

Unit 5: TI モジュール

Application: Move it! (それを動かせ!)

この応用では、電卓の矢印キーを使って画面上のオブジェクトを移動することにより、インタラクティブなグラフィカルプログラムの核心を構築します。

目標

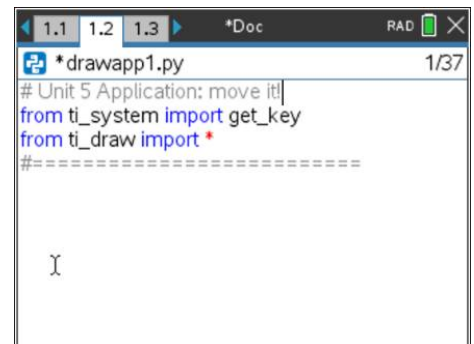
- リッチで魅力的なグラフィカルプログラミングプロジェクトで、このコースのスキルと概念を組合せます。

キーを押すことに反応し、画面上でキーの方向に移動するものを描画するプログラムを作成します。



1. キーの押下を監視するには `get_key()` が必要であり、動き回る何かの画像を作成するには描画関数が必要です。

空白のPythonファイルで、`ti_system`と`ti_draw`の2つをインポートします。または、テンプレートの1つを使って、不足しているインポートを追加します。



2. 押されたキーを知り、操作できるようにするため、キー値を変数 `k` に格納します。

`k = get_key()`

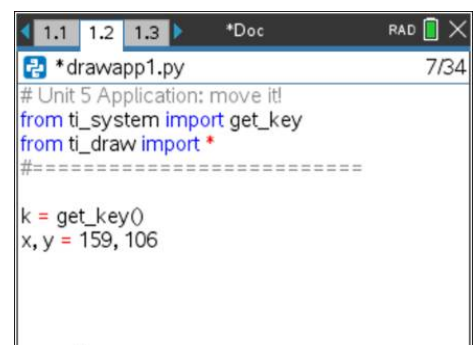
画面中央からオブジェクト(円)をスタートします。

`x=159`

`y=106`

これは次のように書くこともできます：`x,y = 159,106`

矢印キーを押すと、これらの値の1つが変更され、オブジェクトが新しい位置に再描画され、移動しているように見えます。

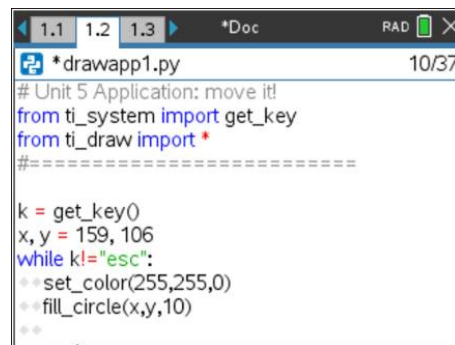


Teacher Tip: 形を描いたり消したりする関数を書くように学生に勧めることを検討してください。単純なerase(消去)ステートメントは、画面全体をクリアする`clear()`です。また、`clear(x,y,w,h)`は長方形の領域をクリアします。

3. メインループを開始します。

```
while k != "esc":
```

カラフルな円を作成します。好きな色と半径を使ってください。



```
1.1 1.2 1.3 *Doc RAD 10/37
*drawapp1.py
# Unit 5 Application: move it!
from ti_system import get_key
from ti_draw import *
#=====
k = get_key()
x, y = 159, 106
while k!="esc":
    set_color(255,255,0)
    fill_circle(x,y,10)
```

Teacher Tip: 色と半径の値は任意です。

4. 押されたキーを確認します。

```
if k == "up":
    y = y - 10          (または y -= 10)
```

これは、上矢印が押されたとき、y値から10を引くことを意味します。これにより、円が画面上で10ピクセル上に移動します。

他の3つの矢印キー(down(下), left(左), right(右))のifステートメントを記述し、xとyを適当に変更します。

すべてのifステートメントは、whileブロックの一部です。

最初のk = get_key()ステートメントはループの一部ではないため、ブロックの下部でwhileループ内のキー押下(k = get_key())を再度読み取ります。

完了したら、プログラムをテストします。

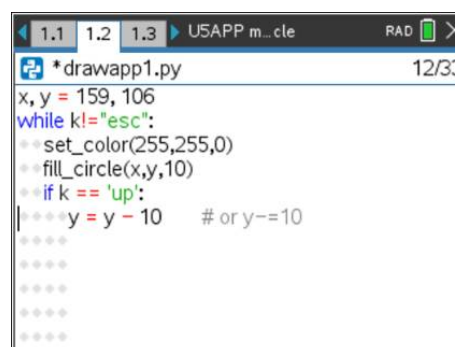
5. 矢印キーを使って円を移動します。円は動きますか、もしそうなら、おめでとうございます。

しかし、それは動いた跡を残します。

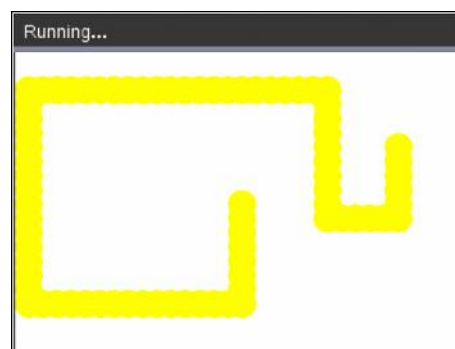
次の2つを使って、新しい円を描く前に古い円を消去します。

```
set_color(255,255,255)
fill_circle(x,y,10)
```

xとyの値を変更する前です。



```
1.1 1.2 1.3 USAPP m...cle RAD 12/33
*drawapp1.py
x, y = 159, 106
while k!="esc":
    set_color(255,255,0)
    fill_circle(x,y,10)
    if k == 'up':
        y = y - 10 # or y-=10
```



6. 円はいつも黄色で描画され、すぐに白で描画されるため、点滅します。黄色の円が描かれたときのみ画面を更新する必要があります。

そのため、プログラムの先頭(whileの前)に、次のステートメントを置きます。

use_buffer()

これは、TI Draw > Controlにあります。

```

1.1 1.2 1.3 *Doc RAD 5/32
drawapp1.py
# Unit 5 Application: move it!
from ti_system import get_key
from ti_draw import *
=====
use_buffer()
key = get_key()
x, y = 159, 106
while key!="esc":
    set_color(255,255,0)
    fill_circle(x,y,10)
    paint_buffer()
    
```

そして、黄色い円が描かれた直後に、次のステートメントを配置します。

paint_buffer()

プログラムを再実行してください。

良くなりましたか。画面は、黄色の円が描かれた後にのみ、再描画されます。

Teacher Tip: use_buffer()と paint_buffer()についてひと言：

これらの2つの機能により、画面の描画がいつ更新されるかをより正確に制御できます。

use_buffer()は、画面の外にあるメモリの一部であるメモリバッファにデータを描画するようコンピュータに指示します。

paint_buffer()は、バッファ内のすべてを画面に転送するためのシグナルです。したがって、黄色の円を消去して新しい円を描画することは、画面の外で行われ、一度にペイントされるため、表示されるのは黄色の円(別の場所)だけです。

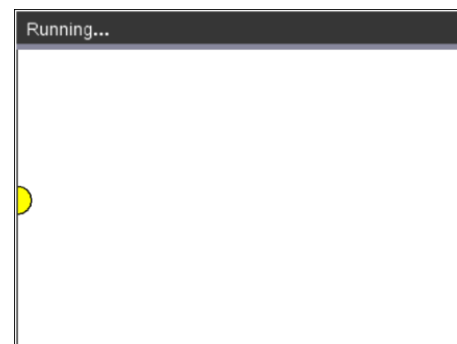
Note: バッファ(buffer)とは、衝撃を吸収するもの、すなわち緩衝を意味します。

7. 最後に(今のところ)、円が画面から消えるとどうなりますか。それはただ進み続けます。画面の端を検出して動作するように、より多くのifステートメントでプログラムを変更します。ここには2つのオプションがあります。

- 端に着いたら、そこにとどまり、画面から離れない。
($x < 0$ のとき、 $x = 0$ にする、など)
- 画面の一方の端から離れるときは、反対側に折り返す。
($x < 0$ のとき、 $x = 317$ にする、など)

幸運を！

追加課題：キーが押されていないときでも、円が最後の方向に移動し続けるようにプログラムを変更します。



このプロジェクトのサンプルコードが次ページにあります。



Teacher Tip: このプロジェクトのサンプルコード :

```
# Unit 5 Application: move it!
from ti_system import get_key
from ti_draw import *
#=====
use_buffer()
key = get_key()
x,y = 159,106
while key!="esc":
    set_color(255,255,0)
    fill_circle(x,y,10)
    set_color(0,0,0)
    draw_circle(x,y,10)
    paint_buffer()
    set_color(255,255,255)
    fill_circle(x,y,10)
    draw_circle(x,y,10)
    if key == 'up':
        y = y - 10 # or y-=10
    if key == 'down':
        y+=10
    if key == 'left':
        x-=10
    if key == 'right':
        x+=10
    if x<0:x=317
    if x>317:x=0
    if y<0:y=211
    if y>211:y=0
    key = get_key(0)
```