Level 1

Teaching Turtles

{code club}

Keep track of your progress by ticking off the boxes below:

Step 1: Back to Square One

✓ Activity Checklist

Open IDLE, the editor and open a new file by going to File > New Window, and let's get started.

Remember you should have two windows open. One is 'Python Shell', and the other is to write your code in.

Like before, the first line will always be `from turtle import *` to tell Python we want to draw!

```
from turtle import *

for n in range(4):
    forward(100)
    right(90)
```

Save it as a new file, and Run it from the Menu by going to Run > Run Module.

Remember that for `n in range(4)` repeats the code, and that the code needs to be grouped using spaces, or indented to be part of the the for loop. Use the 'Tab' key (it's above caps lock!) to move the code around.

Step 2: A variable square

Like we did last time, let's use variables to make our program clearer, and easier to change:

✓ Activity Checklist

1. Edit the file to look like the following: ☐

```
from turtle import *
sides = 4
length = 100
angle = 360/sides

for n in range(sides):
    forward(length)
    right(angle)
```

2. Run it using Run > Run Module in the Menu. Do you get the same square as before? Make sure it works before you move on. ☐

This is a long program, but now we can change it to draw any shape we wanted to, but we'd need to copy it over and over again. Like before, we can write some code to stop having repeating ourself. This time we will define a new command.

Step 3: A new command appears

Activity Checklist

1. We will edit the code and add def `poly():`, indent the code (you can select it and press Tab), and call the new command. ☐

```
from turtle import *

def poly():
    sides = 4
    length = 100
    angle = 360/sides

    for n in range(sides):
        forward(length)
        right(angle)

pencolor('red')
poly()
right(180)
poly()
```

2. Run it, it should draw two red squares. ☐

We've saved a little bit of time by defining a new command in python, and now we can draw a square twice, without having to write the whole thing twice. These new commands are called functions in python, and they're a great way to avoid writing so much.

Step 4: Why stop at squares?

We're not finished yet, how about we change the function so it can draw any shape! Like with forward and right, we can pass values into the function, rather than just edit the code each time.

Keep track of your progress by ticking off the boxes below:

Activity Checklist

1. Edit the code from the last step to look like this: ☐

```
from turtle import *

def poly(sides, length):
    angle = 360/sides

    for n in range(sides):
        forward(length)
        right(angle)

pencolor('red')
poly(4, 100)
right(180)
pencolor('blue')
poly(3, 150)
```

2. Run it, and see what happens. ☐

Let's take it slowly here because this is quite cool. Instead of setting the variables in the function, we say that the function takes some values, with some names, and then we put the values in where we call them.

We've moved the settings outside of the function, and moved them into the code that uses it. Now with one function we can draw *any* shape, of *any* colour. Pretty mindblowing! So we can teach the computer new instructions, and use them.

Being able to define new commands, that can behave differently based on the values given, is one of the most powerful tools in programming.

Step 5: Turtle Dash

Activity Checklist

Although the turtle is a little robot that can draw, it can also move without drawing Remember that we can use `penup()` and `pendown()` to turn drawing on and off.

1. Open a new Python file, and put the following code in: ☐

```
from turtle import *

length = 200
for num in range(8):
    forward(length/16)
```

```
        penup()
        forward(length/16)
        pendown()
```

2. It draws a dashed line across the screen. Run it and check! ☐

Step 6: Dashing Shapes

We can put the shape and dashed line program together, by replacing the `forward` with the code we have for dashes. We repeat the outer layer to draw each side of the shape, and then repeat on the inside to draw all the dashes.

Activity Checklist

1. Edit your code to look like the following: ☐

```
from turtle import *
speed(11)
shape("turtle")

def dashpoly(sides, length):
    angle = 360/sides

    for n in range(sides):
        for num in range(8):
            forward(length/16)
            penup()
            forward(length/16)
            pendown()
        right(angle)

pencolor('red')
dashpoly(4, 100)
right(180)
pencolor('blue')
dashpoly(3, 150)
```

2. Run your code and see what it does. ☐

We have two `for` loops inside each other, an outer one and the inner one. The outer loop `for n in range(sides)` draws each side of the shape, and each time runs the inner `for loop for num in range(8)` which draws the dashes.

The outer loop uses the variable `n` to keep track of how many times it has repeated, and the inner loop uses the variable `num` to keep track. You have to use different variable named loops, or python will get confused.

Step 7: Building blocks for shapes

Activity Checklist

1. Let's use functions again to clean up the code. Edit your code from step 6, and let's split the code apart. ☐

```
from turtle import *

speed(11)
shape("turtle")

def dashforward(length):
    for num in range(8):
        forward(length/16)
        penup()
        forward(length/16)
        pendown()

def dashpoly(sides, length):
    angle = 360/sides

    for n in range(sides):
        dashforward(length)
        right(angle)

pencolor('red')
dashpoly(4, 100)
right(180)
pencolor('blue')
dashpoly(3, 150)
```

2. Run your code and it should do the same thing. ☐

Protip

The trick is that instead of building programs by copy and pasting, we can define new commands and re-use them, making the code a little shorter and a little easier to understand.

Step 8: A little bit random

Just before we stop, how about doing something a little random? We can ask the computer to pick a number for us, or pick a colour for us, a bit like rolling dice. Scratch does this too with the `pick` operator

Activity Checklist

1. In a new file, type the following in ☐

```
from turtle import *

from random import randrange, choice
colors = ['red', 'blue', 'green']

def poly(sides, length):
    angle = 360/sides

    for n in range(sides):
        forward(length)
        right(angle)

for count in range(10):
    pencolor(choice(colors))
    right(randrange(0,360))
    poly(randrange(3,9), randrange(10,30))
```

2. Save and Run your Code ☐

It should draw ten shapes, in different colours, of different sizes. The line `from random import randrange, random` introdices two new functions to use, `randrange()` and `choice()`.

`randrange()` lets you pick a number between a low and a high number, so `randrange(1, 10)` will pick a number between 1 and 9 (Python starts at 1, and stops just before 10).

`choice()` lets us pick an item from a list. A list is a collection of values, like `[1, 2, 3]`, and above we have the list colors, which has the values `'red'`, `'blue'`, and `'green'`.

By using `choice()` and `randrange()` we can ask the computer to pick the colour, size and shape of what we're drawing, and it will be different every time you run the program.

Things to try

Why not try adding more colours, or changing the numbers? What happens?