



Getting Started with Turtle Graphics

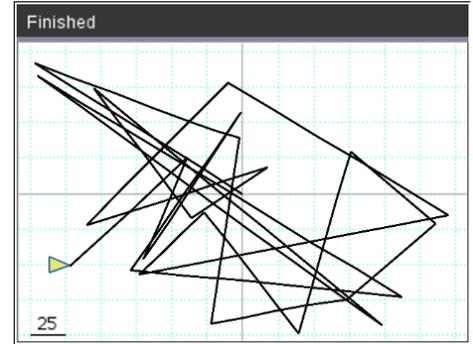
Polygons

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

Note: This document was based on the turtle module v3.2.1. Future modules may have additional features.

- 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 small blue & yellow triangle represents the turtle object. 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-Nspire CX II**, you must have the turtle module **turtle.tns** in your **PyLib** folder on your device. The free turtle module and documentation, including installation instructions, are available at

<https://education.ti.com/en/product-resources/turtle-module/nspire-python>

- 1. Begin a new, blank Python program. Ours is named **turtle1**.

Press **[menu] > More Modules > Turtle Graphics** and get the statement

from turtle import *

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



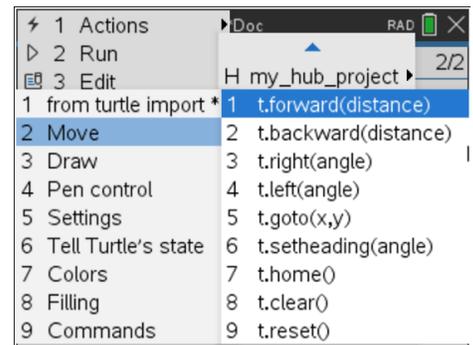
If Turtle Graphics does not appear on the More Modules list, then the turtle module is not installed or there are too many items (> 20) for the menu to display. Go back to step 0 and read the directions.

- 2. See **[menu] More Modules > Turtle Graphics** again.

There are eight sub-menus (2..9):

Move Draw Pen Control Settings Tell Turtle’s State Colors Filling Commands

Each contains specific turtle functions (and more). All turtle functions begin with object variable **t**. which is the default turtle name. You can change the turtle name (and even have multiple turtles in a 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 just two statements:

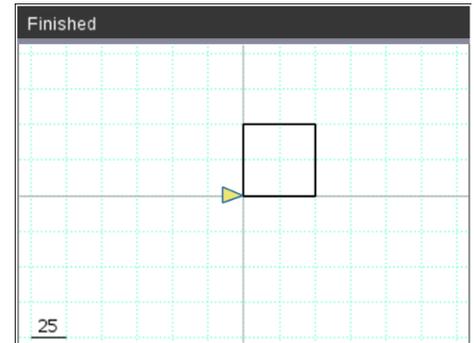
```
for i in range(4):  
    ♦♦ t.forward(50) # pixels  
    ♦♦ t.left(90) # degrees
```

```
1.1 1.2 **Python...OS6 RAD 8/9  
*test.py  
from turtle import *  
t=Turtle()  
  
for i in range(4):  
    t.forward(50)  
    t.left(90)
```

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 lines are 25 pixels apart as indicated by the legend in the lower left corner of the screen. '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 a key.

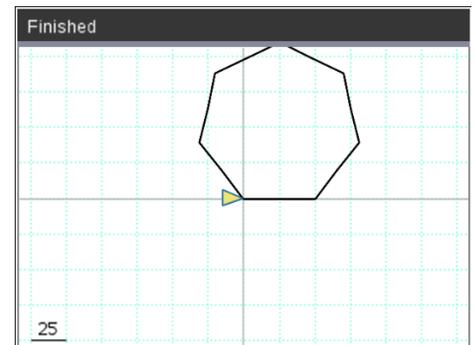


5. Modify the square program to let the user *enter* the number of sides so that the turtle can draw a regular polygon. Use an **input** statement:

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

and change 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.





- Change the speed of the drawing using

t.speed(0) (fastest)

found on [menu] > More Modules > Turtle Graphics > Settings

The complete polygon code is shown to the right.

Note that the input statement must come before the t=Turtle() constructor because the constructor creates the graphics screen.

```

1.1 1.2 *Doc RAD 2/8
turtle1.py
from turtle import *
n=int(input("Number of sides? "))
t=Turtle()
t.speed(0)
for i in range(n):
    t.forward(50)
    t.left(360/n)
  
```

- Adding color:**

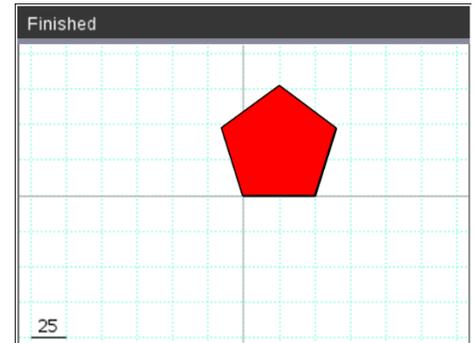
There are two color methods on the **Turtle Graphics** menus:

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

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

You can change **pencolor()** on the fly, so each side of the polygon can even be a different color (*not shown*).

But **fillcolor()** must be set *before* drawing a shape. See this red pentagon example in the next step...



- When selecting **t.pencolor()** from the Pen Control menu you are given a pop-up list of color names from which to choose. You may select one of these colors or... ignore the list (press [esc]), remove the quotes, and enter three integer values (*red, green, blue*) to establish a color:

Use either form of this statement:

t.pencolor("red")

t.pencolor(255, 0, 0)

Only the color "names" on the pop-up list are allowed, but any numeric values from 0 to 255 are permitted.

```

1.1 1.2 1.3 **Python...OS6 RAD 3/9
*test.py red
from turtle import *
t=Turtle()
t.pencolor("red")
for i in range(5):
    t.forward(50)
    t.left(90)
  
```

- To make the red pentagon in the prior step:

Before making the polygon:

set the

fillcolor() (from the turtle **Filling** menu, with the same options as **pencolor**)

and issue the

t.begin_fill() command also found on the **Filling** menu..

During the drawing loop, the system keeps a list of vertices visited.

after the polygon is drawn use the

t.end_fill() method to fill the polygon.

```

1.1 1.2 *Doc RAD 10/10
turtle1.py
from turtle import *
n=int(input("Number of sides? "))
t=Turtle()
t.speed(0)
t.fillcolor(255,0,0)
t.begin_fill()
for i in range(n):
    t.forward(50)
    t.left(360/n)
t.end_fill()
  
```

What value was entered for n to make the pentagon?



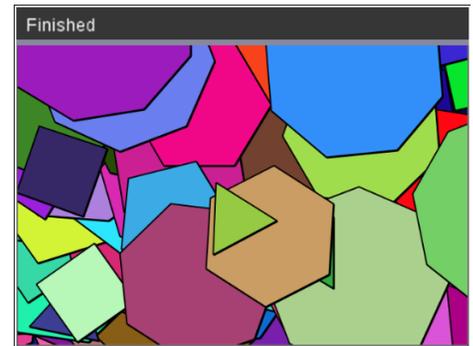
9. Now modify the program to make *randomly* colored polygons at *random* places on the screen until the **[esc]** key is pressed.

- Use **t.hideturtle()** and **t.hidegrid()** to improve the appearance and speed.
- Use **while get_key()!="esc"**: for the main loop.
- 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 with a *random* side length.

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 with a *random* number of sides.



10. Some of the code is shown in this image.

Get creative!

```
1.1 1.2 *Doc RAD X
turtle1.py 13/19
from turtle import *
from random import *
#n=int(input("Number of sides? "))
t=Turtle()
t.hidegrid()
t.hideturtle()
t.speed(0)
while get_key()!="esc":
    n=randint(3,9)
    t.penup()
    t.goto(randint(-150,150),randint(-105,105))
    t.pendown()
    t.setheading(randint(-90,90))
```