Unit 6: micro:bit with python

この応用では，micro:bitを使ってデータを収集するプログラムを作成し，TI-Nspireの分割画面でドットプロットが大きくなるのを観察しながらプログラムを実行します。

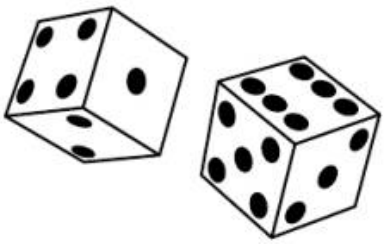
Application: サイコロ投げ

目標

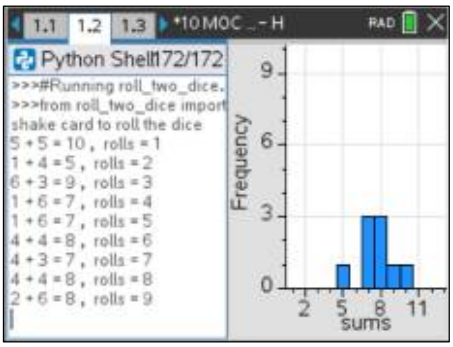
- micro:bitデータ収集プログラムの作成
- 収集されたデータの動的なデータと統計プロットを作成

1. この応用のプロジェクトは，前の3つのスキルビルダーで学習したすべてのmicro:bitスキルをまとめたものです。

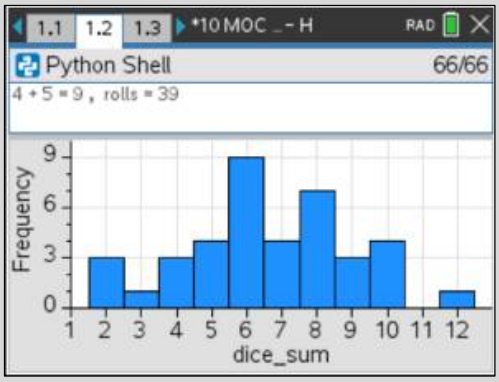
   'shake'(あるいはボタンを押す)などのジェスチャを使ってデータを収集し，リストをTI-Nspire変数と…



2. ...それから，TI-Nspireページを設定します。
   - 画面左側(Pythonシェル)でPythonプログラムを実行し，
   - プログラムの実行中に収集されたデータのドットプロット(またはヒストグラム)を画面右側のData & Statistics(データと統計)アプリで表示します。

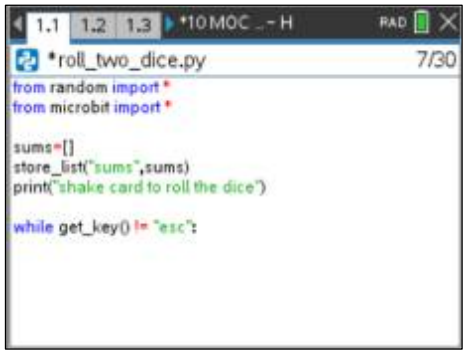


Teacher Tip: 上記のTI-Nspire分割画面のレイアウトは，垂直または水平のいずれかになります。ctrl-4を押して2つのアプリを1つの画面にグループ化すると，Student Lessonで見られるように，既定値は垂直になります。水平レイアウトに切り替えるには，**[doc] > Page Layout > Select Layout > Layout 3** (ドキュメント>ページレイアウト>レイアウト選択>レイアウト3)と押します。
セパレータバーを調整して，Pythonシェルアプリを小さくし，Data & Statistics(データと統計)アプリを大きくすることもできます(下図参照)。



このレッスンでは，ヒストグラムではなくドットプロットを作成します。Data & Statisticsアプリでドットプロットからヒストグラムに変換する手順については，このドキュメントの**Teacher Tip** (後出)に記述されています。

3. micro:bitプログラムは，randomモジュールを含む通常のインポートで開始し，sums(合計)と名付けた空のリストから開始します。



sums = [ ]

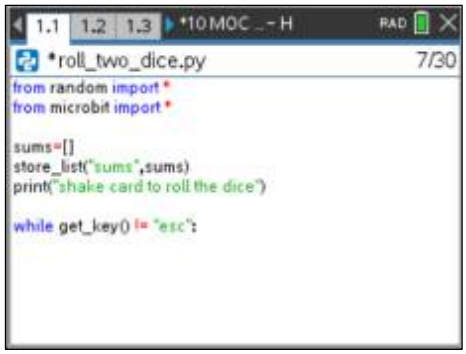
すぐに，このリストを(同じ名前を使って)TI-Nspire変数に保存します。

store_list(“sums”, sums)

TI-Nspireリストもクリアされるようにします。

Print()ループが始まる前にユーザーにいくつかの指示を出します。’shake’(シェイク)ジェスチャを使ってサイコロを転がします。

4. **while**ループ本体で，ジェスチャを使って



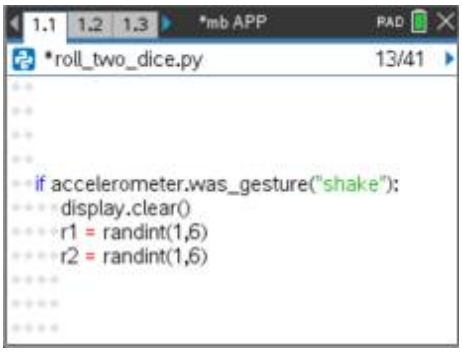
- 2つのサイコロを投げます(**toss**)。2つのランダム整数を生成します。
- それらの和(**add**)を求めます。
- 和をリストsumsに追加(**append**)します。
- TI-Nspire画面に2つのサイコロの値，それらの和と振った回数を**print**します。

  Hint: len(sums)は振った回数です。**len**は，リストの場合，要素の数を返します。
- micro:bitに2つのサイコロの目の数を表示(**display**)します。
- リストをTI-Nspire変数に保存(**store**)します。

やってみましょう。

Teacher Tip: 学生がこのプログラムを行うには，スキルビルダー1, 2, 3から十分な知識を得ている必要があります。メニューに苦労しているときは，前のスキルビルダーをもう一度見るように伝えてください。

5. サイコロを投げるには，ジェスチャまたはボタンを押します。

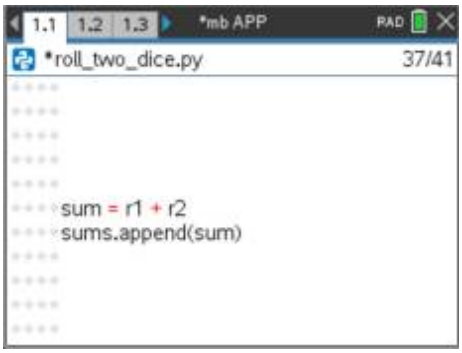


♦♦if accelerometer.was_gesture("shake"):
♦♦♦♦display.clear()
♦♦♦♦r1 = randint(1,6)
♦♦♦♦r2 = randint(1,6)

インデントに注意します。

6. それらを**add**(足し算)し，**sum**をリスト**sums**に**.append**(追加)します。

```
sum = r1 + r2
sums.append(sum)
```



7. micro:bitディスプレイに2つのサイコロの目の数を順に表示します。2つのサイコロの目が同じである可能性があるため，2つが実際に表示されることを確認する必要があります。

```
display.clear()
display.show(r1)
sleep(250)
display.clear()
display.show(r2)
sleep(250)
```

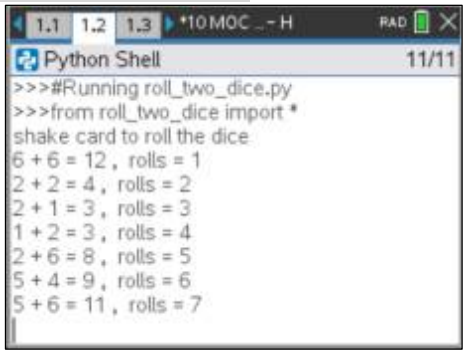
sleep()コマンドは，遅延を長くしたい場合に使います。
コードを正しく，正しい順序で入力した場合は，今すぐプログラムを実行して，micro:bitを振ってみてください。micro:bitに2つの数字が表示されているはずです。

> **Teacher Tip:** 別の**sleep()**は，micro:bitにジェスチャを監視する機会を与えるため，ループで役立つ場合があります。

8. TI-Nspire画面にサイコロの目の数，合計，rolls(振った回数)を印刷するコードを追加します。次のような1つのprint()ステートメントを使います。

♦♦♦♦**print (r1, "+", r2,"=",sum,", ","rolls =", len(sums))**

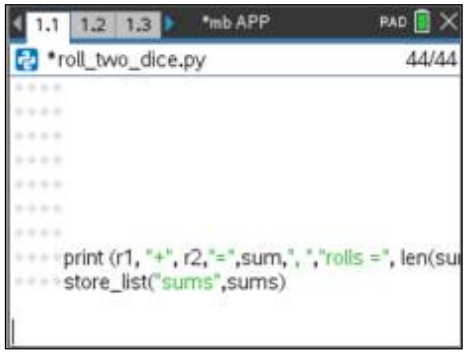
その結果，右の画像に示されている行になります。
句読点に注意してください。



9. Pythonリストの合計を同じ名前のTI-Nspireリストに保存します。

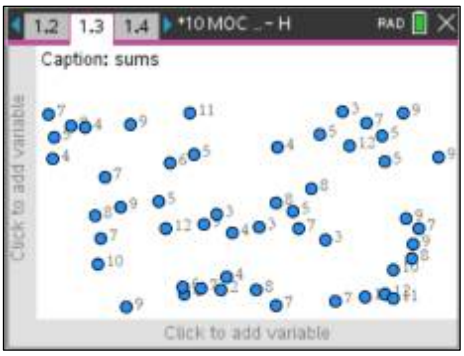
♦♦♦♦**store_list("sums", sums)**

この**store_list()**ステートメントはwhileブロックとifブロックの奥にあるため，新しいサイコロのペアが生成されるたびにTI-Nspireリストが更新されます。



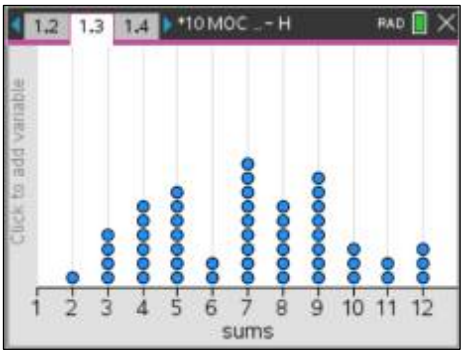
Teacher Tip: 上記のprintステートメントにcounter変数がないことに注意します。プログラマーは振った回数(rolls)として**len(sums)**を使うだけなので，これは必要ありません。

10. プログラムが正常に機能していることを確認したら，Pythonプログラムを TI-Nspireプロット機能に接続する準備が整います。プログラムを実行して，約50回サイコロを振ります。**[esc]**を押してプログラムを終了します。

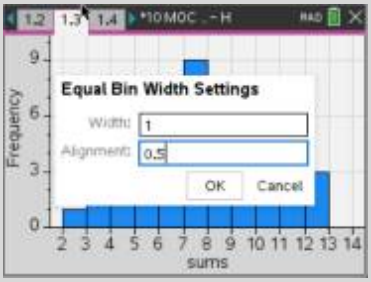


Pythonシェルで(コマンドプロンプト>>>で)**[ctrl] [doc]**または **[ctrl] [I]**を押してページを挿入します。Data and Statistics(データと統計)アプリを選択します。右のような画面が表示されます。sums(合計)データは画面全体に散らばっています。

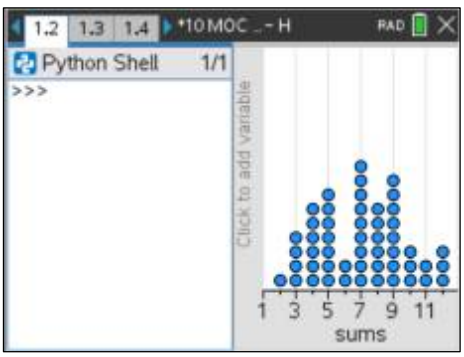
11. 画面の下部にある'Click to add variable(クリックして変数を追加)' メッセージをクリックし，変数sumsのリストを選択します。分散したデータ点は，その値に従ってx軸に沿って整理され，ウィンドウはデータに適合しています。これはドットプロットです。



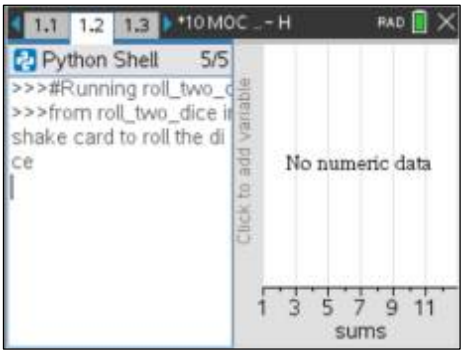
Teacher Tip: プロットをヒストグラムに変更できます：**[menu] > Plot Type > Histogram** (メニュー>プロットタイプ>ヒストグラム)を押します。ただし，バーがx軸の値の中央にくるように，ビンの配置を0.5に調整する必要もあります。**[menu] > Plot Properties > Histogram Properties > Bin Settings** (メニュー>プロットプロパティ>ヒストグラムプロパティ>ビン設定)。そして，**Equal Bin Width**(ビン幅を等しく)とし，**Alignment**(配置)を0.5に設定します。



12. 1ページ戻ってPythonシェルに戻り(**[ctrl]+左矢印)**，**[ctrl] [4]**と押してPythonシェルをData and Statistics(データと統計)アプリとグループ化し，分割画面ページを作成します。右図に示すように，左側がPythonシェルアプリ，右側がData & Statisticsアプリです。



13. シェルは再初期化されているため，**[ctrl] [R]**を押してもプログラムは再実行されません。Pythonエディタに戻り，**[ctrl] [R]**を押してプログラムを実行します。右図に示すように，左側画面のシェルで実行されます。プログラムはすぐに空のリストを保存するため，右側に'No numeric data(数値データなし)'と表示されます。



データを収集すると(micro:bitを振ってサイコロを振る)，右側のData & Statistics(データと統計)アプリにsums値がドットとして表示されていきます。

[esc]を押すとプログラムが終了し，TI-Nspire環境で他の多くの1変数データ分析ができます。

ここで**[ctrl] [R]**をもう一度押すと，(Pythonシェルで)プログラムが再実行されます。

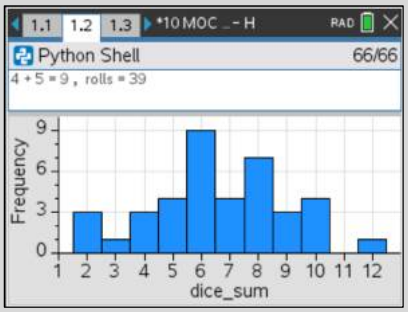
Hint: 各実行の開始時にシェルをクリアするには，次のステートメントをプログラムの最初に追加します。

clear_history()

これは，**[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。

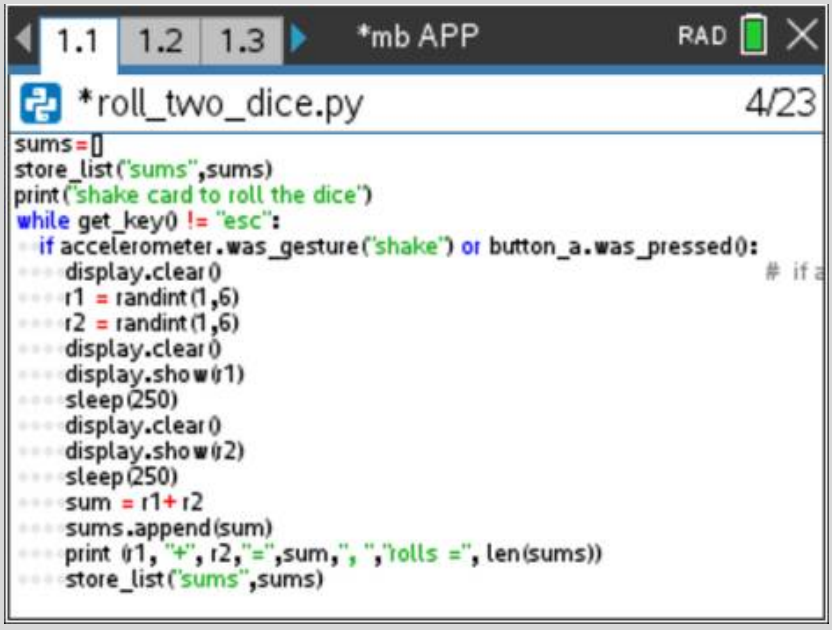
ドキュメントを保存することを忘れないでください。

Teacher Tip: (再出) TI-Nspireの分割画面レイアウトは，垂直または水平のいずれかになります。ctrl-4を押して2つのアプリを1つの画面に結合すると，Student Lessonで見られるように，既定値は垂直になります。ハンドヘルドで水平レイアウトに切り替えるには，[doc] > Page Layout > Select Layout > Layout 3 (ドキュメント>ページレイアウト>レイアウト選択>レイアウト3)を押します。セパレータバーを調整して，Pythonシェルアプリを小さくすることもできます(下図参照)。



ドットプロットをヒストグラムに変更するには，**[menu] > Plot Type** (メニュー>プロットタイプ)を押してHistogramを選択します。

解答例：



オプションの拡張機能：dieの各面にピップを表示
micro:bitに表示するカスタム画像を簡単にデザインできます。次のコードは，6つのdie面(pips)を作成します。画像の大文字のIIに注意してください。

最初のブロックにone=Image(…と入力し，他の5つの面をコピー/貼り付け/編集します。つぎに，要素#0にNoneを使って，die値がリストのインデックスと一致するように，dice_imagesのリストを作成します。
Pythonは，区切り文字(コンマなど)なしで別々の行に記述された2つの文字列を1つの文字列として解釈するため，
“aaa” と “bbb”は，次と同じです
“aaabbb”
次のコードでは，画像の設計を容易にするために，5行のmicro:bitがImage()関数で複製されています。各LEDの明るさを制御するために，画像の各桁の値を0～9にすることができます。

```
############################################################
from microbit import *
from random import *

one=Image(
"00000:"
"00000:"
"00900:"
"00000:"
"00000")
two=Image(
"00000:"
"09000:"
"00000:"
"00090:"
"00000")
```

```
three=Image(
"00000:"
"09000:"
"00900:"
"00090:"
"00000")

four=Image(
"00000:"
"09090:"
"00000:"
"09090:"
"00000")

five=Image(
"00000:"
"09090:"
"00900:"
"09090:"
"00000")

six=Image(
"00000:"
"09090:"
"09090:"
"09090:"
"00000")

dice_images=[None, one, two, three, four, five, six]
#    index:   0   1   2    3     4    5    6
print("running...")
while get_key() != "esc":
 if button_a.is_pressed():
   die=randint(1,6)
   display.show(dice_images[die])  # display one of the dice's faces using pips
```