UCL BBC micro:bit MicroPython Tutorial

Release

Rae Harbird, Stephen Hailes

Apr 26, 2018

This documentation can be found at <http://microbit-challenges.readthedocs.io/en/latest/index.html>.

The tutorial sheets are designed to give students an introduction to the features of the micro:bit. Short practical examples are provided and students are invited to design solutions to problems using the fundamental building blocks presented.

Teaching students to code using a microprocessor with embedded sensors on the board enables learners to get immediate feedback from their code without having to learn any electronics beforehand. This approach to learning coding was designed by Prof. Stephen Hailes, UCL. His team developed the Engduino expressly for this purpose . The design of the micro:bit was strongly influenced by the Engduino and some of this material is taken directly from the Engduino tutorial sheets. Likewise, in some places, the content is an abridged version of the BBC Micro:bit MicroPython documentation.

The Challenge sheets can be used for team competitions or just for fun in the classroom. Some of them were adapted from the exercises by M. Atkinson on the website Multiwingspan.

To download this documentation in pdf, epub or html format, click on the link at the bottom of the sidebar on the left:

If you would like to contribute to this resource, go ahead! Install git and create a branch. It would be great to have more challenges and some projects.

Micro:bit - Getting Started

The BBC micro:bit is a tiny computer that you can use to create all kinds of projects from robots to musical instruments – the possibilities are endless. Let's take a look at the features that you can use in your designs:

- 25 red LED lights that can flash messages.

- Two programmable buttons (A and B) that can be used to tell the micro:bit when to start and stop things.

- A thermistor to measure the temperature.

- A light sensor to measure the change in light.

- An accelerometer to detect motion.

- A magnetometer to tell you which direction you're heading in.

- A radio and a Bluetooth Low Energy connection to interact with other devices.
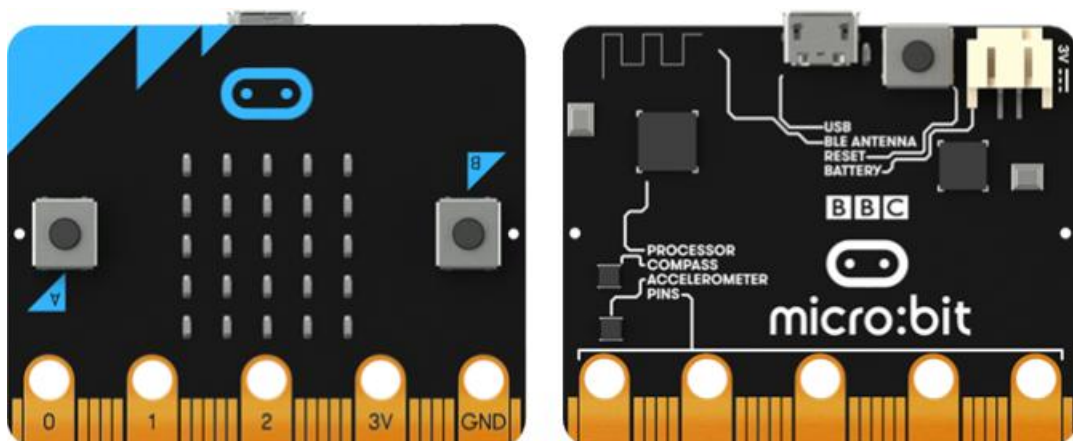
Now you will create your first micro:bit program; after that, well we've listed a few ideas but it's really up to you.

1.1 Your First Program

Coding using the micro:bit is composed of these 4 steps. You can expect to go around the loop quite a few times before you get your code working.

Life cycle of a computer program
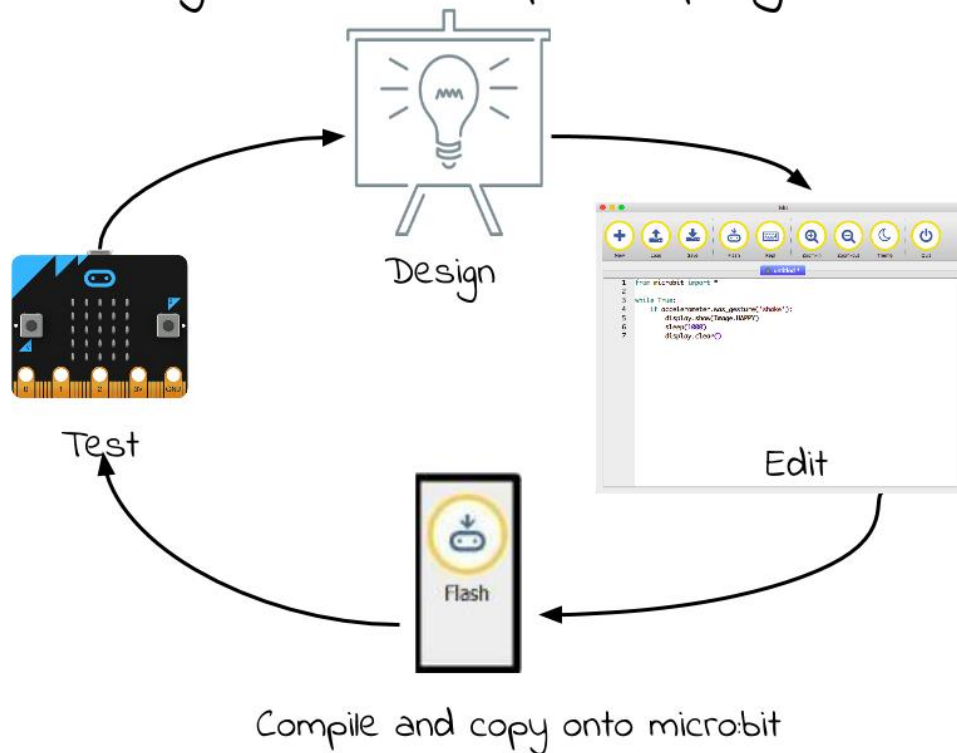
1.1.1 Design the Code

First of all you are going to write a program to display the message "Hello UCL!" followed by an image on the display of your micro:bit. There's not much planning and design to do here, but just so that you understand what a plan might look like:

My Plan

Repeat forever:
1. Display "Hello UCL!"
2. Display butterfly
3. Print "Hi There!!"
4. wait for 2 seconds

1.1.2 Write the Code

We will use a special text editor to write our programs, it looks like the one shown here:
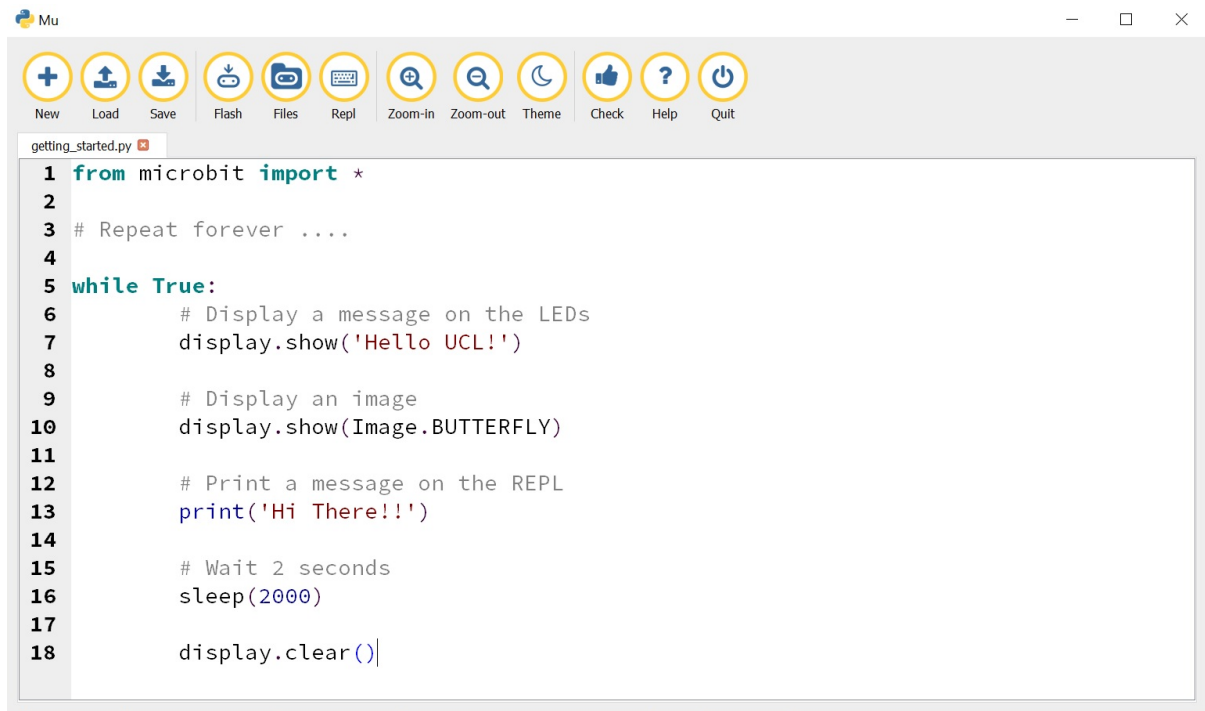
Let's go through this line-by-line:

```python
while True:
```

This means do something (whatever follows this statement and is indented) forever and ever and ever. This is called a loop, it's a bit like a video clip that's stuck on repeat. `True` and `False` have a special meaning in python. `True` is always, well `True`. The rest of the program is straightforward:

```python
from microbit import *

while True:
    display.show('Hello UCL!')
    display.show(Image.BUTTERFLY)
    print('Hi There!!!')
    sleep(2000)
```

This displays `Hello UCL` on the LED display one character at a time and then shows the butterfly. The statement `print('Hi There!!')`, will print the message in the REPL. Press the REPL button in the menu now to show the REPL window:
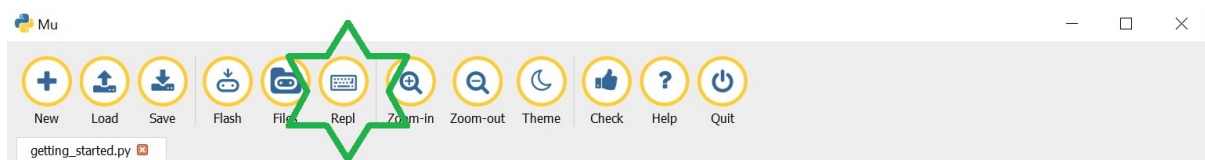
The REPL window shows us messages from the micro:bit and also allows us to send commands directly to the micro:bit. For now, we'll just be using the REPL to see messages that we print and error messages.

You might be wondering why we've asked the micro:bit to sleep for `2000`! This value is in microseconds so we've really only asked it to sleep for 2 seconds. That will give us enough time to see the image before the micro:bit starts all over again.

1.1.3 Upload the Code

Final checks. Is your micro:bit connected to your computer? Yes? Then press the flash button:
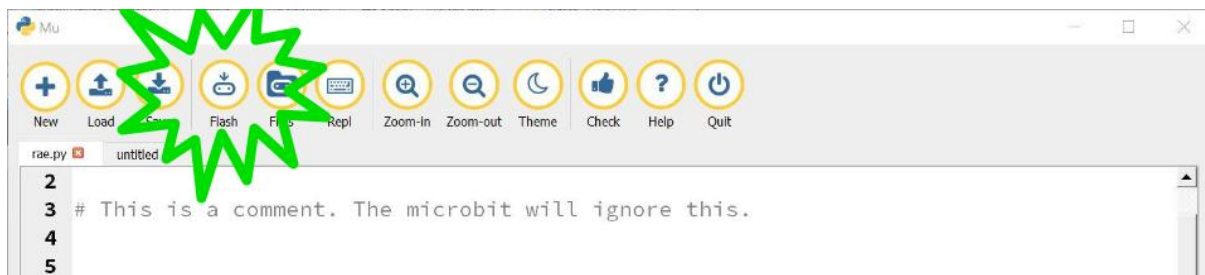
You should see the message and the image displayed on the micro:bit and the message "Hello There!!" should be printed on the REPL.
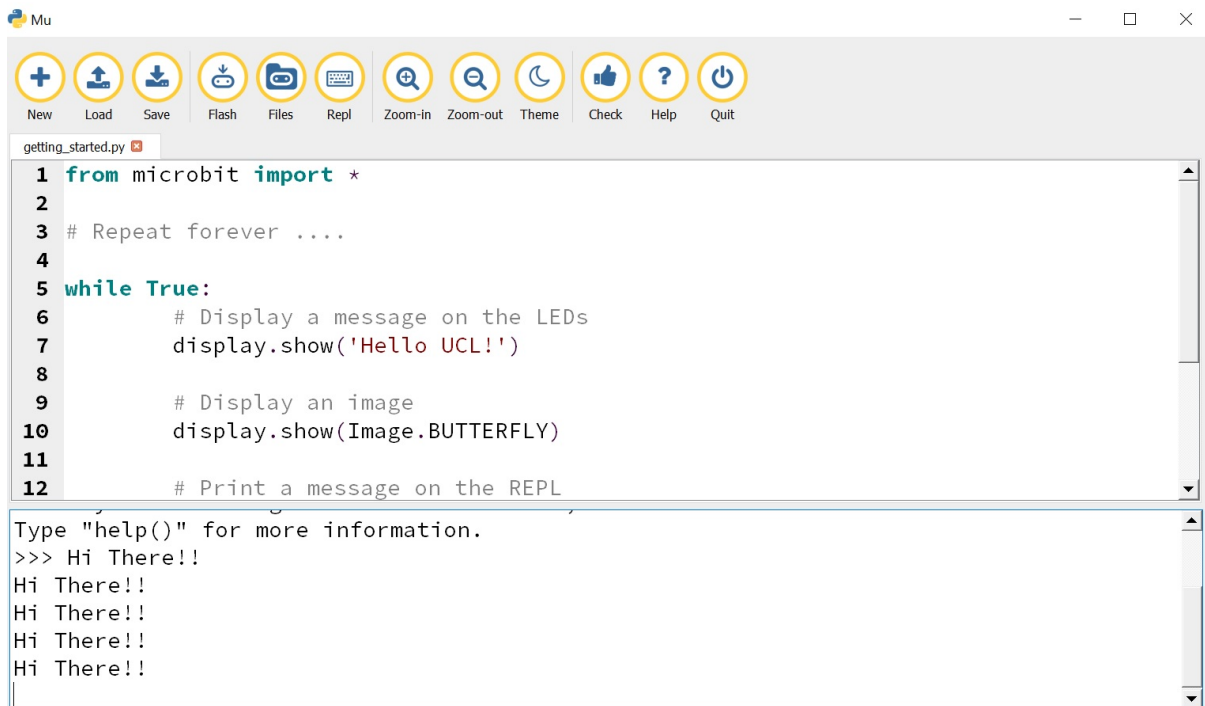
1.1.4 Make a change

Change the text that is displayed on the screen and make it scroll across the LED display. You can do this by changing the word `show` to `scroll`. Don't forget to save your program and remember to `flash` the new code to the micro:bit.
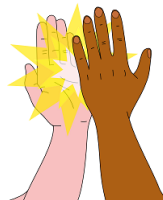
You have written your first program. Carry on and see what else you can do with the micro:bit.

LED Display

This is a quick guide to some of the things you can do with the LED display. The idea is that you can use this information to experiment and create something for yourselves. Try stuff out – see what happens and see what you can make. There are 25 LEDs set out like the picture below. The LEDs are numbered from (0,0) in the top left hand corner, to (4,4) in the bottom right hand corner. You can use the LEDs like a very tiny screen to display single characters, a string of characters or a small picture like the smiley face here. The LEDs can be set to different brightnesses. Let's find out how to use them.
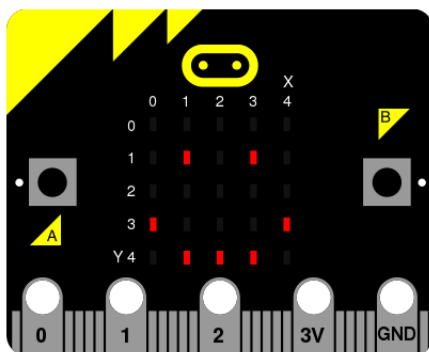
2.1 Basic Functions

2.1.1 Display a string or an image

You can display characters on the LED display very easily like this:

```python
from microbit import *

display.show("Hello")
```

The characters you display must be within a pair of quotes, either " " or ' '.

MicroPython comes with lots of built-in pictures to show on the display. For example, to make the device appear happy you type:

```
from microbit import *

display.show(Image.HAPPY)
```

Here's some of the the other images you can use:

- Image.HEART, Image.HEART_SMALL

- Image.HAPPY, Image.SMILE, Image.SAD, Image.CONFUSED, Image.ANGRY, Image.
  ASLEEP, Image.SURPRISED, Image.SILLY, Image.FABULOUS, Image.MEH, Image.YES,
  Image.NO

- Image.ARROW_N, Image.ARROW_NE, Image.ARROW_E, Image.ARROW_SE, Image.ARROW_S,
  Image.ARROW_SW, Image.ARROW_W, Image.ARROW_NW

- Image.MUSIC_CROTCHET, Image.MUSIC_QUAVER, Image.MUSIC_QUAVERS

- Image.XMAS, Image.PACMAN, Image.TARGET, Image.ROLLERSKATE, Image.
  STICKFIGURE, Image.GHOST, Image.SWORD, Image.UMBRELLA

- Image.RABBIT, Image.COW, Image.DUCK, Image.HOUSE, Image.TORTOISE, Image.
  BUTTERFLY, Image.GIRAFFE, Image.SNAKE

2.1.2 Scroll a string

To continuously scroll a string across the display you can use:

```
from microbit import *

display.scroll("Hello!")
```

2.1.3 Clear the display

If you want to clear the LED display, you can do so like this:

```
from microbit import *

display.clear()
```

2.2 Advanced Functions

2.2.1 Set a pixel

You can set a pixel in the LED display using the set_pixel method:

```
from microbit import *

display.set_pixel(0,4,9)
```

This sets the LED at column 0 and row 4 to a brightness of 9. The brightness value can be any whole number between 0 and 9 with 0 switching the LED off and 9 being the brightest setting. You could use a for loop to set all the LEDs one at a time:

```
from microbit import *

display.clear()
for x in range(0, 5):
```

```
    for y in range(0, 5):
        display.set_pixel(x,y,9)
```

The `for loop` lets you execute a loop a specific number of times using a counter. The outer loop:

```
for x in range(0,5)
```

will execute the loop five times substituting x consecutive values in the range 0 to 4 for x each time. The loop will stop before it reaches the final value in the range.

The inner loop:

```
for y in range(0,5):
```

will execute the loop five times substituting y consecutive values in the range 0 to 4 for y each time. The loop will stop before it reaches the final value in the range.

2.2.2 DIY images

Of course, you want to make your own image to display on the micro:bit, right?

That's easy. Each LED pixel on the physical display can be set to one of ten values. If a pixel is set to 0 (zero) then it's off. It literally has zero brightness. However, if it is set to 9 then it is at its brightest level. The values 1 to 8 represent the brightness levels between off (0) and full on (9).

Armed with this information, it's possible to create a new image like this:

```python
from microbit import *

boat = Image("05050:"
             "05050:"
             "05050:"
             "99999:"
             "09990")

display.show(boat)
```

In fact, you don't need to write this over several lines. If you think you can keep track of each line, you can rewrite it like this:

```python
boat = Image("05050:05050:05050:99999:09990")
```

(When run, the device should display an old-fashioned "Blue Peter" sailing ship with the masts dimmer than the boat's hull.)

Have you figured out how to draw a picture? Have you noticed that each line of the physical display is represented by a line of numbers ending in : and enclosed between " double quotes? Each number specifies a brightness. There are five lines of five numbers so it's possible to specify the individual brightness for each of the five pixels on each of the five lines on the physical display.

2.2.3 Animation

Static images are fun, but it's even more fun to make them move. This is also amazingly simple to do with MicroPython ~ just use a list of images!

Luckily we have a couple of lists of images already built in. They're called `Image.ALL_CLOCKS` and `Image.ALL_ARROWS`:

```python
from microbit import *

display.show(Image.ALL_CLOCKS, loop=True, delay=100)
```

We tell MicroPython to use `Image.ALL_CLOCKS` and it understands that it needs to show each image in the list, one after the other. We also tell MicroPython to keep looping over the list of images (so the animation lasts forever) by saying `loop=True`. Furthermore, we tell it that we want the delay between each image to be only 100 milliseconds (a tenth of a second) with the argument `delay=100`.

Now, here's how to create your own animation. First you need to create a list. Here is a list of boats:

```
all_boats = [boat1, boat2, boat3, boat4, boat5, boat6]
```

You can store anything in a list with Python, even images. In my example I'm going to make my boat sink into the bottom of the display. To do that, I'm going to create 6 images and put them into a list called `all_boats`:

```python
from microbit import *

boat1 = Image("05050:"
              "05050:"
              "05050:"
              "99999:"
              "09990")

boat2 = Image("00000:"
              "05050:"
              "05050:"
              "05050:"
              "99999")

boat3 = Image("00000:"
              "00000:"
              "05050:"
              "05050:"
              "05050")

boat4 = Image("00000:"
              "00000:"
              "00000:"
              "05050:"
              "05050")

boat5 = Image("00000:"
              "00000:"
              "00000:"
              "00000:"
              "05050")

boat6 = Image("00000:"
              "00000:"
              "00000:"
              "00000:"
              "00000")

all_boats = [boat1, boat2, boat3, boat4, boat5, boat6]
display.show(all_boats, delay=200)
```

Finally, we can tell MicroPython to animate a list of images using `display.show`.

2.3 Projects with LED Display

- Try out some of the built-in images to see what they look like.
- Animate the `Image.ALL_ARROWS` list. How do you avoid looping forever (hint: the opposite of `True` is `False`). Can you change the speed of the animation?

- Make your own image. Next try to make it fade out and then fade in again?

- Make a sprite, use a single LED on the display. Can you make it jump when you press a button?

Buttons

The micro:bit has two buttons: labelled `A` and `B`.
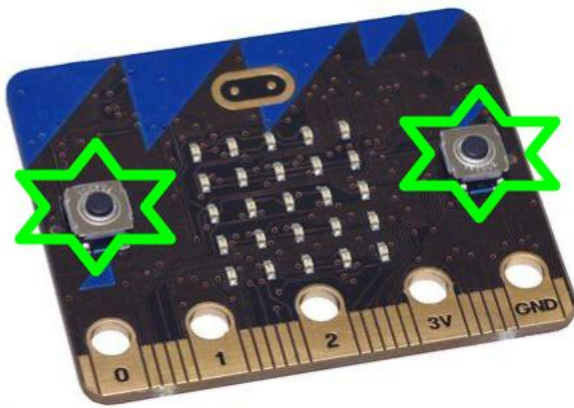
You can use the buttons to get input from the user. Perhaps you'd like to start or stop your program with a button press or maybe you'd like to know how many times each button has been pressed.

3.1 Basic Functions

3.1.1 Checking whether a button is pressed

Sometimes we just want a program to wait until something happens, for example: we could ask the micro:bit to wait until, say, button `A` is pressed and then print a message. We could do that like this:

```python
from microbit import *

while True:
    if button_a.is_pressed():
        display.scroll("A")
```

Let's break this up into parts. The first bit:

```python
while True:
```

This is a standard way to tell the micro:bit to repeat whatever code follows it forever and ever and ever . . . . . you get the idea. The next line looks similar to normal English:

```
while True:
    if button_a.is_pressed():
        display.scroll("A")
```

This means, if button `A` is pressed then display an `A` on the LED screen.

Of course, we usually want to do something a bit more complicated than that. There is a way to check two things, we can use an `if` with an `else` like this:

```
while True:
    if button_a.is_pressed():
        display.scroll("A")
    else:
        display.scroll(Image.ASLEEP)
```

This means, if button `A` is pressed then display an `A` on the LED screen, otherwise, display `Image.ASLEEP`.

3.1.2 Counting the number of presses

Sometimes you might want to count the number of button presses in a time period. You can do this using the `get_presses()` method. Here is an example:

```
from microbit import *

    while True:
        sleep(3000)
        count = button_a.get_presses()
        display.scroll(str(count))
```

The micro:bit will sleep for 3 seconds and then wake up and check how many times button `A` was pressed. The number of presses is stored in a variable called `count`. We can't print numbers directly on the LED screen so we convert `count` to a string and then display it. Can you think of another way to do this? (Hint: check whether the button has been pressed and add 1 to a counter if it has).

3.2 Advanced Functions

3.2.1 Checking for both buttons

It is possible to check a series of events by using `if`, `elif` and `else`. Say you wanted to check whether button `A` was pressed or button `B` was pressed or whether both buttons were pressed at the same time. We could do that like so:

```
from microbit import *

while True:
    if button_a.is_pressed() and button_b.is_pressed():
        display.scroll("AB")
        break
    elif button_a.is_pressed():
        display.scroll("A")
    elif button_b.is_pressed():
        display.scroll("B")
    sleep(100)
```

Note: The keyword `elif` just means `else if`. You can use the longer form `else if` if you want.

The code above displays the letter corresponding to the button. If both buttons are pressed at the same time it displays `AB`.

3.2.2 Has the button been pressed?

The problem with using `is_pressed()` is that unless you are pressing the button at that precise moment then you won't detect whether the button was ever pressed or not.

The `was_pressed()` function is useful is you want to write code that occasionally checks whether the button has been pushed but then goes on to do something else. While the code is doing the something else, it might be the case that the user pushes the button and lets it go and, because you haven't checked to see if it's pressed then, you might miss it. Well, the following function will tell you whether a button has been pushed or released since you last called that function - while your code is doing something else. In this way you need never miss a button press again:

```python
from microbit import *

while True:
    if button_a.was_pressed():
        display.scroll("A")
    else:
        display.scroll(Image.ASLEEP)
    sleep(1000)
```

What you'll see is that the display will show an `A` for a second if you press the button, and then `Image.ASLEEP` is displayed. If you press the button while the program is delaying, then the `A` won't show up immediately, but they will show up when it next tests to see if the button has been pressed. You'll see this more clearly if you make the delay bigger.

Now try using `button_a.isPressed()` instead of `button_a.was_pressed()`. What you will find is that if you press the button while the code is delaying the micro:bit will never realise that you pressed it at all.

3.3 Ideas for Projects with the Buttons

- Change what is displayed when you press the button.
- Games that need user input... can you make one up?

Accelerometer

The accelerometer on a BBC micro:bit measures acceleration, as its name suggests. The accelerometer can measure accelerations of between +2g to -2g.
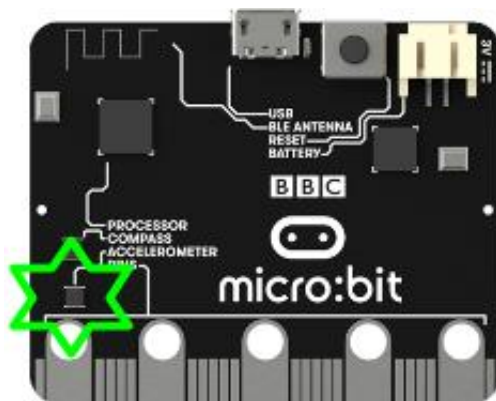
The micro:bit measures movement along three axes:

- X - tilting from left to right.
- Y - tilting forwards and backwards.
- Z - moving up and down.
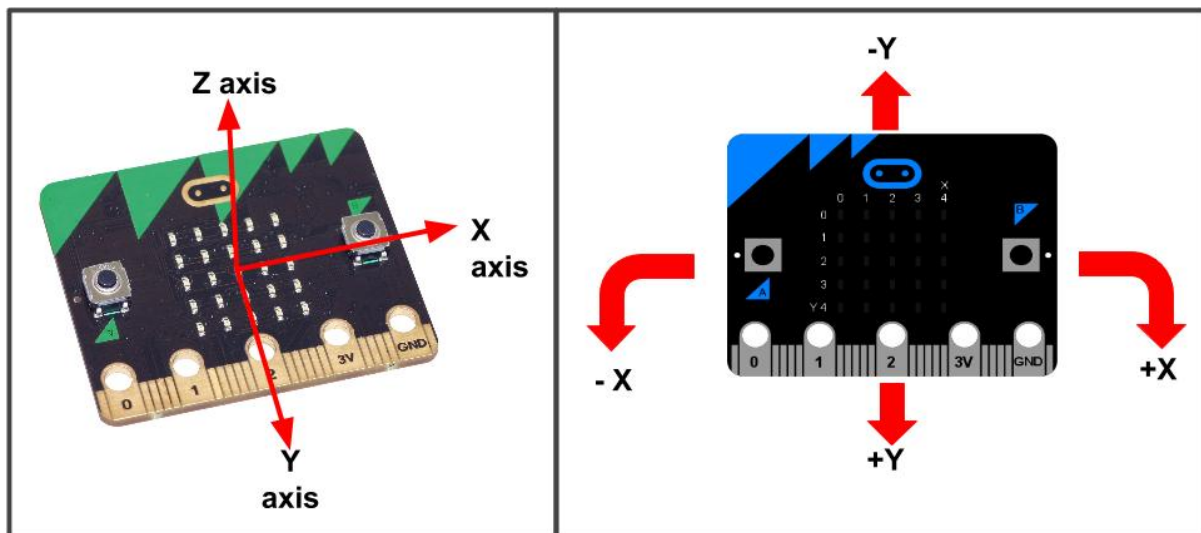
4.1 Basic Functions

The measurement for each axis is a positive or negative number indicating a value in milli-g's. When the reading is 0 you are "level" along that particular axis. 1024 milli-g is the acceleration due to gravity.

You can access acceleration measurements one at a time or get all three values at once and store them in a list. Ask your teacher about lists later, but for now, just use the following code:

```python
from microbit import *

while True:
    x = accelerometer.get_x()
    y = accelerometer.get_y()
    z = accelerometer.get_z()
    print("x, y, z:", x, y, z)
    sleep(500)
```

Upload this and open the serial monitor. Hold the micro:bit flat with the LEDs uppermost. You should see that the X and Y accelerations are near to zero, and the Z acceleration is close to -1024. This tells you that gravity is acting downwards relative to the micro:bit. Flip the board over so the LEDs are nearest the floor. The Z value should become positive at +1024 milli-g. If you shake the micro:bit vigorously enough, you'll see that the accelerations go up to ±2048 milli-g. That's because this accelerometer is set to measure a maximum of ±2048 milli-g: the true number might be higher than that.

If you've ever wondered how a mobile phone knows which up to show the images on its screen, it's because it uses an accelerometer in exactly the same way as the program above. Game controllers also contain accelerometers to help you steer and move around in games.

4.1.1 Gestures

The really interesting side-effect of having an accelerometer is gesture detection. If you move your BBC micro:bit in a certain way (as a gesture) then MicroPython is able to detect this.

MicroPython is able to recognise the following gestures: `up`, `down`, `left`, `right`, `face up`, `face down`, `freefall`, `3g`, `6g`, `8g`, `shake`. Gestures are always represented as strings. While most of the names should be obvious, the `3g`, `6g` and `8g` gestures apply when the device encounters these levels of g-force (like when an astronaut is launched into space).

To get the current gesture use the `accelerometer.current_gesture` method. Its result is going to be one of the named gestures listed above. For example, this program will only make your device happy if it is face up:

```
from microbit import *

while True:
    gesture = accelerometer.current_gesture()
    if gesture == "face up":
        display.show(Image.HAPPY)
    else:
        display.show(Image.ANGRY)
```

Once again, because we want the device to react to changing circumstances we use a `while` loop. Within the *scope* of the loop the current gesture is read and put into `gesture`. The `if` conditional checks if `gesture` is equal to `"face up"` (Python uses == to test for equality, a single equals sign = is used for assignment - just like how we assign the gesture reading to the `gesture` object). If the gesture is equal to `"face up"` then use the display to show a happy face. Otherwise, the device is made to look angry!

4.2 Advanced Functions

There aren't any for the accelerometer. But it's worth looking at how we can use the 3D acceleration to detect different kinds of motion like a being shaken. Acceleration is what is known as a vector quantity (talk to your maths teacher) – it has a magnitude (size, length) and a direction. To get the overall magnitude, irrespective of orientation, we need to do a bit of maths. If we only had X and Y axes (i.e. we had a 2D accelerometer) the situation would be:
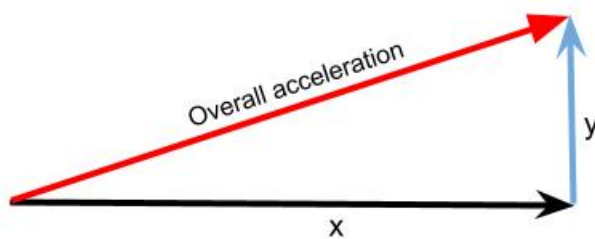
We can calculate the magnitude (length) of the resultant from Pythagoras' rule:

$$acceleration = \sqrt{x^2 + y^2}$$

BUT, we have a 3D accelerometer - and the same principle holds where we have X, Y and Z axes. So the overall magnitude of the resultant acceleration vector is:

$$acceleration = \sqrt{x^2 + y^2 + z^2}$$

Here is the code to calculate the overall acceleration:

```
from microbit import *
import math

while True:
    x = accelerometer.get_x()
    y = accelerometer.get_y()
    z = accelerometer.get_z()
    acceleration = math.sqrt(x**2 + y**2 + z**2)
    print("acceleration", acceleration)
    sleep(500)
```

Now if you keep the the accelerometer still (put it on the desk), this will give an acceleration of about 1g, irrespective of what orientation you have the BBC micro:bit in – and it will be different to that as you move it about. Actually, the value will vary slightly even if you keep it still, because the accelerometer isn't a perfect measuring device. Dealing with this is a process called calibration and is something we have to do when we need to know a quantity accurately.

4.3 Ideas for Projects with the Accelerometer

- Using the BBC micro:bit music library, play a note based on the the reading from the accelerometer. Hint: set the pitch to the value of the accelerometer measurement.

- Display the characters 'L' or 'R' depending on whether the BBC micro:bit is tilted to the left or the right.

- Make the LEDs light up when the magnitude of the acceleration is greater than 1024 milli-gs.

- Shake the micro:bit to make the LEDs light up.

- Make a dice, hint: use one of the Python random functions. Type `import random` at the top of your program and use `random.randrange(start, stop)`. This will generate a random number between `start` and `stop` - 1.

Compass

A magnetometer measures magnetic field strength in each of three axes. It can be used to create a digital compass or to explore magnetic fields, such as those generated by a permanent magnet or those around a coil through which a current is running.
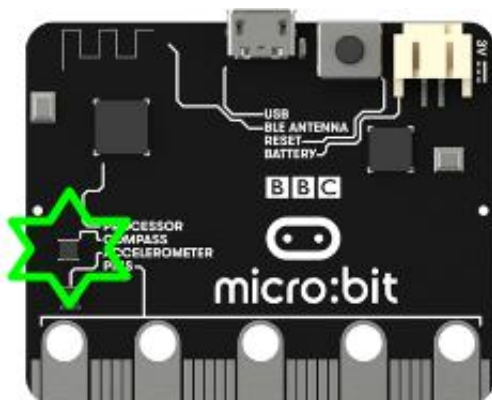
The interpretation of magnetic field strength is not easy. The driver for the magnetometer returns raw values. Each magnetometer is different and will require calibration to account for offsets in the raw numbers and distortions to the magnetic field introduced by what are known as hard and soft iron interference.

Before doing anything else, you should calibrate your BBC micro:bit but beware:

> **Warning:** Calibrating the compass will cause your program to pause until calibration is complete. Calibration consists of a little game to draw a circle on the LED display by rotating the device.

5.1 Basic Functions

The interface to the magnetometer looks very much like the interface to the accelerometer, except that we only use the x and y values to determine direction. Remember, before using the compass you should calibrate it, otherwise the readings may be wrong:

```python
from microbit import *
```

```
compass.calibrate()

while True:
    x = compass.get_x()
    y = compass.get_y()
    print("x reading: ", x, ", y reading: ", y)
    sleep(500)
```

This reads the magnetic field in two dimensions (like an actual compass) and outputs the values, which seems easy enough. The stronger the field, the bigger the number. Try it out and figure out which is the x axis for the magnetometer. If you want to know the direction you need to calculate $tan^{-1}(y/x)$, in python this is written as:

```
import math
from microbit import *

compass.calibrate()

while True:
    x = compass.get_x()
    y = compass.get_y()
    angle = math.atan2(y,x) *180/math.pi
    print("x", x, " y", y)
    print("Direction: ", angle)
    sleep(500)
```

The 180/PI is because the angle returned is in radians rather than degrees. Fortunately, the BBC micro:bit has a function to calculate the heading automatically:

```
compass.heading()
```

This gives the compass heading, as an integer in the range from 0 to 360, representing the angle in degrees, clockwise, with north as 0. You will still to calibrate the device before you use `compass.heading`.

5.2 Ideas for Projects with the Compass

- Make the micro:bit into a compass that illuminates the LED closest to where north lies.

- Calibrate your magnetometer. Find out whether the calibration stays (about) the same over time and whether it is the same inside or outside a building or near something that has a lot of steel in it (e.g. a lift).

Thermometer

The thermometer on the micro:bit is embedded in one of the chips – and chips get warm when you power them up. Consequently, it doesn't measure room temperature all that well. The chip that is used to measure temperature can be found on the left hand side of the back of the micro:bit:
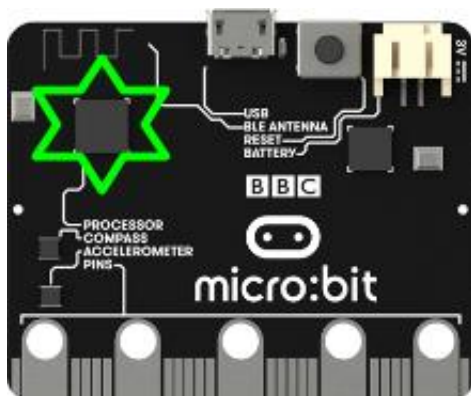
6.1 Basic Functions

There is only one basic function for the thermometer – to get the temperature, which comes back as an integer in degrees Celsius:

```python
from microbit import *

while True:
    temp = temperature()
    display.scroll(str(temp) + 'C')
    sleep(500)
```

Compile and run the code and see what happens.

You will see that the The temperature the thermometer measures will typically be higher than the true temperature because it's getting heated from both the room and the electronics on the board. If we know that the temperature is 27°C but the micro:bit is consistently reporting temperatures that are, say, 3 degrees higher, then we can correct the reading. To do this accurately, you need to know the real temperature without using the micro:bit. Can you find a way to do that?

6.2 Ideas for Projects with the Thermometer

- Try calibrating the thermometer. Does it still give the right temperature when you move it to a warmer or cooler place?

- Make the LEDs change pattern as temperature changes

- Find out how much the temperature changes in a room when you open a window – what do you think that tells you about heating energy wasted?

Music

This is a quick guide to some of the things you can do with micro:bit music. The idea is that you can use this information to experiment and create something for yourselves. You can use the micro:bit to play simple tunes, provided that you connect a speaker to your board.

If you are using a speaker, you can connect your micro:bit using crocodile clips like this:
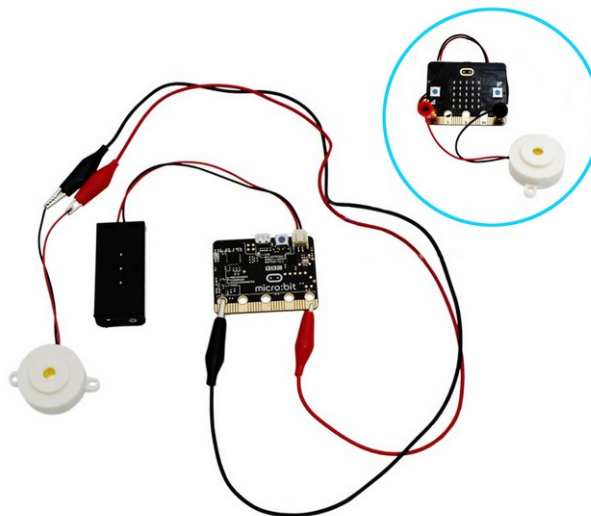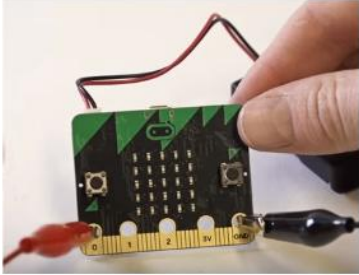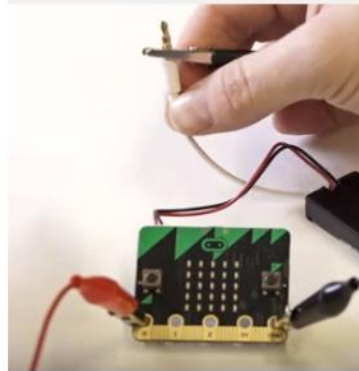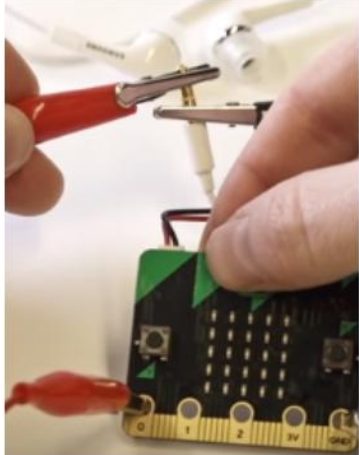
Fig. 7.1: Image from: <https://www.kitronik.co.uk/blog/microbit-alarm-kitronik-university/>

> **Warning:** You cannot control the volume of the sound level from the micro:bit. Please be very careful if you are using headphones. A speaker is a better choice for work with sound.

If you are using headphones you can use crocodile clips to connect your micro:bit to some headphones like this:

Connect a clip to pin 0 and another to GND.	Connect the other end of the cable connected to GND to *the base* of the headphone jack..	Connect the other end of the cable connected to pin 0 to *the tip* of the headphone jack.

7.1 Basic Functions

7.1.1 Play a tune

Let's play some music:

```python
from microbit import *
import music

music.play(music.NYAN)
```

Note: You must import the `music` module to play and control sound.

MicroPython has quite a lot of built-in melodies. Here's some of them, try them out:

- `music.DADADADUM`
- `music.ENTERTAINER`
- `music.PRELUDE`
- `music.ODE`
- `music.NYAN`
- `music.RINGTONE`

7.1.2 Make your own tune

You can write your own tune, here is a snippet of code showing how to play a sound. The number after the note is the octave and an octave can be a number from 1 to 8. The number after the colon says how long the note will last:

```python
from microbit import *
import music

# Play a 'C'
```

```
music.play('C')

# Play a 'C' for 4 beats long
music.play('C:4')

# Play a 'C' in octave number 3 for 4 beats long
music.play('C3:4')
```

Playing a series of notes one after the other is easy, you just put the notes you want to play in a list:

```
from microbit import *
import music

# Tune: Frere Jacques
tune = ["C4:4", "D4:4", "E4:4", "C4:4", "C4:4", "D4:4", "E4:4", "C4:4",
        "E4:4", "F4:4", "G4:8", "E4:4", "F4:4", "G4:8"]
music.play(tune)
```

7.2 Advanced Functions

You can also specify the note you want to play as a `frequency`. Take a look at this example where we make a police siren. The clever thing here is that the frequency or note is controlled by a `for` loop:

```
while True:
        for freq in range(880, 1760, 16):
                music.pitch(freq, 6)
        for freq in range(1760, 880, -16):
                music.pitch(freq, 6)
```

Can you guess what this does? Each time around the loop a new frequency is calculated by adding (or subtracting) 16.

7.3 Ideas for Projects with Music

- Make up your own tune.

- Make a musical instrument. Change the pitch of the sound played based on the readings from the accelerometer.

Radio

Your micro:bit has a radio chip that can be used to transmit and receive messages.
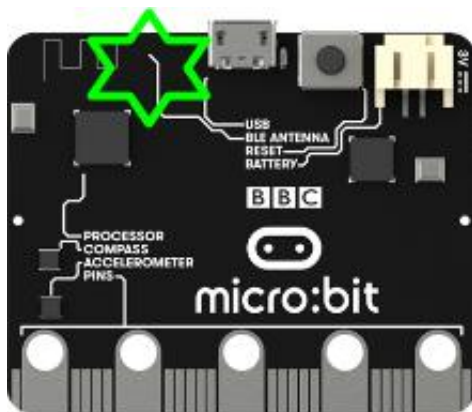
8.1 Basic Functions

8.1.1 Getting ready

Before you can use the radio you must remember to `import` the radio library and to turn the radio on on. Once the radio is on it will hear the messages from any other micro:bit that is within range:

```python
from microbit import *
import radio

radio.on()                          # Switch the radio on.
```

Setting a channel number

If you only want share messages within a group of devices then each micro:bit in the group must be configured to share the same channel number. The channel number must be a number between `0` and `100`. You can do that like this:

```
radio.config(channel=19)          # Set the channel number to 19
```

It is important to do this if you are in a room with other people using their micro:bits because otherwise your micro:bit will overhear all the messages nearby and that is not what you want generally.

Setting the power level

Finally, you should set the power level for the radio, by default, your micro:bit will be transmitting on power level 0 which means that your messages might not get transmitted very far. The power level can be a value between 0 and 7:

```
radio.config(power=7)    # Set the power level to 7
```

8.1.2 Sending and receiving a message

Now you are ready to send or receive a message. You can send a string which is up to 250 characters in length in the message. Here is an example:

```
my_message = "Be nice to yu turkeys dis christmas, Cos' turkeys just wanna hav fun,
↪ Turkeys are cool, turkeys are wicked, An every turkey has a Mum."

radio.send(my_message)
```

Receiving a message is similar in nature, just use:

```
message_received = radio.receive()
```

8.1.3 Putting it together

Your micro:bit is smart, it can send and receive messages in quick succession. Just tell the micro:bit to keep checking for messages or sending them like this:

```python
from microbit import *
import radio

radio.on()
radio.config(channel=19)          # Choose your own channel number
radio.config(power=7)             # Turn the signal up to full strength

my_message = "Be nice to yu turkeys dis christmas, Cos' turkeys just wanna hav fun,
↪ Turkeys are cool, turkeys are wicked, An every turkey has a Mum."

# Event loop.
while True:
        radio.send(my_message)
        incoming = radio.receive()
        if incoming is not None:
            display.show(incoming)
            print(incoming)
        sleep(500)
```

If you print the incoming message, you will see that sometimes it contains the value None. That is because sometimes the micro:bit checks for a message but nothing has arrived. We can ignore these non-events by checking whether incoming equals None and ignoring it if that is the case.

8.2 Ideas for Projects with the Radio

- Send a message every time button `A` is pressed.

- You will need a pair of micro:bits. Program one micro:bit to receive messages and print the message received using the `print()` method. eave this micro:bit plugged into your computer with a USB cable. Program the other micro:bit to send accelerometer readings or the temperature readings in messages every second. Unplug this micro:bit and use a battery pack to power it. Congratulations! you have created a data logger!

Control Structures

Sometimes you need to control what happens in your program, do you need to repeat something over and over again? Loops will help you there. Perhaps you only want something to happen if a particular event has occurred? Take a look at conditions to help you there. Of course, in any program you will likely use all of these to make your micro:bit do what you want and that's where things get interesting.

Sequence

Loop
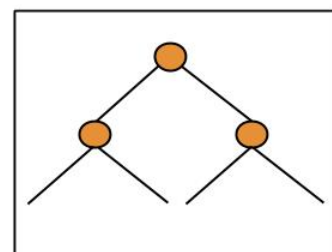
Selection
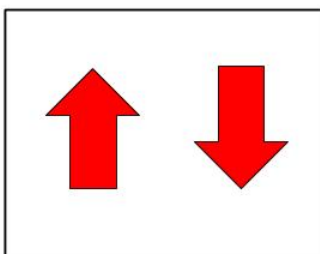
Input / Output
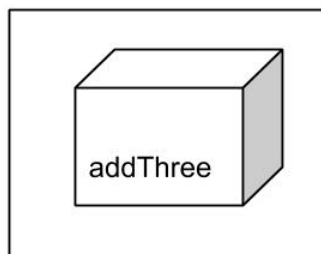
Functions

You have already seen how to write instructions or statements sequentially and generated input and output. In the remainder of this chapter we will look at loops, selection and functions.

9.1 Loops

There are two types of loops: `for` loops let you count the number of times you do something and `while` loops which let you perform an action until a condition you've specified is no longer happening.

9.1.1 For loops

There are times when you want to do an action a specific number of times. For example: perhaps you'd like to turn on the LEDs on two sides of the display to make a track. You can use a `for` loop to count for you like this:

```python
from microbit import *

for i in range (5):
    display.set_pixel(0,i,9)     # set the pixel in column 0, row i to 9
    display.set_pixel(4,i,9)     # set the pixel in column 4, row i to 9
```

Here is another example. You could use a `for loop` to set all the LEDs on one at a time:

```python
from microbit import *

display.clear()
for x in range(0, 5):
    for y in range(0, 5):
        display.set_pixel(x,y,9)
```

The `for loop` lets you execute a loop a specific number of times using a counter. The outer loop:

```python
for x in range(0,5):
```

will execute the loop five times substituting `x` for consecutive values in the range `0` to `4` each time. The loop will stop before it reaches 5, the final value in the range.

The inner loop:

```python
for y in range(0,5):
```

will execute the loop five times substituting `y` for consecutive values in the range `0` to `4` each time. Again, the loop will stop before it reaches the final value in the range.

9.1.2 While loops

One of the most common things you might want to do with a `while` loop is to do something forever, that is until the micro:bit is turned off or reset. Maybe you have programmed your micro:bit with your favourite game or perhaps it is collecting temperature data in the corner of a classroom. Here is an example of some code to repeat forever:

```python
from microbit import *

while True:
    display.scroll("Hello UCL)
```

This code will repeatedly display the message `Hello UCL`. You will likely have at least one `while True:` loop in your program to keep the micro:bit going.

But what if you want to do an action only whilst something is happening? Perhaps you would like to display an image if the temperature on the micro:bit goes below a certain value so you'll need to test the temperature:

```
from microbit import *

while temperature() < 18:
    display.scroll(Image.SAD)
    sleep(1000)

display.clear()
```

9.2 Selection

Choices, choices, choices. Sometimes you want to trigger an action if something specific has happened. We can use an `if ... else` statement for that. In this example we display a flashing heart if button `A` is pressed and a happy face if button `B` is pressed. If no buttons are pressed then we display a ghost:

```
from microbit import *
import love

while True:
    if button_a.is_pressed():
        love.badaboom()

    elif button_b.is_pressed():
        display.show(Image.HAPPY)

    else:
        display.show(Image.GHOST)

    sleep(100)
```

Notice how we have shortened `else if` to become `elif`, you don't have to do this if you don't want to, it just saves typing.

9.3 Functions

Functions and methods are used in programming to 'parcel up' useful snippets of code and use them whenever we want. You have likely already used both functions and methods without us needing to talk about it. In this tutorial we will not discuss methods further but we will explain how to use functions and how to write your own.

9.3.1 Using Functions

A great thing about functions is that we can use them in more than one program if we want to. In the same way we can use functions that other people have written too. In python, useful functions can be bundled up into modules (although you don't have to do this), the random module is a good example. To use functions in the random module we must first *import* the module. Once we've done that, we can use any of the functions in that module. Here are two examples of functions in the random module that you might . nt to use.

Random number in a range

Most of time, we will want to generate a random integer in a given range. The `random.randint()` function will allow us to do that:

```
from microbit import *
import random

display.show(str(random.randint(1, 6)))
```

In the code above, a random number between 1 and 5 will be generated - the upper bound, 6 in this case, is never included.

Random choice

In this code snippet, the function `random.choice` will check how many elements are in the names list, generate a random integer in the range 0 to the list length and return the list element for the random integer:

```python
from microbit import *
import random

names = ["Mary", "Yolanda", "Damien", "Alia", "Kushal", "Mei Xiu", "Zoltan" ]

display.scroll(random.choice(names))
```

9.3.2 Writing your own Functions

Writing your own functions can help you to organise your code and keep it neat and tidy. Here is an example of a simple function that prints out a message:

```python
def showGreeting():
        print("Hello Friend!")
```

To use the function we've just written we can call it like this:

```python
showGreeting()
```

That's not a very interesting function is it? We can make functions more powerful by using *parameters* and *return values*. If we think of a function like a black box then a parameter is an input value and a return value is what we will get out of the other end. Let's say we wanted to write a small program that will greet some friends with a message containing their name and age. Our program might look like this:

```python
from microbit import *

def printBirthday(name, age):
    return "Happy Birthday " + name + ", you are " + str(age) + " years old"

display.scroll(printBirthday("Tabitha", 8))
display.scroll(printBirthday("Henry", 9))
display.scroll(printBirthday("Maria", 11))
```

The function `printBirthday` composes the birthday message for us and returns a string. We have used the python function `str()` to turn `age`, which is a number, into a string. You don't have to use functions or return values in your functions unless you want to.

Data Types

We will use values of different types in our micro:bit programs, for example: we could capture acceleration values from the accelerometer. Alternatively, we might want to count the number of button presses the user has made or to show a message to the user telling them the temperature of the room. In order to do these things we need to be able to describe the data we want to use. Python, and most other programming languages, recognise several data types including:

- Integers - these are whole numbers.

- Floats - these are numbers that contain decimal points or for fractions.

- Strings - these can contain a combination of any characters that we want to treat as text such as letters, numbers and symbols.

- Boolean - used for True and False values.

In a simple program we might use all of these. Here are the data types we could use for a program storing information about our favourite micro:bit games:

Title	Rating	TimesViewed	Favourite
Zombie Attack	9.5	83	True
True Love	8.0	5	True
Mission: Pluto	2.5	1	False

Fig. 10.1: Image from: <http://www.bbc.co.uk/education/guides/zwmbgk7/revision/3>

10.1 Variables

A variable can be thought of as a box that the computer can use to store a value. The value held in that box can change or 'vary'. All variables are made up of three parts: a name, a type and a value. In the figure below there are three variables of different types:
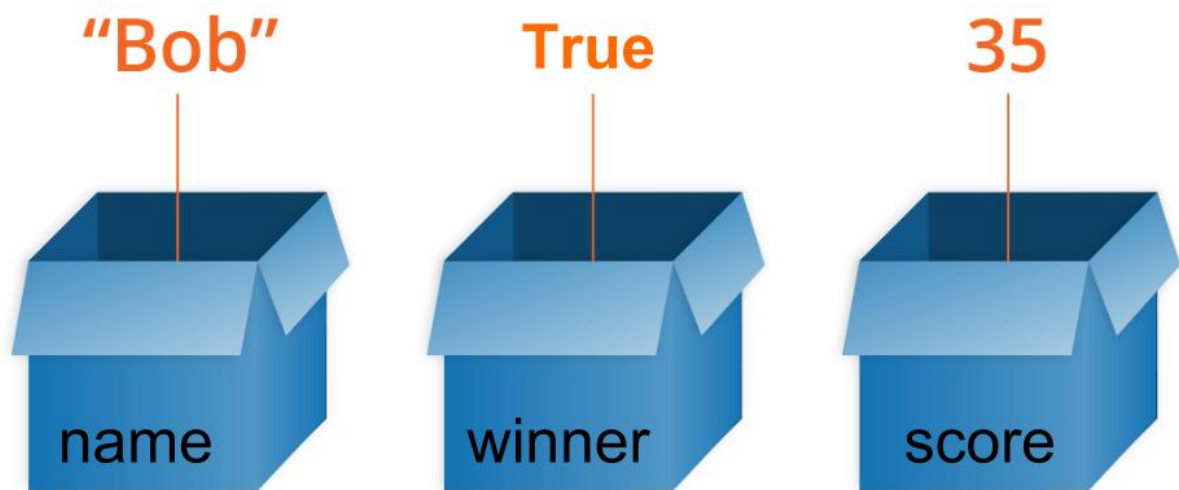
Fig. 10.2: Image from: <https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/Variables>

The variable `name` contains the string `Bob`, the variable `winner` contains the value `True` and the variable `score` contains the value `35`.

In Python we must give the variables we want to use a name and once we have done that we can start to use them, assigning and manipulating values:

```python
from microbit import *

myCount = 0

while True:
    if button_a.was_pressed():
        myCount = myCount + 1
    sleep(2000)
    print("Number of presses: " + str(count))
```

Here we have used the variable `myCount` to count the number of button presses for button `A`. Can you tell what else this snippet of code does?

10.2 Operations

10.2.1 Numbers

You can use numeric values with the basic arithmetic operators: `+, -, *, /` in the same way as you would with a calculator. Let's look at an example using arithmetic operators. Imagine that you want to convert the temperature you read from the microbit in Celsius to Fahrenheit, you could use code like this:

```python
celsiusTemp = temperature()
fahrenheitTemp = celsiusTemp * 9 / 5 + 32
```

The special operator `%`, called `mod` is used to calculate the remainder when one value is divided by another. For example: maybe you'd like to know whether a number is odd or even, you could try dividing it by 2, if it is even then there will be no remainder:

```python
theNumber = 3
if theNumber % 2 == 1:
    print("The number is odd")
else:
    print("The number is even")
```

If the remainder is equal to 1 then this program will print the message "The number is odd", otherwise. the program will print the message "The number is even". You might have written this program in a different way, This shows that people think about problems in different ways and not two programs are likely to be the same.

10.2.2 Strings

The main thing to note about strings is that you can add them together, or concatenate them, with a + symbol. The code:

```python
name = "Hayley"

message = "Well done " + name + ". You are a winner!"
```

Will concatenate the items on the right hand side of the = and put the result in the variable called `message`.

You cannot join numbers and strings together; you must first convert the number to a string using the `str()` function if you want to do that:

```
x = temperature
if temperature < 6:
    display.scroll("Cold" + str(temperature))
```

10.3 Comparisons
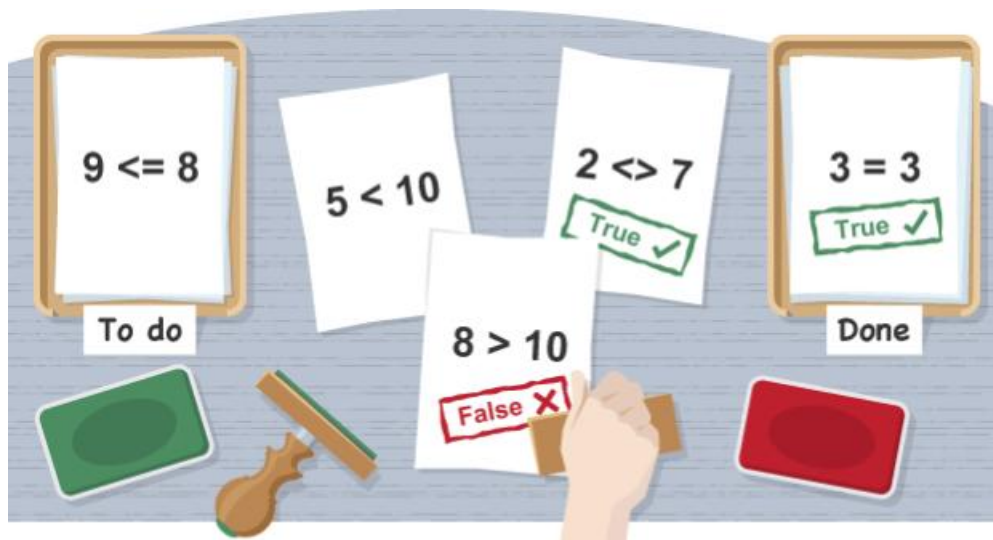
Fig. 10.3: Image from <http://www.bbc.co.uk/education/guides/zy9thyc/revision>

Often in programming we want to compare one value to another, a kind of test. We use these tests or comparisons in selection or loops. Here are some examples of comaparisons written in English:

```
score is greater than 100
name equals "Harry"
x acceleration is not equal to 0
```

Python has a set of comparison operators that allow us to write comparisons easily:

Comparison Operator	Meaning
==	Equal to
<, <=	Less than, less than or equal to
>, >=	Greater than, greater than or equal to
!=	not equal to

Rewriting the comparisons above in Python would be:

```
score > 100
name ==  "Harry"
acceleration  != 0
```

10.3.1 Using Comparisons

The result of a comparison is either `True` or `False`. `True` and `False` are special values known as **Boolean values** and we can use can use them to determine what our programs will do. You may have already used some examples that do this. In this example, the micro:bit will show an arrow pointing in the direction of the tilt in the x axis:

```python
from microbit import *

while True:

    x_acceleration = accelerometer.get_x()

    if x_acceleration > 100:
        display.show(Image.ARROW_E)

    if  x_acceleration < 100:
        display.show(Image.ARROW_W)
```

10.4 Lists

Fig. 10.4: Image from <http://www.bbc.co.uk/education/guides/zy9thyc/revision>

Lists are useful for storing several values together. Let's say we want to store a player's scores, we could use a list like the one pictured above. The list has one box for each value. The cells or boxes are known as *elelments*.

Let's see how to use a list in Python. To create a list we can tell Python the name of the list and what it will contain:

```python
from microbit import *

highScores = [25, 20, 10, 15, 30]        # Create a list and store some values in
→it.
print(highScores[0])                     # Print 25
print(highScores[3])                     # Print 15
```

Finding the value of one of the elements in a list is easy as long as you remember that Python counts the elements from '0'. In our `highScores` list above, `highScores[0]` is 25 and `highScores[3]` is 15.

Not surprisingly, Python has some features to help us do things with lists. The code snippet below will go through the array elements one by one so that we can sum them and calculate the average high score:

```python
print("Average High Score: ")

total = 0
for score in highScores:                 # For each element ...
        total = total + score

average = total / len(highScores)  # Use the len() function here to find the
→length of the array
print(average)
```

10.4.1 Add to a List

There will be times when we don't know how large to make an array in advance or what the values in the list are going to be. You might want to fill a list with temperature readings or accelerometer values, for example. This code illustrates how you can do that:

```python
from microbit import *

recordedTemperature = []                # Create an empty list
for i in range(100):                    # Add 100 temperature values
        recordedTemperature.append(temperature())
        sleep(1000)
```

The `for` loop is executed 100 times and `i` will have values from 0 to 99. This will measure the temperature every second for 100 seconds and append the value on to the end of the list.

10.4.2 Delete from a List

There are two ways to delete elements from lists that are helpful, you might want to delete an element with a particular value from a list:

```python
highScores.delete(24)
```

This will delete the first element with the value 24. Alternatively, you might want to delete an element at a specific position, if you know it:

```python
highScores.pop(3)
```

This will delete or 'pop' the element at the given position in the list. Note that:

```python
highScores.pop()
```

will delete the last element in the list.

Bop-it

Total points possible	Uses
10	LED display, buttons

11.1 Description

In this challenge you will make a reaction game similar to bop-it. The first bop-it game, pictured below, was released in the 1960s and since then there have been many variants which test your reactions with light, sound and action.

Fig. 11.1: The original bop-it design, source: Wikipedia

In the version of the game you create, the player must press the correct button, A or B, in under 1 second. If the player presses the wrong button, the game ends. If they press the correct button, they get a point and can play again.

11.2 Basic Game

Collect points for these stages:

Tasks	Points
Display a welcome message.	1
Create loop for the game which repeats every second. Prompt the player to press either button `A` or button `B`. Hint: use `random.randint(0,1)`.	1
Test whether the user has pressed the correct button. If the user has pressed the correct button, display a suitable image or message.	1
If the user has pressed the wrong button, finish the game.	2
If the player's answer was correct, add 1 to the player's score.	1
At the end of the game, display the player's score.	1

11.3 Extra points

Tasks	Points
If the player succeeds, make the time interval for the next round shorter.	1

Consonant or Vowel?

Total points possible	Uses
10	LED display, buttons

12.1 Description

In this challenge you will make a reaction game called Consonant or Vowel. The micro:bit must show the player a letter which is either a consonant or a vowel. The player must press button `A` if it is a consonant and button `B` if it is a vowel. They must choose an answer in 1 second. If the player presses the wrong button, the micro:bit displays a suitable image or message and the game ends. If they press the correct button, the micro:bit displays a message or image and the player gets a point and can play again.

12.2 Basic Game

Collect points for these stages:

Tasks	Points
Display a welcome messge.	1
Create loop for the game which repeats every second.	1
Create a list of consonants in your program and a list of vowels. Hint: `vowels = ['A', 'E', 'I', 'O', 'U']`	1
Randomly select either consonants or vowels. Hint: use `random.randint(0,1)`. Randomly select a letter from the list. Display the letter.	1
Test whether the user has pressed the correct button. If the user has pressed the correct button, display a suitable image or message.	1
If the user has pressed the wrong button, finish the game.	2
If the player's answer was correct, add 1 to the player's score.	1
At the end of the game, display the player's score.	1

12.3 Extra points

Tasks	Points
If the player succeeds, make the time interval for the next round shorter.	1

Obstacles

Total points possible	Uses
10	LED display, buttons

13.1 Description

In this challenge the player is driving down a road or race track. There are obstacles in the road and the player must dodge them using the `A` and `B` buttons on the micro:bit. If the player hits an obstacle the game ends.

13.2 Basic Game

Collect points for these stages:

Tasks	Points
Display a welcome message.	1
Create loop for the game which repeats every second.	1
Create a player sprite in any column in the bottom row. Display the pixel at maximum brightness.	
Move the player sprite 1 pixel upwards each time the loop repeats. If the player sprite is in row 0, move the player sprite back to row 4. Hint: store the position of the player in a list: `player = [0,0]`.	1
Move the player sprite to the right if button `B` is pressed and left if button `A` is pressed.	1
Create an obstacle sprite in any column in the top row. Display the pixel at a low brightness. The obstacle should move down 1 pixel every time the game loop repeats. Hint: store the position of the obstacle in a list `obstacle = [0,0]`.	2
Now when you create the obstacle sprite, place it in a random column on the display. Hint: use `random.randint(0,4)`.	1
Every time the loop repeats, check whether the player has collided with the obstacle. If there is a collision, the game ends.	2

13.3 Extra points

Tasks	Points
At the start of the game, give the player 3 lives. Subtract a life every time there is a collision. End game when the player has no lives left.	1

Catch the Eggs

Total points possible	Uses
10	LED display, buttons

14.1 Description

In this challenge you will test the player's ability to catch an egg in a basket. Imagine there is a chocolate egg dropping from the sky. The player must catch the egg before it hits the ground. The player can move either right or left by using the `A` and `B` buttons on the micro:bit. If the egg hits the floor the game ends.

14.2 Basic Game

Collect points for these stages:

Tasks	Points
Display a welcome message.	1
Create loop for the game which repeats every second.	1
Create a player sprite in column 0, row 3.	1
Move the player sprite to the right if button `B` is pressed and left if button `A` is pressed. Hint: store the position of the player in a list `player = [0,0]`.	1
Create an egg sprite. The egg will start at the top of the display and move down 1 pixel every time the game loop repeats. Hint: store the position of the egg in a list `egg = [0,0]`.	1
Create the egg sprite in a random column on the display. Hint: use `random.randint(0,4)`.	1
Every time the loop repeats, check whether the player has collided with the egg. If there is a collision, the player has caught the egg and the game ends. Display an image or message to congratulate the player.	2
If the egg drops to the bottom of the display and there has been no collision then set the position of the egg to a random column at the top of the display.	1

14.3 Extra points

Tasks	Points
Let the player continue catching eggs until they drop one. When the game ends, display the player's score.	1

Where's the Cheese?

Total points possible	Uses
10	LED display, buttons

15.1 Description

In this challenge the player is a mouse and the mouse must eat the cheese before the time is up. The player can move around the display by tilting the micro:bit to the left or right, up or down. When the player lands on the cheese, call function: `love.badaboom()` and see what happens. The player must eat the cheese within a time limit or the game ends.

15.2 Basic Game

Collect points for these stages:

Tasks	Points
Display a welcome messge.	1
Create loop for the game which repeats every second.	1
Create a player sprite. Display the player sprite in top left hand corner of the display. Hint: store the position of the player in a list `player = [0,0]`.	1
Move the player sprite to the right if the micro:bit is tilted right and left if the micro:bit is tilted left. Hint: Use the values from `accelerometer.get_x()` to determine the tilt of the board.	2
Move the player sprite upwards if the micro:bit is tipped up. Move the player sprite downwards if the micro:bit tipped downward. Hint: use *accelerometer.get_y()*' to determine the tilt of the board.	2
Create a cheese sprite. Assign a random position to the cheese sprite. Hint: store the position of the cheese in a list `cheese = [0,0]`.	1
When the player moves, check whether the mouse is in the same position as the cheese. If it is then the player has won. call function `love.badaboom()` and see what happens.	1
Create a timer, if the mouse does not find the cheese before the timer expires, the game ends. Hint: at the start of the game get the start time by using function `microbit.running_time()`.	1

15.3 Extra points

Tasks	Points
Repeat the game when the mouse reaches the cheese. Reduce the amount of time the player has to find the cheese.	1

Caesar Cipher - Part I

Total points possible	Uses
10	LED display, buttons

16.1 Description

Humans have been interested in hiding messages for as long as they have been able to communicate by writing things down. One of the earliest ciphers is known as the Caesar cipher, named after Julius Caesar, and was used by the Roman emporer to communicate with troops on the battlefield. Using the Caesar cipher you encrypt all the letters in a message by shifting the alphabet a number of places. The figure below shows how to encrypt a message with a shift of 3 letters:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
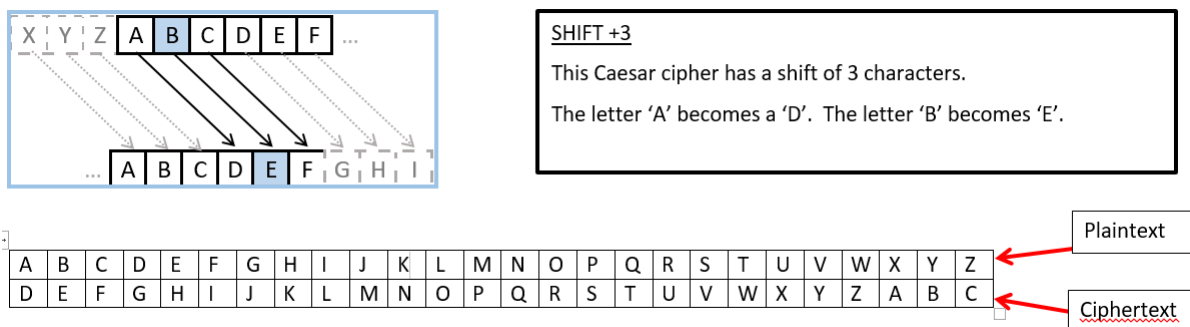
Your goal is to turn your micro:bit into a machine that can **encode** messages using the Caesar cipher. We call the message to be encrypted *plain text* and the encrypted message *cipher text*.

There is a trick you can use to encrypt, or shift the message. The trick relies on the fact that your micro:bit sees the letters of the alphabet as numbers. You can translate a letter to a number, and back again using the python functions `ord()` and `chr()`.

Let's say you want to shift each character by 4 places. Try using this code to turn the character into a number and add 4:

```
ascii_char = ord(plaintext_char) + 4
```

In English this means: translate `plaintext_char` into a number using the `ord()` function and add 4, the number of characters we want to shift.

But hold on, there is one more thing that we need to do. If you look at the picture above, you will see that we need to wrap around going from Z back to A. To do this we need to subtract 26 (the number of letters in the alphabet) if we have gone past the letter 'Z':

```
ascii_char = ord(plaintext_char) + 4
if ascii_char > ord('Z')
        ascii_char = ascii_char - 26
encrypted_char = chr(ascii_char)
```

Try this out, experiment using the REPL.

16.2 Basic Challenge

Collect points for these stages:

Tasks	Points
Display a welcome messge.	1
You need to find out from the player how many places to shift the message. One way to do this is to shift the message by the number of button presses the user makes. Keep a count of how many times button A is pressed. The player can press button B to indicate they have finished.	2
In your program you should store the message you want to translate in a string like this: `message = 'KEEP THIS A SECRET`.	1
Now display the message a character at a time using a `for` loop. Hint: to get each character in the message use `for c in message:`.	1
Use the `ord()` function to translate each character into an numeric value and add the number of characters you want to shift.	1
Make sure you have wrapped the result around. Hint: Check whether the shifted value is greater than the numeric value for `Z`.	1
Use `chr()` to translate each number back into a character Hint: Don't encrypt the spaces.	2
Display the encrypted text on the micro:bit and print encrypted text in the REPL using the `print()` function.	1

Caesar Cipher - Part II

Total points possible	Uses
10	LED display

17.1 Description

This challenge is about decrypting messages that have been encoded using the Caesar cipher. If you haven't tried encrypting messages with the Caesar cipher then maybe you could attempt that first.

Using the Caesar cipher you can encrypt or decrypt all the letters in a message by shifting the alphabet a number of places. The figure below shows how to encrypt a message with a shift of 3 letters:
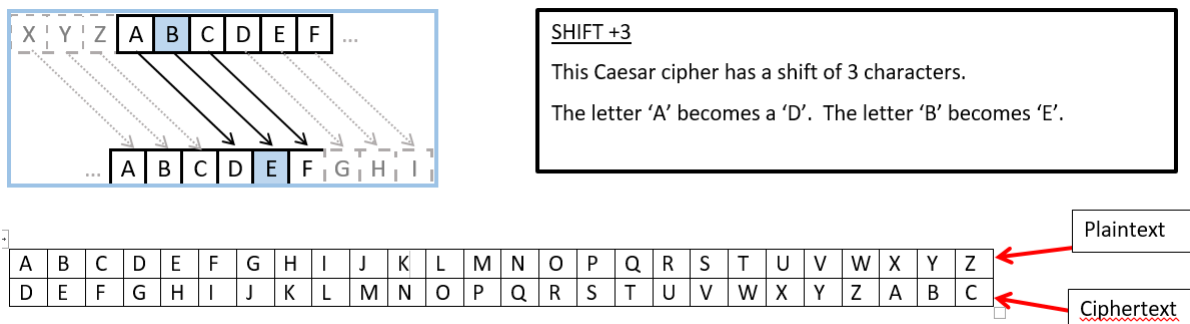
Your goal is to turn your micro:bit into a machine that can **decode** messages that have been encrypted using the Caesar cipher. We call the encrypted message *cipher text* and the decrypted message *plain text*.

There is a trick you can use to decrypt, or shift the message. The trick relies on the fact that your micro:bit sees the letters of the alphabet as special numbers known as *ascii values*. You can translate a letter to an ascii number, and back again using the python functions `ord()` and `chr()`.

Here is an example: Let's say you have an encrypted message which has been shifted by 4 places. Try using this code to turn the character into a number and subtract 4:

```
ascii_char = ord(encrypted_char) - 4
```

In English this means: translate `encrypted_char` into an ascii number using the `ord()` function and subtract 4, the number of characters we want to shift.

But hold on, there is one more thing that we need to do. If you look at the picture above, you will see that we need to wrap around going from A back to Z. To do this we need to add 26 (the number of letters in the alphabet) if we have gone past A:

```
ascii_char = ord(plaintext_char) - 4
if ascii_char > ord('Z')
        ascii_char = ascii_char + 26
encrypted_char = chr(ascii_char)
```

Try this out, experiment using the REPL.

17.2 Basic Challenge

Collect points for these stages:

Tasks	Points
Display a welcome message.	1
You don't need a computer for this part! Use a cipher wheel (ask your teacher) to encrypt a message about 10 characters long. **Important:** remember how many characters you shifted the message.	2
In your program you should store the encrypted message in a string like this: `encrypted_message = 'QKKV OZ G YKIXKZ`.	1
Now display the message a character at a time using a `for` loop. Hint: to get each character in the message use `for c in message:`.	1
Use the `ord()` function to translate each character into an ascii value and subtract the number of characters that the ciphertext was shifted.	1
Make sure you have wrapped the result around. Hint: Check whether the ascii value is less than 'A'.	1
Use `chr()` to translate each ascii character in the message to plain text. Hint: Don't decrypt the spaces.	2
Display the plain text on the micro:bit and print the plain text in the REPL using the `print()` function.	1

Substitution Cipher

Total points possible	Uses
10	LED display

18.1 Description

The substitution cipher is deceptively easy. Messages are encrypted using a key which is created in advance. You make the key by jumbling up the alphabet like this:

```
A   B   C   D   E   F   G   H   I   J   K   L   M   N   O   P   Q   R   S   T   U   V   W   X   Y   Z
↕   ↕   ↕   ↕   ↕   ↕   ↕   ↕   ↕   ↕   ↕   ↕   ↕   ↕   ↕   ↕   ↕   ↕   ↕   ↕   ↕   ↕   ↕   ↕   ↕   ↕
V   J   Z   B   G   N   F   E   P   L   I   T   M   X   D   W   K   Q   U   C   R   Y   A   H   S   O
```

Your goal is to turn your micro:bit into a machine that can encode messages using a substitution cipher. We call the message to be encrypted *plain text* and the encrypted message *cipher text*. You will need to store the alphabet with the substition cipher in your program. You can use a python *dictionary* to do this. A python dictionary for the substitution cipher above looks like this:

```
cipher_key = { 'A':'V', 'B':'J', 'C':'Z', 'D':'B', 'E':'G', 'F':'N', 'G':'F', 'H':
→'E', 'I':'P', 'J':'L', 'K':'I','L':'T','M':'M','N':'X','O':'D','P':'W','Q':'K','R
→':'Q','S':'U','T':'C','U':'R','V':'Y','W':'A','X':'H','Y':'S','Z':'O'}
```

In English, this means: the character 'A' should be substituted with the character 'V'; the character 'B' should be substituted with the character 'J' and so on. You can print a dictionary using `print(cipher_key)`. Try this out, experiment using the REPL.

18.2 Basic Task

Collect points for these stages:

Tasks	Points
Display a welcome message.	1
Create a dictionary containing the alphabet and the corresponding cipher text. Print the dictionary using the REPL.	2
In this version of your program you should store the message to encode in a string like this: `message = 'KEEP THIS A SECRET`.	1
Now display the message a character at a time using a `for` loop. Hint: to get each character in the message use `for c in message:`.	1
Use the dictionary you created to translate each character in the message to a corresponding encrypted character. Hint: `cipher_key['A']` will give you the cipher character corresponding to the letter 'A' in your dictionary.	4
Display the encrypted text on the micro:bit display and print encrypted text in the REPL using the `print()` function.	1

Decrypting a Substitution Cipher

Total points possible	Uses
10	LED display

19.1 Description

The substitution cipher is deceptively easy. Messages are encrypted using a key which is created in advance. You make the key by jumbling up the alphabet like this:

```
A   B   C   D   E   F   G   H   I   J   K   L   M   N   O   P   Q   R   S   T   U   V   W   X   Y   Z
↕   ↕   ↕   ↕   ↕   ↕   ↕   ↕   ↕   ↕   ↕   ↕   ↕   ↕   ↕   ↕   ↕   ↕   ↕   ↕   ↕   ↕   ↕   ↕   ↕   ↕
V   J   Z   B   G   N   F   E   P   L   I   T   M   X   D   W   K   Q   U   C   R   Y   A   H   S   O
```

Your goal is to turn your micro:bit into a machine that can decode messages using a substitution cipher. We call the message that has been encrypted *cipher text* and the decrypted message *plain text*. You will need to store the alphabet with the substition cipher in your program. You can use a python *dictionary* to do this. A python dictionary for the substitution cipher above looks like this:

```
cipher_key = { 'A':'V', 'B':'J', 'C':'Z', 'D':'B', 'E':'G', 'F':'N', 'G':'F', 'H':
→'E', 'I':'P', 'J':'L', 'K':'I','L':'T','M':'M','N':'X','O':'D','P':'W','Q':'K','R
→':'Q','S':'U','T':'C','U':'R','V':'Y','W':'A','X':'H','Y':'S','Z':'O'}
```

In English, this means: the character 'A' should be substituted with the character 'V'; the character 'B' should be substituted with the character 'J' and so on. You can print a dictionary using `print(cipher_key)`. Try this out, experiment using the REPL.

19.2 Basic Task

Collect points for these stages:

Tasks	Points
Display a welcome message.	1
Create a dictionary containing the alphabet and the corresponding cipher text. Print the dictionary using the REPL.	2
In this version of your program you should store the message to be decrypted in a string like this: `encrypted_message = 'IGGW PC V UGZQGC`.	1
Now display the message a character at a time using a `for` loop. Hint: to get each character in the message use `for c in encrypted_message:`.	1
Use the dictionary you created to translate each character in the message to a corresponding plain text character. Hint: `for key, value in cipher_key.items():` will go through the dictionary, item by item giving you the `key` and `value` pairs. So, the first values returned will be `key` of `A` and a `value` of `V`. You will have to search through the whole dictionary until you find the value for the ciphertext character.	4
Display the decrypted text on the micro:bit display and print the decrypted text in the REPL using the `print()` function.	1

Morse Code

Total points possible	Uses
10	LED display, speaker

20.1 Description

Morse code was invented in 1836 by a group of people including the American artist Samuel F. B. Morse. Using Morse code a message is represented as a series of electrical pulses which can be sent along wires to an electromagnet at the receiving end of the system. The symbols used for each letter are shown in the figure below.

A	•—	J	•———	S	•••	1	•————
B	—•••	K	—•—	T	—	2	••———
C	—•—•	L	•—••	U	••—	3	•••——
D	—••	M	——	V	•••—	4	••••—
E	•	N	—•	W	•——	5	•••••
F	••—•	O	———	X	—••—	6	—••••
G	——•	P	•——•	Y	—•——	7	——•••
H	••••	Q	——•—	Z	——••	8	———••
I	••	R	•—•	0	—————	9	————•

Source: raspberrypi.org

Of course, you aren't limited to electrical pulses, you can transmit a Morse code message using light or even sound. A Morse code message sent over electrical wires is known as a telegram, a message is translated to Morse code by an operator at the sending end using a a telegraph key like the one pictured here.

Telegraph key, source: Wikipedia

The message is converted back to normal text by another operator at the receiving end.

Your goal is to turn your micro:bit into a machine that can encode messages using Morse code. We will call the message to be converted *plain text*. You will need to store the alphabet with the morse code in your program. You can use a python *dictionary* to do this. Here is part of a python dictionary for morse code:

```
morse_code = { 'A':'.-', 'B':'-...', 'C':'---.', 'D':'-..', 'E':'.', 'F':'..-.', 'G
→':'..-.', 'H':'--.', ...   }
```

In English, this means: the character 'A' should be substituted with the string '.-'; the character 'B' should be substituted with the string '-...' and so on. You can print a dictionary using:

```
print(morse_code)
```

Try this out, experiment using the REPL.

20.2 Basic Task

Collect points for these stages:

Tasks	Points
Display a welcome message.	1
Create a dictionary containing the alphabet and the corresponding morse code. Print the dictionary using the REPL.	2
In this version of your program you should store the message to encode in a string like this: `message = 'KEEP THIS A SECRET`.	1
Now display the message a character at a time using a `for` loop. Hint: to get each character in the message use `for c in message:`.	1
Use the dictionary you created to translate each character in the message to a corresponding encrypted character. Hint: `morse_code['A']` will give you the morse code symbol corresponding to the letter 'A' in your dictionary.	3
Display the morse code on the micro:bit display and print the code in the REPL using the `print()` function.	1
Connect a speaker to the micro:bit and play a series of of beeps to represent the morse code message.	1

Sprit Level

Total points possible	Uses
10	Accelerometer, LED Display

21.1 Description

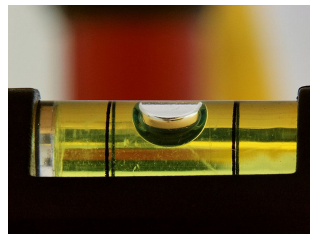In this project, we will use the accelerometer to make a spirit level.

Fig. 21.1: Image source: Wikipedia

A spirit level, like the one in the picture above, is used to tell whether a surface is flat. If you put a spirit level on a flat surface the bubble will rest in the middle of the tube. If the surface is tilted to the left or right, the bubble will also move to the left or right indicating visually that there is a slope. You can use accelerometer values to determine whether the micro:bit is on a flat surface or not and give the user a signal by displaying an arrow, for example, to indicate a tilt.

21.2 Basic Task

Collect points for these stages:

Tasks	Points
We are going to use the accelerometer values in the x axis. Write some code to print accelerometer values for the x axis.	2
Write down (on paper) the x axis values for the accelerometer when you tilt the board left, when you tilt the board to the right and when the board is held flat, face-up. We need to know the minimum and maximum values for the range.	2
If the micro:bit is tilted to the left, display an arrow pointing leftwards.	2
If the micro:bit is tilted to the right, display an arrow pointing leftwards.	2
If the micro:bit is flat, display a dot in the middle of the LED display.	2

Theremin

Total points possible	Uses
10	Accelerometer, speaker

22.1 Description

In this project, we will use the accelerometer to control the frequency of a tone.

The theremin is a weird and wonderful electronic instrument that requires no physical contact. Have a listen to a program about them at: http://www.bbc.co.uk/programmes/b0076nqv. The theremin was invented in 1920 by Léon Theremin, an early Russian electronic engineer. It is played by moving one's hands near two antennas – the first controls the volume of the output and the second the pitch. For those that are musical it is worth knowing that the Theremin inspired Robert Moog to invent the synthesiser, so, although it's a little-used instrument, it has had a powerful effect on the history of music.

Fig. 22.1: Image: Leon Theremin, source: Wikipedia

22.1.1 Aside: a quick look at lists

As part of this task, you will have to collect and store some values from the accelerometer in a list and to calculate the average. Here is some information about how to do that. A list is a data structure used in just about all programming languages. In our case, it's a numbered collection of accelerometer values. In essence it's a set of boxes into which we can put values – each box has a number, starting at 0 and going up. Say we needed to keep the last 30 values of the accelerometer, then we create a list and add accelerometer value to it every time that we go around the loop like this:

```
while True:

    x_acceleration = accelerometer.get_x()

    # Add to the list of readings
    readings.append(x_acceleration)

    # If readings contains more than 30 values then pop the first value from the␣
↪beginning
    if len(readings) > 30:
        readings.pop(0)
    sleep(1)
```

As you can see, if the array has more than 30 entries in it we will just delete the first entry, element number 0:

```
readings.pop(0)
```

22.2 Basic Task

Collect points for these stages:

Tasks	Points
We are going to use the accelerometer values in the x axis. Write some code to print accelerometer values for the x axis.	1
Write down (on paper) the x axis values for the accelerometer when you tilt the board left, when you tilt the board to the right and when the board is held flat, face-up. We need to know the minimum and maximum values for the range.	1
Collect 30 accelerometer readings in a list. Have a look at the notes above for some information about this. Print the list to the REPL.	2
Now let's try making some sound. Connect the speaker to the micro:bit using some crocodile clips. Import the music library `import music` and play some sounds using `music.pitch(frequency, time)` where frequency is between 50 and 4000 Hz and time is a value in milliseconds.	1
You need to translate the value of the accelerometer reading into a sound frequency value. Can you think of a way to do this? Experiment with your program, print values to the REPL.	1
Play a frequency that corresponds to the accelerometer value.	1
You will find that the sound wavers because the accelerometer values vary quickly. One way to address this is to calculate the average value and play a . frequency that corresponds to that. Calculate the average accerometer value and print it. Hint: There are two python functions that can help us with that: `sum(readings) / len(readings)` sum adds up all the elements of a list and `len` returns the length of a list so we have the total value of all the accelrometer readings divided by the number of readings.	2

Send a Message

Total points possible	Uses
10	Two Micro:bits, Battery pack with 2 AAA batteries, Radio, LED Display

23.1 Description

In this project, we will use the radio on the micro:bit to send messages from one microbit to another. You will display the number of messages sent between a pair of micro:bits on the LED display.

23.2 Basic Task

Collect points for these stages:

Tasks	Points
Read the radio tutorial sheet. Write code to transmit and receive messages. Print the message sent on the REPL.	2
Add some code so that the message is sent when button a is pressed.	1
Connect the battery pack to the second micro:bit. Upload the same code the second micro:bit.	2
Test the pair of micro:bits. You should be able to send and receive messages.	2
Now you should find a way to count the messages. You could add 1 to a counter every time you send a message,	1
You need to do something special to send and receive numbers - you need to translate the number to a string using the `str()` function before sending it in the message. At the receiving end you need to translate the string back into a number using the `int()` function.	2

Ping Pong

Total points possible	Uses
10	Two Micro:bits, Battery pack with 2 AAA batteries, Radio, LED Display

24.1 Description

In this project, we will use the radio on the micro:bit to send messages from one microbit to another. You will light up an LED when a message is received so that it looks like the people holding the micro:bits are playing ping pong.

24.2 Basic Task

Collect points for these stages:

Tasks	Points
Read the radio tutorial sheet. Write code to transmit and receive messages. Print the message sent on the REPL.	2
Add some code so that the message is sent when button A is pressed.	1
Connect the battery pack to the second micro:bit. Upload the same code the second micro:bit.	2
Test the pair of micro:bits. You should be able to send and receive messages.	2
When a message is received, light up an LED for a second.	1
Keep the score on each device. Count the number of messages received.	1
If button B is pressed, display the score.	1

Rock, Paper, Scissors

Total points possible	Uses
10	Two micro:bits, Battery pack with 2 AAA batteries, Radio, LED Display

25.1 Description

In this project, we will make a rock, paper, scissors game using use the radio on the micro:bit to send messages from one microbit to another.

25.2 Basic Task

Collect points for these stages:

Tasks	Points
Read the radio tutorial sheet. Write code to transmit and receive messages. Print the message sent on the REPL.	2
Choose which micro:bit will host the R,P,S game and which micro:bit will be used by the player. On the host micro:bit, display a welcome message.	1
Everyone should know how to play Rock, Paper, Scissors. Play the game without using a computer and make up your own instructions for the challenge.	7

Python Module Index

m

M