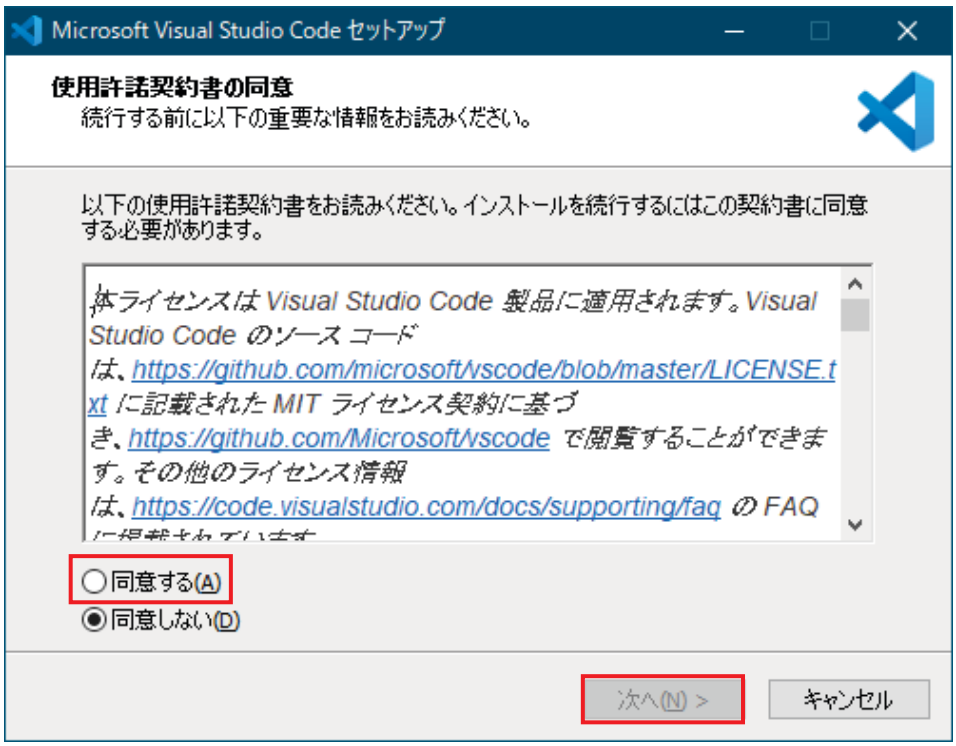


「ファイルを保存」を左クリックして，「VSCodeSteup-x64-1.44.2.exe」をパソコン側にダウンロードします。

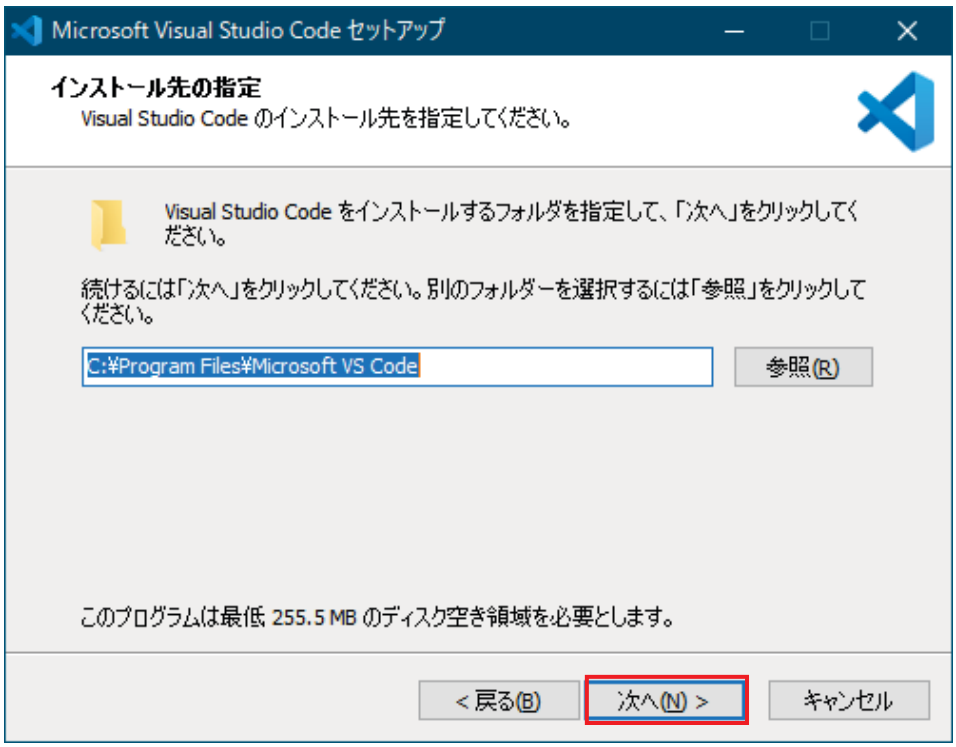


「VSCodeSteup-x64-1.44.2.exe」(Windows において，「ファイル名拡張子」を表示するようにチェックを入れておくと，.exe が表示されます。）というファイル名がダウンロードされます。

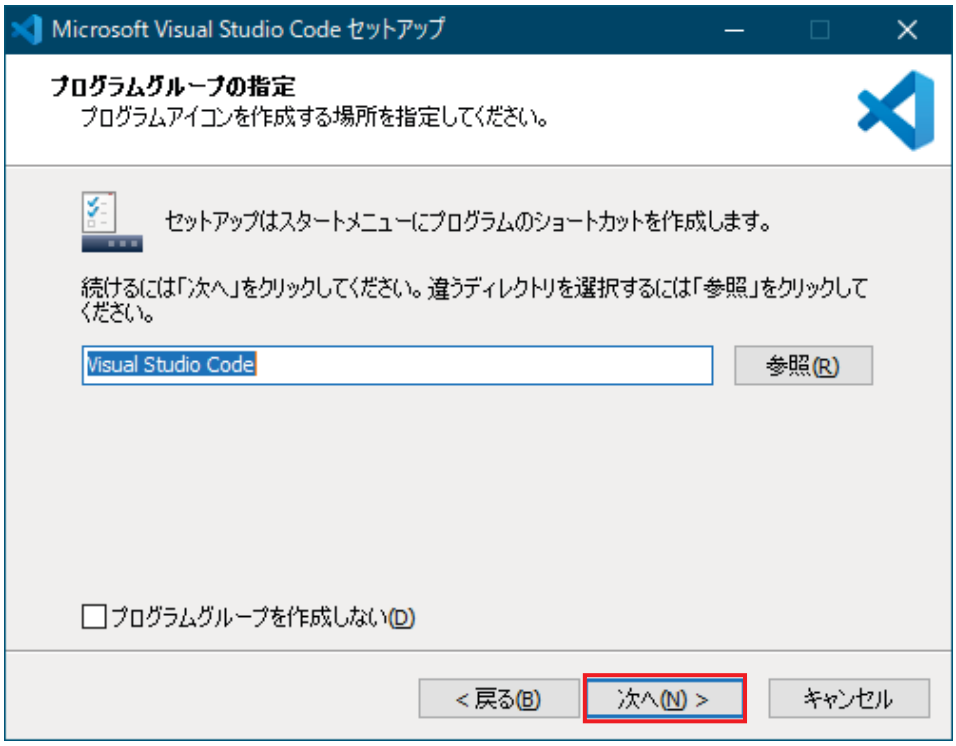
「VSCodeSteup-x64-1.44.2.exe」を右クリックして，「管理者として実行」を左クリックします。
「同意する」を左クリックして，「次へ」ボタンが有効になったら，左クリックします。



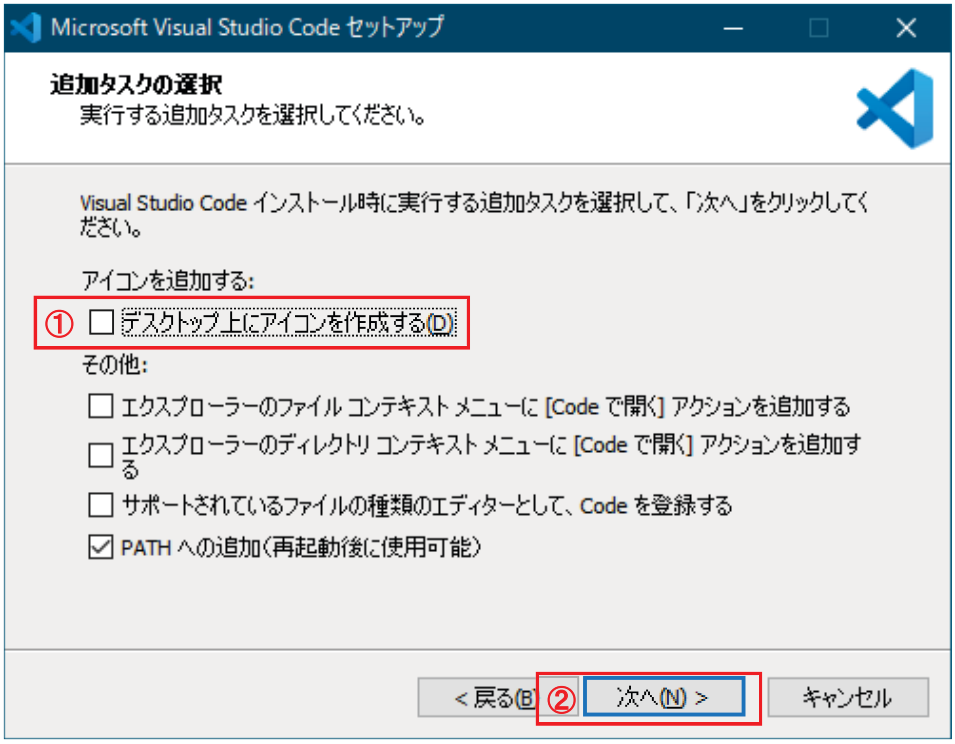
「次へ」を左クリックします。



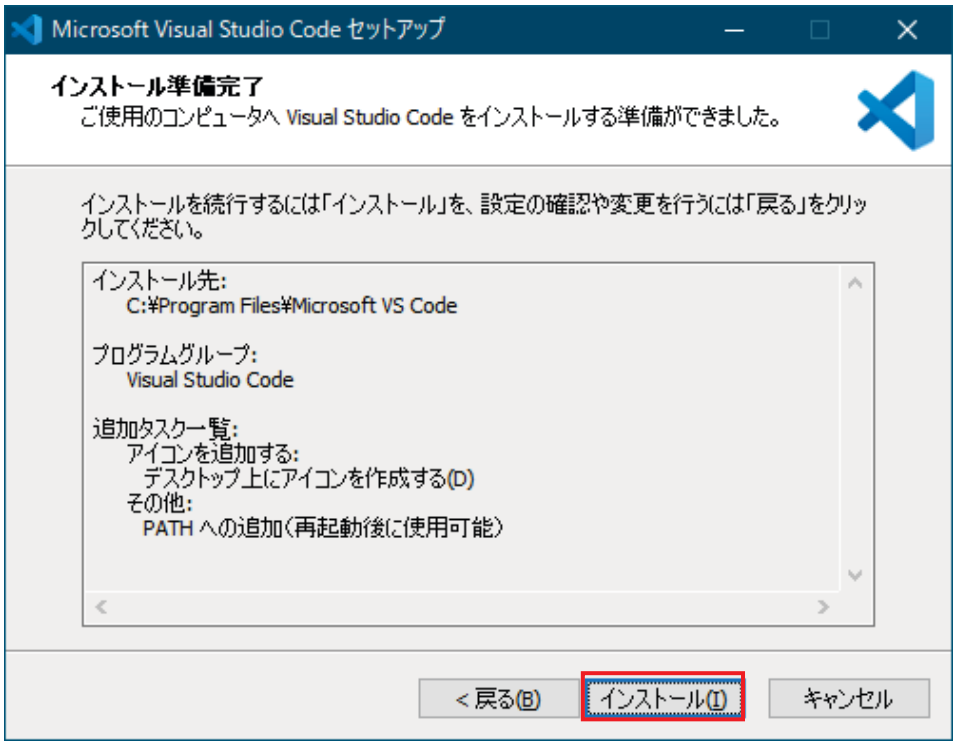
「次へ」を左クリックします。



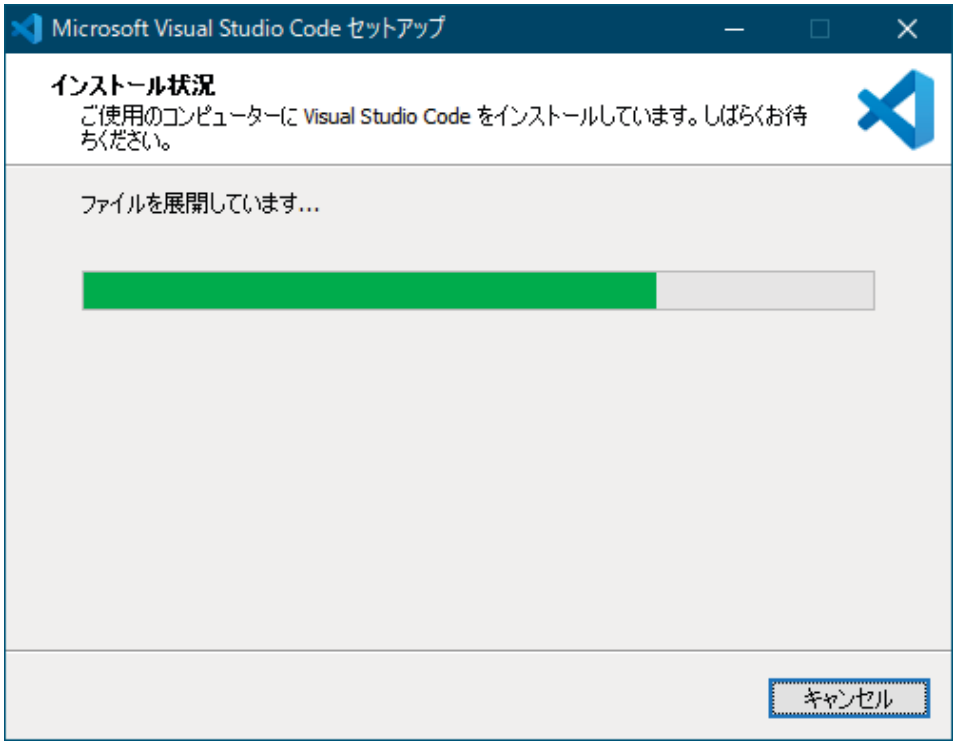
①「デスクトップ上にアイコンを作成する」の□にチェックを入れます。
②「次へ」を左クリックします。



「インストール」を左クリックします。



インストールが始まります。



①「Visual Studio Code を実行する」の□のチェックを外します。

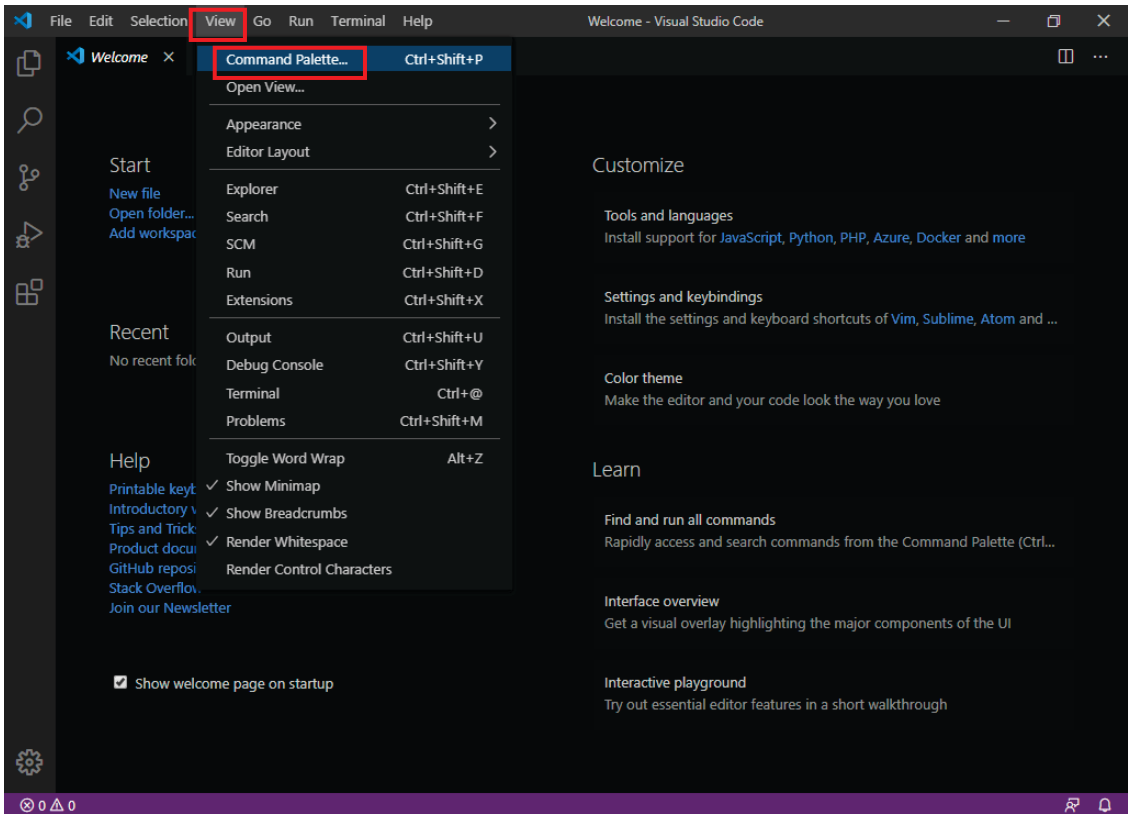
②「完了」を左クリックします。

Windows を再起動し，サインインした後，デスクトップに，VS Code のアイコンが追加されていることを確認します。

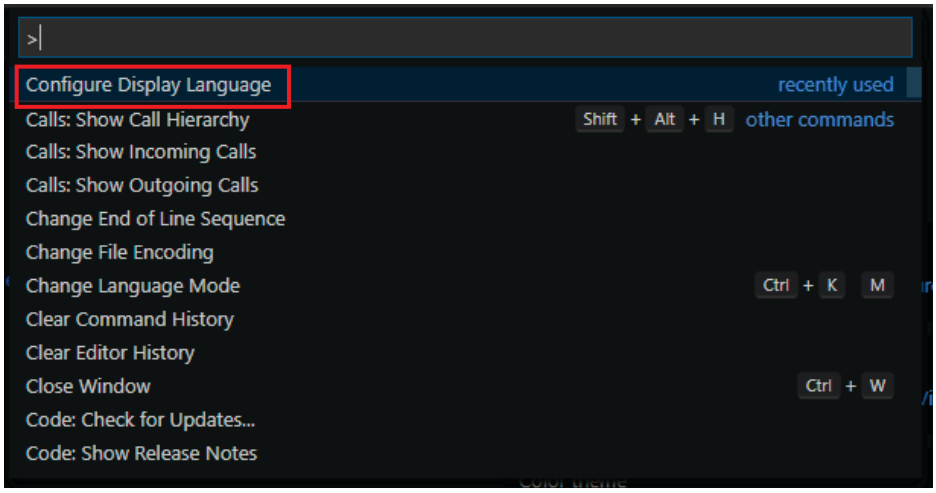
【ステップ３】Visual Studio Code の設定

Windows パソコンを起動し，個々のアカウントでサインインします。
Visual Studio Code (VS Code)のアイコンをダブルクリックして起動します。

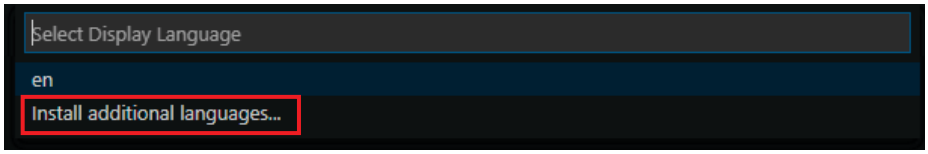
まず，VS Code の英語表示を日本語表示に切り替えます。
「View」メニューを左クリックし，「Command Palette」を左クリックします。



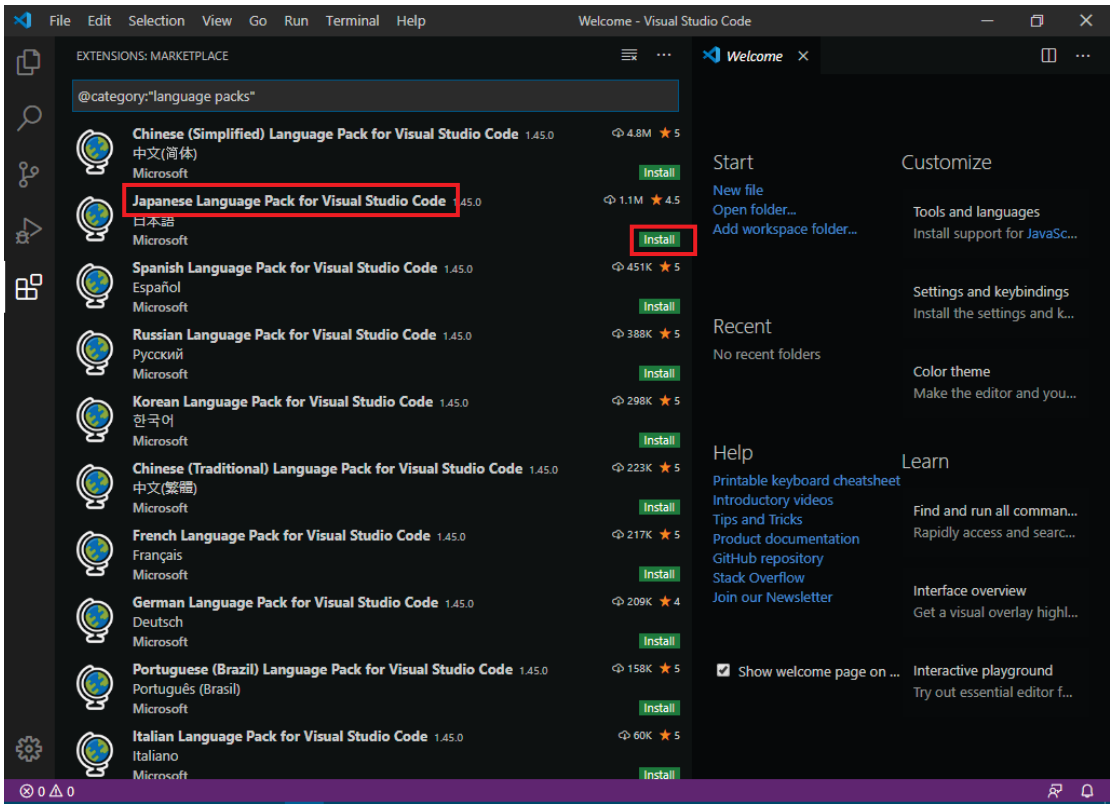
表示された枠の「>」の次から，「Configure Display Language」と入力するか，または，下に候補が表示されたら，「Configure Display Language」を左クリックします。



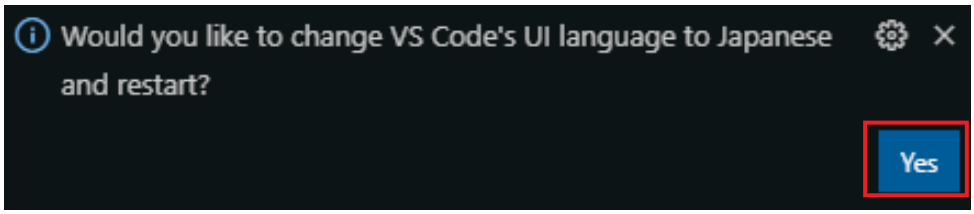
「Install additional languages」を左クリックします。



「Japanese Language Pack for Visual Studio Code」の「Install」を左クリックします。



VS Code 画面の右下隅に表示された「Yes」を左クリックします。

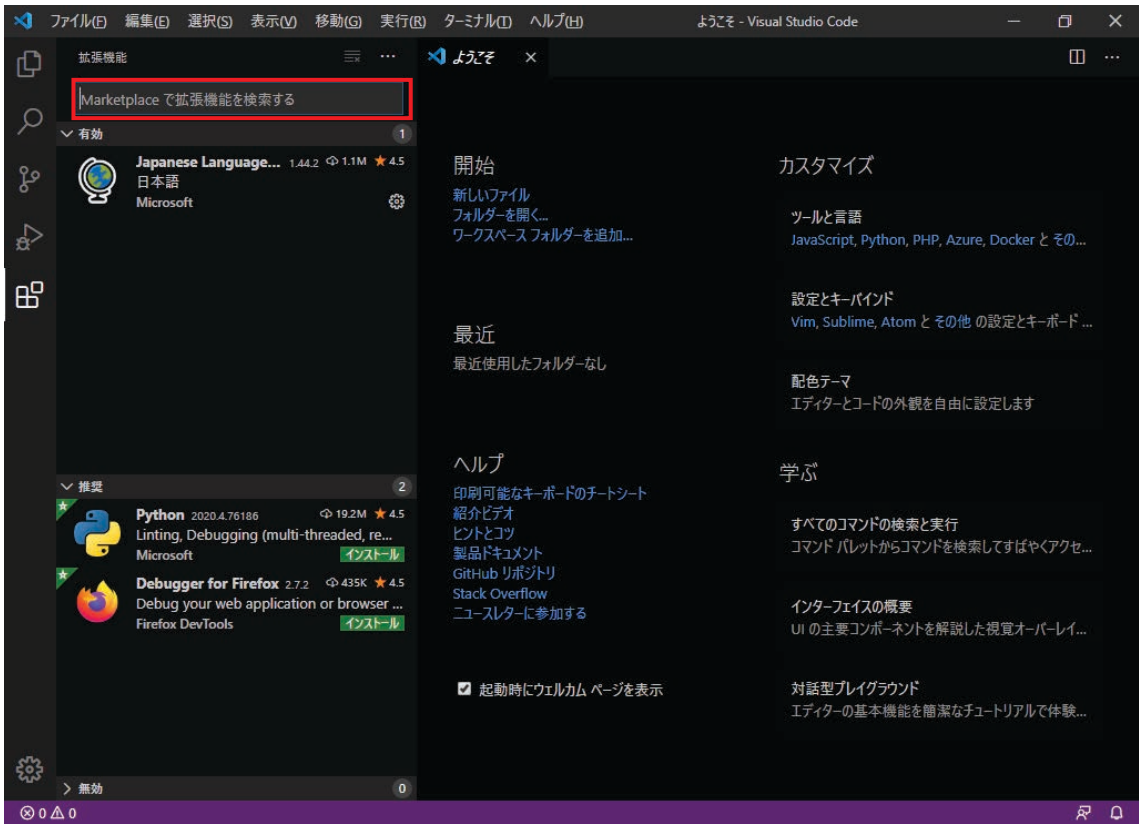


VS Code が自動的に再起動され，メニューが日本語に変わります。

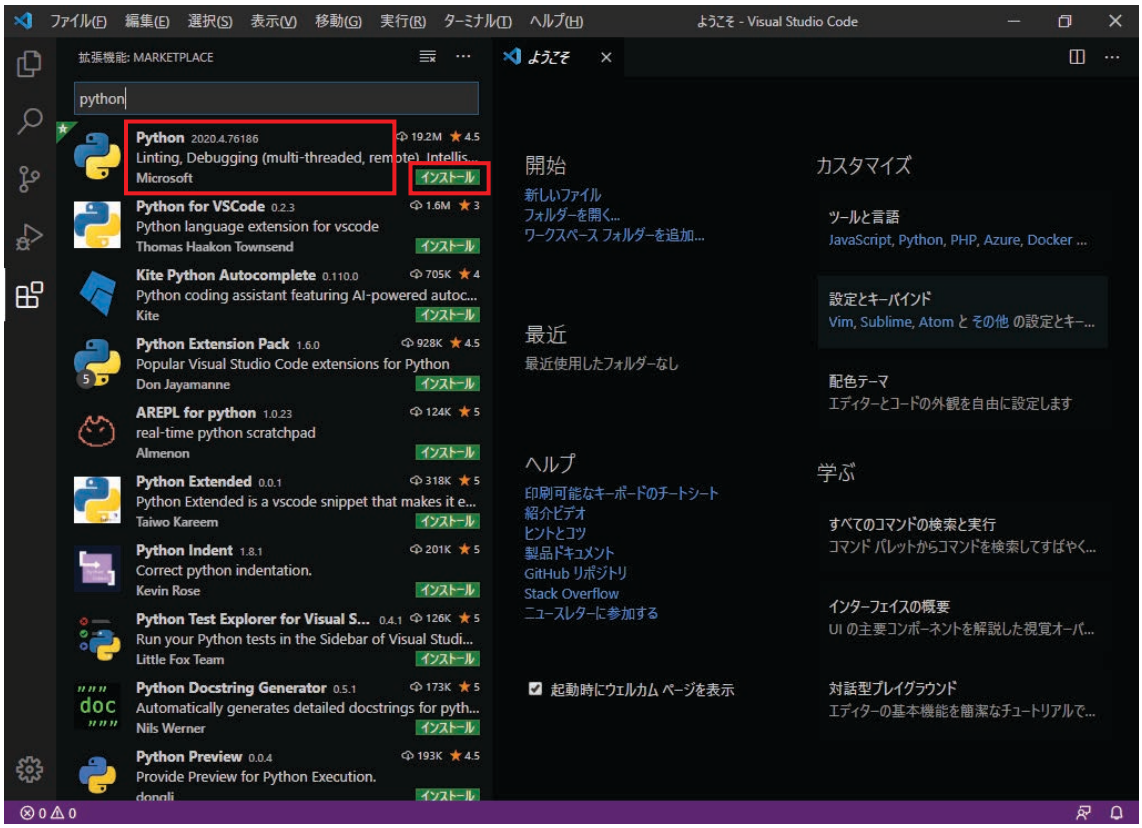
次に，VS Code に拡張機能をインストールしていきます。

拡張機能アイコン  を左クリックします。

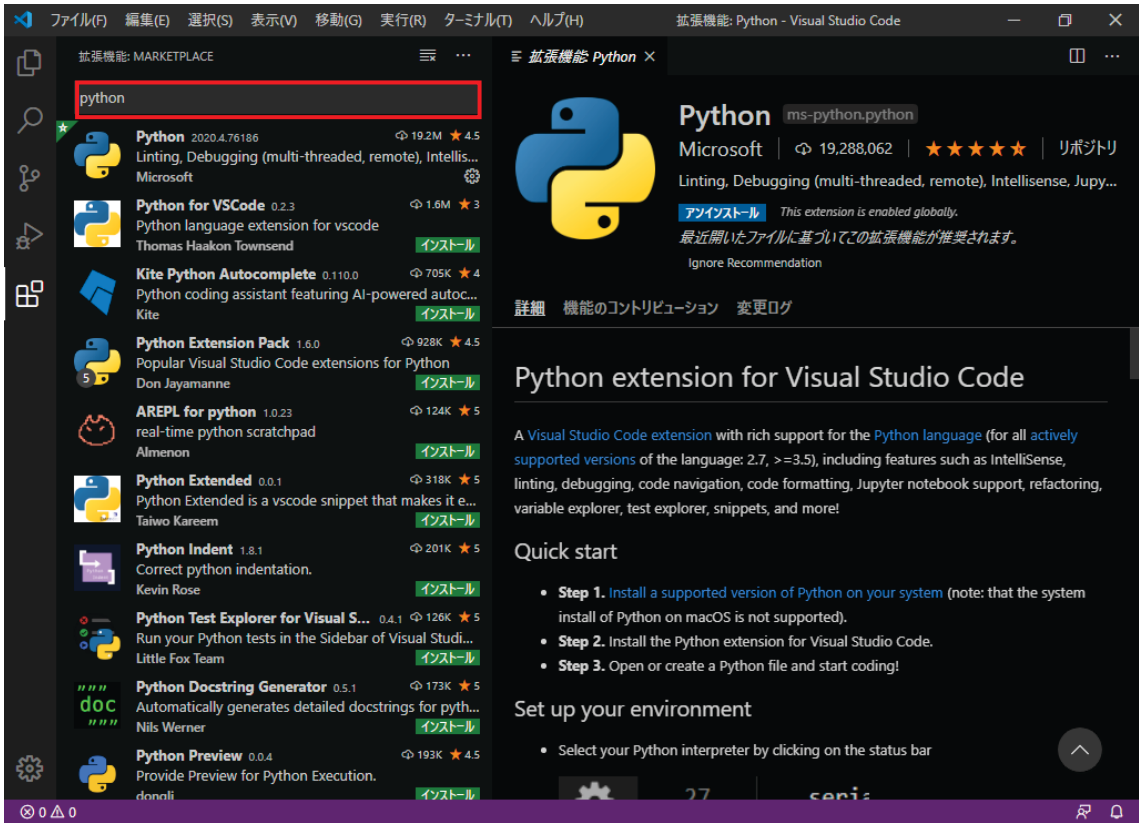
「拡張機能」の下にある枠に「python」と入力します。



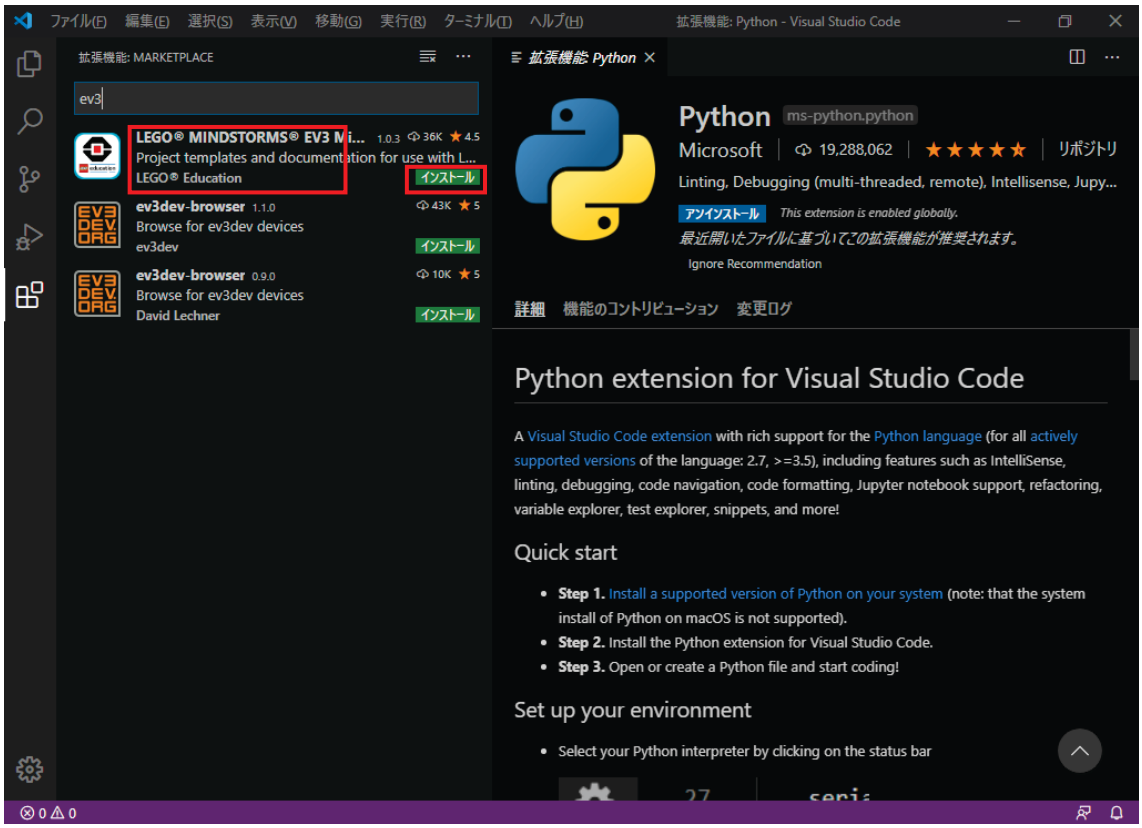
「Python (Microsoft)」にある「インストール」を左クリックします。



インストールが完了すると，以下の画面が表示されます。
「拡張機能」の下にある枠に「ev3」と入力します。



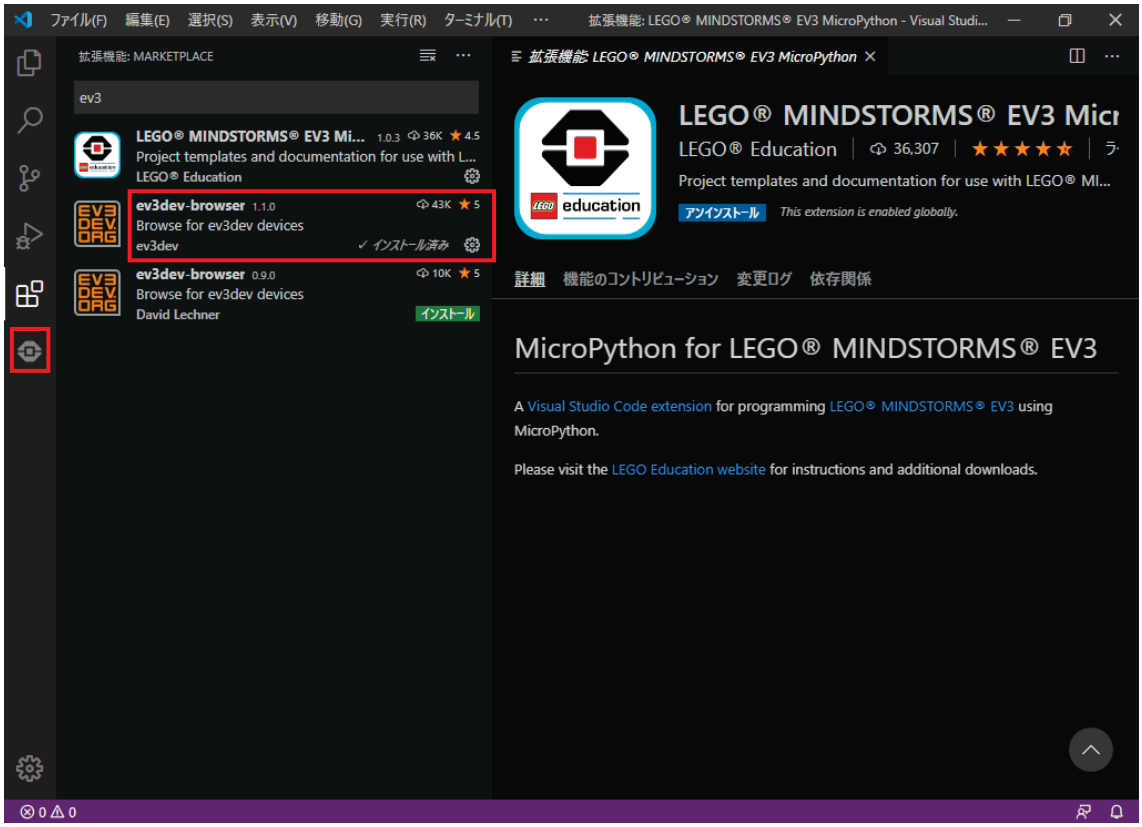
「LEGO MINDSTORMS EV3」にある「インストール」を左クリックします。



インストールが完了すると，以下の画面が表示されます。

表示画面の左側の「アクティビティバー」と呼ばれる領域に，EV3 用のアイコン  が追加されます。

「ev3-dev-browser」も「インストール済み」になります。



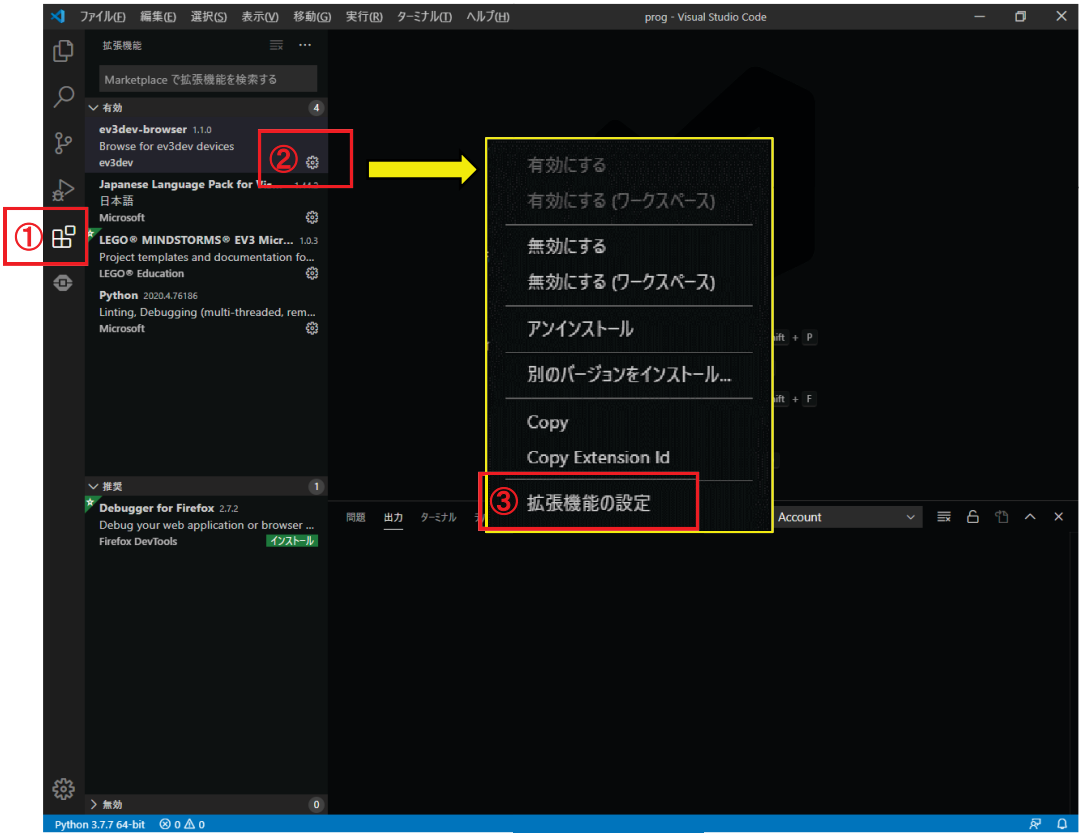
なお，「アクティビティバー」に表示するアイコンは，「アクティビティバー」を右クリックして表示されるメニューに一覧が表示されるので，アイコンの表示・非表示を切り替えることができます。

Windows パソコンを EV3 に接続したときの継続時間を延長します。

①画面左側の拡張機能アイコン  を左クリックします。

②「ev3dev-browser」の歯車アイコン  を左クリックします。

③「拡張機能の設定」を左クリックします。

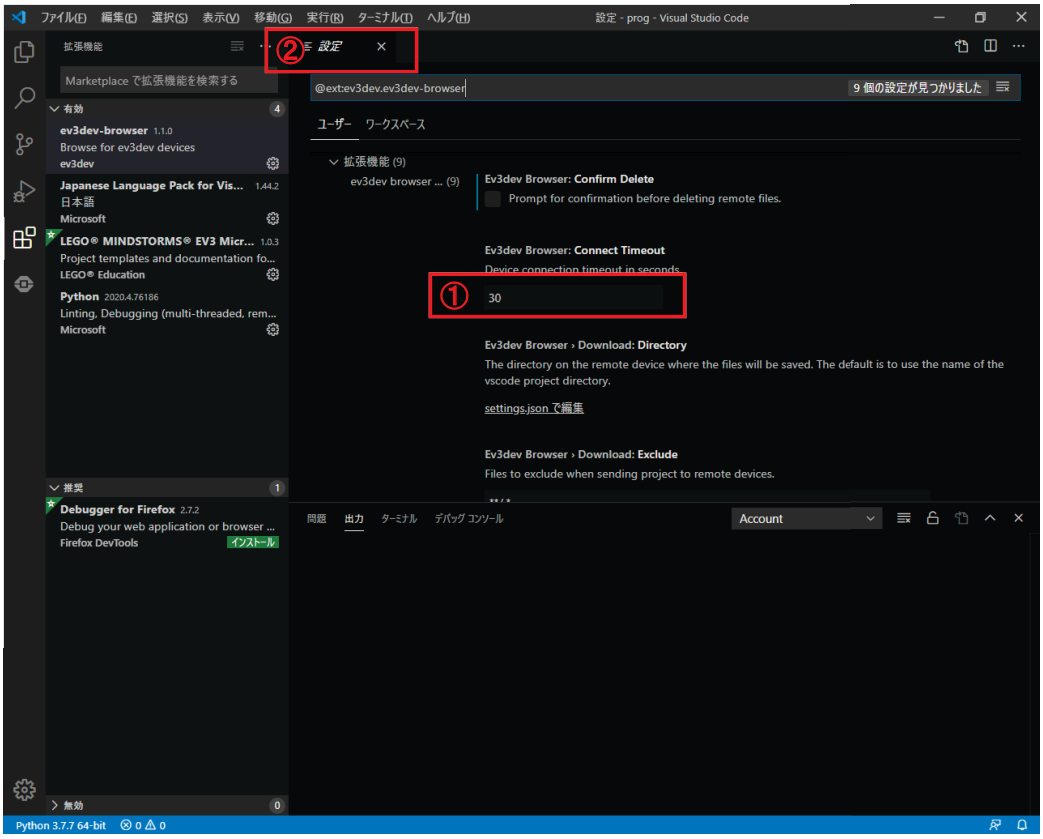


①「Ev3dev Browser: Connect Timeout」の下にある枠内の「30」を「120」に変更し，「エンターキー」を押します。

この場合，タイムアウト時間は，30 分から 120 分に変更されます。

※　表示されている単位は「秒」ですが，「分」の誤りと思われます。

②「設定」の右に表示されている「×」を左クリックします。



付録 B

ev3dev 用メモリーカードの作成手順

EV3 ブロックは，ev3dev と呼ばれる Linux OS のソフトウェアを書き込んだメモリーカードを装着して利用します。Windows パソコンを使って ev3dev 用メモリーカードを作る手順について説明します。

１．メモリーカードの準備

メモリーカードは，4GB～32GB の microSDHC を利用できます(microSDXC は利用できません。)。ev3dev と呼ばれるファームウェア(Linux OS の一種)の場合，16GB または 32GB を利用します。

なお，EV3 のメモリーカードを装着するスロットは，精密にできているため，頻繁にメモリーカードを抜き差しするような使い方はしないようにします。スロットからメモリーカードを抜きやすくすること，及び，メモリーカードを識別できるように，テプラ等のテープに識別情報を表記したものを貼付しておくと利用しやすくなります。

また，microSDHC を書き込める「メモリーカード／リーダ」も必要です。

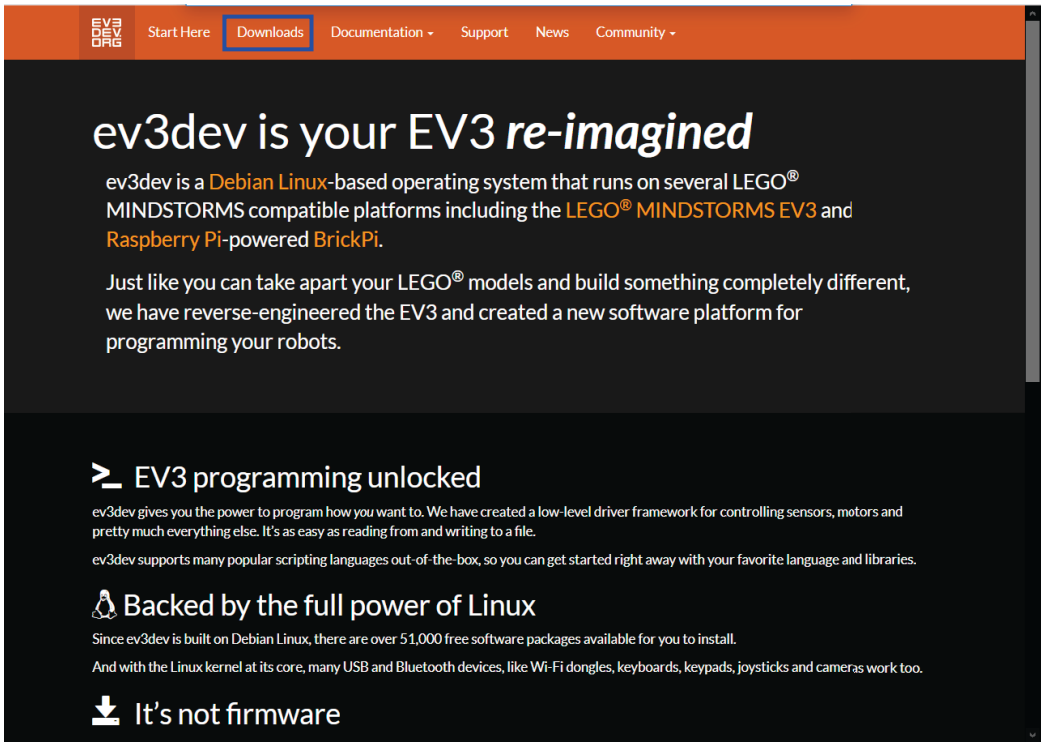
２．ev3dev のイメージ・ファイルのダウンロード

Web ブラウザで

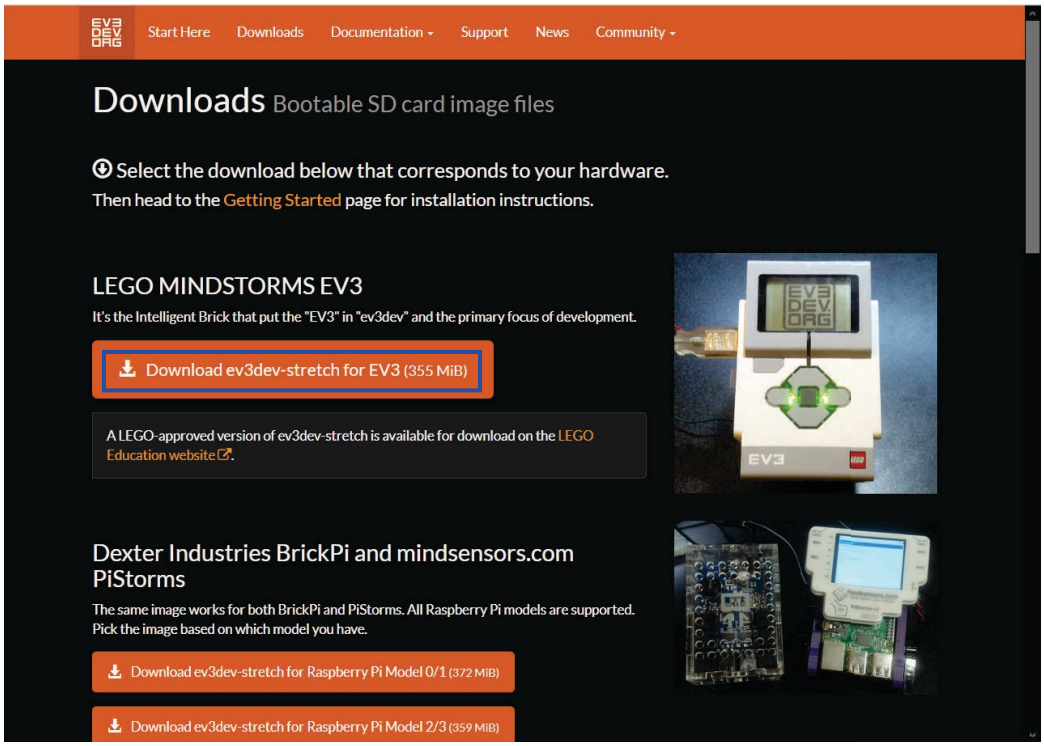
https://www.ev3dev.org/

を開きます。

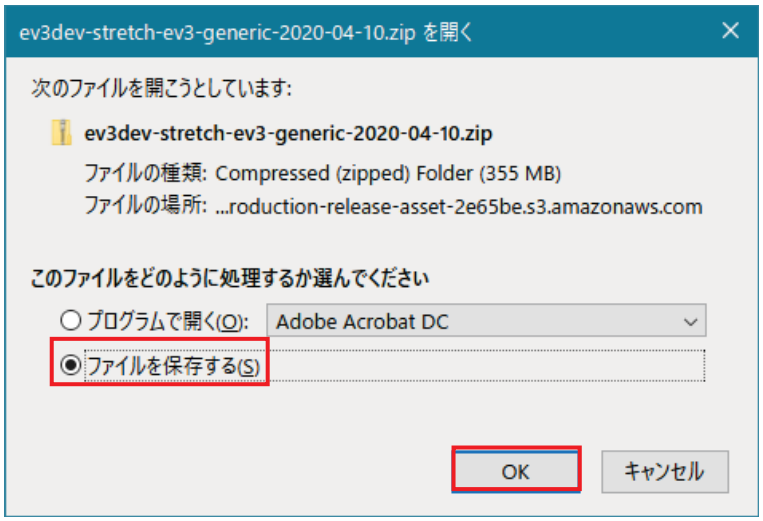
「Downloads」を左クリックします。



「Download ev3dev-stretch for EV3 (355MiB)」を左クリックします。



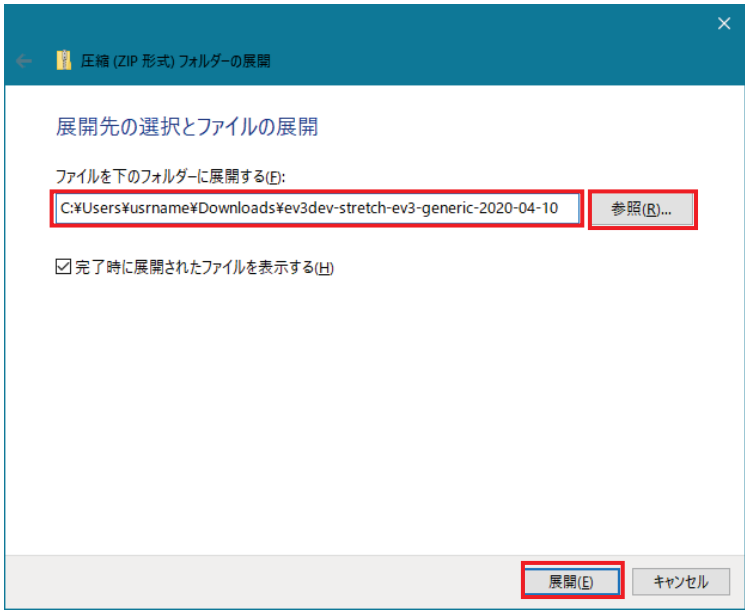
「ファイルを保存する」を選択して，「OK」を左クリックして，ダウンロードします。



「ev3dev-stretch-ev3-generic-2020-04-10.zip」を右クリックして,「すべて展開」を右クリックします。

「ファイルを下のフォルダーに展開する」の枠内に表示されているフォルダを変更したいときは,「参照」を右クリックして，所定のフォルダを選択します。

「展開」を左クリックします。



「ev3dev-stretch-ev3-generic-2020-04-10」という名前のフォルダが作成され，その中に「ev3dev-stretch-ev3-generic-2020-04-10.img」という約 1.7GB のイメージ・ファイルが展開されます。

３．イメージ・ファイルのメモリーカードへの書き込み

まず，イメージ・ファイルをメモリーカードに書き込むためのソフトウェア「Win32DiskImager」をダウンロードします。

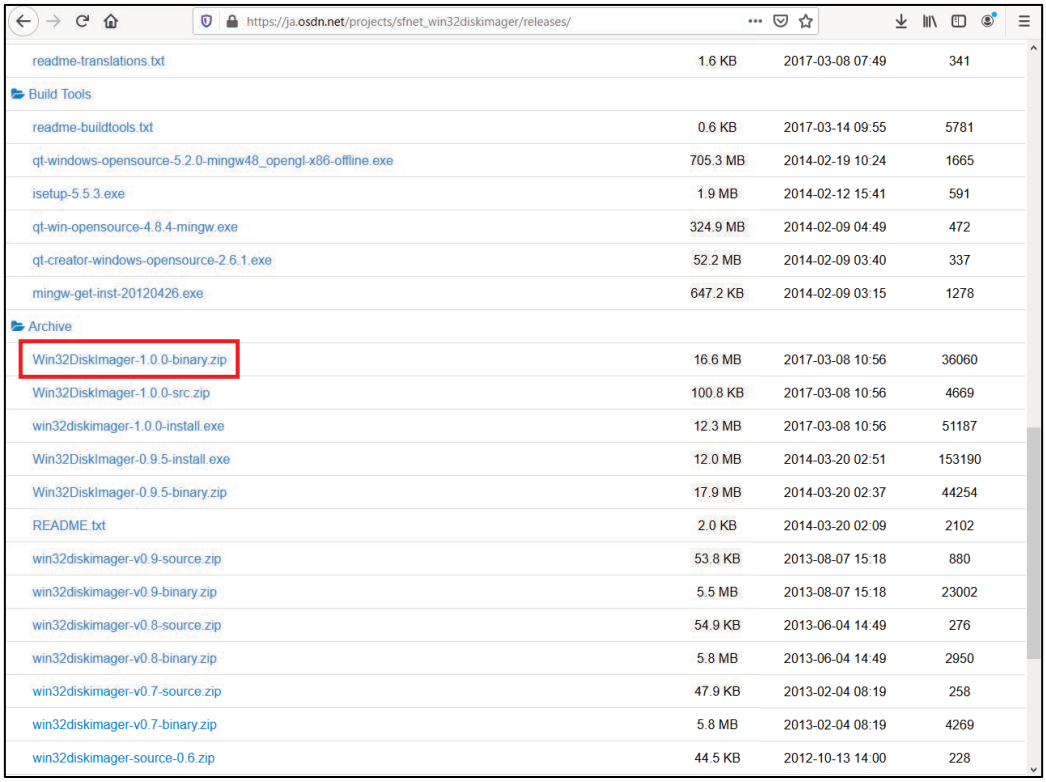
Web ブラウザで

https://ja.osdn.net/projects/sfnet_win32diskimager/

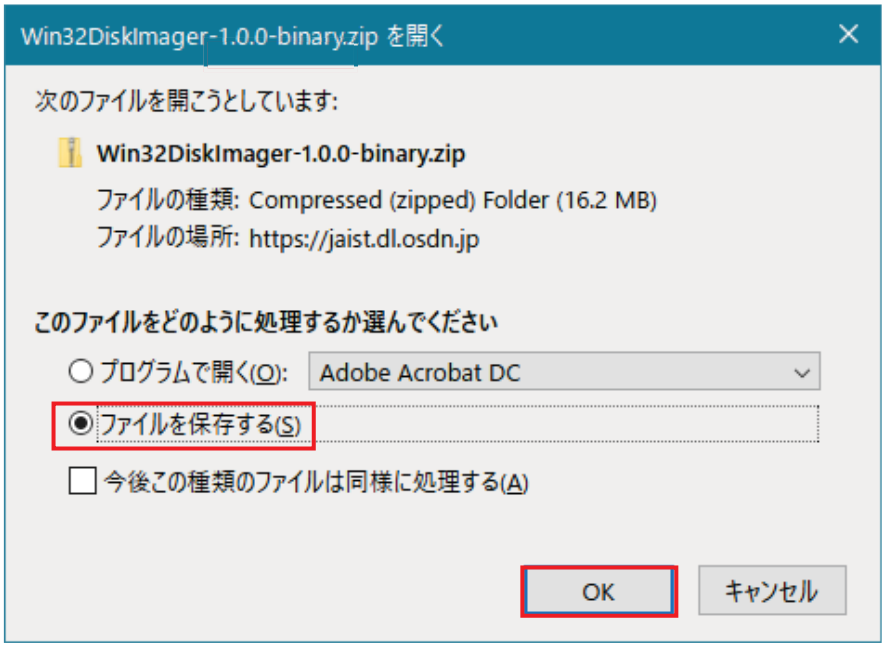
を開きます。

「ダウンロードファイル一覧」を左クリックします。

下にスクロールして「Win32DiskImager-1.0.0-binary.zip」を左クリックします。



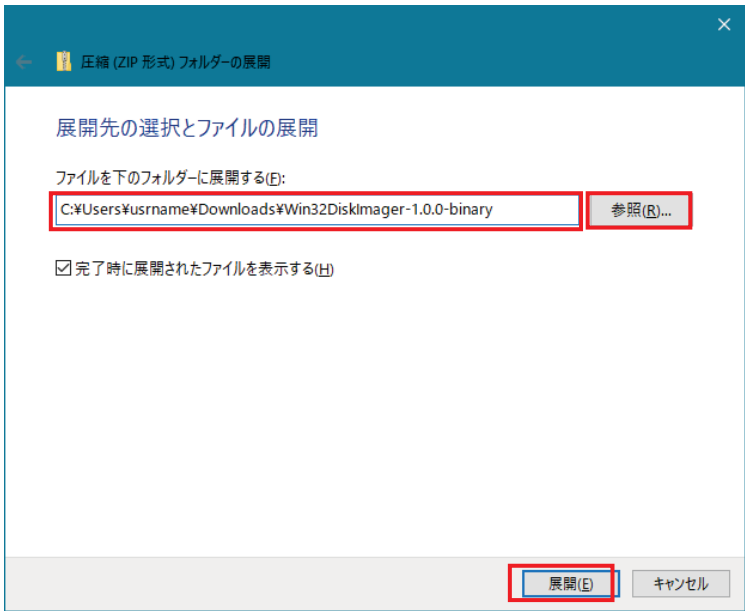
「ファイルを保存する」を選択して，「OK」を左クリックして，ダウンロードします。



「Win32DiskImager-1.0.0-binary.zip」を右クリックして，「すべて展開」を右クリックします。

「ファイルを下のフォルダーに展開する」の枠内に表示されているフォルダを変更したいときは，「参照」を右クリックして，所定のフォルダを選択します。

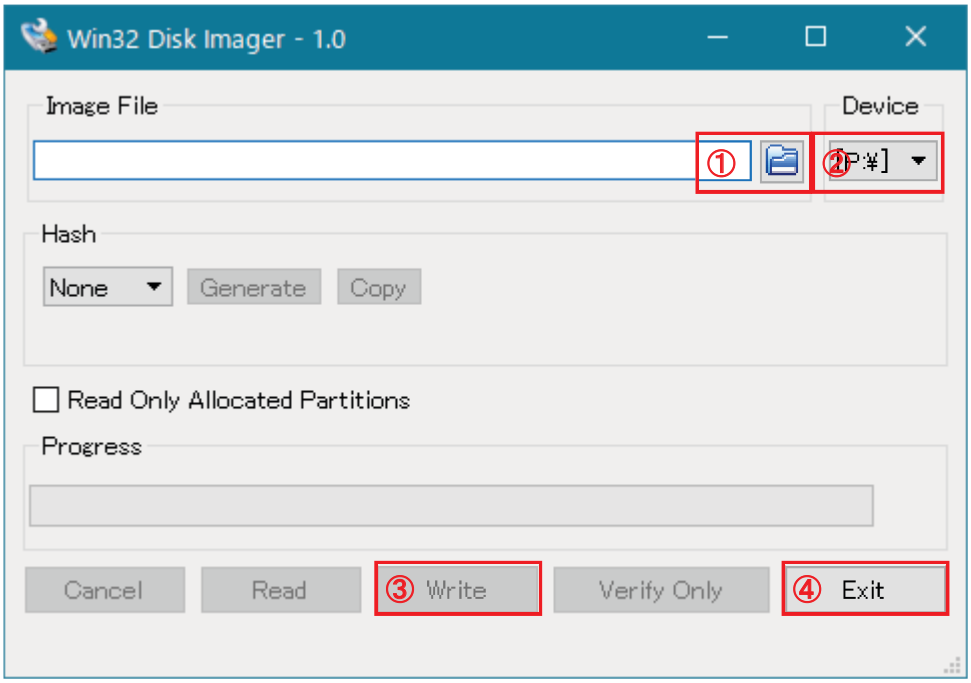
「展開」を左クリックします。



「Win32DiskImager-1.0.0-binary」という名前のフォルダが作成され，その中に「Win32DiskImager.exe」という実行型式のファイル等一式が展開されます。

Windows パソコンに microSDHC を読み書きできる「メモリーカードリーダー」を USB ケーブルで接続し，microSDHC メモリーカードを挿入します。

「Win32DiskImager.exe」をダブルクリックして起動します。管理者権限の無いアカウントでサインインしている場合は，管理者権限のパスワードを入力します。
①イメージ・ファイル「ev3dev-stretch-ev3-generic-2020-04-10.img」と選択します。
②microSDHC のドライブを選択します。
③「Write」を左クリックします。
※書き込みするとメモリーカードの内容はすべて上書きされます。
④「Exit」を左クリックします。

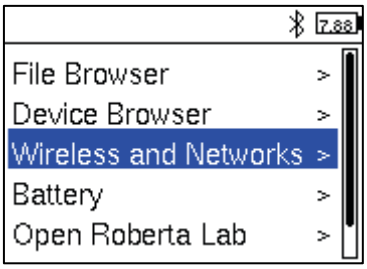


「メモリーカードリーダー」から microSDHC メモリーカードを取り出します。
取り出した microSDHC メモリーカードに，識別文字等を記載したテープを貼付しておきます。

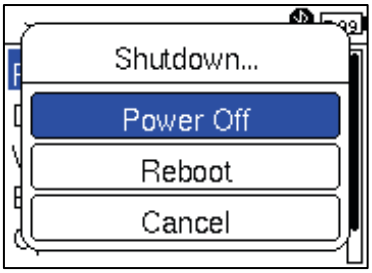
４．メモリーカードの動作確認

EV3 本体の電源がオフの状態で，イメージ・ファイルを書き込んだメモリーカードを本体のスロットに挿入して，中央のボタンを押し，ev3dev を起動します。

小さな文字が表示され，数分程度待つと，以下のような起動画面が EV3 本体に表示されれば，正常に動作しています。



電源を切るときは，EV3 ブロックの「戻るボタン」を押し，「上ボタン」と「下ボタン」を使って「Power Off」を選択し，「中央ボタン」を押します。



小さな文字が液晶ディスプレイに表示され，数秒後に自動的に電源が切れます。

付録 C

パソコンと EV3 ブロックを接続する方法

パソコンを ev3dev を起動した EV3 ブロックに接続する方法は，USB ケーブルによる有線接続，Bluetooth による無線接続，無線 LAN 接続の３種類があります。通常は，いずれか一つの接続方法で利用しますが，同時に複数種類の接続方法でパソコンと EV3 ブロックを接続することもできます。

表Ｃ－１に各接続方法の特徴をまとめます。環境に応じて適切な接続方法を選択します。いずれの接続方法であっても，Visual Studio Code を使って ev3dev を利用する手順は同じです。

ev3dev を起動した EV3 ブロックのメニューは，「上ボタン」，「下ボタン」，「左ボタン」，「右ボタン」（以下，「上下左右ボタン」）で操作し，「中央ボタン」で選択します。戻るときは，「戻るボタン」を押します。

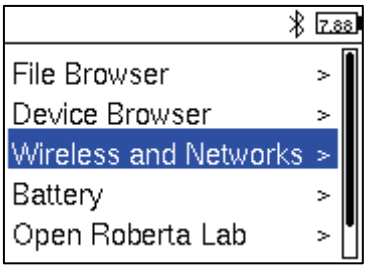
表Ｃ－１　パソコンと EV3 ブロックの接続方法

接続方法	長所	短所	注意点
① USB ケーブル	・接続している EV3 ブロックを USB ケーブル確認できる。 ・パソコン側がインターネット接続されていると「ブリッジ接続」設定することで，EV3 ブロック側からインターネット接続できる。	・USB ケーブルを装着したままロボットを動作させにくい。 ・USB ケーブルの脱着に時間がかかる。 ・EV3 ブロックの USB コネクタ(Type A)を劣化させる。	・EV3 ブロックの USB コネクタを劣化させないようにＬ型等の延長用コネクタを装着しておくことが望ましい。
② Bluetooth	・約 10m の見通し距離までであれば，パソコンから離れていてもプログラムを更新したり実行したりできる。 ・パソコンと一度ペアリングすることで，EV3 ブロックを識別できる。	・約 10m の見通し距離までしか通信できないため，広い教室では対応できない場合がある。 ・通信が不安定になり切れてしまうことがある。 ・パソコンとのペアリング設定が解除されると，再度ペアリングする手間がかかる。	・パソコン側に Bluetooth 機能を備えている必要がある。 ・EV3 ブロックを識別するための名称を ev3dev に設定する必要がある。
③ 無線 LAN	・約 50m の見通し距離までであれば，パソコンから離れていてもプログラムを更新したり実行したりできる。一般的な教室であれば，無線でほぼ対応できる。 ・通信状態が保持されやすい。 ・EV3 ブロック側からインターネットに接続できる。	・EV3 ブロックに無線 LAN 用 USB ドングルを追加する必要がある。 ・約 50m 程度の見通し距離までしか通信できない。 ・SSID による認証であれば問題ないが，他の認証方式では対応できない場合がある。 ・アクセスポイントを設置しなければならない。	・パソコン側に無線 LAN 機能を備えている必要がある。 ・EV3 ブロックで利用できる無線 LAN 用 USB ドングルは ev3dev で識別可能なものでなければならない。 ・EV3 ブロックを識別するため，固定した IP アドレスを ev3dev に設定する必要がある。

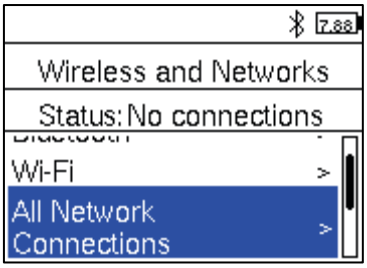
①　USB ケーブルによる接続手順

ev3dev を起動し，パソコン本体と EV3 ブロックを USB ケーブルで接続します。

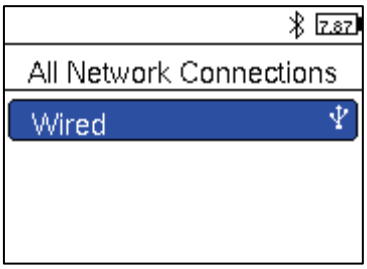
ev3dev の起動画面において，「Wireless and Networks」を選択します。



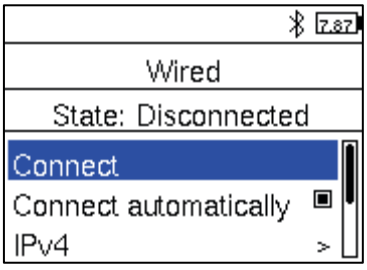
「All Network Connections」を選択します。



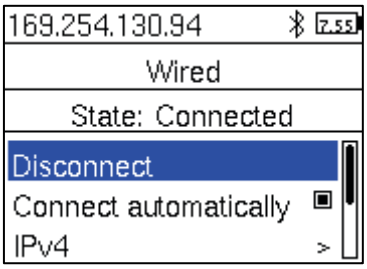
「Wired」を選択します。



「Connect」を選択します。ここでしばらく待ちます。



接続が完了すると，Windows 側で自動割り当てされた IP アドレスが最上段に表示され，「Connected」と表示されます。



接続を終了したいときは，「Disconnect」を選択します。

なお，Windows 側で EV3 ブロックの IP アドレスを自動割り当てすると，接続までに時間がやや要することや，IP アドレスが固定できないという問題があります。この場合，以下の手順で Windows 側のネットワーク設定と EV3 側のネットワーク設定をすることで，IP アドレスを固定することができます。

Windows 側のネットワーク設定は，管理者権限を持つアカウントでサインインして行います。

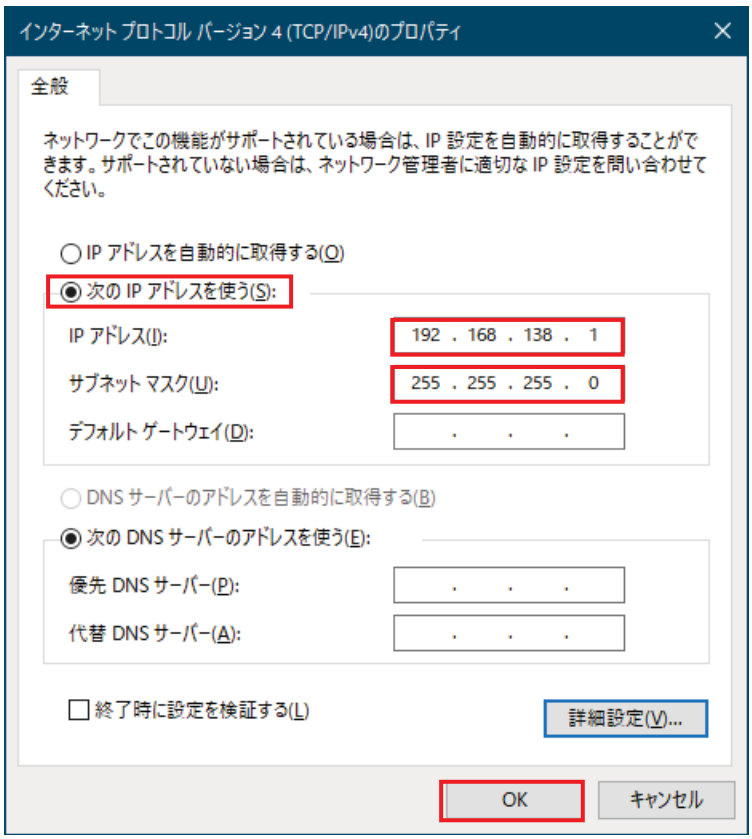
スタートメニューを左クリックし，設定アイコン（歯車）を左クリックし，「ネットワークとインターネット」を左クリックします。

ネットワーク設定の下にある「アダプタのオプション」を左クリックします。

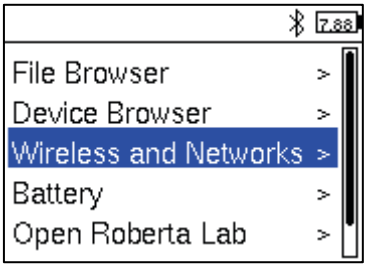
「Remote NDIS Compatible Device」を右クリックし，「プロパティ」を左クリックします。

「インターネット プロトコル バージョン 4 (TCP/IPv4)」を左クリックして選択し，「プロパティ」を左クリックします。

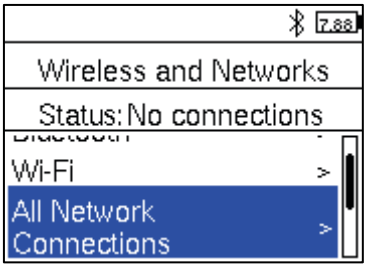
「次の IP アドレスを使う」を選択し，
IP アドレスを「192.168.138.1」，
サブネットマスクを「255.255.255.0」
に設定し，「OK」を左クリックします。
なお，IP アドレスが他の設定と重複する場合，上記のアドレスから変更します。



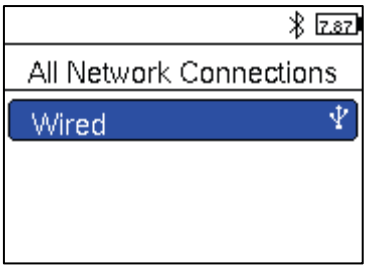
ev3dev の起動画面において，「Wireless and Networks」を選択します。



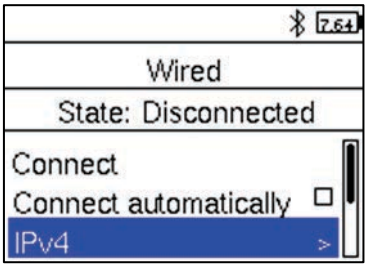
「All Network Connections」を選択します。



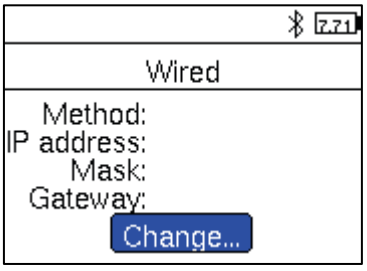
「Wired」を選択します。



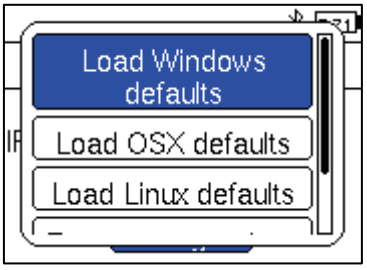
「IPv4」を選択します。



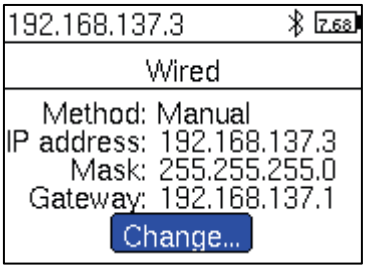
「Change」を選択します。



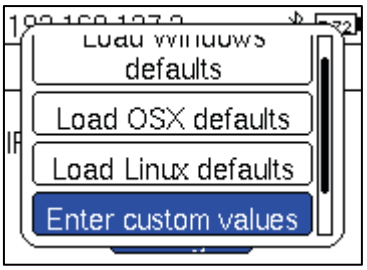
「Load Windows defaults」を選択します。



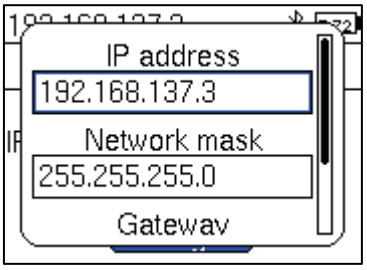
「192.168.137.3」に設定された IP アドレスを「192.168.138.3」に変更するため,「Change」を選択します。



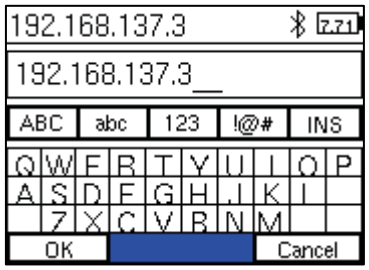
「Enter custom values」を選択します。



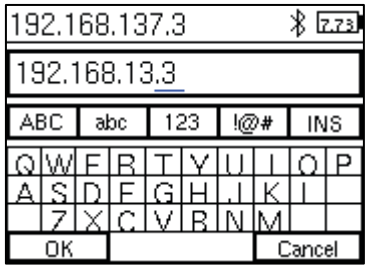
「192.168.137.3」を選択します。



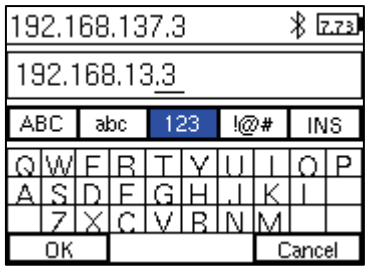
EV3 ブロックの「上下左右ボタン」を使って「192.168.137.3」の枠内を選択し，カーソルを「192.168.137__.3」の位置に移動させます。



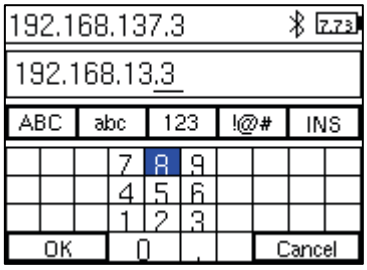
EV3 ブロックの「戻るボタン」を使って，「7」を消します。

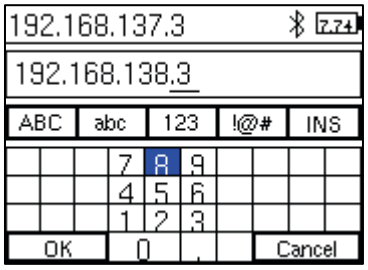


数字を入力するため，EV3 ブロックの「上下左右ボタン」を使って「123」を選択します。

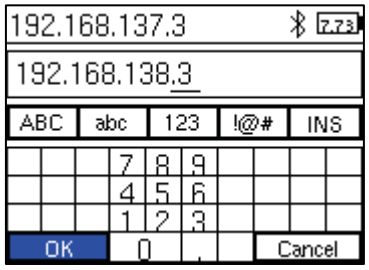


「8」を選択し，「中央ボタン」を押して，入力します。

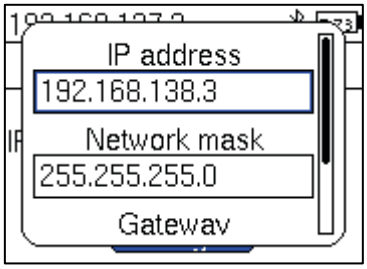


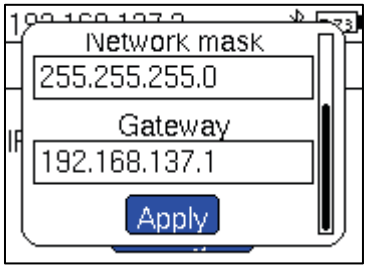


「192.168.138.3」に変更できたことを確認し，「OK」を選択します。

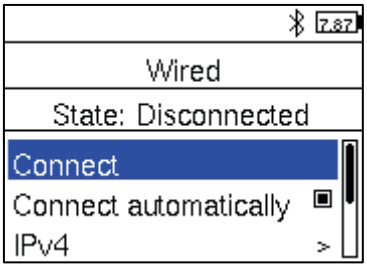


画面をスクロールし「Apply」を選択します。

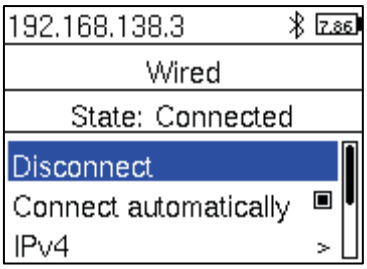




「戻るボタン」を押し，「Connect」を選択します。

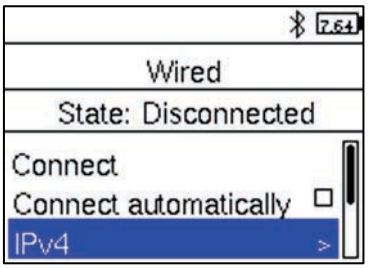


接続が完了すると，設定した IP アドレスが最上段に表示され，「Connected」と表示されます。

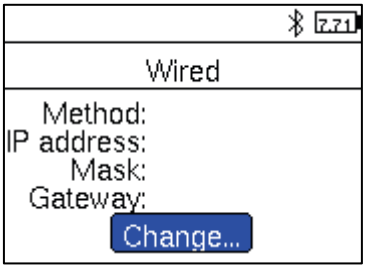


接続を終了したいときは，「Disconnect」を選択します。

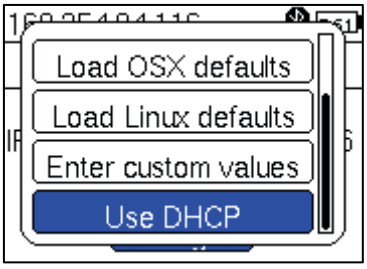
なお，IP アドレスの自動設定に戻したい場合は，以下の手順で設定します。
「IPv4」を選択します。



「Change」を選択します。



「Change」を選択した後,「Use DHCP」を選択します。



②　Bluetooth による接続手順

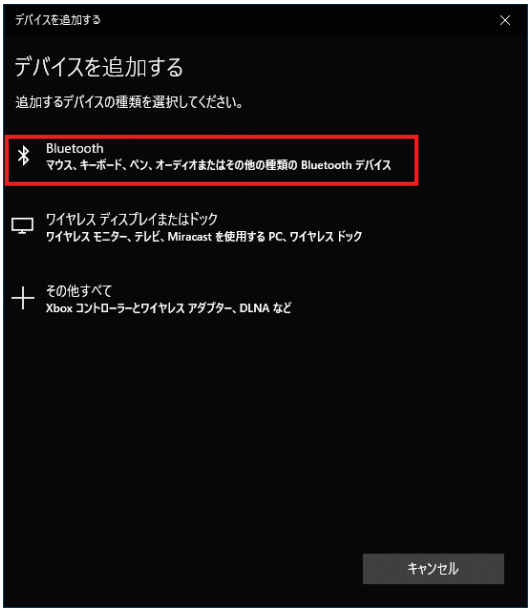
ev3dev を起動します。次に，Windows を起動します。

Bluetooth によるペアリングを行うため，スタートメニューの設定アイコン（歯車）を左クリックします。「デバイス」を左クリックします。

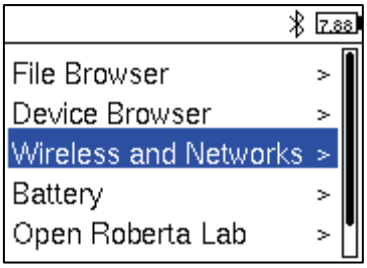
※ペアリング完了している場合は，以下の作業は省略できます。

①「Bluetooth」を「オン」にします。
②「”○○”として発見可能になりました。」と表示されるので，Windows 側の Bluetooth 名を記録しておきます。※この例では「DESKTOP-STG512P」となっています。
③「Bluetooth またはその他のデバイスを追加する」を左クリックします。

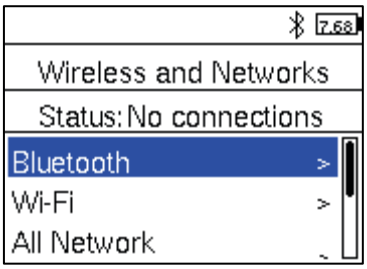
「Bluetooth」を左クリックします。



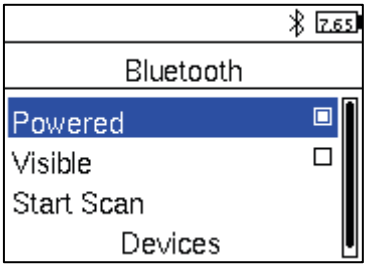
ev3dev の起動画面において，「Wireless and Networks」を選択します。



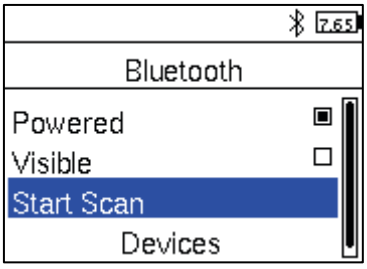
「Bluetooth」を選択します。



「Powered」を選択し，□の中に点を表示させます。

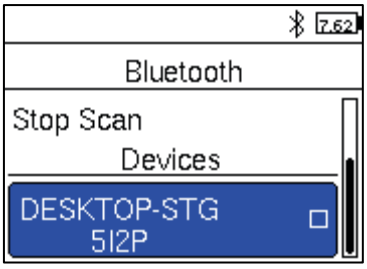


「Start Scan」を選択し，

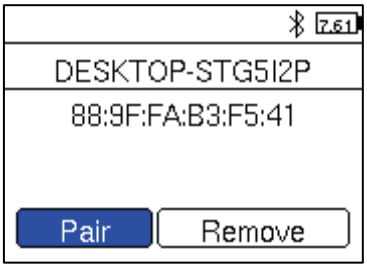


Windows 側の Bluetooth 名を選択します。

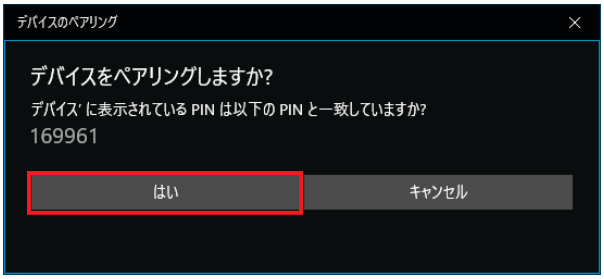
※この例では「DESKTOP-STG512P」となっています。



「Pair」を選択します。



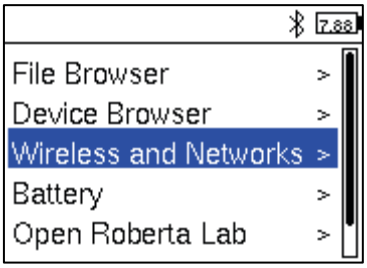
Windows 側でデバイスのペアリングが表示されるので，「はい」を左クリックします。



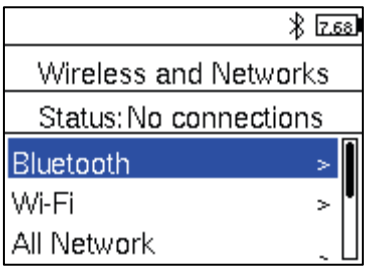
EV3 ブロック側で，「Accept」を選択します。

以上の操作で，Windows と EV3 ブロックのペアリングが完了しました。
一旦ペアリングすると，この情報は記録されます。

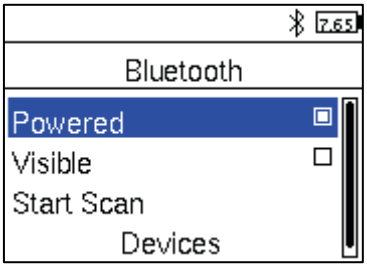
ペアリングが完了している場合は，以下の手順で接続します。
ev3dev の起動画面において，「Wireless and Networks」を選択します。



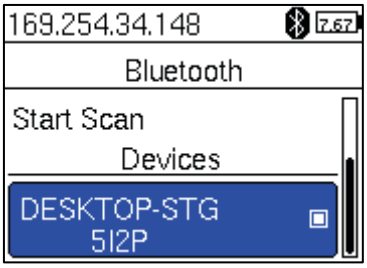
「Bluetooth」を選択します。



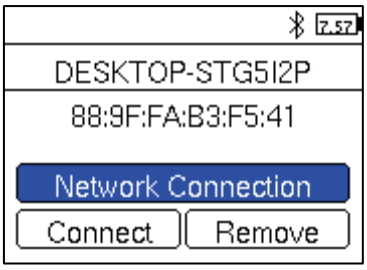
「Powered」を選択し，□の中に点を表示させます。



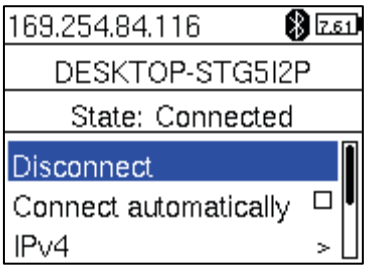
下にスクロールさせて Windows 側の Bluetooth 名を選択します。
※この例では「DESKTOP-STG512P」となっています。



「Network Connection」を選択します。ここでしばらく待ちます。



接続が完了すると，Windows 側で自動割り当てされた IP アドレスが最上段に表示され，「Connected」と表示されます。



接続を終了したいときは，「Disconnect」を選択します。

なお，Windows 側で EV3 ブロックの IP アドレスを自動割り当てすると，接続までに時間がやや要することや，IP アドレスが固定できないという問題があります。この場合，以下

の手順で Windows 側のネットワーク設定と EV3 ブロック側のネットワーク設定をすることで，IP アドレスを固定することができます。

Windows 側のネットワーク設定は，管理者権限を持つアカウントでサインインして行います。

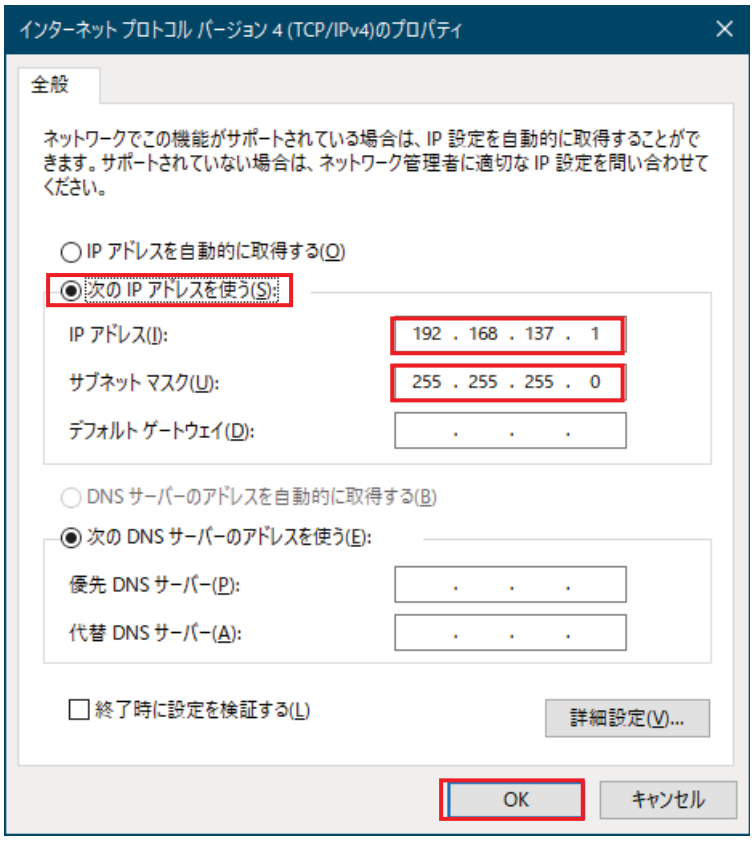
スタートメニューを左クリックし，設定アイコン（歯車）を左クリックし，「ネットワークとインターネット」を左クリックします。

ネットワーク設定の下にある「アダプタのオプション」を左クリックします。

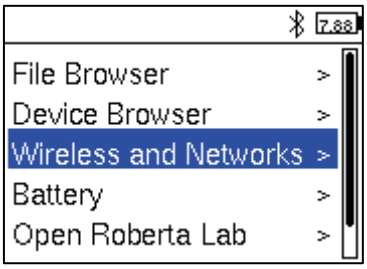
「Bluetooth ネットワーク接続 (Bluetooth Device (Personal Area Network)」を右クリックし，「プロパティ」を左クリックします。

「インターネット プロトコル バージョン 4 (TCP/IPv4)」を左クリックして選択し，「プロパティ」を左クリックします。

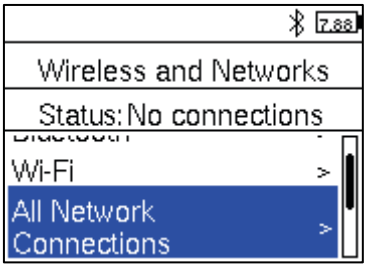
「次の IP アドレスを使う」を選択し，
IP アドレスを「192.168.137.1」，
サブネットマスクを「255.255.255.0」
に設定し，「OK」を左クリックします。
なお，IP アドレスが他の設定と重複する場合，上記のアドレスから変更します。



ev3dev の起動画面において，「Wireless and Networks」を選択します。

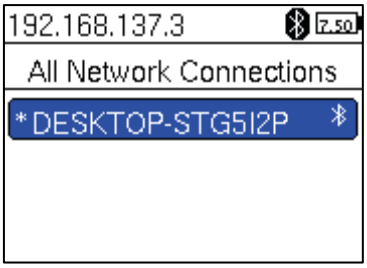


「All Network Connections」を選択します。

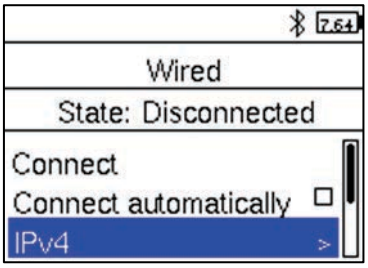


Windows 側の Bluetooth 名を選択します。

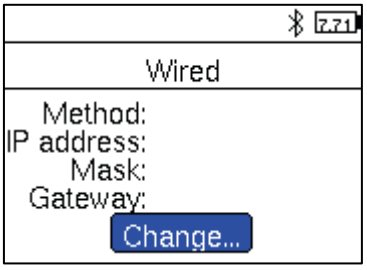
※この例では「DESKTOP-STG512P」となっています。



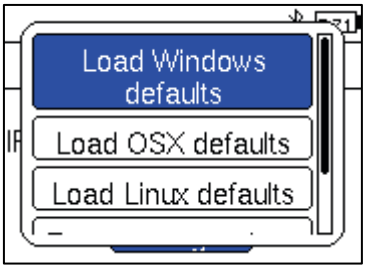
「IPv4」を選択します。



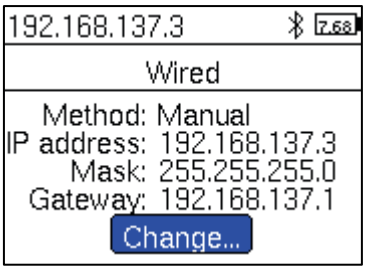
「Change」を選択します。



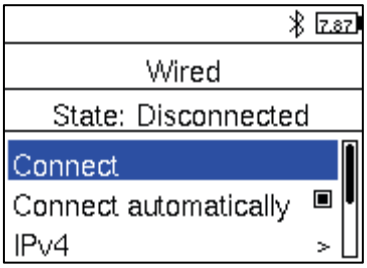
「Load Windows defaults」を選択します。



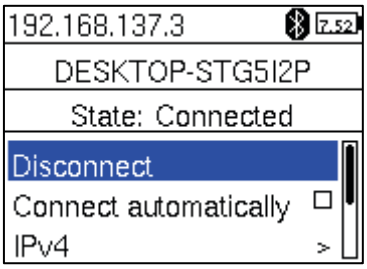
「192.168.137.3」に設定された IP アドレスを「192.168.138.3」に変更するため，「Change」を選択します。



クリアボタンを押し，「Connect」を選択します。



接続が完了すると，設定した IP アドレスが最上段に表示され，「Connected」と表示されます。



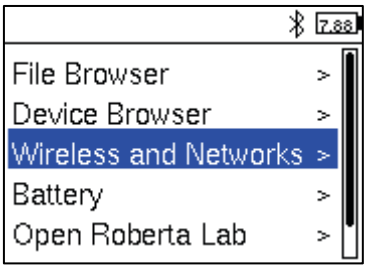
接続を終了したいときは，「Disconnect」を選択します。

③　無線 LAN による接続手順

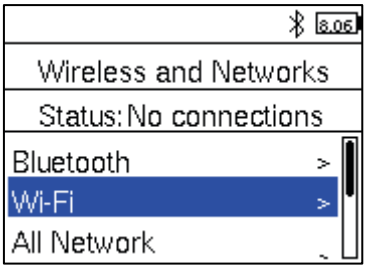
Windows パソコンは，すでに無線 LAN 接続できているとします。

EV3 ブロックに無線 LAN 用 USB ドングルを装着します。
ev3dev を起動します。 ※起動した後，無線 LAN 用 USB ドングルを装着しても動作します。

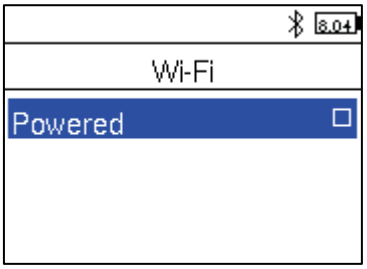
ev3dev の起動画面において，「Wireless and Networks」を選択します。



「Wi-Fi」を選択します。



「Powered」を選択し，□の中に点を表示させます。

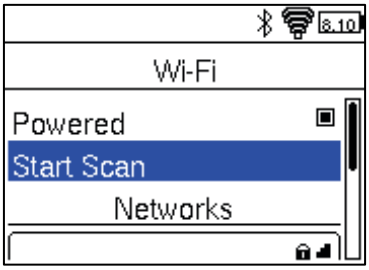


「Start Scan」を選択します。

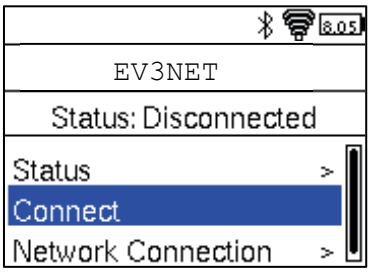
「Networks」の下に，無線 LAN アクセスポイントの SSID の一覧が表示されます。

以下，SSID は「EV3NET」として説明します。

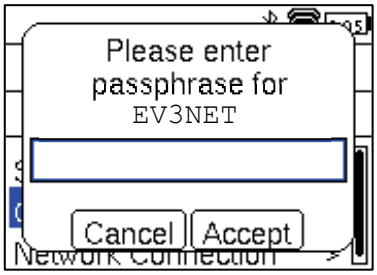
画面をスクロールし，接続先の SSID である「EV3NET」を選択します。



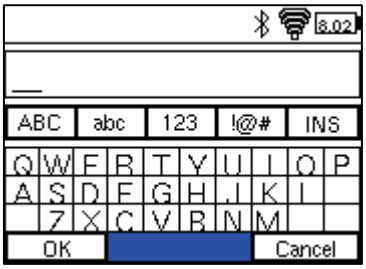
「Connect」を選択します。



「EV3NET」の下にある枠を選択します。

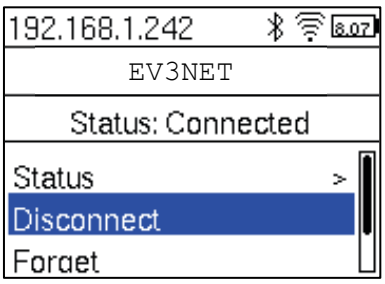


接続先の無線 LAN アクセスポイントのパスワードを入力し，「OK」を選択します。



「Accept」を選択します。

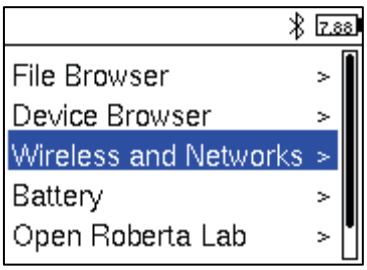
接続に成功すると，最上段に自動的に割り当てられた IP アドレスが表示され，Status が「Connected」と表示されます。



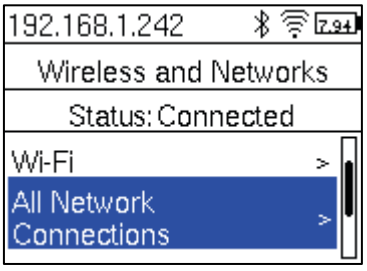
接続を終了したいときは，「Disconnect」を選択します。

EV3 ブロックの IP アドレスを個別に設定しておき，識別可能にする場合は，次の手順で IP アドレスを設定します。

ev3dev の起動画面において，「Wireless and Networks」を選択します。

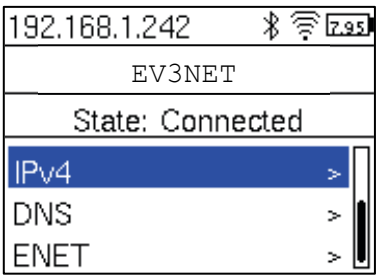


「All Network Connections」を選択します。

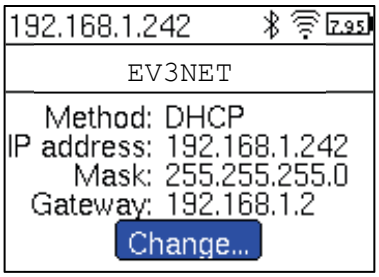


接続先の無線 LAN アクセスポイントの SSID を選択します。

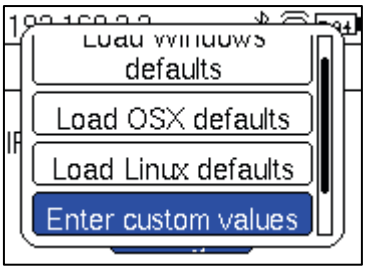
画面をスクロールさせ，「IPv4」を選択します。



「Change」を選択します。Method の「DHCP」は自動割り当てを示しています。



画面をスクロールさせ，「Enter custom values」を選択します。

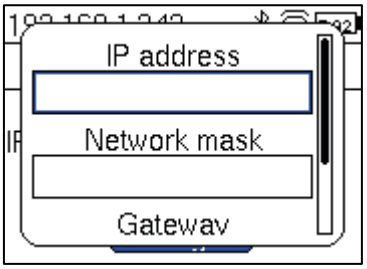


「IP address」の下の枠を選択し，IP アドレスを入力します。

例：192.168.1.11

「Network mask」の下の枠を選択し，ネットワークマスクの値を入力します。

例：255.255.255.0

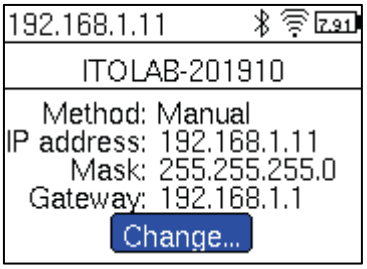


「Gateway」の下の枠を選択し，ゲートウェイの IP アドレスを入力します。

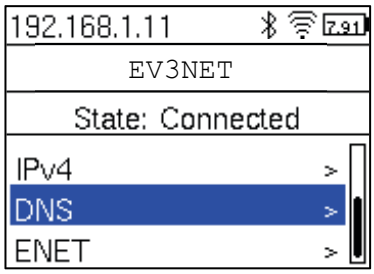
例：192.168.1.1

「Apply」を選択します。

IP アドレスが設定されたことが確認できます。

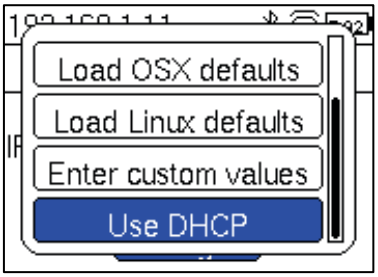


さらに，DNS を設定する場合，画面をスクロールさせ，「DNS」を選択します。



「Add」を選択して，DNS の IP アドレスを入力します。

自動割り当てに戻すときは，「Use DHCP」を選択します。

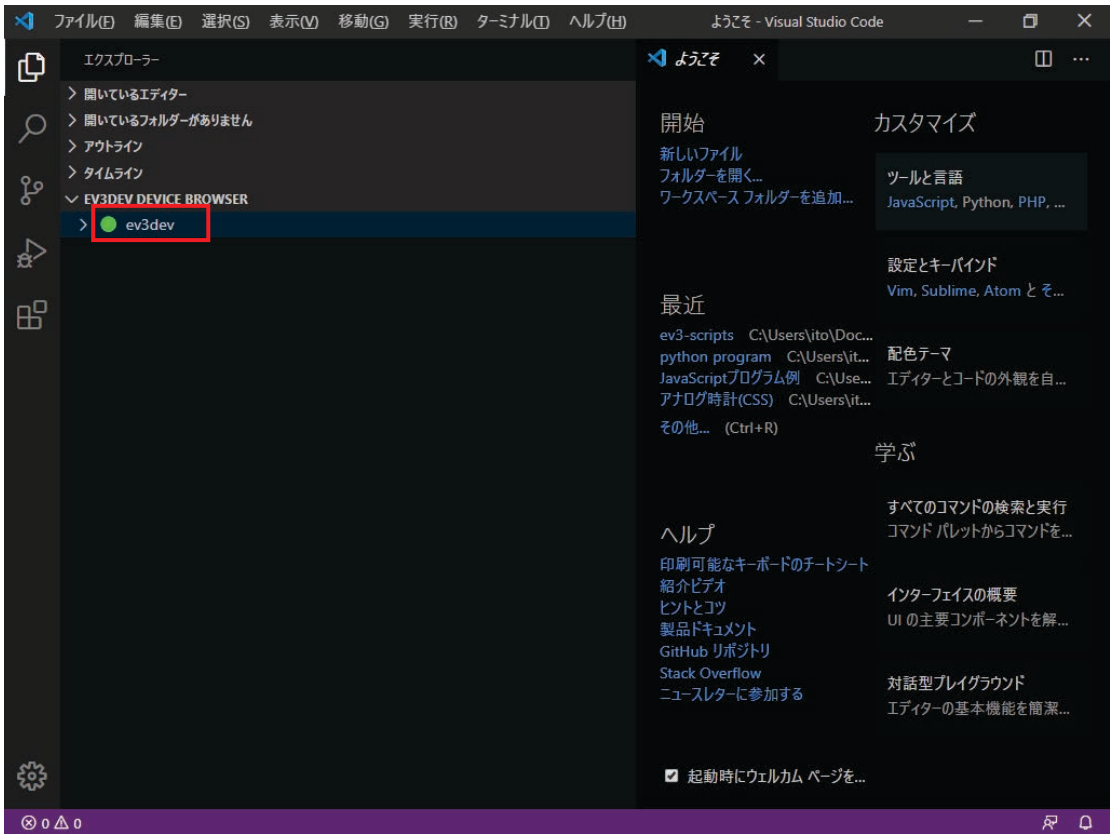


付録 D

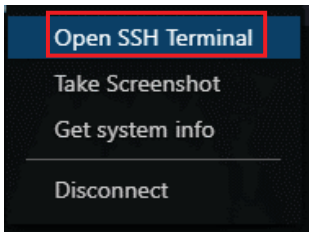
ev3dev の初期設定

ev3dev のホスト名は，デフォルトで「ev3dev」に設定されていますが，EV3 が複数ある場合，識別できなくなる場合，ホスト名を変更します。このホスト名は，それぞれのメモリーカードに保存されます。

パソコンと EV3 を USB ケーブルで接続します。
VS Code を起動し，「EV3DEV DEVICE BROWSER」を左クリックし，ev3dev と接続を完了させ，
● （緑色の丸）を表示させます。
「ev3dev」を右クリックします。



「Open SSH Terminal」を左クリックします。



ターミナルのウィンドウ枠が開きます。

robot@ev3dev:~$の行に，半角英数で

sudo ev3dev-config

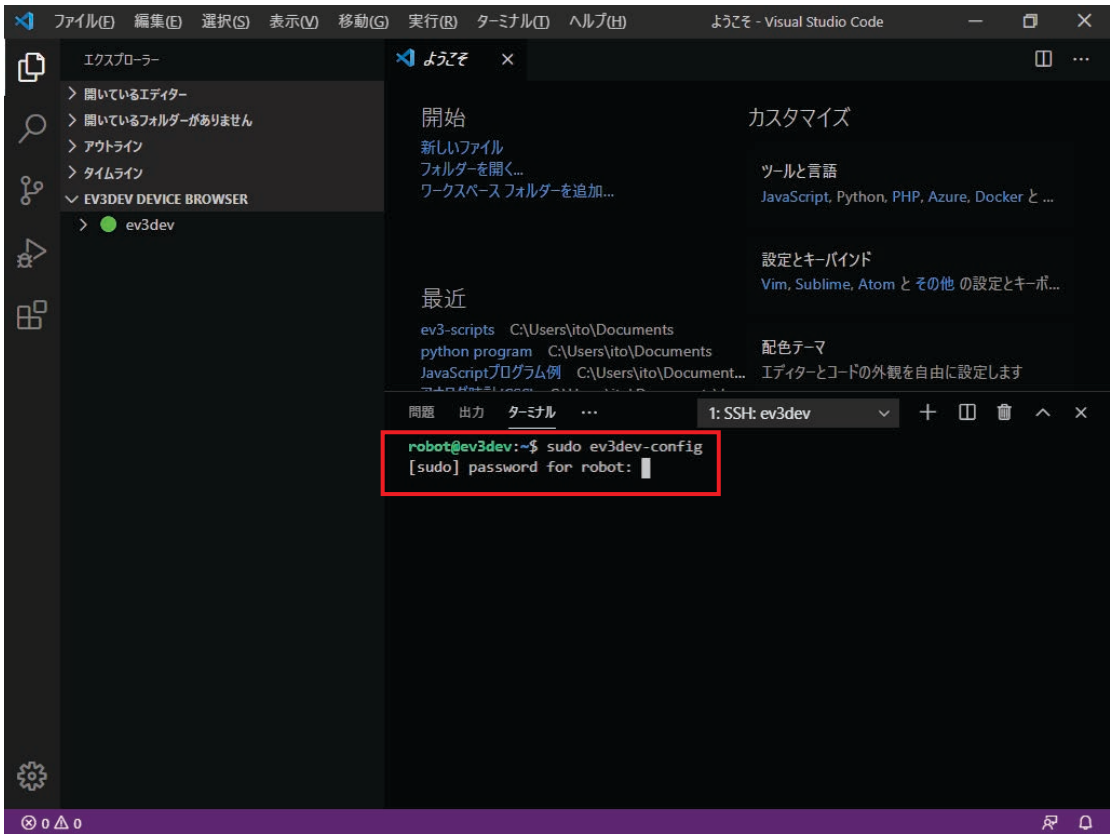
と入力し，エンターキーを押します。

[sudo] password for robot:

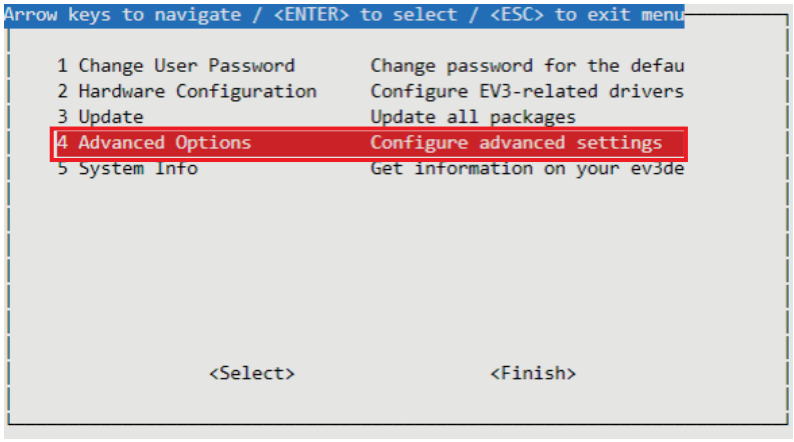
と表示されるので，

maker

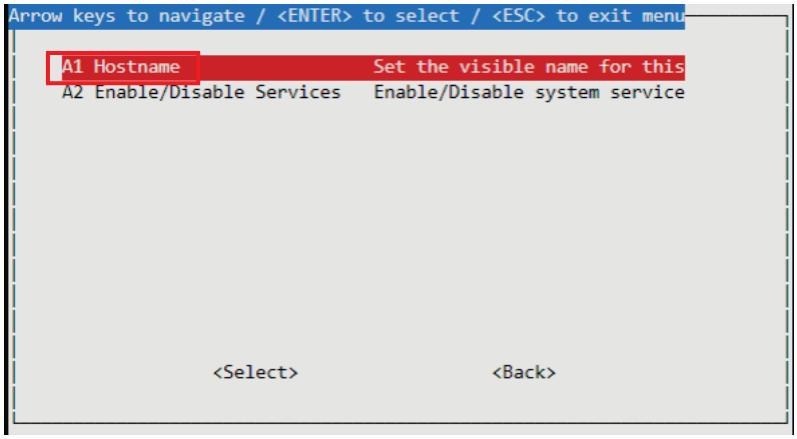
と入力し，エンターキーを押します。　※　パスワードである maker は表示されません。



上下矢印キーを使って「4 Advanced Options」を選択し，エンターキーを押します。

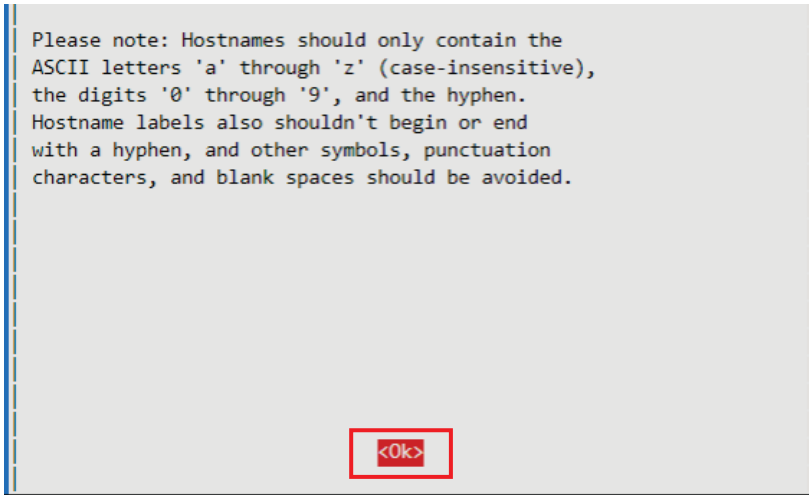


上下矢印キーを使って「A1 Hostname」を選択し，エンターキーを押します。

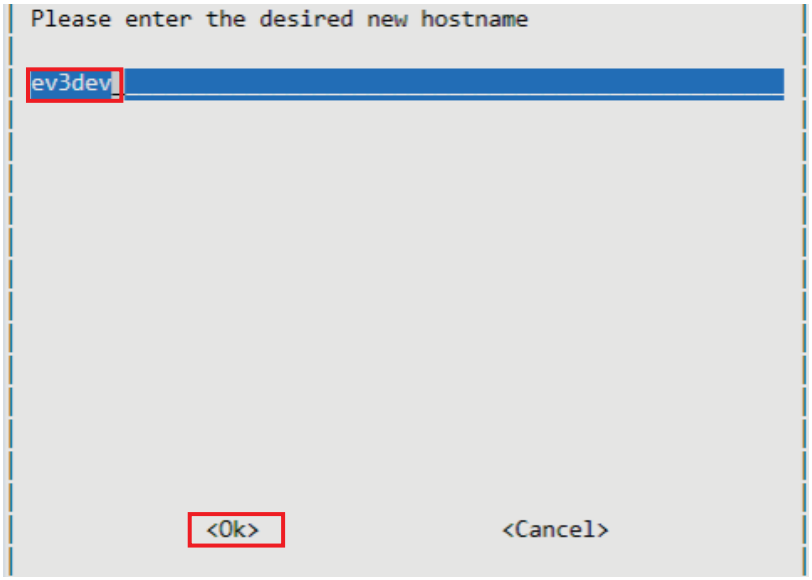


ホスト名は，アルファベットと数字，ハイフンで構成される文字列にするように注意が表示されますので，「OK」を選択し，エンターキーを押します。

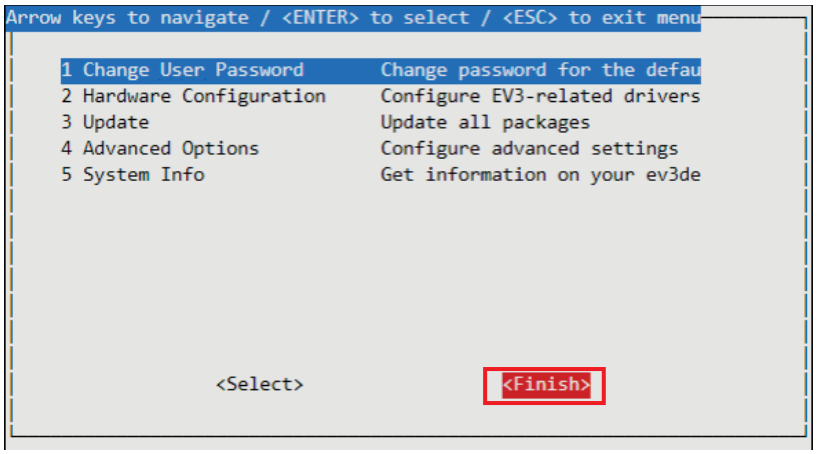
※ホスト名に「スペース」を含むことはできません。



現在設定されているホスト名が表示されている枠内に，新たに設定するホスト名を入力し，タブキーを押して，「Ok」を選択し，エンターキーを押します。



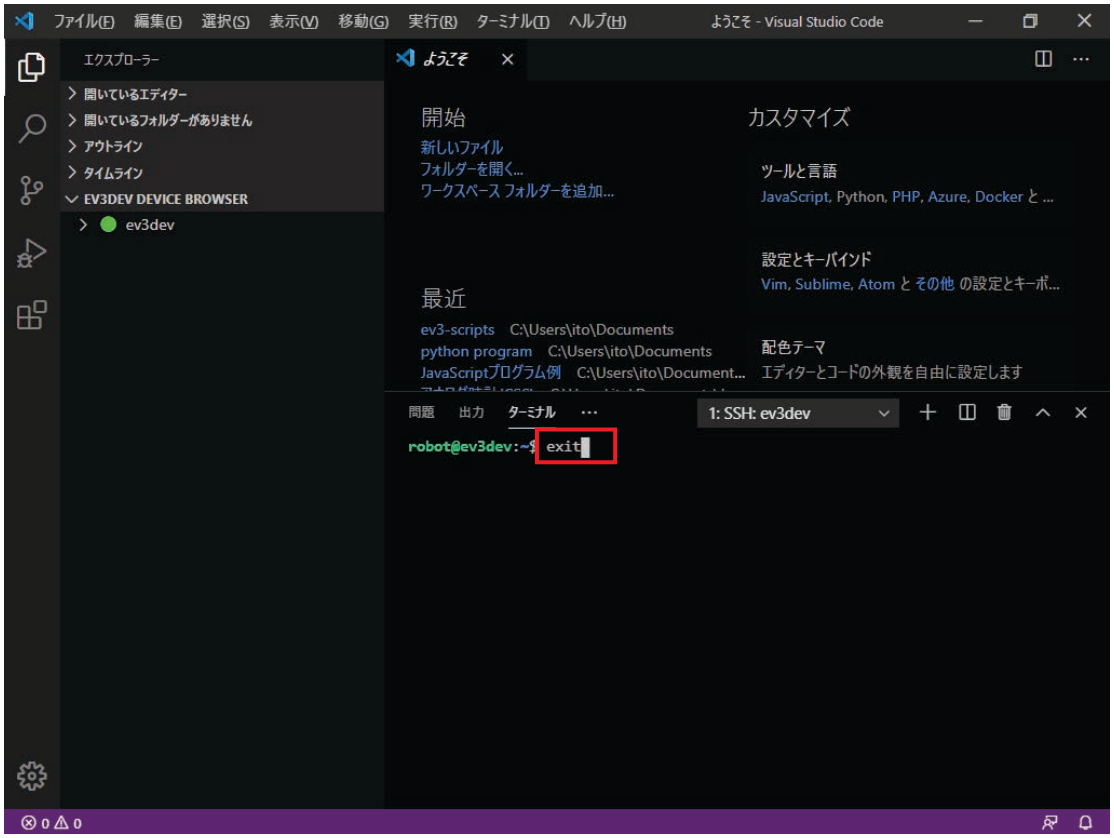
タブキーを押して，「Finish」を選択し，エンターキーを押します。



ターミナルを終了するため

exit

を入力し，エンターキーを押します。



ev3dev を再起動すると，新しく設定したホスト名で EV3 を識別できるようになります。

著者紹介

伊 藤　陽 介（いとう　ようすけ）

1987年　徳島大学大学院工学研究科修士課程修了，博士(工学)

機械製造業，高等専門学校勤務を経て，現在，鳴門教育大学大学院学校教育研究科(技術・工業・情報科教育実践分野)教授。情報技術教育に関する研究に従事。計測・制御教育等に関する研究成果を日本産業技術教育学会誌に論文発表するとともに，学術講演会等において研究成果を多数発表。主な著書：小・中・高等学校でのプログラミング教育実践(九州大学出版会)，教科内容学に基づく小学校教科専門科目テキスト 初等技術・情報(徳島県教育印刷㈱)。

謝　辞

本テキストを制作するにあたり，石塚仁志さん(2005年度修了，2003年度卒業)，森誉範さん(2006年度修了)，船橋知里さん(2008年度卒業)，中山詩衣奈さん(2010年度卒業)，塩谷音々さん(2019年度修了）にご協力いただきました。

プログラミング言語Pythonによる計測・制御入門

2020年12月24日　暫定版

著　者　　伊 藤　陽 介

発行者　　国立大学法人　鳴門教育大学大学院
技術・工業・情報科教育実践分野
伊藤研究室

国立大学法人　鳴門教育大学

学校名					
学　年		組		番号	
氏　名					