Level 1

Turtle Power

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

Introduction:

This term we're going to be learning a computer programming language called Python. The person who created Python named it after his favourite TV show: Monty Python's Flying Circus. Python is used by loads of programmers for many different things. Python is a powerful and clever language, used at YouTube, NASA, CERN and others. If your club has a Raspberry Pi, you can use Python to program it. Many people love Python because it is easy to read (at least compared to some other programming languages). Being able to read code is a very important skill, just as important as writing it.

Step 0: Open up the Python Editor

If python is on your computer already, it's time to get started.

- On Windows, find IDLE in the start menu.
- On Mac OS X, open up Terminal.app and type idle and press enter.
- On Linux, open up a Terminal, and type idle and press enter.

If you don't have Python yet, don't panic! You can download the latest version of Python from http://www.python.org/. The exact version doesn't really matter but we're using Python 3, so it should start with 3 (not 2).

When IDLE starts you will see an Output window called `Python Shell`. We need to open a new window to write code in. Go to `File -> New Window` so you're ready for Step 1. Make sure you have both windows visible.

Step 1: Hello, World!

Activity Checklist

1. Open IDLE, the editor that comes with Python. All our code will be written in this editor. When you open it, you will see an Output window, where errors and results will appear. ☐
2. If you haven't yet, choose File -> New Window. An empty window will ☐

appear, with 'Untitled' in the title bar.

You should have two windows open now, one for writing your program, another for showing output. Make sure you write in the right one!

3. Enter the following code into the new window: ☐

```
print("Hello, World!")
```

4. Double check you're not in the Python Shell window, and Save your code. ☐

You can do this by choosing File -> Save. When prompted for a filename, enter hello.py, and save the file to your desktop. Then choose Run -> Run Module.

Congratulations on your first Python program :D (PS! You can tell it to print anything you like, why not change it to say hello to you? Change it to say your name instead)

ProTip:

On **Windows** and **Ubuntu**, use Ctrl-N to create a new shell window, use ctrl-S to save your file after you've finished editing, and press F5 to run your program. On some computers you may need to press a Fn key too.

On **Mac OS X**, cmd-N to create a new shell window, Command-S to save your file, and hold down the function (fn) key and press F5 to run your program.

Step 2: Hello, Turtle!

Next, we're going to have some fun with turtles. A turtle is a tiny robot that draws on your screen, we can tell it to move around using Python commands.

Activity Checklist

1. Open a new code window (From the File menu) and write the following: ☐

```
from turtle import *
forward(100)
```

2. Save your program as myturtle.py and choose Run -> Run Module. See how the turtle moved forward 100 pixels? The turtle has a pen attached, and draws lines as it moves around the screen. ☐

ProTip: *Python program files should always be given a name ending in '.py'.*

Turtle Power

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

3. Let's make the turtle move around the canvas! Try using `backward(distance)` as well as turning the turtle by using `right(angle)` and `left(angle)`. Eg `backward(20)` tells the turtle to move backwards 20 pixels, and `right(90)` tells the turtle to turn right 90 degrees. You can give more than one instruction at a time, they will be executed in order.

```python
from turtle import *

speed(11)
shape("turtle")

forward(100)
right(120)
forward(100)
left(90)
backward(100)
left(90)
forward(50)
```

Angles and Degrees

Play around a bit writing your own shapes, using forward, backward, left, right. Remember, forward and backward move in pixels, but left and right turn in angles. Let's look at a turtle turning right.

```
        North

          0

          |

West      |      East

 270 -----+----- 90

          |

          |

         180

        South
```

When the turtle is facing North, turning right 90 degrees, makes it face East, turning 180 degrees makes it face south, and turning 270 degrees makes it face West. If you turn 360 degrees, you end up where you started.

What about turning left?

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

```
        North
          0
          |
West      |      East
  90 -----+----- 270
          |
          |
         180
        South
```

When the turtle is facing North, turning left 90 degrees, makes it face West, turning 180 degrees makes it face south, and turning 270 degrees makes it face East. If you turn 360 degrees, you end up where you started. One full turn is always 360 degrees.

What does the code at the beginning of our program do?

`from turtle import *` tells Python we want to use the turtle library, a collection of code we can use to draw on the screen. Using a library means we can save time.

`speed()` sets the speed of the turtle, it takes a value between 1 and 11. 11 is the fastest, 1 is the slowest.

`shape()` We are using the "turtle" shape, but it can also take the values "arrow", "circle", "square", "triangle" or "classic".

We will be putting these three instructions at the top of all our programs in this lesson. If you like you can make your turtle a different shape, or go as fast or slow as you want.

Step 3: Drawing Shapes!

Lets make a square by telling the turtle how to move around.

Activity Checklist

1. Open up a new file in IDLE, and write the following code in: ☐

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

```
from turtle import *

speed(11)
shape("turtle")

forward(100)
right(90)
forward(100)
right(90)
forward(100)
right(90)
forward(100)
right(90)
```

Save your program and choose run -> new module. Do you see a square?. The turtle turns 90 degrees four times, and ends up facing back in the same direction as it started. Turning 90, 90, 90 and 90 again, turns the turtle 360 degrees in total.

What about a triangle? A triangle has three corners, so we need to turn three times. If we want to end up back in the same direction, we will need to turn a total of 360, like the square. So we will turn 120 degrees, then 120 degrees, and once more.

2. Edit your code to look like the following, to draw a triangle instead: ☐

```
from turtle import *

speed(11)
shape("turtle")

forward(100)
right(120)
forward(100)
right(120)
forward(100)
right(120)
```

3. Run your code. Do you have a triangle? ☐

Pick a Colour

What's your favourite colour? You can change the colour of the lines using the function `pencolor` (Python uses American spellings), and you can also change the pen size using `pensize`.

1. Edit the code from before to look like the next example, by adding in these new commands: ☐

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

```
from turtle import *

speed(11)
shape("turtle")

pensize(10)
pencolor("red")
forward(100)
right(120)

pencolor("blue")
forward(100)
right(120)

pencolor("green")
forward(100)
right(120)
```

2. Run your code, what does it draw on screen? ☐

This code draws a thick triangle in three different colors.

3. Try changing the colors in your code, run it, and see what happens. ☐

The turtle knows hundreds of different colours, not just red, blue and green, try using your favourite colour!

You can also specify colours in hex like we did in CSS. Instead of using `pencolor("red")` you could use hex `pencolor("#FF0000")`.

What colour is **#FF4F00**?

Step 4: Repeating Yourself (with a for loop)

That last program was the same commands over and over again. Instead of writing them down, let's ask the computer to repeat them for us. You should have encountered *iteration* in Scratch before using the `Forever` and `Repeat`/`Repeat until` blocks. In Python `for` loops are used when you have a piece of code which you want to repeat n number of times. In this case we want to repeat the code (that is indented) 4 times (because a square has 4 sides).

Activity Checklist

1. Open up a new file and type the following in: ☐

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

```
from turtle import *

speed(11)
shape("turtle")

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

2. Save your program and choose: `Run -> Run module`. ☐

Notice the program is indented, or pushed to the left under the `for` loop. Python uses spaces to know which commands to repeat. You can use the Tab key in IDLE to add indents, and use Shift-Tab to remove some.

3. Let's see what happens when we only indent `forward`, and edit your program to look like this one: ☐

```
from turtle import *

speed(11)
shape("turtle")

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

4. Notice how `forward` is indented and `right` isn't? What do you think this program does? Try running it to find out? ☐

Did you get a straight line? Python will repeat `forward` four times, and then turn right. Python uses spaces to group commands together, like scratch uses blocks. Python will complain if the code isn't properly lined up.

5. Let's change it back to the earlier program, and get it to run a square, but instead of using numbers in the code, we can assign names. This makes it easier to see what the program is doing, and also helps us to stop repeating ourselves. ☐

6. Edit the file to look like this: ☐

```
from turtle import *

speed(11)
shape("turtle")

sides = 4
length = 100
angle = 90

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

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

7. Save your program and choose: Run -> Run module. ☐

Challenge: Drawing the other shapes

Can you draw the following shapes by changing the values?

- ☐ a triangle? (three sides)
- ☐ a pentagon? (five sides)
- ☐ a hexagon? (six sides)
- ☐ an octagon? (eight sides)

Remember, a Triangle has three sides, and turns 120 degrees. Turning 120 degrees for each corner means we turn 360 degrees in total. For a Square, we turn 90 degrees for four corners, which also adds up to 360 degrees.

If you are turning six times, how much should you turn so it adds up to 360? Try out numbers and see what shapes you get.

Step 5: Turn, Turn, Turn

Instead of us working out the angle, why don't we let the computer do it for us. Python can let you do some operations on numbers, like addition, subtraction, multiplication and division.

We could write `sides = 4 + 1` instead of 5, or `sides = 4 - 1` instead of 3. For multiplication, Python uses `*`, and for division, we write `/`.

If we need to turn 360 degrees in total, we can work out the angle we'll need. For a square, `360/4` is 90, for a Triangle, `360/3` is 120.

Activity Checklist

1. Change your program to calculate the angle as following. ☐

```
from turtle import *

speed(11)
shape("turtle")

sides = 4
length = 20
```

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

```
angle = 360/sides
for count in range(sides):
    forward(length)
    right(angle)
```

2. Now change the number of sides, does Python get it right? Try drawing a hexagon (6 sides), or any number of sides you want! ☐

Step 6: Solid Shape

Activity Checklist

We can ask the turtle to fill in shapes with a colour, by using `begin_fill()` and `end_fill()`.

1. Change your code to add these commands in: ☐

```
from turtle import *

speed(11)
shape("turtle")

sides = 4
length = 20

fillcolor('red')
pencolor('red')
begin_fill()

angle = 360/sides
for count in range(sides):
    forward(length)
    right(angle)
end_fill()
```

Like with `pencolor`, `fillcolor` sets the color the turtle uses to fill in the shapes you draw. This draws a red square with a red outline.

You use `begin_fill()` to tell the turtle you're drawing a shape to colour-in, and `end_fill()` to say you've finished it.

2. Try changing the colours, sides and lengths to see what shapes you can draw! ☐

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

Step 7: Pen Goes Up, Pen Goes Down

Activity Checklist

If you want to move the turtle without leaving a trail behind it, you can use `penup()` and `pendown()` to turn drawing on and off.

1. In a new file, try the following: ☐

```
from turtle import *

speed(11)
shape("turtle")

pencolor('red')

for count in range(20):
    penup()
    forward(10)
    pendown()
    forward(20)
```

2. This should draw a dashed line across your screen. Run it and see! ☐

Home, home on the screen.

One last little trick: `home()` returns the turtle to the starting position. `clear()` wipes the screen clean, and `reset()` moves the turtle and clears the screen.

Step 8: Go Wild!

You can go `forward()`, `backward()`, `left()`, `right()`, you can loop using `for count in range(4)`, change colours, change speed and even fill in shapes!

Can you draw a house, a bird? a snake? a cat? a dog? a lion? Combine the shapes together and see what you can draw. Can you draw a robot?