

Welcome to the labs!



Tamagotchi! - Micro:bits

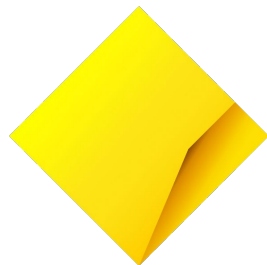


Thank you to our Sponsors!

Platinum Sponsor:



Gold Sponsor:



**Commonwealth
Bank**



Who are the tutors?

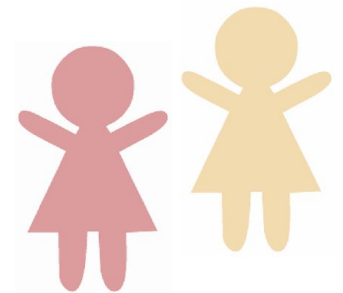
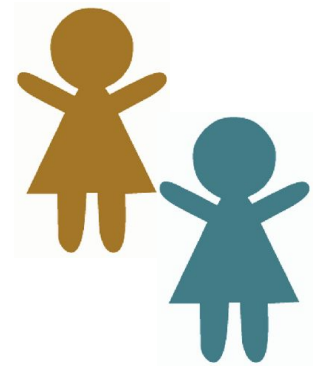


Who are you?



Two Truths and a Lie

1. Get in a group of 3-5 people
2. Tell them three things about yourself:
 - a. Two of these things should be true
 - b. One of these things should be a lie!
3. The other group members have to guess which is the lie



Log on

Log on and jump on the GPN website

girlsprogramming.network/workshop

Click on your location

Melbourne

Perth

Brisbane

Sydney

Burnie

Canberra

Adelaide



Tell us you're here!

Click on the
Start of Day Survey
and fill it in now!

Start of Day
survey



Log on

Click on your Room picture

You can see:

- A link to the **Workbook**
- These **Slides** (to take a look back on or go on ahead)
- Other helpful bits like a Cheatsheet to help you code



Using the workbook!

The workbooks will help you put your project together!

Each **Part** of the workbook is made of tasks!

Tasks - The parts of your project

Follow the tasks **in order** to make the project!

Hints - Helpers for your tasks!

Stuck on a task, we might have given you a hint to help you **figure it out!**

The hints have **unrelated** examples, or tips. **Don't copy and paste** in the code, you'll end up with something **CRAZY!**

Task 6.2: Add a blah to your code!

This has instructions on how to do a part of the project

1. **Start by doing this part**
2. **Then you can do this part**

Task 6.1: Make the thing do blah!

Make your project do blah

Hint

A clue, an example or some extra information to help you **figure out** the answer.

```
print('This example is not part of the project' )
```



Using the workbook!

The workbooks will help you put your project together!

Check off before you move on from a **Part!** Do some bonuses while you wait!

Checklist - Am I done yet?

Make sure you can tick off every box in this section before you go to the next Part.

Lecture Markers

This tells you you'll find out how to do things for this section during the names lecture.

Bonus Activities

Stuck waiting at a lecture marker? Try a purple bonus. They add extra functionality to your project along the way.



CHECKPOINT



If you can tick all of these off you're ready to move the next part!

- ☐ Your program does blah
- ☐ Your program does blob



★ BONUS 4.3: Do something extra!

Something to try if you have spare time before the next lecture!



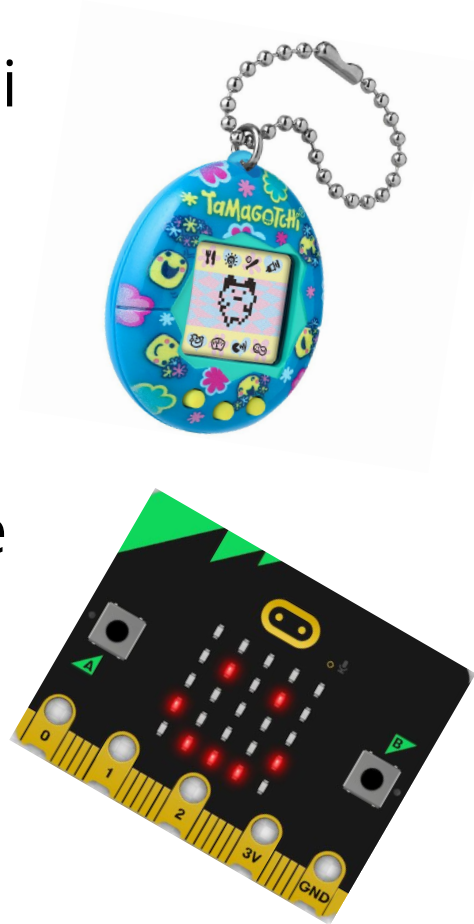
Today's project!

Tamagotchi- Micro:Bit



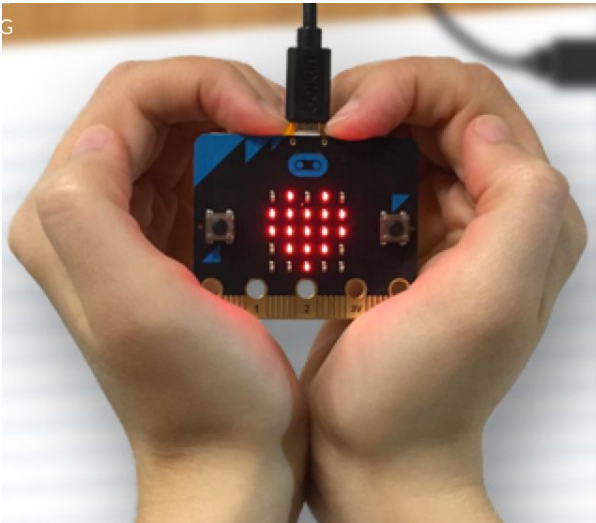
Tamagotchi

- You're going to make your own Tamagotchi electronic pet using a micro:bit
- Tamagotchi pets were a worldwide fad created in Japan in 1996
- Give your pet a name and write some code to feed it, play with it and let it sleep
- **Don't let it get hungry, bored or sleepy!**
- **Keep it alive, watch it grow and change**



Tamagotchi

Sadly you can't keep them at the end of the day. 😞



If you want one for home (maybe for christmas or your birthday!) they're about \$25 .

Find out where to buy them here:
<https://microbit.org/>

Intro to Micro:Bit

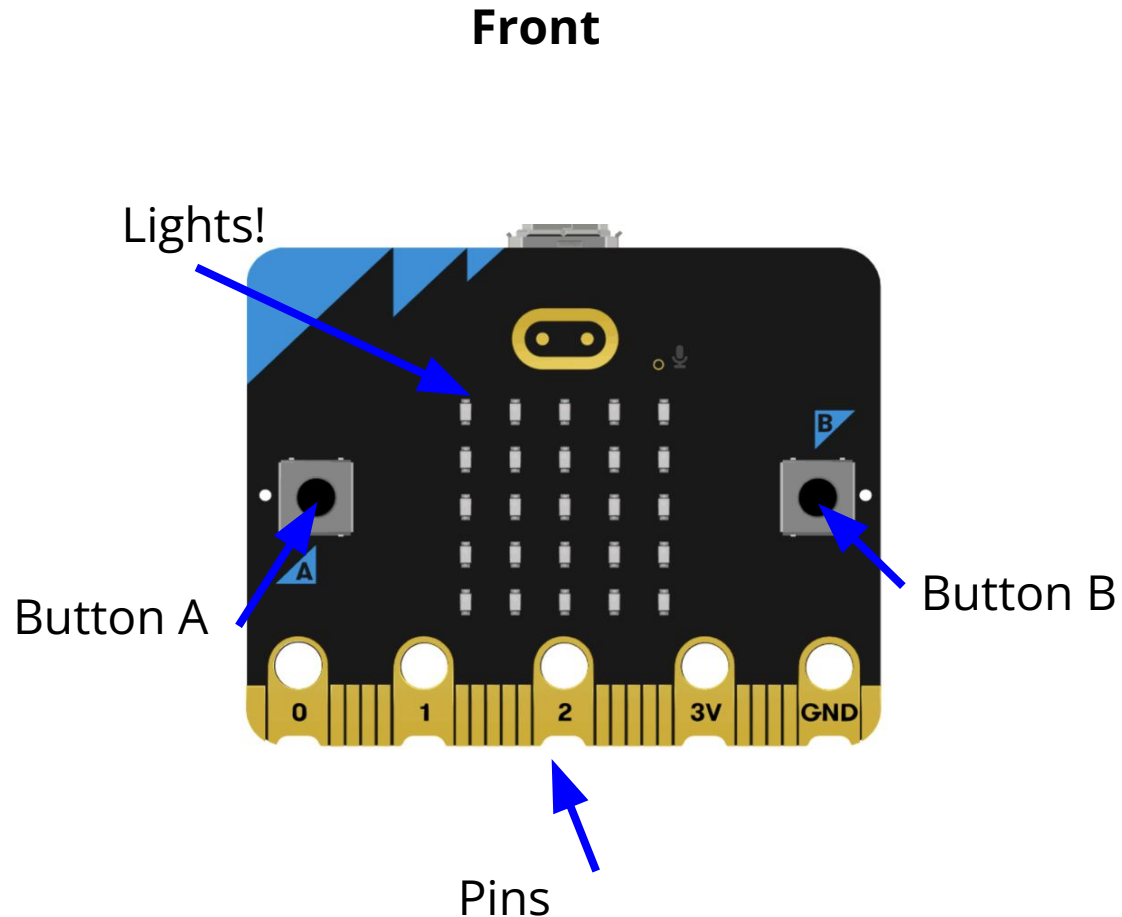


What is a Micro:Bit?

Buttons: We can press these and tell the Micro:Bit to do different things

Lights: We can turn each light on or off to make different images

Pins: These let us connect the Micro:Bit to other devices using wires



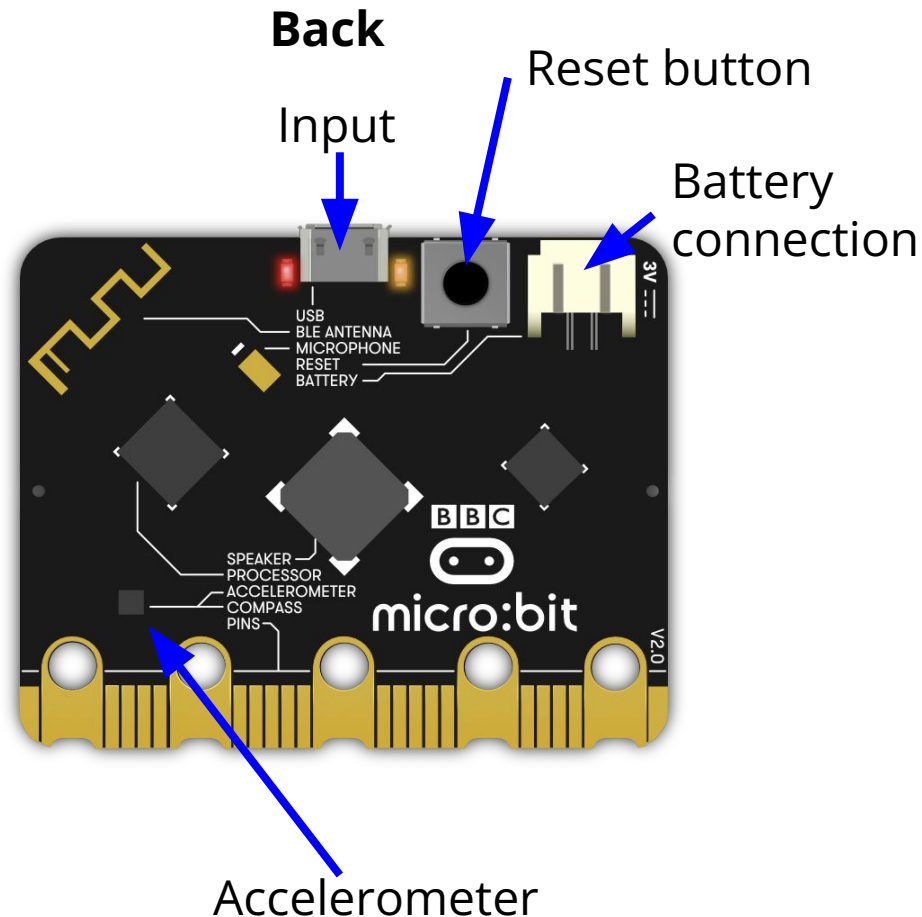
What is a Micro:Bit?

Input: Where we connect the cable from the computer to transfer our code/power to our Micro:Bit

Reset button: Let's you stop your code and starts it again

Battery connection: You can use your micro:bit even when it is not plugged into your computer! Ask you tutor for a battery pack if you need one.

Accelerometer: The Micro:bit can tell us when it is **accelerated** - so it knows when we shake it!



Using python.microbit.org

Today we will be using **python.microbit.org** to program our Micro:Bits.

Go to python.microbit.org

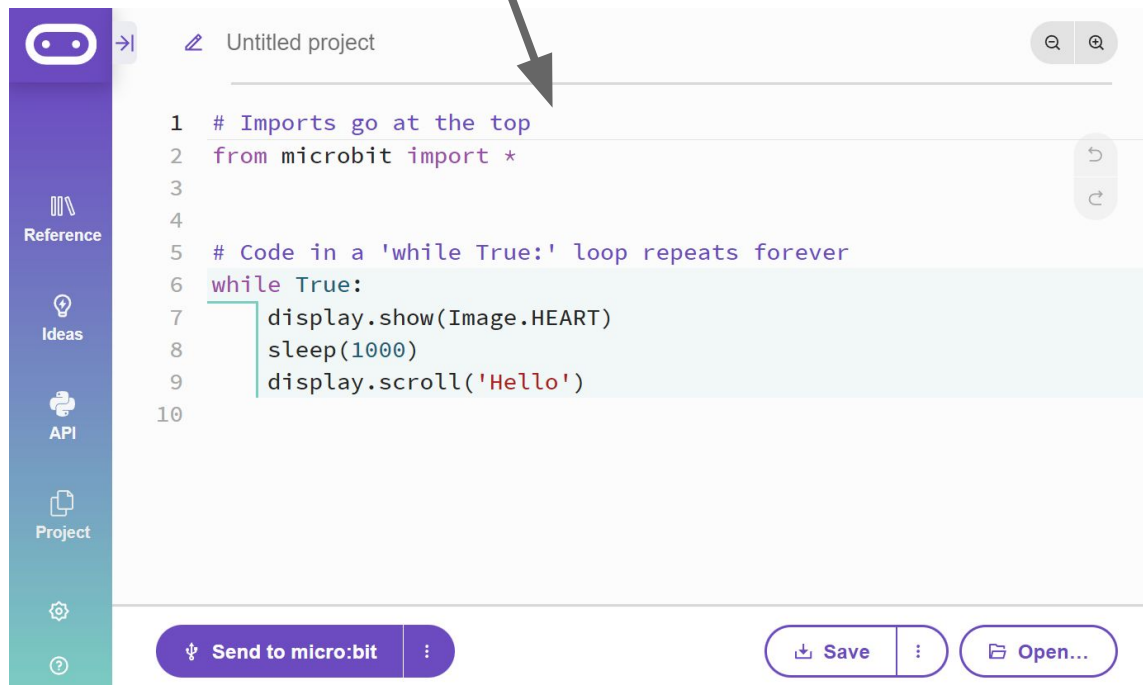


You should see this page pop up!

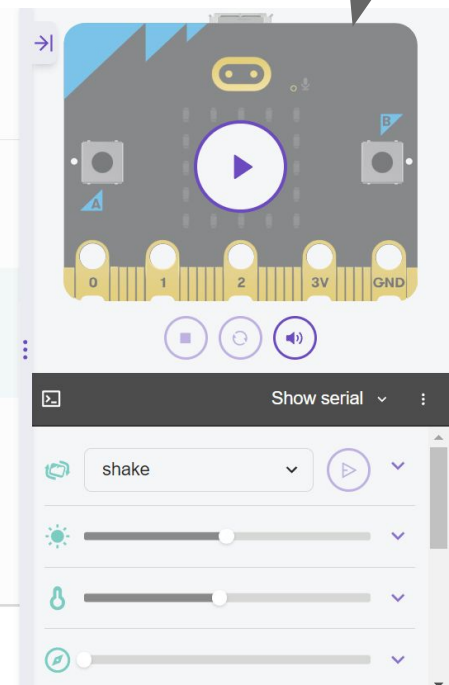


python.microbit.org

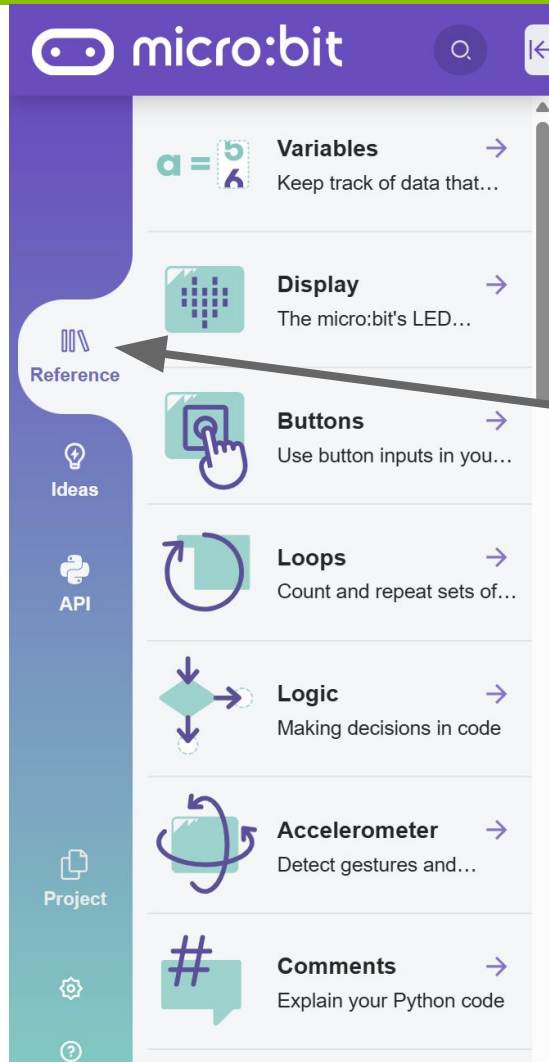
This is where we code



This is the simulator where we test our code



python.microbit.org



This is the reference. Click on the reference button to help you find the syntax for different instructions.

Click here



How do we write code for it?

Micro:Bits use **Python**, which is the programming language that we usually teach here at GPN!

Always make sure this line is at the top of your code!

```
from microbit import *
```

This lets us use lights, sounds, buttons and lots of other cool in our Python code for the Micro:Bit

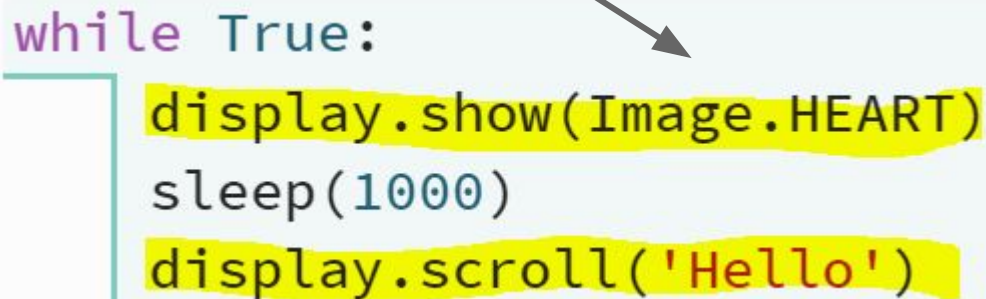


The Display

Your Micro:Bit has a 5 x 5 display grid of little red LEDs on the front!

You can do some cool stuff with the display like:

Show an image, like a heart!



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

The code is shown in a light blue box. The first line is 'while True:'. The next three lines are indented: 'display.show(Image.HEART)', 'sleep(1000)', and 'display.scroll('Hello')'. The first line is purple, and the others are blue. The text 'Image.HEART' and 'Hello' are highlighted in yellow. An arrow points from the text 'Show an image, like a heart!' to the 'Image.HEART' line. Another arrow points from the text 'Scroll a word across the display, like 'Hello'' to the 'display.scroll('Hello')' line.

Scroll a word across the display, like 'Hello'

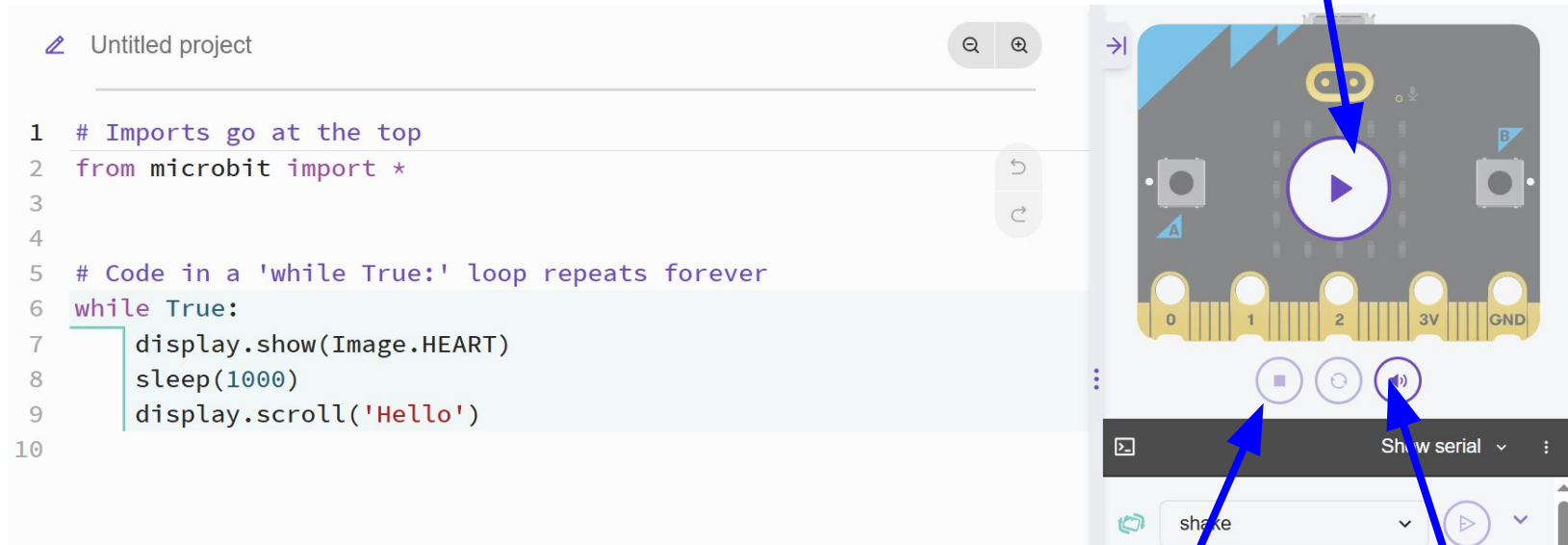
This code is in your **python.makecode.org** coding space - have a look

It's indented in a while loop - so it will repeat forever



Using the Simulator

- **Click the arrow on the Simulator to run the code**
- A heart is displayed for 1 second and then 'Hello'



We can run our code on the Simulator or the real micro:bit!

Stop, Restart, Simulator settings are underneath

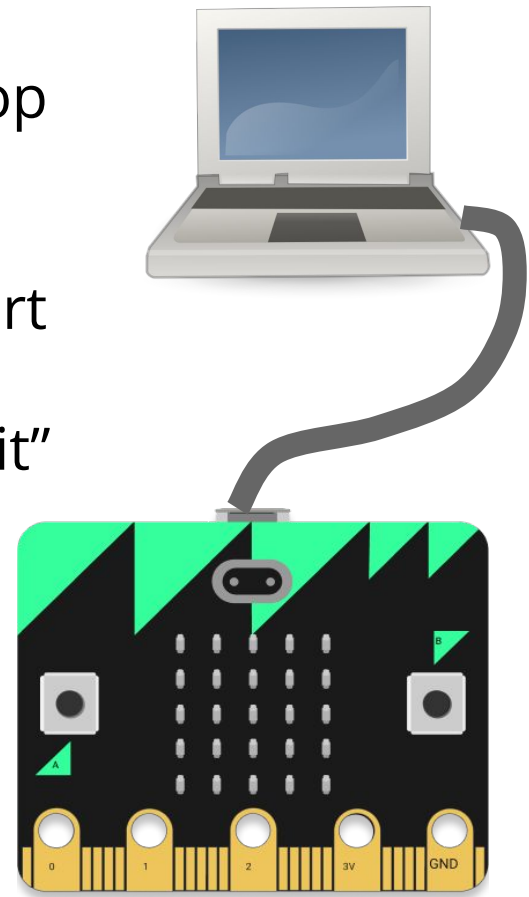
Stop

Restart



Connect the Micro:Bit

- Tutors will hand out the micro:bits & cables
- Connect the small end of the cable to the top of micro:bit
- Connect the other end to computer USB port
- New micro:bits will play a “Meet the Microbit” program for you to follow:
 - Push the buttons
 - Shake
 - Tilt to catch flashing LED
 - Clap a few times
- The tutors will help you

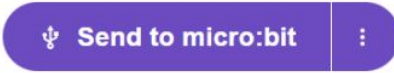


Run the code on the Micro:Bit (Chrome/Edge)

It's fun to mess around with the Micro:Bit on the simulator.
Now let's see your code on a Micro:Bit in real life!



Run your code on your Micro:Bit like this

1. Make sure your Micro:Bit is plugged into your computer
2. Click  bottom left
3. Follow the prompts
4. Choose your micro:bit and click CONNECT
5. **Wait for the red light** on the back of your micro:bit to stop flashing
6. Your code should be running on the micro:bit!

You should see a HEART displayed for 1 second and then HELLO


Want your code to start again? Press black **“reset”** button on the back



Run the code on the Micro:Bit (other browser)

This is for if you don't have the Chrome or Edge browser (eg Safari)

Run your code on your Micro:Bit like this

1. Make sure your Micro:Bit is plugged into your computer
2. Click  bottom left
3. Click Close when you get a popup
4. Name your project and click Confirm and Save
5. Follow the instructions on the popup (drag the file from your downloads folder to the MICROBIT device)
6. **Wait for the red light** on the back of your micro:bit to stop flashing
7. Your code should be running on the micro:bit!

You should see a HEART displayed for 1 second and then HELLO
Want your code to start again? Press black **“reset”** button on the back



Comments

- We use **comments** to write things in our code for humans!
- The computer ignores comments
- Comments start with a **#**

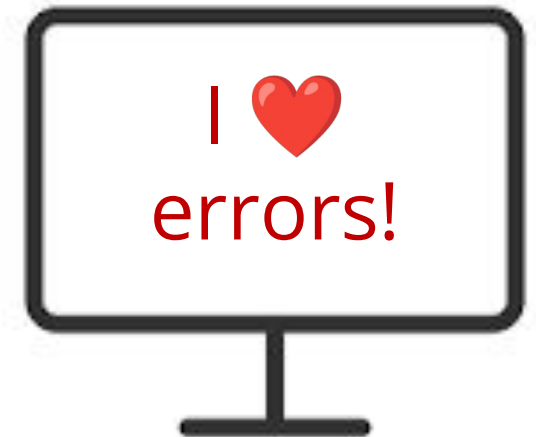
This code was written by Alex

- Programmers use comments to explain what their code does
- You can 'comment out' code to stop it from running


Have a look at the code in the coding space - can you see the purple comments lines starting with the #

Mistakes are Great! Errors on the Micro:bit!

- Programmers make A LOT of errors!
- Error messages give us hints on how to fix the problem
- Mistakes don't break computers!
- Lots of unexpected words on the micro:bit is an error message
- Run on the simulator to see it better



  line 19 NameError: name 'junge'



  line 20 IndentationError: unde

We can learn from our mistakes!



1. Where the error is

2. What went wrong

- In your code - red dot at the start of the line
- Put the cursor over than line of code to get a hint



Project Time!

Let's use our MicroBit!
Try Parts 0 & 1 of your Workbook!

The tutors will be around to help!

You've already done the first task!



Intro to Programming



What is programming?



Programming is not a bunch of crazy numbers!

It's giving computers a set of instructions!



A Special Language

A language to talk to dogs!



Programming is a language to talk to computers



People are smart! Computers are dumb!

Programming is creating a set of instructions, like a recipe.

Computers do **EXACTLY** what you say, every time.

Which is great if you give them a good recipe!

SALAD INSTRUCTIONS

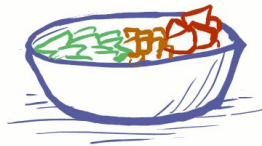
1) GET A LETTUCE HEAD, A CARROT, A TOMATO, A KNIFE, AND A BOWL



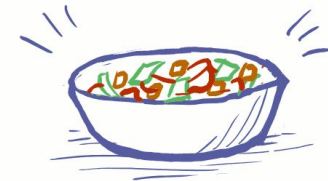
2) USE THE KNIFE TO CUT UP THE LETTUCE HEAD, CARROT, AND TOMATO



3) PUT THE LETTUCE, CARROT AND TOMATO IN THE BOWL



4) MIX THE CONTENTS OF THE BOWL



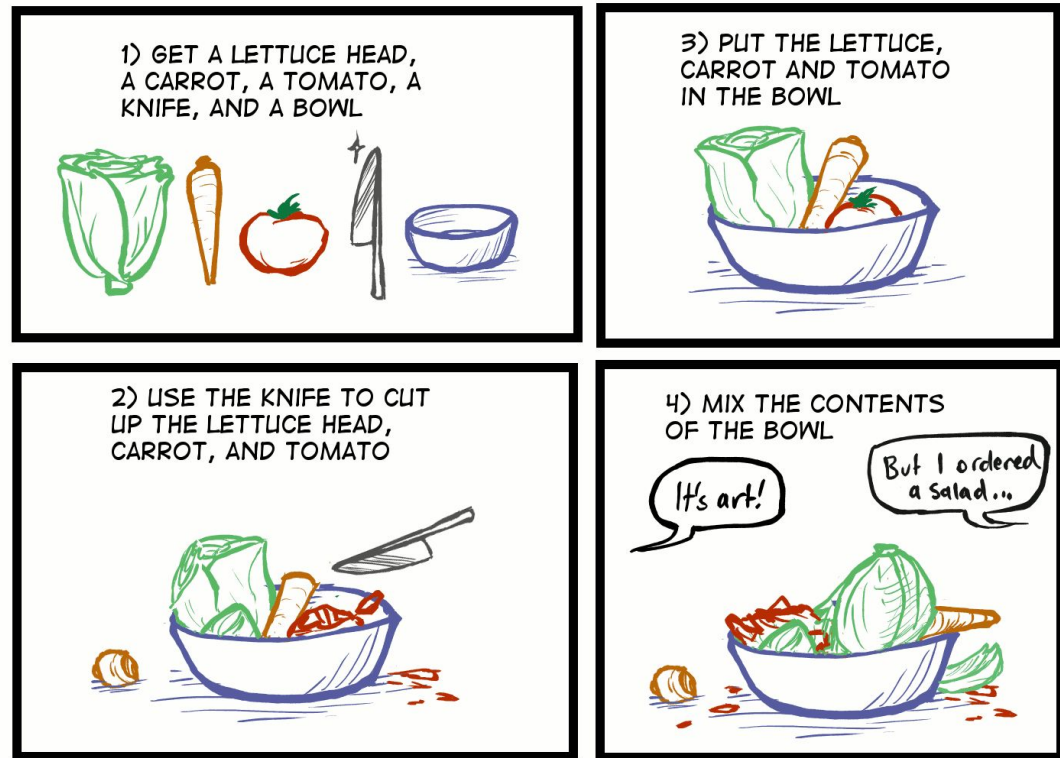
People are smart! Computers are dumb!

But if your recipe is wrong e.g. get it out of order....

A computer wouldn't know this recipe was wrong.

It would still try to make it anyway!

SALAD INSTRUCTIONS



People are smart! Computers are dumb!

Computers are bad at filling in the gaps!

A computer wouldn't know something was missing, it would just freak out!

SALAD INSTRUCTIONS



Everyone & everything has strengths!



How is the human brain different from a computer's brain?

Everyone & everything has strengths!



- Understand instructions very well despite spelling mistakes or typos
- Solve hard problems
- Invent computers and tell them what to do!
- Get smarter by learning



- Only does exactly what humans tell it
- Does it the same way every time
- Will work endlessly
- Really good at being repetitive
- REALLY fast
- Get smarter when humans tell it how



Variables



No Storing is Boring!

It's useful to be able to remember things for later!

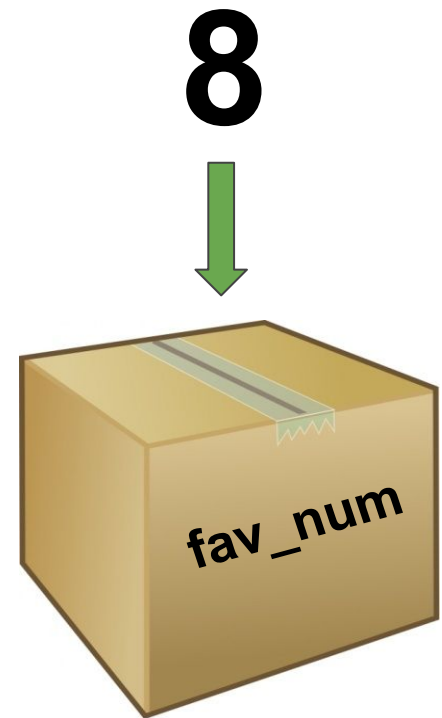
Computers remember things in "**variables**"

Variables are like putting things into a labelled cardboard box.

Let's make our favourite number 8!

In our code we make a variable and set it to a value like this:

fav_num = 8



Variables



Instead of writing the number 8, we can write `fav_num`. The computer will substitute the `fav_num`'s current value.



1. `fav_num - 6`

3. `fav_num + 21`

2. `fav_num * 2`

4. `fav_num / 2`



Variables



Instead of writing the number 8, we can write fav_num.



1. fav_num - 6
2

3. fav_num + 21

2. fav_num * 2

4. fav_num / 2



Variables



Instead of writing the number 8, we can write fav_num.



1. fav_num - 6
2

3. fav_num + 21

2. fav_num * 2
16

4. fav_num / 2



Variables



Instead of writing the number 8, we can write fav_num.



1. fav_num - 6
2

3. fav_num + 21
29

2. fav_num * 2
16

4. fav_num / 2



Variables



Instead of writing the number 8, we can write fav_num.



1. fav_num - 6
2

3. fav_num + 21
29

2. fav_num * 2
16

4. fav_num / 2
4



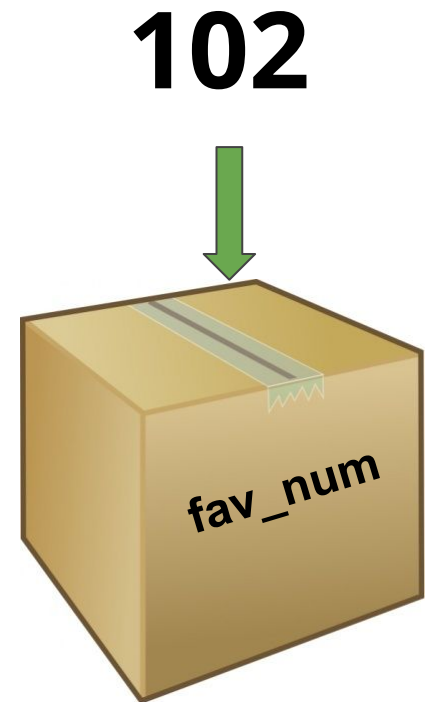
Variables

Variables are useful for storing things that change

Variables contain data that "vary" - hence the word "variable".

Let's change fav_num to **102**.

fav_num = 102



Variables

We're able to use our code for a new purpose, without rewriting everything:



1. fav_num - 6
96

3. fav_num + 21
123

2. fav_num * 2
204

4. fav_num / 2
51



Reusing variables



We can replace values in variables and print it with text:

```
animal = "dog"
display.scroll("My favourite animal is a " + animal)
animal = "cat"
display.scroll("My favourite animal is a " + animal)
animal = animal + "dog"
display.scroll("My favourite animal is a " + animal)
```

What will this output?



Reusing variables



We can replace values in variables:

```
animal = "dog"
display.scroll("My favourite animal is a " + animal)
animal = "cat"
display.scroll("My favourite animal is a " + animal)
animal = animal + "dog"
display.scroll("My favourite animal is a " + animal)
```

```
My favourite animal is a dog
My favourite animal is a cat
My favourite animal is a catdog
```



Variables



Your turn!

Can you guess what each
`display.scroll` will do?

```
>>> x = 3
>>> display.scroll(x)

>>> display.scroll(x + x)

>>> y = x
>>> display.scroll(y)

>>> y = y + 1
>>> display.scroll(y)
```



Variables



Your turn!

Can you guess what each
`display.scroll` will do?

```
>>> x = 3
>>> display.scroll(x)
3
>>> display.scroll(x + x)

>>> y = x
>>> display.scroll(y)

>>> y = y + 1
>>> display.scroll(y)
```



Variables



Your turn!

Can you guess what each
`display.scroll` will do?

```
>>> x = 3
>>> display.scroll(x)
3
>>> display.scroll(x + x)
6
>>> y = x
>>> display.scroll(y)

>>> y = y + 1
>>> display.scroll(y)
```



Variables



Your turn!

Can you guess what each
`display.scroll` will do?

```
>>> x = 3
>>> display.scroll(x)
3
>>> display.scroll(x + x)
6
>>> y = x
>>> display.scroll(y)
3
>>> y = y + 1
>>> display.scroll(y)
```



Variables



Your turn!

Can you guess what each
`display.scroll` will do?

```
>>> x = 3
>>> display.scroll(x)
3
>>> display.scroll(x +
x)
6
>>> y = x
>>> display.scroll(y)
3
>>> y = y + 1
>>> display.scroll(y)
4
```



Strings and Ints - Data Types!

Strings can have any characters in them, even just spaces!

Strings are surrounded by quotes (" or ')

```
"Hello, world!"      "bla bla bla"      "abcd1234"
```

```
":)"      "'I can use single quotes too!'"
```

Integers are whole numbers in python - no quotes (")

We can do maths with **integers** but not **strings**.

```
1      0      22      954865746
```

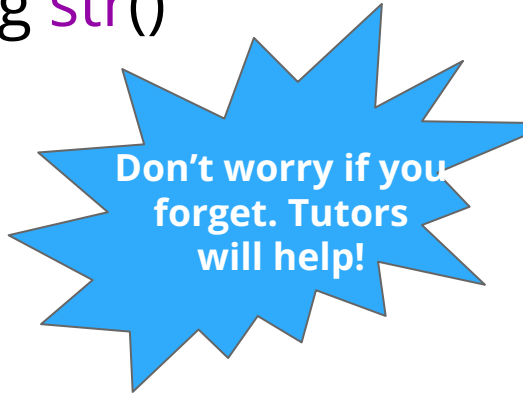


Strings and Ints!

Sometimes we need to turn a string into an integer and vice versa so we can use them as we want to in the code

- we can turn a **string** into an **integer** using **int()**
- we can turn an **integer** into a **string** using **str()**

You'll be doing this in your code!



Don't worry if you
forget. Tutors
will help!



Scroll... Scroll... Scroll... on the micro:bit

Words are too big to display within a 5x5 grid of lights.

Remember we can display words with **display.scroll()**.

```
display.scroll('Hello World')
```

Sometimes the text scrolls across too slowly - you can speed it up with **delay**.

```
display.scroll('Hello World', delay=100)
```

A smaller delay (eg 100 results in faster scrolling).

The default speed is 150!

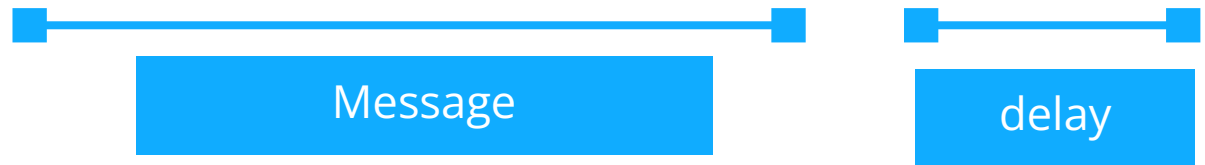


Multiple Instructions

What happens if we want to change the speed **AND** join variables with strings?

This is how you would do it! :)

```
win_count = 3  
display.scroll('Wins: ' + str(win_count), delay=75)
```



See that we need to use **str()** to convert the number win_count to a string before we can join it (+) with the the other string!



Sleep... zzz! ... on the micro:bit

Computers are really fast, sometimes our program moves too quickly to enjoy it!

For example:

```
display.show(Image.HAPPY)
sleep(1000)
display.show(Image.SAD)
sleep(1000)
display.show(Image.CONFUSED)
sleep(1000)
```

Without a sleep, the computer will run through the code so quickly, and we will only see a CONFUSED face.

We can slow it down by using **sleep()**

Sleep is done in milliseconds (so the number of seconds x 1000)



Project Time!

Let's use our MicroBit!
Try Part 2 of your Workbook!

The tutors will be around to help!



While Loops



Loops



We know how to do things on repeat!

Sometimes we want to do some code on repeat!

Introducing ... while loops!

What do you think this does?

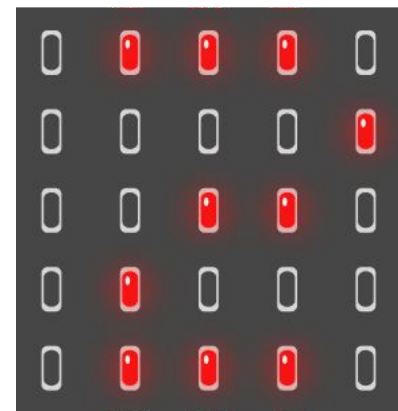
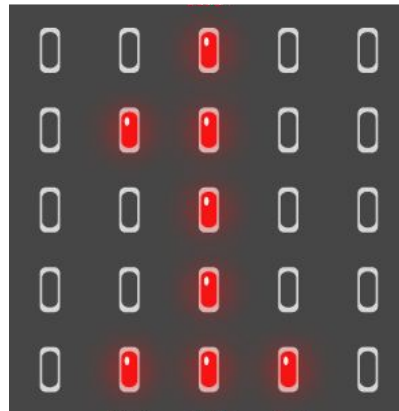
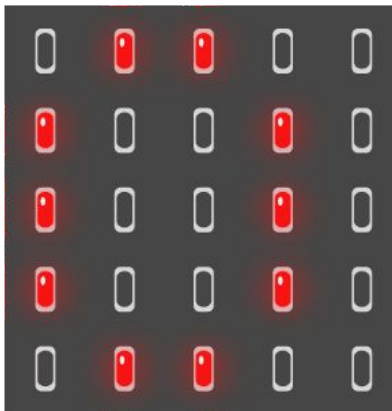
```
i = 0
while i < 3:
    display.scroll(i)
    i = i + 1
```



Introducing ... while loops!

What do you think this does?

```
i = 0
while i < 3:
    print(i)
    i = i + 1
```



Introducing ... while loops!

Stepping through a while loop...



Introducing ... while loops!

One step at a time!

```
◆ i = 0  
  while i < 3:  
    display.scroll(i)  
    i = i + 1
```

MY VARIABLES

i = 0

Set the
variable



Introducing ... while loops!

One step at a time!

0 is less
than 3!

```
i = 0
while i < 3:
    display.scroll(i)
    i = i + 1
```

MY VARIABLES

i = 0



Introducing ... while loops!

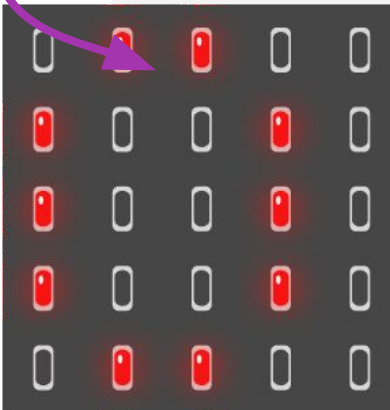
One step at a time!

Print !

```
i = 0
while i < 3:
    display.scroll(i)
    i = i + 1
```

MY VARIABLES

i = 0



Introducing ... while loops!

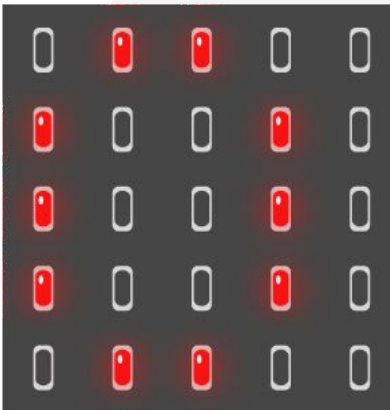
One step at a time!

```
i = 0
while i < 3:
    display.scroll(i)
    i = i + 1
```

MY VARIABLES

~~i = 0~~
i = 1

UPDATE
TIME!



Introducing ... while loops!

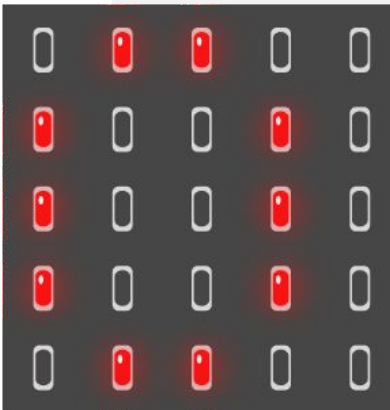
One step at a time!

Take it
from the
top!

```
i = 0
while i < 3:
    display.scroll(i)
    i = i + 1
```

MY VARIABLES

```
i = 0
i = 1
```



Introducing ... while loops!

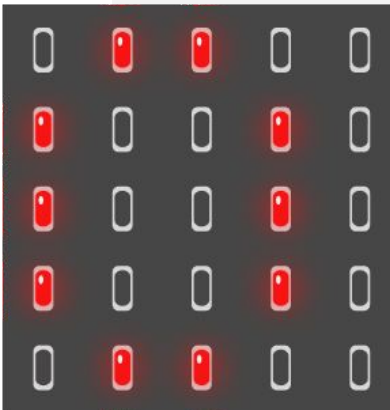
One step at a time!

i is less
than 3!

```
i = 0
while i < 3:
    display.scroll(i)
    i = i + 1
```

MY VARIABLES

```
i = 0
i = 1
```



Introducing ... while loops!

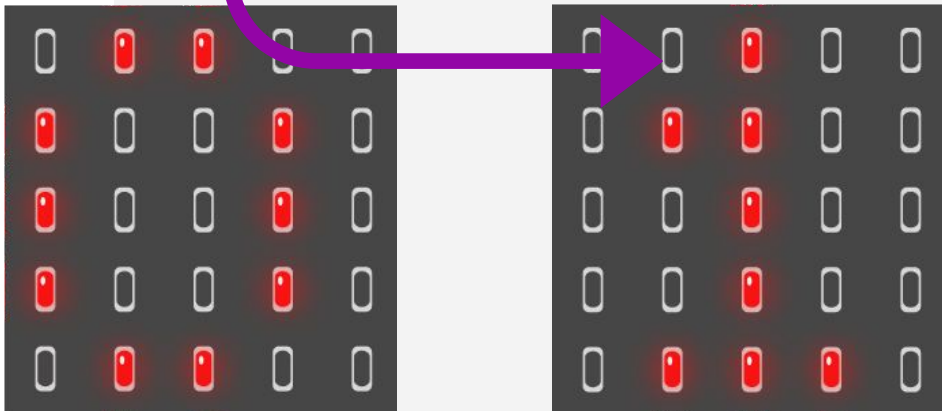
One step at a time!

Print !

```
i = 0
while i < 3:
    display.scroll(i)
    i = i + 1
```

MY VARIABLES

```
i = 0
i = 1
```



Introducing ... while loops!

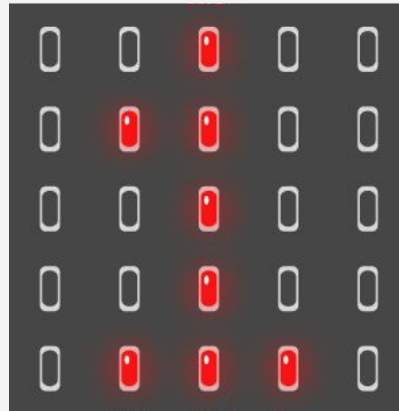
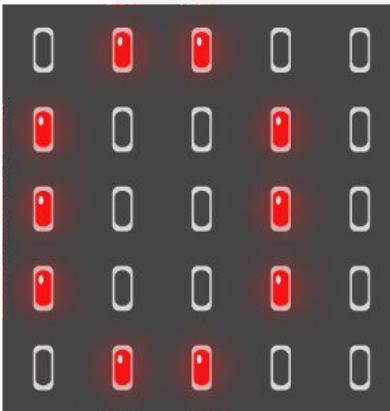
One step at a time!

```
i = 0
while i < 3:
    display.scroll(i)
    i = i + 1
```

MY VARIABLES

~~i = 0~~
~~i = 1~~
i = 2

UPDATE
TIME!



Introducing ... while loops!

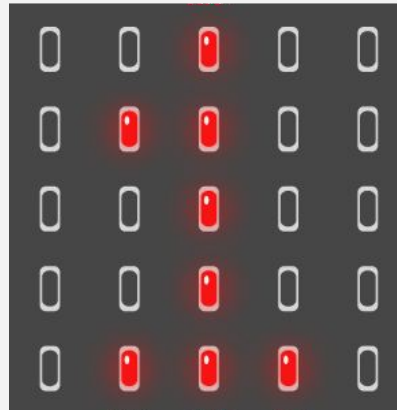
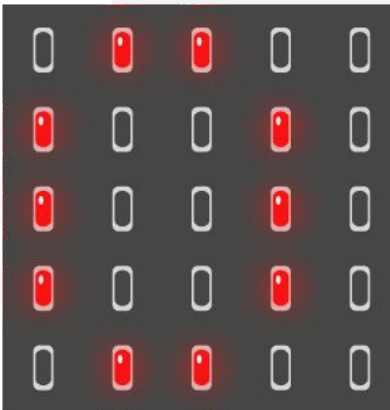
One step at a time!

Take it
from the
top!

```
i = 0
while i < 3:
    display.scroll(i)
    i = i + 1
```

MY VARIABLES

```
i = 0
i = 1
i = 2
```



Introducing ... while loops!

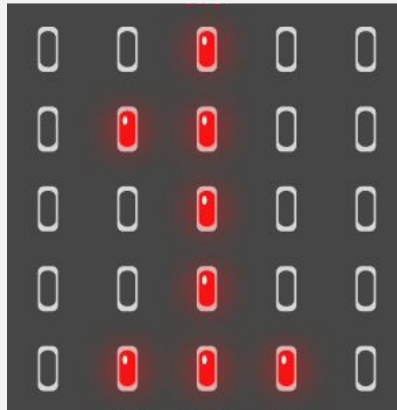
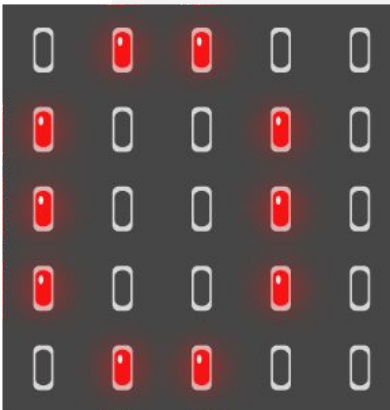
One step at a time!

2 is less
than 3!

```
◆ i = 0
  while i < 3:
    display.scroll(i)
    i = i + 1
```

MY VARIABLES

~~i = 0~~
~~i = 1~~
i = 2



Introducing ... while loops!

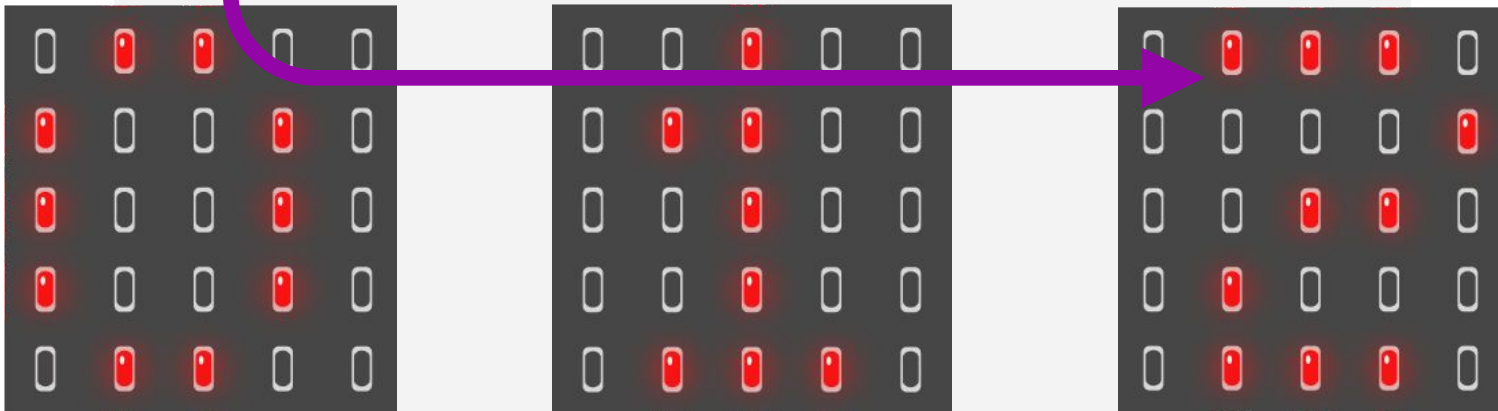
One step at a time!

Print !

```
i = 0
while i < 3:
    display.scroll(i)
    i = i + 1
```

MY VARIABLES

```
i = 0
i = 1
i = 2
```



Introducing ... while loops!

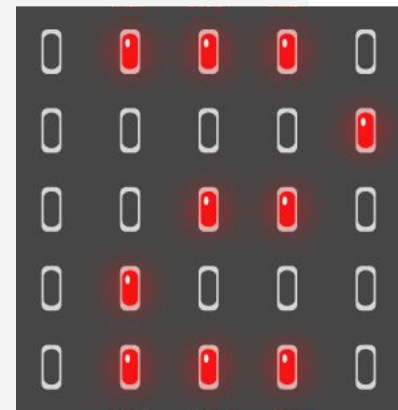
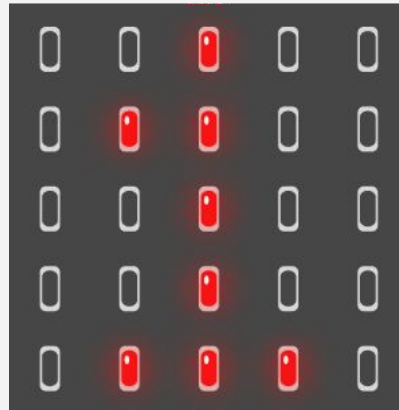
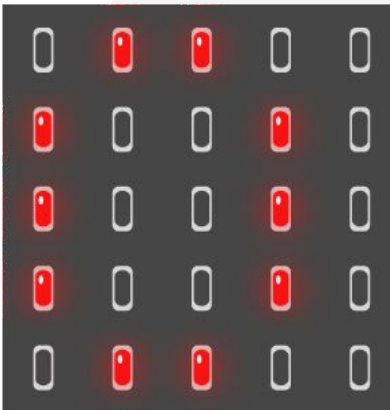
One step at a time!

```
i = 0
while i < 3:
    display.scroll(i)
    i = i + 1
```

MY VARIABLES

~~i = 0~~
~~i = 1~~
~~i = 2~~
i = 3

UPDATE
TIME!



Introducing ... while loops!

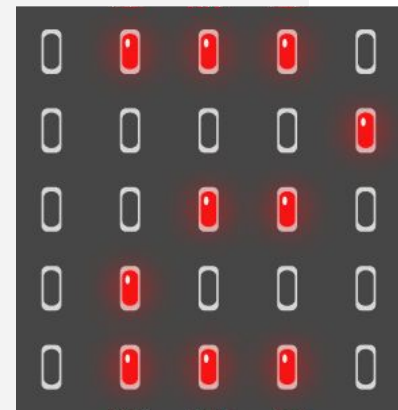
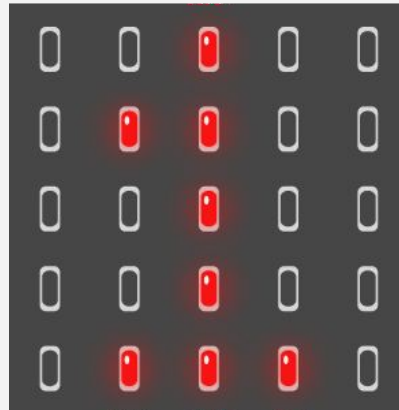
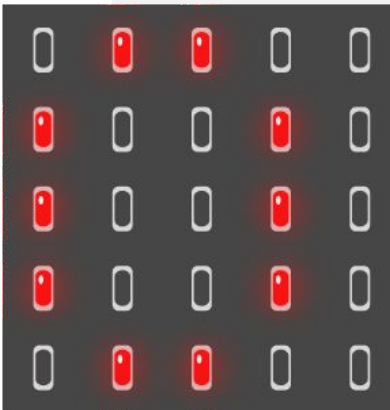
One step at a time!

Take it
from the
top!

```
i = 0
while i < 3:
    display.scroll(i)
    i = i + 1
```

MY VARIABLES

```
i = 0
i = 1
i = 2
i = 3
```



Introducing ... while loops!

One step at a time!

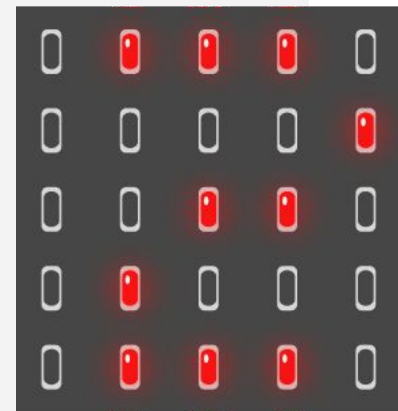
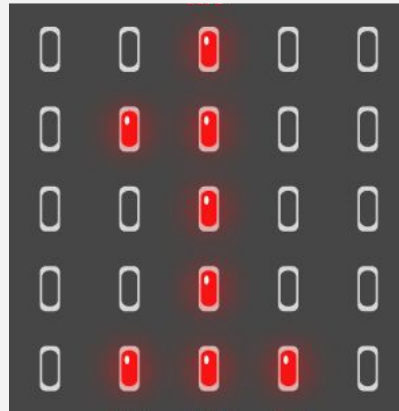
```
i = 0
while i < 3:
    display.scroll(i)
    i = i + 1
```

MY VARIABLES

~~i = 0~~
~~i = 1~~
~~i = 2~~
i = 3

3 IS NOT
less than
3!

We are
done
with this
loop!



Introducing ... while loops!

Initialise the loop variable

Loop condition

```
i = 0  
while i < 3:  
    display.scroll(i)  
    i = i + 1
```

Code to repeat

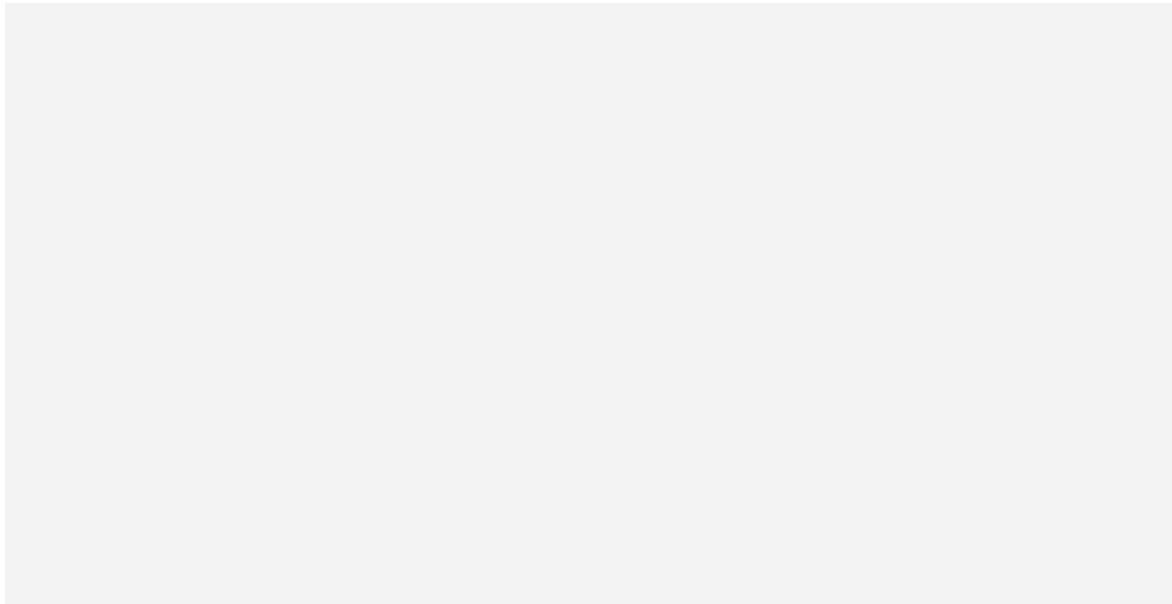
Update the loop variable



What happens when.....

What happens if we forget to update the loop variable?

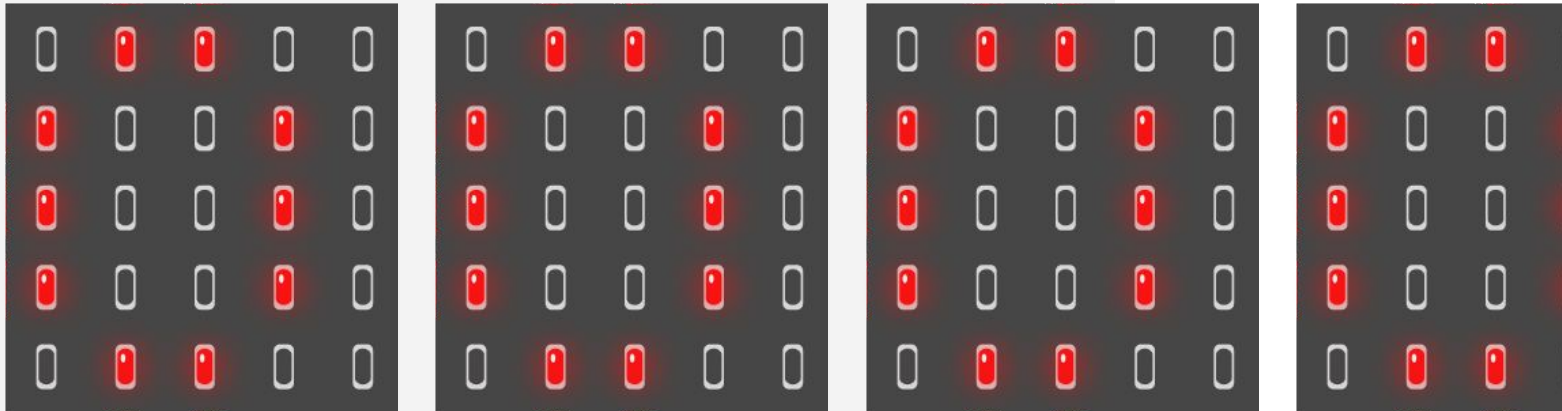
```
i = 0
while i < 3:
    display.scroll(i)
```



What happens when.....

What happens if we forget to update the loop variable?

```
i = 0
while i < 3:
    display.scroll(i)
```



Infinite loop!

Sometimes we want our loop to go forever!

So we set a condition that is always True!

We can even just write True!

```
while True:  
    display.scroll("Are we there yet?")
```



Infinite loop!

Sometimes we want our loop to go forever!

So we set a condition that is always True!

We can even just write True!

```
while True:  
    display.scroll("Are we there yet?")
```

Are we there



Give me a break!

But what if I wanna get out of a loop early?
That's when we use the **break** keyword!

```
from microbit import *  
while True:  
    display.show(Image.HAPPY)  
  
    if button_a.is_pressed():  
        break  
  
display.clear()
```



Micro:Bit Inputs



Conditions!

Conditions let us make a decision.

First we test if the condition is met!

Then maybe we'll do the thing



If it's raining take an umbrella

Yep it's raining

..... take an umbrella

Booleans (True and False)

Computers store whether a condition is met in the form of

True and **False**

To figure out if something is **True** or **False** we do a comparison

`5 < 10`

`3 + 2 == 5`

`5 != 5`

`"Dog" == "dog"`

`"D" in "Dog"`

`"Q" not in "Cat"`



Booleans (True and False)

computers store whether a condition is met in the form of

True and **False**

To figure out if something is **True** or **False** we do a comparison

<code>5 < 10</code>	True	<code>"Dog" == "dog"</code>
<code>3 + 2 == 5</code>		<code>"D" in "Dog"</code>
<code>5 != 5</code>		<code>"Q" not in "Cat"</code>



Booleans (True and False)

computers store whether a condition is met in the form of

True and **False**

To figure out if something is **True** or **False** we do a comparison

`5 < 10` **True**

`3 + 2 == 5` **True**

`5 != 5`

`"Dog" == "dog"`

`"D" in "Dog"`

`"Q" not in "Cat"`



Booleans (True and False)

computers store whether a condition is met in the form of

True and **False**

To figure out if something is **True** or **False** we do a comparison

<code>5 < 10</code>	<code>True</code>	<code>"Dog" == "dog"</code>
<code>3 + 2 == 5</code>	<code>True</code>	<code>"D" in "Dog"</code>
<code>5 != 5</code>	<code>False</code>	<code>"Q" not in "Cat"</code>



Booleans (True and False)

computers store whether a condition is met in the form of

True and **False**

To figure out if something is **True** or **False** we do a comparison

<code>5 < 10</code>	<code>True</code>	<code>"Dog" == "dog"</code>	<code>False</code>
<code>3 + 2 == 5</code>	<code>True</code>	<code>"D" in "Dog"</code>	
<code>5 != 5</code>	<code>False</code>	<code>"Q" not in "Cat"</code>	



Booleans (True and False)

computers store whether a condition is met in the form of

True and **False**

To figure out if something is **True** or **False** we do a comparison

<code>5 < 10</code>	<code>True</code>	<code>"Dog" == "dog"</code>	<code>False</code>
<code>3 + 2 == 5</code>	<code>True</code>	<code>"D" in "Dog"</code>	<code>True</code>
<code>5 != 5</code>	<code>False</code>	<code>"Q" not in "Cat"</code>	



Booleans (True and False)

computers store whether a condition is met in the form of

True and **False**

To figure out if something is **True** or **False** we do a comparison

<code>5 < 10</code>	<code>True</code>	<code>"Dog" == "dog"</code>	<code>False</code>
<code>3 + 2 == 5</code>	<code>True</code>	<code>"D" in "Dog"</code>	<code>True</code>
<code>5 != 5</code>	<code>False</code>	<code>"Q" not in "Cat"</code>	<code>True</code>



Conditions

So to know whether to do something, they find out if it's **True**!

```
fave_num = 5
if fave_num < 10:
    display.scroll("that's a small number")
```



Conditions

So to know whether to do something, they find out if it's **True**!

```
fave_num = 5
if fave_num < 10:
    display.scroll("that's a small number")
```

That's the
condition!

Conditions

So to know whether to do something, they find out if it's **True**!

```
fave_num = 5
if fave_num < 10:
    display.scroll("that's a small number")
```

That's the
condition!

Is it **True** that fave_num is less than 10?

- Well, fave_num is 5
- And it's **True** that 5 is less than 10
- So it is **True**!



Conditions

So to know whether to do something, they find out if it's **True**!

```
fave_num = 5
if True
    display.scroll("that's a small number")
```

Put in the
answer to
the question

Is it **True** that fave_num is less than 10?

- Well, fave_num is 5
- And it's **True** that 5 is less than 10
- So it is **True**!



Conditions

So to know whether to do something, they find out if it's **True**!

```
fave_num = 5
if True
    display.scroll("that's a small number")
```

What do you think happens?

```
>>>
```



Conditions

So to know whether to do something, they find out if it's **True**!

```
fave_num = 5
if True
    display.scroll("that's a small number")
```

What do you think happens?

```
>>> that's a small number
```



Buttons!

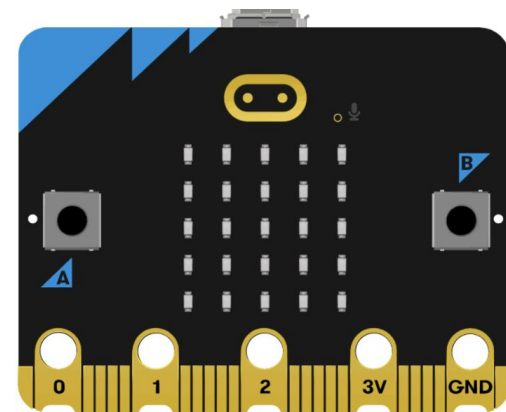
Your Micro:Bit has 2 buttons: Button A and Button B

You can use this code to check if a button is pressed:

```
if button_a.was_pressed():
```

```
    If button_b.was_pressed():
```

The statement will be **TRUE** if the button is being pressed at that time and it will be **FALSE** if it is *not* being pressed



Buttons!

What do you think this code does?

```
if button_a.was_pressed():  
    display.show(Image.HAPPY)  
  
if button_b.was_pressed():  
    display.show(Image.SAD)
```

If **button a** is pressed when the Micro:Bit gets to this line of code then what happens?

If **button b** is pressed when the Micro:Bit gets to this line of code then what happens

What do you think happens if *both* button a AND button b are being pressed?



Buttons!

What do you think this code does?

```
if button_a.was_pressed():  
    display.show(Image.HAPPY)  
  
if button_b.was_pressed():  
    display.show(Image.SAD)
```

If **button a** is pressed when the Micro:Bit gets to this line of code then what happens?

The Micro:Bit shows a Happy face

If **button b** is pressed when the Micro:Bit gets to this line of code then what happens

What do you think happens if *both* button a AND button b are being pressed?



Buttons!

What do you think this code does?

```
if button_a.was_pressed():  
    display.show(Image.HAPPY)  
  
if button_b.was_pressed():  
    display.show(Image.SAD)
```

If **button a** is pressed when the Micro:Bit gets to this line of code then what happens?

The Micro:Bit shows a Happy face

If **button b** is pressed when the Micro:Bit gets to this line of code then what happens

The Micro:Bit shows a Sad face

What do you think happens if *both* button a AND button b are being pressed?



Accelerometer!

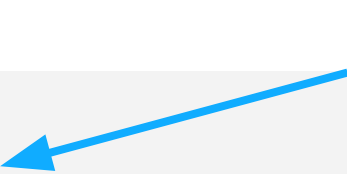
Your micro:bit has a motion sensor (accelerometer).

This sensor has the ability to detect when you shake it or tilt it left to right, backwards and forwards and up and down.

We can use a **while loop** in your code like this to continually check if the micro:bit has been shaken:

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

Information
from the sensor



Accelerometer!

What do you think this code does?

```
while True:  
    if accelerometer.was_gesture('shake'):  
        display.scroll('I'm getting dizzy')
```



Accelerometer!

What do you think this code does?

```
while True:
    if accelerometer.was_gesture('shake'):
        display.scroll('I'm getting dizzy')
```

It will display 'I'm getting dizzy' every time the micro:bit is shaken




Indentation

Whenever we have an if statement or while loop, there is something we have to do to make sure it only runs what we want it to run inside the if statement.

... that is called indentation

```
while True:
    if num>10:
        display.scroll('a big number')
```



These gaps are
indentation!

Indentation

Whenever we have an if statement or while loop, there is something we have to do to make sure it only runs what we want it to run inside the if statement.

... that is called indentation

```
while True:
    if num>10:
        display.scroll('a big number')
```

We use the indentation to tell the code that a piece of code is "inside" another, for loops this means any code that has at least one extra gap after the loop, will be run.



But how do we indent?

There are a couple of ways to make sure a line of code is indented.

One is pressing the **TAB** button on your keyboard before a line of code.

Another is selecting the lines you want to indent and pressing the **TAB** button to indent them all at once.

And the last main one is to select all the lines you want to indent and press the **CTRL** and the **]** button at the same time.

Remember you need to indent for your code to work right!

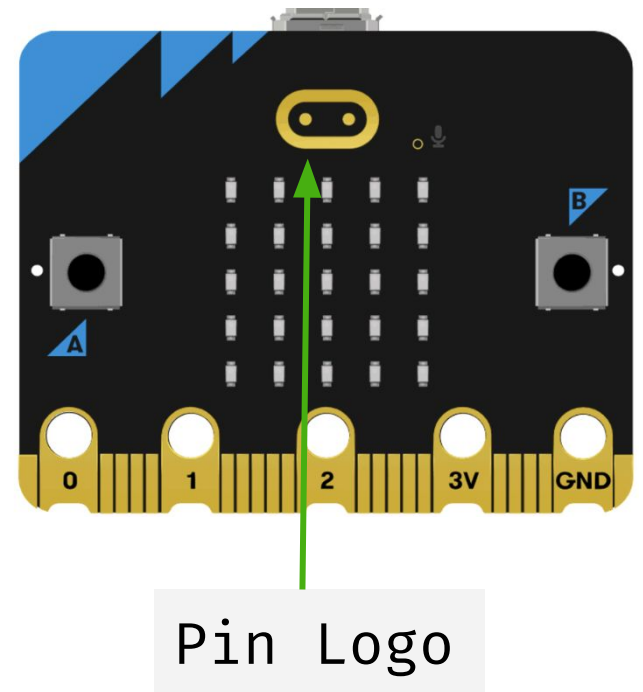


Pin Logo!

Your Micro:Bit has touch sensitive pin logo at the top of the Micro:bit.

You can use this code to check if the pin logo is being touched.

```
if pin_logo.is_touched():
```



Pin Logo!

What do you think this code does?

```
while True:
    if pin_logo.is_touched():
        display.show(image.DUCK)
    else:
        display.clear()
```



Pin Logo!

What do you think this code does?

```
while True:
    if pin_logo.is_touched():
        display.show(image.DUCK)
    else:
        display.clear()
```

While the pin is being touched, the duck is displayed on the microbit screen.

While the pin is not being touched, the screen is blank.



Running Time

Sometimes you want to time things. Like, for example, if you wanted to put a time limit on a game and see how many points you can get in 30 seconds!

To figure out how long the Micro:Bit program has been running (in milliseconds) you can use this command:

```
time = running_time()
```

What would `running_time()` be after 4 seconds?

What about after **10 and a half** seconds?

Running Time

Sometimes you want to time things. Like, for example, if you wanted to put a time limit on a game and see how many points you can get in 30 seconds!

To figure out how long the Micro:Bit program has been running (in milliseconds) you can use this command:

```
time = running_time()
```

What would `running_time()` be after 4 seconds?

4000

What about after **10 and a half** seconds?



Running Time

Sometimes you want to time things. Like, for example, if you wanted to put a time limit on a game and see how many points you can get in 30 seconds!

To figure out how long the Micro:Bit program has been running (in milliseconds) you can use this command:

```
time = running_time()
```

What would `running_time()` be after 4 seconds?

4000

What about after **10 and a half** seconds?

10,500



Project Time!

Does that press your buttons?

Try to do Parts 4-9!

The tutors will be around to help!



Tell us what you think!

Click on the
End of Day Form
and fill it in now!