# Drawing Geometric Shapes with Python

| | | |
|---|---|---|
| Instructor: | Rory MacLysaght | Axiom Learning Demo |
| Level: | Beginner – no coding experience required | |
| Required materials: | web browser and internet connection | |

**Objectives**
Students will:
- learn to use a browser-based tool to write and test Python code
- be able to draw simple to complex geometric shapes using introductory Python commands
- learn how to create a "for" loop to repeat a block of code
- experiment and continue writing their own code to draw more complex shapes
- discover the relationship between angles and repetition required to create geometric shapes

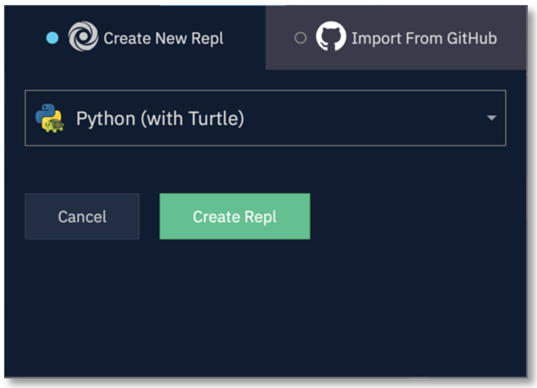
**Instructions:**

Go to this address: http://repl.it

(Note that this will work on any device with a browser, but it's going to be easier to write code if your device has a keyboard)

In the center of the screen you'll see a blue button that says "start coding".

Click that button, then scroll down until you see "Python (with Turtle)". This is important – don't select any of the other Python options – we definitely need "Python (with Turtle)".



Once you've selected "Python (with Turtle)", click the green Create Repl button to get started.

You have three screen areas. The large white box on the left is where you're going to write your code. The smaller white box at top right is our "stage" where you'll see the graphics your code creates, and the small black box at the bottom is for any text output from your program – or for error messages.

Type your first line of code:

```
import turtle
```

That by itself won't do a lot, but what we're doing is linking to some pre-built libraries of code that allow us to draw on the stage. You won't see that code, but it's available for you to call on in your program. In Python, these are called Modules.

But now, let's call our first useful piece from that code library. Skip a line, then type:

```
tom = turtle.Turtle()
```

What we've done here is pull in the code for turtle, and give it a name. The name can be anything you like – I've decided to call him "tom' because it's short and easy to type, but it can really be any name you like – as long as there are no spaces or special characters.

NOTE: Pay attention to capitalization, as that is very important to computer languages. In general, everything should be lowercase apart from this line we just typed, where the 2nd Turtle has a capital T.

Now he has a name, we can give our turtle some properties. I'm going to add a color, and also set the speed our turtle moves at to a low number, so you can see it work:

```
tom.color('purple')
tom.speed(1)
```

Now skip a line, and let's have our turtle do something. At the top of the window, you should see a green button that says "run >". Go ahead and click that now. Nothing much will happen, but you might notice a little purple triangle appeared in the middle of the stage. That's tom, our turtle! Skip another line and type this, then click the "run >" button again:

```
tom.forward(100)
```

Now, if everything went well, you should see tom move 100 spaces across the stage, leaving a purple trail behind. That's how we're going to draw some interesting shapes.

If instead you got a red error message in the black box at the bottom, carefully check your code and make any changes and hit run again.

Now, let's have tom do something more interesting. On the next line, have him turn right 90 degrees, then move forward another 100 places:

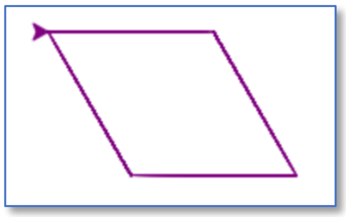
```
tom.right(90)
tom.forward(100)
```

Ok, now can you figure out what we'll need to do to get our turtle to draw a square? We just keep repeating the right and forward commands two more times.

But I think you can see that could get pretty tedious – you'd need seven repetitive lines of code just to draw a simple box. Luckily, that's where the power of Python comes in.

Go ahead and delete the previous forward, right, forward commands. We're going to change this to have our turtle turn 60 degrees. But instead of repeating it multiple times, now we're going to insert this piece of code in what's called a LOOP. A loop takes a piece of code, and repeats it a set number of times. Let's take a look at how this looks:

```
for i in range(2):
  tom.forward(100)
  tom.right(60)
  tom.forward(100)
  tom.right(120)
```



If all went well, you should now have a parallelogram.
Notice all the lines after the first one are indented by hitting the tab key. And the program helpfully shows vertical gray lines to indicate that.
One thing to be very, very careful with is to notice that our new line of code ends with a colon. Then the following lines are indented exactly one tab. This is how Python knows which parts of your code to repeat.
This line: `for i in range(2):` tells the program to run any lines of code that follow exactly two times – until you skip a line and get rid of the tab.

Again, if you got an error message, carefully check your code for any small typos.

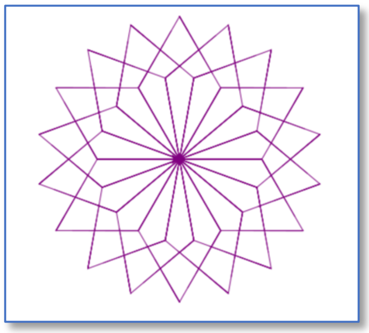
So now we have a parallelogram, but what if we used the same technique to repeat our parallelogram? Since the entire block, including the line that starts our loop, is needed for one parallelogram, we're going to take that block and wrap it in another loop.

Before we do that, let's speed up our turtle a little. Go back to that line where you set the speed to 1 and increase that to 10. That will let us see our shapes more quickly.

```
tom.speed(10)
```

Insert a new line before the first "for", and then carefully indent all the following lines by one more tab. But then, let's add one more line at the end, but this time hit the backspace or delete key to get rid of one of the tabs:

```
for x in range(18):
  for i in range(2):
    tom.forward(100)
    tom.right(60)
    tom.forward(100)
    tom.right(120)
  tom.right(20)
```



If all went well, you should now have a pretty multi-pointed star!

Now go ahead and play around with changing the first `range(18)` and the final `right(20)` numbers. Decrease one and raise the other; try 9 and 40, then 6 and 60, then 36 and 10.

Did you notice something? If you want a symmetrical circular shape, the two numbers need to equal 360 when multiplied.

Here's a final piece of code to make your patterns a little more interesting. Change the very first line to:

```
import turtle, random
```

Skip a line and type this:

```
colors = ['red', 'green', 'blue', 'purple', 'cyan', 'magenta', 'orange']
```

Now go to the end of the first (outer) for loop, hit enter, and type a new line (indented one tab like the line above it). So your first two lines of loop code should look like this:

```
  for x in range(36):
    tom.color(random.choice(colors))
```

The first new line we entered - `colors = ['red', 'green',` …etc. – set up a list of colors. You can add as many as you like, just wrap each in single quotes.
The, inside our first loop, we randomly pick a color from that list each time we draw our shape.

**Next Steps**

Now the rest is up to you. Experiment. Have fun. Here are a few things to try:

- What if the inner shape is a triangle?
- Since the turtle ends up back at the center, what happens if you add another new shape after drawing the first one?
- What happens if you change colors each time your inner "loop" draws a line with the `forward(100)` instruction?
- What if you draw circles, instead of polygons?

There's a huge variety of things you can do with Python turtles – you can have a lot of fun and create some cool drawings.

If you want to take this even further, I suggest you check out this page, which has all of the options for what you can do with your turtle:

https://realpython.com/beginners-guide-python-turtle/