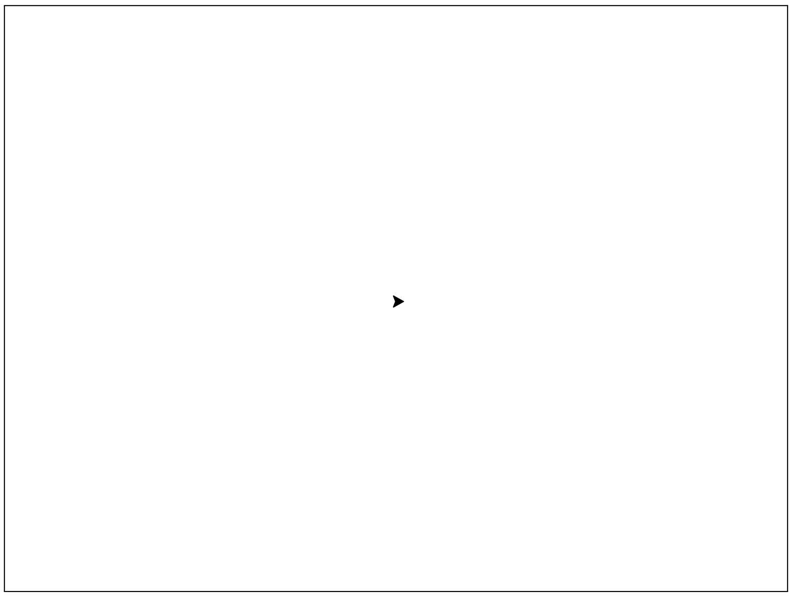
Introduction

Here is a simple turtle programme (`turtle_doing_nothing.py`):

```
1 from turtle import *
2
3 tom=Turtle()
4
5 tom.getscreen()._root.mainloop()
```

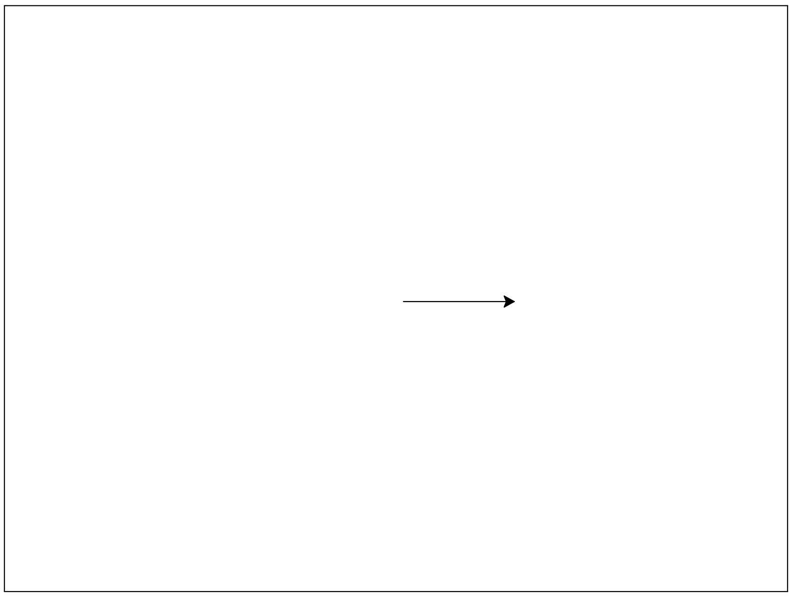
Line 1 and **line 5** aren't worth spending much time on at first, the first line imports the library of commands related to turtle, **line 5** prevents the computer from closing the graphics window when the programme has finished running; we won't include this line again, though it is needed. **Line 3** is important, it tells the computer to make an object, in this case a `Turtle` and call it `tom`, it knows what a `Turtle` is from the library it imported in line 1; in the instructions on what to do when making a `Turtle` the computer is told to open a graphics window and to draw the turtle, a little arrow shape.



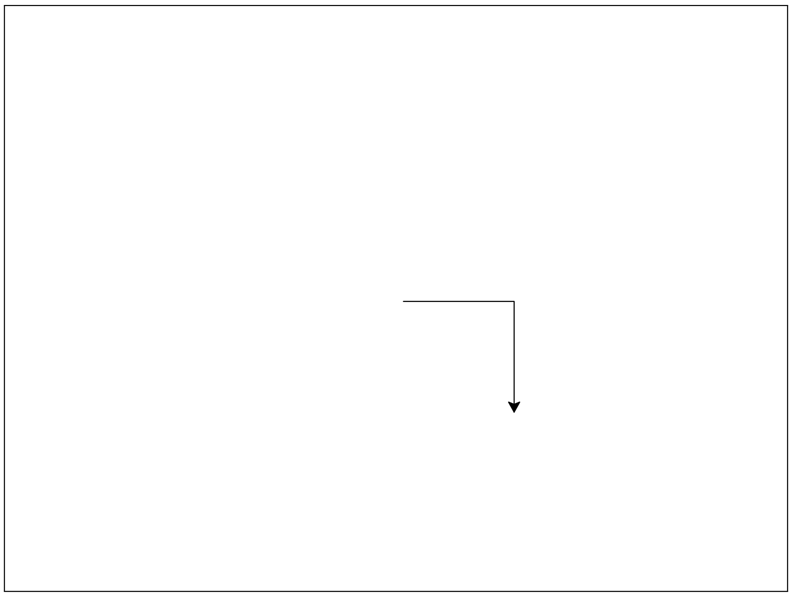
Here the turtle does something (`line.py`):

```
1 from turtle import *
2
3 tom=Turtle()
4
5 tom.forward(100)
```

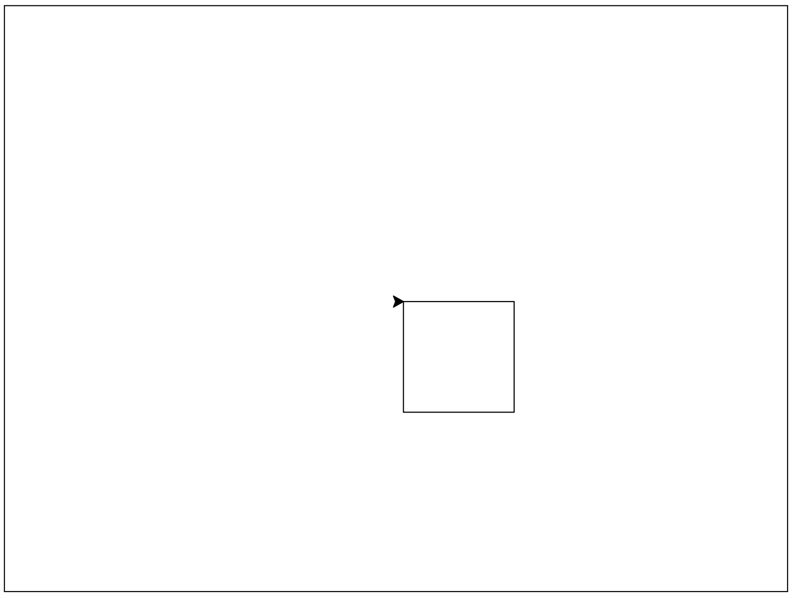
The extra line, **line 5**, tells the turtle to move forward by 100 units, this is an important piece of Python syntax, to tell an object to do something you use a dot followed by the command, here it tells the `Turtle` called `tom` to perform the command `forward`. Of course, the command has to make sense for whatever type of object it is dotted onto, but here it does, `forward` is one of the defined commands for a `Turtle` object.



`Turtle` objects have another command `right(90)` which turns the turtle by 90°. QUESTION: Can you write a programme to draw this:



QUESTION: How about a square?



Perhaps you programme to draw a square looked like this

```
from turtle import *

tom=Turtle()

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

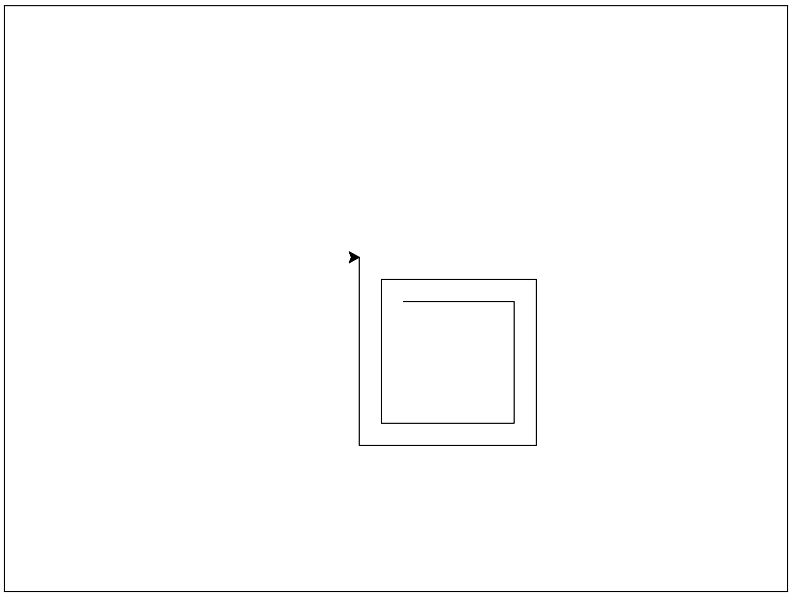
There are two problems with this, most obviously it is boring writing in the same two lines again and again; secondly, the programme is inflexible and hard to read, we'll deal with the inflexible bit later, but as for the hard to read, to know that it draws four lines and has four corners you need to count the lines; it would be better if the 'fourness' was more apparent, as it is in this programme (`square_loop`):

```
1 from turtle import *
2
3 tom=Turtle()
4
5 for i in range(0,4):
6     tom.forward(100)
7     tom.right(90)
```

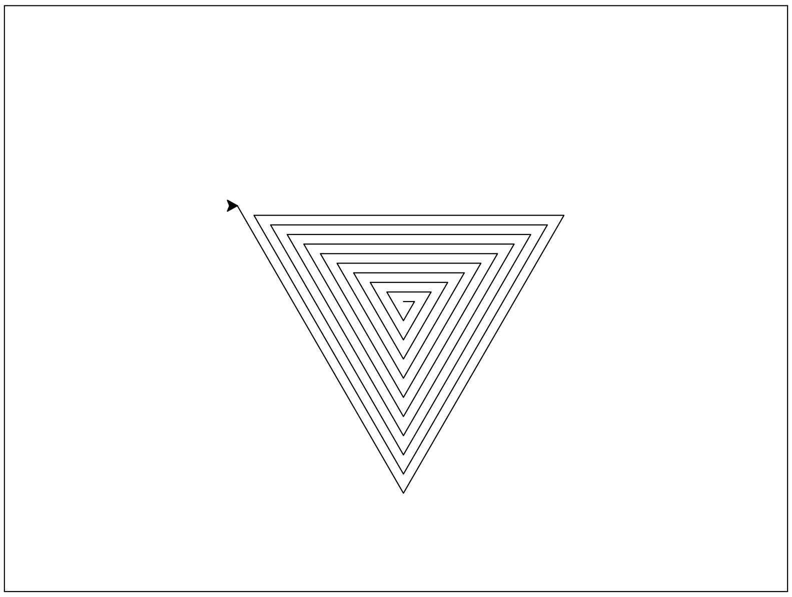
The business part of this program is **line 5**; `range(0,4)` is the list of numbers `[0,1,2,3]`, so starting at zero and ending one before four; the full command says that `i` takes each value in this list in turn and then does all the stuff belonging to the command. Here the command is a `for`, a command that says 'do everything that belongs to the for once for every value the variable, in this case i, is instructed to take'. In Python stuff belonging to a command is indented, so that means it does **line 6** and **line 7** once for each item in the list, that is four times. Of course, in this programme `i` isn't used for anything except counting but it could be (`spiral1.py`):

```
1  from turtle import *
2
3  tom=Turtle()
4
5  for i in range(0,8):
6      tom.forward(100+10*i)
7      tom.right(90)
```

giving



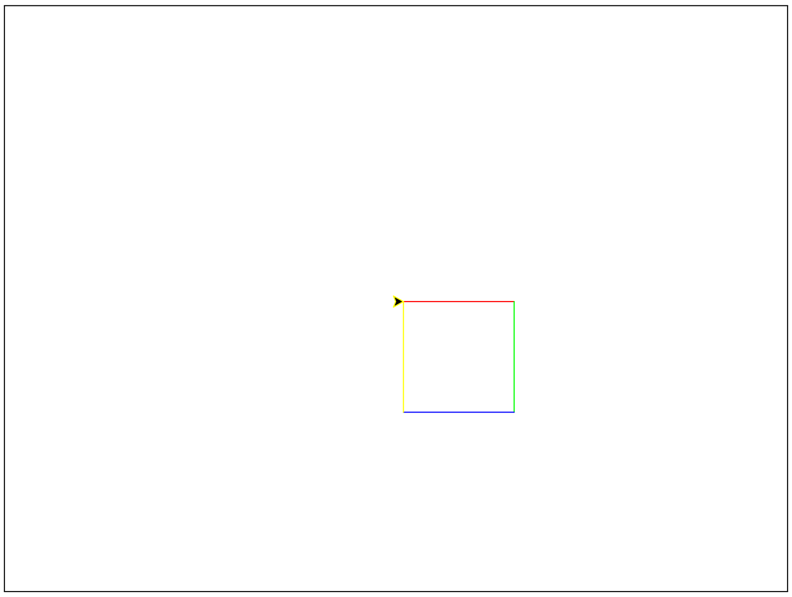
`i` is variable, it stores some data, in this case a number which changes each time the programme goes around the `for` loop. Once slightly confusing thing is 'scoping', which is where the variable is defined; the `i` is only defined inside the `for` loop, that is, it only exists while the programme is executing **line 5** to **line 7**, but that's fine, that's where we use it. QUESTION: Can you draw a triangular spiral like this:



The list in the `for` loop doesn't have to be a `range`, in this example (`square_color.py`)

```
1  from turtle import *
2
3  tom=Turtle()
4
5  colors=['red','green','blue','yellow']
6
7  for color in colors:
8      tom.pencolor(color)
9      tom.forward(100)
10     tom.right(90)
```

`colors` defined in **line 5** is a list of colours and the variable in the `for` command takes each of these values in turn. `pencolor` is another `Turtle` method, it changes the pen colour so we get

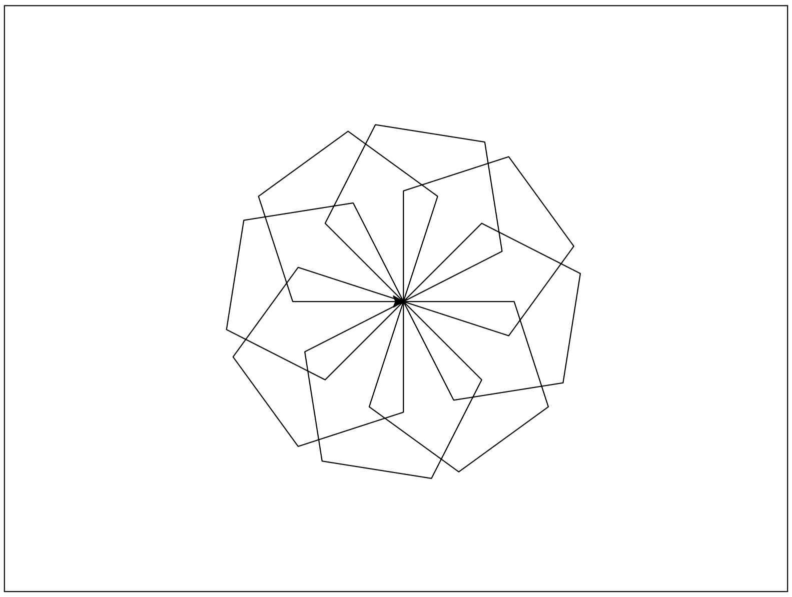


In the colour example we use a variable `colors` to store a list; here is an example where we use a variable to store an integer `n` which gives a number of sides.

```
1 from turtle import *
2
3 tom=Turtle()
4
5 n=8
6
7 for i in range(0,n):
8     tom.forward(100)
9     tom.right(360.0/n)
```

This is useful because we need the number of sides twice, once in **line 7** where it gives the number of sides, and the second time in **line 9** where it is used to calculate the turning angle. Obviously it would be possible to write the number in those two places, but that would be a very bad programming style because it would mean changing it in those two places if you wanted to change the number of sides to the polygon. It would be easy to get this wrong, maybe not for only two places, but in a more complex programme there may be many places a value needs to be changed, leading to errors. Furthermore, keeping `n` seperate makes the programme easier to understand.

Now imagine you wanted to draw this:



This is made of eight pentagons with an eighth of a full turn between each one; this can be made using the programme:

```
 1 from turtle import *
 2
 3 tom=Turtle()
 4
 5 repeats=8
 6 polygon_sides=5
 7
 8 for i in range(0,repeats):
 9     for j in range(0,polygon_sides):
10         tom.forward(100)
11         tom.right(360.0/polygon_sides)
12     tom.right(360/repeats)
```

So this programme has two `for` loops, the `j` loop from **line 8** to **line 10** draws the pentagons, the `i` loop repeats that eight times and rotates between pentagon. Notice the two loops need different variables, `i` and `j`, and you can tell what belongs to which loop by the indent, **line 9** and **line 10** belong to the `j` loop so these command are run $40 = 8 \times 5$ times whereas **line 9** and **line 11** are only run eight times.

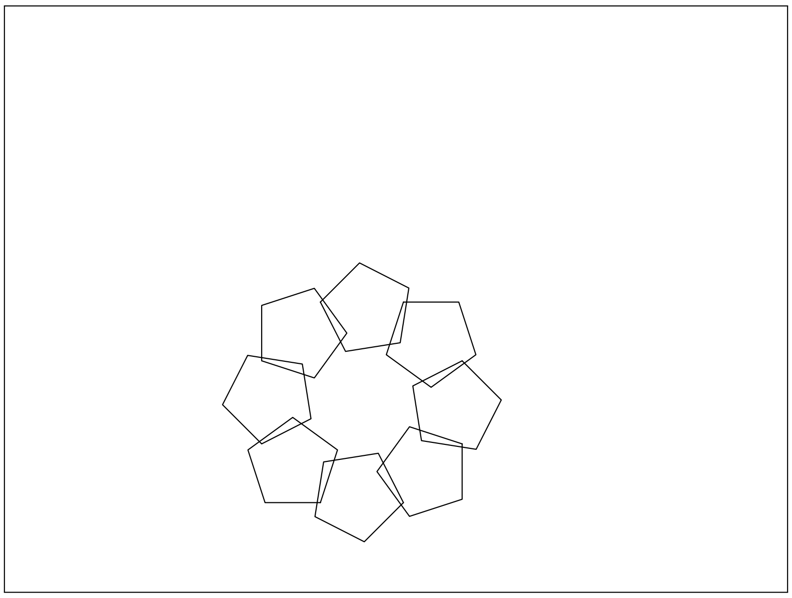
This programme works, but it fails our readibility and adaptability test; to work out what it does you need to figure your way through the double loop and if you wanted to draw another pentagon later you would have to cut and paste the lines you already had. A much better programme would use a function (`many_polygons.py`):

```
1 from turtle import *
2
3 def polygon(n):
4
5     for i in range(0,n):
6         tom.forward(100)
```

```
 7            tom.right(360.0/n)
 8
 9 tom=Turtle()
10
11 repeats=8
12 polygon_sides=5
13
14 for i in range(0,repeats):
15      polygon(5)
16      tom.right(360.0/repeats)
```

Now in **line 3** to **line 7** we have defined a `function`, the command `def` announces in **line 3** that a function definition is on the way, all the indented lines are the definition. The function is named `polygon`; the brackets after the function name are to give the arguements, values that are sent to the function, in this case the function has one arguement which will give the number of sides of the polygon. Now we have a new command, `polygon(n)`, whenever the command is given, the programme goes up to the function and runs the code belonging to the function, this happens in **line 17**.

QUESTION: can you guess how this was drawn:



The `Turtle` commands `penup()` and `pendown()` lift and drop the turtle's pen, when the pen is up the turtle moves without leaving a line; `hideturtle()` hides the turtle (`many_polygons_extra.py`)
A more complicated example (`vanishing_square.py`)