Getting Started with TI Draw — The Shield

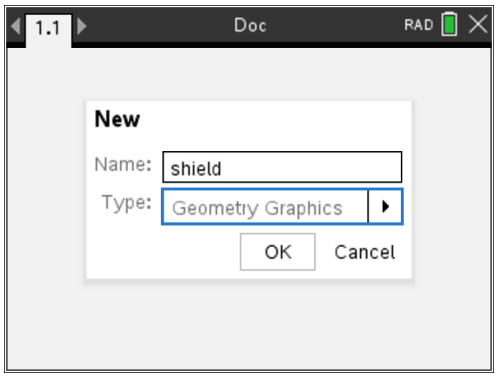
Computer graphics is the most fun of all programming projects. We introduce the **ti_draw** module by making the **CapTIan**'s Shield using some of the tools in the module. Be warned that this project uses some trigonometry, but can easily be revised to avoid the trig functions.

Introduction: Our superhero, the **capTIan**, has the powerful shield seen here. Lets' make the shield using Python and the **ti_draw** module.

1. Start a new program using the Type: **Geometry Graphics** template. This template provides the **import** statement:

   from ti_draw import *

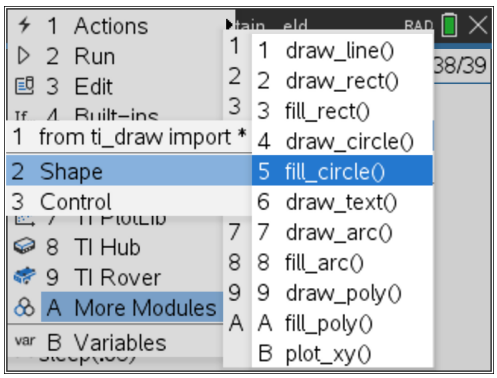
   which gives access to many drawing functions such as circles, segments, and rectangles.



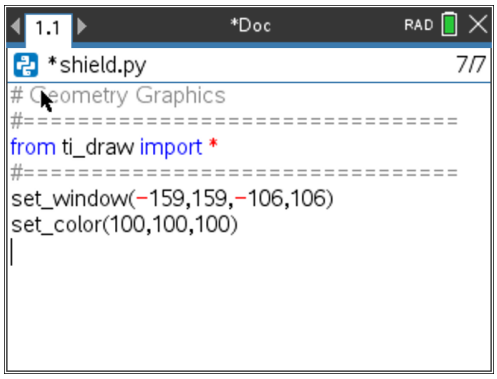
2. The **ti_draw** module contains functions that let you create graphical images on your TI-Nspire CX II screen. The graphics 'canvas' is a separate display from the Python Shell and is activated when any of the **ti_draw** functions are processed. The canvas disappears at the end of the program.

   Use **[menu] More Modules > TI Draw** to access all these graphics tools on two submenus: **Shape** and **Control**. Visit them now.

   As long as the drawing 'canvas' is showing (even if the program is 'Finished'), the TI-Nspire is locked. When the program displays 'Finished' at the top of the screen, press any key to close the canvas.



3. The first two functions to use are **set_window()** and **set_color()**. Both are found on **[menu] More Modules > TI Draw > Control** Add the arguments shown here:



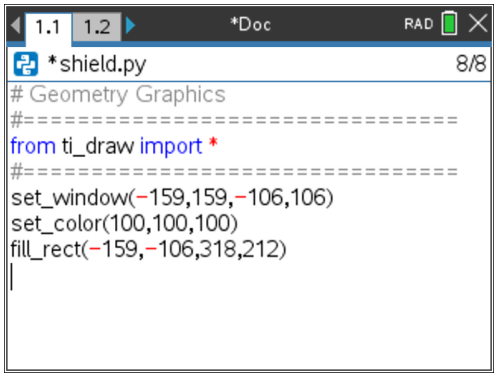
set_window(-159, 159, -106, 106)

Since the screen is 318x212 pixels, this window places the origin near the center of the screen and makes each pixel represent one graphical 'unit'.

set_color(100, 100, 100)

This function establishes the drawing color using three values: red, green, and blue, each between 0 and 255. (100,100,100) is a light gray.

4. To make the gray background, use the function:



fill_rect(-159, - 106, 318, 212)

found on **[menu] More Modules > TI Draw > Shape**

(-159, -106) is the lower left corner of the screen (remember our window setting). 318 is the width of the screen and 212 is the height of the screen. *When using a* **window** *setting, width is measured left-to-right and height is measured* ***bottom-to-top****.*

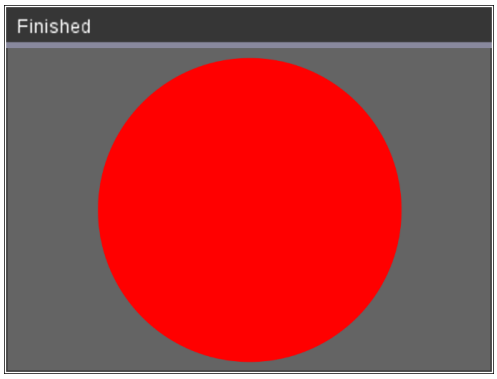
Run the program now to see the gray screen. Press **any key** to close the canvas and return to the Shell. Then press **ctrl_leftarrow** to return to your Editor.

5. Make the red and black circles next, from outermost to innermost so that the smaller ones cover part of the interior of the larger ones. The centers of the circles are all (0,0) and the largest radius that will keep all the circles on the screen is 106 (the distance from the origin to the top and bottom of the screen). Our radii will be 100, 80, 60 and 40 for the 'bulls-eye' in the center.

   First use **set_color(**r, g, b**)** to establish the color of a circle then use **fill_circle(**x, y, r**)** to make the circle.

```
set_color(255,0,0)  # red
fill_circle(0,0,100)  # largest circle
```

   results in the red circle shown here.



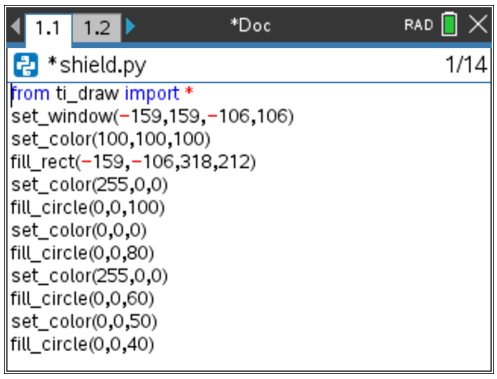
6. Make two more circles: a black one and another red one, using radii of 80 and 60. Use **set_color( )**, too.

7. Make a dark blue center circle with a radius of 40:

```
set_color(0, 0, 50)   # dark blue
fill_circle(0, 0, 40)   # bulls-eye
```

   The innermost circle here may appear black.

8. Our code so far…



9. Now comes the gold star, where **Trigonometry** comes into play!

10. The 5-pointed star shown here is a **decagon** (10 vertices, 10 sides). Five of the vertices are on the last circle you made. The other five vertices are all on a smaller circle shown here in gray. To make the star we will use **fill_polygon()** which takes two lists as arguments: a list of x-coordinates and a list of y-coordinates of the vertices of the polygon.

    So, the next task is to make those lists…

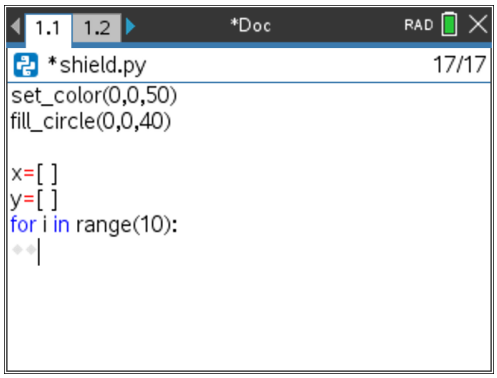
11. Start with two empty lists called x and y:

    x=[ ]
    y=[ ]

    The brackets are found on the **ctrl-left_parenthesis** key. We will add values to these lists next using a for loop:

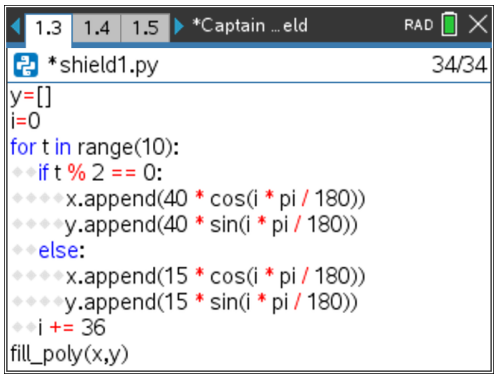
    for i in range(10):

    found on **[menu] Built-Ins > Control**



12. Here is the complete polygon code.

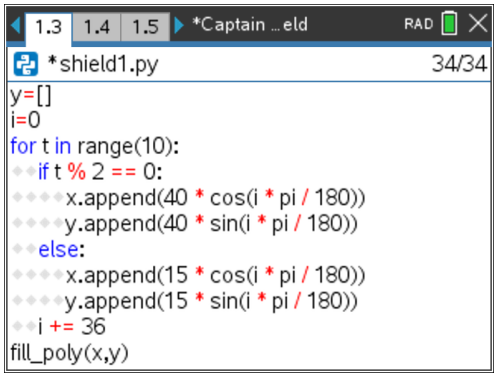
```
i = 0  # angle measure
for t in range(10):
  if t % 2 == 0:
    x.append(40 * cos(i * pi / 180))
    y.append(40 * sin(i * pi / 180))
  else:
    x.append(15 * cos(i * pi / 180))
    y.append(15 * sin(i * pi / 180))
  i += 36
fill_poly(x,y)
```



see the next step for the explanation…

13. Let's analyze one of these statements:

```
if t % 2 ==0:
  x.append(40 * cos(i * pi / 180))
```



if t % 2 == 0: accounts for every *other* vertex being on a different circle.

x.append(adds a value to the end of the x-list.

40 is the radius of the *outer* vertices.

The variable **i** represents the angle, measured counter-clockwise starting from 'east'.

cos(i * pi / 180) and **sin(i * pi / 180)** calculate the x- and y-coordinates of the point on the *unit* circle that is **i** degrees around the circle. It is multiplied by **pi / 180** to convert degrees to radians which is the domain of the Python trig functions.

Add **from math import *** at the top of your program to gain access to **sin**, **cos** and **pi**.

At the bottom of the loop, the variable **i** is increased by **36** degrees (1/10 of 360).

The **else:** block calculates the *inner* vertices that will have a radius of **15** instead of 40. Why 15? Try other values here to see the impact.

The **for** loop calculates the coordinates of all 10 vertices of the star and places them into the two lists **x** and **y**.

After the loop, **fill_poly(x, y)** draws the polygon. But… add a **set_color()** function just before the **fill_poly()** function. We used gold:

```
set_color(255,215,0)
```

14. When you run the program you will see the gold star in the center, but it is 'tilted' because the *starting point* is facing to the right. To get the star to point upward, change the initial value of **i** before the **for** loop begins to:

    i = 90

15. Your star now points upward.

In the next section you will make the star spin!

16. **Part 2: Spin the star!**

    Animation is produced in the CX graphics canvas by redrawing a section of the screen. Since the star is in the smallest circle it is only necessary to redraw that circle with the star in a different position. We can accomplish this animation using the loop

    while get_key() != 'esc':

    found in the **ti_system** module so you will need to add

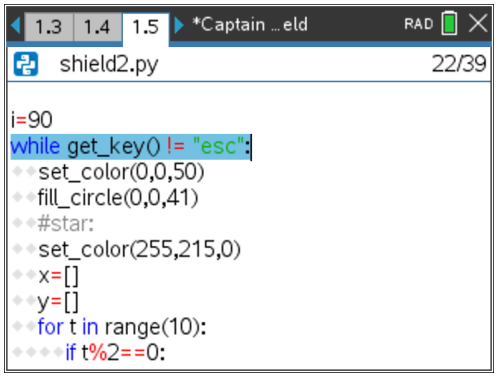
    from ti_system import *

    to the top of your program.

    The **while** statement goes *above* the code for the innermost (blue) circle.

    Indent the entire polygon routine to become the body of the **while** loop.

    Move the initial value of **i=90** before this **while** statement.

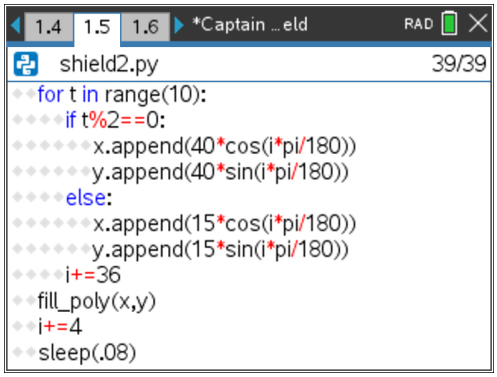


17. After the polygon is drawn, add a small amount to the value of **i**. We use

    i += 4

    This causes the next polygon to start 4 degrees counterclockwise from the last one. You should try other values, too.

    Perhaps add a **sleep( )** function in the loop body to slow the rotation. If you do, remember to add **from time import *** to your program.



18. If all is well, then the star spins counter-clockwise. Press **[esc]** to end the program.

    Go back into the code and edit some of the numbers to see the effect each has on the **capTIan's shield**.

    Challenges:
    - Can you make the star spin clockwise?
    - Can you change the star to a six-point star?
    - How about an n-point star?

(the star is spinning)