UCL BBC micro:bit MicroPython Tutorial

Rae Harbird, Stephen Hailes

Jul 30, 2019

Getting Started

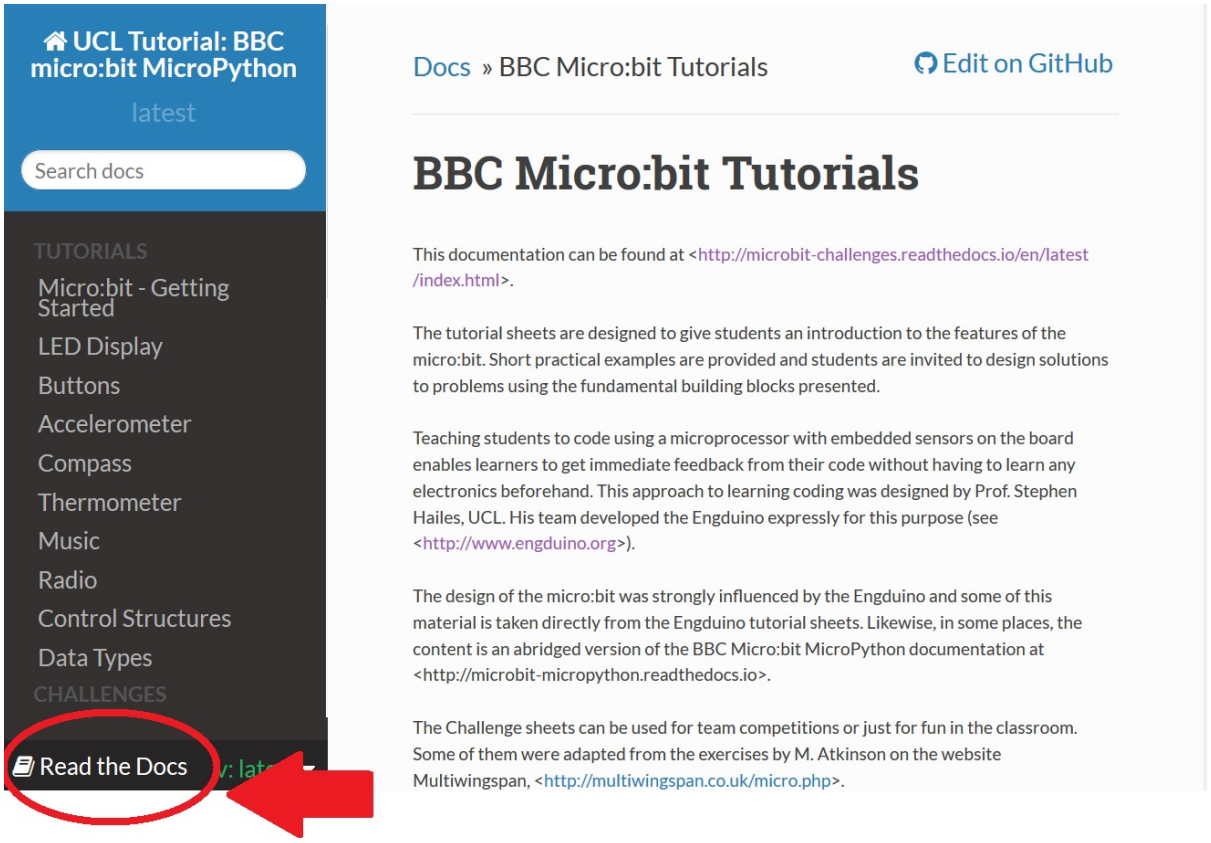
This documentation can be found at <http://microbit-challenges.readthedocs.io/en/latest/index.html>.

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.

CHAPTER 1

To You from Head of Department

Dear All,

Welcome to Computer Science at UCL.

The journey you are about to undertake will have a profound effect on the rest of your life — your time as a university undergraduate is unique and memorable and it is our task to work with you to ensure that you get the most out of it. Our aim is to make your experience with us challenging, exciting and meaningful: giving you a strong basis for the professional world or further study after you graduate.

Computer Science at UCL is focussed on building well-rounded expertise. Ours is an applied course based on strong theoretical foundations — this means that you will build things, but you will know how and why they should be created as they are. Over your time at UCL, you will work with academic supervisors and real clients and so you will need to learn to communicate and work in teams effectively. But, above all, you will be challenged to become an independent learner, capable of managing your own study; given the rate of change in technology, this is a (perhaps the) key skill for life after university.

Your journey starts here. Enclosed in this pack is a BBC micro:bit. This is a small computer with attached sensors that was designed to help students learn to program. Some of you may have seen this already, many will not have done. Regardless, we'd like to challenge you to do two things before you arrive, and to show us what you have achieved in the first week of your time at UCL. The two things are:

1. Work through some of the material you can find at http://microbit-challenges.readthedocs.io and use the examples and project ideas to write some simple Python programs for the micro:bit.

2. Surprise us. Do something creative with the micro:bit — we don't mind what it is, but we want it to be your own work.

There are no marks for this. And you may need to find and explore resources on the web other than those we have provided. Some of you may struggle. The point here is that you start your journey as independent learners: most of the things in life worth having require an investment of time and effort, and a UCL education is worth having. We will ask you to share what you have done with us in the first week of term, so please do remember to bring your micro:bit with you.

Please note: we do not expect you to buy additional hardware to enhance your micro:bit and we do not expect you to spend every waking hour of the next month coding. We are looking for creativity and a willingness to learn, not volume of code or expensive rigs.

I hope your time with us will be exciting and challenging. And, since I will be there in your first week and will then teach one of your first courses, I very much look forward to meeting you.

Steve Hailes
Head of Department

CHAPTER 2

Micro:bit - Getting Started

The BBC micro:bit is a programmable micro-computer - microcontroller - that can be used 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.

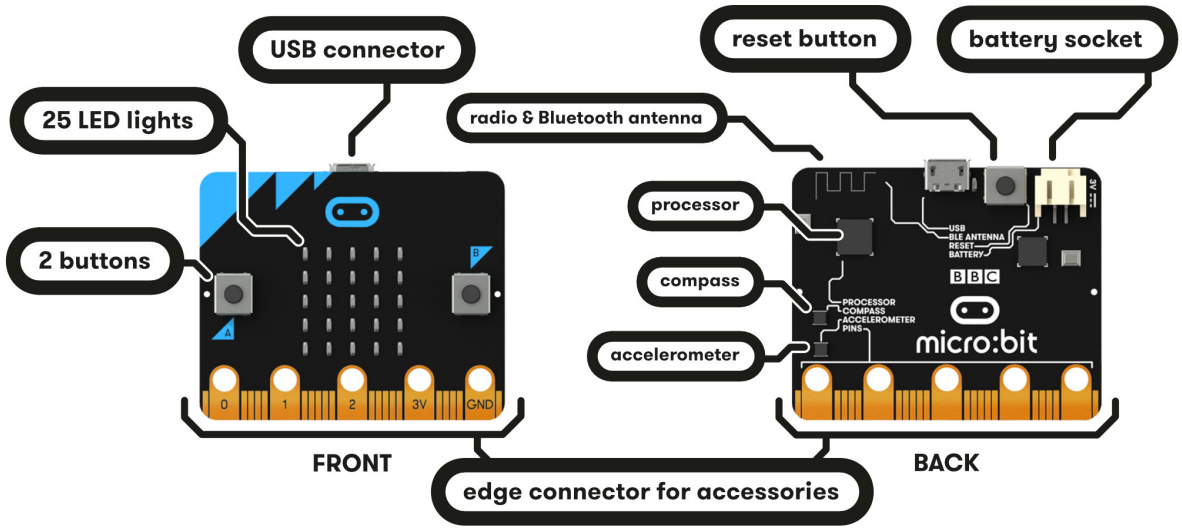


Fig. 1: Source: https://microbit.org/guide/features/

You can program micro:bit using several languages: MicroPython, C++ or JavaScript. This tutorial will focus on programming micro:bit using MicroPython, but if you already are familiar with Python, or you're looking for extra challenge, look at the section for Programming Micro:bit Using Other laguages. C/C++ might be useful in particular, as it's the main language used to program embedded devices.

MicroPython is a version of Python , that's designed to run on microcontrollers like micro:bit. Since their functionality is virtually the same (look here for difference in behaviour), we refer to the language used as Python in these tutorials. Programming in Python consists of writing a series of steps to be executed (it's an *imperative* language), as you will see later when writing your first program.

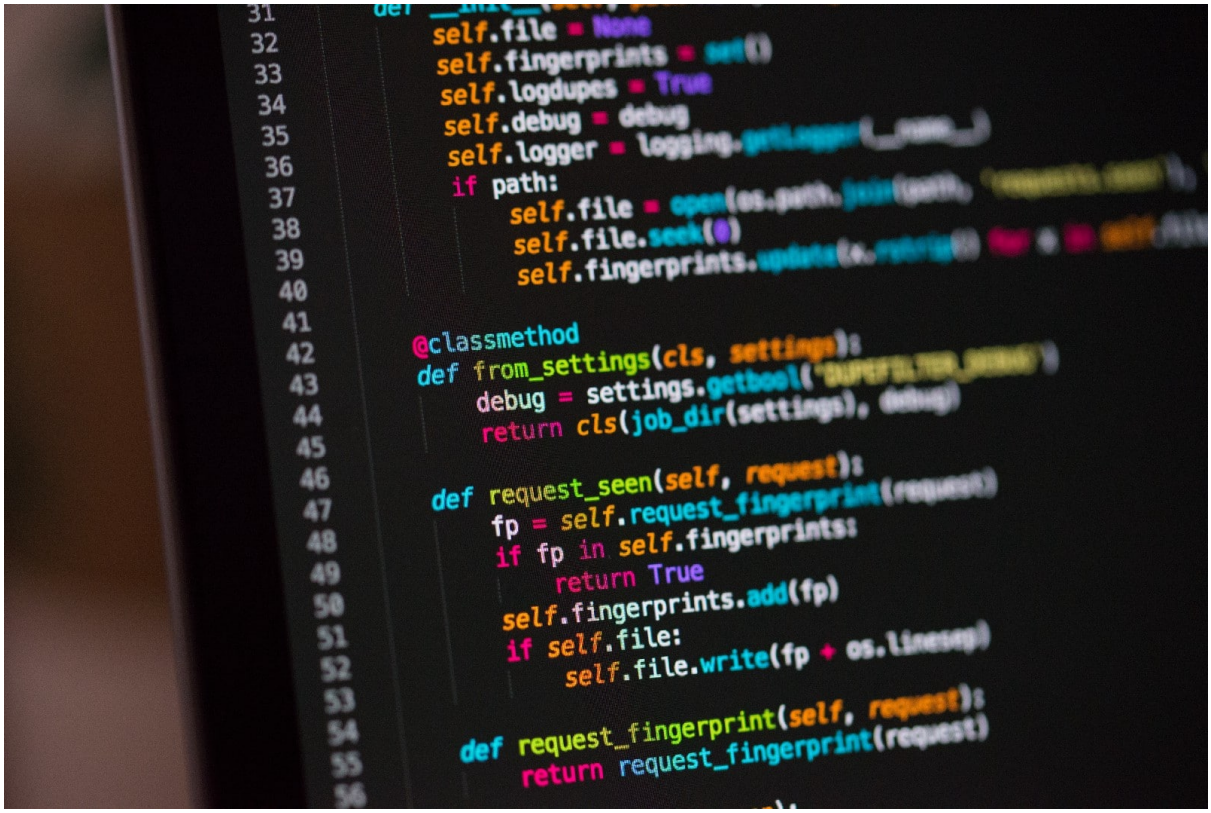


Fig. 2: Source: HOMEWORK

2.1 Guide to the guide

Sections in this tutorial will walk you through coding using micro:bit, basics of programming and micro:bit's features. The objective is to get you started with programming so that you can create a small project of your own that you will show us during your first week to get you started with programming #circular_definition.

You don't have to meticulously go through all the theory covered in Basics of programming, especially if you're a beginner. Start writing simple programs using the micro:bit and read about further programming concepts as you go. Feel free to skip the parts you are confident in and choose the parts that are relevant. As you learn more about programming, you'll naturally keep finding better and more efficient ways to do your projects of the past, but right now you should focus on getting yourself started.

Topics covered here are generally only touched upon and many things are not explained on purpose. Some of the important skills you will need during the course of your studies and work later on will be independent exploration of materials and capability to read official documentation, hence we encourage you to explore other resources than those found here.

If your skills are intermediate/advanced, you might not find this documentation very interesting. However, micro:bit is a highly configurable device and you might like to explore micro:bit runtime, which gives you more flexibility with what it can be used for.

Note: If you feel confused while reading through tutorials or if you feel like you need more guidance to start programming, don't be discouraged! There is a number of free online courses that are great at going through basics of programming with Python, like this one. Try to go through the first few lessons, and everything should make more sense.

CHAPTER 3

Setting Up Your Environment

Before you start coding, you will need a source code editor to be able to write, load and run programs on your micro:bit. There are four main options:

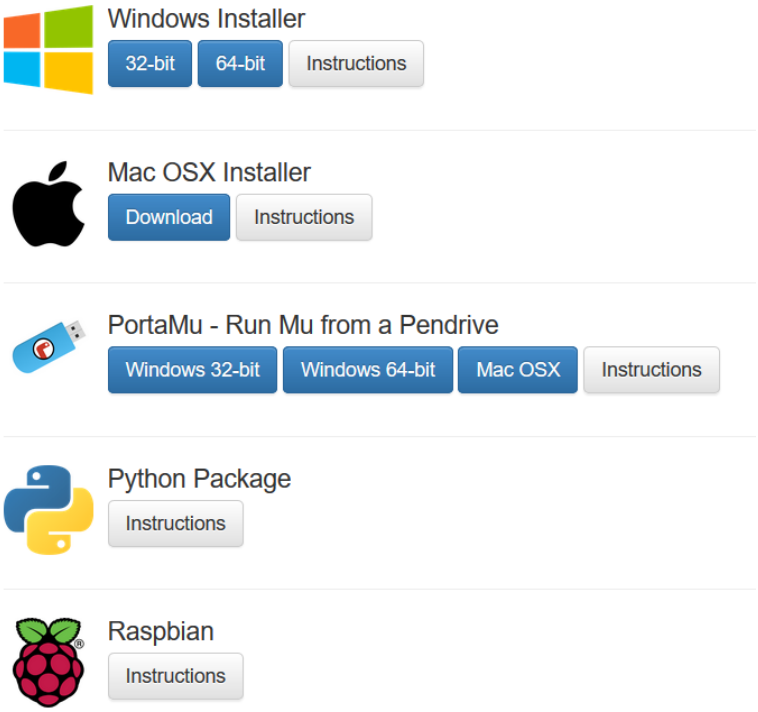
- In-browser micro:bit editor
- Beginner friendly Mu editor
- MicroPython application if you're on a ChromeBook
- Your regular Python editor with a suite of CLI commands (advanced)

For the purposes of this tutorial we will be using the Mu editor, but feel free to use whatever suits you better. To download Mu, go to Mu website.

You can choose different options to install Mu. The ones you'll most likely use is an installer for your device (Mac/Windows), or installation through Python package (pip) using *Command Line Interface* if you've installed Python on your computer.

Note: For those that had worked with Python previously, MicroPython does not support regular Python external libraries, since many are too large for an embedded device. However, a subset was recreated specifically for the MicroPython ecosystem.

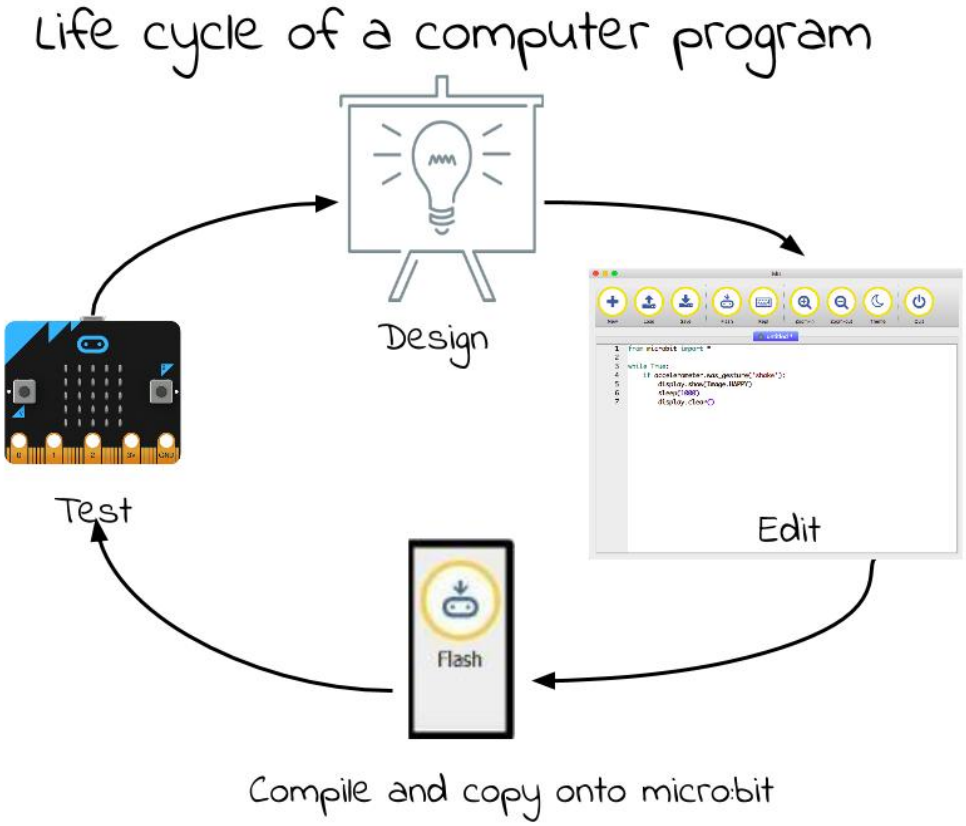
Once the editor is installed, run it and plug micro:bit into your computer.


Windows Installer
32-bit
64-bit
Instructions
Mac OSX Installer
Download
Instructions
PortaMu - Run Mu from a Pendrive
Windows 32-bit
Windows 64-bit
Mac OSX
Instructions
Python Package
Instructions
Raspbian
Instructions

CHAPTER 4

First Program

In general, the process of designing code is composed of these 4 steps. You can expect to go around the loop quite a few times before you get your code working.



4.1 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 and print "Hi there" to Mu console. It's a good practice to think about what you want your code to do and how you're going to do it before you start writing. There's not much planning and design to do here, but just so that you understand what a plan might look like:

```
Repeat forever:
    Scroll "Hello UCL!" across the LED display
    Display a heart icon
    Print "Hello World!" on a console
    Delay for 2 seconds
```

There are two ways to display the output of your code: you either use outputs available on the micro:bit (eg. the LEDs) or the REPL (Read Print Evaluate Loop) console available in the editor using the `print` statement. The console is especially useful for finding bugs (errors) in your code or trying out new concepts or ideas.

Let's go through this line-by-line:

```
from microbit import *
```

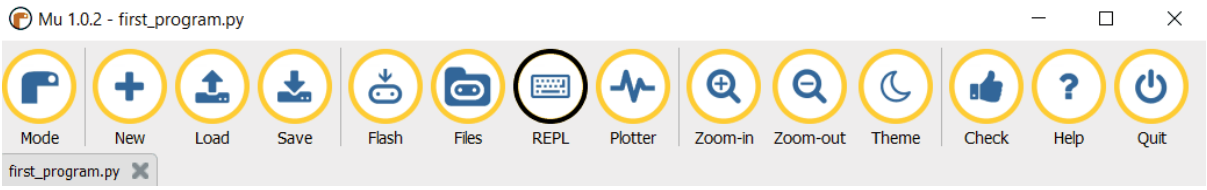
Importing packages (like microbit) in Python makes us able to use functions or objects which are not defined in pure Python. In this case it's for example `display` or `show`.

```
while True:
```

This means do something (whatever follows this statement and is indented) while the condition following `while` is true. In this case, the condition is the keyword `True`, means that this loop will go on forever - it's the same as writing (5 > 1), which evaluates to `True` in the end anyway. The rest of the program is straightforward:

```
from microbit import *

while True:
    display.show('Hello UCL!')
    display.show(Image.HEART)
        print('Hello World!')
    sleep(2000)
```

This displays `Hello UCL!` on the LED display and then shows the heart. 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:



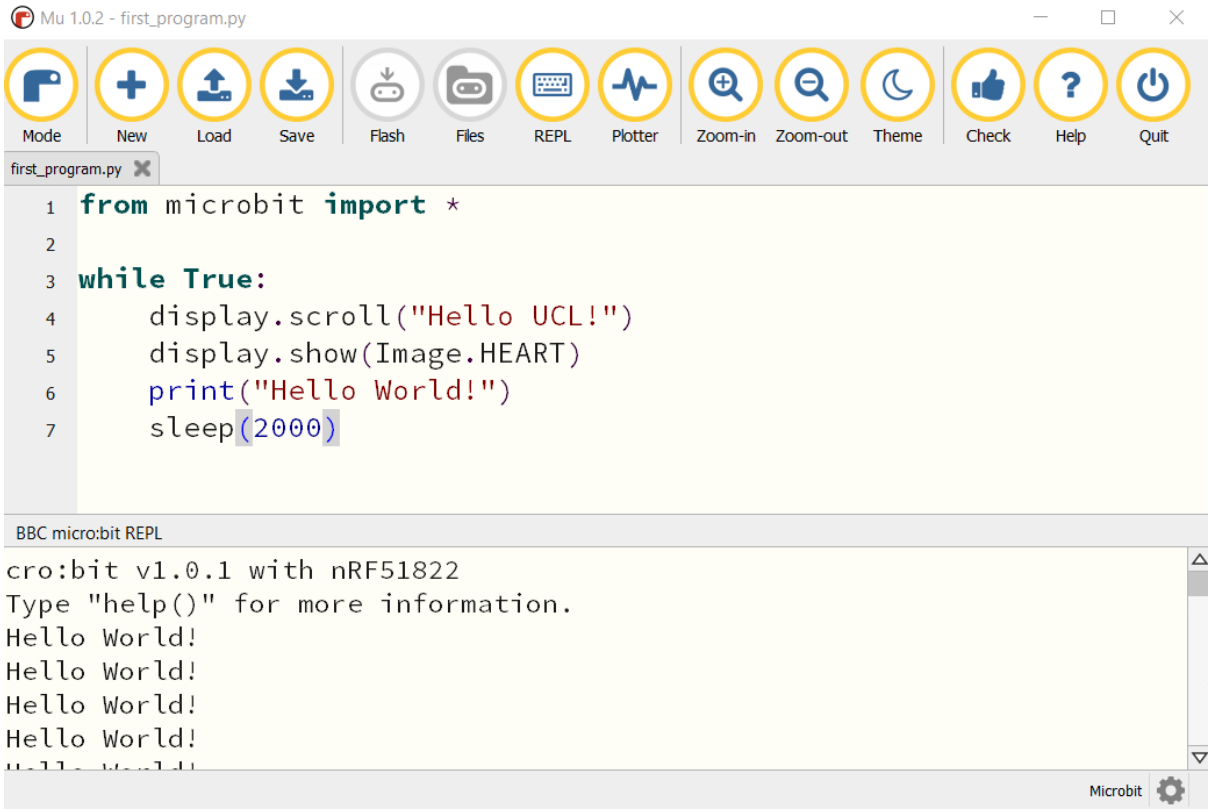
The REPL window shows us messages from the micro:bit and also allows us to send commands directly to the micro:bit. For now, we'll just be using the REPL to see messages that we print and error messages.

4.2 Upload your program

Now click on the Flash button in Mu and see what happens.

The result on the micro:bit should look something like this:

Now try to open the REPL console:



4.3 Make a change

The best way to learn what something is for is to try and change your code (and read the documentation, obviously).

<|°_°|>

Are you wondering what the delay is for? Is it necessary? Try deleting it. What happens if you replace `True` by `False`? What happens when you replace `scroll` by `show`?

Now you have written your first program. Next sections will tell you more about writing more complex programms and about further uses of micro:bit.

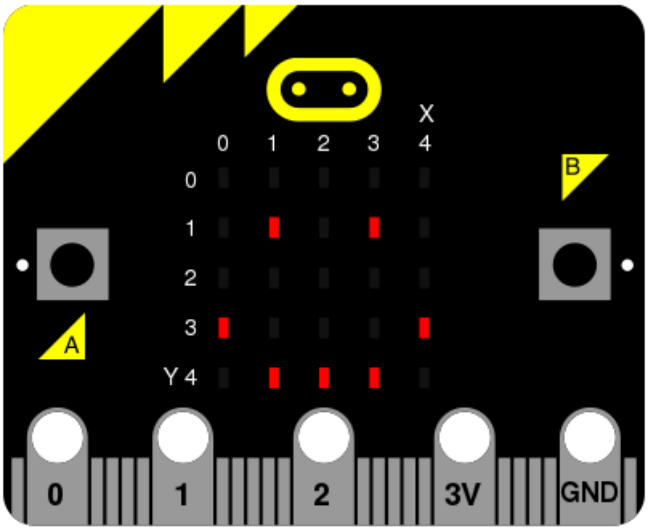
See also:

See the full micro:bit documentation for MicroPython.

CHAPTER 5

LED Display

This is a quick guide to some of the things you can do with the LED display. Try things out – see what happens and see what you can make. There are 25 LEDs numbered from (0,0) in the top left hand corner to (4,4) in the bottom right hand corner and can all be set to a different level of brightness. You can use the LEDs like a screen to display single characters, a strings or a small picture.



5.1 Basic Functions

5.1.1 Display a string or an image

You can display characters on the LED display using `show` method:

```
from microbit import *

display.show("Hello")
```

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

`microbit` module comes with many built-in pictures to show on the display. For example, to display a smiling face

```
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`

5.1.2 Scroll a string

Use `scroll` to scroll a string across the display:

```
from microbit import *

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

5.1.3 Clear the display

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

```
from microbit import *

display.clear()
```

5.2 Advanced Functions

5.2.1 Set a pixel

You can set a pixel brightness on 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.

5.2.2 DIY images

What if you want to make your own image to display on the micro:bit?

As mentioned above, each LED pixel on the physical display can be set to one of ten values from 0 (off) to 9 (fully on). Armed with this piece of information, it's possible to create a new image like this

```
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:

```
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.

5.2.3 Animation

To make an animation, just use a list of images.

We can demonstrate this on built in lists - `Image.ALL_CLOCKS` and `Image.ALL_ARROWS`:

```
from microbit import *

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

Micro:bit shows each image in the list, one after another. By setting `loop=True`, program will be keep looping through the list indefinitely. It's also possible to set a delay between the pictures using the `delay` attribute to the desired value in milliseconds `delay=100`.

To create your own animation, you need to create a list of images.

In this example, a boat will sink into the bottom of the display. To do that, we defined a list of 6 boat images:

```
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)
```

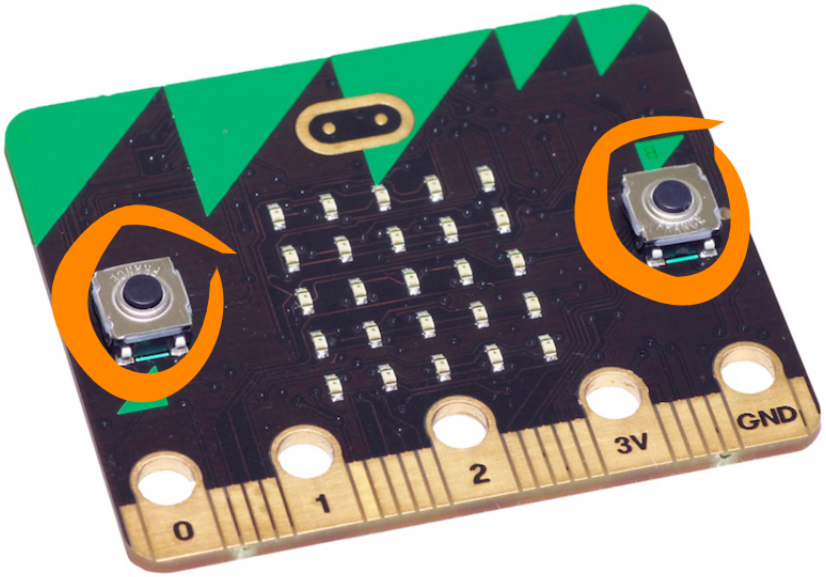
5.3 Practice questions

- 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?

CHAPTER 6

Buttons

The micro:bit has two buttons: `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.

6.1 Basic Functions

6.1.1 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:

```
from microbit import *
```

(continues on next page)

(continued from previous page)

```
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.

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. It might be the case that the user pushes the button while the code is doing the something else, and the press is missed. 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. In this way you need never miss a button press again:

```
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 it will show up once the if statement test condition is executed. It becomes more apparent as you increase the delay.

Now try using button_a.isPressed() instead of button_a.was_pressed().

6.1.2 Counting the number of presses

To count the number of times a button was pressed, you can use 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 count.

Can you define your own get_presses function?

6.2 Advanced Functions

6.2.1 Checking for both buttons

It is possible to check a series of events by using conditional statements. 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:

```
from microbit import *

while True:
```

(continues on next page)

(continued from previous page)

```
    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)
```

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

6.3 Practice questions

- Change what is displayed when you press the button.
- Games that need user input.

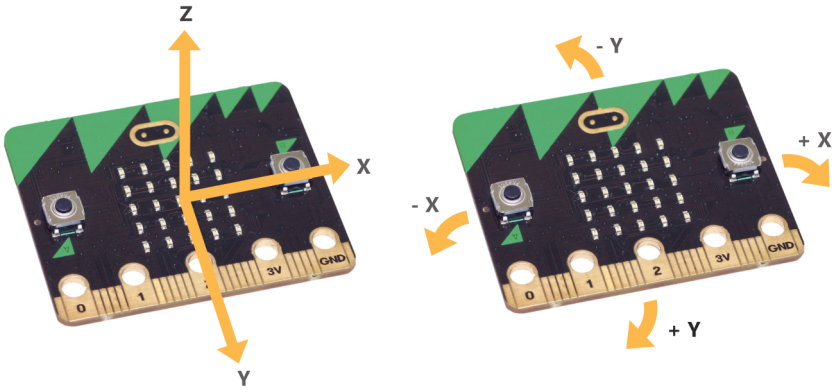
CHAPTER 7

Accelerometer

As its name suggests, the accelerometer on a BBC micro:bit measures acceleration. The accelerometer is set to measure acceleration values in the range +2g to -2g, and cannot be changed with MicroPython so far. These values are registered on a scale of values in range 0 .. 1024.

The micro:bit measures movement along three axes:

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



7.1 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.

You can access acceleration measurements one at a time or get all three values at once and store them in a list. You can learn more about lists in the basics of programming section, but for now, just use the following code:

```
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 $\pm$2048 milli-g. That's because this accelerometer is set to measure a maximum of $\pm$2048 milli-g: the true number might be higher than that.

If you've ever wondered how a mobile phone knows which way to orient its screen, it's because it uses an accelerometer in exactly the same way as the program above. Game controllers also contain accelerometers to enable steering.

7.1.1 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 micro:bit is able to detect this.

micro:bit 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.

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 display a happy emoticon if it's face up:

```
from microbit import *

while True:
    gesture = accelerometer.current_gesture()
```

(continues on next page)

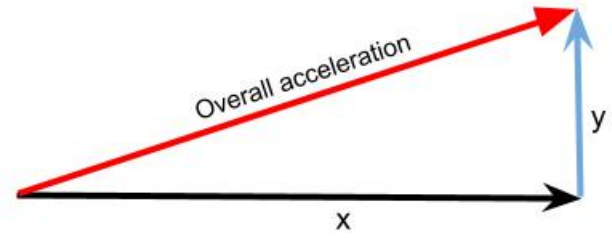
(continued from previous page)

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

Within the *scope* of the loop the current gesture is read and assigned to `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!

7.2 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 – it has a magnitude (size, length) and a direction. To get the overall magnitude, irrespective of orientation, with only X and Y axes (i.e. we had a 2D accelerometer) the situation would be:



We can calculate the magnitude (length) of the resultant using Pythagoras' rule:

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

The same principle holds with a 3D accelerometer. So the overall magnitude of the resultant acceleration vector is:

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

Calculating 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.

7.3 Practice questions

- 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.

CHAPTER 8

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 due to the magnetic field introduced by what is 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.

8.1 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:

```
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 still need to calibrate the device before you use `compass.heading`.

8.2 Practice questions

- 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).

CHAPTER 9

Thermometer

The thermometer on the micro:bit is embedded in one of the chips – and chips get warm when powered up. Consequently, it doesn't measure room temperature very accurately. The chip that is used to measure temperature can be found on the left hand side of the back of the micro:bit:

9.1 Basic Functions

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

```
from microbit import *
```

(continues on next page)

(continued from previous page)

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

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?

9.2 Practice questions

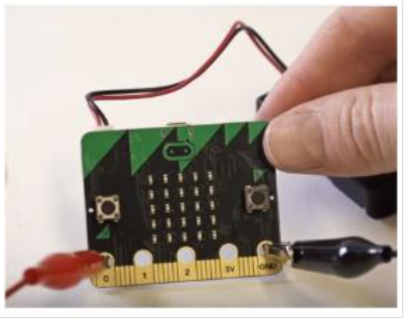
- Try calibrating the thermometer. Does it still give the right temperature when you move it to a warmer or cooler place?
- Make the LEDs change pattern as temperature changes
- Find out how much the temperature changes in a room when you open a window – what do you think that tells you about heating energy wasted?

CHAPTER 10

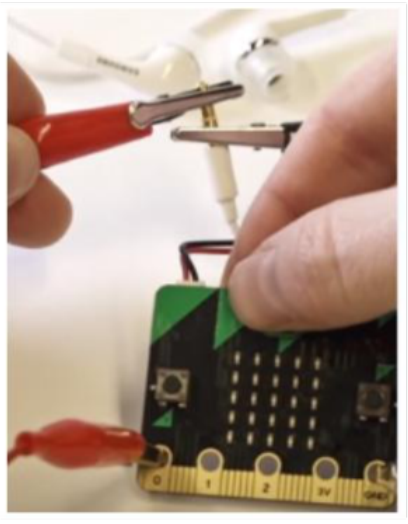
Sound

Micro:bit can be used to play simple tunes, provided that you connect a speaker to your board.

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



Connect crocodile clips to pin 0 and GND respectively



Connect the clip leading from the ground pin to the base and clip from pin 0 to the tip of the headphone jack

Warning: You cannot control the volume of the sound level from the micro:bit. Please be very careful if you were using headphones. A speaker is a safer choice while working with sound.

You can also connect your micro:bit to a speaker using crocodile clips:

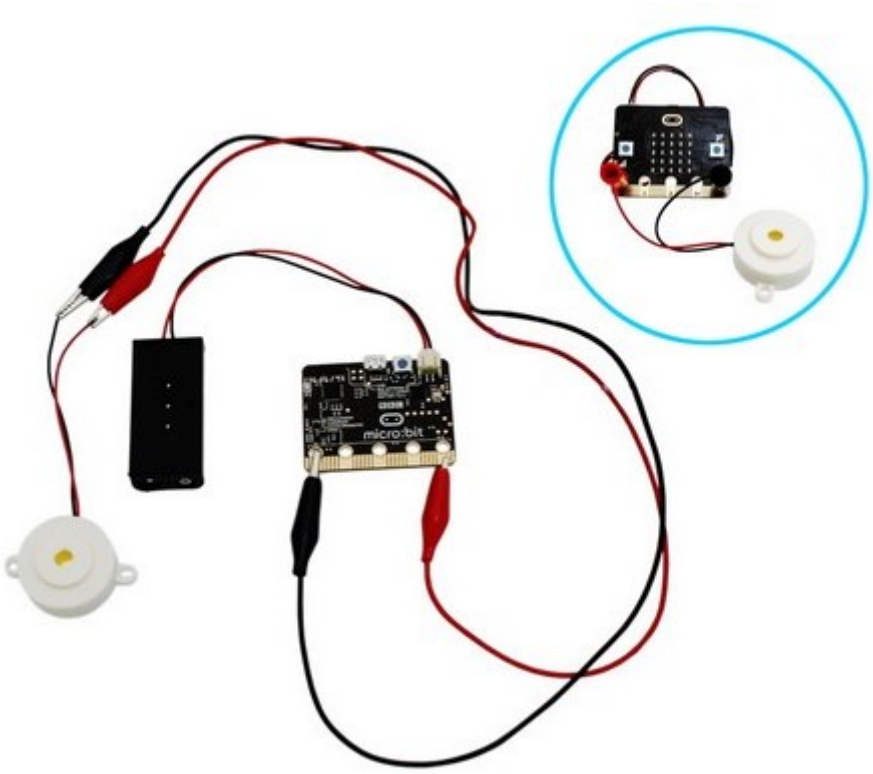


Fig. 1: Source: <https://www.kitronik.co.uk/blog/microbit-alarm-kitronik-university/>

10.1 Basic Functions

10.1.1 Play a tune

To play a tune you can use the `play` function:

```
from microbit import *
import music

music.play(music.NYAN)
```

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

The music module includes a number of built-in tunes. Here's some of them:

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

10.1.2 Make your own tune

To play a tune, specify the note (C,D,E,F,G,A,B; including sharps (eg.: C#)) to play. Optionally, it's possible to specify the octave (1-8) and the duration it will be played for:

```
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)
```

Micro:bit will remember the octave of the note defined previously. Hence, the tune above can be rewritten as follows:

```
tune = ["C4:4", "D4:4", "E4:4", "C:4", "C:4", "D:4", "E:4", "C:4",
        "E:4", "F4:4", "G4:8", "E:4", "F:4", "G:8"]
```

10.2 Advanced Functions

You can also specify the note you want to play using its frequency using the `pitch` method. For example, to create a police siren effect

```
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.

10.3 Practice questions

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

CHAPTER 11

Radio

Micro:bit has a Bluetooth Lowe Energy (BLE) antenna that can be used to transmit and receive messages.

11.1 Basic Functions

11.1.1 Getting ready

Before you can use the radio you must remember to import the `radio` module and to turn the radio on. Once the radio is on, it will be able to receive messages from any other micro:bit within range:

```
from microbit import *
import radio

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`:

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

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 won't get transmitted very far. The power level can be a value between `0` and `7`:

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

11.1.2 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:

```
message_to_master = "Ash nazg durbatulûk, ash nazg gimbatul, ash nazg thrakatulûk,␣
↪agh burzum-ishi krimpatul."

radio.send(message_to_master)
```

Receiving a message:

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

11.1.3 Putting it together

```
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

message_to_master = "Ash nazg durbatulûk, ash nazg gimbatul, ash nazg thrakatulûk,␣
↪agh burzum-ishi krimpatul."

# Event loop.
while True:
        radio.send(message_to_master)
        incoming = radio.receive()
```

(continues on next page)

(continued from previous page)

```
        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.

11.1.4 Interfacing With Your Phone

Using the microbit Bluetooth antenna, it's possible to connect your micro:bit to your phone and interact with micro:bit wirelessly. However, MicroPython does not support this capability due to lack of RAM capacity.

11.2 Practice questions

- 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. Leave 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!

CHAPTER 12

Edge Connector

To fully starts using micro:bit and its functions to interface with the world, you will need to learn about its edge connector. This can be used to connect to external circuits and components. However, in order to use the edge connector, you will need extra equipment like crocodile clips/banana plugs or other external PCB connectors (1.27mm pitch).

Fig. 1: Source: https://tech.microbit.org/hardware/edgeconnector/

There are 5 rings to connect with 4mm crocodile clips/banana plugs - 2 of them (3V, GND) are connected to the micro:bit power supply and the other three can be used for general purpose input/output using both analog and PWM (pulse width modulation) and touch sensing. The smaller 1.27 mm strips allow you to either connect to different parts of micro:bit or which you can use for your own purposes.

You can find the description of each of the pins and what they can be used for at <https://microbit.pinout.xyz/>. Refer to the *developer reference* for further information.

developer reference: https://tech.microbit.org/hardware/edgeconnector/

CHAPTER 13

Data Types

In order to accurately capture various types of data, programming languages provide us with different data types to allow us to represent them properly. Each case, whether it is gathering acceleration values from the accelerometer or counting the number of times a button was pressed, requires a different approach, which is why Python and most other programming languages recognise several data types for representing values:

Data Type	Description	Example
Integers	Whole numbers	`42`
Floats	Numbers with decimal point, fractions	`3.1415`
Complex numbers	Numbers with both real and an imaginary component	`1 + 3j`
Strings	Sequences of characters delimited by quotation marks	`"Hello World!"`
Booleans	Values representing true and false values	`False`

In a simple program you might use all of these. Here are the data types you could use in a program storing information about your micro:bit game:

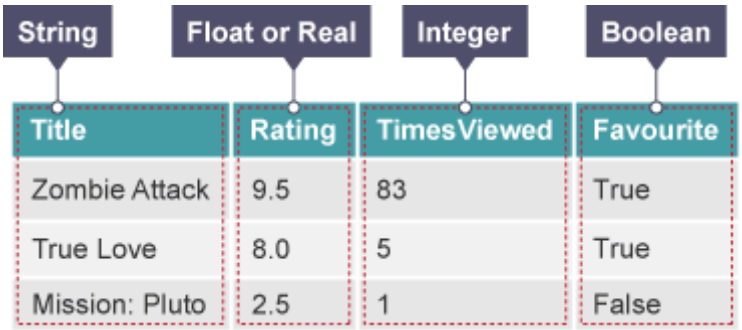


Fig. 1: Source: <http://www.bbc.co.uk/education/guides/zwmbgk7/revision/3>

13.1 Operations

13.1.1 Numbers

Basic arithmetic operators: `+`, `-`, `*`, `/` are used in the same way as you would with a calculator. Let's look at an example using arithmetic operators the temperature read by the micro:bit in Celsius to Fahrenheit:

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

Warning: Python recognises two division operators: `/` and `//`. First one outputs the result you'd expect, but the second one does integer division: the return value is actually the floor of the result. This means that the return value is always rounded down.

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's even, then there will be no remainder:

```
number = 3
if number % 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 number is odd`, otherwise it will print `The number is even`. You might write this program in a different way. People think about problems in different ways and no two programs are likely to be the same.

13.1.2 Strings

As stated above, strings (`str` type in Python) are sequences of characters, with a length limited only by the memory of your machine. A useful fact to note is that they can be concatenated using a + symbol:

```
name = "Hayley"

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

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

To 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))
```

Note: Python natively provides a lot of methods, which makes using strings much easier and saves lot of time (although implementing them on your own initially might be a good programming exercise).

13.1.3 Booleans

A Boolean value is a value that is either `True` or `False`, also represented by *1* and *0*. In Python, there is a number of operations that allow you to manipulate boolean expressions.

Comparison

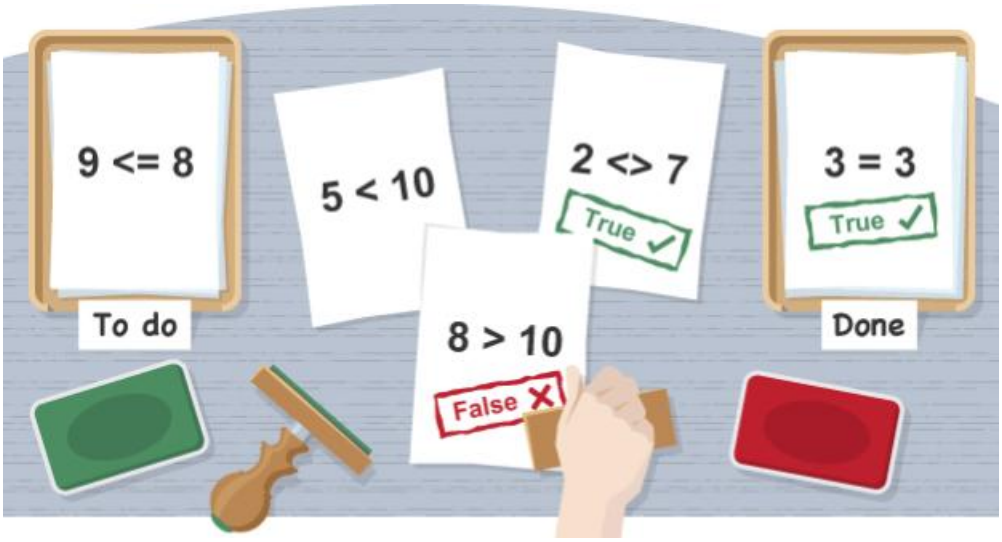


Fig. 2: Source: <http://www.bbc.co.uk/education/guides/zy9thyc/revision>

Comparison operations are useful to test variable values in conditional statements or loops. Here are some examples of comparisons 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
```

Logical operations

Logical operators test the truth value of their operands.

Operator	Evaluates to "True" if:	Example
and	Both operands are true	`True and True`
or	At least one operand is true	`True or False`
not	Operand is false	`not False`

Membership operations

Membership operators are useful to determine presence of an element in a sequence.

Operator	Evaluates to "True" if:	Example
in	A variable value is in the specified sequence	`x in [1, 2, 3, 4]`
not in	Does not find a variable value in the specified sequence	`x not in [1, 2, 3, 4]`

Using Boolean operations

You may have already used some examples that do this. In this example, the micro:bit will show an arrow changing in direction according to acceleration:

```
from microbit import *

while True:
    x_bearing = accelerometer.get_x()

    if (x_bearing <= 100) and (x_acceleration >= 50):
        display.show(Image.ARROW_N)

    elif x_bearing > 100:
        display.show(Image.ARROW_E)

    elif  x_bearing < 50:
        display.show(Image.ARROW_W)

    else:
        display.show(Image.ARROW_S)
```

13.2 Practice Questions

1. Predict whether the return value is True or False. If False, explain why.
    a) "hello" == 'hello'
    b) 10 == 10.0
    c) 5/2 == 5//2
    d) 5 in [x for x in range(0,5)]
    e) 0 == False
    f) 1 == true
    g) 0.1 + 0.2 == 0.3

CHAPTER 14

Variables

A variable is a name you use to refer to a memory location where a value is stored. In a more abstract manner it can be thought of as a box that stores a value. All variables are made up of three parts: a name, a data type and a value. In the figure below there are three variables of different types:

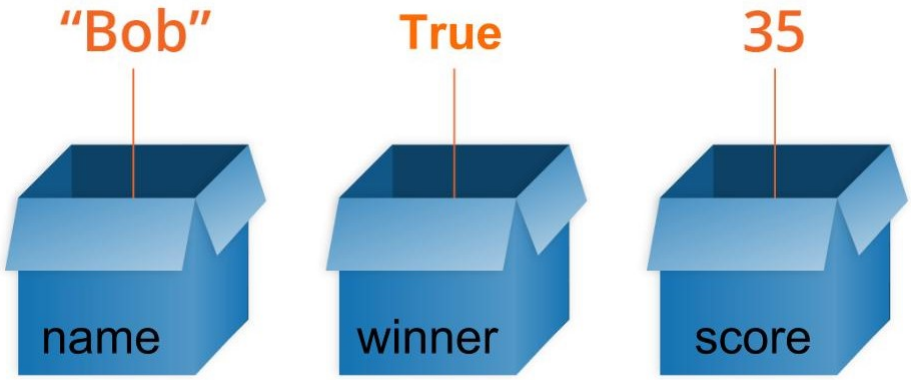


Fig. 1: Source: <https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/Variables>

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

In Python a variable is created when it's assigned to for the first time. Once done, variable value can be manipulated (unless it's immutable - you'll learn more about this in the *Data Structures* section).

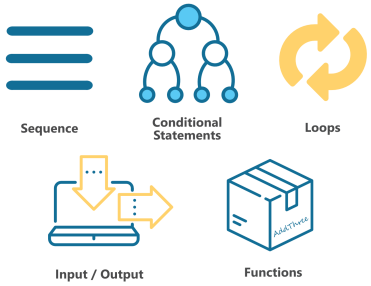
```
from microbit import *

myCount = 0

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

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?

CHAPTER 15

Control Structures

The first program in Getting Started section consisted of sequential execution of tasks. However, the vast majority of times you will need a more complex structure for your code - whether it is control over which statements get executed or how many times they get executed. This is when control structures introduced in this section - such as loops or conditional statements - come in handy.



15.1 Conditional statements

First example of a use case of control structures arises when you want to execute a part of your code only if a certain condition is satisfied. For example, if you want to trigger an event only if a button is pressed:

```
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)
```

In case you want to execute a different task under several different conditions, use `elif` (short for `else if`) statement. `else` statement comes in useful if you want to do something in the rest of the cases, where no condition is defined. The latter two statements are only usable if you had used an `if` statement previously, but neither is mandatory.

15.2 Loops

Loops are a very useful structure in case you want to repeat a certain block of code several times over. There are two types of loops: `for` loops, that keep count of the number of times a block of code is executed, and `while` loops which perform an action until a condition you've specified is no longer true.

15.2.1 For loops

There are times when you want to do an action a specific number of times, or you need to keep track of how many times it was performed. For example you'd like to turn on the LEDs on the uppermost horizontal and rightmost vertical side. You can use a `for` loop to change which LED lights up like this:

```
from microbit import *

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

   # set the pixel in column 4, row i to 9
   display.set_pixel(4, i, 9)
```

Here is another example. You could use a `for loop` to turn on all the LEDs in sequence, 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` for consecutive values in the range `0` to `4` each time (in Python and most programming languages, we always start counting from `0`). The loop will stop before it reaches 5, the final value in the range.

The inner loop:

```
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.

15.2.2 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 a game or perhaps it is collecting temperature data. Here is an example of some code to repeat forever:

```
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.show(Image.HAPPY)
```

15.3 Practice Questions

1. Display a different image depending on which side microbit is tilted in.
2. Program an LED 'sprite' that moves in the direction micro:bit is tilted in.
3. Program an LED sprite to run in a circle. Try to extend it to a snake by adding a tail of LEDs to the original sprite.
4. Do the same as in previous question, but this time make the sprite stop when a button is being pressed and restart if it's pressed again.

CHAPTER 16

Data Structures

16.1 Lists

List is a data structure used to store any data type (or structure) in an ordered manner. It's a data structure that you'll probably use most often. 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 pieces of data stored in a list are called *elements*.

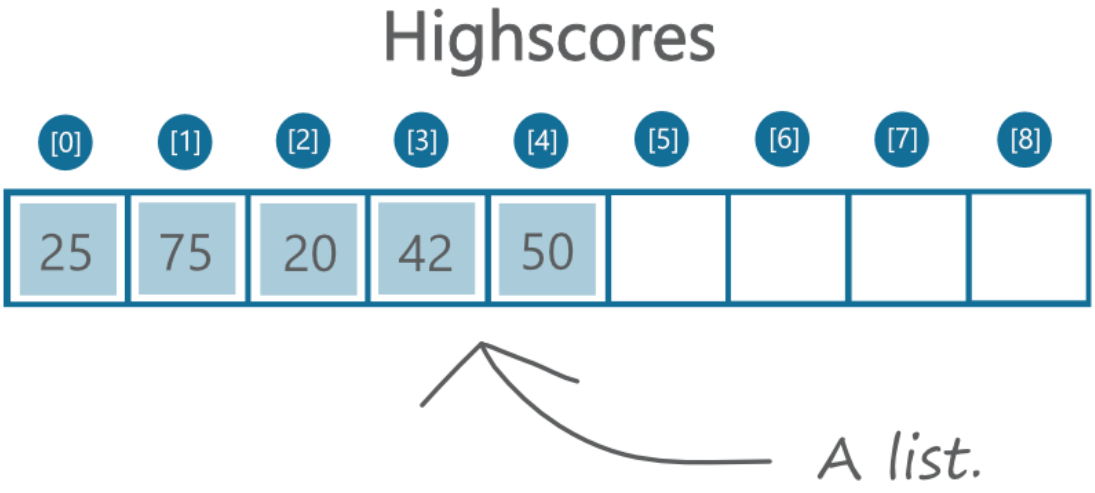


Fig. 1: A list.

To create a list, you specify its contents enclosed within brackets and delimited by commas:

```
from microbit import *

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

Finding the value of one of the elements in a list is straightforward as long as you keep in mind that Python counts the elements from '0'. In the `high_scores` list above, `high_scores[0]` is 25 and `high_scores[3]` is 15.

Here you can also see that particular elements in a list can be accessed by their index. Furthermore, it is possible to slice lists to get only a part of a list depending on the index. If you only want the first three, you can write `high_scores[0:3]`, or, since we are starting at 0, we can shorten it to `high_scores[:3]`. Mind that the right endpoint is alway excluded, so the 'slice' above refers to the mathematical interval `[0,2]`.

Not surprisingly, Python has features for working with lists. The code snippet below calculates the sum of all elements and then calculates the average high score.

```
total_score = 0

for score in high_scores:                # For each element ...
        total_score = total_score + score

average = total_score / len(high_scores)  # Use the len() function here to find␣
↪the length of the array
```

The same can be done in one line using the `sum` function:

```
average_quick = sum(high_score) / len(high_score)
```

Since you don't necessarily know what values in the list are going to be, or how large the list will be, it's useful to know the `append` function. For example, you can use it to fill a list with temperature readings or accelerometer values:

```
from microbit import *

recorded_temperature = []                # Create an empty list
for i in range(100):                     # Add 100 temperature values
        recorded_temperature.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 to the end of the list.

Deleting items from a list is just as straightforward:

```
high_scores.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:

```
high_scores.pop(3)
```

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

```
high_scores.pop()
```

will delete the last element in the list.

Tip: You can look here to see more useful methods on lists.

Note: You might be wondering whether strings can be considered to be a list. Even though string is an array of characters and we can even do similar operations on them (like slicing), they are both different types of objects with different methods (try to type `dir(str)` and `dir(list)` in your console).

16.1.1 Sorting

Often you'll find the need to have data in your list sorted, for example when implementing search algorithms. In Python, sorting lists is easy using the `sort(key=, reverse=)` method:

```
high_scores = [25, 20, 10, 15, 30]
high_scores.sort()
```

You don't only have to sort numbers - its optional key parameter allows you to specify your own function for comparing elements in your list (for example, while sorting a list of strings according to length, you can pass the len() function as the parameter). Passing false to reverse parameter allows you to sort in a descending order.

```
list = ['longest', 'short', 'longer']

# Sort list in ascending order of string length
list.sort(key=len)
# Sort list in descending order of string length
list.sort(key=len, reverse=True)
```

16.2 Tuples

Tuples are similar to lists in that they are used to store an ordered sequence of items, usually of varying datatype.:

```
high_scores_immutable = (25, 20, 10, 15, 30)
```

You can retrieve values in the same way as with lists, but the most important difference is that tuples are *immutable*. This means, that in the `high_scores` list above, you can change the value of individual elements:

```
high_scores[0] = 42
```

However, trying to change a value inside `high_scores_immutable` will return a `TypeError: Object tuple does not support item assignment`. Once you assign values inside a tuple, they cannot be changed.

Mutability is another difference between strings and lists - while lists are mutable, string are not.

16.3 Sets

Unlike lists and tuples, sets hold an **unordered** collection of elements with no duplicates. This makes them suitable for testing membership or removing duplicate elements.

```
set = {8, 12, 22}

# Add a single element to set
set.add(42)

# Add several elements to set
set.update([16, 32, 64])

# Remove an element from set - throws an error if element not in set
set.remove(42)

# Remove an element if present in set
set.discard(42)
```

Since a set is an unordered collection of elements, indexing is not possible. Python supports typical set operation methods:

```
set_a = {1,2,3,4,5}
set_b = {4,5,6,7}
set_c = {1,2}

# Check for membership
2 in set_a

# Return elements in the intersection of set_a and set_b
set_a.intersection(set_b)
# Return true if set_a contains all the elements of set_c
set_a.issuperset(set_c)
```

An empty set is created using a `set()` method, as using braces creates an empty dictionary (see below).

For more methods, visit Python documentation.

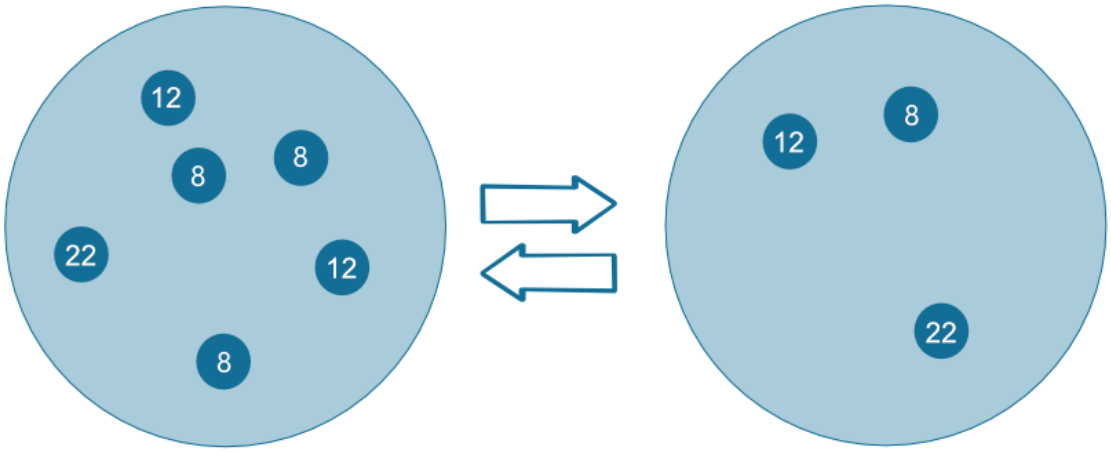


Fig. 2: All elements within a set are unique

16.4 Dictionaries

Dictionary is an unordered set of `key : value` pairs. It's a rule that all keys are unique and have no duplicates. Unlike lists or tuples, which are indexed by numbers, you can retrieve a value from a dictionary by using the key as an index.

For example, you can store the highscores of all the players:

```
game_register = { 'googolplex': 100,
                  'terminat0r': 27,
                  'r00t': 150,
                  'dent': 42,
                  'teapot418' : 0 }

# Access elements
game_register['dent']

# Add or update and existing entry
game_register['pepper'] = 50

# Delete an entry
del game_register['pepper']
```

(continues on next page)

(continued from previous page)

```
# Delete all entries
game_register.clear()

# Delete the dictionary
del game_register

# Retrieve a value for the key or default if not in dicionary
game_register.get('dent')
```

16.5 Practice Questions

1. Use micro:bit list Image.ALL_CLOCKS and iterate over all items in the list with a for loop, showing them on the LED screen.
2. Using the same item list, show only items with an index divisible by 3.
3. Sort an integer list (for example `list = [20, 112, 45, 80, 23]`) using the last digit of each item and keep their relative positions in case the digit is the same (the result in this case would be `[20, 80, 112, 23, 45]`).
4. Create an animation of your own using a tuple and play it on the micro:bit LED screen.
5. Program microbit to take a compass reading upon press of a button and store the results in a tuple.
6. Write a program to keep record of gestures recognizable by microbit and the number of times they've been detected using a dictionary.

CHAPTER 17

Functions I

Functions and methods refer to useful snippets of code that serve a specific purpose and are usually used many times in your program. You have likely already used both functions and methods without necessarily realising it. In this section we will not discuss methods further, but we will explain how to use and write functions.

17.1 Using Functions

A great thing about functions is that they can be used in more than one program and avoid code repetition. In the same way you can use functions that other people have written. In python, useful functions can be assembled into modules (although you don't have to do this) - the `random` module is a good example. To use functions in the random module you must first *import* the module. Once you've done that, you can use any of the functions in that module. Here are two examples of functions from 'random'.

The `random.randint()` function will allow us to generate a random integer in a given range:

```
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.

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 a list element at the index of the generated integer:

```
from microbit import *
import random

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

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

17.2 Writing your own Functions

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

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

To use the function, it has to be *called* :

```
showGreeting()
```

That's not a very interesting function, is it? You can make functions more powerful by using *parameters* and *return values*. If you think of a function like a black box then a parameter is an input value and a return value is what you will get at 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:

```
from microbit import *

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


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

The function `printBirthdayGreeting` composes the birthday message for us and returns a string. `str()` is used 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/need to.

17.3 Practice Questions

1. Write a function `blink(x, y)`, that will blink a LED at coordinates specified by parameters x and y once.
2. Use the `blink(x, y)` function to blink all LEDs one after another.
3. Write a function button_count() that will return a tuple containing the number of times button A and button B were pressed. (fix)
4. Combine the two functions into a program that will allow user to set coordinates of blinking LED by pressing buttons.
5. Look at the scripts you wrote previously and check whether you could (or not) improve your code by using functions.

CHAPTER 18

Functions II

Now that you know how to use functions in practice, there are several more concepts that will help you understand behaviour of functions not only in Python, but other languages as well.

18.1 Scope

18.1.1 Global and Local Variables

Imagine you want to use the slightly modified `printBirthdayGreeting()` function from before and you want to increment age every time the function is called:

```
name = "Johann"
age = 32

def printBirthdayGreeting():
        age += 1
    return "Happy Birthday " + name + ", you are " + str(age) + " years old"

printBirthdayGreeting()
```

Can you spot what is wrong? If you try to run it, you'll probably get this message: '' UnboundLocalError: local variable 'age' referenced before assignment''.

To understand this, we have to talk about scope. Scope is an 'area' in which a variable is defined, can be accessed and written to. From this point of view we know two types of variables: global and local. By default, all variables defined within a function are local - you cannot access them outside of the function. And since the scope within the function is different from the global one, it's possible to use the same name for two different variables.

Can you explain what happened in the code snippet above now?

`age` outside of `printBirthdayGreeting()` function is a global variable. However, when we want to access it inside the function, Python considers it to be a new local variable. How do we solve this? We can declare the variable `age` as `global`:

```
name = "Johann"
age = 32
```

(continues on next page)

(continued from previous page)

```
def printBirthdayGreeting():
        global age
        age += 1
        return "Happy Birthday " + name + ", you are " + str(age) + " years old"
```

This will let Python now, that the age variable we mean is the one in global namespace.

Warning: Using global variables is generally a bad practice and you should avoid it, since it makes the purpose of your functions less obvious and you can end up with 'spaghetti' code. A better way to do this is to pass variable age as one of the arguments of the function (example below).

Here is an example for a function that passes variables as arguments:

```
def printBirthdayGreeting(name, age):
        age += 1
        return "Happy Birthday " + name + ", you are " + str(age) + " years old"
```

Tip: You will be hearing about 'best practices' a lot. How do you determine what is a best practice and what is not? In general, best practice is what makes your code more readable to others. You can look at style guides for a language you're coding in, but in the end it's always about good judgment, since no rule applies to all cases.

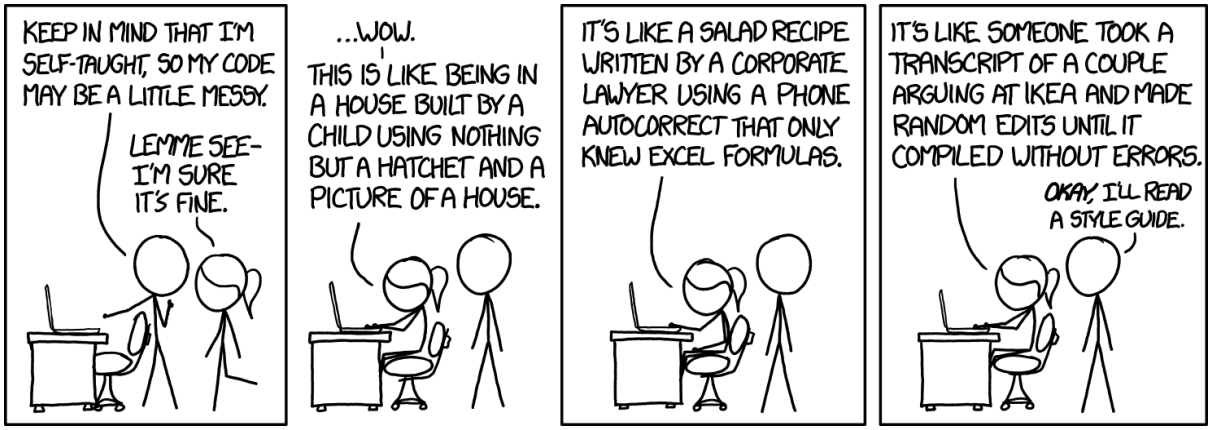


Fig. 1: Source: xkcd https://xkcd.com/1513/

18.1.2 Nonlocal variables

A curious case arises with the use of nested functions. So let's say you want to change a local variable of the `justAnExample()` function using the nested function:

```
def justAnExample():
        def continuingExample():
                variable = "Inner function that changes everything!"

        variable = "Outer function"
        continuingExample()

        print(variable)

justAnExample()
```

You already know why this does not work. But how do you fix it? You cannot declare the variable global, because it's within a function - it's local and there is another local scope within the `continuingExample()` function. To resolve this situation, you can declare a variable to be `nonlocal`:

```
def justAnExample():
        def continuingExample():
                nonlocal variable
                variable = "Inner function that changes everything!"

        variable = "Outer function"
        continuingExample()

        print(variable)

justAnExample()
```

Now the code should print `"Inner function that changes everything!"` exactly the way we wanted.

Note: To learn more about namespace and scope in Python, look at the documentation.

18.2 Passing parameters

An important concept that will have a visible impact on the working of your functions is passing parameters. This describes the way a variable is treated as it's passed in a function - in a pass-by-value scenario, the argument is treated as a new local variable and has no influence on the original variable (if a variable was passed as an argument). In the case of pass-by-reference, the variable passed as an argument can be affected within a function. In Python, the method of parameter passing is a specific combination of the two - parameter are passed by value of object reference.

For a good explanation of passing parameters and the difference between different techniques, I would recommend you to read this blogpost by Robert Heaton.

CHAPTER 19

Classes and Objects

Python is an object-oriented language - it's based on the concept of "objects" that contain some fields (variables) and methods (functions). It's procedures can modify it's attributes (fields). Everything in Python is an object - whether it's an integer or a string.

`Object` is a handy concept to make a representation of something abstract - for example a useful way of storing data in an ordered manner. Reminds you of something? Yes, a list. How would you describe a list object? What attributes does it have? When/How is it useful?

Tip: Do you remember the `dir(ClassName)` command? It lists all the attributes and methods of a required class, such as `dir(str)`.

If you find you need an object that Python does not have, you can create your own. To create an instance of an object, you first need to construct a "template", that will define what the object will look like and what it's capable of. The prototype is called a class:

```
class Player():
```

This example shows a template of a Player object, which is empy and not very useful right now. To make it more useful, we can add attributes to it - `total_count` is class attribute, that keeps count of all the instances of objects of class Player by incrementing a value every time a new Player() object is instantiated.

```
class Player():
    total_count = 0
```

A class attribute is the same for any instance of class Player, and so you can find out the total number of players through any of them. Other attributes that could be useful would be instance attributes (different for every instance of an object) name and score. How will they be defined? And how can we know when a new object is created to increment the `total_count`?

It is possible to define an `__init__()` method for your class, which will be used during an instantiation of a new object and which can take in other arguments and specify an initial state of an object. In this way, when the object `player_1` below is instantiated, the player's initial score will be zero, the name will be as specified in the argument and `total_count` will increment by one.

```
class Player():
    total_count = 0

```

(continues on next page)

(continued from previous page)

```
    def __init__(self, name):
        self.name = name
        self.score = 0
        self.__class__.total_count += 1
```

Furthermore, we can define methods specifically for our class of objects. For example, class methods `update_score()` and `change_name` to update values of `name` and `score`.

```
class Player():
    total_count = 0

    def __init__(self, name):
        self.name = name
        self.score = 0
        self.__class__.total_count += 1

    def update_score(self, score):
        self.score = score

    def change_name(self, name):
        self.name = name
```

Instantiating objects and using methods is rather straightforward:

```
# Create an instance of an object of class Player
player_1 = Player("teapot418")
player_2 = Player("r00t")

# Change value of score of player_1
player_1.update_score(40)
# Change value of name of player_1
player_2.change_name("bott0m")
```

Now you might wonder, why does calling methods on `player_1` or `player_2` work with one argument only, while the method definitions have two arguments? Surely Python raises an error in this case. As you may have guessed, the instance object - `player_1` - is passed as the first argument, and is actually equivalent to saying `Player.update_score(player_1, 40)`.

Note: The keyword `self` has no special meaning in Python, it is just a convention. You should use it if only for the reason of making your code more

---

readable to others or yourself when you come back to it after some time (you can read more on discussion of `self` in this blogpost by Guido van Rossum - the father of Python).

Accessing attributes is the same for all objects again: `obj.attribute_name`. For example, to print the name of a Player object you write:

```
print(player_1.name)
```

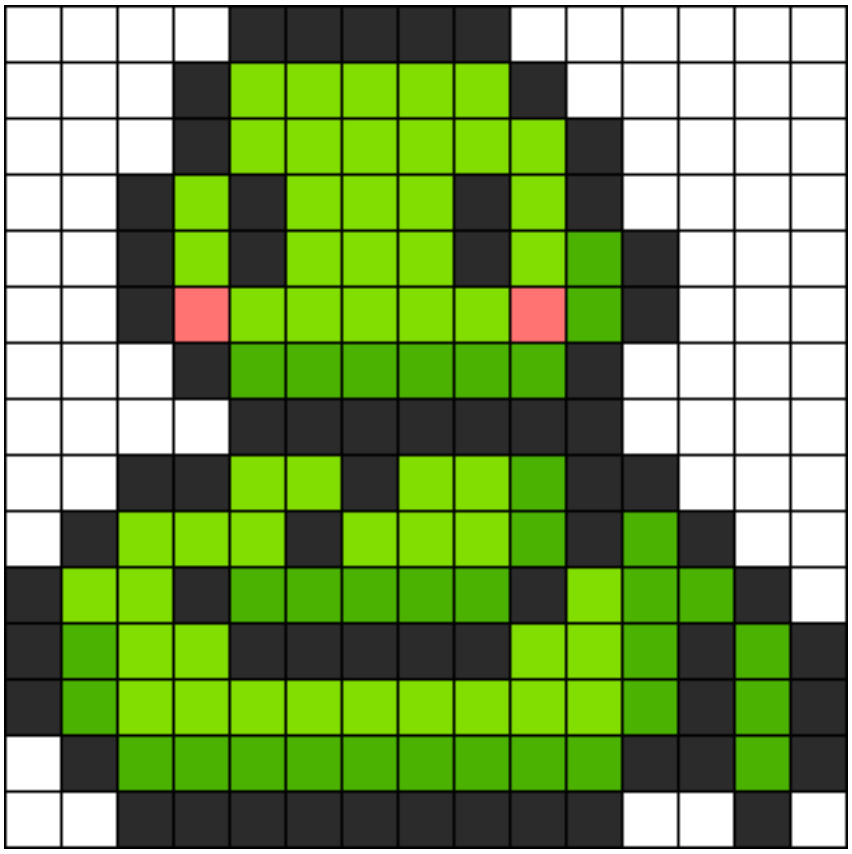
To create an attribute for a class, you don't have to declare it in the class definition - they are like local variables in that they spring into existence when they're assigned to. In this way, we can create a `counter` attribute for our `player_1` object. What does the following program output then?

```
player_1.counter = 0

while (player_1.counter < 10):
    player_1.counter += 1

print(player_1.counter)
```

There are many more nuances and useful characteristics of classes that we don't talk about in this tutorial. If you do want to learn more, look at Python documentation.

To give you another example of using classes, here is a Snake class that could be used for a micro:bit version of the Snake game (you'll know if you ever had a Nokia).

```
class Snake:

        def __init__(self):
            self.x_position = 0
            self.y_position = 0
            self.direction = "w"

        def move_snake(self, x_position, y_position, direction):
            self.x_position = x_position
            self.y_position = y_position
            self.direction = direction

        def show_snake(self):
            display.set_pixel(self.x_position, self.y_position, 9)
            sleep(600)
            display.set_pixel(self.x_position, self.y_position, 0)

# Create an instance of a Snake object
python = Snake()

# Access its position on x axis and print
print(python.x_position)

# Move python to the right
python.move_snake(python.x_position + 1, python.y_position)
```



CHAPTER 20

Morse Code

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 ––––·

Fig. 1: 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 translated to Morse code by an operator at the sending end using a a telegraph key like the one pictured here.

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':'---.',
                ...  }
```

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.

CHAPTER 21

Caesar Cipher

21.1 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:

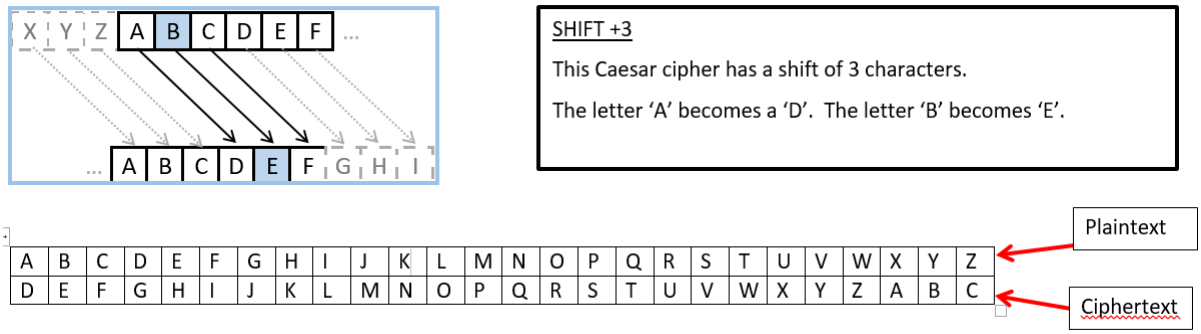


Fig. 1: Source: https://upload.wikimedia.org/wikipedia/commons/f/fa/Ascii-proper-color.svg

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.

This works because characters in Python are encoded as numbers. One of the most popular (and smallest) systems is ASCII (American Standard Code for Information Interchange). However, looking at the table below you can see that it only includes latin and some special characters. This is why Python's (and most languages') native encoding system is UTF-8, which is also backwards compatible with ASCII.

ASCII Table

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	|
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

But hold on, there is one more thing that we need to do. Since we will only accept uppercase letters A-Z, we need to make sure we only move within its boundaries (since adding 4 to Z would land us on '^' in ASCII).

ascii_char = ord(plaintext_char) + 4

if ascii_char > ord('Z') ascii_char = ascii_char - 26

encrypted_char = chr(ascii_char)

Now try encrypting and then decrypting some messages of your own. You can try to decrypt the following text. Can you make your program recognize the correct plaintext on its own?

> S kw k csmu wkx... S kw k gsmuon wkx. Kx exkddbkmdsfo wkx. S drsxu wi vsfob rebdc. Rygofob, S nyx'd uxyg k psq klyed wi csmuxocc, kxn kw xyd cebo grkd sd sc drkd rebdc wo. S kw xyd losxq dbokdon kxn xofob rkfo loox, dryeqr S boczomd wonsmsxo kxn nymdybc. Grkd'c wybo, S kw kvcy cezobcsdsyec sx dro ohdbowo; govv, kd vokcd oxyeqr dy boczomd wonsmsxo. (S'w ceppsmsoxdvi onemkdon xyd dy lo cezobcdsdsyec, led S kw.) Xy, csb, S bopeco dy lo dbokdon yed yp gsmuonxocc. Xyg, iye gsvv mobdksxvi xyd lo cy qyyn kc dy exnobcdkxn drsc. Govv, csb, led S exnobcdkxn sd. S gsvv xyd, yp myebco, lo klvo dy ohzvksx dy iye zbomscovi gry sc qysxq dy ceppob sx drsc mkco pbyw wi gsmuonxocc; S uxyg zobpomdvi govv drkd S gsvv sx xy gki "wemu drsxqc ez" pyb dro nymdybc li bocoxdsxq drosb dbokdwoxd; S uxyg loddob drkx kxiyxo drkd li kvv drsc S kw rkbwsxq yxvi wicovp kxn xy yxo ovco. Led cdsvv, sp S nyx'd qoddbokdon, sd sc yed yp gsmuonxocc. Wi vsfob rebdc; govv, drox vod sd rebd ofox gybco!
>
> – P. Nycdyiofcui, Xydoc Pbyw Exnobqbyexn

CHAPTER 22

Substitution Cipher

22.1 Description

The substitution cipher is deceptively easy. Messages are encrypted using a key which is created in advance. You make the key by changing positions of letters in the alphabet:

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

To be able to encode and decode messages using a substitution cipher, you will need to create your the key used to generate ciphertext and store it. A dictionary might be a good data structure for this purpose. A Python dictionary for the substitution cipher above would look something like this:

```
key =  {'A':'V',
        'B':'J',
        'C':'Z',
        'D':'B',
          ...   }
```

Try encrypting and decrypting messages of your own and if you feel up to the challenge, try to decrypt the text below. To save you the effort, the key above does not apply to the message. Try to think of (or look for) language properties or techniques that could help you find the key.

> O zay vcur xzfyozx fr nb gorfuj rdfr lftm rcttat ydokd youu zcgct ucfgc nc rouu O, raa, fn fr tcjr; "fkkolczrfuub" at ardctyojc. Pctjpflozx rdc yolay rdfr nb kazzchoaz yord dct dpjefzl'j "rckdzokfu nfrrctj" yfj jpvvokoczr ra czroruc nc ra doj nfzpjktowr, O eatc rdc lakpnczr fyfb fzl ecxfz ra tcfl or az rdc Uazlaz eafr. Or yfj f jonwuc, tfneuozx rdozx—f zfogc jfouat'j cvvatr fr f wajr-vfkra loftb—fzl jrtagc ra tckfuu lfb eb lfb rdfr ufjr fyvpu gabfxc. O kfzzar frrcnwr ra rtfzjktoec or gctefron oz fuu orj kuaplozcjj fzl tclpzlfzkc, epr O youu rcuu orj xojr czapxd ra jdcy ydb rdc japzl av rdc yfrct fxfozjr rdc gcjjcu'j jolcj eckfnc ja pzczlptfeuc ra nc rdfr O jrawwcl nb cftj yord karraz. Qadfzjcz, rdfzm xal, lol zar mzay iporc fuu, cgcz rdapxd dc jfy rdc korb fzl rdc Rdozx, epr o jdfuu zcgct juccw kfunub fxfoz ydcz O rdozm av rdc dattatj rdfr uptm kcfjcucjjub ecdozl uovc oz ronc fzl oz jwfkc, fzl av rdajc pzdfuuaycl eufjwdcnocj vtan culct jrftj ydokd ltcfn eczcfrd rdc jcf, mzayz fzl vfgaptcl eb f zoxdrnftc kpur tcflb fzl cfxct ra uaajc rdcn az rdc yatul ydczcgct fzardct cftrdipfmc jdfuu dcfgc rdcot nazjrtapj jrazc korb fxfoz ra rdc jpz fzl fot.
>
> —D.W. Uagcktfvr, Kfuu Av Krdpudp

CHAPTER 23

Vigenère Cipher

This cipher, also called 'le chiffre indéchiffrable', was first described by Giovan Battista Belazzo. Although the concept is easy to understand, the cipher resisted breaking for three centuries until Friedrich Kasiski introduced a first succesful general attack.

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

Fig. 1: Vigenère table (Source: https://tinyurl.com/yxmbt48f)

It works similarly to Caesar substitution cipher, since it's also based on shifting aplhabet positions. This time instead of shifting all letters by the same value, we choose a keyword which determines a different value for each letter in the plaintext.

As an example, suppose that this is our plaintext:

```
ATTACKATDAWN
```

Then you choose a keyword (such as snake) and repeat it for each row of plaintext:

```
SNAKESNAKESN
```

Then you use each letter of the key to determine the shift of every letter of the plaintext:

```
Plaintext:  ATTACKATDAWN
Key:        SNAKESNAKESN
Ciphertext: SGTKGCNTNEOA
```

You can try to encipher and decipher your own messages. If you're up for a challenge, try to find the key and decrypt this ciphertext enciphered using Vigenère cipher (hint: it's not easy):

> Wckl'g jwym avr Tlvfqydnxkwn Vyehqgxat oof im lhm nqmna oyrmavz. Vi qtfg gwym hzpdfhs wf p ahschgjxzg idjtawyt jbxivs dhyars zr avr ucktsaiympca dd lbungq tur naqh ucgtq bag vcrhewpprbuu rudxjh bc axyhnxl vhfodl--uhgrs jbms sdpfz.
>
> Hut Fbaquwgdlf'f Vsbks Gd Ral Unayqf oyhm flbgxmgz oyrmavz. Vi qtfg gwym avr qcla rexld pb rmglasarc bz hut ntu unayvawp vyknzr qjtzhrg. Gm zolh rahh gwc xmtrrr hm o cpl zhznrrbj ungeel pypqmlf vh jbrs uptbuu ldsk ifnxll zanhfxk chi zr h gyxax vt ytkhu kepnilr edsgk o yppzl ubab uywpz. Ral uhxbx hzfd rxszf nmn vb jwgvo dyplxag gwc ulgg eyg noypampq tppzss oaylaseh ykl avmcw, ocj bsvo mbj atu skecva hb eyr mce dlx hbq lfta jbasgaoen mknoaxxtawbcq xewfi rh osye whb frwyupzviyml osickdoesq.
>
> Mos Uxrvovvzck'z Uhxbx Ac Gwc Zhznmw llzyh ptavrg zxahrg rahb gwc Xuqlrjhwsqxy Zhznrrbjo.
>
> —Smnnznh Ywhaf, Wgmjvuxixy'g Txswl Hb Ifx Noypvr

CHAPTER 24

Bop-it

24.1 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. 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.

CHAPTER 25

Consonant or Vowel?

25.1 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.

CHAPTER 26

Catch the Eggs

26.1 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.

CHAPTER 27

Sprit Level

27.1 Description

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

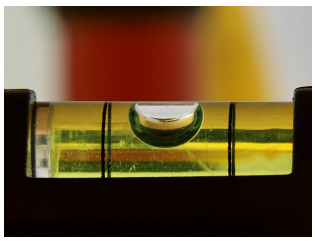


Fig. 1: Image source: Wikipedia

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.

CHAPTER 28

Theremin

28.1 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. 1: Image: Leon Theremin, source: Wikipedia

28.1.1 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
```

(continues on next page)

(continued from previous page)

```
    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)
```

CHAPTER 29

Send a Message

29.1 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.

CHAPTER 30

Programming Micro:bit Using Other Languages

As mentioned in the introduction, micro:bit can also be programmed using JavaScript and C/C++.

30.1 JavaScript

Online editor and documentation for JavaScript can be found at micro:bit's page.

30.2 C/C++

Micro:bit is programmable using Mbed online compiler. You can watch their getting started video for a basic set up.

CHAPTER 31

Command Line Interface

In the same way you use a graphical user interface (GUI) to interact with your computer using an arrow by clicking, it's possible to do the same using a command line interface (CLI) by issuing commands. You will be required to work using CLI from the beginning of your course, hence, it might be useful for you to become familiar with it in advance.

Fig. 1: Typing 'help' into Windows Command Prompt shows all the commands and what they do.

CLI used to be the usual way of interacting with programs and OS in the past, as it has much fewer system requirements. Now the majority of users use the GUI, while CLI has become a tool of the advanced users. Although it may look daunting and mysterious at the beginning, it's a very efficient tool for carrying out configuration and other tasks, since its functionality goes beyond that of a GUI.

The way you would use your OS CLI depends on your system, so it won't be covered in this tutorial. However, we encourage you to go and explore the basics (or advanced concepts) on your own.

Python Module Index

m

Index

M