Python Turtle Graphics

Focus: Introduce Python programming via vector graphics using a cursor (turtle) on a canvas

Primary Objectives:

- Students will learn the basics of programming in Python
- Students will create their own Turtle designs by editing the code provided by the instructor

Secondary Objectives:

Estimated Time: 45-60 minutes

Resources:

School	TECH CORPS	Instructor/Volunteer
Student access to Trinket through a web browser		
Computers		
Projector		

Planning Notes:

- Student content can be found in a GitBook located in the Supplemental Resources section of this lesson.
- Please make sure you are familiar with Python, Turtle, and Trinket by reviewing the lesson plan and activity documents prior to the lesson. There are many different ways to accomplish the goals of this lesson in the platform.
- Feel free to let students change and create as they feel. This lesson is structured to help prepare students to create their own project, but there are many benefits to letting students work freely as they follow along.
    - They can change the colors to their liking.
    - They can move the turtle any way they want.
- One of the easiest ways to make interesting designs is to update the numbers in example code. If a student struggles to change these on their own, suggest a few numbers they can try to change.
- At minimum, show students the basics (drawing a triangle, using a loop) along with a few of the more interesting examples found in the Supplemental Resources section of this lesson. This should give students enough examples to be able to play around on their own.
- Students may work in pairs for this activity. Pair programming is a technique in which one student writes code, while the other student observes, constantly checking for mistakes. The two partners switch roles frequently. Pair programming adds the benefit that errors can be caught and fixed early in the development process.

Activity:

1. Introduction to Programming, Python, and Trinket
    - Ask the students what they know about programming and Python.
    - Ask the students if they like to draw and what they like to draw.
    - Direct the students to the starter Trinket link in the Supplemental Resources and explain the basic layout.
    - It is possible to increase the code font size from the hamburger menu.
2. Python Code-Along
    - Using the Python Code-Along activity document as a guide, allow students to follow along as they create their own project using Turtle.
    - Optional extra sections have blue labels and can be easily skipped for time.
    - For the code-along portions, the instructor should follow the instructions and write the code in a Trinket. The students should see the code the instructor types and type the same code on their computers. The instructor can ask leading questions and allow the students to suggest what to type next.
    - There are challenges built into the instructions, which give the students an opportunity to update the code on their own. Challenges are meant to be completed by the students without direct instruction from the teacher.
    - Loops are powerful for creating interesting designs. The students can update the number of repeats, along with the turn angle and the forward distance, to create cool shapes. Using the basic loop code, they can complete the many-pointed star challenge.
3. Python Challenges
    - Direct the students to the Python Challenges. A link has been provided in the Supplemental Resources section of this lesson.
    - The instructor can walk through one or two of the challenges (including the many-pointed star) to give students an idea of what they can create using Turtle.
    - The main purpose of the challenges is to inspire students to continue playing with the code. If they want to update their code in a completely different way, the instructor should encourage them to try new things as they wish!
4. Turtle Examples
    - These examples show the students more advanced Turtle graphics with Python.
    - The students can take the example code and customize it.
    - The students can also copy the code and update their projects with it.

Supplemental Resources:

- Student Content: https://hylandtechoutreach.github.io/CSEdWeek/
- Interesting Examples: https://hylandtechoutreach.github.io/CSEdWeek/TurtleExamples.html
- Starter Trinket Bitly Link: bit.ly/PyTrinket
- Python Challenges Bitly Link: bit.ly/PyChallenges
- Turtle Graphics Wiki: https://en.wikipedia.org/wiki/Turtle_graphics
- Trinket Python Documentation: https://trinket.io/docs/python#turtle
- RGB Color Picker: https://www.google.com/search?q=color+picker

Code-Along Activity – Turtle Graphics

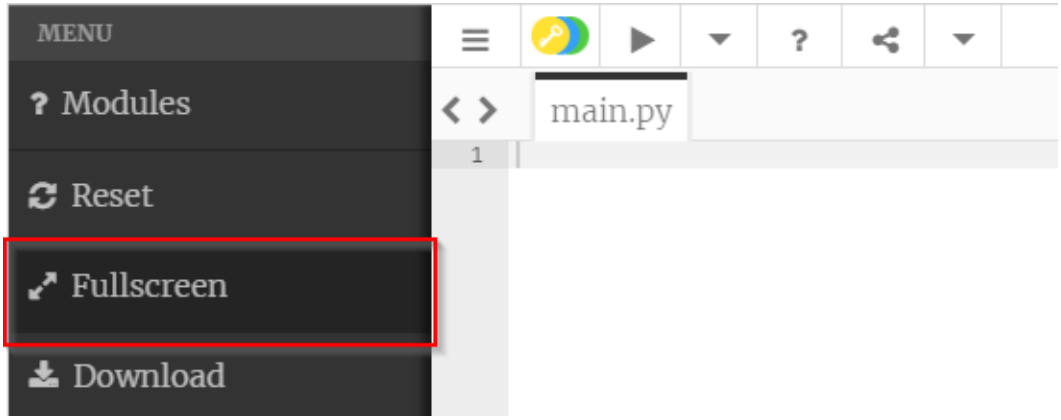
In this activity, students will write a Python application and play around with some two-dimensional graphics. This activity will provide a glimpse into the power of programming and show how fun it can be to write code!

Python is a popular programming language used by developers all over the world. It is a general-purpose language, which means that developers can use it to write scripts that do almost anything: create websites, analyze large data sets, control robots, design video games... the list goes on and on!

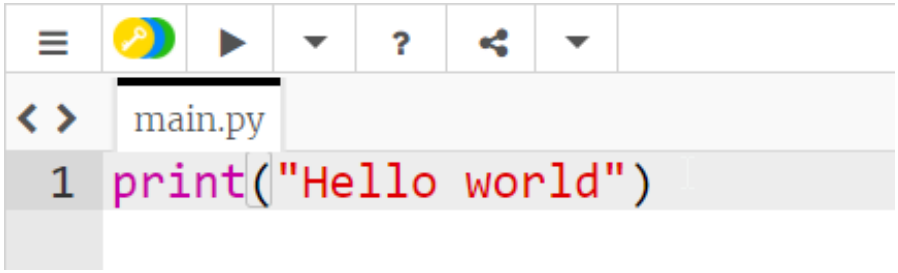
Trinket is a website that allows a developer to write and run code in a web browser. It works in any browser on any device. This makes it easy to practice coding and show off any programs that are created. During this activity, Trinket will be used to build some cool Python programs.

The first program a developer normally writes when learning a new programming language is a "Hello World" program. The goal of these programs is to display a message of "Hello World" to the user. In Python, that program is fairly simple. Follow the steps below to create a Python "Hello World" program:

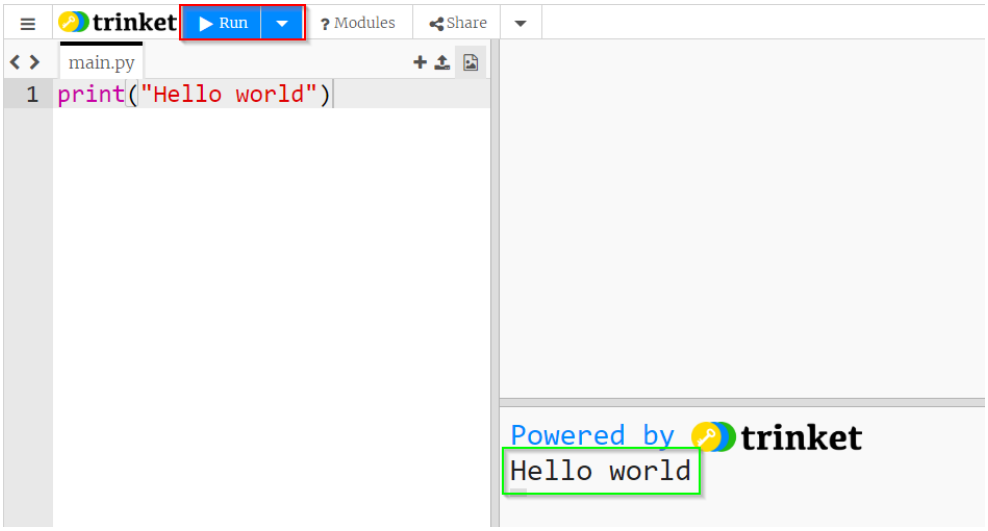
1. Navigate to bit.ly/PyTrinket to start building the program.
2. Open the hamburger menu in the upper left and select "Fullscreen" so it is easier to use:



3. In the main.py file, type the Python command to print a message to the screen:



4. Click the "Run" button to run the program:



When a developer "runs" their program, they are telling the computer to execute all the code they wrote and perform the commands they specified. Running the "Hello World" program in Python will print the message on the console screen.

Challenge: Change the Message
Update the simple "Hello World" program so that instead of saying "Hello World" it says something else!

Python Commands – Cooking Example

In Computer Science, commands are like instructions in a recipe. They tell the computer what to do. It can be helpful to consider this when writing code. All instructions must be *specific* and *precise* to be successful!

Imagine reading a recipe for cookies. If one of the steps in the recipe said “add sugar,” but did not specify *how much* sugar, it could be disastrous! The chef could end up with cookies that are way too sweet, or they could have no flavor at all. The author of a recipe needs to consider this and make sure every step has all the necessary information in the right format. Writing Python commands is very similar, only instead of a chef following a recipe, the *computer* follows instructions in Python.

Python Library – Turtle Graphics Introduction
Python has a multitude of **libraries** that allow developers to use pre-existing code to build their own applications. One such library is for turtle graphics, which are vector graphics that use a cursor (or "turtle") to create images on a Cartesian plane.

Using the turtle library, it is possible to create turtles that draw on the screen. The turtles will follow the commands provided in the Python code.

Comments in Python

Before continuing, it would make sense to get rid of the print command. However, it might be nice to have a reference to that code, in case it is helpful in the future. To do this, turn the code into a **comment** so the computer ignores it. In Computer Science, *comments* exist to provide information to developers, and they do not actually affect the program in any way. They can be incredibly helpful to understand code and keep it organized or to reference old code.

Add a hashtag (#) on the same line as the print command before any other character. This should turn the text green:

```
# print("Hello World")
```

Now, this line of code will do nothing!

Creating the Turtle
On the next line of the file, add the following command:

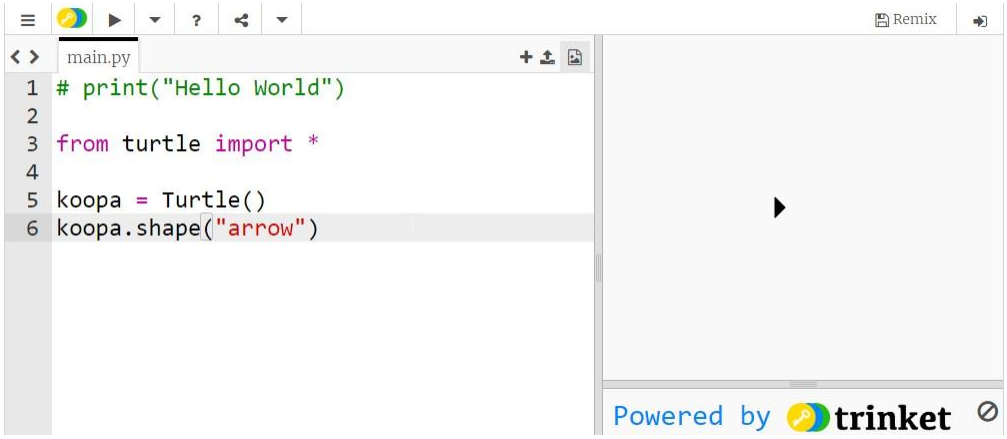
```
from turtle import *
```

This command tells the computer to bring in all the code from the turtle library so it is usable in the program. The next step is to create the turtle. First, add a blank line underneath the first line. Then, on the third and fourth lines of the file, add the following commands:

```
koopa = Turtle()
koopa.shape("arrow")
```

These lines create a turtle and set its shape. Make sure to copy the code exactly. Proper syntax (capitalization, parentheses, quotation marks, etc) is imperative. Without proper syntax, programs will not be able to run successfully and may fail in unexpected ways.

Click the "Run" button to see the "turtle" appear on the page:



Challenge: Change the Shape to "turtle"
Update the code so that instead of setting the shape to "arrow", it sets the shape to "turtle". Run the program again to see the shape shift.

Turtle Names

It is possible to name the turtle something other than “koopa” by changing the *variable* name. Variables in Python are containers for information. The “koopa” variable contains a turtle! Variable names must follow these rules:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)

Add Color
Black and white is boring, so we can add some color to the program! On the next line of the main.py file, add the following command:

```
koopa.color("green")
```

This command will turn the turtle green. Next, we can change the background color. Add a blank line, and then add the following commands:

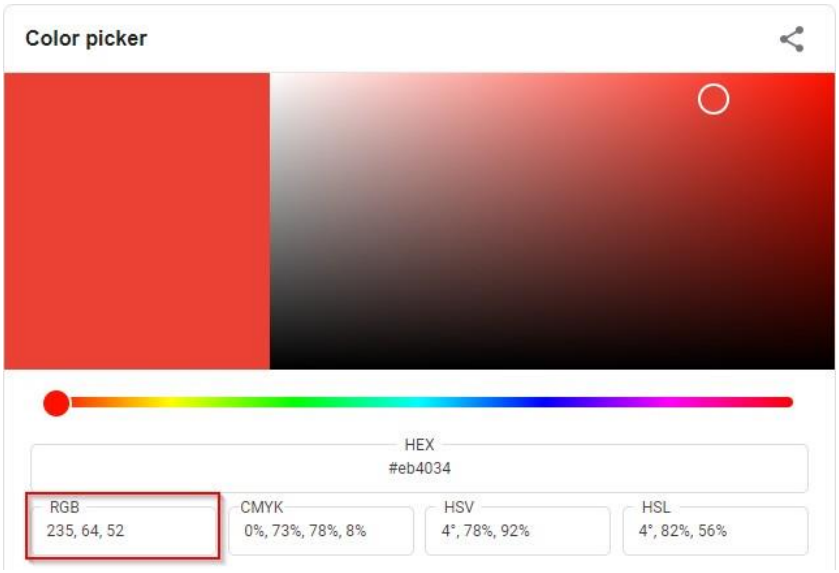
```
paper = koopa.getscreen()
paper.bgcolor("azure")
```

These commands get the *canvas* and set its background color to azure. The turtle’s canvas can be related to real-world space (e.g. whiteboard), or the Cartesian plane to help students understand how the space is laid out. Run the program again to see the new colors on the page.

Challenge: Change the Colors
Update the code so that instead of green and azure, the turtle and the background have different colors. For example, the turtle could be blue, and the background could be pink!

Custom Colors

In addition to the many built-in colors, it is possible to choose custom colors using RGB values. Search Google for “color picker” and select the desired color. Then, copy the RGB values:



Next, paste those values into the code where the color is set. It should look something like this:

```
koopa.color(235, 64, 52)
```

Turtle Graphics – Basic Abilities
Turtles can do a lot! Developers use the turtle abilities to create interesting designs and experiences.

Turtles can talk by displaying messages on the screen. Make a new line, and add the following command:

```
koopa.write("My name is Koopa")
```

Unfortunately, sometimes turtles cover their own messages! To move the message over, add some spaces at the beginning of the text.

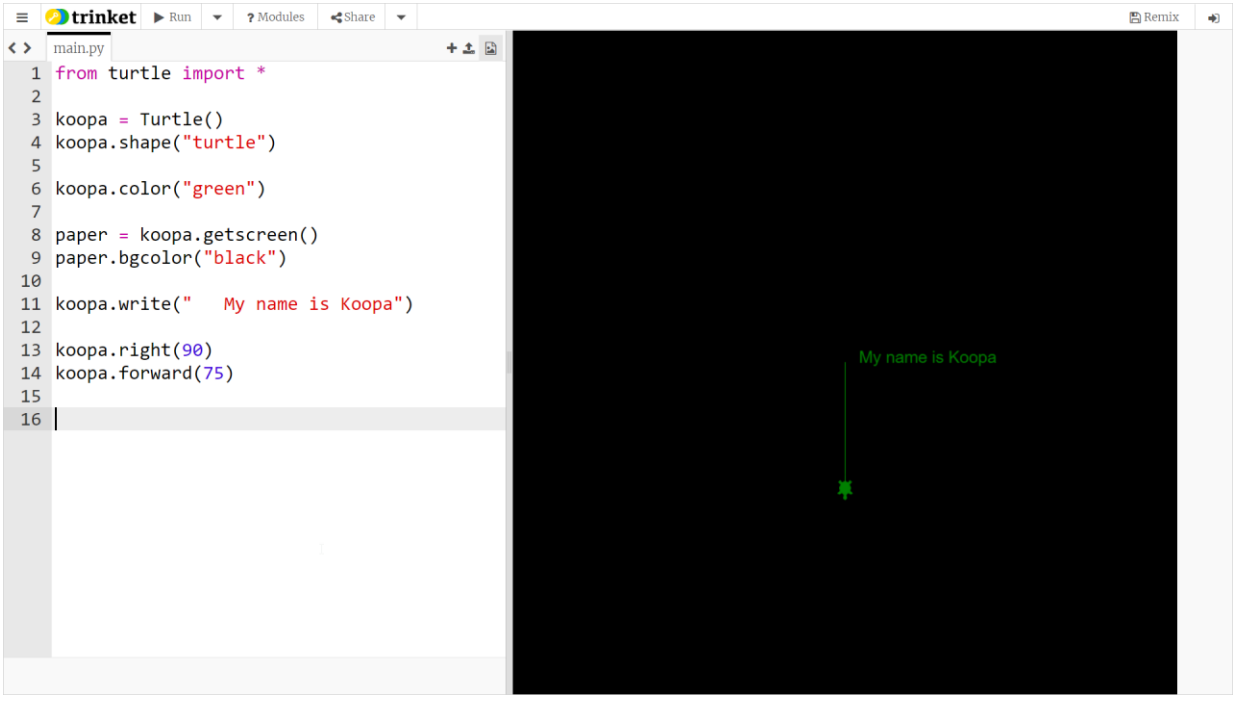
Turtles can change the direction they face by turning. They can turn either **left** or **right**. Use the following command on a new line to turn the turtle 90 degrees to the right so it faces down:

```
koopa.right(90)
```

Turtles can also move across the canvas. If they move **forward**, they will travel in the direction they currently face. Use the following command on a new line to move the turtle down 75 pixels on the screen:

```
koopa.forward(75)
```

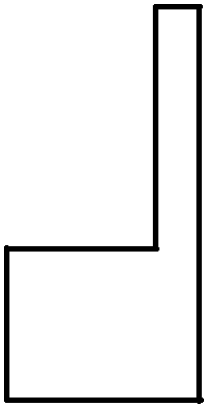
NOTE: If the students are not familiar with angles, or the Cartesian plane, the instructor may need to briefly explain how the turtle turns and travels.



Turtle Graphics – Drawing and Storytelling
Using these three simple abilities (*talking*, *turning*, and *moving*) it is possible to tell stories and create images that illustrate those stories. Use Koopa to draw a picture of a musical quarter note while describing the drawing.

Before writing code, developers usually spend time planning what they want to do from a broad perspective. This technique helps avoid problems, and provides a vision of the bigger picture.

Start by drawing a musical note on paper or on a whiteboard. Koopa has already begun this drawing, so continue from there. A quarter note should look something like this:



Try to figure out which commands will be necessary to complete the drawing. While drawing on paper or on a whiteboard, notice the path of the writing utensil. Keep in mind how far the cursor travels across the Cartesian plane, the different positions, and the angles between each line. Each of these will relate to the code.

Drawing the Base
First, have Koopa display a message and then continue down 75 for the base of the note:

```
koopa.write(" I like to draw on my computer")
koopa.forward(75)
```

This will complete the right side of the base. Next, draw the bottom line. This will require Koopa to turn to the right 90 degrees and move forward 100 pixels:

```
koopa.right(90)
koopa.forward(100)
```

Next, start draw the left side of the base. The turtle should turn right 90 degrees to face upward, and then move forward 50 pixels:

```
koopa.right(90)
koopa.forward(50)
```

At this point, Koopa should display a message to the user to continue the story:

```
koopa.write(" I can draw anything!")
```

Finally, to complete the base, Koopa should continue up the left side and then turn to draw the top:

```
koopa.forward(50)
koopa.right(90)
koopa.forward(85)
```

Drawing the Stem

Complete the quarter note by drawing the stem. Koopa should turn left 90 degrees to face upward, then travel 150 pixels forward:

```
koopa.left(90)
koopa.forward(150)
```

Next, to draw the top of the stem, Koopa should turn right 90 degrees and move 15 pixels to the right:

```
koopa.right(90)
koopa.forward(15)
```

Koopa still has a couple of things to say! Use the write command to display a message saying "I like music" next to the turtle:

```
koopa.write(" I like music")
```

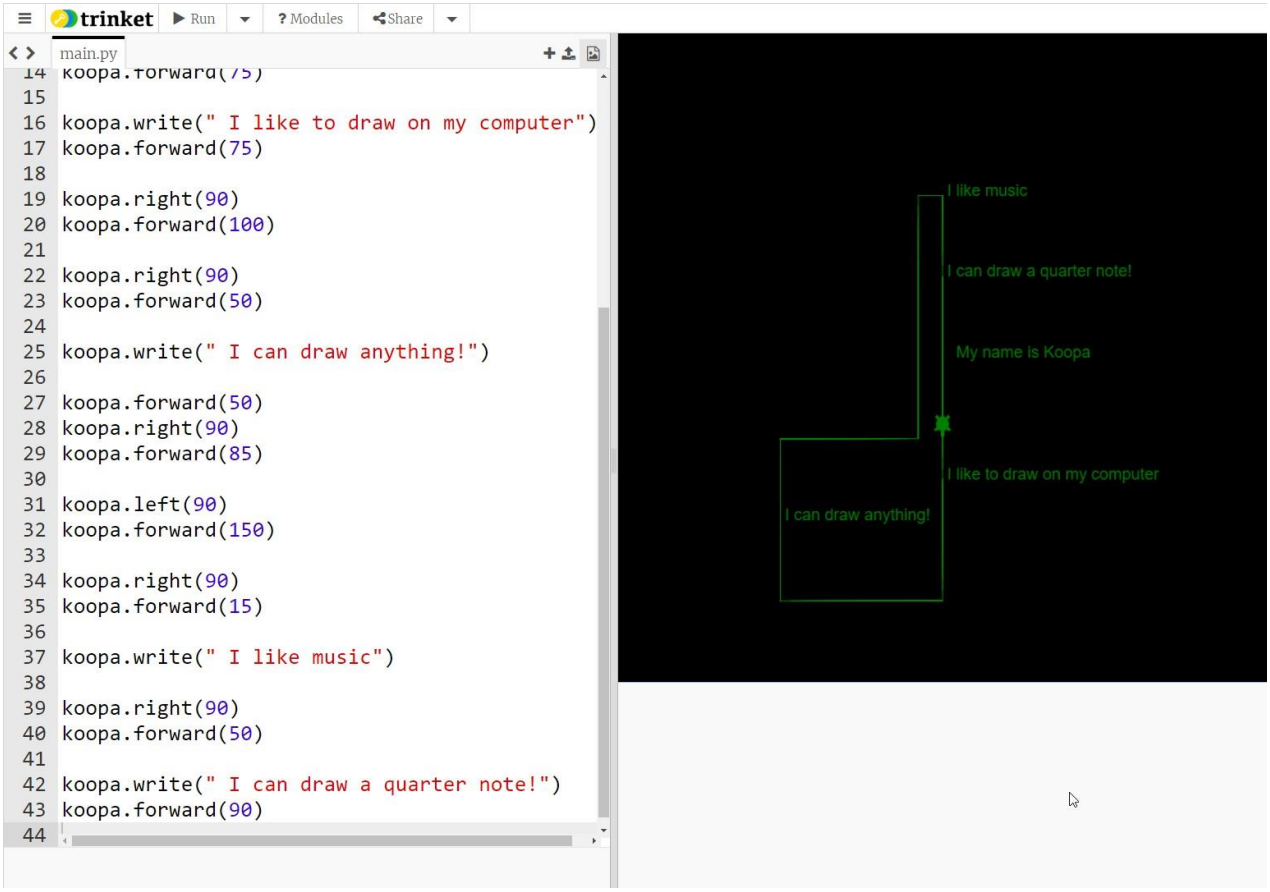
Now, all that's left is the right side of the quarter note. Turn Koopa to face downward and move 50 pixels down:

```
koopa.right(90)
koopa.forward(50)
```

Finally, Koopa should display a message saying "I can draw a quarter note!" and complete the drawing.

```
koopa.write(" I can draw a quarter note!")
koopa.forward(90)
```

With the quarter note code, the program should look something like this:



Filling in the Drawing

It is also possible to fill in drawings similar to using a paint bucket tool. To do this, make Koopa use the begin_fill command before drawing, and the end_fill command when the drawing is complete. Make sure to place these commands in the proper place!

```
koopa.begin_fill()

koopa.write(" My name is Koopa")
# ... code ...
koopa.write(" I can draw a quarter note!")
koopa.forward(90)

koopa.end_fill()
```

Turtle Graphics – Multiple Turtles

So far, the program has used one sole turtle to carry out all of the commands. However, it is also possible to create more than one turtle, and each turtle can do different things. We can create a new turtle and have it draw a **triangle**.

Creating the *new* turtle will be very similar to creating the original turtle. The only difference will be the name of the turtle (turtles are not allowed to have the same name in Python). Copy the commands used to create the koopa turtle and update them to create a turtle named Shelly. This code should go under everything thus far, so it does not interfere with the other turtle's code. Add the following commands at the bottom of the main.py file:

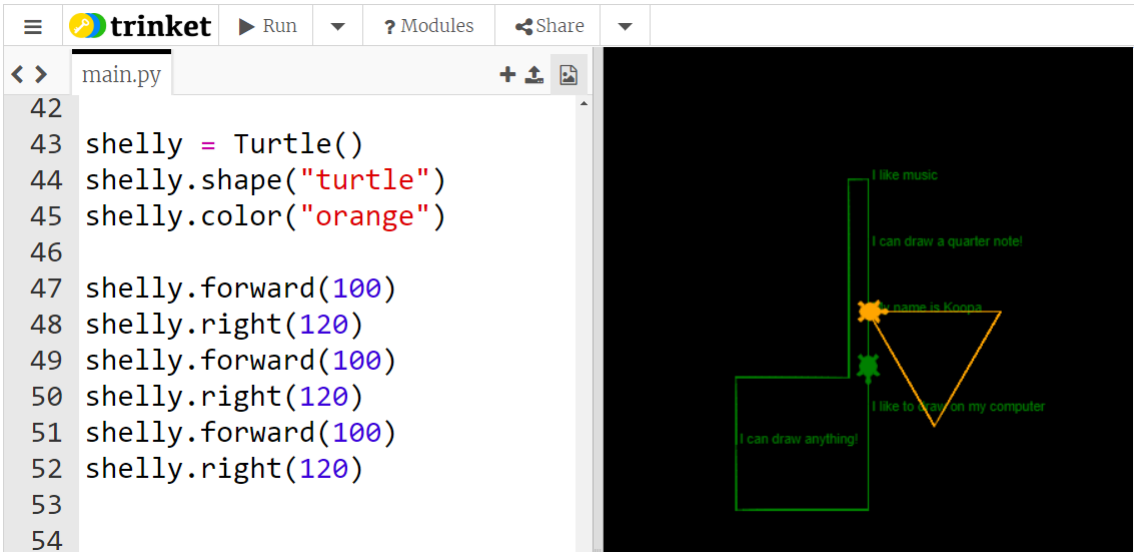
```
shelly = Turtle()
shelly.shape("turtle")
shelly.color("orange")
```

Run the program to see the new orange turtle appear at the end!

To draw a triangle, Shelly should draw three lines of equal length 100, turning 120 degrees between each line. Add the following commands at the bottom of the main.py file:

```
shelly.forward(100)
shelly.right(120)
shelly.forward(100)
shelly.right(120)
shelly.forward(100)
shelly.right(120)
```

Run the program to see Shelly draw the triangle!



Changing the Starting Point

When running the current program, Shelly draws all over Koopa's good work. Fix this by moving Shelly before the triangle drawing begins. Two commands make this possible: penup and goto.

NOTE: It can be helpful to visualize the movement of multiple turtles and their pens using a whiteboard with markers of different colors.

First, lift Shelly's pen before changing the position. This command should happen right after Shelly is created before the triangle is drawn. This will prevent errant pen marks from messing up the canvas:

```
shelly.penup()
```

Next, Shelly should move to a proper starting point on the screen. The top left corner will work well. Plotting this position on the Cartesian plane, the coordinates are (-150, 150):

```
shelly.goto(-150, 150)
```

Now, Shelly moves in a triangle at the right place, but there is still more to do. After moving, Shelly wants to provide a little introduction. Write a message to the screen saying " My name is Shelly... I like triangles" next to Shelly:

```
shelly.write(" My name is Shelly... I like triangles")
```

Now, Shelly has to move out of the way of the message on the screen. Use goto to move the turtle down 10 pixels:

```
shelly.goto(-150, 140)
```

Finally, Shelly has to put the pen back down in order to draw! Use pendown to accomplish this:

```
shelly.pendown()
```

Now Shelly can properly draw a triangle without interfering with any of Koopa's artwork. Next, see how Shelly can draw the triangle more efficiently using Loops.

Loops
Loops are a very powerful programming tool. They allow developers to repeat blocks of code automatically without having to write the commands over and over again. One of the major benefits is that if the developer needs to change something, they only have to change it in one place.

In the turtle code, there is a block of commands that could be improved with loops. Where do those repetitive commands occur?

```
shelly.forward(100)
shelly.right(120)
```

These commands repeat when Shelly draws the triangle! The two lines, shelly.forward and shelly.right, are repeated three times here. It takes up extra space in the file, and in order to update the program, each individual line would require update. For example, to make the triangle bigger, each 100 would have to change. Imagine if, instead of three times, the developer had to change three *hundred* lines. Or, if the program needed to dynamically update the number of times to repeat the code, that would be impossible without loops.

Use a for loop to automatically repeat the forward and right commands. Replace the triangle-drawing code with the following set of commands:

```
for x in range(3):
    shelly.forward(100)
    shelly.right(120)
```

This will make the two lines repeat 3 times!

Note that the two lines under the for statement are *indented*. This is how the loop knows which commands should repeat. The for x in range() part is the same for any number of repetitions.
The 3 specifies that the loop should repeat *three* times.

With the use of loops, the file is shorter and easier to maintain. Loops are one of the most important parts of programming!

Challenge: Change the Triangle size
Update the code so that the triangle Shelly draws is larger.

Final Code

```
from turtle import *

koopa = Turtle()
koopa.shape("turtle")
koopa.color("green")

paper = koopa.getscreen()
paper.bgcolor("azure")

koopa.begin_fill()

koopa.write(" My name is Koopa")
koopa.right(90)
koopa.forward(75)

koopa.write(" I like to draw on my computer")
koopa.forward(75)

koopa.right(90)
koopa.forward(100)

koopa.right(90)
koopa.forward(50)

koopa.write(" I can draw anything!")
koopa.forward(50)

koopa.right(90)
koopa.forward(85)

koopa.left(90)
koopa.forward(150)

koopa.right(90)
koopa.forward(15)

koopa.write(" I like music")
```

Final Code (continued)

```
koopa.right(90)
koopa.forward(50)

koopa.write(" I can draw a quarter note!")
koopa.forward(90)

koopa.end_fill()

shelly = Turtle()
shelly.shape("turtle")
shelly.color("orange")

shelly.penup()
shelly.goto(-150, 150)
shelly.write(" My name is Shelly... I like triangles")
shelly.goto(-150, 140)
shelly.pendown()

for x in range(3):
  shelly.forward(100)
  shelly.right(120)
```

Turtle Challenges

Go to bit.ly/PyChallenges (case-sensitive) to see the additional challenges along with some examples! Students may feel free to create their own projects at this time.