

Unit 6: micro:bit with python

Skill Builder 3: 光センサ

このレッスンでは、micro:bitの光センサをモニターし、さらに分析するためにデータをTI-Nspireリストに保存します。

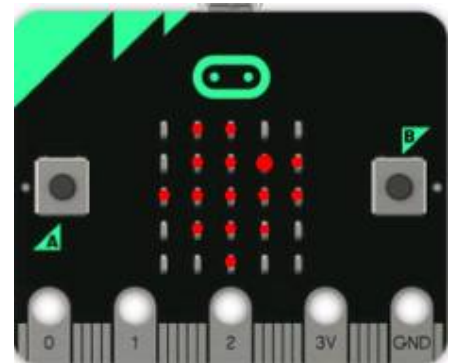
目標

- Micro:bitの輝度を読み取って表示
- PythonからTI-Nspire CX IIIにデータ転送
- micro:bitから収集したデータの調査

Teacher Tip: このレッスンでは、点光源からの照度の逆2乗の法則の近似式を得ます。法則は、時間ではなく光源からの距離と光の強度の関係です。しかし、光源から離れるmicro:bitの、周到で規則的な一定の動きは、時間と距離の間に線形関係を作ります。時間は距離の良い代わりになっています。

1. micro:bitは、ディスプレイLEDを使って周囲の光レベルを読み取ることができます。

ディスプレイLEDは、出力だけでなく入力デバイスとしても使用でき、micro:bitに当たる光の量を測定するセンサとして利用できます。



Teacher Tip: micro:bitの光レベルの詳細については、次をご参照ください。
[Sensing changes in light on the micro:bit : Help & Support](#)

2. 新規のドキュメントで、新規のPythonプログラムを開始します。

[home]を押し、**New > Add Python > New...** (新規>Pythonの追加>新規...)を選択します。
 プログラムに**bright**という名前を付けます。

光センサが検出できる値を確認するには、BBC micro:bitメニューを使って次の短いプログラムを作成します。

```
from microbit import *

while get_key() != "esc":
    ♦♦bright = display.read_light_level()
    ♦♦print("brightness = ",bright)
```

[menu] > More Modules > BBC micro:bit > Sensors (メニュー>その他のモジュール>BBC micro:bit>センサ)で **var= .read_light_level()**を見つけます。brightは入力する変数名になります。

Note: [menu] > More Modules > BBC micro:bit > Commands > **while get_key() != "esc"** (メニュー>その他のモジュール>BBC micro:bit>コマンド>while get_key() != "esc")
 最後の2つのステートメントは、whileループ本体になるようインデントすることを忘れないでください。



- プログラムを実行し、micro:bitのディスプレイを光源に向けます。ディスプレイに何が表示されていても問題ありません。Micro:bitを光源に近づけたり遠ざけたりして、TI-Nspire CX II 画面で値の変化を観察します。0から255の間で変化する値が表示されるはずで

想像どおり、光源から離れるほど光レベルの値は低くなります。つぎに、プログラムにコードを追加して、光レベルデータを収集し、光と時間の散布図を作成します。

[esc]を押してプログラムを終了し、エディタに戻ります。

```
Python Shell 65/102
Brightness = 250
Brightness = 245
Brightness = 242
Brightness = 236
Brightness = 229
Brightness = 224
Brightness = 216
Brightness = 210
Brightness = 205
Brightness = 204
Brightness = 162
```

- whileループの前に2つの空のリスト**times**(時間)、**brights**(明るさ)を作成します。

```
times = []
brights = []
```

角かっこは、キーパッド[ctrl]+左括弧または[menu] > Built-ins > Lists (メニュー>組み込み>リスト)により入力します。

また、whileループの前に、0から開始する時間カウンター変数(t)を追加します。

```
t = 0
```

timeモジュールがあるため、'time'という単語を変数として使うことは避けてください。また、リスト名には多くの値が含まれているため、複数形にすることをお勧めします。

```
*bright.py 5/7
from microbit import *
times=[]
brights=[]
t=0
while get_key() != 'esc':
    bright=display.read_light_level()
    print("Brightness = ",bright)
```

- whileループ本体のprintステートメントの後に、タイマー変数tを増やすステートメントを追加します。光の読み取りの間に1秒の時間間隔を使うので、次を使います。

```
◆◆t = t + 1
```

```
*bright.py 8/8
from microbit import *
times=[]
brights=[]
t=0
while get_key() != 'esc':
    bright=display.read_light_level()
    print("Brightness = ",bright)
    t = t + 1
    I
```

- 次のステートメントを使って、bright値(明るさ)とt値(時間)をそれぞれのリストに追加します。

```
◆◆times.append(t)
◆◆brights.append(bright) ←“s”に注意!
```

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

これらのステートメントは、現在のbright値とt値をリストに追加します。

```
*bright.py 11/11
from microbit import *
times=[]
brights=[]
t=0
while get_key() != 'esc':
    bright=display.read_light_level()
    print("Brightness = ",bright)
    t = t + 1
    times.append(t)
    brights.append(bright)
```

7. サンプリングのタイミング間隔を制御するには、次を追加します。

◆◆**sleep(1000)**

2つの**append**ステートメントの後に置きます。これにより、サンプル間で1秒間データ収集が一時停止します。

Sleep()は次の場所にあります：**[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)

micro:bitを使う場合、**sleep()**は引数にミリ秒を使うことを思い出してください。このサンプリングは1秒に1回行われます。

```

1.6 1.7 1.8 *SB3 RAD 11/11
*bright.py
from microbit import *
times=[]
brights=[]
t=0
while get_key() != "esc":
    bright=display.read_light_level()
    print("Brightness = ",bright)
    t = t + 1
    times.append(t)
    brights.append(bright)
    sleep(1000)
    
```

8. **while**ループ本体の後に(インデントは不要です), PythonリストをTI-Nspireリストに格納します。両方の環境で同じ名前を使います。

新しい行の先頭から開始します(インデントはありません)。

[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)で、**store_list()**を見つけます。引用符で囲まれた“TI-Nspire list name”(TI-Nspireリスト名)と、引用符で囲まれていないPythonリスト名の2つの引数を入力します。ツールチップのプロンプトに従います。

```

1.6 1.7 1.8 *SB3 RAD 14/14
*bright.py
t=0
while get_key() != "esc":
    bright=display.read_light_level()
    print("Brightness = ",bright)
    t = t + 1
    times.append(t)
    brights.append(bright)
    sleep(1000)
store_list("times",times)
store_list("brights",brights)
    
```

このプログラムでは、Pythonリストは**[esc]**を押してループを終了したとき、プログラムが終了する直前にのみTI-Nspireリストに保存されます。

9. プログラムを実行(**[ctrl] [R]**)します。光源に近いmicro:bitから始めます。露出した電球やスマートフォンのフラッシュライトがうまく機能します。明るさの読み取り値が10未満になるまで、ゆっくりと、しかし着実にmicro:bitを光から一定の速度で遠ざけます。**[esc]**を押してプログラムを終了します。

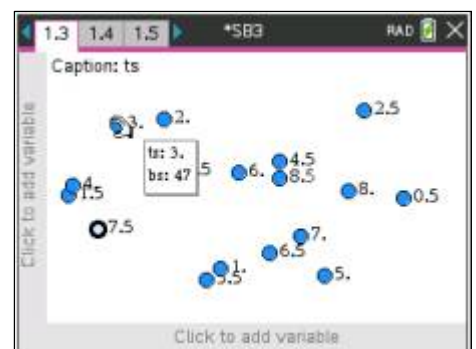
良いデータだと思うまで、このプロセスを繰り返します。

```

1.1 1.2 1.3 *SB3 RAD 492/508
Python Shell
>>>219
219
214
124 +
72
47
33
25
21
19
15
    
```

10. プログラムが終了したら、**[ctrl] [doc]**を押してドキュメントにページを追加します。'Add Data & Statistics(データと統計を追加)'を選択します。

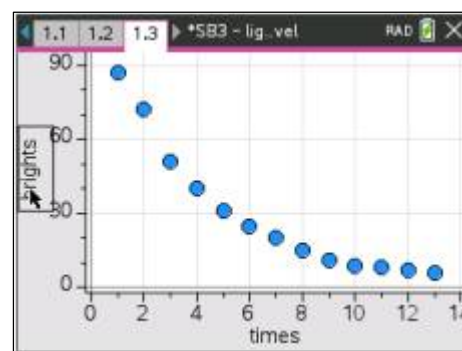
右図のようなページが表示されます。リストの1つが表示され、点が画面に散らばっています。プロットを整理する必要があります。



11. 画面下部にある'Click to add variable(クリックして変数を追加)'をクリックし、リストtimesを選択します。これは独立変数です。

画面左側にある'Click to add variable(クリックして変数を追加)'をクリックし、リストbrightsを選択します。これは従属変数です。これにより、右図のように散布図は整理されます。

ここに示すような滑らかなプロットを得るには、プログラムを数回実行する必要があるかもしれません。別データになると、ウィンドウを調整する必要があるときもあります。



サンプル間の時間間隔を調整することもできますが、必ず `sleep(1000)` 引数とカウンター値(`t=t+1`)の両方を編集してください。

何か気づきましたか。

- どのようなパターンが見られますか。この物理現象に適する数学モデルは何ですか。
- TI-Nspireのデータ分析ツールを使って、データに適した数学モデルを決定します。

Teacher Tip: このレッスンでは、光源からの明るさの逆2乗の法則を調べましたが、この法則は、時間ではなく光源からの距離と光の強度の関係です。

$$\text{強度} = k / (\text{距離}^2), \text{ kは定数}$$

光源から離れるmicro:bitの一定の動きは、時間と距離の間に線形関係を確立するので、逆2乗の式は適用可能です。

アクティビティを変更して、光源からmicro:bitまでの距離測定を使い、キーを押して各データ点を収集できます。たとえば、収集の瞬間ごとに、micro:bitを光源から10cm離します。`get_key(1)`を使って、micro:bitが次のサンプルに適切に配置されるまでプログラムを一時停止します。