technoteach technocamps

UNDEB EWROPEAIDD
EUROPEAN UNION
Llywodraeth Cymru
Welsh Government
Cronfa Gymdeithasol Ewrop
European Social Fund

Prifysgol Abertawe Swansea University
CARDIFF UNIVERSITY PRIFYSGOL CAERDYDD
PRIFYSGOL BANGOR UNIVERSITY
Cardiff Metropolitan University | Prifysgol Metropolitan Caerdydd
i.t.wales
PRIFYSGOL ABERYSTWYTH UNIVERSITY
Prifysgol Glyndŵr Wrecsam | Wrexham Glyndŵr University
University of South Wales Prifysgol De Cymru

Micro:Bits

Python

Python

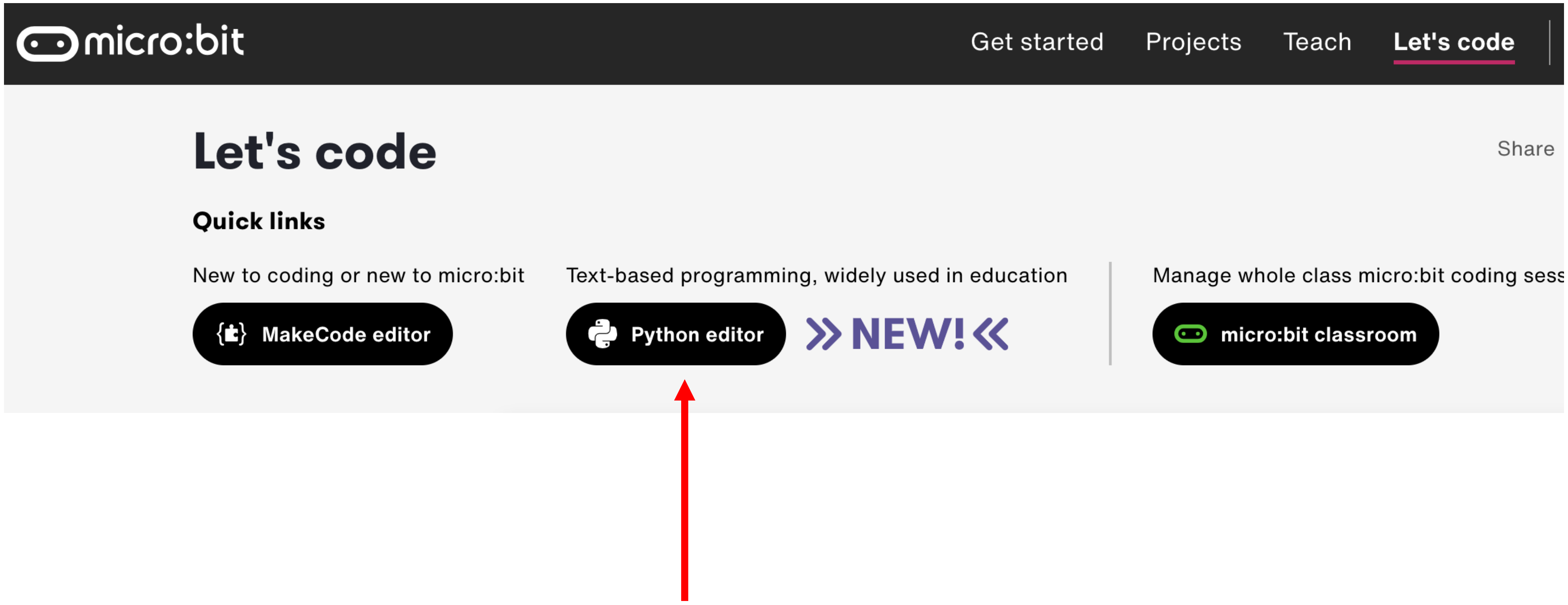
The Python Language

In the early 1990's, Guido van Rossum designed the Python programming language

Other programming languages were optimised for writing large, complex programs – van Rossum wanted his to be:

- Quick for writing simple programmes
- Easy to modify existing programmes
- Simple with straightforward syntax
- Easy to learn

micro:bit Python



Let's code

Quick links

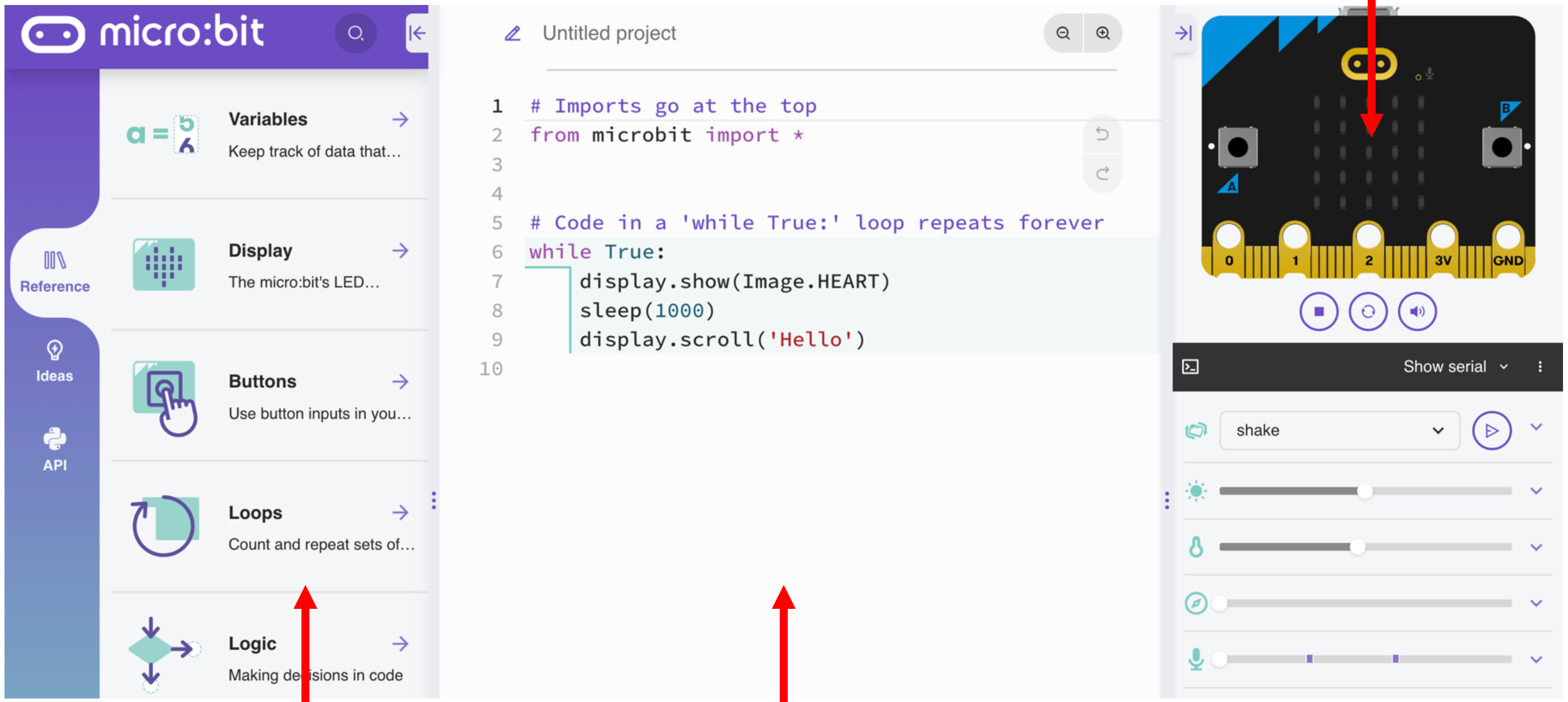
New to coding or new to micro:bit

Text-based programming, widely used in education

Manage whole class micro:bit coding sess

micro:bit Python

Simulator



```
# Imports go at the top
from microbit import *


# Code in a 'while True:' loop repeats forever
while True:
    display.show(Image.HEART)
    sleep(1000)
    display.scroll('Hello')
```

Code Menu

Code Window

Syntax in Python

What happens if you…

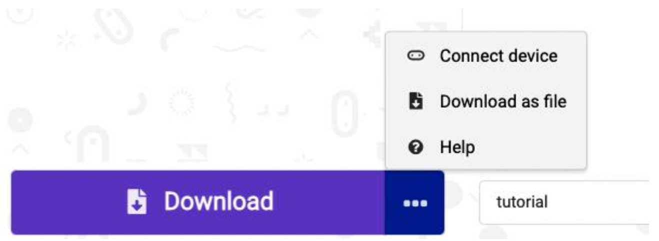
• Misspell a word:	primpt("Hello World!")
• Use wrong case:	Print("Hello World!")
• Leave out quotes:	print(Hello World!)
• Mismatch quotes:	print("Hello World!’)
• Don’t match brackets	print(‘Hello’

Try it to see what error messages are generated

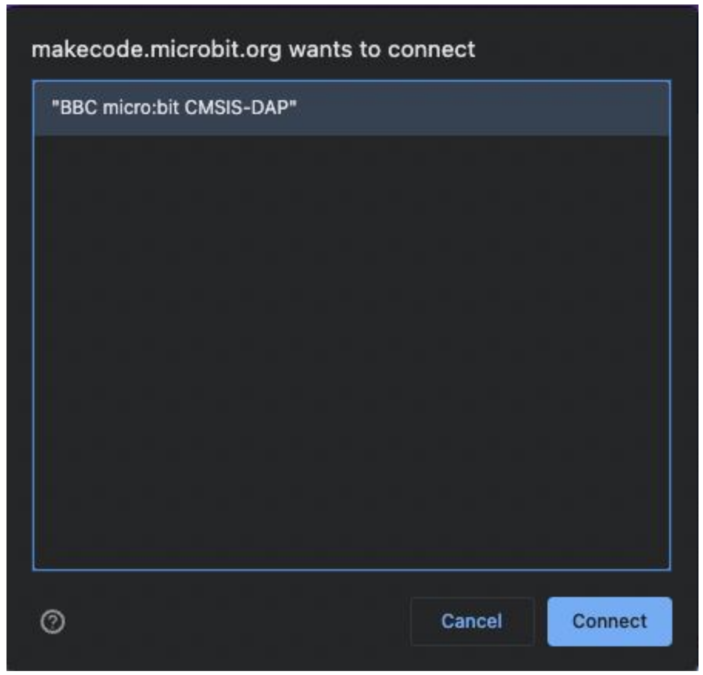
Connecting the micro:bit

1. Plug the micro:bit into your computer

2. In the bottom left of your screen, click the 3 dots next to 'Download', then click 'Connect Device'



3. Follow the on-screen instructions until you see this popup



makecode.microbit.org wants to connect

"BBC micro:bit CMSIS-DAP"

4. Click the name of your device (it should be the only option)

5. Click connect

Activity:
Smile!

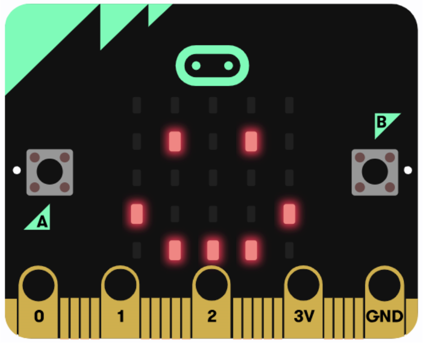
Smile!

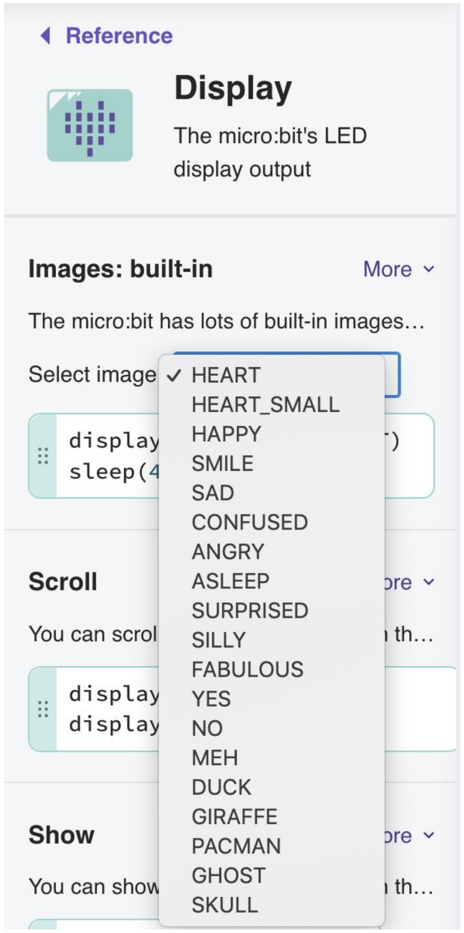
Let's start by displaying an image:

1. Start by deleting the starting code, except for the "from micro:bit import *"
2. Click on the **display** section
3. Select one the smiling face from the **Select image** dropdown menu
4. Drag the code to the code window
5. Click on send to micro:bit

What happens to the micro:bit?

Try changing the image!





Reference

Display

The micro:bit's LED display output

Images: built-in More

The micro:bit has lots of built-in images…

Select image ✓ HEART

```
display
sleep(4
```

Scroll

You can scrol

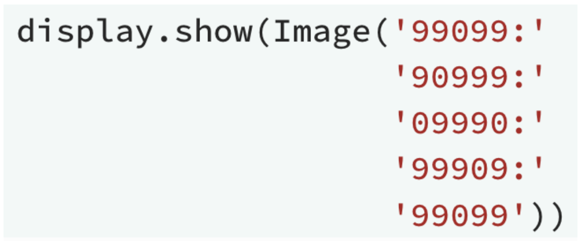
```
display
display
```

Show

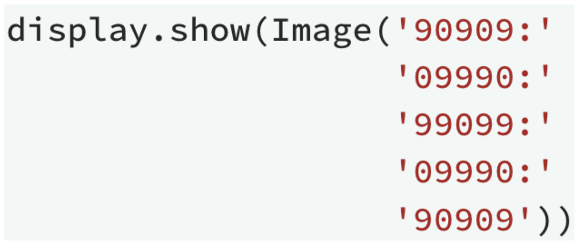
You can show

Custom Icons

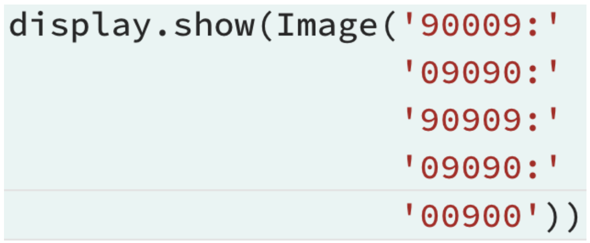
- We can customize exactly which LED lights light up
- This can be used to create your own icons
- Drag the **Images: make your own** block to your code
- Change the numbers to create your own icons. The number represents the intensity and location of the light
- Try the examples below or see which other patterns or icons you can make



```
display.show(Image('99099:'
                   '90999:'
                   '09990:'
                   '99909:'
                   '99099'))
```



```
display.show(Image('90909:'
                   '09990:'
                   '99099:'
                   '09990:'
                   '90909'))
```



```
display.show(Image('90009:'
                   '09090:'
                   '90909:'
                   '09090:'
                   '00900'))
```

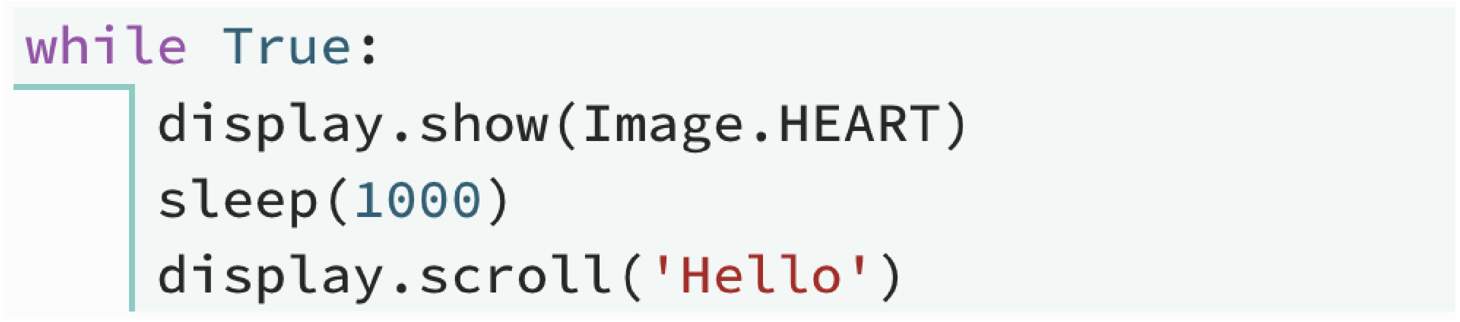
Activity: Changing Faces

Loops

- Loops allow us to repeat commands
- They can be repeated forever, for a certain number of times, or for a given condition
- These commands are found in the **Loops** section

Default Loop

- We have a default loop command when we start the micro:bit project – the **while true** loop
- This acts as a "forever loop" and runs a set of commands until the micro:bit is unplugged or reset



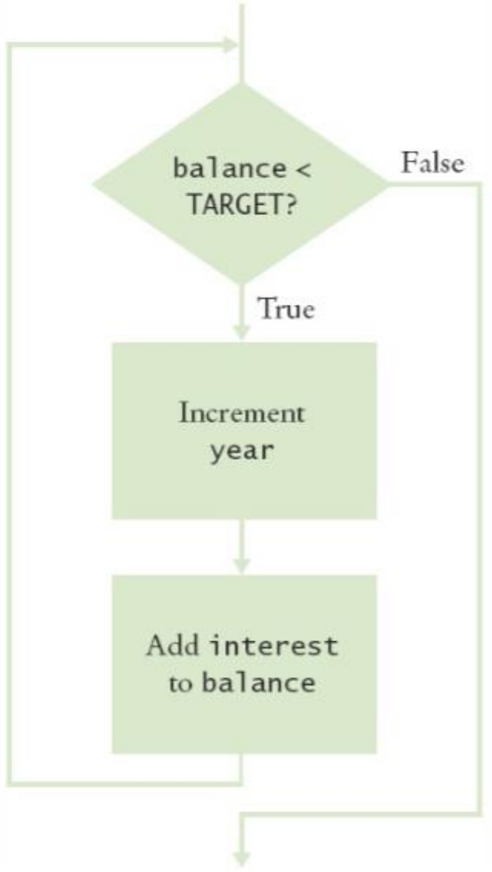
```python
while True:
    display.show(Image.HEART)
    sleep(1000)
    display.scroll('Hello')
```

While Loop

- A while loop repeats instructions while a condition is true

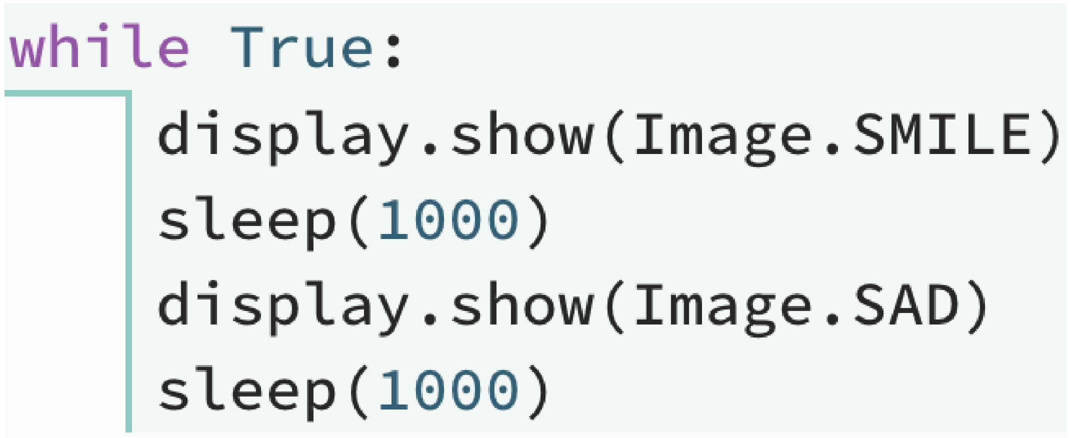
-
```
while x < y:
    x = x + 1
```

- This makes while loops very useful for input validation



Changing Faces

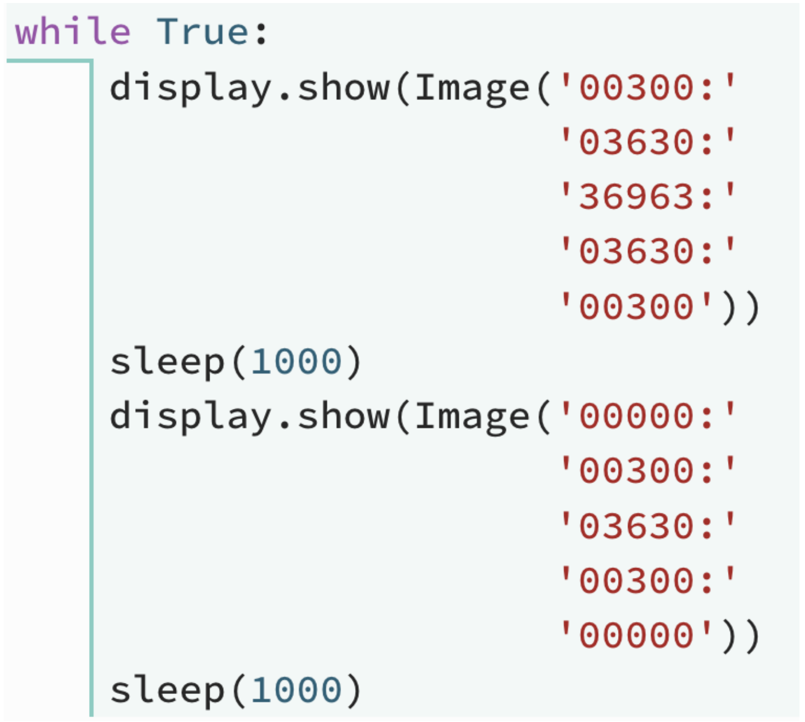
- We have already created a program which shows an icon on the micro:bit
- Now we can extend our program to show an animation
- Add a second icon after the first one
- What happens when you download the program onto the micro:bit?



```
while True:
    display.show(Image.SMILE)
    sleep(1000)
    display.show(Image.SAD)
    sleep(1000)
```

Make Your Own Animation

- So far, we have used the default icons to create our animations
- Try using the custom icon command you saw earlier to create your own animation



```
while True:
    display.show(Image('00300:'
                       '03630:'
                       '36963:'
                       '03630:'
                       '00300'))
    sleep(1000)
    display.show(Image('00000:'
                       '00300:'
                       '03630:'
                       '00300:'
                       '00000'))
    sleep(1000)
```

Activity: Positivity!

Positivity Generator

- A positivity generator is a device that displays a random positive statement when prompted
- It works like a magic eight ball that shows you a random answer when you shake it
- To do this, we need to use variables to define the quotes

Variables

- Variables are items that can be remembered and changed by the micro:bit

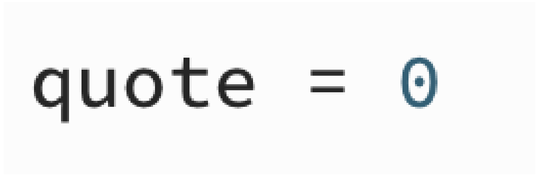
- They can take different forms such as a number or a text

- We can change their value and use them in many ways in our code, but first we need to define them

Creating Variables

- To create a variable, simply give it a name and assign a value to it using the ‘=‘ sign

- Lets start by creating a variable for the quote. We will assign the value 0 to it



```
quote = 0
```

Conditionals

- We want the positivity generator to become active when we shake it. To do this we need to use conditionals

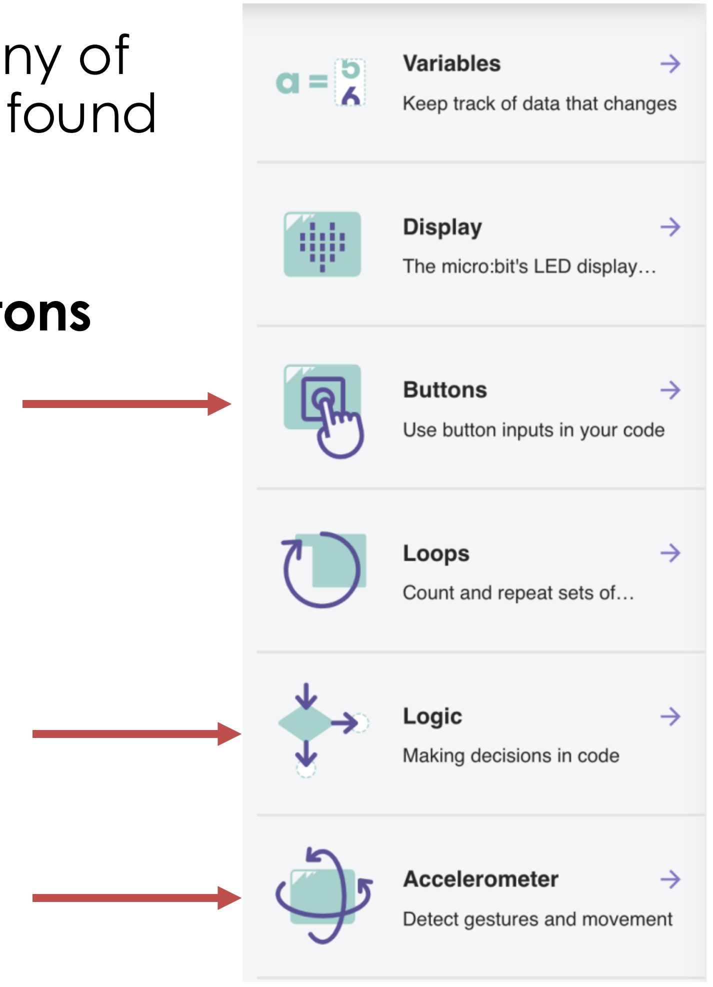
- Conditionals help us perform different actions based on different conditions

- For example:
  - **If** my homework is done, **then** I can go outside to play
  - **If** I have eaten my dinner, **then** I can have dessert

- Can you think of some more examples?

Logic

- In the micro:bit Python editor, many of these conditional statements are found in the **Logic** section
- As you have seen, there are also conditional statements in the **Buttons** and **Accelerometer** sections

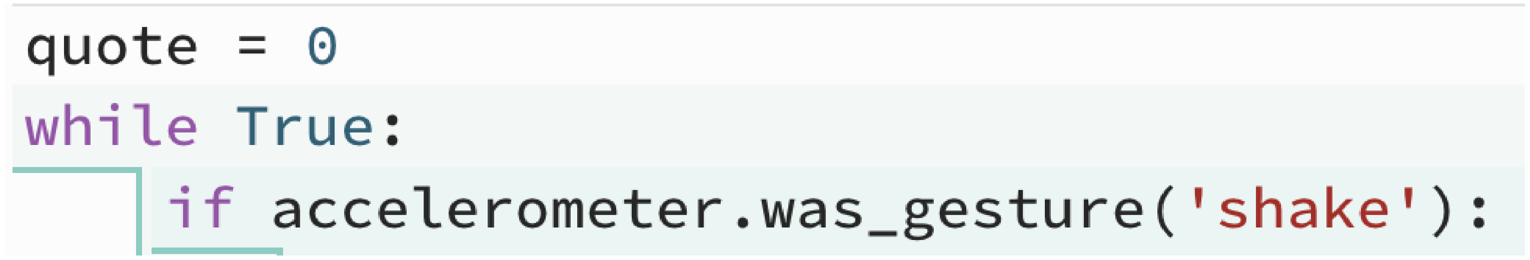

Variables
Keep track of data that changes
Display
The micro:bit's LED display…
Buttons
Use button inputs in your code
Loops
Count and repeat sets of…
Logic
Making decisions in code
Accelerometer
Detect gestures and movement

Positivity Generator

Let's make our positivity generator!

1. First, drag the **shake gesture** command from the **accelerometer** section to your code window

Delete the **display.show()** command

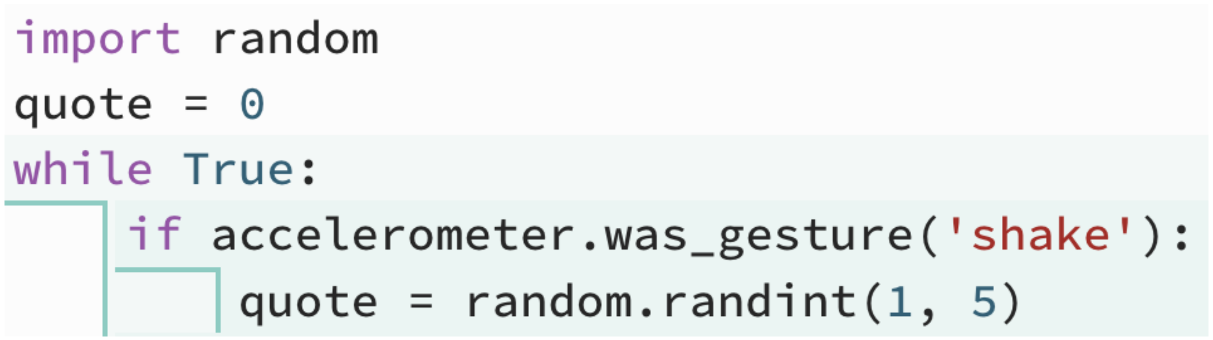


```
quote = 0
while True:
    if accelerometer.was_gesture('shake'):
```

Positivity Generator

Now we want the quote variable to be assigned a random number between 1 and 5. This will allow the micro:bit to randomly choose from 5 quotes

2. Import the random module
3. In the shake **if-statement**, use **random.randint(1, 5)** to assign a random number to your variable



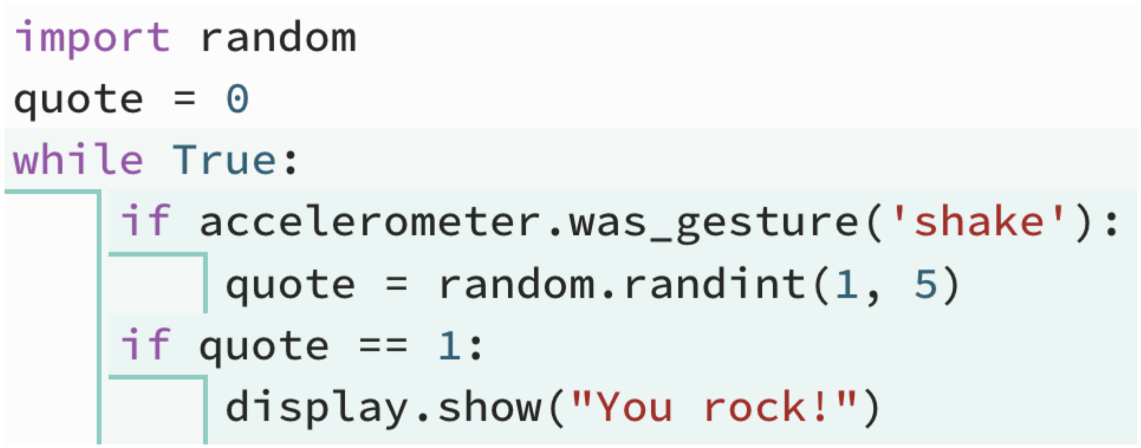
```python
import random
quote = 0
while True:
    if accelerometer.was_gesture('shake'):
        quote = random.randint(1, 5)
```

What do you think the next step will be?

Positivity Generator

When we shake the device, the variable will be assigned a random integer between 1 and 5. But how will the micro:bit know what to do with each number? We need more conditionals

4. Add an **if-statement** to tell the micro:bit what to do if the **quote** variable has the value 1

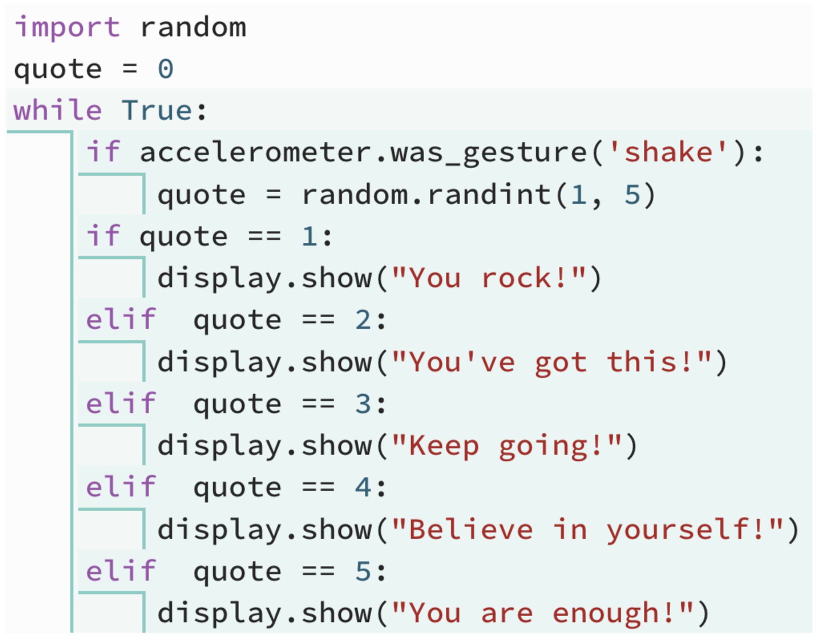


```
import random
quote = 0
while True:
    if accelerometer.was_gesture('shake'):
        quote = random.randint(1, 5)
    if quote == 1:
        display.show("You rock!")
```

5. Now do the same for values 2, 3, 4, and 5

Positivity Generator

Your code should look something like this:



```
import random
quote = 0
while True:
    if accelerometer.was_gesture('shake'):
        quote = random.randint(1, 5)
    if quote == 1:
        display.show("You rock!")
    elif  quote == 2:
        display.show("You've got this!")
    elif  quote == 3:
        display.show("Keep going!")
    elif  quote == 4:
        display.show("Believe in yourself!")
    elif  quote == 5:
        display.show("You are enough!")
```

Try running the code. Does the positivity generator work? Are there any problems with it?

Positivity Generator

When we use our positivity generator, it keeps displaying the positive message on repeat. It would be more helpful if it only showed the message once, and then we could use it again to generate a different positive message

How do you think we can do this? Have a look through the **Display** section for any commands we could use

Positivity Generator

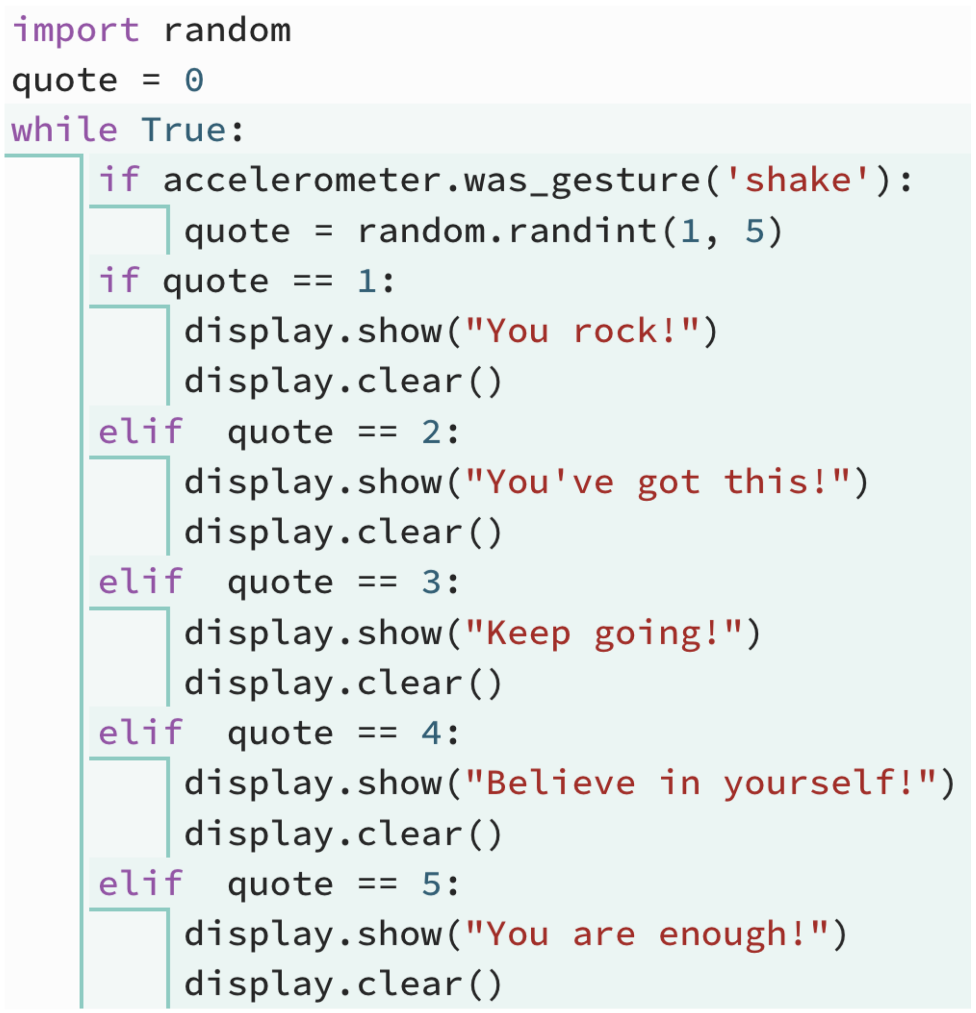
When we use our positivity generator, it keeps displaying the positive message on repeat. It would be more helpful if it only showed the message once, and then we could use it again to generate a different positive message

How do you think we can do this? Have a look through the **Display** section for any commands we could use

The **display.clear()** command will clear the micro:bit screen once the message has been displayed once. Add this to all your if-statements

Positivity Generator

Your code should now look like this:

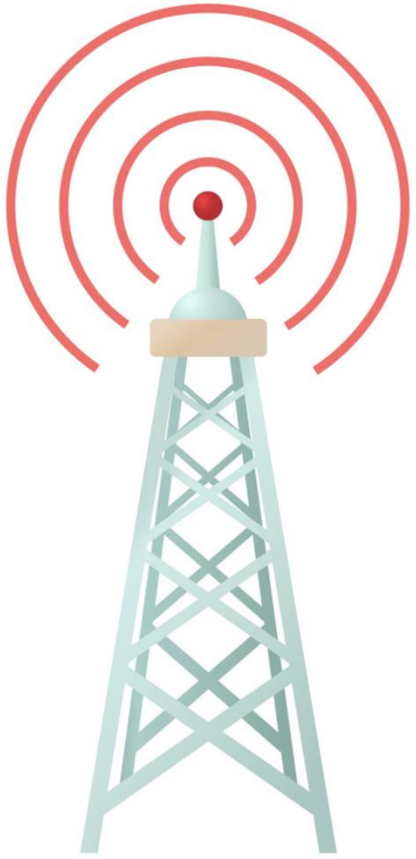


```python
import random
quote = 0
while True:
    if accelerometer.was_gesture('shake'):
        quote = random.randint(1, 5)
    if quote == 1:
        display.show("You rock!")
        display.clear()
    elif  quote == 2:
        display.show("You've got this!")
        display.clear()
    elif  quote == 3:
        display.show("Keep going!")
        display.clear()
    elif  quote == 4:
        display.show("Believe in yourself!")
        display.clear()
    elif  quote == 5:
        display.show("You are enough!")
        display.clear()
```

Activity: Morse Code

Radio Communication

- Radio is a way of communicating through radio waves
- This means we can send information from one place to another over very long distances
- Can you think of something that might use radio?



Modules (Python Libraries)

- Python has a **standard library** of **modules** that must be included on any Python system

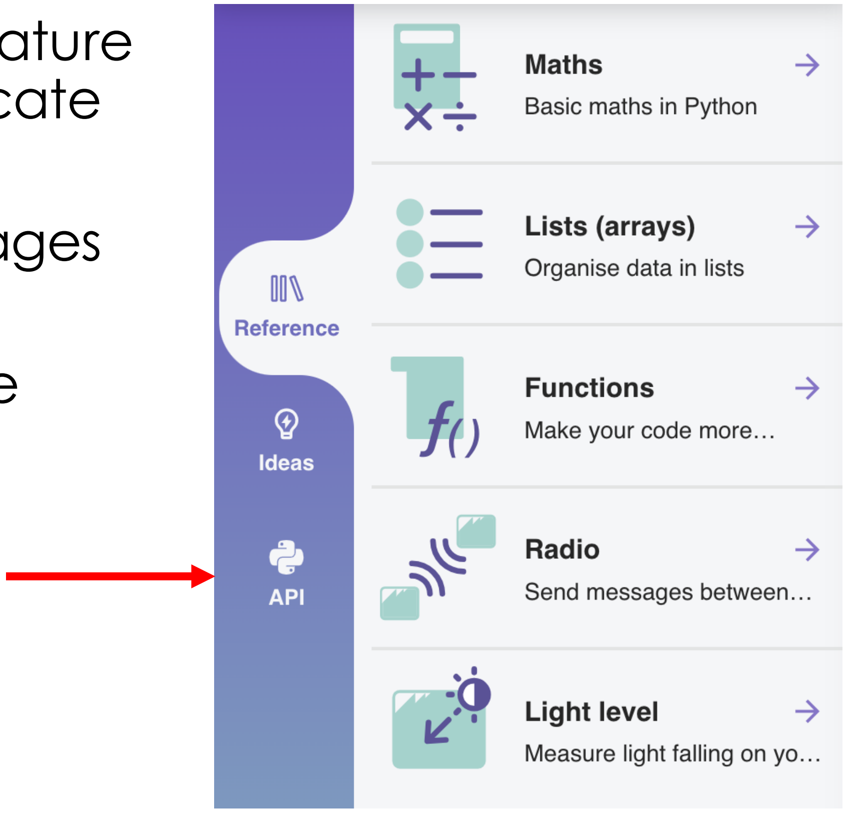
- You can import these **modules** or specific functions from them

- This allows Python to do much more without having to load lots of unnecessary functions

- The Micro:Bit Python Editor has an additional library of micro:bit specific modules – including the radio module

Radio Communication

- Micro:bits have a Bluetooth feature that allows them to communicate with each other
- We can use this to send messages between micro:bits
- The blocks we need for this are found in the Radio section



Maths Basic maths in Python

Lists (arrays) Organise data in lists

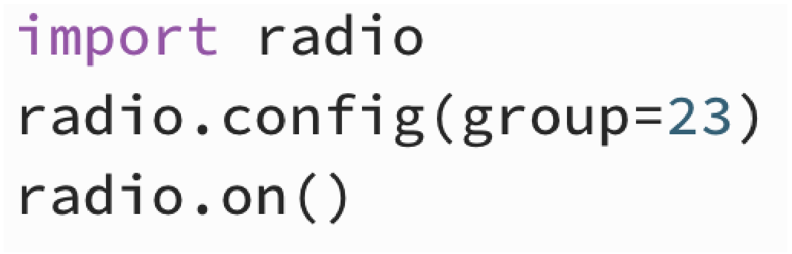
Functions Make your code more…

Radio Send messages between…

Light level Measure light falling on yo…

Radio Groups

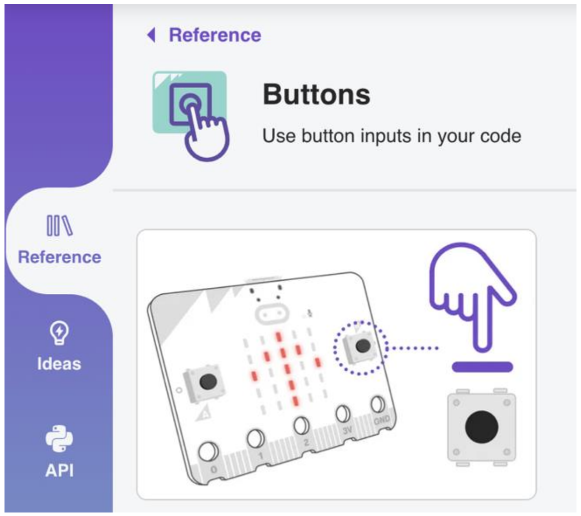
- Before we can send messages to each other we have to import radio and set the radio group
- Radio groups are like channels – any micro:bit using the same group will receive the message
- Try setting both micro:bits to the same radio group - this can be any number, just make sure they are the same!
- We also need to turn the radio on before we can use it. All these commands are found in the **Radio** section



```
import radio
radio.config(group=23)
radio.on()
```

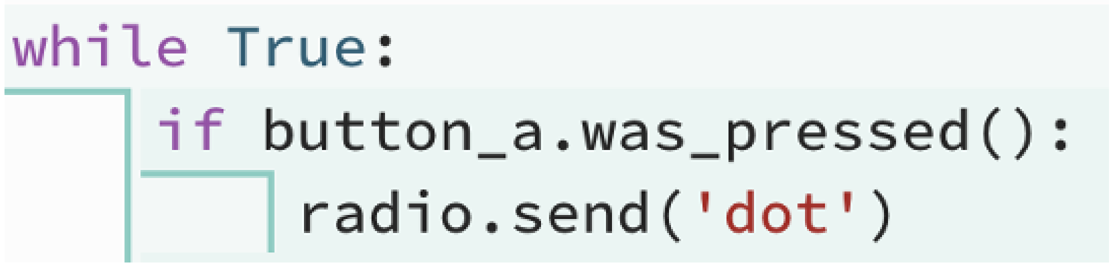
Events

- Each micro:bit has two buttons, A and B
- These buttons help us choose which action to take without reprogramming the micro:bit each time
- For example, we can show a happy face when we press button A and a sad face when we press button B
- The commands we need are found in the **Buttons** section



Sending...

- With the micro:bits in the same radio group, we can start sending messages between them
- We can use the **radio.send()** command in the **Radio** section to do this
- Try sending a string called "dot" when button A is pressed



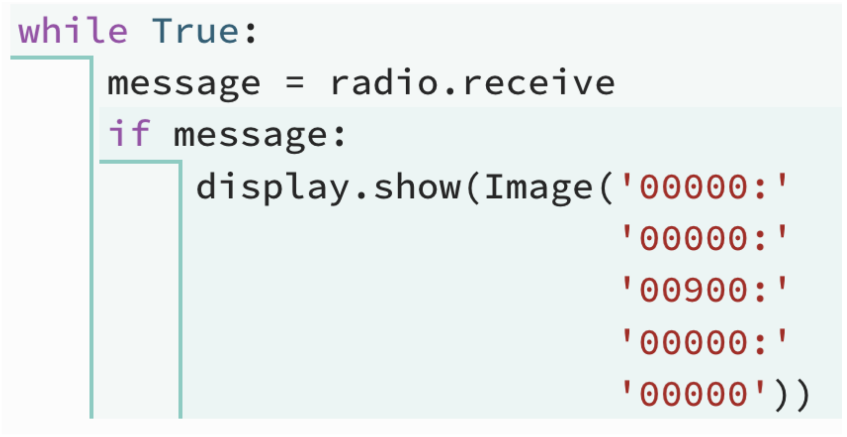
```
while True:
    if button_a.was_pressed():
        radio.send('dot')
```

… and Receiving

- Now that one micro:bit has sent a message, we need to tell the other micro:bit what to do with the message it received

- Drag the **Receive a message** block from the **Radio** section into your code window

- Change the display command to show a dot on the screen



```
while True:
    message = radio.receive
    if message:
        display.show(Image('00000:'
                           '00000:'
                           '00900:'
                           '00000:'
                           '00000'))
```

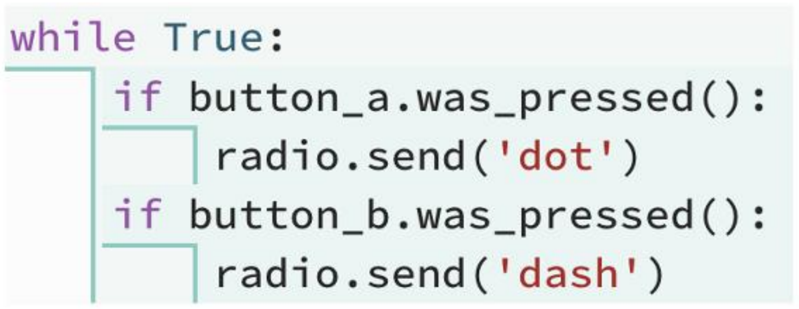
- Try running your codes, can the sender send a message to the receiver?

Sending Different Messages

We can use the buttons on the micro:bit to choose different messages to send.

Add code for a second button press to send a ‘dash’ string.

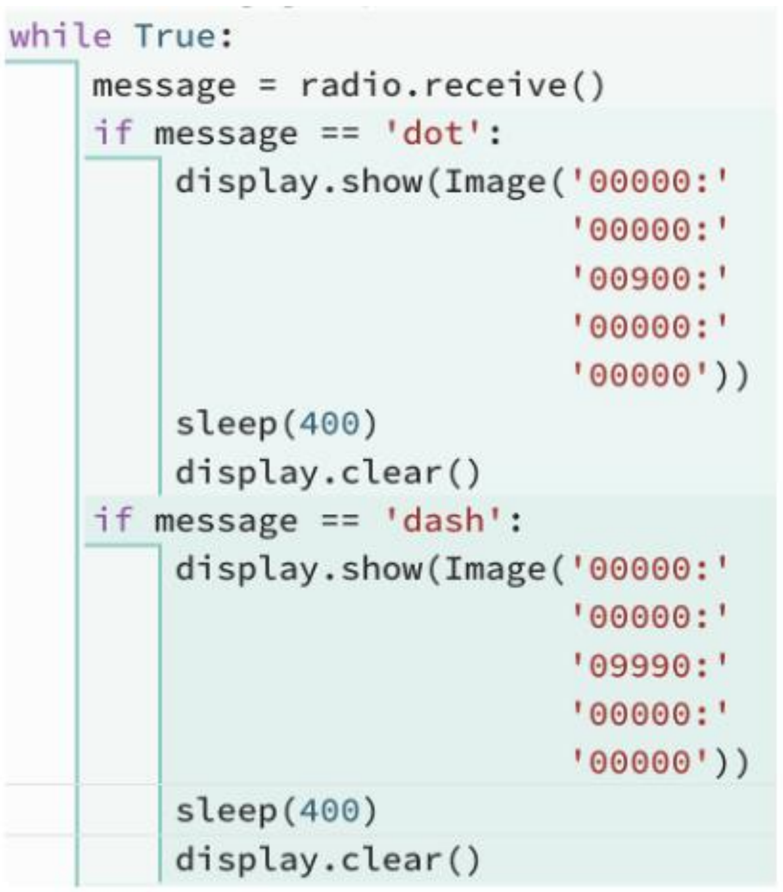
Try running the code to see what happens. Can you spot the problem? How do you think we can solve this?



```
while True:
    if button_a.was_pressed():
        radio.send('dot')
    if button_b.was_pressed():
        radio.send('dash')
```

Receiving Different Messages

- Now we can tell the micro:bit to display a different icon when receiving different messages
- Add another **if-statement** to the code
- Change the conditions to check if the message received is ‘dot’ or ‘dash’
- Display a dot or dash icon
- Try running the code to see what happens!

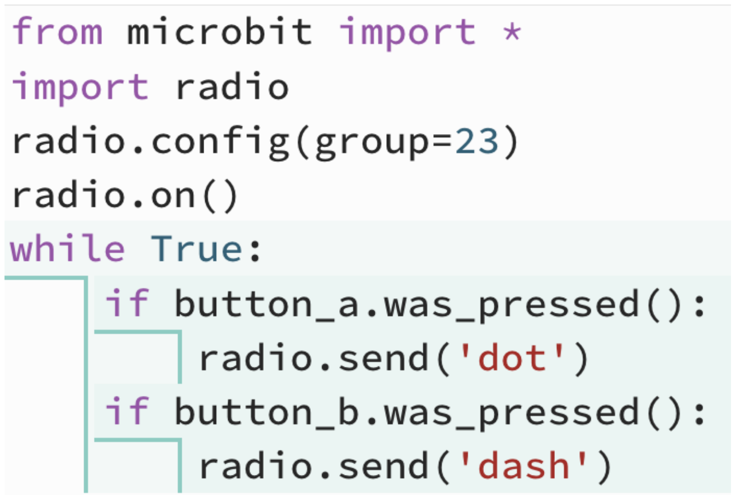


```
while True:
    message = radio.receive()
    if message == 'dot':
        display.show(Image('00000:'
                           '00000:'
                           '00900:'
                           '00000:'
                           '00000'))
        sleep(400)
        display.clear()
    if message == 'dash':
        display.show(Image('00000:'
                           '00000:'
                           '09990:'
                           '00000:'
                           '00000'))
        sleep(400)
        display.clear()
```

Micro:bit Morse Code

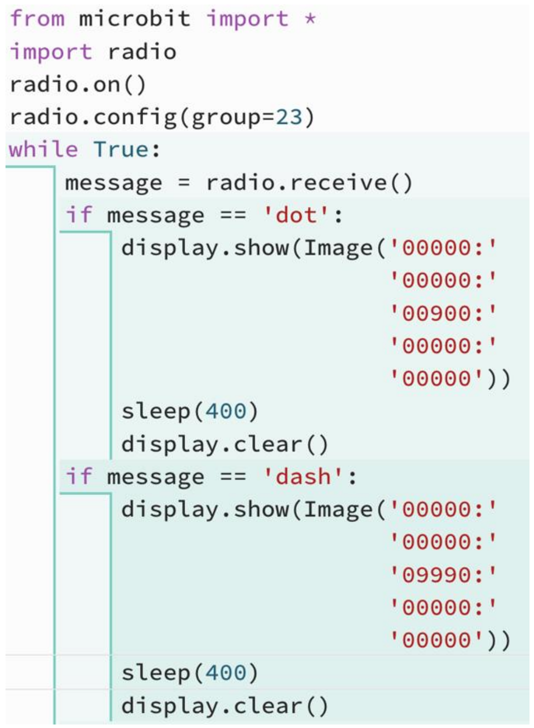
This is what your final code should look like:

Sender:



```
from microbit import *
import radio
radio.config(group=23)
radio.on()
while True:
    if button_a.was_pressed():
        radio.send('dot')
    if button_b.was_pressed():
        radio.send('dash')
```

Receiver:



```
from microbit import *
import radio
radio.on()
radio.config(group=23)
while True:
    message = radio.receive()
    if message == 'dot':
        display.show(Image('00000:'
                           '00000:'
                           '00900:'
                           '00000:'
                           '00000'))
        sleep(400)
        display.clear()
    if message == 'dash':
        display.show(Image('00000:'
                           '00000:'
                           '09990:'
                           '00000:'
                           '00000'))
        sleep(400)
        display.clear()
```

Dots and Dashes

- Try sending and translating some messages in morse code!

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

Activity: Melody

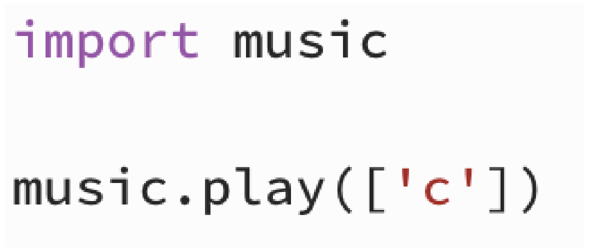
A Single Note

Let's start by programming the micro:bit to play a single note:

1. Delete the default code
2. Add the **Import music** command to get access to the music module
3. Add the **music.play([])** command
4. Inside the square brackets, add a note in quotation marks

What happens to the micro:bit?

Try changing the note

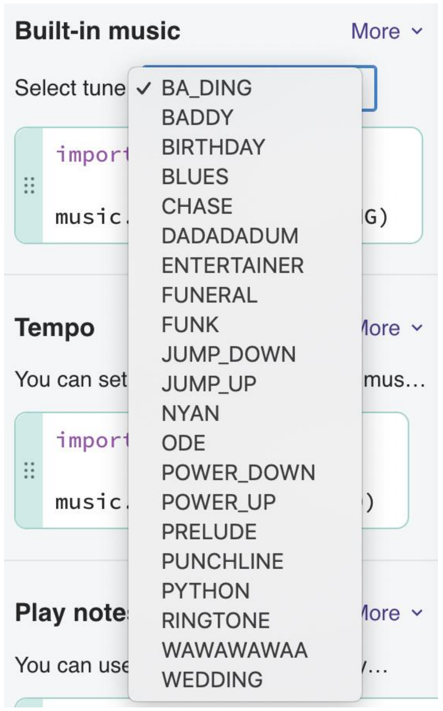


```
import music

music.play(['c'])
```

Melody

- MakeCode also has preset melodies that we can use
- Try playing one of these melodies by dragging the 'built-in music' code to the code window
- Try a few different melodies!

```python
import music

music.play(music.BA_DING)
```



Built-in music

Select tune

✓ BA_DING
BADDY
BIRTHDAY
BLUES
CHASE
DADADADUM
ENTERTAINER
FUNERAL
FUNK
JUMP_DOWN
JUMP_UP
NYAN
ODE
POWER_DOWN
POWER_UP
PRELUDE
PUNCHLINE
PYTHON
RINGTONE
WAWAWAWAA
WEDDING

Tempo

You can set … mus…

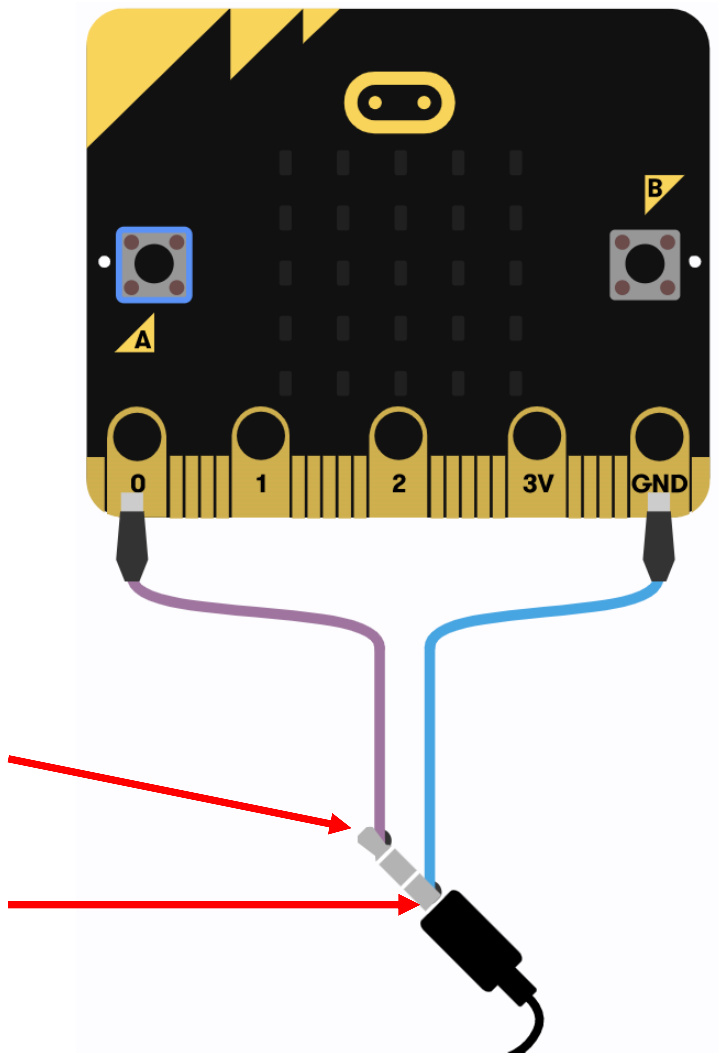
Play note…

You can use …

Activity: Headphones

Connecting headphones

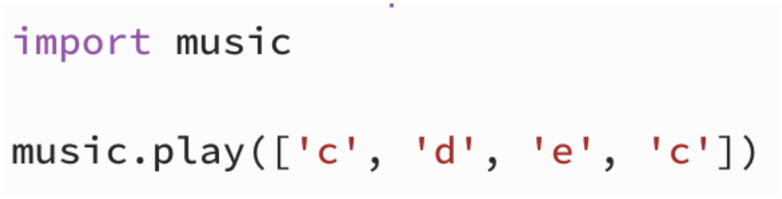
- With so many micro:bits playing music at the same time, it can be difficult to hear your own
- We can connect headphones by attaching them to the pins at the bottom of the micro:bit with crocodile clips



- Connect pin 0 to the tip of the headphone plug
- Connect the GND pin to the longer part of the headphone plug

Make Your Own Music

- We've learnt how to get our micro:bit to play a single note and a preset melody
- Now we can try and create our own melody
- Add several notes commands, one after the other, in the **music.play([])** command
- Be creative and see what melodies you can make!



```
import music

music.play(['c', 'd', 'e', 'c'])
```

technoteach technocamps

Activity: On Repeat

Extended Melody

- What if we wanted to play a much longer melody?
- Try building and running this code
- The code only plays a short tune but includes a lot of blocks
- Now imagine how many blocks we would need if we wanted to play a song that was twice as long, or 5 times, or 100 times!
- How could we do this without adding individual blocks for each note?

```
music.play(['c', 'd', 'e', 'c', 'c', 'd', 'e', 'c', 'e', 'f', 'g', 'e', 'f', 'g'])
```

Extended Melody

- Our song has repeated blocks of notes
- Instead of adding new blocks every time, we can use loops to repeat these segments

```
music.play(['c', 'd', 'e', 'c', 'c', 'd', 'e', 'c', 'e', 'f', 'g', 'e', 'f', 'g'])
```

$\underbrace{\texttt{'d', 'e', 'c', 'c', 'd', 'e', 'c'}}_{\text{C D E C repeated}}\quad\underbrace{\texttt{'e', 'f', 'g', 'e', 'f', 'g'}}_{\text{E F G repeated}}$

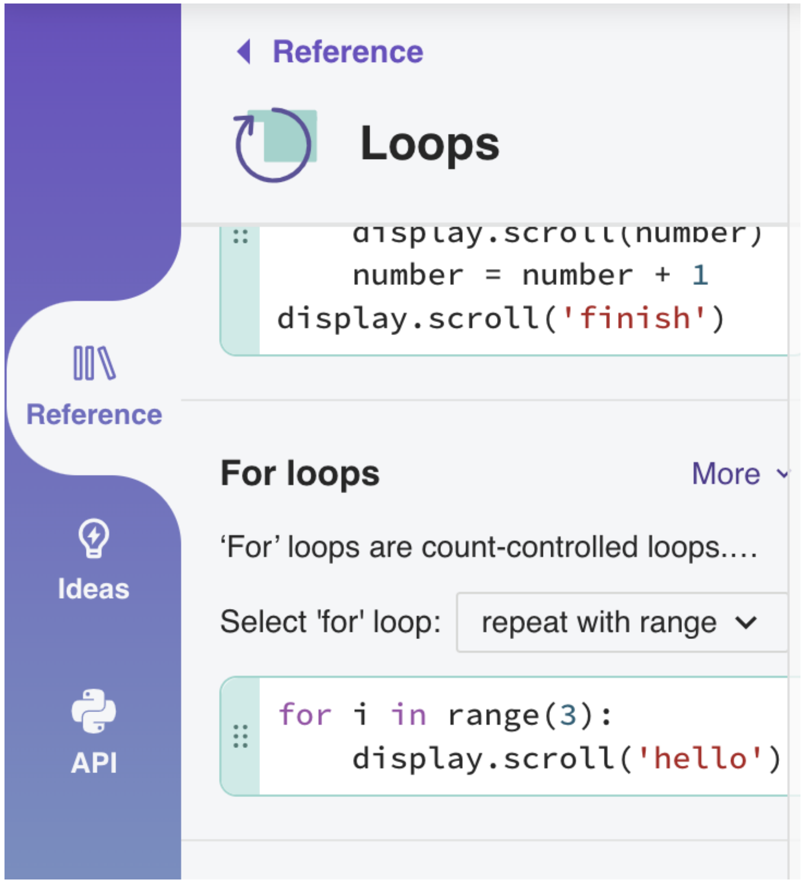
C D E C
repeated

E F G
repeated

technoteach technocamps

Specific Loops

- We can also have more specific loops that don't necessarily repeat forever
- For instance, we can use the **for loop** to repeat a code segment a certain number of times
- For our melodies this is useful if only certain parts of the melody need to be repeated. Then we don't need to add a new note in the code every time
- Let's try to make a tune!



```
        display.scroll(number)
        number = number + 1
display.scroll('finish')
```

'For' loops are count-controlled loops....

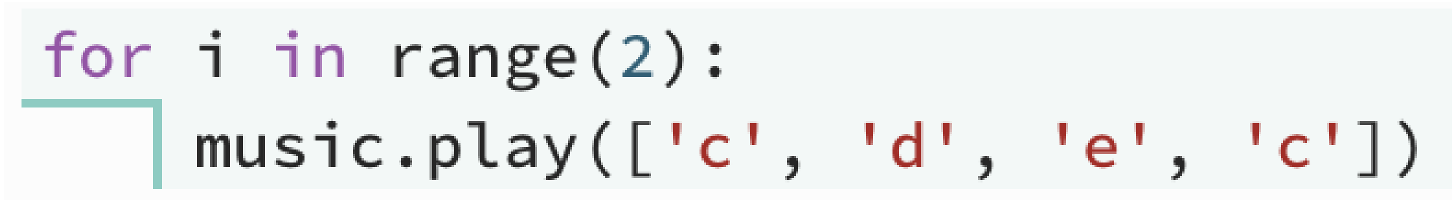
Select 'for' loop: repeat with range

```
for i in range(3):
    display.scroll('hello')
```

technoteach technocamps

Melody Loop

- Begin by drag and dropping the **for loop** into your code window and change the range to 2
- Remove the **display.scroll()** command and add a **music.play()** command.
- Add the first 4 notes of our tune (C, D, E, C)

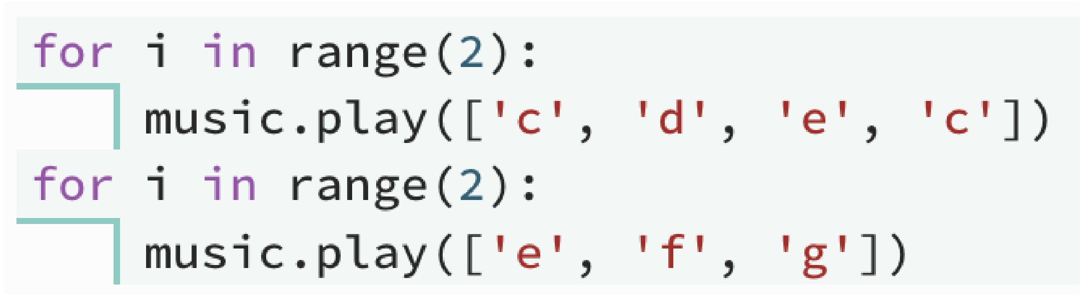


```
for i in range(2):
    music.play(['c', 'd', 'e', 'c'])
```

Melody Loop

- Add another **for loop** with some more notes
- Try running the code!

- Now the code is cleaner and more efficient. If we want to repeat it more times, we can just change the number on the loop



```
for i in range(2):
    music.play(['c', 'd', 'e', 'c'])
for i in range(2):
    music.play(['e', 'f', 'g'])
```