Getting Started with Turtle Graphics — Polygons

Turtle Graphics is a TI-developed Python module ideal for folks new to graphics programming. Create simple or complex patterns, shapes, geometric forms and more in an entertaining, active, pseudo-physical learning experience.

---

0. **Background:** "Turtle Graphics" is a concept originally introduced into the programming language **Logo** created by Feurzeig, Solomon, and Papert in the mid-1960's. The 'turtle' is a graphical object that responds to programming commands like **forward(***distance***)** and **right(***angle***)**. The turtle module is object-oriented and the basic turtle functions are included on the turtle menus.

In the image to the right, the blue and yellow triangle represents the 'turtle'. The turtle carries a 'pen' that can leave a trail behind as the turtle moves. The black segments are the path that the turtle followed.

To use turtle graphics on your **TI-84 Plus CE Python**, you must have the turtle module (**TURTLE.8xv**) on your calculator. Use TI-Connect CE to install the file to your calculator. The free turtle module and documentation are available at

https://education.ti.com/en/product-resources/turtle-module/ti84ce-python

This Unit assumes that you are using OS version 5.7 (or later) which includes a special 'soft key' to access some special TI-developed **<Add-On>** modules.

1. Begin a new Python program. Ours is named **TA**. (**T**urtle program **A**)

Select **<Fns…> Modul**. At the bottom of the screen there is an **<Add-On>** soft key. Select **<Add-On>** and select

from turtle import *

You also get the statement **t = Turtle( )** in your code which creates a Turtle *object* variable named **t**.

2. All turtle statements (methods) by pressing **[math] turtle…**
Note that there are five sub-menus at the top of the screen:

Move Draw Pen Settings State

each contaning specific turtle functions. All turtle functions begin with 't.' which is the default turtle name. You can change the turtle name (and even have multiple turtles in the same program) but you are then responsible for editing the turtle name. The **Move** menu is shown here. Check out the items on the other menus.

3. To get the turtle to make a square, write a **for** loop that contains the two statements

t.forward(50) # pixels
t.left(90) # degrees

After the **for** loop add the statement

t.done()

which keeps the graphics on the screen until you press the **[clear]** key.

4. After entering the code, <Run> the program to see the turtle make a square.

The turtle starts at the origin facing to the right. '**50**' is 50 pixels on the screen. The grid dots are 25 pixels apart.. '**90**' represents 90 degrees.

At the end of the program, the turtle is back at the origin facing to the right. The graphics remain on the screen until you press the **[clear]** key.

Note: If you do not see the axes and the grid points, install the appvar **GRID.8xv** *supplied with the* **TURTLE.8xv** *module onto your calculator.*

5. Modify the square program to let the user enter the number of sides so that the turtle can draw a regular polygon. Use an input statement:

n=int(input("num of sides? "))

and alter the loop code.

The tricky part is to figure out what the turning *angle* should be when you know the *number of sides*. The user entered **7** sides to produce this screen showing a septagon.

6. Change the speed of the drawing using

t.speed(0) (fastest)

found on **[math] turtle… Settings**

The complete polygon code is shown here.
Note that the **input** statement precedes the **t = Turtle()** statement because the constructor puts the screen into the graphics mode.

7. **Adding Color**

There are two color settings on the **[math] turtle…** menus:

t.pencolor(r, g, b) (on the **Pen** submenu)

t.fillcolor(r, g, b) (on the **Draw** submenu)

You can change **pencolor( )** on the fly, so each side of the polygon can be a different color (not shown). But **fillcolor( )** must be set before drawing a shape. See the red hexagon example in the next step…

8. To make the red hexagon in the previous step:
Set the **fillcolor(, ,)** *before* making the polygon and issue the **t.begin_fill()** command.
During the drawing process the system keeps a list of vertices visited.
After the polygon is drawn use the **t.end_fill()** method to fill the polygon.

9. Now modify the program to make *randomly* colored polygons at *random* places on the screen until the **[clear]** key is pressed.
   - Use **while not escape():** for the main loop.
   - Use **t.hideturtle()** and **t.hidegrid()** to improve the appearance.
   - Use **t.penup(); t.goto(,); t.setheading()** and **t.pendown()** to place the turtle at a *random* starting point on the screen facing in a *random* direction.

The turtle screen ranges from -159 to 159 (pixels) horizontally and -106 to 106 vertically and it's OK if the turtle goes off the screen.
The heading can be any angle (degrees).
Use the **for** loop to make each ploygon.

10. Some of the code is shown in this image.
Get creative!

Teacher Tip: The free turtle module and documentation are available at

https://education.ti.com/en/product-resources/turtle-module/ti84ce-python

No Rover? No problem! As seen on the above page, the turtle 'drives' like the Rover in many ways. But it lacks the sensors and TI-Innovator ports for making physical programming more interactive. The turtle module is based on the standard Python API but not all functions are included.

If you are using **TI-SmartView CE** version **6.0** you will have emulator **OS v5.7** which includes the **<Add-On>** soft key but you still need to add the file **TURTLE.8xv** using the Explorer workspace.