Getting Started Guide

Note: the following sections assume that you have updated your **TI-84 Plus CE Python** to **version 5.7** OS and Apps or later. If you have not updated to version 5.7 or later, please visit https://education.ti.com/en/product-resources/whats-new-84-ce. Be sure to back up or archive files and data on your TI-84 Plus CE Python prior to updating to version 5.7 or later.

Transfer the Turtle Module to Your Calculator and (if used) TI-SmartView Software

1) Transfer the Turtle module and Grid files to your TI-84 Plus CE Python calculator(s)

After downloading the Zip and extracting files:

- Open your TI Connect CE desktop software
- Connect your TI-84 Plus CE Python 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 Calculators panel in the Calculator Explorer workspace.
- Transfer the **TURTLE.8xv** and **GRID.8xv** files to the connected calculator(s) by dragging the files into the Connected Calculators window.

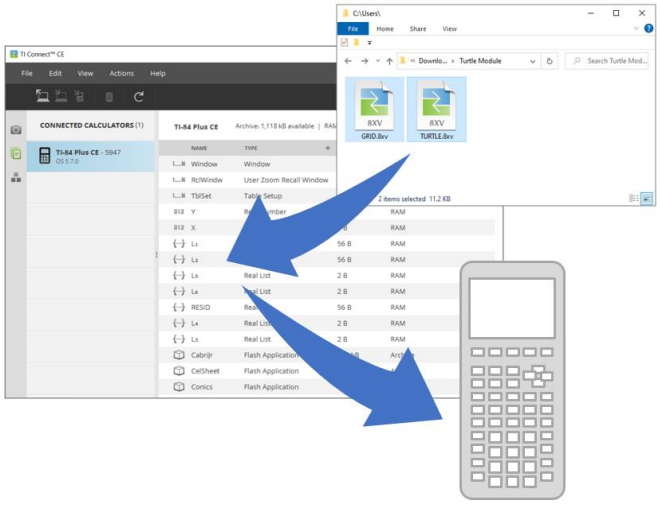
Note: the GRID.8xv file is an image used as a grid background in Turtle programs.

Note: The Turtle and Grid files may alternately be transferred from calculator to calculator through a unit-to-unit USB cable linking process. See your calculator guidebook for the calculator-to-calculator file transfer process.

- Next, go to step 3) **Import the Turtle module**

This step requires use of (free) **TI Connect CE** desktop software for PC or Mac

https://education.ti.com/en/products/computer-software/ti-connect-ce-sw



2) Transfer the Turtle module and Grid files to your TI-SmartView desktop software (PC or Mac)

This step is for individuals who use TI-SmartView CE desktop software for TI-84 Plus family products.

After downloading the Zip and extracting files:

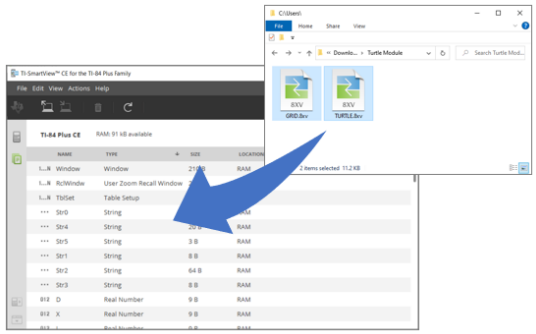
- Drag and drop the **TURTLE.8xv** and **GRID.8xv** files into the **Emulator Explorer** workspace of your TI-SmartView CE desktop software.

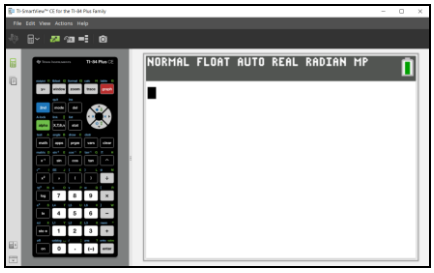
- Return to the Emulator workspace

- Next, go to step 3) **Import the Turtle module**

TI-SmartView CE desktop software is a desktop software application for PC and Mac that emulates TI-84 Plus family products.

https://education.ti.com/en/products/computer-software/ti-smartview-ce-for-84





Note: at the time of Turtle module release, **OS/App version 5.7** was not available for implementation on **TI-SmartView CE for the TI-84 Plus Family**. See note in section 3 regarding visibility of the [Add-on] tab and an alternative method to import the Turtle module by hand-typing the import statement.

3) Import the Turtle module

Once the Turtle module and Grid files are transferred, create a Python program and import the Turtle module in order to access the Turtle module menu selections.

- On your TI-84 Plus CE Python calculator (or emulator), open the Python App. It can be found under the [**prgm**] key or the [**apps**] key. The Python App will open to a File Manager screen.
- Create a new Python program by selecting the tab labeled **New** (press the [**zoom**] key)
- Give the Python program a name (example: **SQUARE**), then select **Ok**

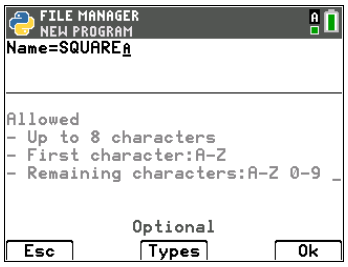
  You are now in a Python program Editor screen. Next, enter a Turtle import statement and Turtle object.

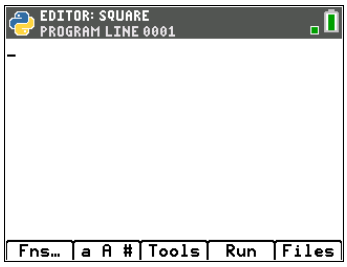
- Select the [**Fns…**] tab (at the bottom of the screen) and arrow (left or right) to the Modul tab (at the top of the screen).
- At the bottom of the screen select the [**Add-on**] tab.
- Select option **4:from turtle import *** to paste the Turtle module import statement. Notice that it also pastes a constructor **t=Turtle()** to assign **t** as the turtle object.
- Return to the Modul tab and notice the appearance of the **8:turtle…** menu selection.

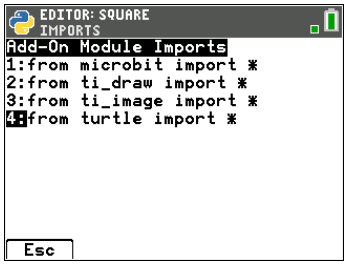
Note: the [**Add-On**] tab is visible only with OS version 5.7 or later. If you are using a prior OS version, update to version 5.7 or later; otherwise you may hand-type the import statement and t=Turtle() constructor.

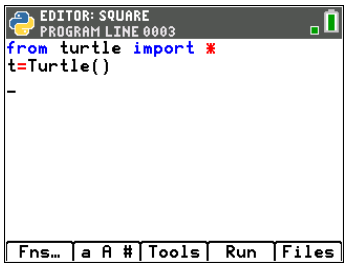
At this point you are ready to access the Turtle module menu selections and write a Turtle program!

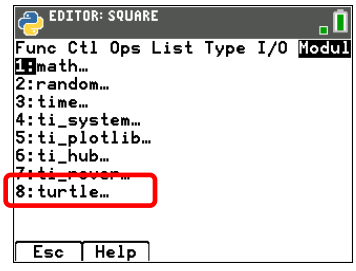
- Proceed to the next section: **Creating your First Turtle Program: Let's Draw a Square**











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

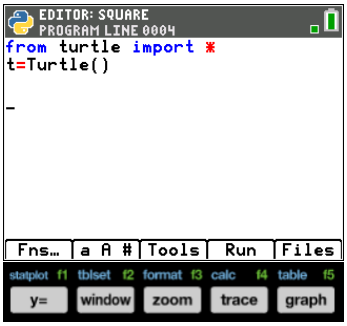
1) Enter a loop statement

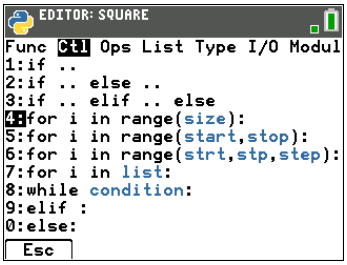
Continuing from the prior section (in the Editor screen)

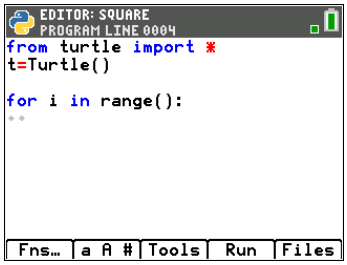
- Press [**enter**] to go to the next line.
- Select the [**Fns…**] tab (press the [**y=**] key) then right arrow to highlight the Ctl tab.
- Arrow down to selection 4:**for i in range(size):**
- Press [enter] to paste it into the Editor

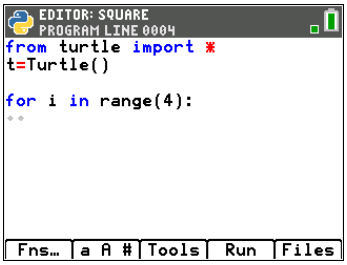
Note that the cursor is blinking inside the parentheses and that an automatic indent is inserted under the "for" loop statement.

- Enter a value of **4** in the parenthesis. This determines the number of cycles in the "for" loop that is defined with index i.







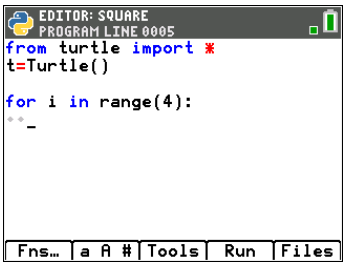


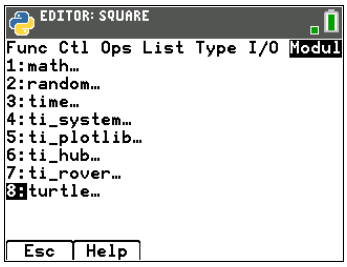
2) Enter a statement to move the turtle forward

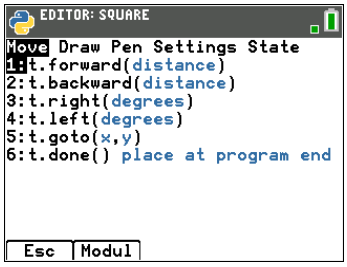
- Arrow down to the indented line.
- Select the [**Fns…**] tab (at the bottom of the screen) and arrow (left or right) to the Modul tab (at the top of the screen).
- Arrow up (or down) to selection 8:**turtle…** and press [**enter**] to see the Turtle module menu selections.

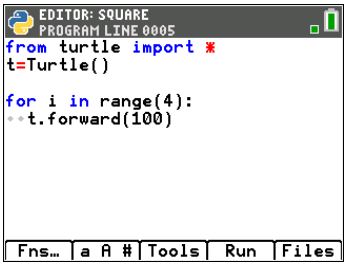
Notice the menu tabs and menu selections in the Turtle module. We will start by selecting a method to move the turtle forward.

- Under the Move tab with the cursor on selection 1:**t.forward(distance)**, press [enter].
- Enter a value of **100** in the parentheses. This is the number of pixels to move forward.



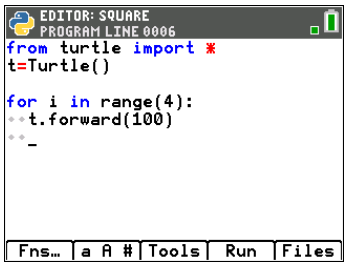


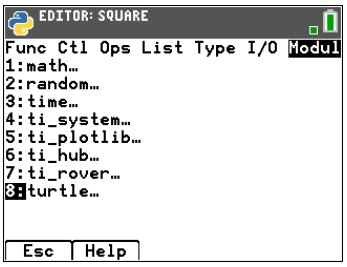


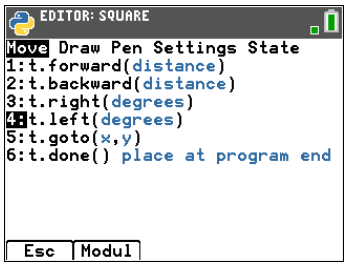


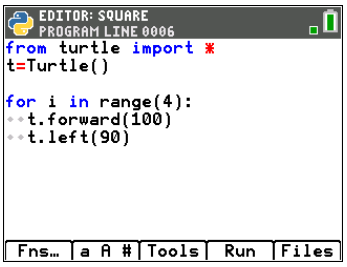
3) Enter the direction and angle to turn

- At the end of the **t.forward(100)** line, press [enter] to create a new indent line.
  (Alternately press [2nd] [enter] to jump to a new line)
- Select the [**Fns…**] tab (at the bottom of the screen) and then arrow (left or right) to the Modul tab (at the top of the screen).
- Arrow up (or down) to selection **8:turtle…** and press [enter] to see the Turtle module menu selections.
- Under the Move tab with the cursor on selection **4:t.left(degrees)**, press [enter].
- Enter the angle in degrees you want the turtle to turn left. For a square, we will enter **90** degrees.





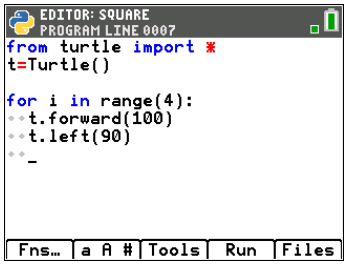


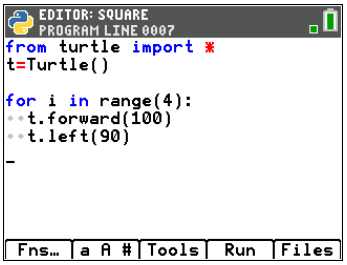


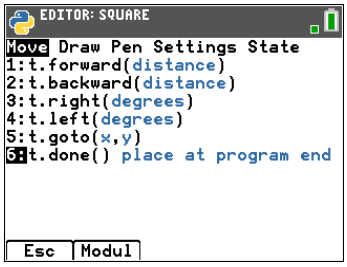
4) Enter a "done" statement at the end of the program
This makes the Turtle drawing stay on your screen when it is complete.

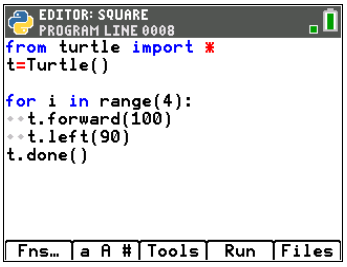
- At the end of the **t.left((90)** line, press [enter] to create a new line.
- Remove the indent by pressing the [del] key twice. (Alternately, select the [**Tools**] menu, then selection **2:Indent ◀**.)
- Return to the Turtle module menu as before.
- Under the Move tab with the cursor on selection **6:t.done()**, press [enter].

You are ready to run your first Turtle program!









5) Run your Turtle program

Simply press the [**Run**] selection at the bottom of the screen (by pressing the [**trace**] key).

By default, the turtle is visible and you see it draw. Also notice that the Grid is visible by default. (The Grid scale is 25 pixels per Grid square).

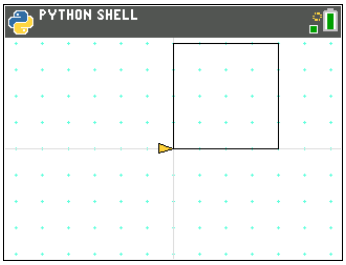
When you finish admiring your work, press the **[clear]** key to clear the screen. Notice that you will be on a Python Shell screen. This is where your python program was processed.

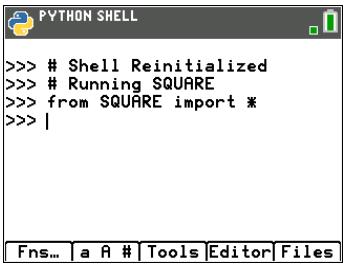
Return to the Editor by pressing the [**Editor**] selection at the bottom of the screen.

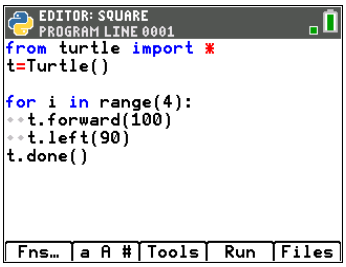
From here, modify your program and run it again!

Challenges:

- Change the pen thickness
- 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





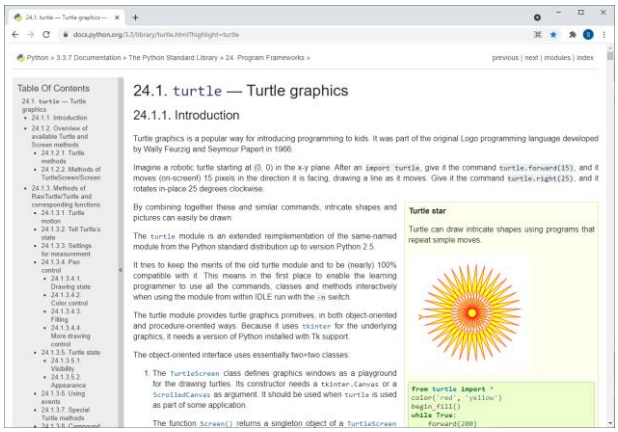


Turtle Module Methods

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

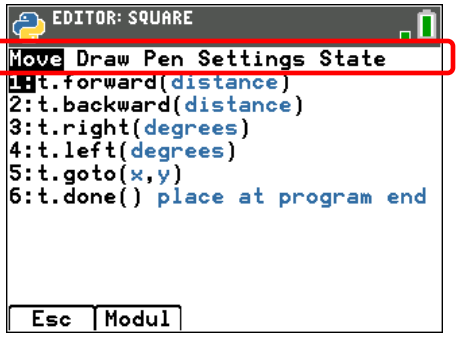
Please see the **Default Settings and Known Exceptions** section for an overview of notes and known exceptions.



Module menu selections

These are the menu tabs at the top of the Turtle module screen

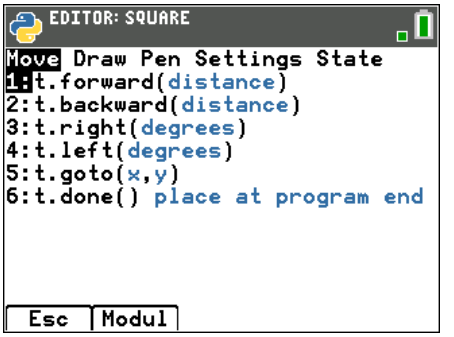
- **Move**
- **Draw**
- **Pen**
- **Settings**
- **State**



Move

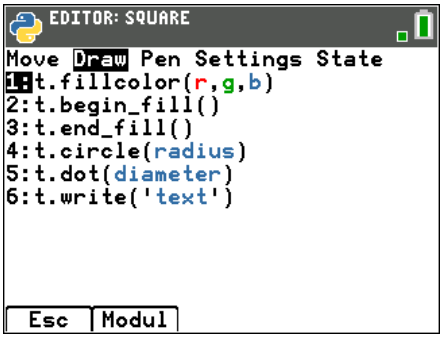
- **t.forward(distance)** – specifies the forward direction and distance in pixels
- **t.backward(distance)** - specifies the backward direction and distance in pixels
- **t.right(angle)** – specifies turning right and the angle in degrees.
- **t.left(angle)** – specifies turning left and the angle in degrees.
- **t.goto(x,y)** – specifies (x,y) coordinates for turtle to move to.
- **t.done()** – used at the end of the program to display the resulting turtle drawing.

The Turtle coordinate system is oriented with (0,0) at the center of the screen.



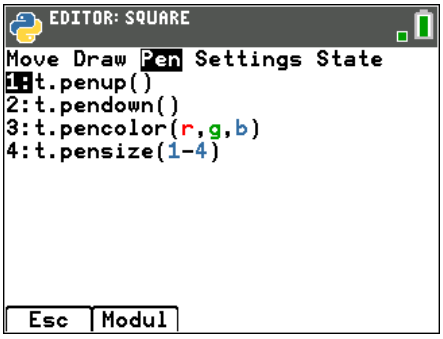
Draw

- **t.fillcolor(r,g,b)** – specifies a fill color. RGB: **r,g,b** are the red, green, and blue arguments for fill color, each with a value in the range of 0 through 255.
- **t.begin_fill()** – establishes when the fill method begins.
- **t.end_fill()** – establishes when the fill method ends.
- **t.circle(radius)** – draws a circle with specified radius. Circle center is one radius to the left of the turtle.
- **t.dot(diameter)** – draws a dot with specified diameter. Applies the color specified with t.pencolor(). Default black.
- **t.write('text')** – writes text specified in the 'text' string.



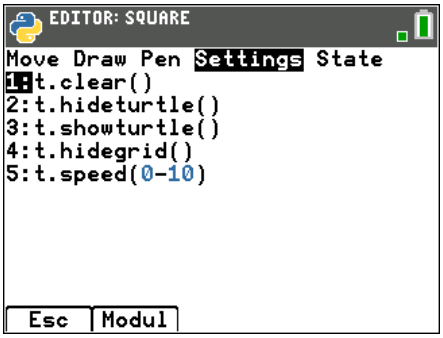
Pen

- **t.penup()** – enables turtle to move without drawing a line
- **t.pendown()** – used after t.pendow(), enabling turtle to draw a line
- **t.pencolor(r,g,b)** – specifies pen color. RGB: **r,g,b** are the red, green, and blue arguments for fill color, each with a value in the range of 0 through 255.
- **t.pensize(1-4)** – changes the pen size. Four pen sizes; 1 through 4. Default value of 1 when t.pensize() is not specified.



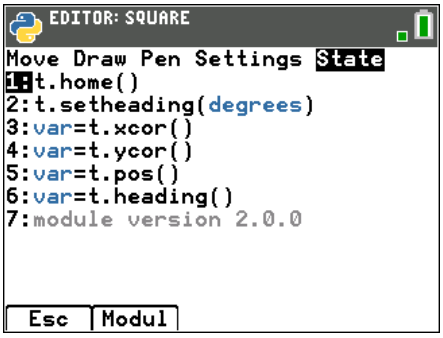
Settings

- **t.clear()** – clears lines that have been drawn.
- **t.hideturtle()** – makes the turtle invisible.
- **t.showturtle()** – makes the turtle visible.
- **t.hidegrid()** – hides the grid.
- **t.speed(0-10)** – specifies eleven turtle speeds ranging from 1 (slow) to 10 (fast). Speed zero (0) is the fastest setting. Default value of 5 when t.speed() is not spedified.



State

- **t.home()** – move turtle to the origin (0,0) and sets its heading to its start orientation of zero (pointing to the right).
- **t.setheading(degrees)** – sets turtle heading in degrees. Positive values are counter-clockwise starting from zero heading. Negative values are clockwise.
- **var=t.xcor()** – Return the turtle's x coordinate.
- **var=t.ycor()** – Returns the turtle's y coordinate.
- **var=t.pos()** – Return the turtle's current location (x,y).
- **var=t.heading()** – Return the turtle's current heading.
- **module version 2.0.0** – the Turtle module version number.



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.
Grid	**Default on**. Use t.hidegrid() to hide the grid. Grid scale is 25 pixels per grid square
Speed	**Defaults setting is 5 when no speed is specified.** Use t.speed() to change the speed. Valid arguments are 0 to 10. While values 1 through 10 range from slow to fast, t.speed(0) is the fastest (in accordance with the Turtle Graphics API).
Pen	**Default pen size 1**. To adjust the pen size, use t.pensize() with valid arguments 1, 2, 3, and 4. **Default pen color is black**. Use t.pencolor(r,g,b) to change pen color. RGB (Red, Green, Blue) values are valid arguments for pen color with values in the range of 0 through 255 for each color variable. **Default pen down.** To move the turtle to a location without drawing a line, use the t.penup() method. The t.pendow() method will subsequently need to be applied to continue drawing.
Fill	**Default color is black.** Use t.fillcolor(r,g,b) 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.

Exceptions

Due to Python processor memory size limitations you may experience memory errors with large programs or programs that are processor intensive. If this occurs, try the following

a) import only the needed functions from a module; for example, **from random import randint** (which imports only the **randint** function) instead of **from random import *** (which imports all functions in the module).
b) Adjust the range of values used in an argument; for example, **t.fillcolor(randint(150,255),0,0)** instead of **t.fillcolor(randint(0,255), randint(0,255), randint(0,255))**
c) Eliminate portions of the program that may consume excessive memory. In Example 4 (below) we remove Fill methods from the program.

Example Programs:

Example 1: My First Star

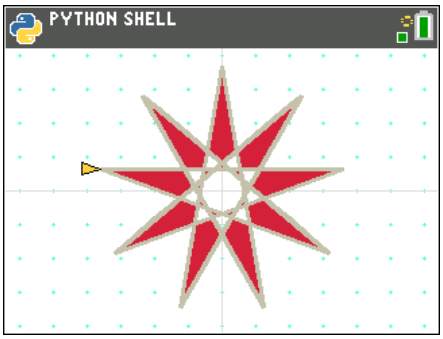
Draw a single star. The star is drawn with a random line color and filled with a random color.

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

```
from random import *
from turtle import *
t=Turtle()
t.pensize(2)
t.penup()
t.goto(-90,16)
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(180)
    t.right(160)
    if t.heading() < 1:
        break
t.end_fill()
t.done()
```

Note: The **break** function is found in the [**catalog**] menu. Press [**2nd**] [**catalog**] and scroll to the selection for **break**. **True** is found under the Ops menu tab.

Example file name: **STAR1**



Challenges:

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

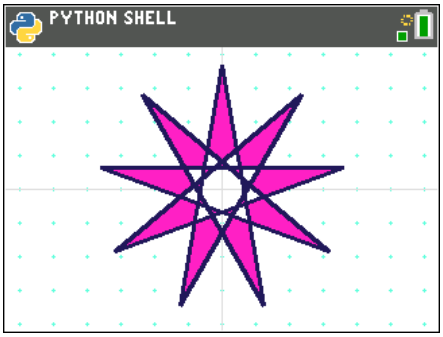
Example 2: PhiX 177 Super Nova
Build on the prior program by automatically re-generating a new star every one and one half second until you press [clear]

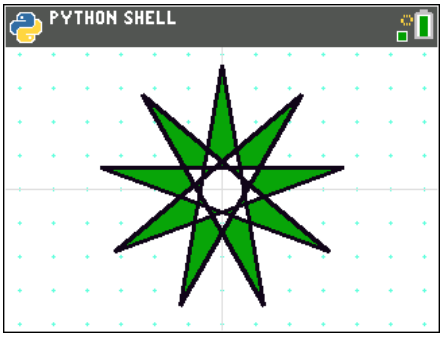
```
from time import *
from random import *
from turtle import *
t=Turtle()
t.hideturtle()
t.pensize(2)
t.speed(10)
while not escape():
  t.penup()
  t.goto(-90,16)
  t.pendown()
  t.pencolor(randint(0,255),randint(0,255),randint(0,255))
  t.fillcolor(randint(0,255),randint(0,255),randint(0,255))
  t.begin_fill()
  while True:
    t.forward(180)
    t.right(160)
    h=t.heading()
    if h<1:
      break
  t.end_fill()
  sleep(1.5)
t.done()
```

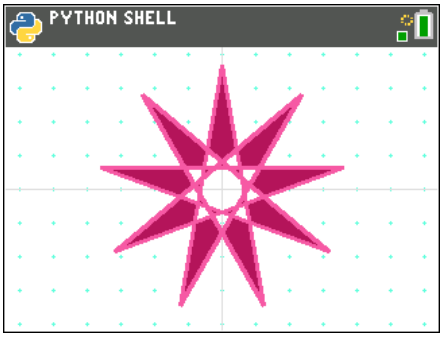
Note: The **time** module is imported so that the **sleep()** function can be used to pause each star for a moment before the next is drawn.

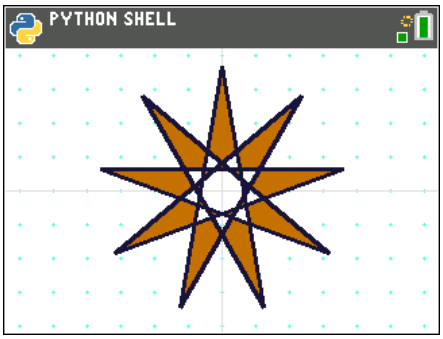
Note: The **while not escape():** function is found under the **ti_system…** module menu. When using the Turtle module, it is not necessary to import the ti_system module in order to use the **while not escape():** function. Use of other functions in the ti_system menu will require import of the ti_system module.

Example file name: **STAR2**







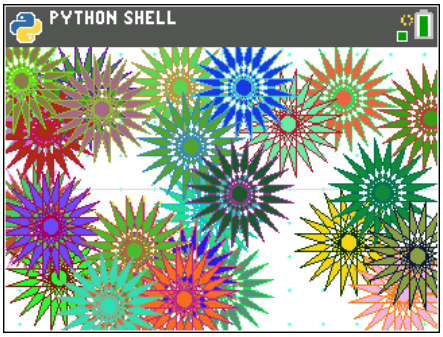


Example 3: Zinnias in the Summer

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

```
from random import randint
from turtle import *
t=Turtle()
t.pensize(1)
t.hideturtle()
t.speed(0)
while not escape():
    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)
        h=t.heading()
        if h<1:
            break
    t.end_fill()
    t.done()
```

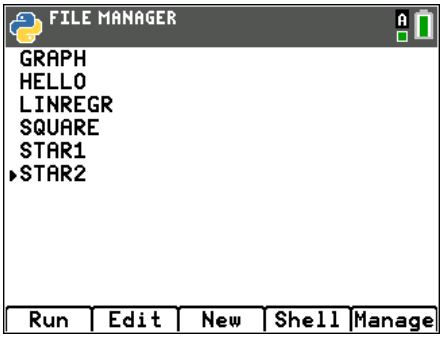
Example file name: **STAR3**

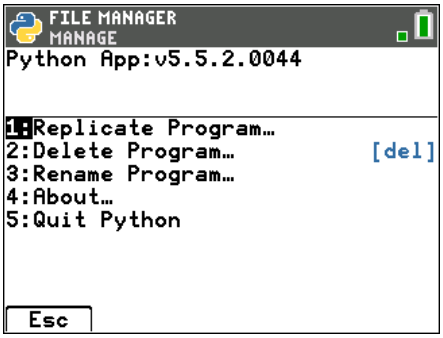


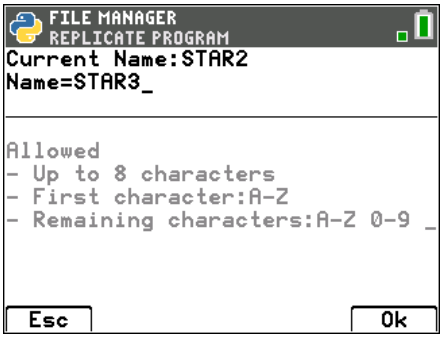
Challenges:

- Instead of stars, draw squares or rectangles
  - 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

Tip: When building on an existing program it may be convenient to make a copy of the existing program, then modify the code. This may be accomplished from the File Manger by 1) selecting the program to copy (in this example, STAR2), then 2) selecting the [**Manage**] tab, then 3) selecting option **1:Replicate Program…** and 4) entering the name for a new program (in this example, STAR3).





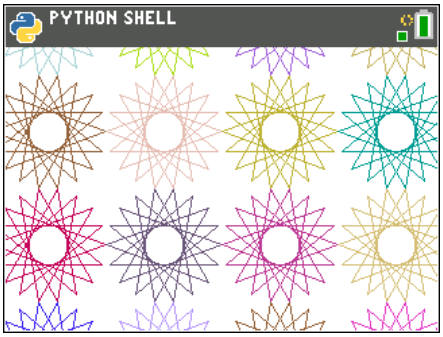


Example 4: Grandma's Quilt

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

```
from random import randint
from turtle import *
t=Turtle()
t.hideturtle()
t.hidegrid()
t.speed(0)
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))
    while True:
      t.forward(80)
      t.left(140)
      h=t.heading()
      if h<1:
        break
t.done()
```

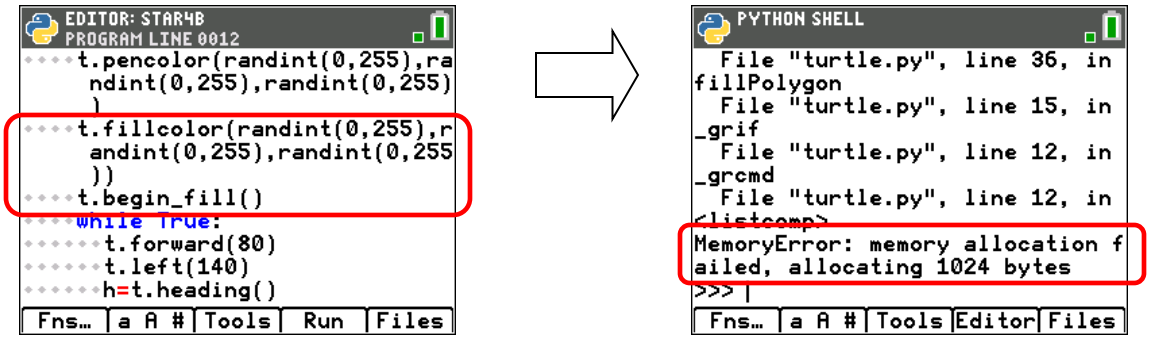
Example file name: **STAR4**



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.

Note: Notice in this example we do not use a Fill method. This is because the Python processor on TI-84 Plus CE Python has limited memory. If a program exceeds the available memory in the Python processor, a Memory Error may occur, as illustrated below when a Fill method is used in Example 4. If a Memory Error occurs, modify your program to optimize (minimize) the program size. In this instance we removed the Fill methods. In other instances, it may involve importing only the functions in a module that are needed such as **from random import randint** (which imports only the **randint** function) instead of **from random import *** (which imports all functions in the module).

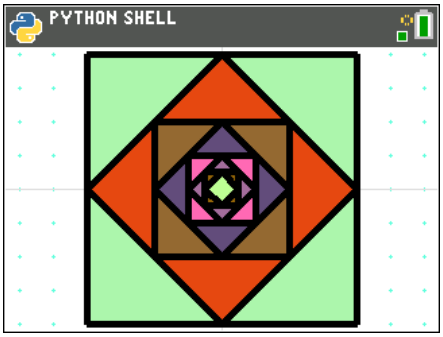


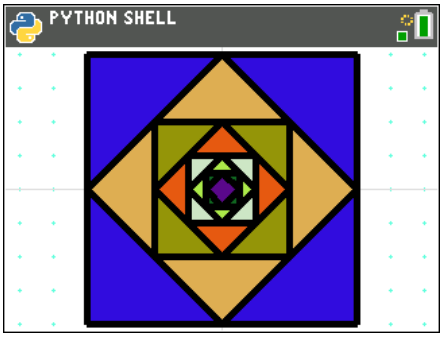
Example 5: Stained Glass

Build a series of squares inside of squares.

```
from random import randint
from ti_system import escape
from math import sqrt
from time import sleep
from turtle import *
t=Turtle()
t.speed(0)
t.pensize(3)
t.hideturtle()
while not escape():
  d=200
  t.penup()
  t.goto(-d/2,-d/2)
  t.setheading(0)
  t.pendown()
  for i in range(8):
    t.fillcolor(randint(0,255),randint(0,255),randint(0,255))
    t.begin_fill()
    for j in range(4):
      t.forward(d)
      t.left(90)
    t.end_fill()
    t.penup()
    t.forward(d/2)
    t.left(45)
    t.pendown()
    d=sqrt((d**2)+(d**2))/2
  sleep(1.5)
t.done()
```

Example file name: **SQRLOOP**





Challenges:

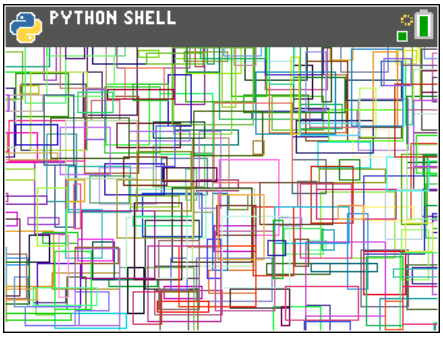
- Decrease or increase the number of squares drawn.
- Change the orientation angle of squares that are draw.
- Create a “tile floor” with smaller tiles that look like the original.

Example 6: Threadbare

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

```
from random import randint
from ti_system import escape
from turtle import *
t=Turtle()
t.hidegrid()
t.hideturtle()
t.speed(0)
while not escape():
  t.penup()
  t.goto(randint(-200,150), randint(-110,100))
  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)
t.done()
```

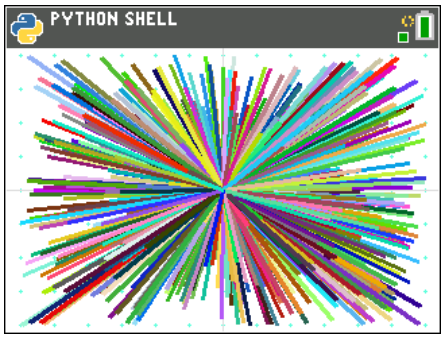
Example file name: **RANDREC**



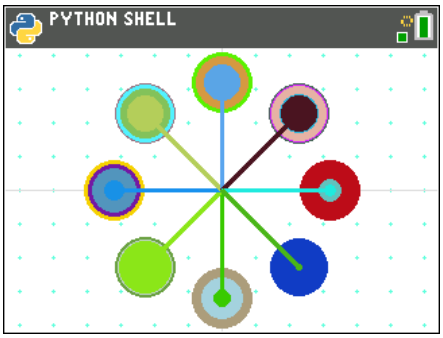
Challenges:

- Fill the rectangles.
- Make squares instead of rectangles.
- Adjust the size of the rectangles or squares.
- Adjust the color so it is more uniform; shades of the same color.

Challenge: Write your own code to create these images



Example file name: **SPLAT1**

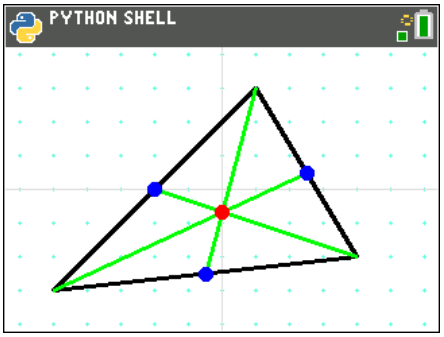


Example file name: **SPLAT2**

Example 7: Centroid
Build a triangle, midpoints, median segments and centroid.

```
from turtle import *
t=Turtle()
t.speed(1)
t.hideturtle()
t.pensize(2)

# 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(10)

# 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(0,255,0)

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(255,0,0)
plot_point(centroid)

# draw the midpoints in blue
t.pencolor(0,0,255)
plot_point(mid_1_2)
plot_point(mid_2_3)
plot_point(mid_1_3)

t.done()
```

Example file name: **CENTROID**



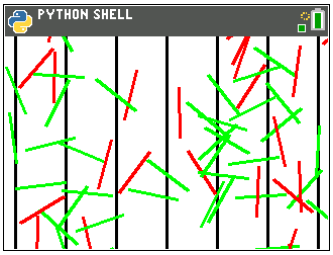
Challenges:

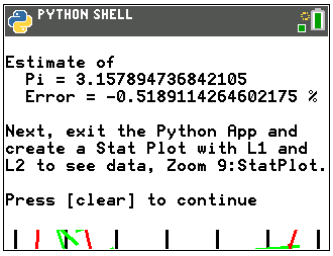
- Change the coordinates of one or more vertex and observe the result.

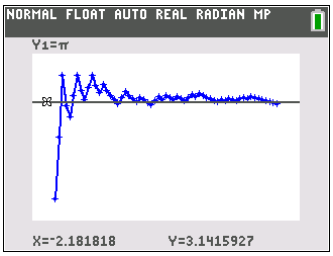
Example 8: Buffon's Needle
Simulate Buffon's needle experiment to estimate pi.

```
from ti_system import *
from random import *

# use shell before entering turtle environment
disp_clr()
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 60 ? ") or '60')
the_pies=[]
the_count=[]
# set up turtle environment
from turtle import *; t=Turtle()
t.hidegrid()
t.hideturtle()
t.pensize(2)
t.speed(0)
w,h=320,210
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+1):
  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(255,0,0)
# 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(0,255,0)
      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("1",the_count)
store_list("2",the_pies)
error=(pi-estimate)*100/pi
sleep(2)
disp_at(7,"    Press [clear] to continue","left")
disp_wait()
disp_clr()
print("Estimate of \n  Pi =", estimate,"\n Error =",error,"%\n")
print("Next, exit the Python App and\ncreate a Stat Plot with L1 and  L2 to see data, Zoom 9:StatPlot.")
print("\nPress [clear] to continue")
t.done()
```

Example file: **BUFFON**







Exit the Python App. Do a Stat Plot with L1 and L2. ZoomStat and Trace.

Challenges:

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

Note: Due to memory limitations you may see Memory Errors, even with the default input entries.