DigiLocal

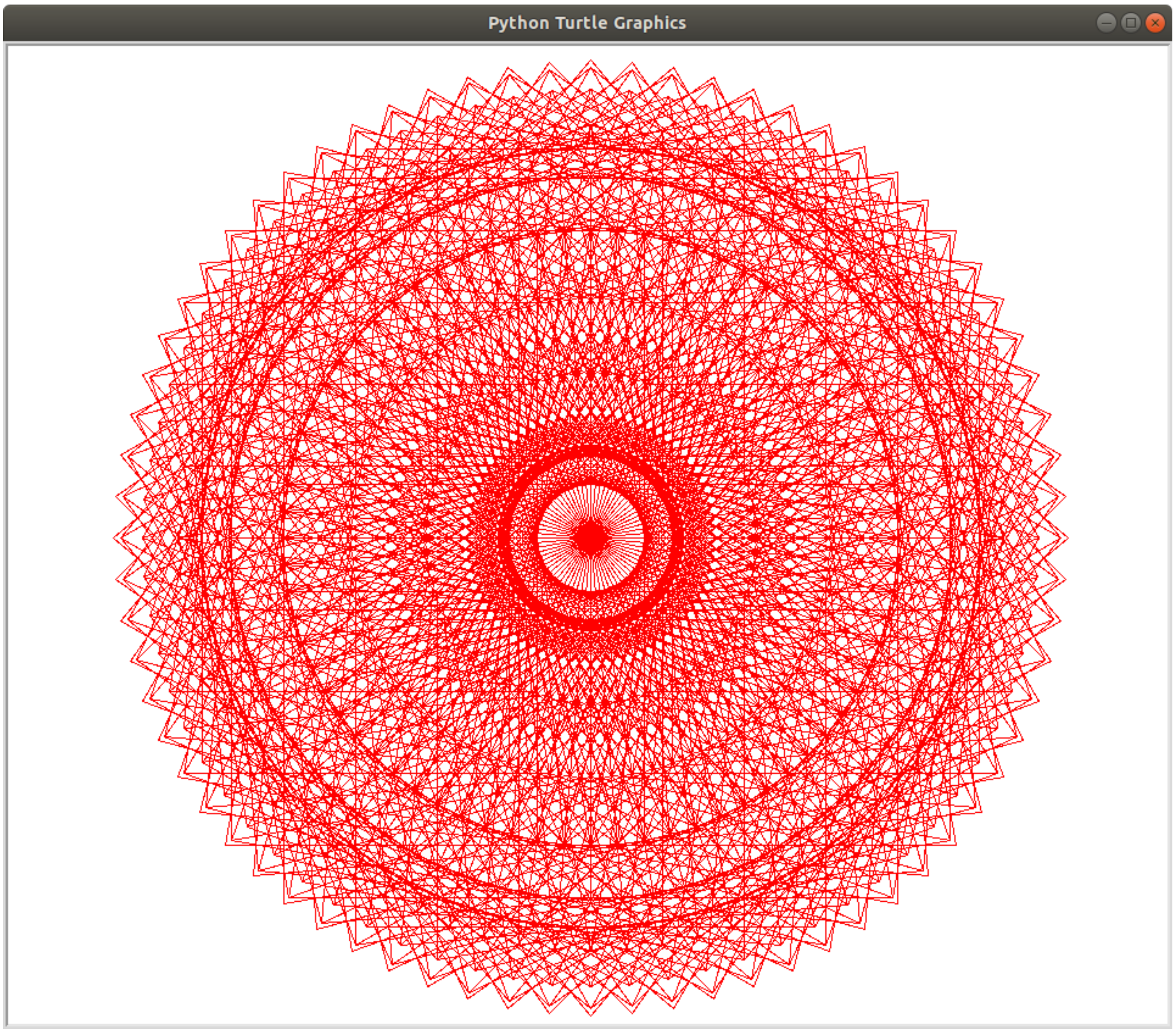
MAKING OUR DIGITAL FUTURE

Terrific Turtles!

INTRODUCTION

The turtle library in python gives us a very quick path to drawing shapes, patterns and playing cool games with turtles.

In this project we'll learn about the basics of controlling turtles and drawing patterns.

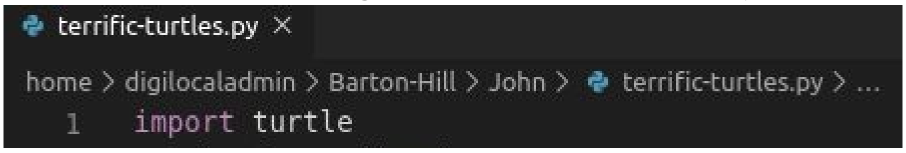


Step One: Getting started

- Open VS Code from the icon in your side bar.

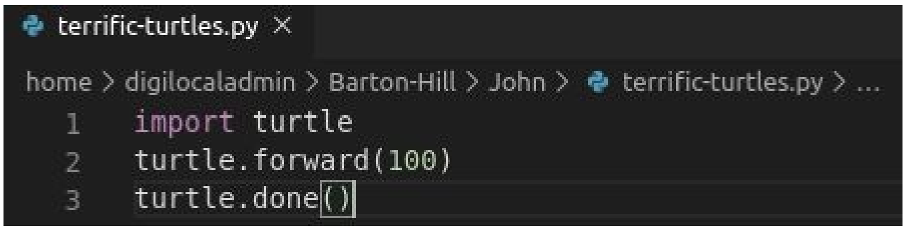
- We need a new file to begin editing. Start a new file and save it as 'terrific-turtles.py'.

- The core of our game is the turtle library so let's import that first.



```
terrific-turtles.py ×
home > digilocaladmin > Barton-Hill > John > terrific-turtles.py > ...
import turtle
```

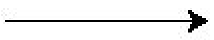
- We can test that everything is working with a couple of simple commands.



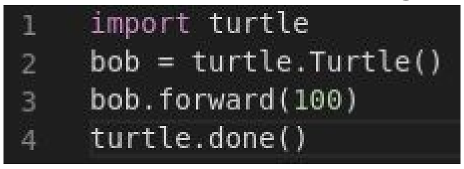
```
terrific-turtles.py ×
home > digilocaladmin > Barton-Hill > John > terrific-turtles.py > ...
import turtle
turtle.forward(100)
turtle.done()
```

Test your code!

- You should have a small arrowhead at the end of a longer line.

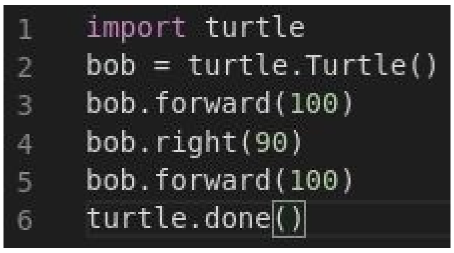


We can make things a little easier for ourselves by naming our turtle.



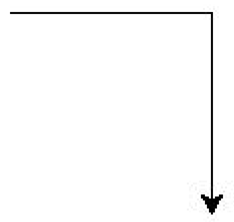
```python
import turtle
bob = turtle.Turtle()
bob.forward(100)
turtle.done()
```

We can also make bob turn.

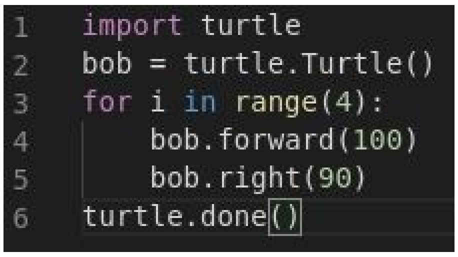


```python
import turtle
bob = turtle.Turtle()
bob.forward(100)
bob.right(90)
bob.forward(100)
turtle.done()
```

Test your code!

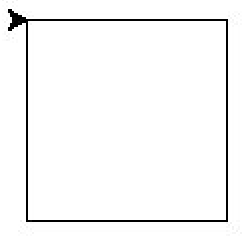


We could keep adding lines like that to make bob go in a square, but it's much easier to add a for loop and let the computer do the hard work.



```python
import turtle
bob = turtle.Turtle()
for i in range(4):
    bob.forward(100)
    bob.right(90)
turtle.done()
```

Test your code!



Challenge time!

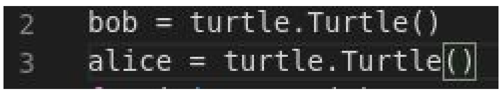
Can you make bob draw a triangle?

Can make bob draw a hexagon (6 sides)?

Can you make bob draw a house?

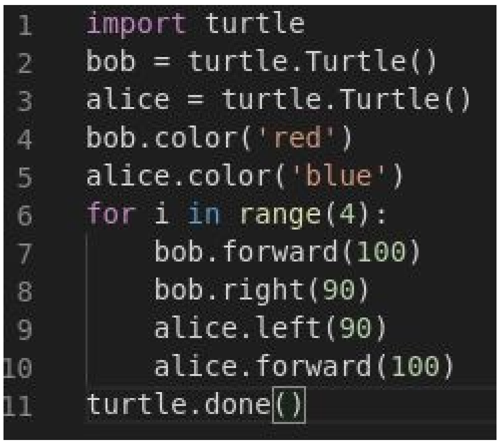
Step Two: More turtles!

- Let's make a second turtle.



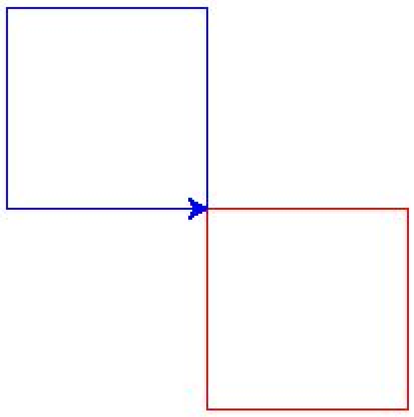
```
bob = turtle.Turtle()
alice = turtle.Turtle()
```

- Now we can set the parameters for each turtle. There are lots of things we can change. Here are a few. We can change the color of turtles (note the American spelling).

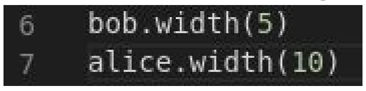


```
import turtle
bob = turtle.Turtle()
alice = turtle.Turtle()
bob.color('red')
alice.color('blue')
for i in range(4):
    bob.forward(100)
    bob.right(90)
    alice.left(90)
    alice.forward(100)
turtle.done()
```

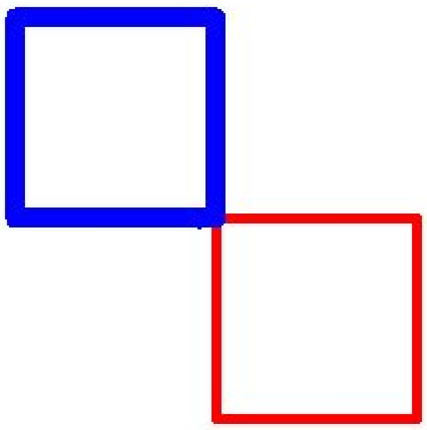
Test your code!



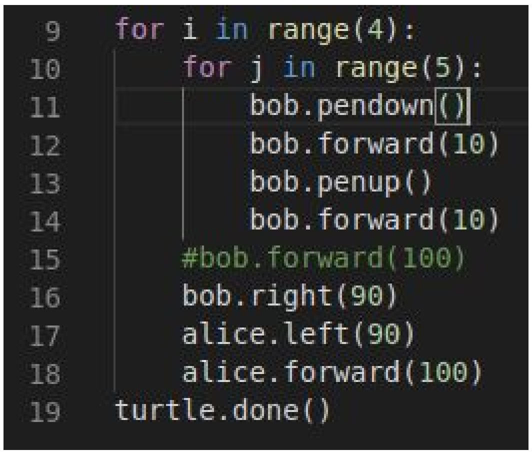
We can change the line width they draw.



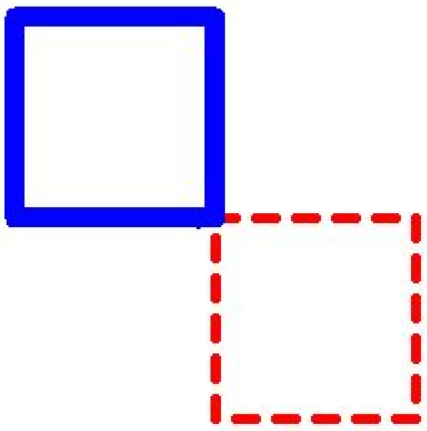
```
bob.width(5)
alice.width(10)
```



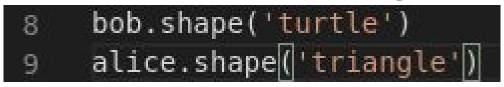
- We can also control if they have the penup, or down. Notice, we haven’t written the whole loop out again, we just comment out line 15 (using the #), and add in lines 10 to 14.



```
 9  for i in range(4):
10      for j in range(5):
11          bob.pendown()
12          bob.forward(10)
13          bob.penup()
14          bob.forward(10)
15      #bob.forward(100)
16      bob.right(90)
17      alice.left(90)
18      alice.forward(100)
19  turtle.done()
```

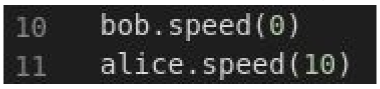


- We can also change the shape of bob and alice.



```
bob.shape('turtle')
alice.shape('triangle')
```

- We can change their speed.



```
bob.speed(0)
alice.speed(10)
```

- Note that speed = 0 is the fastest possible speed. Otherwise speed is a number between 1 and 10, anything over 10 is rounded down to just 10.

Test your code!
A few times.

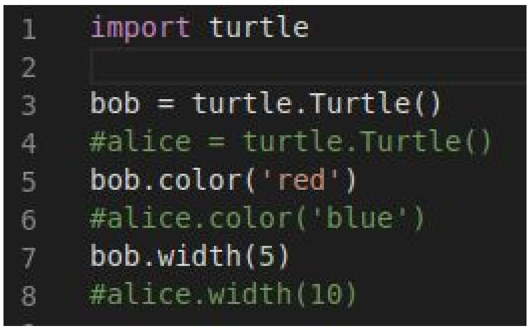
Challenge time!

What things could you draw with different turtles?

Step Three: Advanced turtling

- Let’s go back to just bob. Comment out alice (you can use her again later), and we use the loop command to create some amazing patterns.

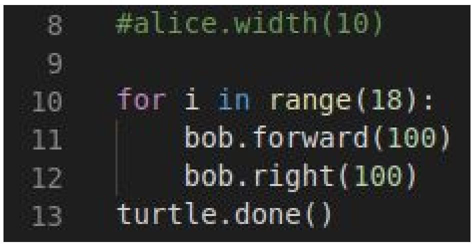
- You may want to start a new file, or ‘save as’ to create a copy and work with that. Edit your code until you have the starting programme below.



```
1  import turtle
2
3  bob = turtle.Turtle()
4  #alice = turtle.Turtle()
5  bob.color('red')
6  #alice.color('blue')
7  bob.width(5)
8  #alice.width(10)
```

- We’re going to add our new code at line 10. Notice that we’ve left a blank line between our import. In python these blank lines are ignored, but they may your code easier to read.

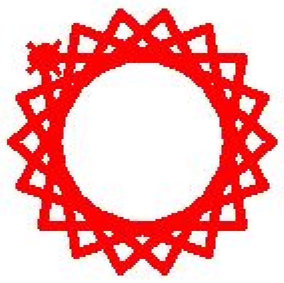
- First we can draw a simple star pattern. Let's add a simple loop to draw our star.



```
#alice.width(10)

for i in range(18):
    bob.forward(100)
    bob.right(100)
turtle.done()
```

Test your code!

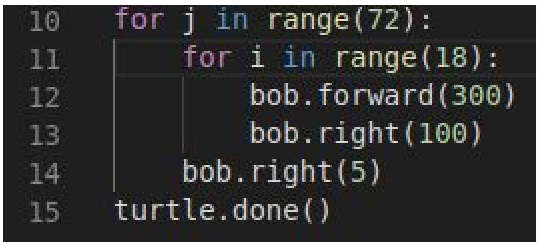
- This should draw a star like this.



- Try different angles and number for the range of your loop.

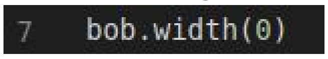
- We can also put loops inside loops!

- ✓ Modify your code to include the new loop at line 10. Remember to increase the indent of your original loop by one <TAB> so it looks like the picture below. We also change the forward command so we can see the new star more easily.



```
10  for j in range(72):
11      for i in range(18):
12          bob.forward(300)
13          bob.right(100)
14      bob.right(5)
15  turtle.done()
```

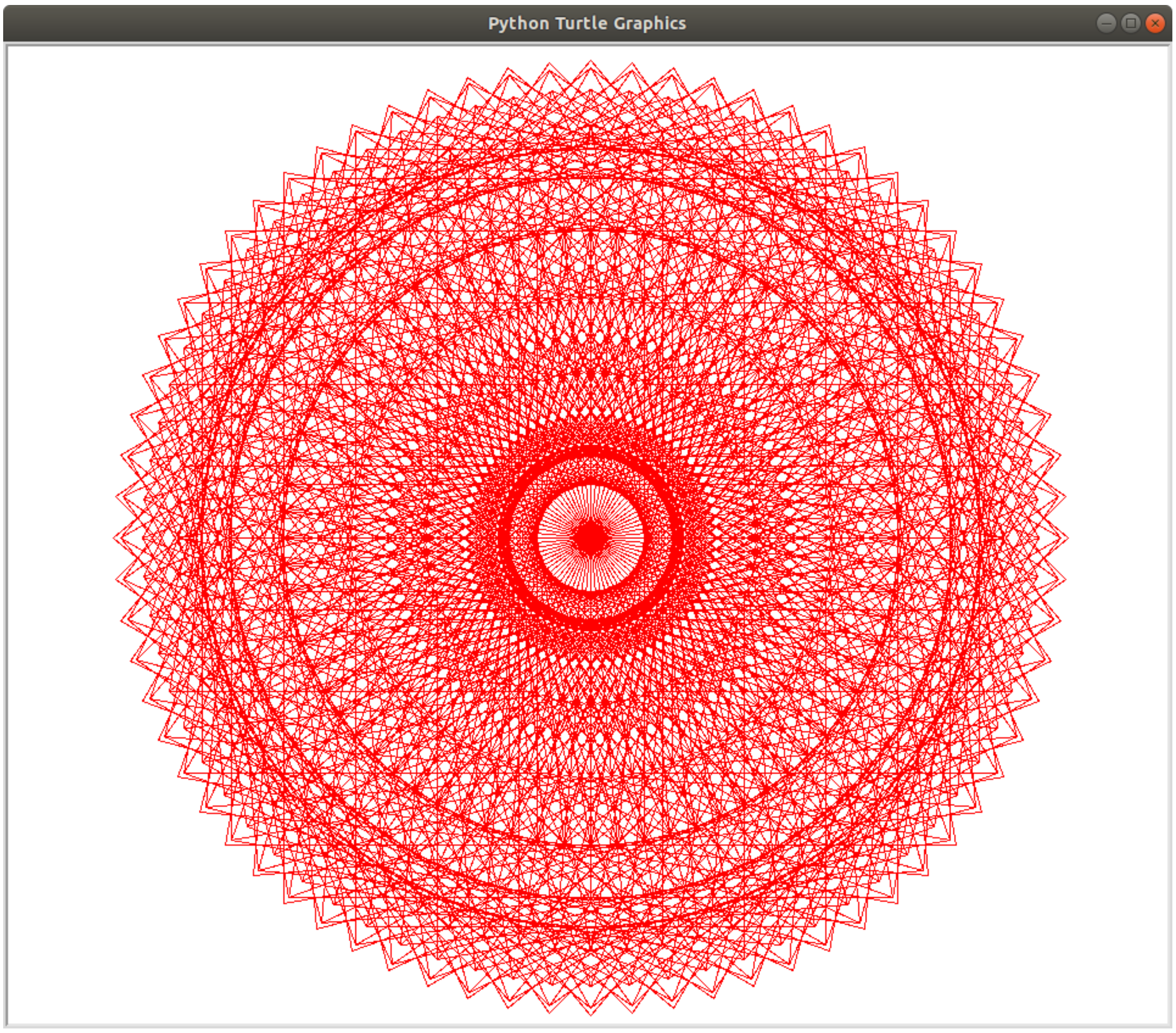
- ✓ The only last thing to do is make the line thinner, otherwise it'll end up one big red blob.



```
7   bob.width(0)
```

- ✓ This time, after we draw our star, we turn 5 degrees to the right, and draw another, then another, then 72 in total. Your pattern should now look like this:

▷ Test your code!



Challenge time!

What other patterns can you make with different angles, lengths, and loops?

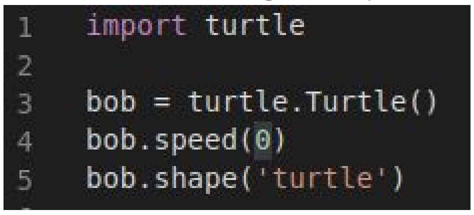
Step Four: Colourful turtling

- Our turtle colour is given as ‘red’, but we can control the colour much more accurately. In fact the colour is a combination of red, green, and blue.

- The full command is bob.color(red, green, blue) where each of the colours can take a value from 0 (nothing) to 1 (full). So red is the same as (1, 0, 0), and blue is (0, 0, 1).

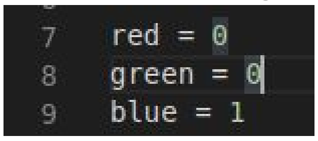
- We can use this to make a rainbow pattern!

- This is going to be quite a major re-write of our code, so you may want to start a new file. We start out the same, by defining bob as a turtle, and setting his speed to 0 (the fastest).



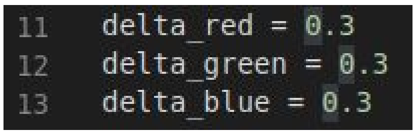
```python
import turtle

bob = turtle.Turtle()
bob.speed(0)
bob.shape('turtle')
```

- Next we need to define 3 variables to hold our values for red, green and blue, and give them starting values.



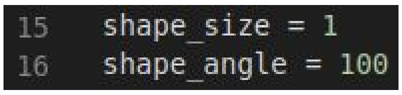
```python
red = 0
green = 0
blue = 1
```

- We want our colours to change. The fancy way of saying change, or difference between, is delta. So we call the amount we change each colour in each step.



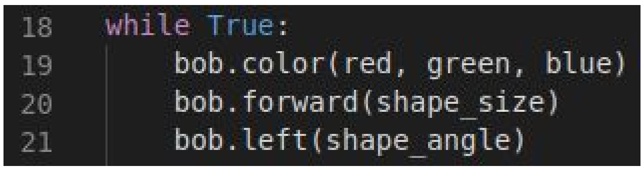
```python
delta_red = 0.3
delta_green = 0.3
delta_blue = 0.3
```

- We need quite a lot of lines so we can see all the colours. So we'll start out with a small shape and make it get larger. We'll also set the angle we turn at each loop.



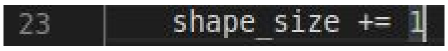
```python
shape_size = 1
shape_angle = 100
```

- We don't really want our programme to end, instead draw an endless colourful pattern. So we use a while True loop (the python equivalent of Scratch forever loop). We set bob's colour, make him move, and then turn.



```python
while True:
    bob.color(red, green, blue)
    bob.forward(shape_size)
    bob.left(shape_angle)
```

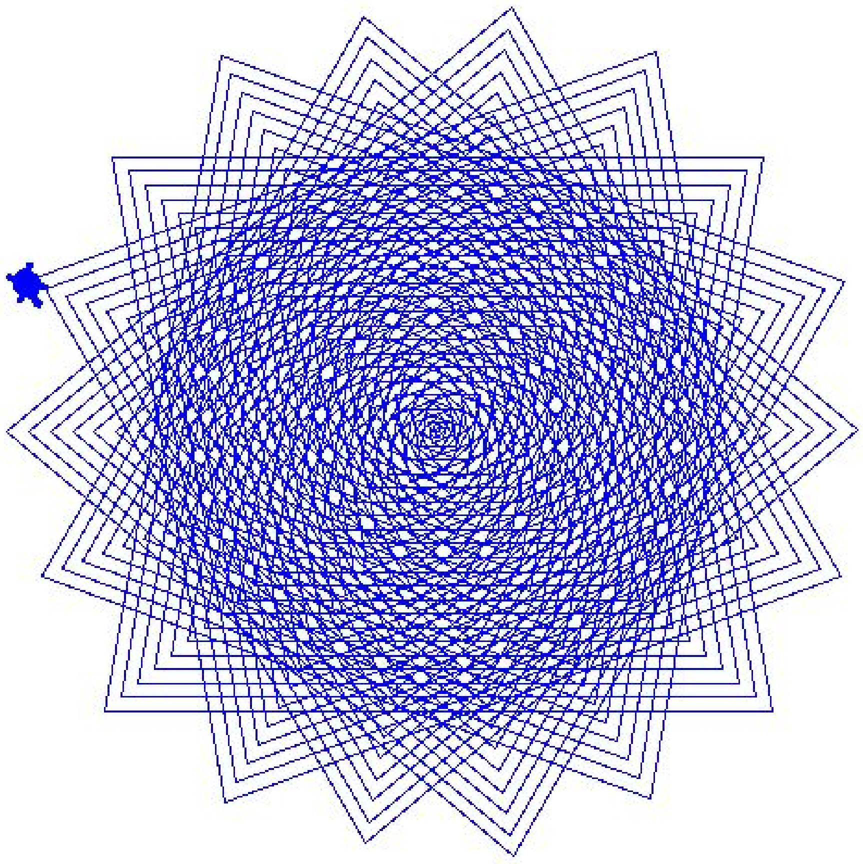
- Next we make the shape a little bit bigger.



```python
    shape_size += 1
```

Test your code!

- You should have a pattern like the one below, you'll need to close the window to continue.



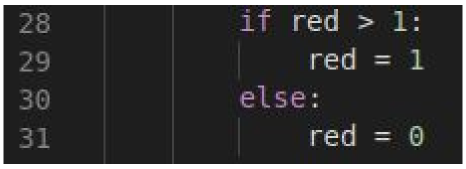
- Changing the colour is a little complex. First we want to add a little bit of red.

```
    red += delta_red
```

- When our red is more than 1, we want to start taking red away. If red is less than 0 we want to start adding red again. This will give a nice colour change effect.

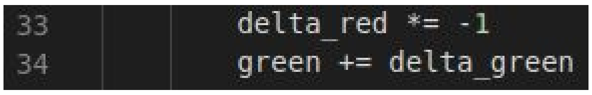
```
    if red > 1 or red < 0:
```

We need to set red back to the upper and lower limits first.



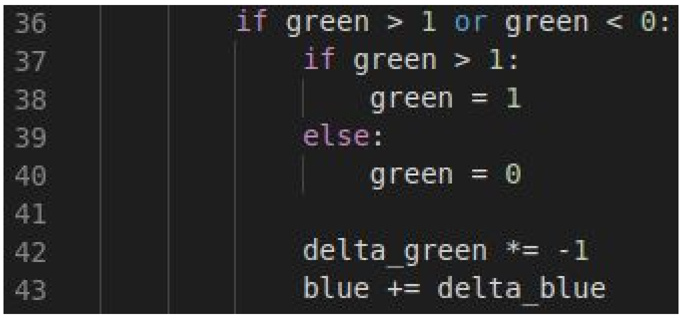
```
        if red > 1:
            red = 1
        else:
            red = 0
```

Changing if we are adding or taking away red is easy. We just multiply delta_red by -1. We also add a little bit of green.



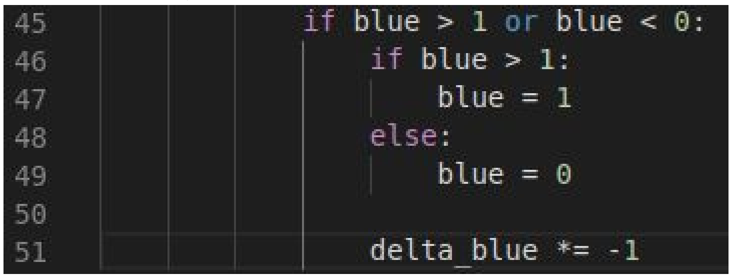
```
        delta_red *= -1
        green += delta_green
```

Now, if green as reached a limit, we need to do the same thing. But we add a little bit of blue at the end.



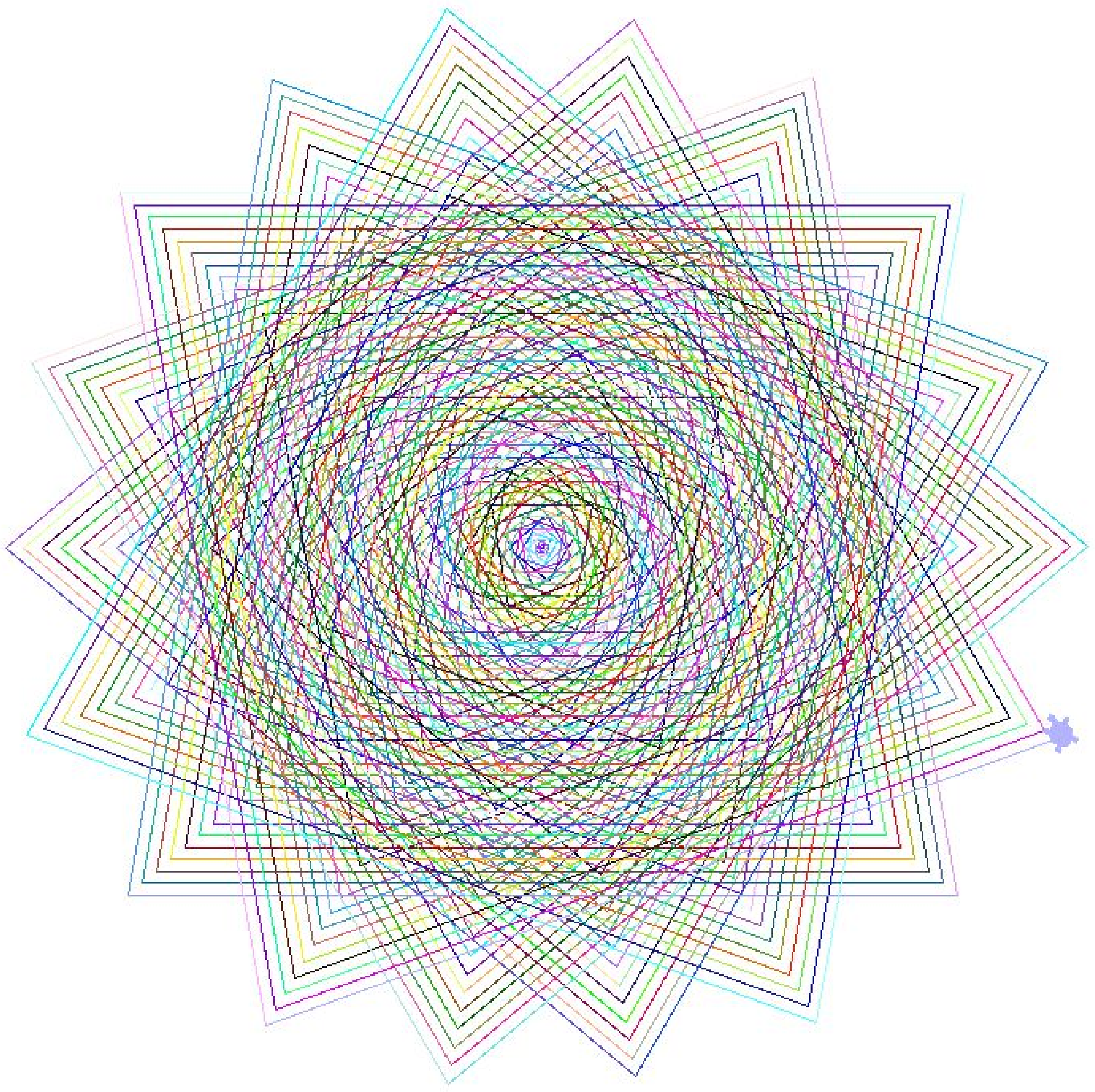
```
        if green > 1 or green < 0:
            if green > 1:
                green = 1
            else:
                green = 0

            delta_green *= -1
            blue += delta_blue
```

Finally we need to handle blue when it reaches it’s limit.



```
            if blue > 1 or blue < 0:
                if blue > 1:
                    blue = 1
                else:
                    blue = 0

                delta_blue *= -1
```

Test your code!
A few times.



With thanks to Jane from DigiLocal @ Bedminster for her spiral pattern.

Challenge time!

Can you change your code to make different patterns?

Can you change it so colour 'shift' looks different?