Unit 6: micro:bit with python

このレッスンでは，micro:bitのボタンとジェスチャについて学習し，つぎにサイコロを投げてデータプロットに転送するリストの値を収集するプログラムを作成します。
このレッスンには2つの部分があります。
Part 1：ボタンとジェスチャ
Part 2：ボタンやジェスチャを使ったデータ生成

Skill Builder 2: ボタンとジェスチャ

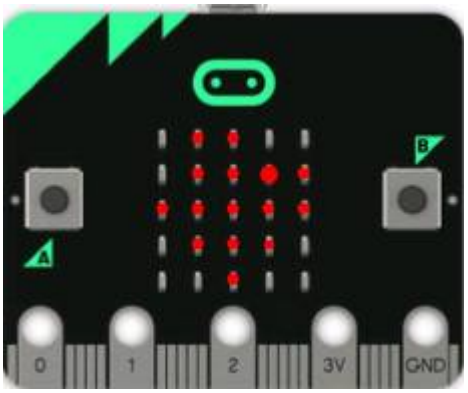
目標

- micro:bitのAボタンとBボタンを読み取って処理
- **.was**と**.is**の違いを観察
- PythonからTI-Nspireへのデータ転送
- micro:bitから収集したデータ調査
- ジェスチャを使ったディスプレイ制御

Teacher Tip: スキルビルダー1と同様，このレッスンは裏返しに設計されています。すなわち，コードは順番に導入されるのではなく，最初にmicro:bitのボタンとジェスチャ機能に焦点を当て，つぎにPythonリストとTI-Nspireリストを接続するように開発されています。

1. micro:bitには2つのボタンがあります。それらはディスプレイの両側にA，Bのラベルが付いています。Python micro:bitモジュールには，ボタンを読み取り，それらのボタンに基づいてタスクを実行する2つのメソッドがあります。
最初にメソッドをテストし，つぎにTI-Nspire CX Ⅱの他の場所でデータを収集して分析できるプログラムを作成します。

   micro:bitの背面には3軸加速度計/コンパスチップがあり，micro:bitの動きと向きを解釈するメソッドもあります。



Teacher Tip: micro:bitバージョン2では，ディスプレイの上にタッチボタン(金色の楕円形)もあります。モジュールでは，これは‘Logo Touch’(ロゴタッチ)と呼ばれ，メソッドis_touched()を使います。このタッチボタンについてレッスンでは説明していませんが，このレッスンで見た内容に基づいて，簡単に計画に組み込むことができます。以下の.is_と.was_の動作の違いに注意してください。

2. **Part 1**：ボタンとジェスチャ
新規ドキュメントで新規のPythonプログラムを開始します。
[home]を押して**New**を選択し，**Add Python >New…**(Pythonの追加>新規…)を選択します。
プログラムにbuttons_gesturesという名前を付けました。

   [menu] > More Modules > BBC micro:bit (メニュー>その他のモジュール>BBC micro:bit)から，リスト一番上の**import**ステートメントを選択します。

   from microbit import *

3. **while**ループを追加します。

   while get_key()! = 'esc':

   これは，次のところから選択します。

   [menu] > More Modules > BBC micro:bit > Commands
   (メニュー>その他のモジュール>BBC micro:bit>コマンド)

   ほとんどすべてのmicro:bitプログラムはこのように設計されます。

4. ボタンAをテストするには，**if:**構造を追加します。

   ♦♦if button_a.was_pressed():
   ♦♦♦♦print("Button A")

   ifは**while**ループの一部としてインデントされ，**print()**はさらにインデントされてifブロックの一部になります。Pythonでは適切なインデントが非常に重要であることを忘れないでください。インデントを間違えると，構文エラーやコードの不適切な実行が発生する可能性があります。インデント間隔を示す薄い灰色のひし形(♦♦)に注意します。

   if は，**[menu] > Built-ins > Contro**l (メニュー>組み込み>制御)にあります。

   条件**button_a.was_pressed()**は，次のところにあります。
   [menu] > More Modules> BBC micro:bit > Buttons and Logo Touch (メニュー>その他のモジュール>BBC micro:bit>ボタンとロゴタッチ)

   Print()は，**[menu] > Built ins > I/O** (メニュー>組み込み>I/O)にあります。
   Print()関数の中には，"Button A"というテキストを入力します。

   Note: is_pressed()については，後で説明します。

5. これで，プログラムをテストする準備が整いました。**[ctrl] [R]**を押してプログラムを実行します。何も起きていないようです。micro:bitのボタンAを押して放します。電卓の画面に'Button A'が表示されます。ボタンを押すたびに，右の画像のようにテキストが表示されます。

   [esc]を押して，Pythonエディタに戻ります。

6. 条件**button_b.ispressed()**を使ってボタンBをチェックするため，別のifステートメントを追加します。'IS' は 'WAS' とは異なることに注意してください。

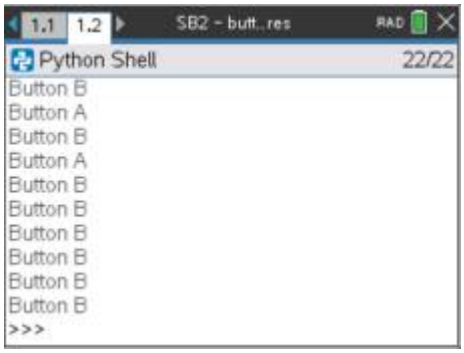
それらがどのように異なるかがすぐに分かります。

♦♦if button_b.is_pressed():
♦♦♦♦print("Button B")

Tip: 繰り返しになりますが，インデントに注意してください。

Teacher Tip: 2つの関数の動作は異なります。2つの異なる動作のどちらを選択するかが重要になるプログラミング環境があります。

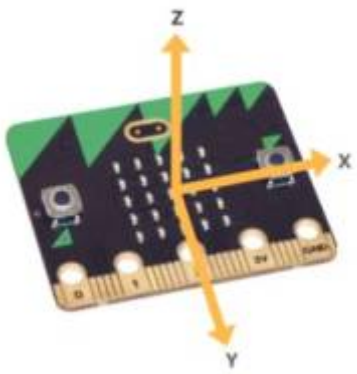
7. プログラムを再度実行します([ctrl] [R])。ボタンAとボタンBの両方を試してください。
各ボタンをタップしたり，各ボタンを押し続けたります。
ボタンBは押されている限り'Button B'が繰り返し表示されますが，'Button A'は表示されません。**was_pressed()** (リセットするにはボタンを離す必要があります)と，ステートメントが処理された瞬間にボタンが押されているかどうかを確認する**.is_pressed()**には違いがあります。



Note: ボタンBをすばやくタップすると，ifステートメントが処理されている瞬間にボタンが押されていないため，プログラムがボタンBを表示しない場合があります。

Teacher Tip: .was_pressed()は個々のプレス(クリック)を検出するため，完全な'down-up'(ダウンアップ)アクションを必要とします。
.was_pressed()は，ループ実行中にボタンがクリックされた場合でも，発生する可能性のあるダウンアップイベントを検出します。別のクリックを発生させるには，ボタンを離す必要があります。micro:bitはボタンが押されたことを記憶します。
.is_pressed()は'is down?'(ダウン?)イベントのようなものです。一部のイベント駆動型プログラミング言語には，'mouse_down'など同様の機能があります。**.is_pressed()は，ステートメントが処理される正確な瞬間にボタンが押されている場合にのみ**True**を生成します。**
このレッスンの残りの部分では，ボタンBのコードは無視されます。

8. **ジェスチャの紹介**　micro:bitには，3つの異なる方向の加速度を測定する加速度計があります(3軸または3D加速度計，右図参照)。micro:bitは各方向の加速度の数値を提供するだけでなく，これらの方向の値に基づく'face up' (表が上)や'face down'(表が下)などのシンプルで分かりやすいジェスチャも提供します。



このレッスンでは，これらのジェスチャについて説明し，micro:bitを使ってTI-Nspire CXⅡをプログラミングする際，それらを使う方法を示します。

9. micro:bitからジェスチャ値を取得し，変数gに格納します。

♦♦**g = accelerometer.current_gesture()**

Tip: 繰り返しになりますが，インデントに注意してください。

♦♦**g =**と入力してから，次を入力します。

accelerometer.current_gesture()

これは，**[menu] > More Modules > BBC micro:bit > Sensors > Gestures >** (メニュー>その他のモジュール>BBC micro:bit>センサ>ジェスチャ>) から選択します。

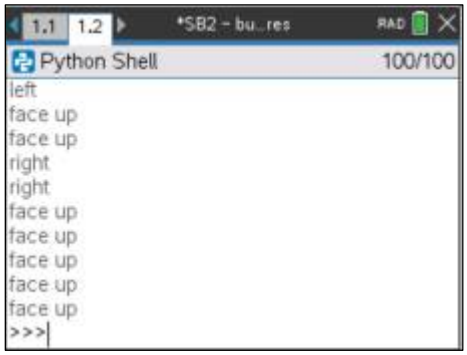
つぎに，gの値を出力します。

♦♦**print(g)**

Note: print()は，**[menu] > Built ins > I/O** (メニュー>組み込み>I/O)にあります。

これらの2つのステートメントは，**while**ループの一部であるが，**if**ステートメントの一部ではないようにインデントされていることに注意します。各行のインデントが意味を決定することを忘れないでください。

10. プログラムを実行し，micro:bitを空中で動かしながら電卓の表示でさまざまな値を確認します。micro:bitを裏返し，両端に立て，振って，ガタガタと転がします。サンプル実行の一部が右図に示されています。



可能なジェスチャには，face up(表が上), face down(表が下), up(上向き), down(下向き), left(左向き), right(右向き), shake(シェイク)などがあります。これらは文字列が値として返されます。これらすべてのジェスチャを画面に表示できますか。

11. 図に示されているように，最後の2つのジェスチャステートメントを#コメントにし([ctrl] [T]を使って#を行頭に置く)，同じメニューからこの他のジェスチャ関数を追加します。

♦♦**if accelerometer.was_gesture(“face down”):**
♦♦♦♦**print(“face down”)**

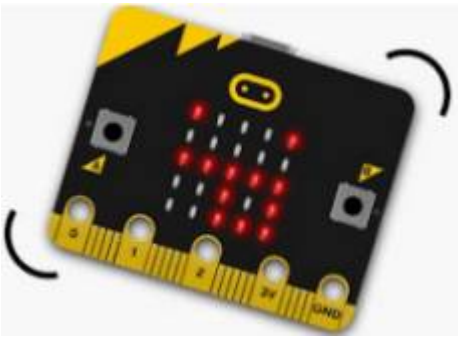
ジェスチャメソッドとジェスチャ文字列はすべてオンになっています：**[menu] > More Modules > BBC micro:bit > Sensors > Gestures** (メニュー>その他のモジュール>BBC micro:bit>センサ>ジェスチャ)

ジェスチャ文字列を入力するだけですが，**.was_gesture()**メソッドは，メニュー項目と正確に一致する必要があります。

このプログラムを実行すると，出力の変化に気付くでしょう。
.current_gesture()はつねにジェスチャを出力します。
.was_gesture()はジェスチャが変更されたときだけ出力されます。

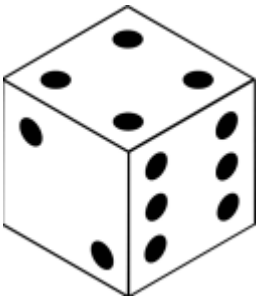
12. ボタンとジェスチャは，micro:bitから入力を取得し，TI-Nspire CXⅡ画面またはmicro:bitディスプレイ，あるいはその両方で結果を生成する方法です。

    このレッスンの次のパートでは，TI-Nspire CXⅡをさらに調べるため，micro:bitを使ってデータを生成するアクティビティを作成します。



13. **Part 2:** サイコロを投げ(各面に1～6の番号が付いた立方体)：**ボタンAが押されたら**，1から6までのランダムな整数を変数に割り当てます。ボタンA，ボタンB，または任意のジェスチャを使えます。

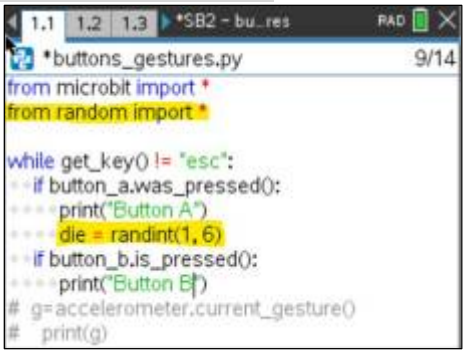
    micro:bitのみに値を表示します。次のステップに進む前に，自分で試してみてください。現在のプログラムを使って，コードを追加してサイコロをシミュレートします。



    図のサイコロの底にあたる目の数は分かりますか。

    Teacher Tip: 解答：底の値は3です。サイコロのある面の目と，反対側の面の目の数の合計はつねに7になります。

14. 右図に示すように，強調表示された2つのステートメントを追加します。

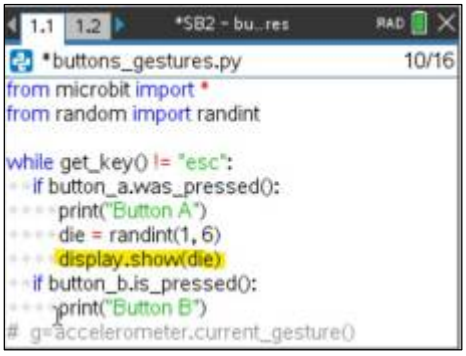


    from random import *と randint()は，両方とも**[menu] > Random** (メニュー>ランダム)にあります。

    ステートメントの残りの部分♦♦♦♦**die = randint(1, 6)**は手動で入力し，**if button_a...**ブロックの一部であるため，インデントされます。変数dieは英語でサイコロを表します。

    繰り返しますが，インデントに注意してください。

15. サイコロの目の値が定まったら，次のステートメントを追加します。

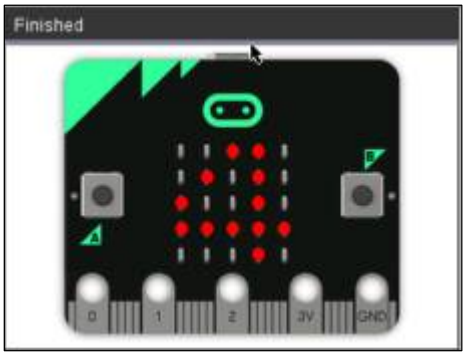


    display.show(die)

    micro:bitディスプレイにサイコロの目の値を表示します。

    プログラムを再実行してください。ボタンAを押すと，電卓画面に‘Button A’が表示され，micro:bitディスプレイの数が変わりますが，毎回ではありません。選択した乱数が最後の数と同じである場合もありますが，それで問題ありません。「ボタンを押す」は独立したイベントです。

16. **データ収集**：ボタンを押すだけでサイコロ投げになるのはナイスです。さらに研究するため，データを解釈できるよう値をすべて保存します。どの目が最もよく出るのか，平均はいくつか，などなど…。



プログラムにステートメントを追加して，次のようにします。
- 空のリストを作成します。
- サイコロの目の値をリストに追加(**.append**)します。
- 分析のためPythonからTI-Nspire CXⅡシステムにリストを保存します。

これらの3つの作業はそれぞれ，プログラムの特別な場所に配置するステートメントに変換されます。次のステップに進む前に，自分で試してみてください。

17. 空のリストの割り当ては，プログラムの開始時に置きます。

tosses = [ ]

変数名は**tosses**(トス)，角かっこはキーパッドや**[menu] > Built-ins > Lists** (メニュー>組み込み>リスト)にあります。

サイコロの目が決定すると，次のようにリストに追加されます。

tosses.append(die)

.append()は，**[menu] > Built-ins > Lists** (メニュー>組み込み>リスト)にあります。

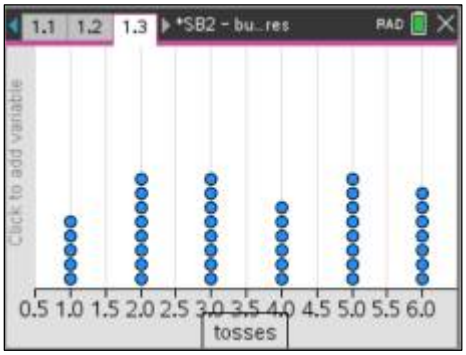
プログラムの最後に，最終的なリストがTI-Nspireリストに保存されます。

store_list(“tosses”, tosses)

store_list関数は，**[menu] > More Modules >BBC micro:bit >Commands** (メニュー>その他のモジュール>BBC micro:bit>コマンド)にあります

Note: コメント付きの2行は，コードがすべて画面に収まるように削除されました。プログラムにはそれらのコメントを残しておいてもかまいません。

18. ここでプログラムを実行します。ボタンAを50回ぐらい押します。プログラムの終了は**[esc]**を押します。



Pythonプログラムにはtossesという名前のリストがあり，TI-Nspireにもtossesという名前のリストがあります。これら2つは別々のリストです。

[ctrl]-[doc] または **[ctrl]-[I]**を押してページを挿入し，Data & Statistics(データと統計)アプリを追加します。たくさんのドットがグラフの周りに散らばっています。アプリ画面下にある’Click to add variable(クリックして変数を追加)’というメッセージをクリックし，リストtossesを選択します。右の画面が現れましたか。各ドットは，サイコロ投げの1つを表します。このグラフか

らどのようなことが分かりますか。グラフをヒストグラムに変更できますか。**ヒント**：アプリの**[menu]**を押します。

このプログラムはいくつかの方法で変更できます。たとえば，サイコロを投げている間にサイコロを投げた回数を表示(print)します。

Note: ボタンBとジェスチャは，サイコロには影響しません。micro:bitを'shake(振る)'のときだけ，サイコロを投げるようにコードを変更してみてください。

ドキュメントを保存することを忘れないでください。

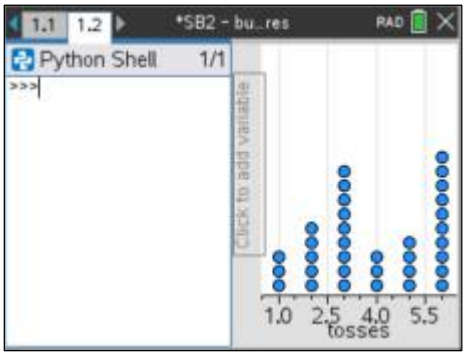
Teacher Tip: ボタンBはリセットボタンとして使用でき，データをクリアして空のリストからやり直すことができます。

これまでのプログラムの#comments：

19. 発展：

- **store_list()**メソッドをインデント(字下げ)して，**button_a**を押すたびにリストがTI-Nspireに保存されるようにします。
- プログラムの最後に**print("done.")**を実行して，プログラムが終了したことを確認します。
- **if buttonB…**コードと**if accelerometer…**コードにコメントします。

20. シェルアプリに移動し，**[ctrl] [4]**を押して，シェルとData & Statistics(データと統計)アプリを右図のように1つのページに結合します。



シェルで**[ctrl] [R]**を押してプログラムを実行します。micro:bitのボタンAを押して，サイコロを投げ始め，ドットプロットが成長するのを確認します。

tossesリストがプログラムによって空にされたため，プロットには最初は'No numeric data(数値データなし)'と表示されますが，恐れることはありません。ボタンAを押すとデータプロットが開始されます。

21. ボタンを押すたびにボタンAだけをprintする代わりに，tossesのリスト全体をprintできます。コードのどこに**print(tosses)**しますか。