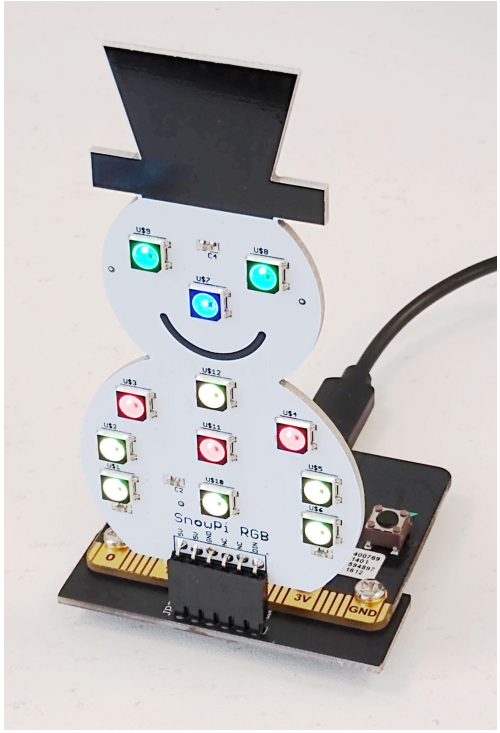
Microbit SnowPi & Python



The SnowPi is a prebuilt board of Red, Green and Blue Light Emitting Diodes, RGB LED or more commonly known as NeoPixels.

The board, shaped like a Snowman, has 12 of the NeoPixels built into it. Each NeoPixel can be any colour at any time including off. It is possible to chain hundreds of these NeoPixels together as long as there is a decent supply of power to them.

Normally with RGB LED each LED has four wires attached to it. One for ground, like the negative of a battery and then one connection for each colour. With those LED each connection needs to be attached to the controller. Here that would be the Microbit. So 12 LED would have one common ground but then 12 x 3 red, green and blue connections. 36 more wires. Neopixels only need three wires for all the LED. Ground, power and data. The data wire carries the information for each LED in the chain informing it what colour it should be then passing the data onto the next. If you look carefully at the LED on the Snowman you can just see there are three tiny LED in each one. Plus you may see a very tiny chip in there too. The chip reads the data, reacts accordingly and passes the data onto the next.

So how about lighting up these LED then?

Mu Editor

The Mu editor will be used to program the SnowPi. It provides very easy access to sending the code from the editor to the Microbit. The programming language being used is Python or to be precise MicroPython, though there is little difference between them.

Open the Mu editor and select a new clean file. Save this empty file so that if there are any problems your work will be saved. Attach the SnowPi to the Microbit so that the SnowPi has it’s back to the Microbit. Then connect the Microbit to the computer. Flash the empty code to the Microbit to make sure all connections are working.

All is now ready to get started.

First Code

There are three lines of code to start with. The first brings into your program everything related to the Microbit. The asterix * is the everything. The second does the same for the NeoPixels. The third sets the connections for the NeoPixels on the board. They are connected to Pin 2 and there are 12 NeoPixels on the SnowPi. Type these lines into the editor just as they are written.

```
from microbit import *
from neopixel import NeoPixel

snowman = NeoPixel(pin2, 12)
```

Everytime the code needs to do something with the LED the snowman line is called to work. The line above could easily be

cold_snowman = NeoPixel(pin2, 12)

As long as cold_snowman was called instead the code would still work. The snowman word here is just a useful reference. It would be a little odd to call it bear or dog instead. Though you could and it would still work!

Flash the code to the Microbit and check for any error messages on the Microbit.

Didn't Work?

Neopixel is spelt two different ways. NeoPixel and neopixel.

First Light

The LED have numbers to refer to each one. The nose is number 9. So let's start by lighting up the Snowman's nose. Add this code to the original and flash to the Microbit.

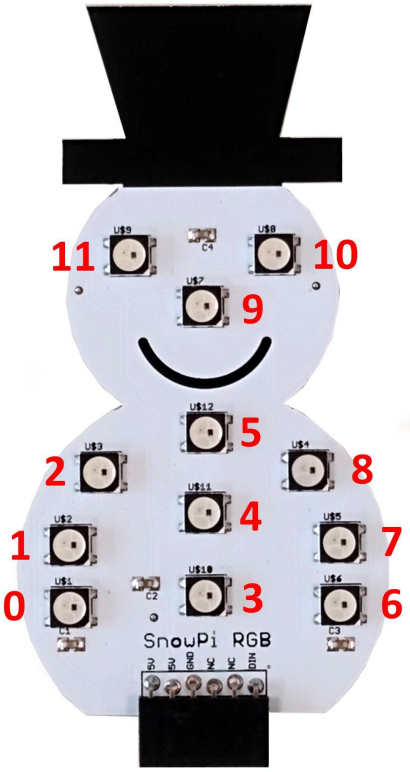
```
snowman[9] = (0, 0, 32)
snowman.show()
```

The nose should light up blue.
The (0, 0, 32) is the colour blue. The values in the brackets are (red, green, blue). So there is only a value for the blue part of the NeoPixel. No red or green. Try changing the values and reflashing the code. The snowman.show() command sends the data to all the LED and turns them on or off.
Do not use any value above 32, the Microbit or SnowPi might burn out if the LED is too bright.

Didn't Work?

There are two kinds of brackets used. [ ] and ( ).

All the LED



In computers numbering starts at zero so the first LED is number 0. As you can see the nose is number 9.

To light up all the LED just requires the LED number to be used in the square brackets in a new line. Do this for all the LED and then have just one snowman.show() line at the very bottom. Here are the first two LED lines for you to get started. Can you do all the rest?

```
snowman[0] = (0, 0, 32)
snowman[1] = (0, 0, 32)
snowman.show()
```

Hack the Code

Can you make all the LED different colours? Can you make both eyes the same colour but different to the rest of the Snowman? How about the nose being different too?

Didn't Work?

You will need 12 lines with the LED number and the colour values with just a single show line at the bottom. The code still requires the first three lines importing the information.

Tuples

With twelve lines of code with the colours there is an awful lot of numbers to type all the time. It would be much easier to give them a name. It would be a lot easier to see what the colour was meant to be in the code as well.

A tuple is a group of data within the brackets ( ). The data is in an order and once written can not be changed by the code. No more adding or taking away values. So the colours are a tuple.

Naming a tuple makes it easier to refer to it within the code. If you want a blue LED then say blue instead of (0, 0, 32).

Near the top of all your code add this line under the third line. And anywhere you would want an LED to be blue replace the numbers in the brackets with the reference blue. Like this:

```
from microbit import *
from neopixel import NeoPixel

snowman = NeoPixel(pin2, 12)

blue = (0, 0, 32)

snowman[9] = blue
snowman.show()
```

Hack the Code

Add in more colours to use. Red, green or yellow. Make some colours up and give them your own names. The only name rules are that they are one word but underscores are allowed i.e. bright_pink. The name can not start with a number. Python etiquette says that names are all lower case to. Not Pink, PINK but pink.

Simplify with Lists

There are parts of the snowman where you may colour each LED the same. Two eyes maybe. The snowman could also be wearing a scarf or a tie. By grouping the parts together in a list the code can be shortened. The list can also be worked through by the code turning the LED on one at a time.

Change your code so that it now looks like this. Maybe open a new Mu file and copy across what is needed instead of retyping everything. Use your own colours too.

```
from microbit import *
from neopixel import NeoPixel

snowman = NeoPixel(pin2, 12)

blue = (0, 0, 32)
green = (0, 32, 0)
red = (32, 0, 0)
yellow = (32, 32, 0)

tummy = [0, 1, 2, 3, 4, 5, 6, 7, 8]
scarf = [2, 4, 8]
tie = [3, 4, 5]
nose = [9]
eyes = [10, 11]

for light in tummy:
    snowman[light] = yellow
    snowman.show()
    sleep(1000)
```

The last block of code is a for loop. The loop goes through each number in the list and calls them light. The first light value in tummy is 0. The LED is set to yellow, turned on and then the code sleeps for 1 second (1000 milliseconds in the code). It then resets light to the next value, 1, and repeats until the list runs out of values. Then the code stops.

Flash the code and see if the tummy lights up yellow or maybe with one of your own colours.

Hack the Code

Now add another for loop for the eyes, nose and either the scarf or tie. All in different colours.

Didn’t Work?

The colon : at the end of the for line is VERY important. So are the indented lines after the colon.

Stop Repeating Code

The last part added yet again a lot of repeated code. There is a way to reuse the same code again and again but with different values. Now the code needs some functions.

A function needs to be defined before it is used. So functions need to go at the top of the code before all the real action starts. So under the lines setting up the tummy and eyes add this:

```
def light_up():
```

This is the first part. The def is short for define. The light_up is just the name we have given it. You could use “turn_on_light” or “switch_on” or “turn_on_the_snowman_lights_to_a_colour”. All would work and be allowed. Always make the names of the functions, like the tuples and lists and other variables, something useful and memorable.

The two brackets ( ) are always needed with functions. At the moment there is nothing in these brackets but there will be.

Now add more to the function:

```
def light_up():
    for led in tummy:
        snowman[led] = yellow
        snowman.show()
        sleep(5)
```

The very short sleep at the end stabilises this particular code. It may be due to the Microbit or the SnowPi or the combination. Here it just makes the code work better in the physical world of the LED on the SnowPi.

This function can now be called into action with the simple line:

```
light_up()
```

In your code where you turn the tummy LED on, replace that code with the light_up(). Flash your code and everything should be the same as before.

But you haven’t used less code yet, in fact more. Now the code needs to be changed so that the same function, light_up(), is used for all parts of the snowman. To do this we need to send to the function each time it is used the colour of the LED you require and which part of the snowman it is for. Like this:

```
def light_up(body_part, colour):
    for led in body_part:
        snowman[led] = colour
        snowman.show()
        sleep(5)   # pause needed for stability

light_up(tummy, yellow)
light_up(eyes, green)
```

Now the program uses the same for each body part of the snowman. The values sent on the light_up() call are substituted into the function where they are needed. Light up the whole body with the function.

Adding Interactivity

The final part of this SnowPi worksheet is to add some interaction between the SnowPi and you using the buttons. It is easy to check if a button has been pressed with this code.

```
if button_a.was_pressed():
   # do something if Button A was pressed
```

Then if there is a loop that forever checks if the buttons have been pressed then the Microbit can react to your inputs. Every time the button is pressed a different colour snowman appears.
Change the code you have to something like this. Use your own colours wherever you like.

```
from microbit import *
from neopixel import NeoPixel

snowman = NeoPixel(pin2, 12)

blue = (0, 0, 32)
green = (0, 32, 0)
red = (32, 0, 0)
yellow = (32, 32, 0)

tummy = [0, 1, 2, 3, 4, 5, 6, 7, 8]
scarf = [2, 4, 8]
tie = [3, 4, 5]
nose = [9]
eyes = [10, 11]

def light_up(where, colour):
    for led in where:
        snowman[led] = colour
        snowman.show()
        sleep(5)   # pause needed for stability

while True:
    if button_a.was_pressed():
        light_up(tummy, yellow)
        light_up(eyes, blue)
        light_up(scarf, red)
        light_up(nose, green)
    elif button_b.was_pressed():
        light_up(tummy, yellow)
        light_up(eyes, blue)
        light_up(tie, red)
        light_up(nose, green)
    else:
        display.scroll("Press a button")
        snowman.clear()
```