16 The Python turtle

16.1 About

The classic programming language Logo was widely used in education and was frequently used together with a physical robot known as a turtle. The turtle usually held a pen, and Logo programs could tell the turtle to move around or draw shapes. Python includes a turtle library that can be used to draw shapes on screen.

16.2 Create a script

Create a new blank project called **turtleProject**.

At the top of the script import the turtle library.

```
from turtle import *
```

Add the following line, which resets the turtle's position.

```
reset()
```

16.3 Draw a shape

The turtle can be moved forwards and backwards a number of pixels, and can turn left or right a number of degrees.
Draw a square using the following code:

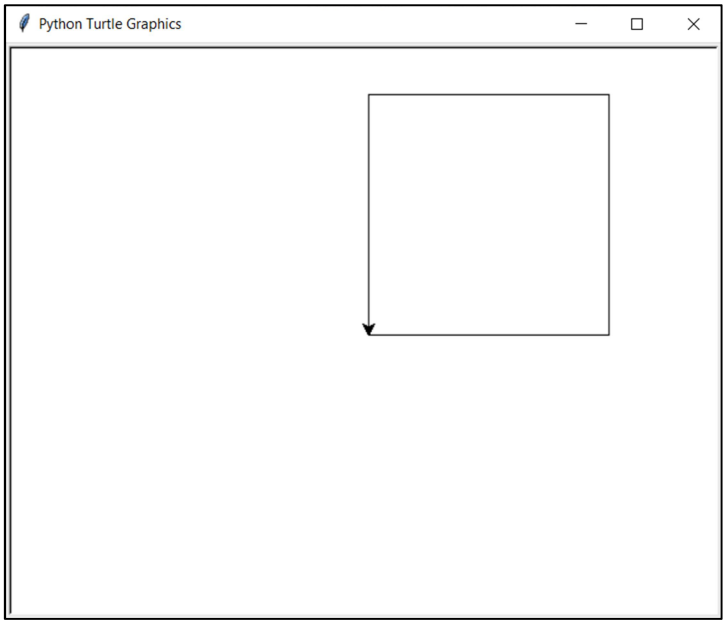
```

```

```
forward(200)
right(90)
forward(200)
right(90)
forward(200)
right(90)
forward(200)
```

The turtle window closes automatically at the end of the program so add a statement that waits for the user to click in the window before ending the program:

```
exitonclick()
```

Run the program. A turtle graphics window appears and draws the square:



16.4 Rewrite the program

Since the code that draws the square is the same lines four times around, rewrite it as a loop:

```
for i in range(1,5):
    forward(200)
    right(90)
```

Run again and make sure you get the same shape.

16.5 Rewrite the program again

Now let's change the program so it draws several squares. Since the code that draws a square will be the same each time, turn the code into a function. We'll also add the **pendown()** and **penup()** functions to put the pen down at the beginning of the drawing and pick the pen up again at the end.

```
def square():
    pendown()
    for i in range(1,5):
        forward(200)
        right(90)
    penup()
```

Call the function a couple of times.

```
square()
square()
```

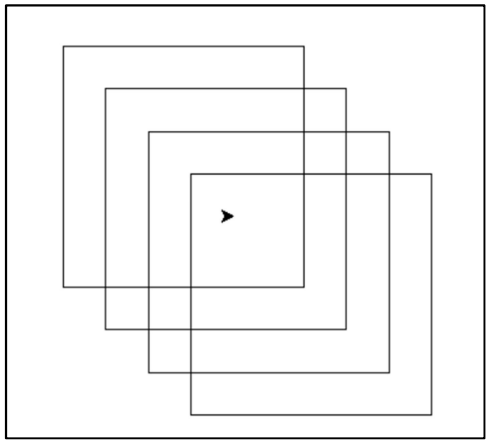
Run the program. The turtle will draw the square, but then just draw it again in the same place.

16.6 Draw multiple squares

Rewrite the part of the code that calls the function so that it calls the function in a loop, and moves the cursor between each iteration.

```
for i in range(1, 5):
    square()
    right(45)
    forward(50)
    left(45)
```

Run again. Observe the turtle as it moves. The output should be like this- resize the window if the turtle moves off the side of the window.



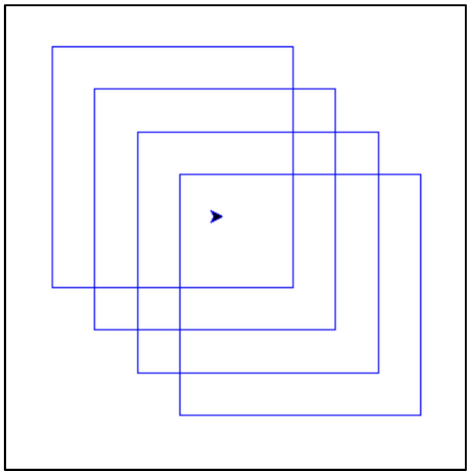
16.7 Change colours

The turtle can draw in any colour. The colours can be specified in a number of different ways. Here we will use RGB values, which are in the range 0 to 255. They are specified in the **pencolor()** function as a tuple. Note that by default the turtle colours are expressed as a decimal in the range 0 to 1, so we also use the **colormode()** function to say that we are using RGB values.

Add these statements before the main loop:

```
colormode(255)
pentuple = (0, 0, 255)
pencolor(pentuple)
```

Run again. This time the squares are blue:



Try some different values for the colours- for example **(255, 0, 0)** or **(0, 255, 0)** or **(128, 128, 0)**.

16.8 Checkpoint

At this point your script should look like this:

```
from turtle import *

def square():
    pendown()
    for i in range(1,5):
        forward(200)
        right(90)
    penup()

reset()
colormode(255)
pentuple = (0, 0, 255)
pencolor(pentuple)
```

```
for i in range(1, 5):
    square()
    right(45)
    forward(50)
    left(45)
```

16.9 Explore more

16.9.1 Filling

The turtle can fill in the shapes it draws. To do this, set a fill colour with **fillcolor()** (using an RGB tuple- the same as **pencolor()**) then call **begin_fill()** just before drawing a shape to be filled, and **end_fill()** afterwards- this is when the shape is filled in. Note that the shape must be completely enclosed (like a square).

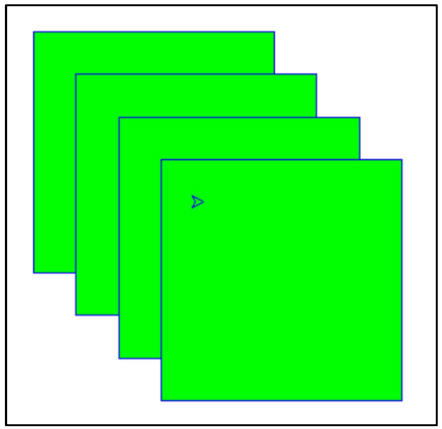
Example of **fillcolor()**:

```
fillcolor(0, 255, 0)
```

begin_fill() and **end_fill()** can be used to fill in the squares in the function, like this:

```
def square():
    pendown()
    begin_fill()
    for i in range(1,5):
        forward(200)
        right(90)
    end_fill()
    penup()
```

Change the program so that it produces output like this:



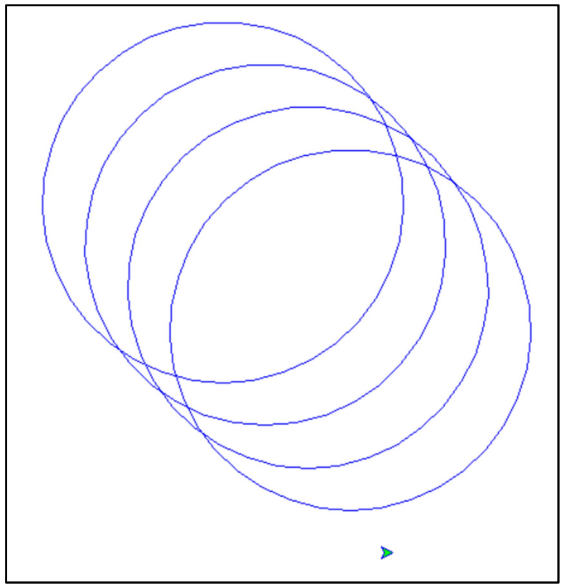
16.9.2 Circles

The turtle can also be used to draw circles.

Add some code like this:

```
for i in range(1, 5):
    pendown()
    circle(100)
    penup()
    right(45)
    forward(50)
    left(45)
```

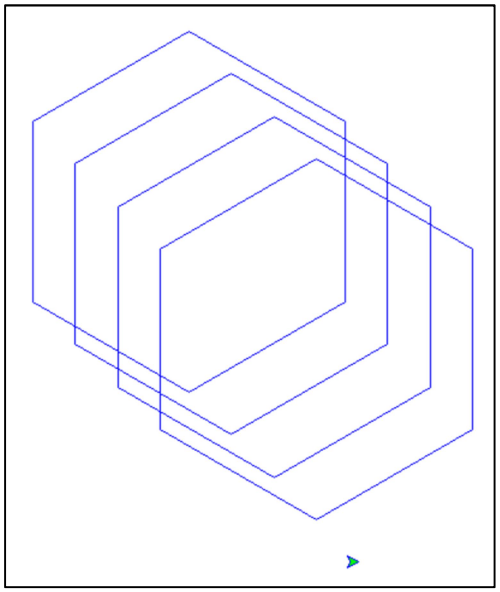
The output should be similar to this:



Note that the circles drawn by the turtle are actually polygons drawn in tiny steps. By changing the number of steps, the **circle** command can draw regular polygons. For example, to draw hexagons, change the **circle** command as follows:

```
circle(radius = 150, steps = 6)
```

This will produce output like this:



Try drawing shapes with increasing numbers of sides- a triangle, then a square, then a pentagon, then a hexagon, and so on.

16.9.3 A fast turtle

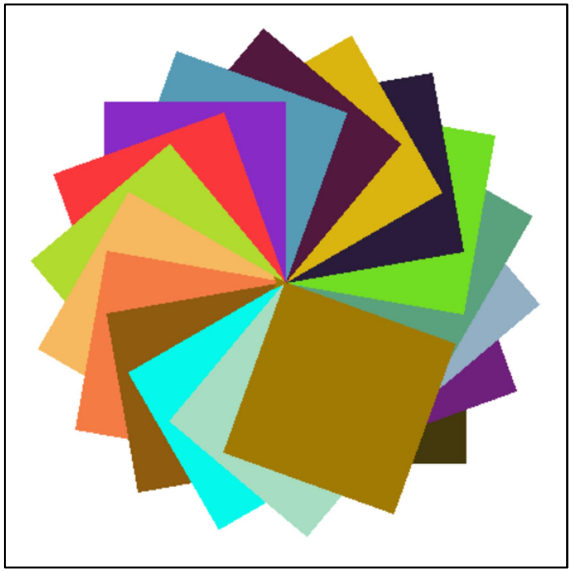
The turtle draws quite slowly, but that helps to see what it is doing.
The speed of the turtle can be changed. The speed ranges from 0 (fastest) to 15 (slowest). Add this command after the **reset** command to speed up the drawing:

```
speed(0)
```

17 Challenge : Turtle patterns

17.1 Turtle pictures

Write a script that uses the turtle to produce a geometric pattern like this:



The colour for each shape should be random. The fill colour and pen colour should be the same.
Make each shape offset by 20 degrees from the previous one- therefore since there are 360 degrees in a circle, the script will draw 18 shapes.

17.2 Modifications

Modify the script so that it leaves a hole in the middle of the shapes.