Unit 7: TI-RGB Array

Skill Builder 1: それらを照らす

このレッスンでは，TI-RGB Array上の16個のLEDをまとめて(一度に)，および個別に(一度に1つずつ)制御する方法を学習します。

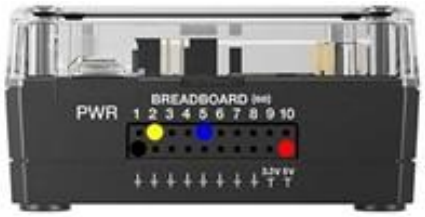
Note: このユニットでTI-RGB Arrayのlight(ライト)を再現するため使われるシミュレーションは，デモンストレーション目的でのみ使用されます。このツールはTI製品ではなく，TIから購入や配布はされません。
Caution: ライトが速く点滅することは，一部の学生にとっては邪魔になることがあるため，**sleep()**ステートメントを使って速度を少し遅くすることをお勧めします。

目標

- すべてのLEDを点灯し，ループを使って同時に点滅
- 別のループを使ってLEDを一度に1つずつ点灯および消灯

TI-RGB Array

バック



ブレッドボードポート

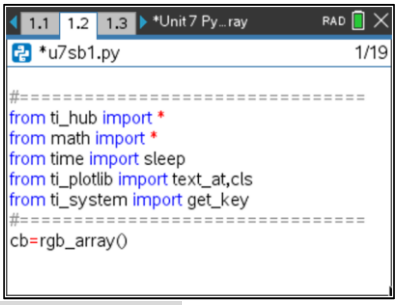
TI-RGB Arrayは16色のLEDとコントローラーチップを備えた回路基板であり，短い4線ケーブルが付属しています。TI-Innovator Hubのブレッドボード(BB)ポートを使って，TI-Innovator Hubに接続します。 回路基板の背面にある配線手順に従って，回路基板をTI-Innovator Hubに接続します。また，TI-Innovator HubはTI-Nspire CX IIに接続します。

Teacher Tip: TI-RGB Arrayデバイスは，オブジェクト指向プログラミング(OOP)の世界に少し深く入ります。TI-Nspire CX II PythonのTI-Innovator Hubの実装は，クラス(オブジェクトの定義)を使って設計されています。このコースのユニット1，2，3で説明したように，オンボードアクセサリ(光，色，音，明るさ)には，**ti_hub**モジュールですでに定義されているオブジェクトがあります。例：**light.on()** - lightはオブジェクトの名前であり，**on()**はそのクラスの定義内のメソッドまたは関数です。
TI-Innovator Hubに接続されているすべての外部アクセサリでは，そのクラスのメソッドを使う前に，プログラマがインスタンス変数(オブジェクト)を作成する必要があります。クラス名はTI-Innovator Hubメニューから選択されますが，メソッドはメニューにまったく含まれていません。入力すると(表示されるように)，エディタに現れます。

1. Hub Projectテンプレートを使って新規のPythonプログラムを開始します。

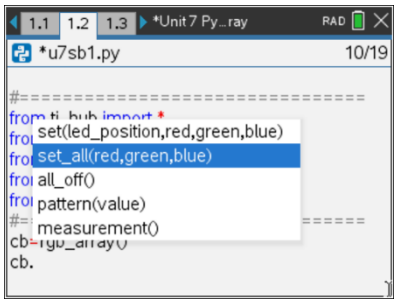
   menu > TI Hub > Add Output Device(メニュー>TI Hub>出力デバイス追加)を押して，TI-RGB Arrayを選択します。

   varプレースホルダー(仮に確保した場所)の代わりに，任意の変数名を入力します(右図参照)。私たちは，**cb** (circuit board(回路基板)用)を使いました。



Teacher Tip: 一般的な変数名はrgbですが，一部の学生にとっては混乱を招く恐れがあります。

2. ライトを一度に点灯させるには，プログラムの次の行に変数名の後にピリオドまたは小数点を入力します。

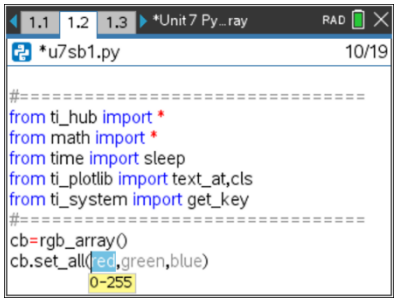


cb.

rgb_array()クラスで使えるすべてのメソッド(関数)を示すダイアログボックスがエディタにポップアップ表示されます。変数**cb**はそのクラス(オブジェクト)のインスタンスであり，これらのクラスメソッドのいずれかを使います。

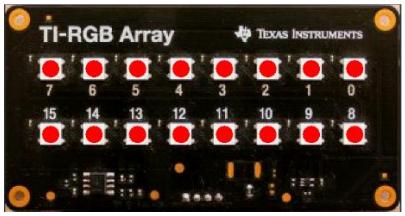
図のように，**set_all(red, green, blue)**を選択します。

3. ステートメントは，変数名，ピリオド，ポップアップリストから選択した関数で構成されます。
Pythonメニューの他の多くのコマンドと同様，このコマンドには3つのインラインプロンプト(red, green, blue)が含まれ，それぞれに許可された値(0～255)を示すツールチップがあります。



Teacher Tip: .measurement()関数は，アレイボードによって消費されている電流(mA)を返します。電流は，点灯しているLEDの数と，読み取り時に使われている色によって異なります。このため，TI-RGB Arrayは'TI Hub > Add Input Device'(TI Hub>入力デバイス追加)メニューにも表示されます。これらのレッスンでは，この機能については説明していません。

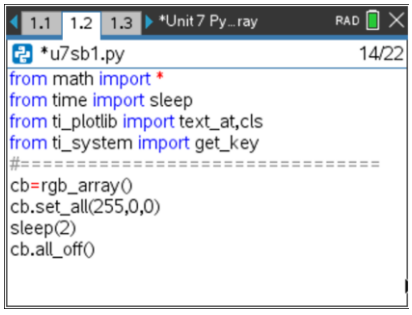
4. 3色の値を選択して入力します。



プログラムを実行して，16個のLEDすべてが自分の色で点灯することを確認します。
プログラムが終了した後も，LEDは点灯したままであることに注意します。

5. LEDをオフにするには，次のステートメントを使います。

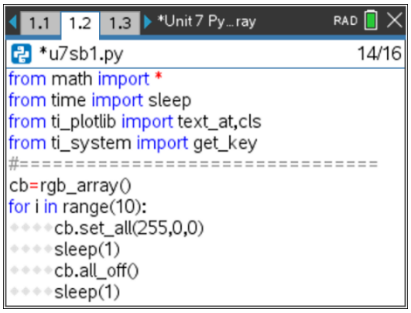
cb.all_off()



変数名とピリオドをもう一度入力し，リストから**all_off()**を選択します。
そして，点灯と消灯の間に**sleep(2)**(秒)を追加します。そうしないと，何も表示されません。

ここでプログラムを実行すると，LEDは2秒間点灯したままになります。

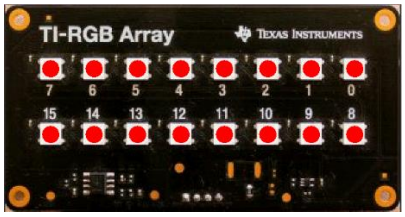
6. LED制御ステートメントを**for**ループに入れて，数回点滅させます。それらがオフにされた後，別の**sleep()**を追加します。物事を少しスピードアップするためsleep時間を調整したいかもしれません。ループブロック内のすべてのステートメントをインデント(字下げ)してください。



> **Note:** コンストラクターステートメント**cb = rgb_array()**をループブロックに含めないでください。一度だけ定義する必要があります!

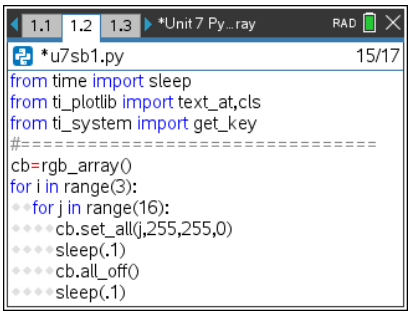
続行する前に，プログラムを実行してテストします。

7. プログラムがすべてのLEDを一度に点滅させたとき，成功です。つぎに，内部ループを使ってLEDを一度に1つずつ制御しましょう。



(demo1.1.gif)

8. **for i...**ループの下に，内側のループを追加します：**for j in range(16):**
4つのループブロックステートメントをすべてインデント(字下げ)して，内側のループに適用されるようにします。
ステートメント**cb.set_all(...)**を削除しますが，空白行は残します。その代わりに，次を入力します。
cb.
そして，**set(led_position, red, green, blue)**を選択します。

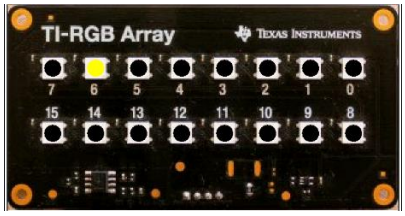


内側のループ変数**j**を**led_position**として使い，色の値を入力します。
スリープ値と外側のループ**range()**を変更して，処理を少し高速化します。

> **Teacher Tip: cb.all_off()**の代わりに，**cb.set(j, 0, 0, 0)**を使えます。

9. プログラムを実行します。これで，16個のLEDが一度に1つずつ3回点灯します。

ドキュメントを保存することを忘れないでください。



(demo1.2.gif)