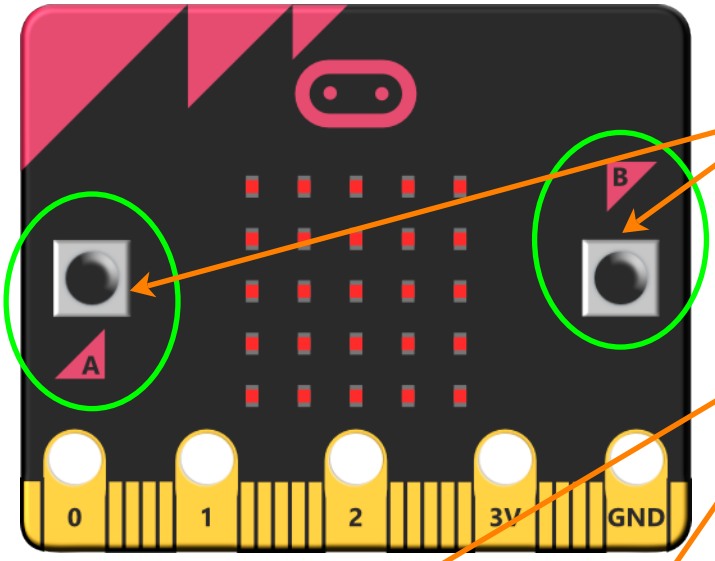
BBC micro:bit "BUTTONS"

Worksheet: 3



The BBC micro:bit has quite a few ways of interacting with it, however, most obvious are the clearly labelled A and B buttons on the front.

Begin a new program (blank editor) and enter the code listed below.

This code needs some explanation. We start with a few variables **mbx** (LED x position), **mby** (LED y position), **mbdir** (movement direction) and **press** (status of the buttons).

We then clear (**MBCLS**) the micro:bit display and **MBPLOT** an LED at the x and y positions at full brightness. 0 is off, 255 is full brightness.

Next we start a main **LOOP** and check first to see if both buttons are not being pressed and if so sets the **press** variable to 0. The next **IF** statement now checks to see if either button **is** being pressed but only if the press variable is **FALSE** (no buttons pressed).

This allows us to block a key being held down so it must be released before it can move again. Great for stopping auto-fire in games!

```
mbx = 2
mby = 4
mbdir = 0
press = FALSE
MBCLS
MBPLOT (mbx, mby, 255)
LOOP
  IF MBBUTTONA = FALSE AND MBBUTTONB = FALSE THEN press = FALSE
  IF MBBUTTONA = TRUE OR MBBUTTONB = TRUE AND press = FALSE THEN
    press = TRUE
    PLAYNOTE (0, 150 + (20 * mbx), 0.05)
    MBPLOT (mbx, mby, 0)
    IF MBBUTTONA THEN mbdir = -1
    IF MBBUTTONB THEN mbdir = 1
    mbx = mbx + mbdir
    IF mbx < 0 THEN mbx = 0
    IF mbx > 4 THEN mbx = 4
    MBPLOT  (mbx, mby, 255)
  ENDIF
ENDIF
UPDATE
REPEAT
```

If a button is pressed we set the **press** variable to **TRUE** and then play a little sound using **PLAYNOTE**(channel, frequency, duration). The frequency is adjusted depending on the **mbx** position so gets higher and lower depending which way it's moving.

Next we turn off the LED at the old position and then check for left or right and add either '-1' or a '1' to the position. Adding -1 will reduce the position by 1 and adding a 1 will increase it. Now we check to see if it has got to the edge of the display and if so keep it there.

Finally we light up the LED in the new position and **REPEAT** the **LOOP**.

ADVANCED CHALLENGE:
This program could be the start of a simple 'falling ruby' game. Could you make a ruby (an LED!) fall from the top of the matrix and if it's caught play a sound and keep a score?