MicroBit Neopixel Ring

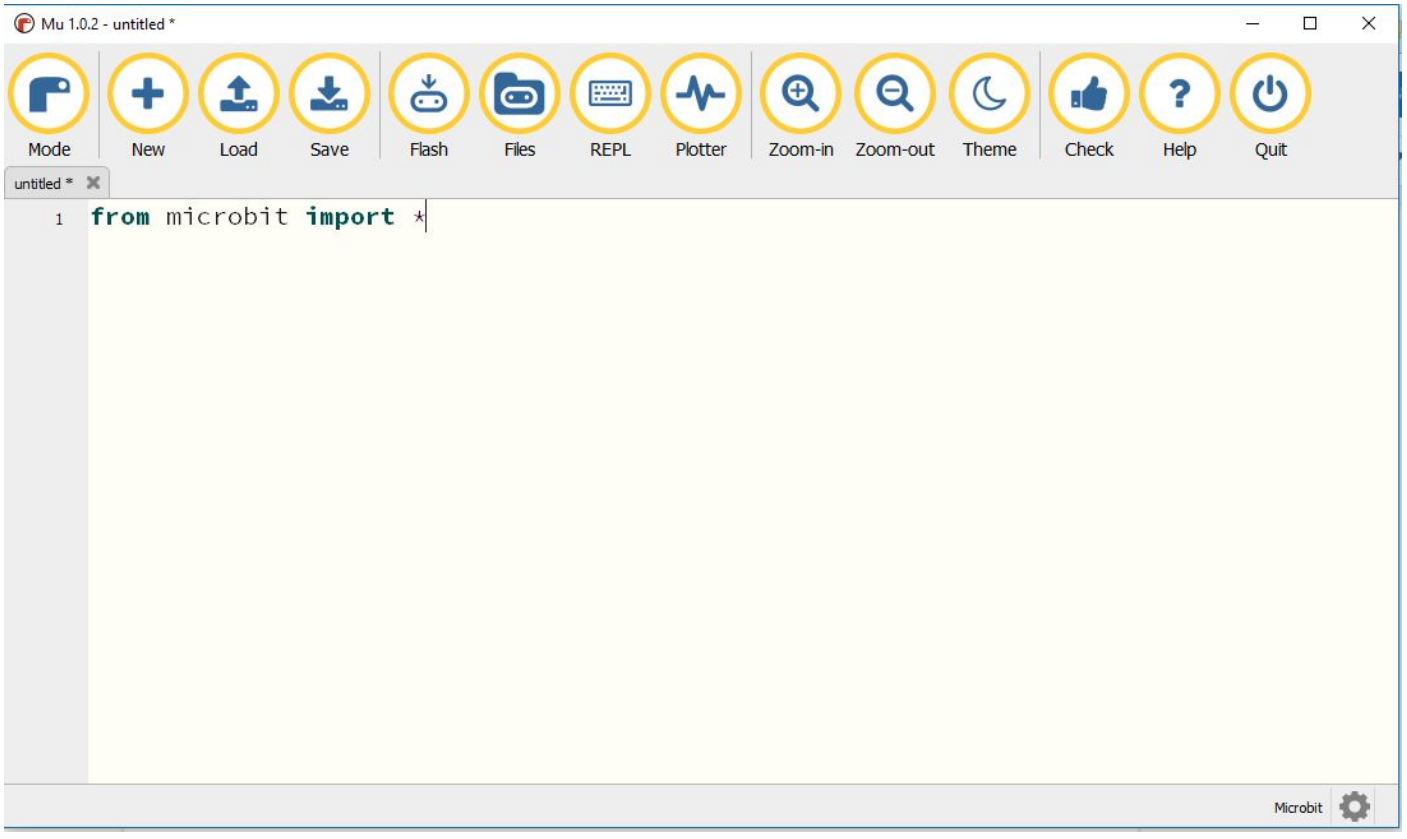
A NeoPixel is a name given to an LED that has within it three separate red, green and blue LED and also a small chip. There are also versions with a white LED too. To control them a signal is sent to the chip which contains the values the individual LED must be illuminated too. The signal is then passed onto the next NeoPixel and it reads it's values and passes it on. Each individual NeoPixel can be set to different colours.

Connecting the Ring

There are three wires to connect to the Microbit. The black clips to the GND edge connector. The red to the 3v and the green to Pin 0. (It will of course also work with pin 2 or 3). A small number of Neopixels can be powered directly from the MicroBit but 16 is about as many as we'd recommend. Any more and a separate power supply would be needed with the ground from that connected to the MicroBit too. Then you could use up to around 300 NeoPixels.

Editor

You will be using the Mu editor and a version of Python, MicroPython, to code the MicroBit. Open the Mu editor from the Pi-Top from the Programming part of the Menu. Then connect your Microbit.



Basic Programme

Start with getting a single LED to light. **Actual code to type is in blue. Code you should already have is black. Comments and explanations in green.** The lines with the hashtag # are comments and explain to you what the code is doing. You do not need to type these in. Though commenting your code is good practice.

```
# import all the import bits about a microbit & neopixels into the program

from microbit import *
import neopixel

# Setup the pin and how many LED are used as variable “ring”
# "ring" is just a word to name the variable

ring = neopixel.NeoPixel(pin0, 16)

# The colours are Red,Green,Blue
# Here 75 red with no green or blue
# Zero is no colour, 255 maximum colour brightness

ring[0] = (75, 0, 0)

# ring[0] is the first LED

ring.show()
```

To load this code onto the MicroBit by pressing Flash. The LED on the back of the MicroBit should flash for a bit and then an LED on the ring should light up. Errors in the code will be written onto the MicroBit and highlight the lines of your code that might be incorrect.

Did the Code Not Work?

Check neopixel is spelt correctly. It should be “neopixel” and “NeoPixel”. Different brackets are used ( ) and [ ].

Hack the Code

Change the value in the line “ring[0]” from a zero to another number up to 15. Also change the colour by using different values between zero and 255 in the brackets “(75, 0, 0)”

Light Up All the LED

One LED is quite easy to light up but it is much nicer with them all lit up. Here is the code. You can use your own colour from the hack above if you like. Don't worry about getting it wrong, you're unlikely to break anything. If your code doesn't work then use this code and try again.

```
from microbit import *
import neopixel

ring = neopixel.NeoPixel(pin0, 16)

# set a variable called blue to a colour
blue = (0, 0, 75)

# next_led is another variable that changes from zero
# up to but not including 16 (0 ~ 15)
# The code loops 16 times lighting up the led one after another
# The sleep is just a small pause of 100 milliseconds (1/10th of a second)

for next_led in range(0, 16):
    ring[next_led] = blue
    ring.show()
    sleep(100)
```

Did the Code Not Work?

Check neopixel is spelt correctly. It should be "neopixel" and "NeoPixel". Different brackets are used ( ) and [ ]. Did you miss the colon (:) at the end of the for next line? And the code after the colon must be indented as above. The Mu editor does this automatically if you remember to put in the colon.

Hack the Code

Change the colour by adding a new variable. Change the speed of the lighting by altering sleep. Switch the LED off. Clues for that are in the first code.

Changing the Colours

This time the program will change the colours one after another. Here a number of colours are used. Feel free to add you own, just add the names and values to the list following the example.

The list called colours is just that. This time there are two for lines. The first takes each colour name (which represents a series of numbers) from the colours list and calls it use_this_colour. Then like before the colour is set to the LED with the ring line. But because we are now using a variable (which of course can vary) we can change it.

The “while True:” line is a forever loop. While something is true whatever code that follows that is indented will be run. The while here is True so the code runs again and again.

```
from microbit import *
import neopixel

ring = neopixel.NeoPixel(pin0, 16)

# set some variables for colours
red = (75, 0, 0)
green = (0, 75, 0)
blue = (0, 0, 75)

# create a list of colours to be read as use_this_colour
colours = [red, green, blue]

while True:
    for use_this_colour in colours:
        for next_led in range(0, 16):
            ring[next_led] = use_this_colour
            ring.show()
            sleep(100)
```

Did the Code Not Work?

Did you miss the colon (:) at the end of the while and for line? And the code after the colons must be indented as in above. The Mu editor does this automatically if you remember to put in the colon. The list has square brackets too.

Hack the Code

Change the sleep value to speed the ring up or slow it down. Add your own colours to the list. Make the change go backwards by changing the (0, 16) to (16, 0, -1).

Chases and Functions

Here the code is extended to make the lights chase around the ring changing colour. The code is also broken down into a function that is called to work.

Functions break a piece of code down to hide the complexity. It also means that the same piece of code can be used again and again without being rewritten all the time and also used at any point in a script. A function in Python is placed at the top of the main code so that Python knows what it is before it is needed.

```
from microbit import *
import neopixel

ring = neopixel.NeoPixel(pin0, 16)

# set some variables for colours
red = (75, 0, 0)
green = (0, 75, 0)
blue = (0, 0, 75)
off = (0, 0, 0)

# define (def) a function and give it a clear, simple name.
# Here the colours are sent from the call at the bottom
# and renamed as use_this_colour.

def chase(use_this_colour):
    for led_number in range(0, 16):
        ring[led_number] = use_this_colour
        ring[led_number - 2] = off
        ring.show()
        sleep(50)

while True:
    chase(red)
    chase(green)
    chase(blue)
```

Hack the Code

Add your own colours again. Add their own calls for the function to the list under while True. See how easy it is to add more colour sweeps without complicating the code. Make sure you understand how the function is working.

Random Colours

Final part adds randomly selected colour values to the above programme. An additional call to a new function minimises the code even further. The very last line first of all calls the random colour generating function and then when those colour values are returned to the call the colour is then sent to the chase function to be used to sweep around the ring. There are two functions called from the last line.

```
from microbit import *
import neopixel
from random import randint

ring = neopixel.NeoPixel(pin0, 16)

# initially set the colour values to zero
# the variables need to be defined before use

red = 0
green = 0
blue = 0
off = (0, 0, 0)

def chase(use_this_colour):
    for led_number in range(0, 16):
        ring[led_number] = use_this_colour
        ring[led_number - 2] = off
        ring.show()
        sleep(50)

def random_colour():
    red = randint(0, 75)
    green = randint(0, 75)
    blue = randint(0, 75)
    colour = (red, green, blue)
    return colour

while True:
    chase(random_colour())
```

Did the Code Not Work?

There is an extra import at the top. And extra brackets in the last line.