

Some More Turtle Graphics

Introduction

In this worksheet we will revisit turtle graphics basics to make sure that you are happy with the main ideas and cover some aspects of turtle graphics that we skipped over last week. So this week we will look at

① **Generating Spirals using Turtle Graphics**

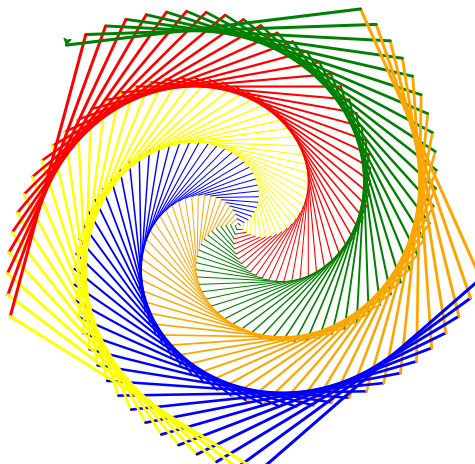
We will start with a simple turtle graphics program that generates a spiral and add then pimp it up, with colour and changing number of slides and line width.

② **To Fill or Not Fill — That is the Question**

So far all shapes have only drawn using their outline. We now want to see about filling these shapes in.

③ **Practice makes perfect**

Finally we will end with some sample diagrams constructed using turtle commands.





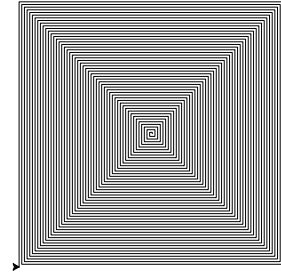
1 A Simple Turtle Program



Create a new file with the following code and save as `Square_Spiral_1.py`.

`Square_Spiral_1.py`

```
1 # draw a square spiral
2
3 import turtle
4
5 bob = turtle.Turtle()
6
7 for x in range(200):
8     bob.forward(2*x)
9     bob.left(90)
```



1.1 How it works

Let's break the program down line by line to see how it works.

line 1: The first line of `Square_Spiral_1.py` is a comment. A comment begins with a hash mark (`#`). Comments allow us to write notes in our programs to ourselves or to other humans who might read the program later. The computer doesn't read or try to understand anything after the hash mark; the comment is just for us to write something about what the program is doing. In this case, I've put the name of the program in the comment, as well as a brief description of what it does.

line 3: The next line of code gives us the ability to draw turtle graphics.

Importing code that's already been written is one of the coolest things about programming. If you program something interesting and useful, you can share it with other people and even reuse it yourself. Some cool Python programmers built a library—a reusable set of code—to help other programmers use turtle graphics in Python. When you type

```
3 import turtle
```

you are saying that you want your program to be able to use the code that those Python programmers wrote. The little black arrow in represents the turtle, drawing with its pen as it moves around the screen.

line 3: The next line of our program,

```
5 bob = turtle.Turtle()
```

tells the computer to create a new turtle called `bob`. By sending `bob` commands we can move `bob` and change its properties, such as colour and size.

line 7: The next line is the most complex. Here we're creating a `for` loop, which repeats a set of instructions a number of times (it loops through those lines of code over and over again). This particular loop sets up a range, or list, of 200 numbers from 0 to 199. (Computers almost always start counting at 0, not 1 like we usually do.) The loop then steps the letter



x through each of the numbers in that range. So x starts as 0, and then it becomes 1, then 2, and so on as it counts all the way up to 199, for a total of 200 steps.

This x is called a **loop variable**. A variable stores a value that can change, or vary, as we move through our program. We'll be using variables in almost every program we write, so it's good to get to know them early.

lines 8&9: The next two lines are indented, or spaced over from the left. That means that they are in the loop and go with the line above, so they'll be repeated each time x gets a new number in the range from 0 to 199, or 200 times.

1.2 So what happens?

Let's see what happens the first time Python reads this set of instructions. The command

```
8 bob.forward(2*x)
```

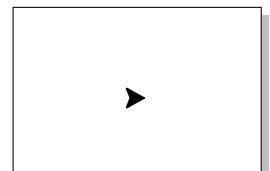
tells the turtle to move forward $2*x$ dots on the screen. Because x is 0, the pen doesn't move at all. The last line,

```
9 bob.left(90)
```

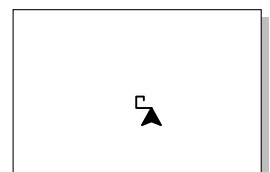
tells the turtle to turn left by 90 degrees, or a quarter turn.

Because of the **for** loop, the program continues to run, and it goes back to the starting position of our loop. The computer adds 1 to variable x to the next value in the range, and since 1 is still in the range from 0 to 199, the loop continues. Now x is 1, so the turtle moves forward 2 dots. The turtle then moves again to the left by 90, because of `bob.left(90)`. This continues again and again. By the time x gets to 199, the last time through the loop, the turtle is drawing the long lines around the outside of the square spiral.

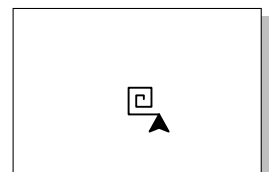
After one iteration: (This is a fancy way of saying at the start!) We have done nothing yet so all we see is bob the turtle.



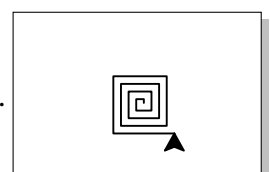
After iterations 0 to 4: The first four lines (the first line is tiny) are drawn (and now x stores 4).



After iterations 0 to 8: The first 8 lines are drawn (and now x stores 8).



After iterations 0 to 16: The first 16 lines are drawn (and now x stores 16).





The dots, or pixels, on your computer screen are probably too tiny for you to see them very well. But, as `x` gets closer to 200, the turtle draws lines consisting of more and more pixels. In other words, as `x` gets bigger, `bob.forward(x)` draws longer and longer lines.

The turtle arrow on the screen draws for a while, then turns left, draws some more, turns left, and draws again and again, with longer lines each time.

By the end, we have a hypnotising square shape. Turning left 90 degrees four times gives us a square, just like turning left four times around a building will take you around the building and back where you started.

The reason we have a spiral in this example is that every time we turn left, we go a little farther. The first line that's drawn is just 2 dots long (when `x = 1`), then 4 (the next time through the loop), then 6, then 8, and so on, all the way through 200 steps, when the line is 398 pixels long. Again, the pixels are probably so tiny on your screen that you can't easily see the individual dots, but they're there, and you can see the lines get longer as they contain more pixels.

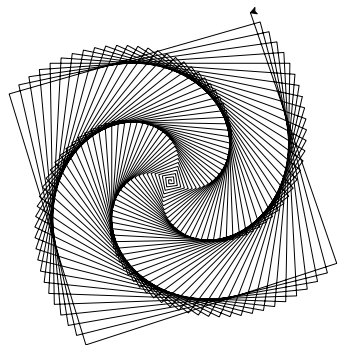
1.3 Let's Turn Things a Bit ...

Let's see what happens when we change one of the numbers in the program. One way to learn new things about a program is to see what happens when you change one part of it. You won't always get a pretty result, but you can learn even when something goes wrong.

 Create a new file with the following code and save as `Square_Spiral_2.py`.

`Square_Spiral_2.py`

```
1 # draw a square spiral
2
3 import turtle
4
5 bob = turtle.Turtle()
6
7 for x in range(200):
8     bob.forward(2*x)
9     bob.left(91)
```



All we did was to change the last line of the program from `bob.left(90)` to `bob.left(91)`. Making a 90-degree left turn creates a perfect square. Turning just a little more than 90 degrees — in this case, 91 degrees every turn — throws the square off just a bit. And because it's already off a bit when it makes the next turn, our new shape looks less and less like a square as the program continues. In fact, it makes a nice spiral shape that starts to swirl to the left like a staircase, as you can see in .

This is also a nice visual to help you understand how being off by just one number can drastically change the result of your program. One degree doesn't seem like a big deal, unless you're off by one degree 100 times (which adds up to 100 degrees), or 1,000 times, or if you're using a program to land an airplane ...

 If you don't know how degrees work yet, don't worry about it for now. Just play with the numbers and see what happens.

- How can we change the size of the spiral?
- How can we change the space between the lines in the spiral?



- How can we change the space between the lines in the spiral without changing the overall size?

1.4 Don't be a square

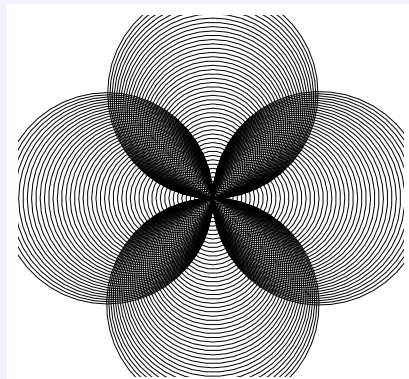
Turtle graphics can draw lots more interesting shapes than just straight lines. We'll come back to the square shape again in the next section, but let's take a short detour to check out more of the Python Turtle library.

Let's change one more line of code: `bob.forward(2*x)`. We saw earlier that this command moves the turtle forward `x` pixels and draws a straight line segment; then the turtle turns and does it again. What if we changed that line of code to draw something more complex, like a circle?

Fortunately for us, the command to draw a circle of a certain size, or radius, is as simple to code as the command to draw a straight line. Change `bob.forward(2*x)` to `bob.circle(x)`, as shown in the following code.

Circle_Spiral_1.py

```
1 # draw a circle spiral
2
3 import turtle
4
5 bob = turtle.Turtle()
6
7 for x in range(200):
8     bob.circle(x)
9     bob.left(90)
```



Changing one command from `bob.forward` to `bob.circle` gave us a much more complex shape, as you can see in . The `bob.circle(x)` function tells the program to draw a circle of radius `x` at the current position. Notice that this drawing has something in common with the simpler square spiral shape: there are four sets of circle spirals just like there were four sides to our square spiral. That's because we're still turning left just a little over 90 degrees with the `bob.left(91)` command. If you've studied geometry, you know that there are 360 degrees around a point, like the four 90-degree corners in a square ($4 \times 90 = 360$). The turtle draws that spiral shape by turning just a little more than 90 degrees each time around the block.

- We have removed the multiplication by two in the drawing of a circle but the shape is still as large as before (in fact it is larger) Why?

Think radius versus diameter of a circle.

A radius of `x` means that the diameter, or total width, of the circle will be two times `x`. In other words, `bob.circle(x)` draws a circle 2 pixels across when `x` is equal to 1, 4 pixels across when `x` is 2, all the way up to 398 pixels across when `x` is 199.

- Also drawing a circle is very slow. so consider speeding up bob by using the command

```
6 bob.speed("fastest")
```



1.5 Adding a touch of colour

These spirals are nice shapes, but wouldn't it be cooler if they were a bit more colourful? Let's go back to our square spiral code and add one more line to our program, right after the `bob = turtle.Turtle()` line, to set the pen colour to red:

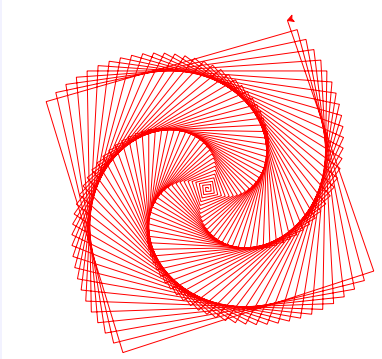
 Create a new file with the following code and save as `Square_Spiral_3.py`.

`Square_Spiral_3.py`

```

1 # draw a square spiral
2
3 import turtle
4
5 bob = turtle.Turtle()
6 bob.color("red")
7
8 for x in range(200):
9     bob.forward(2*x)
10    bob.left(91)

```



Try replacing "red" with another common colour, like "blue" or "green", and run the program again. You can use hundreds of different colours with the Turtle library, including some weird ones like "salmon" and "lemon chiffon". Or, like we did last week, you can defined your own colours by saying how much red , green and blue is to be mixed together.

Making the whole spiral a different colour is a nice step, but what if we wanted to make each side a different colour? That's going to take a few more changes to our program.

1.6 A four-colour spiral

Let's think through the — that is, the set of steps — that will turn our one-colour spiral into a four-colour spiral. The complete code is on the next page but try to write this yourself with following the structure given below

- Import the turtle module and set up a turtle.
- Tell the computer which colours we'd like to use.

```
7 colors = ["red", "yellow", "blue", "green"]
```

- Set up a loop to draw 200 lines in our spiral.
- Pick a different pen colour for each side of the spiral.

```
10 bob.color(colors[x%4])
```

The "%" computes the remainder when we divide x by four so when

x variable	=	0	1	2	3	4	5	6	7	8	9	10	11	...
x%4	=	0	1	2	3	0	1	2	3	0	1	2	3	...

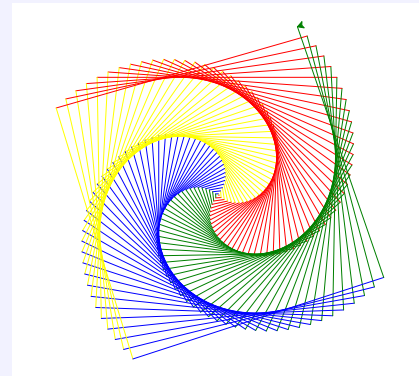
- Move the turtle forward to draw each side.
- Turn the turtle left to get ready to draw the next side.



Create a new file with the following code and save as `Color_Square_Spiral.py`.

Color_Square_Spiral.py

```
1 # draw a square spiral
2
3 import turtle
4
5 bob = turtle.Turtle()
6
7 colors = ["red", "yellow", "blue", "green"]
8
9 for x in range(200):
10     bob.color(colors[x%4])
11     bob.forward(2*x)
12     bob.left(91)
```



1.7 Changing the background colour

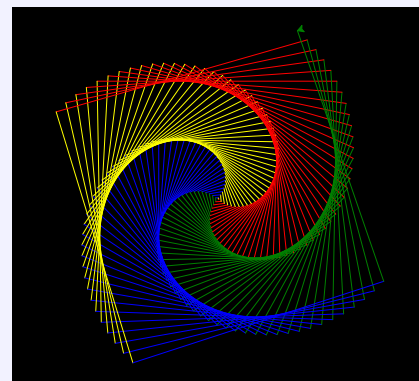
Let's mix things up a bit again to see if we can make the generated image better. One thing that we could do is to change the background to a dark colour so that the colours will have better contrast.



Save `Color_Square_Spiral.py` as `Color_Square_Spiral_2.py` and insert lines 7 and 8 as shown below.

Color_Square_Spiral_2.py

```
1 # draw a square spiral
2
3 import turtle
4
5 bob = turtle.Turtle()
6
7 window = turtle.Screen()
8 window.bgcolor("black")
9
10 colors = ["red", "yellow", "blue", "green"]
11
12 for x in range(200):
13     bob.color(colors[x%4])
14     bob.forward(2*x)
15     bob.left(91)
```



We have inserted two lines:

line 7: Gives us access to the `window` within which our turtle, `bob` lives.

line 8: We set the background colour using command `bgcolor`.



1.8 How many sides?

OK, we have done a lot with spirals and it is time to move to something new. Before we do that, have a look at the following program

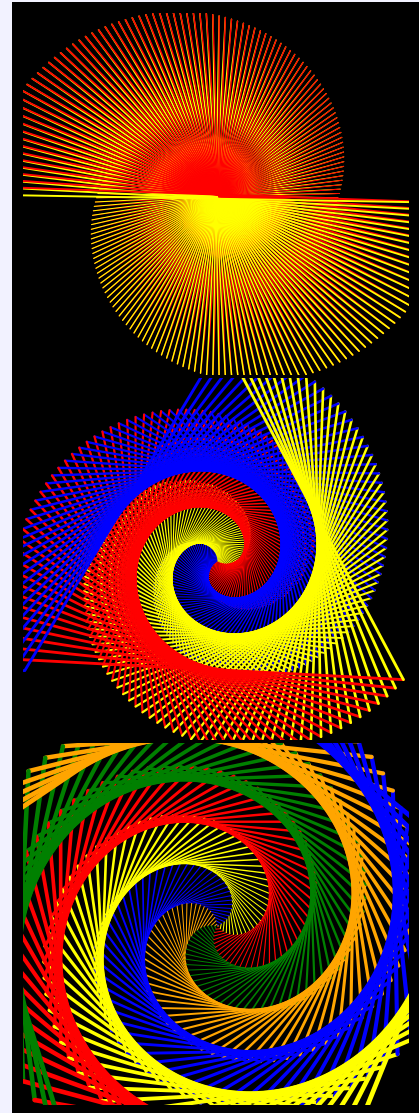


Create a new file with the following code and save as `Color_Spiral.py`.

Change the value of `sides` to get the diagram shown.

`Color_Spiral.py`

```
1 # draw any spirals
2
3 import turtle
4
5 bob = turtle.Turtle()
6
7 window = turtle.Screen()
8 window.bgcolor("black")
9
10 # pick between 2 and 6 sides
11 sides = 6
12
13 colors = ["red", "yellow", "blue",
14           "orange", "green", "purple"]
15
16 for x in range(360):
17     bob.color(colors[x%sides])
18     bob.forward(x * 3/sides + x)
19     bob.left(360/sides + 1)
20     bob.width(x*sides/200)
```





2 Why are my Shapes so Empty?

Remember our first turtle program that just drew a square:

Square.py

```
1 import turtle
2
3 bob = turtle.Turtle()
4
5 bob.color("red")
6
7 for x in range(4):
8     bob.forward(100)
9     bob.left(90)
```



It would be nice if we could fill this square, or any shape, with a colour of our choice. We can



Create a new file with the following code and save as Square_Fill.py.

Square_Fill.py

```
1 import turtle
2
3 bob = turtle.Turtle()
4
5 bob.color("red")
6 bob.fillcolor("blue")
7
8 bob.begin_fill()
9 bob.width(4)
10 for x in range(4):
11     bob.forward(100)
12     bob.left(90)
13 bob.end_fill()
```



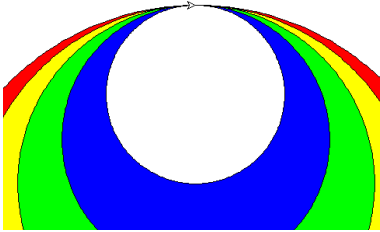
- We set the fill colour using the command `fillcolor`.
- Note: I increased the line width, using the command `width`, so that it was easier to see the outline.



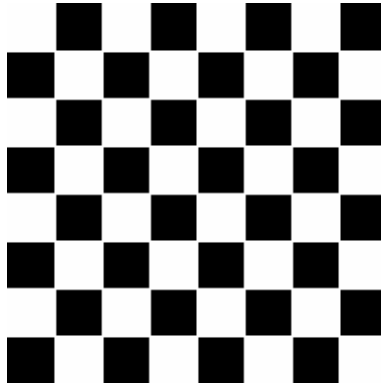
3 Exercises

Try to create each of the following diagrams. All of the diagrams can be draw using the turtle commands that we covered this week and last week.

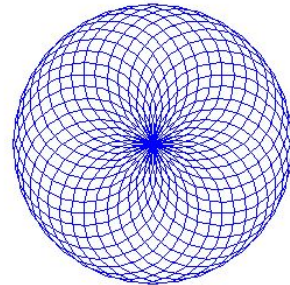
circles



chess



more_circles



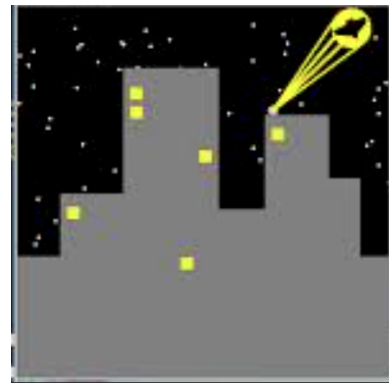
face



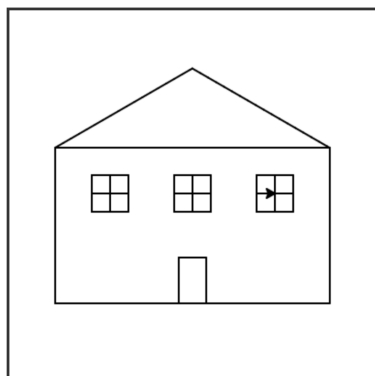
party



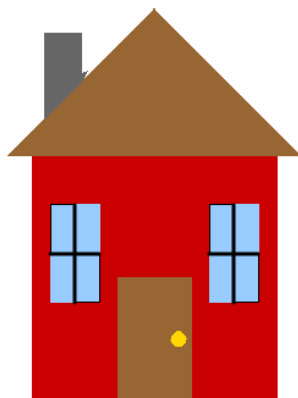
city



house_outline



house_small



house_big

