Getting Started Guide

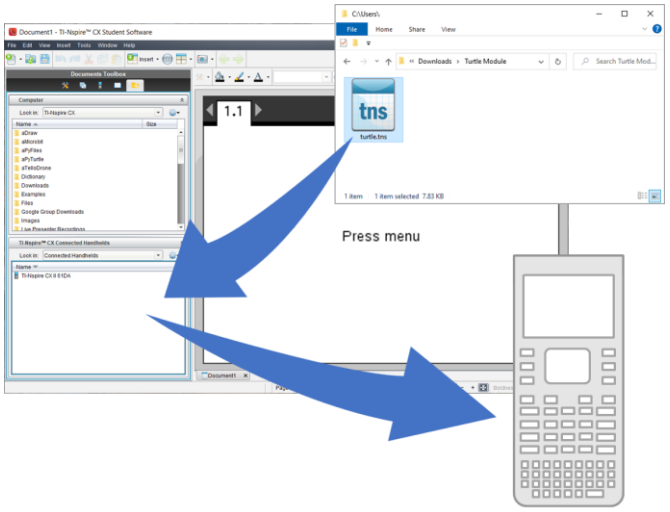
Install the Turtle Module

1) Transfer the Turtle module "tns" file to your TI-Nspire CX II family calculator(s)

After downloading the Zip and extracting files:

- Open your TI-Nspire CX desktop doftware
- Connect your TI-Nspire CX II calculator(s) to your computer using the computer-to-calculator USB cable that comes with the calculator.
- Verify that your connected calculator appears in the Connected Handheld window on the Content Explorer tab of the desktop software.
- Transfer the Turtle "tns" file to the connected calculator by dragging the file into the Connected Handheld window.
- After opening the file on your calculator, go to step 3) **Install the Turtle Module**

This step requires use of TI-Nspire CX desktop software for PC or Mac



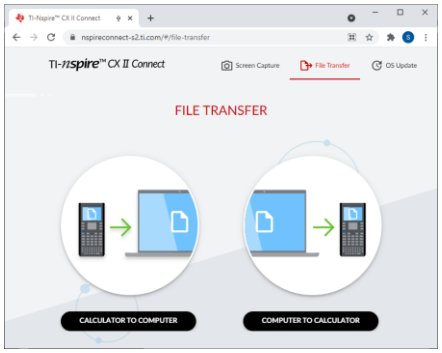
Alternately,

- use the free browser-based TI-Nspire CX II Connect app (https://nspireconnect.ti.com/) to transfer the Turtle "tns" file to a TI-Nspire CX II that is connected to the computer with the computer-to-calculator USB cable.

Note: the Turtle "tns" file may also be transferred calculator-to-calculator via unit-to-unit USB cable linking process prior to installation.

- Go to step 3) **Install the Turtle Module**

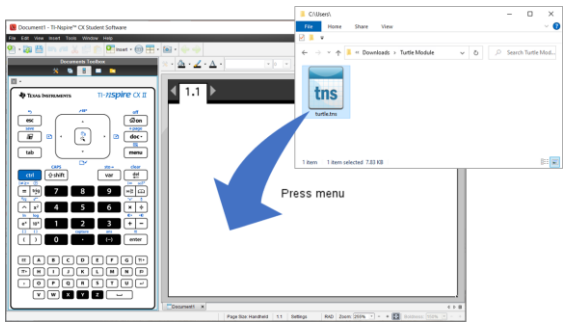
This is an online utility that works with Chromebooks



2) Transfer the Turtle module "tns" file to your desktop software (PC or Mac)

After downloading the Zip and extracting files:

- Drag and drop the Turtle "tns" file onto your TI-Nspire CX desktop software.
- After the file opens, go to step 3) **Install the Turtle Module**

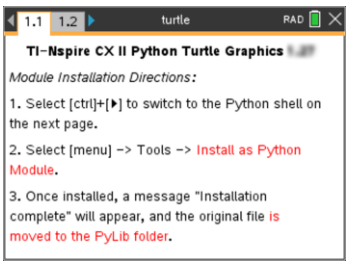


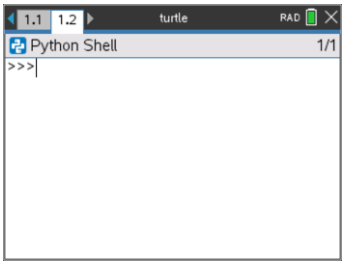
3) Install the Turtle module

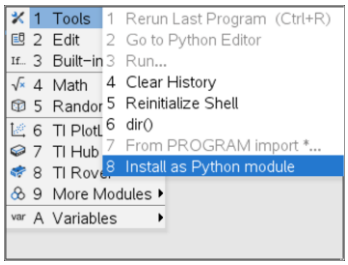
These steps install the module on the handheld calculator. These same steps are also used to install the module on the desktop software.

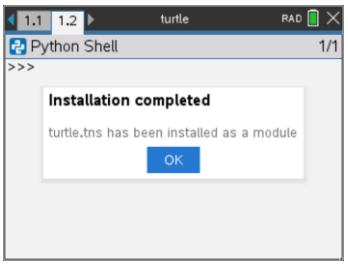
- Open the Turtle file and read instructions on page 1.1
- Press **[ctrl] + [▶]** to switch to the Python Shell on page 1.2.
- Press **[menu]** and select **Tools** > **Install as Python module**
- Once installed, a message “Installation completed” will appear, cofirming istallation.

Note: when installed, the Turtle module file is moved to the Pylib folder.









4) Accessing the Turtle module menu selections

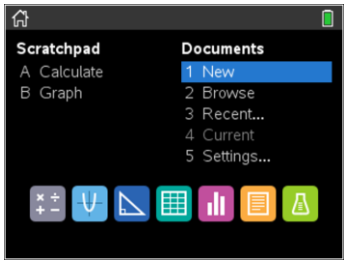
After the Turtle module completes installation, you must create a New document in order to write a Turtle program.

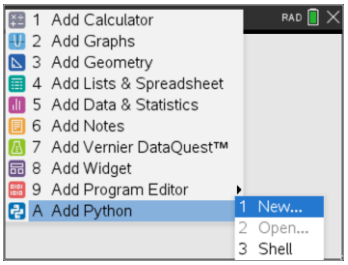
- From the home screen, select **New**
- Select **Add Python**, **New…**
- Give the Python program a name (example: square), OK

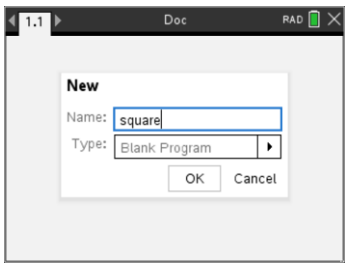
  At this point you are in a Python program Editor page.

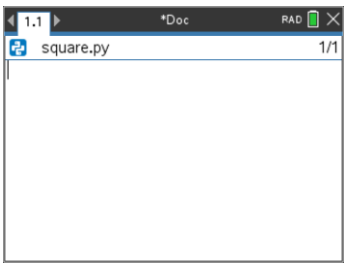
- Press the **[menu]** key then select **More Modules** > **Turtle Graphics**

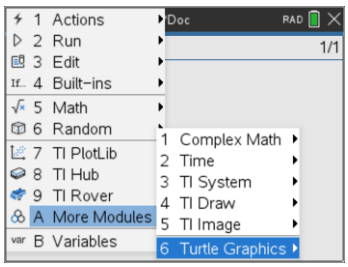
At this point you are ready to begin writing a Python program using Turtle module selections.

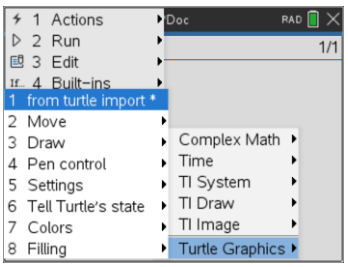












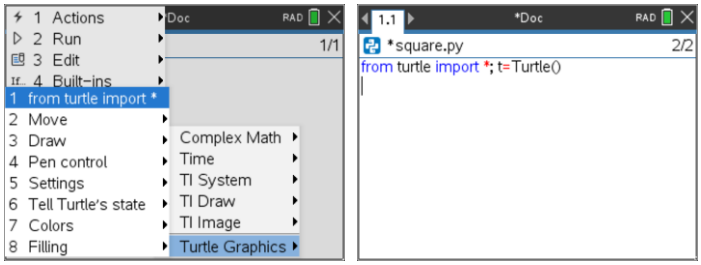
Creating Your First Turtle Program: Let's draw a Square

Continuing from the module menu selections above (while in an Editor page)

1) Select **from turtle import ***

   Pressing **[enter]** will paste the import statement and constructor **t=Turtle()** as the first line in your program. This imports the Turtle module functions and methods into your program and defines your turtle as an object named "t".

   Note: All successive menu selections assume a turtle object named "t". Changing the object name will also require changing the pasted syntax.

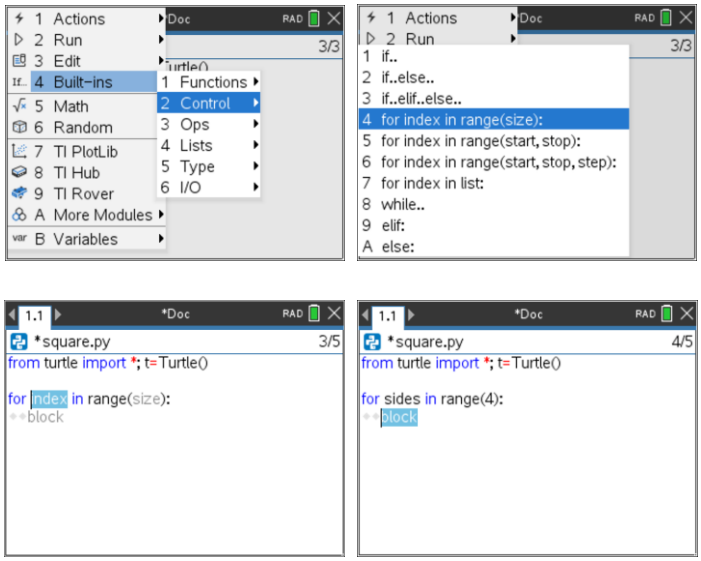


2) Press enter once or twice to advance to line 3.
   Note: line numbers are shown in the top-right corner of the page.

3) Press [menu] and select **Built-ins**, **Control**, and **from index in range(size):**

   Notice the in-line prompt which highlights the word "index". Pressing Tab will cycle through the in-line prompts.

4) Enter "**sides**" for the index and "**4**" for size



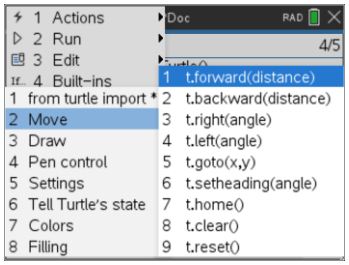
5) With the cursor on the in-line prompt “block”, press **[menu]** and select **More Modules**, **Turtle Graphics**, **Move**, then the **t.forward(distance)** selection

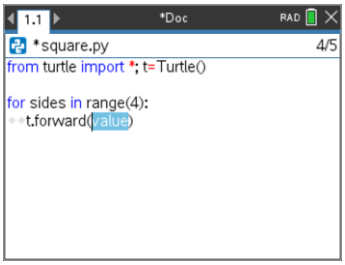
6) Enter the distance you want the turtle to travel forward. In this example, let’s enter 75 units

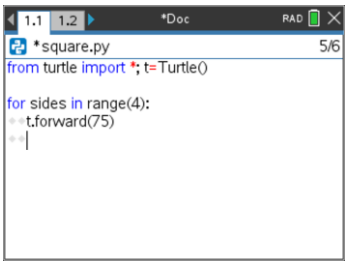
   Note: The distance unit is measured in pixels. By default, the grid scale is set to 25 pixels wide per grid square.

7) Jump to the end of the line by pressing **[tab]**, then press **[enter]** for a new line.

   Alternately, press [ctrl]+[enter] for a new line







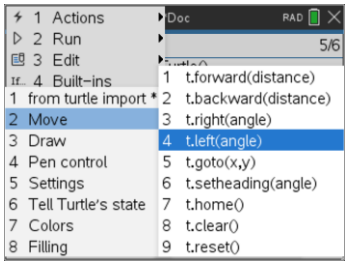
8) Press [menu] and select **More Modules**, **Turtle Graphics**, **Move**, then the **t.left(angle)** selection

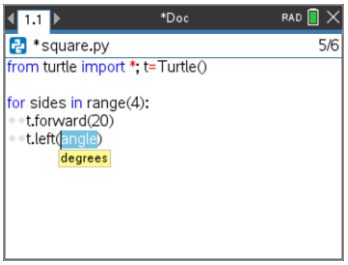
   Notice the “tool tip” which offers the hint “degrees”

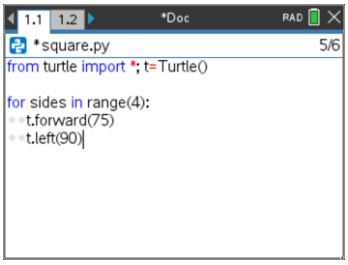
9) Enter the angle in degrees you want the turtle to turn left. For a square, we will enter 90 degrees

   Jump to the end of the line by pressing [tab].

   You are ready to run your first Turtle program!







10) To run the program press **[menu]** then select **Run**, **Run (Ctrl+R)**

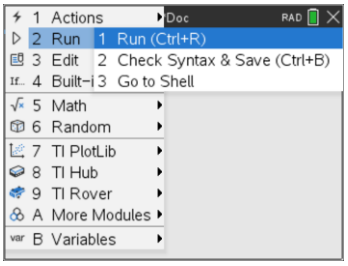
Notice the shortcut hint Ctrl+R in the menu selection. In the future you can simply use the shortcut Ctrl+R to run your program.

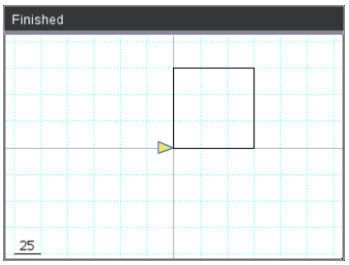
By default, the turtle is visible and you see it draw. When you finish admiring your work, press **[esc]** to clear the screen. You will be on page 1.2, a Python Shell. Press [ctrl]+[◀] to return to the Editor page 1.1. From there, modify your program and run it again.

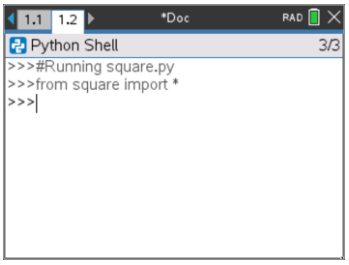
Notice the scale in the bottom-left corner

Challenges:

- Change the pen thinckness
- Change the pen color
- Hide the turtle icon
- Change the speed at which the turtle moves
- Hide the grid and scale indicator
- Fill the square





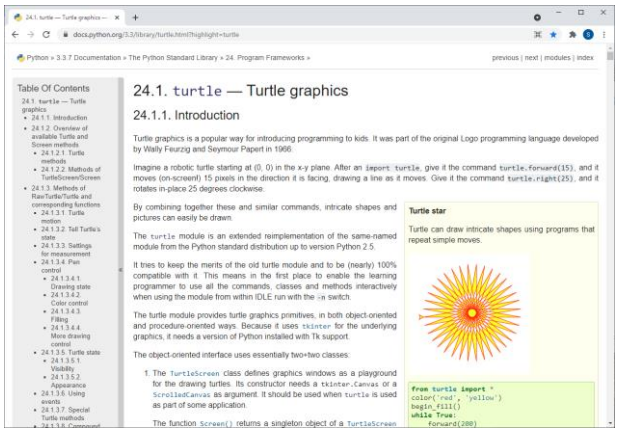


Turtle Module Methods

Every effort has been made to align with syntax and associated behaviors of the Python API (Application Programming Interface) for Turtle Graphics found on the Python documentation website. While there may be slight behaviorial and syntax differences in implementation, please visit this site to familiarize with syntax definitions and turtle behaviors.

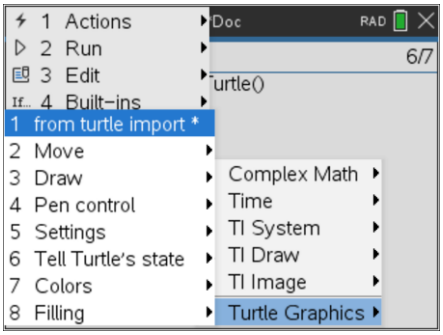
https://docs.python.org/3.3/library/turtle.html?highlight=turtle

Please see the **Default Settings and Known Exceptions** section for an overview of notes and known exceptions.



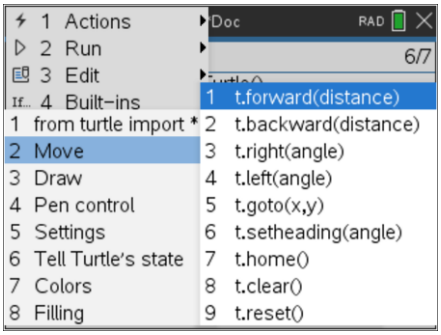
Module menu selections

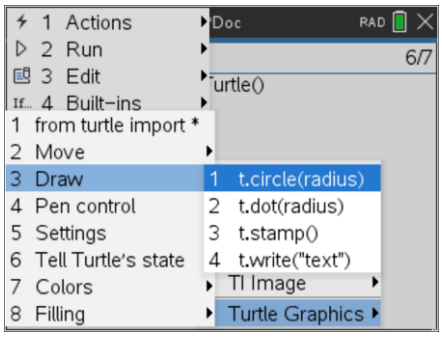
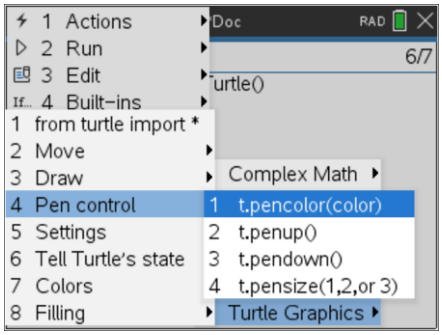
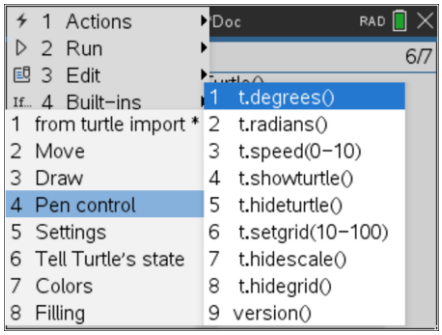
- from turtle import *
- Move
- Draw
- Pen Control
- Settings
- Tell Turtle's state
- Colors
- Filling



Move

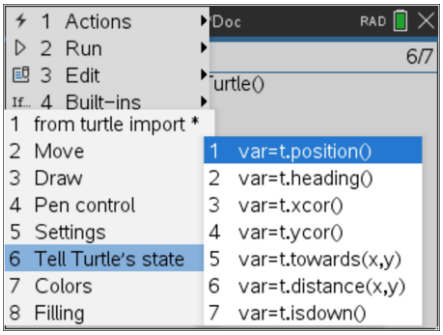
- t.forward(distance)
- t.backward(distance)
- t.right(angle)
- t.left(angle)
- t.goto(x,y)
- t.setheading(angle)
- t.home()
- t.clear()
- t.reset()



Draw • t.circle(radius) • t.dot(radius) • t.stamp() • t.write("text")	
Pen control • t.pencolor(color) • t.penup() • t.pendown() • t.pensize(1,2,or3)	
Settings • t.degrees() • t.radians() • t.speed(0-10) • t.showturtle() • t.hideturtle() • t.setgrid(10-100) • t.hidescale() • t.hidegrid() • version()	

Tell Turtle's state

- var=t.position()
- var=t.heading()
- var=t.xcor()
- var=t.ycor()
- var=t.towards(x,y)
- var=t.distance(x,y)
- var=t.isdown()

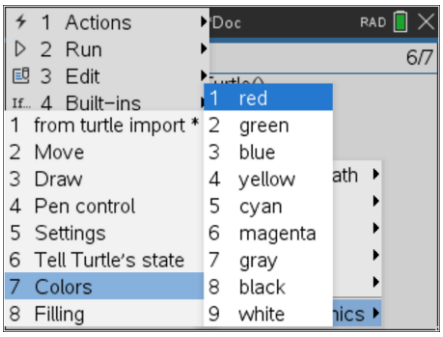


Colors

- red
- green
- blue
- yellow
- cyan
- magenta
- gray
- black
- white

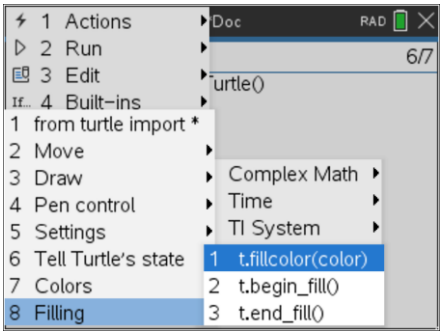
Examples:

```
t.pencolor("red")
t.fillcolor("green")
```



Filling

- t.fillcolor(color)
- t.begin_fill()
- t.end_fill()



Default Settings and Exceptions	
When these methods are not specified programatically, the following defaults are applied	
Turtle	**Default on**. Use t.hideturtle() to hide the turtle icon. When the turtle icon is visible • Pen size is fixed at 1 and cannot be changed even if a different pensize() is specified in the program • Speed can be adjusted only with arguments 1 through 10 • Use of speed(0) will automatically hide the turtle icon • The turtle icon may increasingly slow down when executing long or complex programs. In these instances it is best to apply t.hideturtle().
Grid	**Default on**. Use t.hidegrid() to hide the grid.
Scale	**Default 25 pixels**. Use t.setgrid() to change the scale of the grid. Valid arguments are 10 to 100. Use t.hidescale() to hide the scale indiicator.
Speed	**Defaults setting is t.speed(5)** Use t.speed() to change the speed. Valid arguments are 0 to 10. While values 1 through 10 range from slow to fast, speed(0) is the fastest (per the Turtle Graphics API).
Pen	**Default pen size is 1**. To adjust the pen size, use t.pensize() with valid arguments 1, 2, and 3. Note: Pen size is always 1 when turtle is visible. **Default pen color is black**. Use t.pencolor(color) to change pen color. RGB (Red, Green, Blue) values are valid arguments for pen color with values in the range of 0 through 255. Alternately, populate the t.pencolor(color) argument with selections from the Color menu. **Default pen down.** To move the turtle to a location without drawing a line, use the penup() method. The pendow() method will subsequently need to be applied to continue drawing.
Fill	**Default color is black.** Use t.fillcolor(color) to specify other colors. RGB (Red, Green, Blue) values are valid arguments for fill color with values in the range of 0 through 255. Alternately, populate the t.fillcolor(color) argument with selections from the Color menu.

Example Programs:

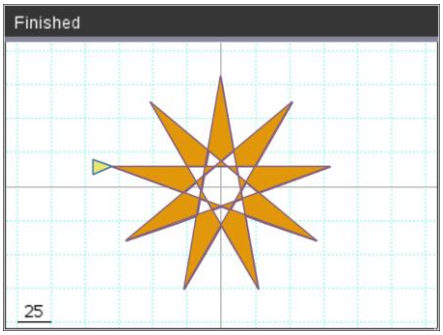
Example 1: My First Star

Draw a single star. The star is drawn with a random line color and filled with a random color.

Notice that t.pencolor() can be defined with R,G,B values (Red, Green, Blue), with values in the range of 0 to 255.

```
from turtle import *; t=Turtle()
from random import randint
t.speed(10)
t.penup()
t.goto(-80,14)
t.pendown()
t.pencolor(randint(0,255),randint(0,255),randint(0,255))
t.fillcolor(randint(0,255),randint(0,255),randint(0,255))
t.begin_fill()
while True:
    t.forward(160)
    t.right(160)
    if t.heading() < 1:
        break
t.end_fill()
```

Note: The “break” function is currently not in a menu selection and needs to be hand-typed. “True” is found under [menu], Built-ins, Ops.



Challenges:

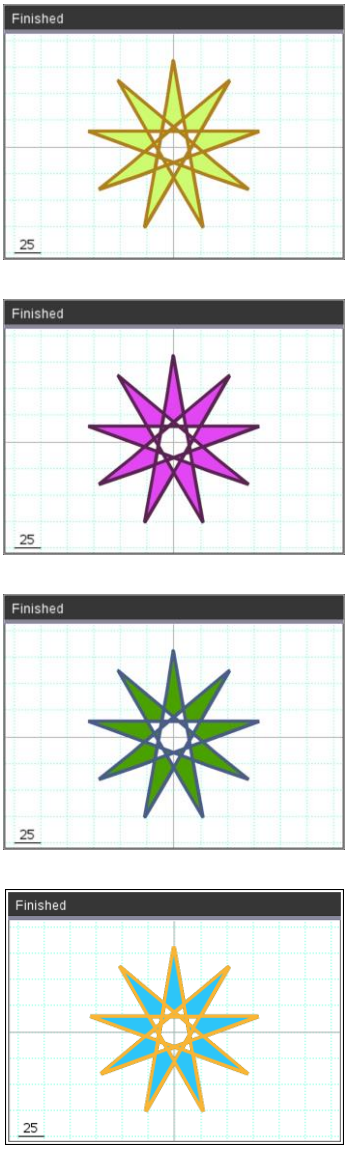
- Change the t.right() angle to create stars of different shapes. What happns when the angle is small. Large.
- Change the t.forward() distance for smaller and larger stars.

Example file name: **random_stars.tns**
Program name: **MyFirstStar.py**

Example 2: PhiX 177 Super Nova

Build on the prior program by automatically re-generating a new star every second until you press [esc]

```
from turtle import *; t=Turtle()
from time import sleep
from random import randint
t.speed(10)
t.hideturtle()
t.pensize(2)
while get_key() != "esc":
  t.penup()
  t.goto(-80,14)
  t.pendown()
  t.pencolor(randint(0,255),randint(0,255),randint(0,255))
  t.fillcolor(randint(0,255),randint(0,255),randint(0,255))
  t.begin_fill()
  while True:
    t.forward(160)
    t.right(160)
    if t.heading() < 1:
      break
  t.end_fill()
  sleep(1)
```



Example file name: **random_stars.tns**
Program name: **PhiX177SuperNova.py**

Example 3: Zinnias in the Summer

Build on the prior programs and randomize the position of stars on the screen.

```
from turtle import *; t=Turtle()
from time import *
from random import randint, uniform
t.speed(0)
t.hideturtle()
t.pensize(1)
while get_key() != "esc":
    t.penup()
    t.goto(randint(-200,120), randint(-100,100))
    t.pendown()
    t.pencolor(randint(0,255),randint(0,255),randint(0,255))
    t.fillcolor(randint(0,255),randint(0,255),randint(0,255))
    t.begin_fill()
    while True:
        t.forward(80)
        t.left(162)
        if t.heading() < 1:
            break
    t.end_fill()
    sleep(uniform(.1,.4))
```

Challenges:

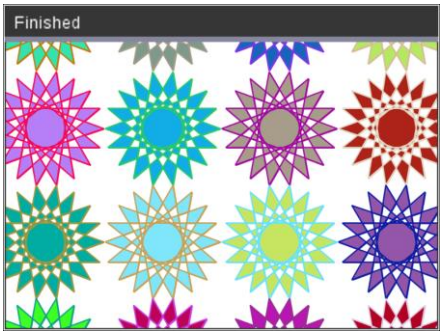
- Instead of stars, draw circles, squares or rectangles
    - Randomize the size of the circles
    - Randomize the size of the squares
    - Randomize the height and width of the rectangles
- Adjust the pen and fill colors so they are shade variations of the same color

Example file name: **random_stars.tns**
Program name: **ZinniasInSummer.py**

Example 4: Grandma's Quilt

Builds on the prior programs but lays the stars out in an orderly "quilt" pattern.

```
from turtle import *; t=Turtle()
from random import randint
t.speed(10)
t.hidegrid()
t.hideturtle()
t.pensize(1)
for j in range(112,-150,-84):
  for i in range(-169,170,86):
    t.penup()
    t.goto(i, j)
    t.pendown()
    t.pencolor(randint(0,255),randint(0,255),randint(0,255))
    t.fillcolor(randint(0,255),randint(0,255),randint(0,255))
    t.begin_fill()
    while True:
        t.forward(80)
        t.left(140)
        if t.heading() < 1:
            break
    t.end_fill()
```



Challenges:

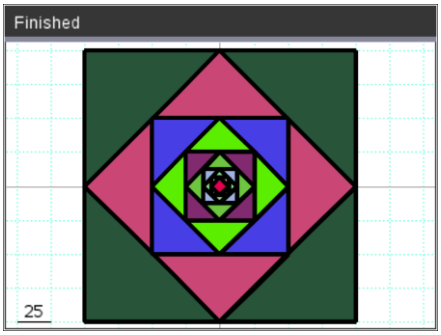
- Adjust the quilt pattern so that stars overlap vertically and horizontally.
- Change the number of points on the star.
- Draw circles or squares instead of stars.

Example file name: **random_stars.tns**
Program name: **GrandmasQuilt.py**

Example 5: Stained Glass

Build a series of squares inside of squares.

```
from turtle import *; t=Turtle()
from random import randint
from math import sqrt
from ti_system import *
from time import sleep
t.pensize(2)
t.hideturtle()
t.speed(10)
while get_key() != "esc":
  n=200
  t.penup()
  t.goto(-n/2,-n/2)
  t.pendown()
  t.setheading(0)
  for i in range(10):
    t.fillcolor(randint(0,255),randint(0,255),randint(0,255))
    t.begin_fill()
    for j in range(4):
      t.forward(n)
      t.left(90)
    t.end_fill()
    t.forward(n/2)
    t.left(45)
    n=sqrt((n**2)+(n**2))/2
  sleep(.5)
```



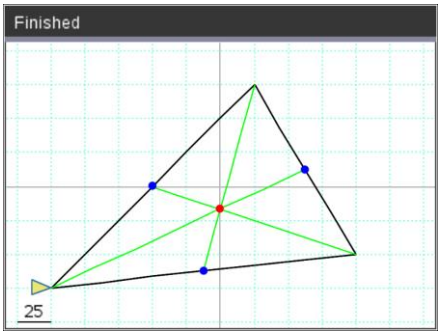
Challenges:

- Decrease or increase the number of squares drawn.
- Change the orientation angle of squares that are draw.
- Create a “tile floor” with smaller tiles that look like the original.

Example file name: **squareloop.tns**
Program name: **StainedGlass.py**

Example 6: Centroid
Build a triangle, midpoints, median segments and centroid.

```
from turtle import *; t=Turtle()

# calculate the midpoint of line segment
def midpoint(pt1,pt2):
  return ((pt1[0] + pt2[0])/2, (pt1[1] + pt2[1])/2)

# plot point
def plot_point(pt):
  t.penup()
  t.goto(pt)
  t.pendown()
  t.dot(3)

# the triangle verticies
v1 = (25,75)
v2 = (-125,-75)
v3 = (100,-50)

# calculate the centroid centroid =((v1[0]+v2[0]+v3[0])/3,(v1[1]+v2[1]+v3[1])/3)

# calculate the midpoints
mid_1_2 = midpoint(v1,v2)
mid_2_3 = midpoint(v2,v3)
mid_1_3 = midpoint(v1,v3)

# draw the triangle in black
t.penup()
t.goto(v1)
t.pendown()
t.goto(v2)
t.goto(v3)
t.goto(v1)

# draw the three median segments in green
t.pencolor("green")

t.penup()
t.goto(mid_1_2)
t.pendown()
t.goto(v3)

t.penup()
t.goto(mid_2_3)
t.pendown()
t.goto(v1)

t.penup()
t.goto(mid_1_3)
t.pendown()
t.goto(v2)

# draw the centroid in red
t.pencolor("red")
plot_point(centroid)

# draw the midpoints in blue
t.pencolor("blue")
plot_point(mid_1_2)
plot_point(mid_2_3)
plot_point(mid_1_3)
```



Challenges:

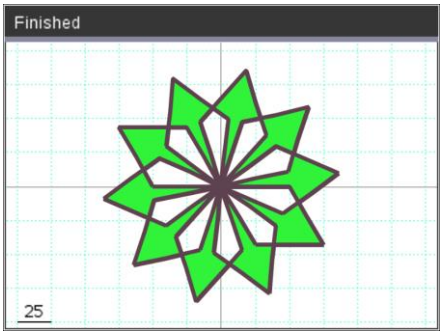
- Change the coordinates of one or more vertex and observe the result.

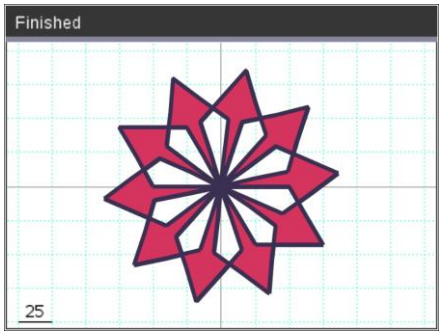
Example file name: **Centroid.tns**
Program name: **tri_centroid.py**

Example 7: RhombuStar1

This deploys a familiar loop strtategy but draws a rhombus that is rotated around the origin. It continues to regenerate every second until you press [esc].

```
From turtle import *; t=Turtle()
from time import sleep
from random import randint
t.speed(10)
t.hideturtle()
t.pensize(2)
while get_key() != “esc”:
  t.pencolor(randint(0,255),randint(0,255),randint(0,255))
  t.fillcolor(randint(0,255),randint(0,255),randint(0,255))
  t.begin_fill()
  for I in range(10):
    for j in range (2):
      t.forward(50)
      t.right(60)
      t.forward(50)
      t.right(120)
    t.right(36)
  t.end_fill()
  sleep(1)
```





Challenges:

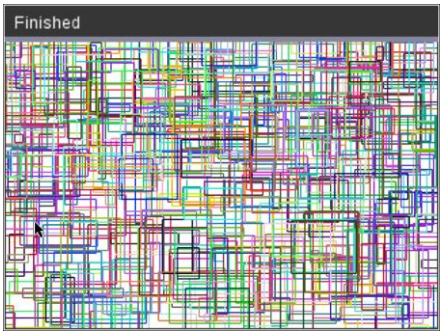
- Rotate a square instead of a rhombus.
- Rotate a circle instead of a rhombus.
- Change the number of rhombus, squares or circles that are rotated around the origin.

Example file name: **geo_stars.tns**
Program name: **RhombuStar1.py**

Example 8: Threadbare

Deploy a familiar loop strtategy and draw a random array of rectangles. It continues until you press [esc].

```
from turtle import *; t=Turtle()
from ti_system import *
from random import randint
t.hidegrid()
t.speed(10)
t.hideturtle()
t.pensize(1)
while get_key() != "esc":
  t.penup()
  t.goto(randint(-200,150), randint(-120,100))
  t.pendown()
  t.pencolor(randint(0,255), randint(0,255), randint(0,255))
  b=randint(5,60)
  h=randint(5,60)
  for i in range(2):
    t.forward(b)
    t.left(90)
    t.forward(h)
    t.left(90)
```



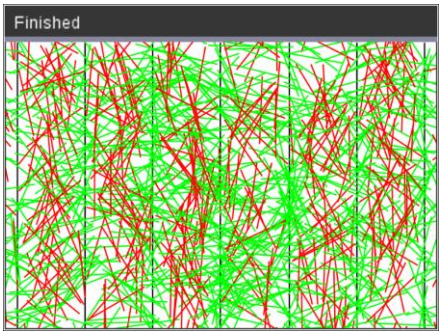
Challenges:

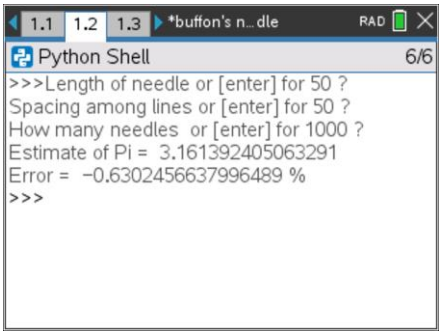
- Fill the rectangles.
- Make squares instead of rectangles.

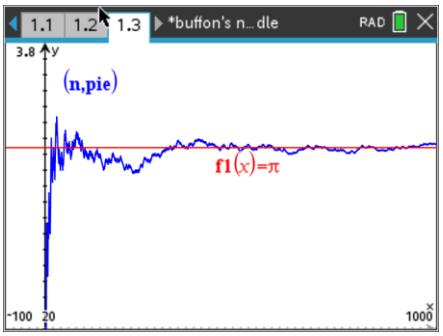
Example file name: **random_rectangles.tns**
Program name: **Threadbare1.py**
Program name: **Threadbare3.py**

Example 9: Buffon's Needle
Simulate Buffon's needle experiment.

```
from ti_system import *
from random import *

# use shell before entering turtle environment
clear_history()
length=int(input("Length of needle or [enter] for 50 ?") or '50')
spacing=int(input("Spacing among lines or [enter] for 50 ?") or '50')
needles=int(input("How many needles or [enter] for 1000 ?") or '1000')
the_pies=[]
the_count=[]
# set up turtle environment
from turtle import *; t=Turtle()
t.hidegrid()
t.speed(0)
w,h=get_screen_dim()

the_lines=[] # list of all x-coordinate of all lines
n_lines=int((w/spacing)/2)+2 # number of lines
crossing = 0 # needles crossing a line
estimate = 0 # calculated estimate of pi
# draw all of the lines and append the_lines
for x in range(-n_lines*spacing,(n_lines+1)*spacing,spacing):
  the_lines.append(x)
  t.penup()
  t.goto(x,-h/2)
  t.pendown()
  t.goto(x,h/2)

# draw the needles
for i in range(needles):
  x1=randint(0,w)-w//2
  y1=randint(0,h)-h//2
  angle=random()*360
  t.penup()
  t.goto(x1,y1)
  t.setheading(angle)
  t.forward(length)
  x2= t.xcor()
  y2= t.ycor()
  t.pencolor("red")
# check if needle touches or crosses a line
  for n in range(len(the_lines)):
    if((x1<=the_lines[n] and x2>=the_lines[n]) or (x2<=the_lines[n] and x1>=the_lines[n])):
      t.pencolor("green")
      crossing+=1
  t.pendown()
  t.goto(x1,y1)
# Buffon's formula
  try:
    estimate =( 2*length*i)/(crossing*spacing)
  except:
# all fun and games until somone divides by zero
    pass
  the_pies.append(estimate)
  the_count.append(i)
store_list("n",the_count)
store_list("pie",the_pies)
error=(pi-estimate)*100/pi
print("Estimate of Pi = ", estimate,"\nError = ",error,"%")
```







Challenges:

- Run the simulation specifying different needle lengths and spacing between lines.

Example file: **buffon's needle.tns**
Program name: **buffon.py**