

# Welcome to the labs!



Tamagotchi! - Micro:bits

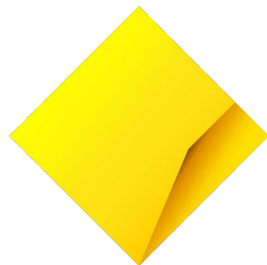


# Thank you to our Sponsors!

Platinum Sponsor:



Gold Sponsor:



**Commonwealth  
Bank**



# Who are the tutors?

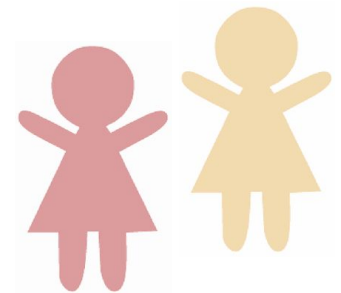
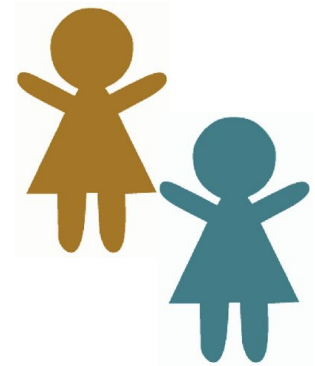


Who are you?



# 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](https://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!



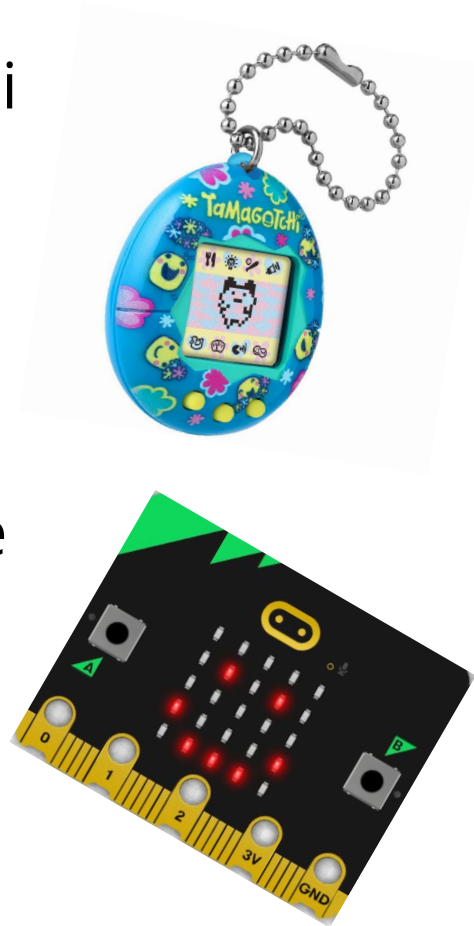
Today's project!

**Tamagotchi - Micro:Bit**



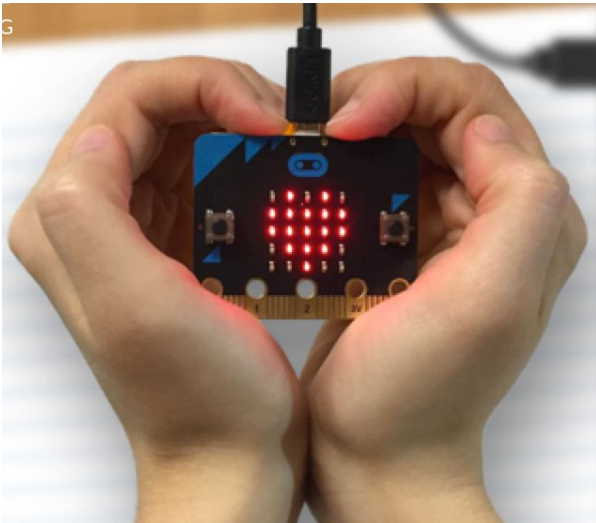
# Tamagotchi

- 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

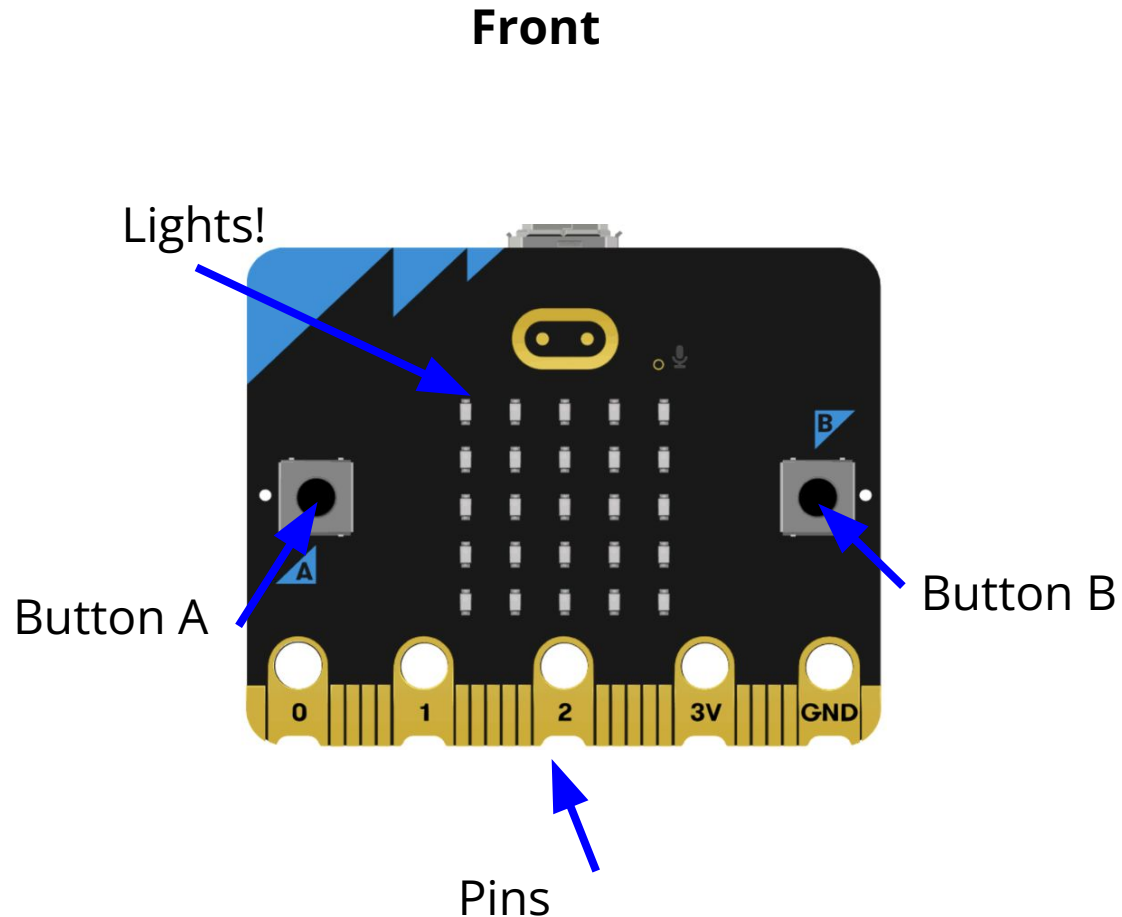


# What is a Micro:Bit?

**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



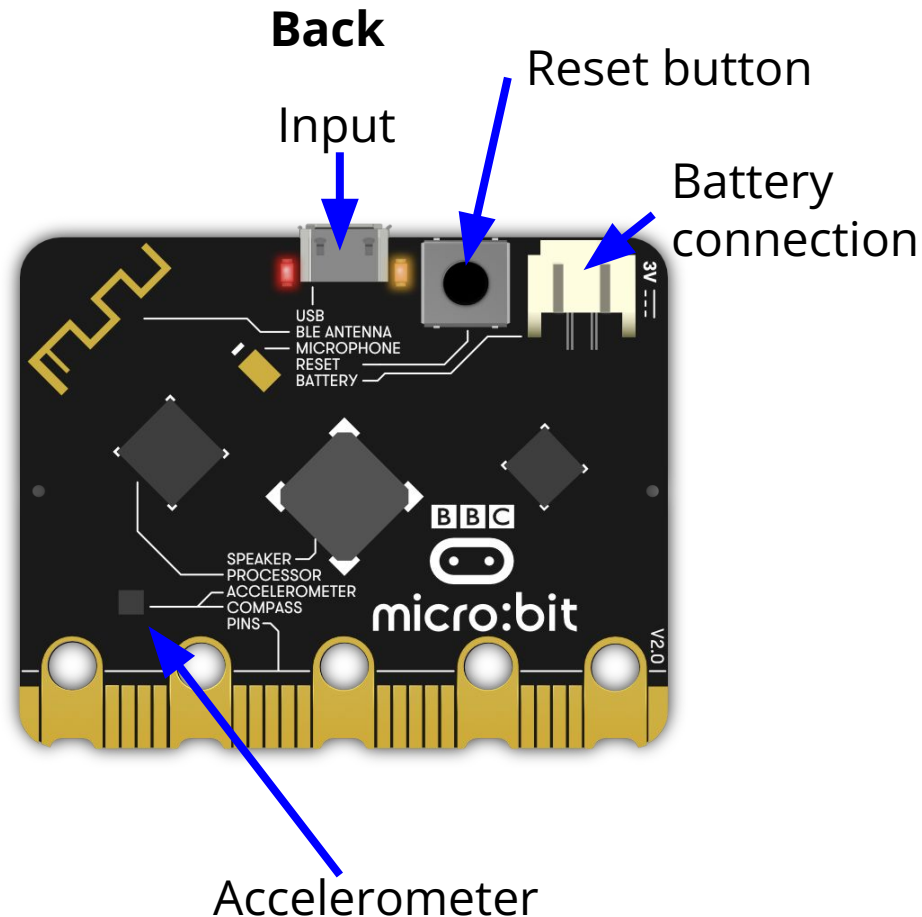
# What is a Micro:Bit?

**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!

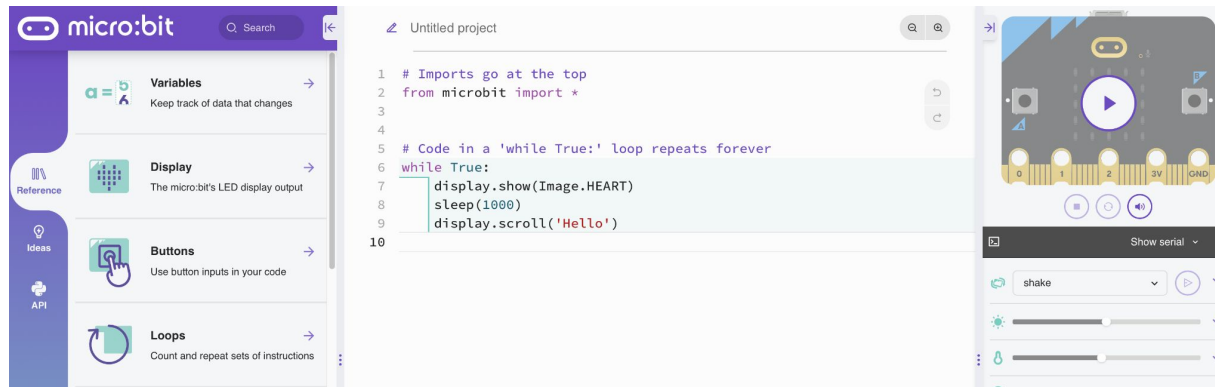




# Using python.microbit.org

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

***Go to python.microbit.org***

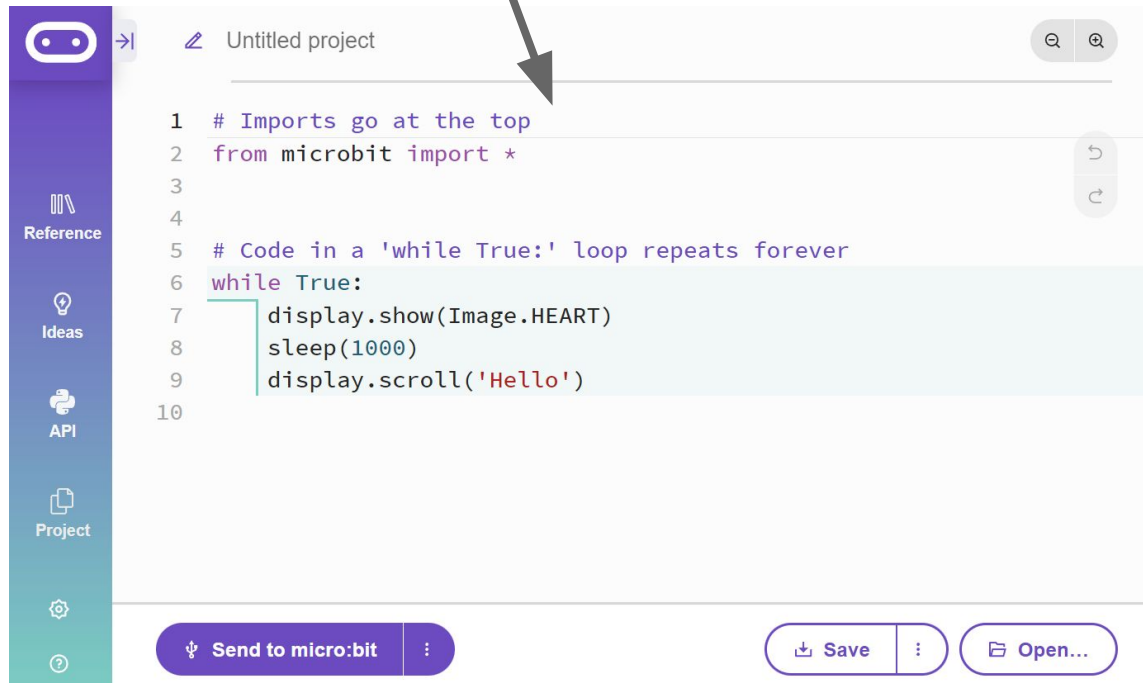


*You should see this page pop up!*

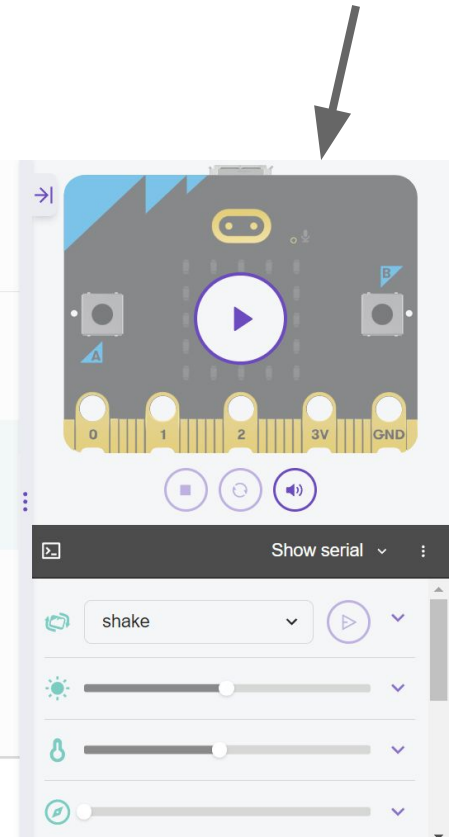


# python.microbit.org

This is where we code



This is the simulator where we test our code



# 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

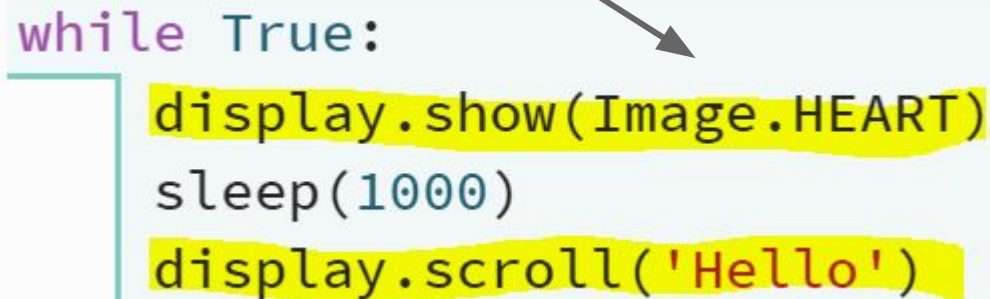


# The Display

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 two lines of the loop are highlighted in yellow. An arrow points from the text 'Show an image, like a heart!' to the 'display.show(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'

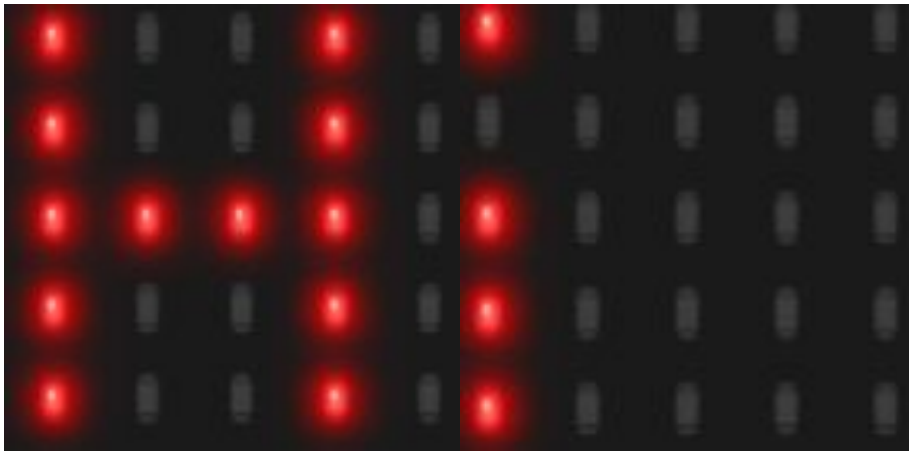
This code is in your **python.makecode.org** coding space - have a look

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

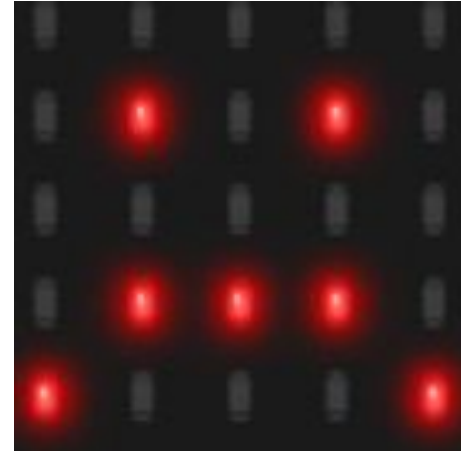
# To show or to scroll....

Remember our microbit has a 5x5 grid of lights.

```
display.scroll('Hi!')
```



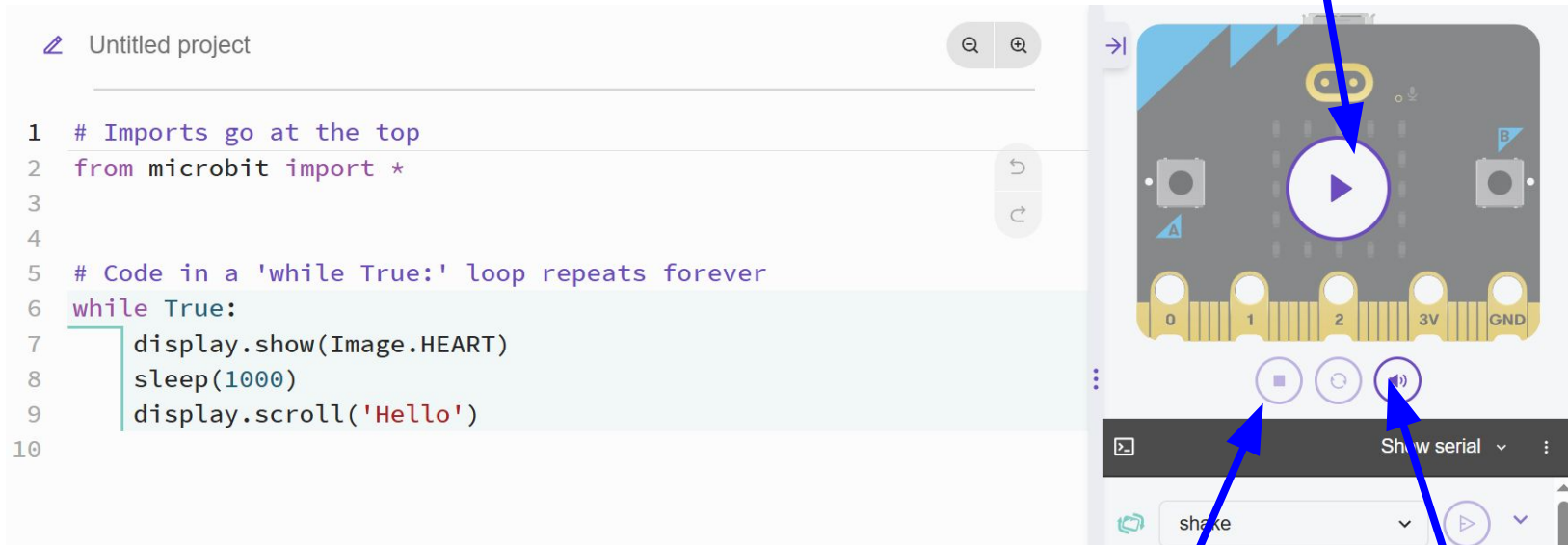
```
display.show(Image.SAD)
```



So if you want to display long words or messages, use scroll. If you want to display something that fits in a 5x5 grid, use show!

# 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

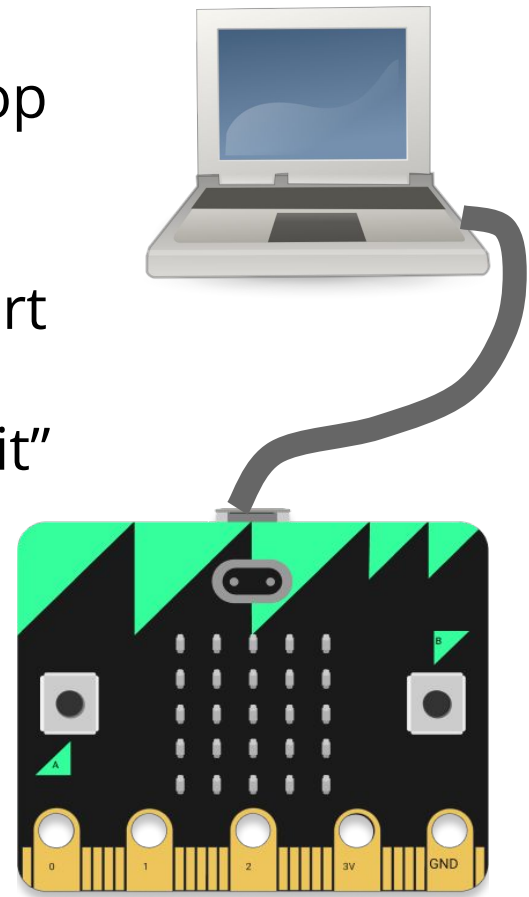
**Stop**

**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

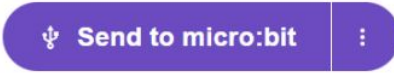


# Run the code on the Micro:Bit (Chrome/Edge)

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



# 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!

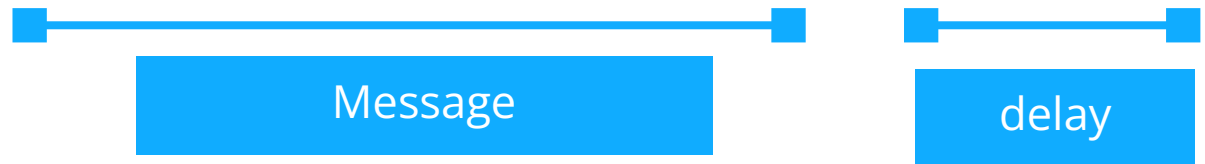


# Multiple Instructions

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)



# Comments

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

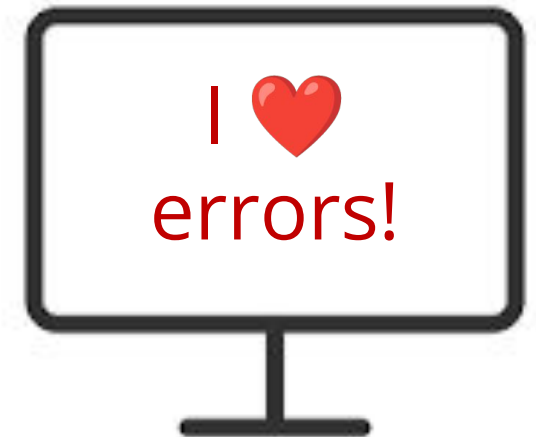
**# This code was written by Alex**

- 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!





# Variables



# No Storing is Boring!

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

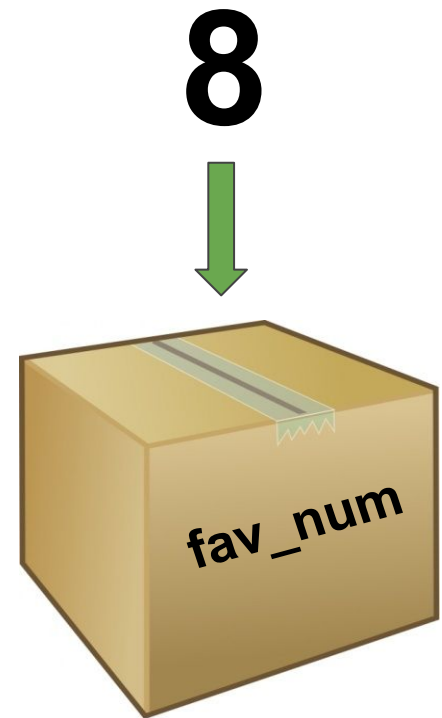
Computers remember things in "**variables**"

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**



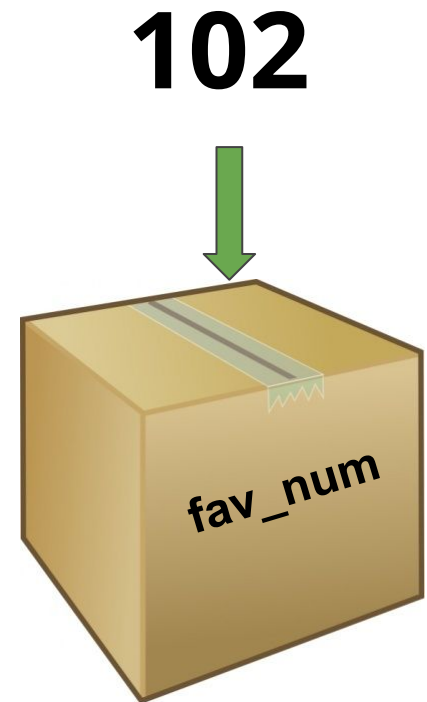
# Variables

## 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`**



# Reusing variables



We can replace values in variables and show it:

```
animal = "dog"  
display.scroll(animal)  
animal = "cat"  
display.scroll(animal)  
animal = animal + "dog"  
display.scroll(animal)
```

What will this scroll?

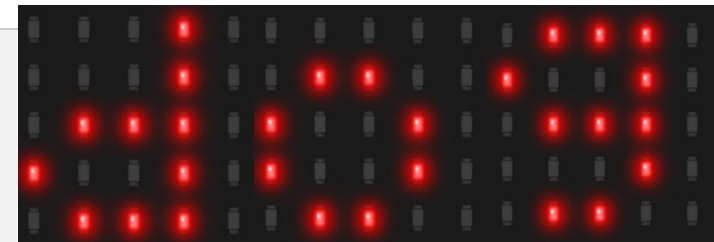


# Reusing variables



We can replace values in variables and show it:

```
animal = "dog"  
display.scroll(animal)  
animal = "cat"  
display.scroll(animal)  
animal = animal + "dog"  
display.scroll(animal)
```



What will this scroll?

# Reusing variables



We can replace values in variables and show it:

```
animal = "dog"
```

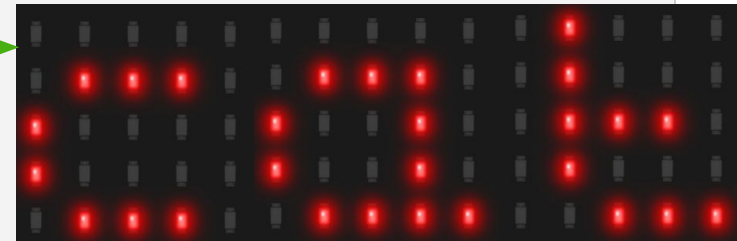
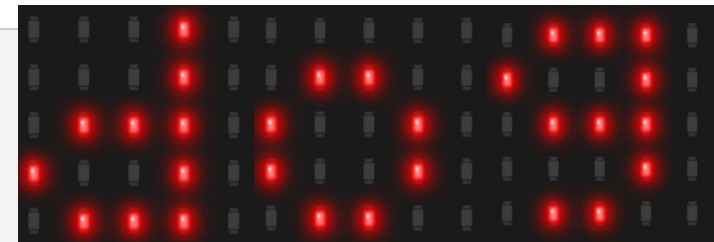
```
display.scroll(animal) →
```

```
animal = "cat"
```

```
display.scroll(animal) →
```

```
animal = animal + "dog"
```

```
display.scroll(animal)
```



What will this scroll?

# Reusing variables



We can replace values in variables and show it:

```
animal = "dog"
```

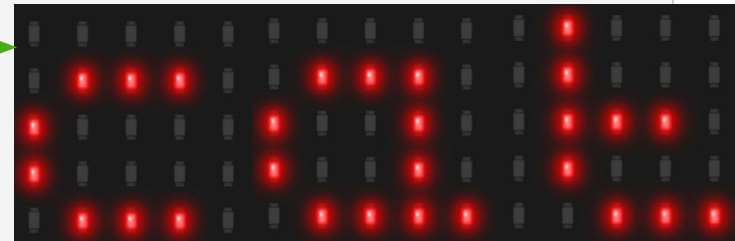
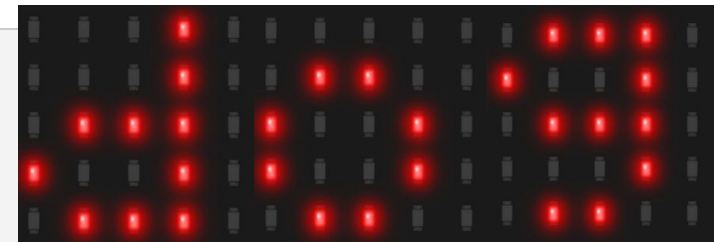
```
display.scroll(animal)
```

```
animal = "cat"
```

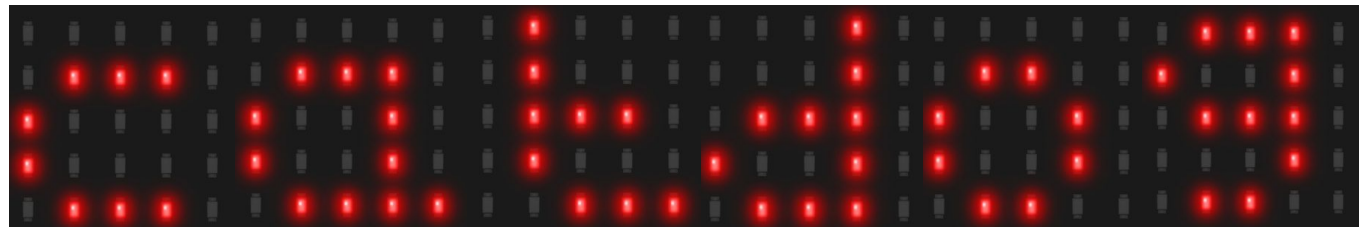
```
display.scroll(animal)
```

```
animal = animal + "dog"
```

```
display.scroll(animal)
```



What will this scroll?



# 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!

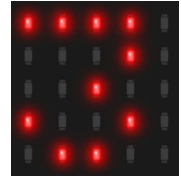
Can you guess what each  
`display.scroll` will do?

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

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)
```

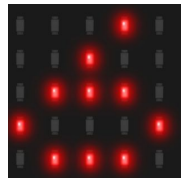
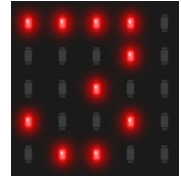
```
display.scroll(x + x)
```

```
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)
```

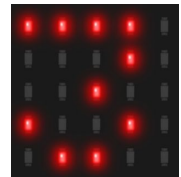
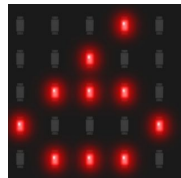
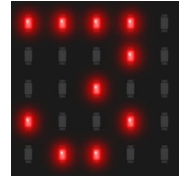
```
display.scroll(x + x)
```

```
y = x
```

```
display.scroll(y)
```

```
y = y + 1
```

```
display.scroll(y)
```



# Variables

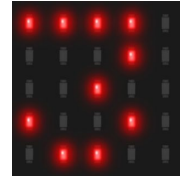


Your turn!

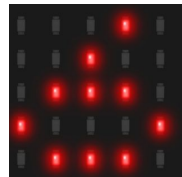
Can you guess what each  
`display.scroll` will do?

```
x = 3
```

```
display.scroll(x)
```

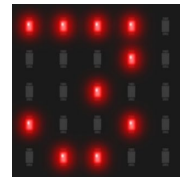


```
display.scroll(x + x)
```



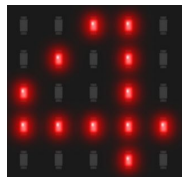
```
y = x
```

```
display.scroll(y)
```



```
y = y + 1
```

```
display.scroll(y)
```



# 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?

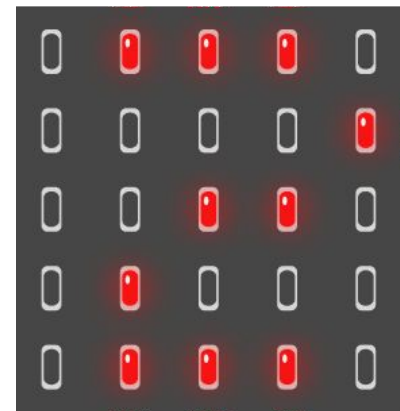
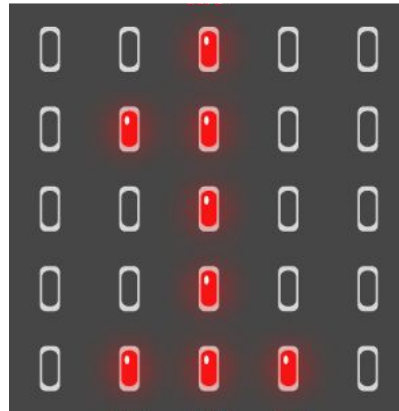
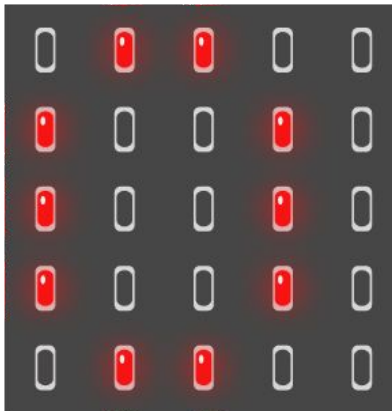
```
i = 0
while i < 3:
    display.scroll(i)
    i = i + 1
```



# Introducing ... while loops!

## What do you think this does?

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

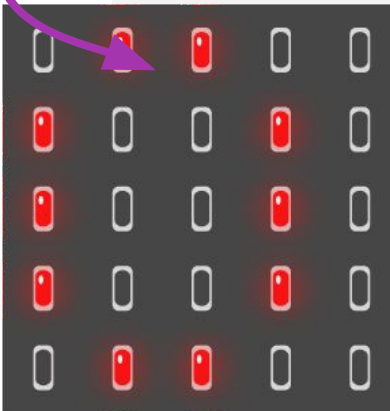
## One step at a time!

Print !

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

MY VARIABLES

i = 0



# Introducing ... while loops!

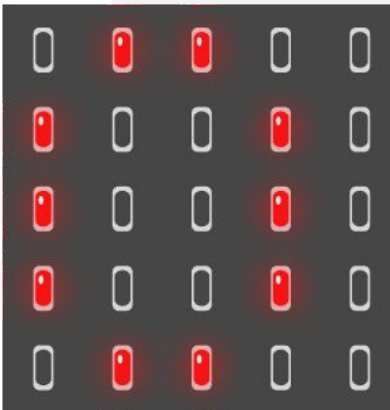
## One step at a time!

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

MY VARIABLES

~~i = 0~~  
i = 1

UPDATE  
TIME!

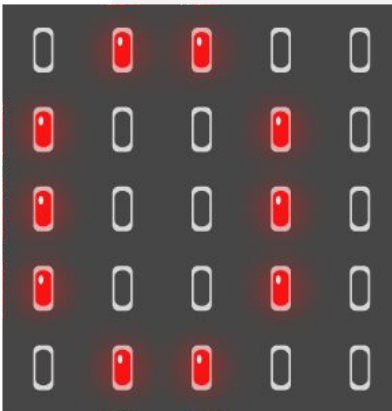


# Introducing ... while loops!

## 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!

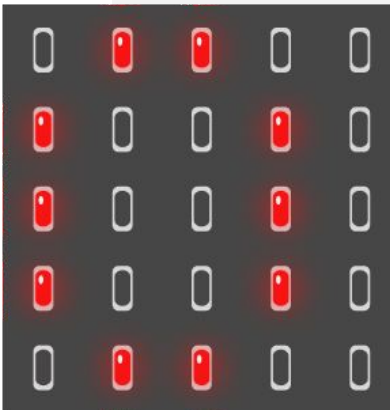
## One step at a time!

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!

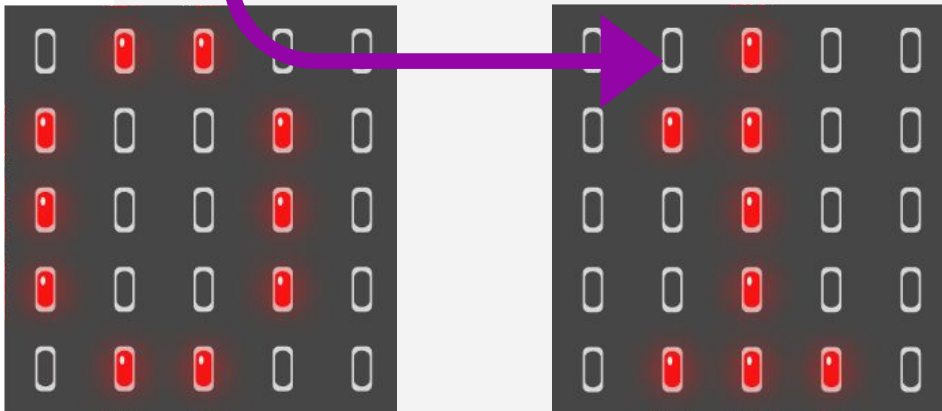
## One step at a time!

Print !

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

MY VARIABLES

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





# Introducing ... while loops!

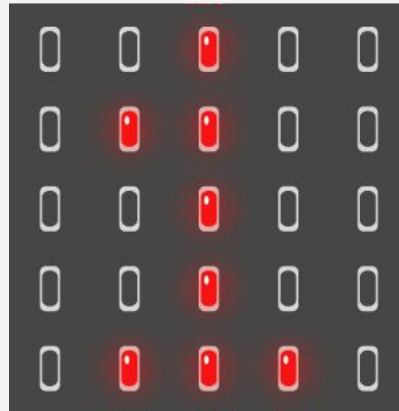
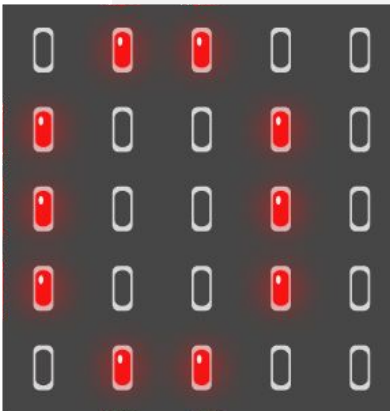
## One step at a time!

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

MY VARIABLES

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

UPDATE  
TIME!



# Introducing ... while loops!

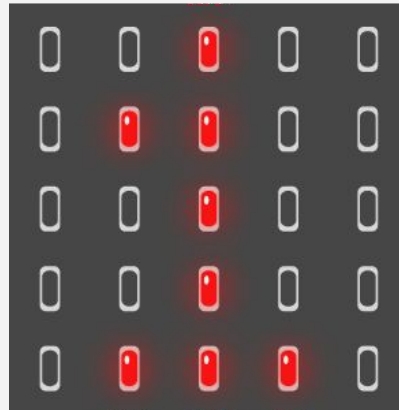
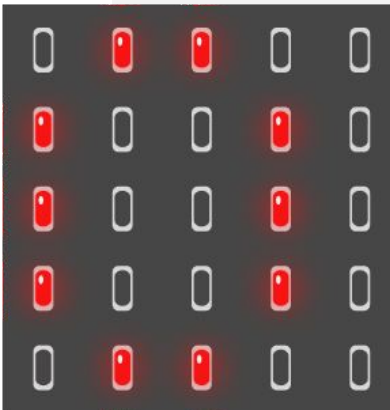
## One step at a time!

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!

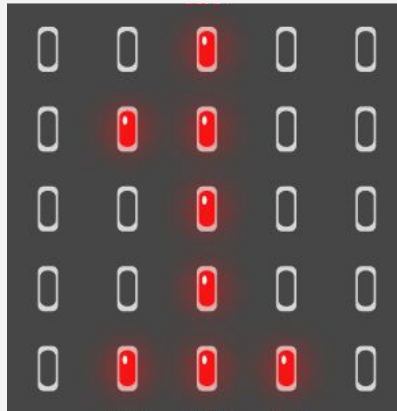
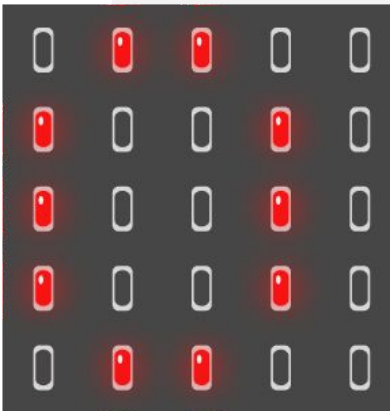
## One step at a time!

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!

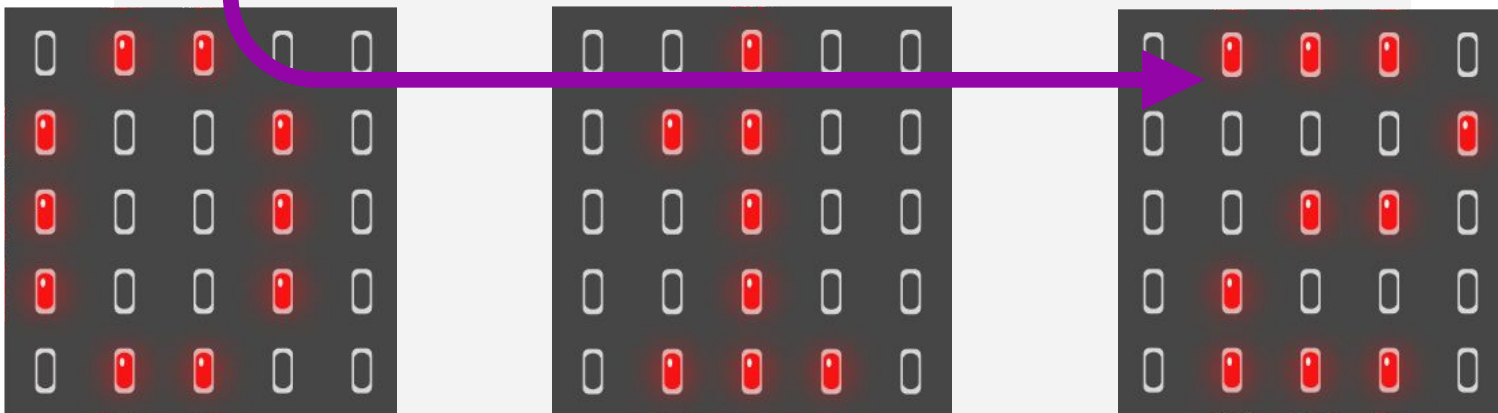
## One step at a time!

Print !

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

MY VARIABLES

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



# Introducing ... while loops!

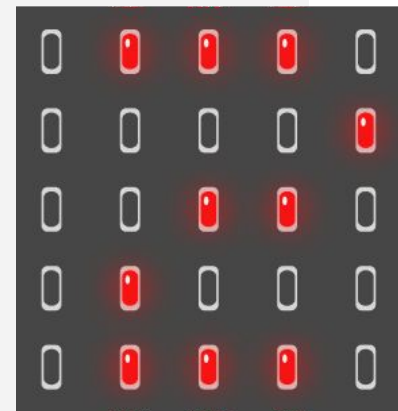
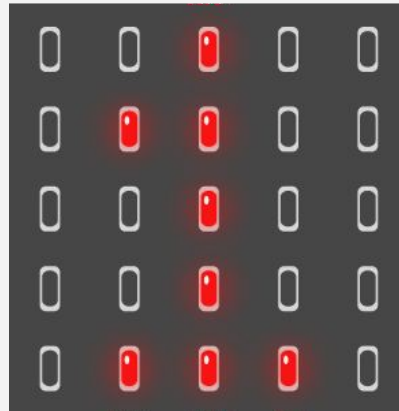
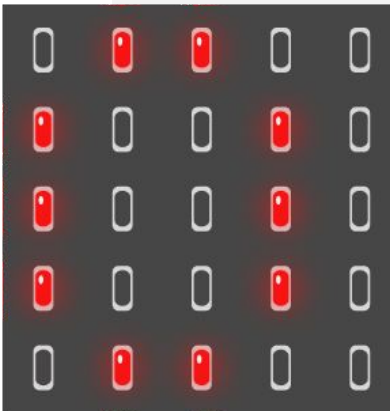
## One step at a time!

```
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!

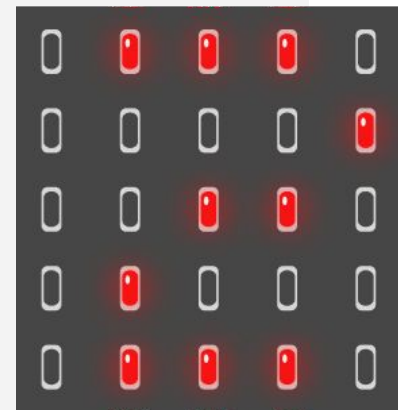
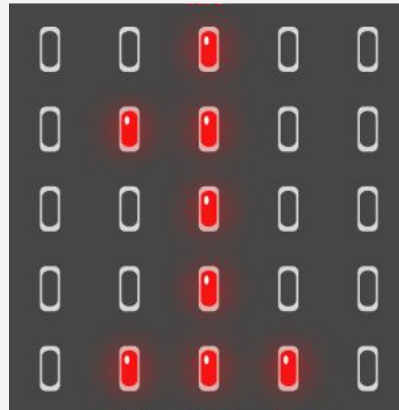
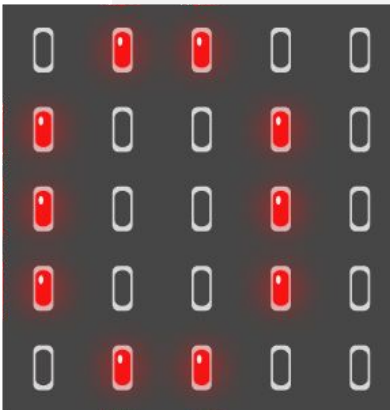
## One step at a time!

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!

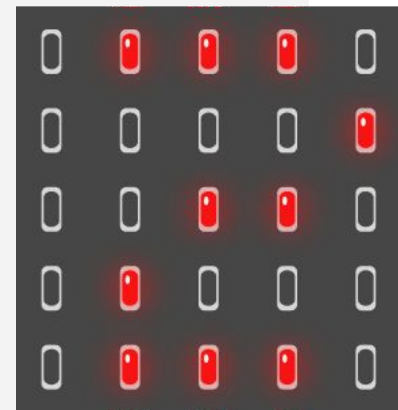
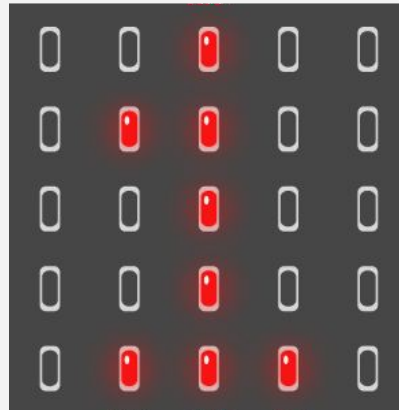
```
i = 0
while i < 3:
    display.scroll(i)
    i = i + 1
```

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:
```

Code to repeat

```
    display.scroll(i)
```

```
    i = i + 1
```

Update the loop variable

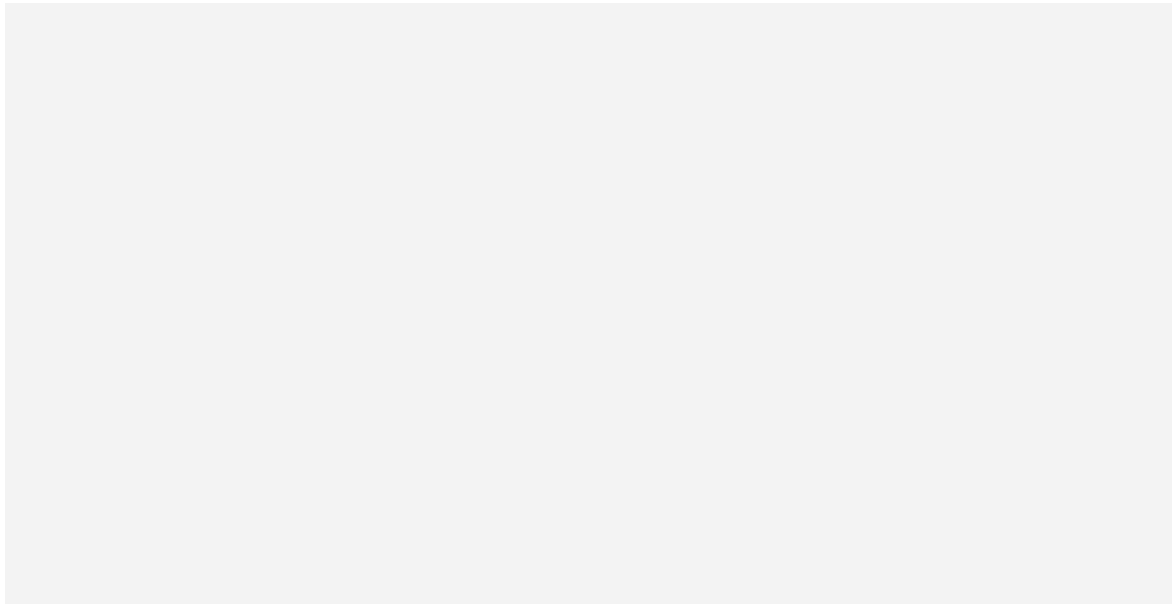




# What happens when.....

What happens if we forget to update the loop variable?

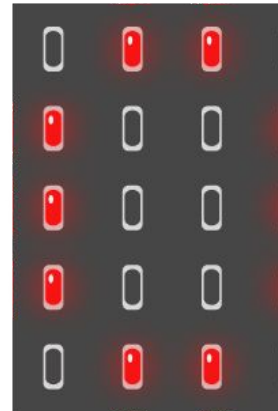
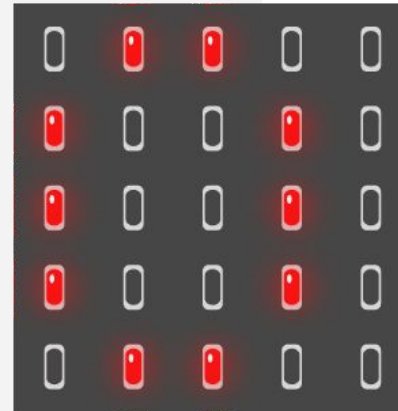
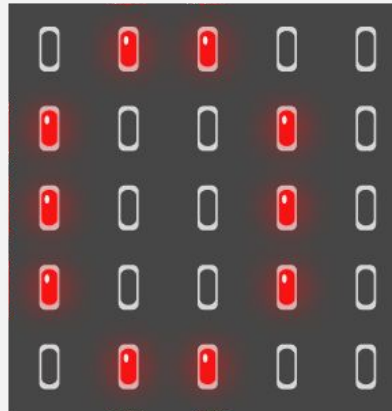
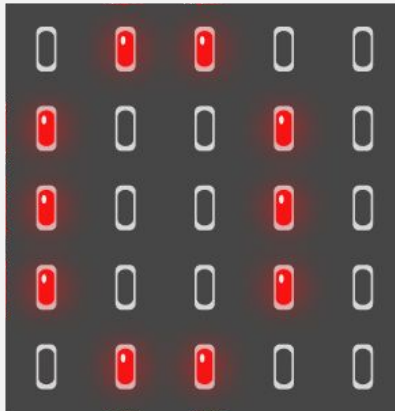
```
i = 0
while i < 3:
    display.scroll(i)
```



# 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



# Infinite loops and Micro:Bits

Your Micro:Bit runs code **really fast!**

When we want to do things like button presses, we need it to check for a button press over and over and over again so that it's just always checking!

This means we want to put almost all of our code **inside** the while True loop!

The only code we want to have **before** the while loop is stuff we only want to happen **once** like setting variables or saying hello!




# 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!





# 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

# 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!

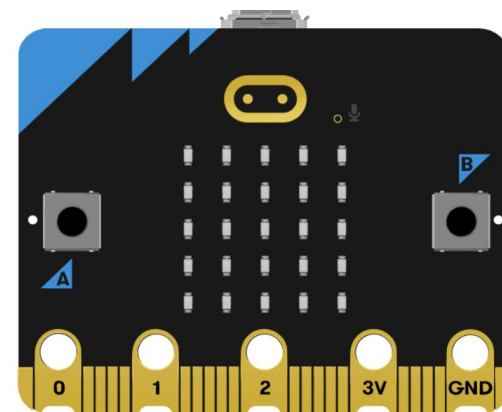
Your Micro:Bit has 2 buttons: Button A and Button B

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?



# 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? **It will show one face and then the other immediately**

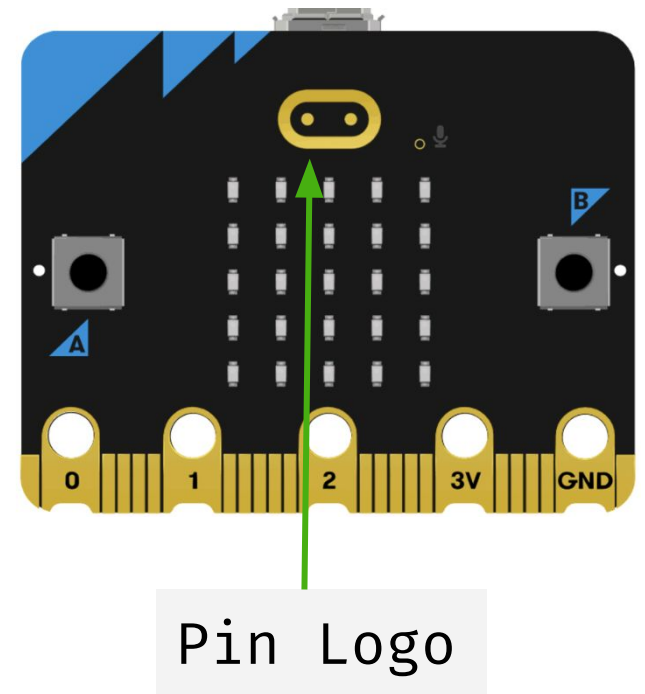


# Pin Logo!

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

# 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**



# Accelerometer!

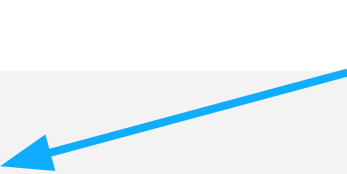
Your micro:bit has a motion sensor.

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

To use the accelerometer, we need a while loop. You can use this code to detect when 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



# Project Time!

**Let's use our MicroBit!**  
**Try Parts 2 & 3 of your Workbook!**

The tutors will be around to help!



# Functions!

Simpler, less repetition, easier to read code!



# How functions fit together!

Functions are like factories!

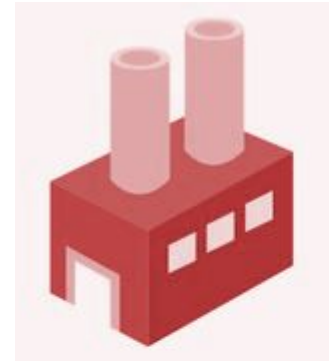
**Your main factory!**



**Timber Mill**



**Metal Worker**

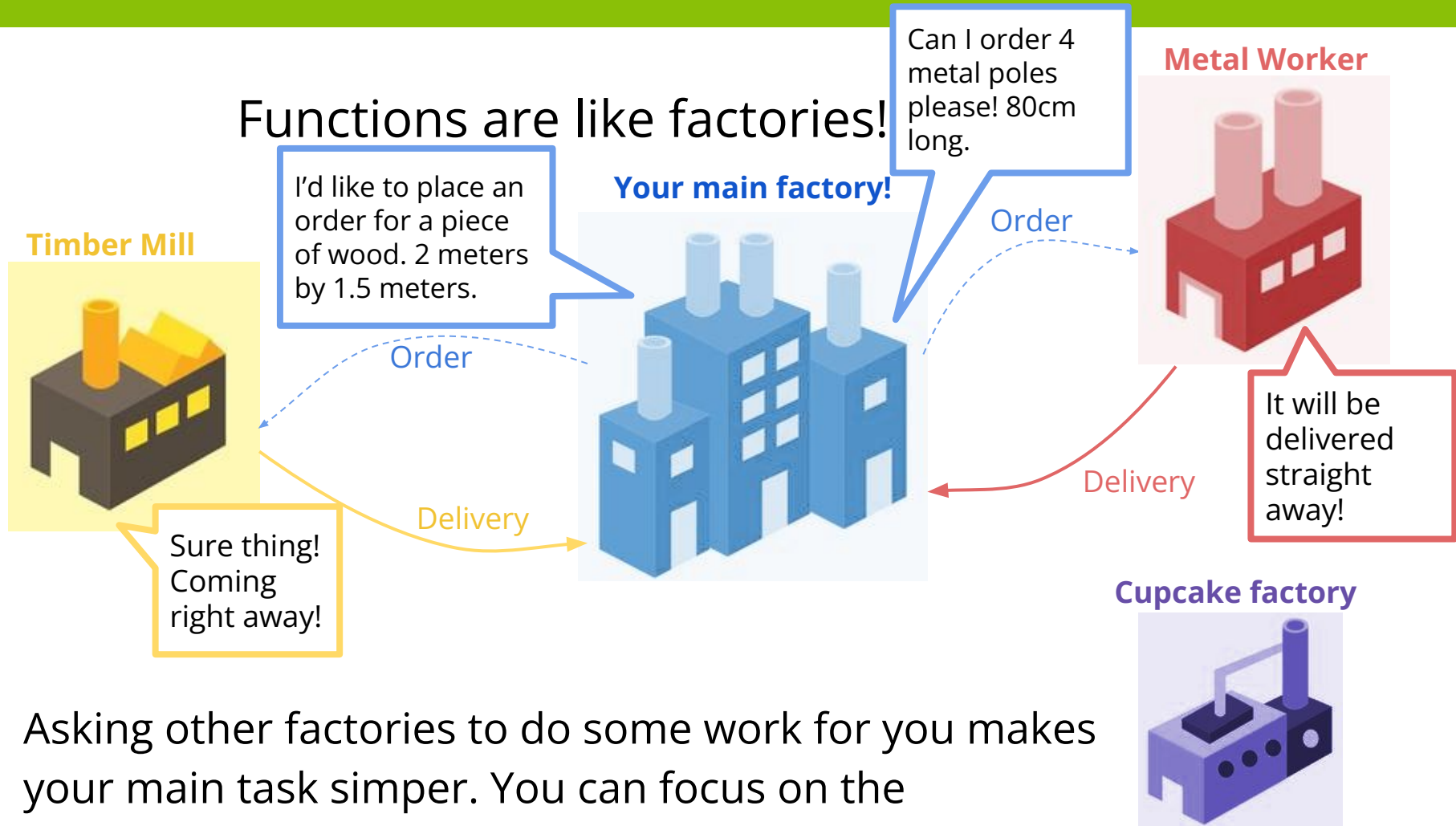


Running a factory doesn't mean doing all the work yourself, you can get other factories to help you out!

**Cupcake factory**



# How functions fit together!



Asking other factories to do some work for you makes your main task simpler. You can focus on the assembly!





# How functions fit together!

Functions are like factories!

**Your main factory!**

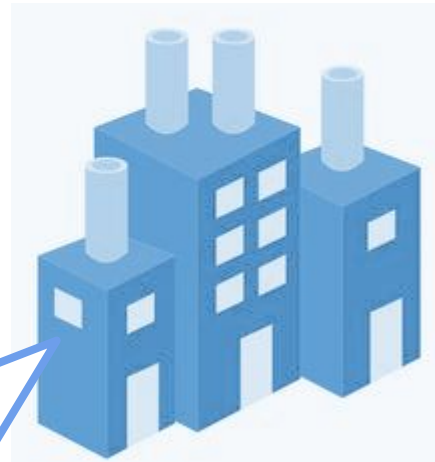
**Timber Mill**



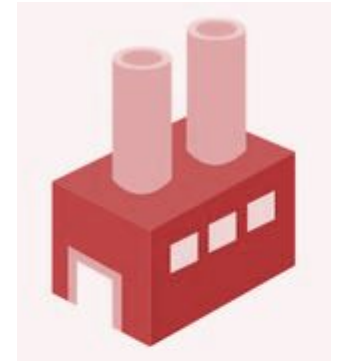
Look at this beautiful table I made!



Outsourcing made it simple!



**Metal Worker**



**Cupcake factory**



# How functions fit together!

## Your main code!



You can write a bunch of helpful functions to **simplify** your **main goal**!

You can **write** these **once** and then **use** them **lots** of times!  
They can be **anything** you like!

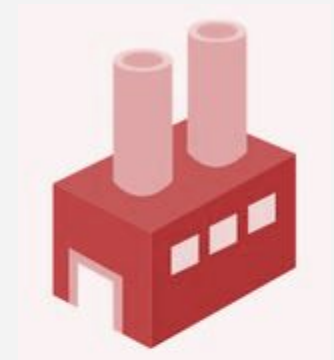
Helps with printing nicely



Uses stats to make decisions



Does calculations



# Don't reinvent the wheel

We're already familiar with some python in built functions like print and input!

**There's lots of functions python gives us to save us reinventing the wheel!**

For instance we can use len to get the length of a string, rather than having to write code to count every letter!

```
>>> len("Hello world")
```



# Don't reinvent the wheel

We're already familiar with some python in built functions like print and input!

**There's lots of functions python gives us to save us reinventing the wheel!**

For instance we can use len to get the length of a string, rather than having to write code to count every letter!

```
>>> len("Hello world")  
11
```



# Don't reinvent the wheel

We're already familiar with some python in built functions like print and input!

**There's lots of functions python gives us to save us reinventing the wheel!**

For instance we can use len to get the length of a string, rather than having to write code to count every letter!

```
>>> len("Hello world")  
11
```

## What do these do?:

```
>>> name = "Renee"  
>>> len(name)  
  
>>> int("6")  
  
>>> str(6)
```



# Don't reinvent the wheel

We're already familiar with some python in built functions like print and input!

**There's lots of functions python gives us to save us reinventing the wheel!**

For instance we can use len to get the length of a string, rather than having to write code to count every letter!

```
>>> len("Hello world")  
11
```

## What do these do?:

```
>>> name = "Renee"  
>>> len(name)  
5  
  
>>> int("6")  
  
>>> str(6)
```



# Don't reinvent the wheel

We're already familiar with some python in built functions like print and input!

**There's lots of functions python gives us to save us reinventing the wheel!**

For instance we can use len to get the length of a string, rather than having to write code to count every letter!

```
>>> len("Hello world")  
11
```

## What do these do?:

```
>>> name = "Renee"  
>>> len(name)  
5  
  
>>> int("6")  
6  
  
>>> str(6)
```



# Don't reinvent the wheel

We're already familiar with some python in built functions like print and input!

**There's lots of functions python gives us to save us reinventing the wheel!**

For instance we can use len to get the length of a string, rather than having to write code to count every letter!

```
>>> len("Hello world")  
11
```

## What do these do?:

```
>>> name = "Renee"  
>>> len(name)  
5  
  
>>> int("6")  
6  
  
>>> str(6)  
"6"
```





# Defining your own functions

Built in functions are great! But sometimes we want custom functions!

Defining our own functions means:

- We cut down on repeated code
- Nice function names makes our code clear and easy to read
- We can move bulky code out of the way



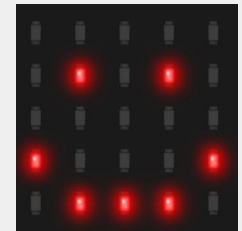
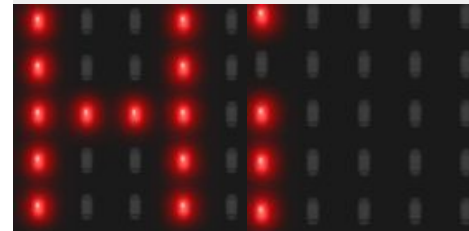
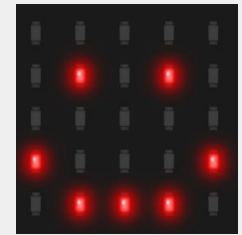
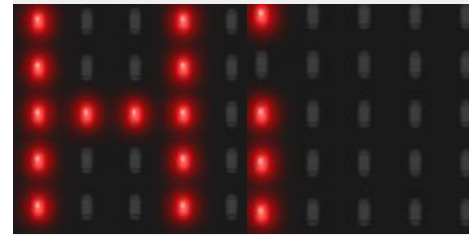
# Defining your own functions

Then you can use your function by calling it!

```
def say_hello():  
    display.scroll("Hi")  
    display.show(Image.HAPPY)
```

```
say_hello()  
say_hello()
```

Which will do this!



# Defining your own functions

Then you can use your function by calling it!

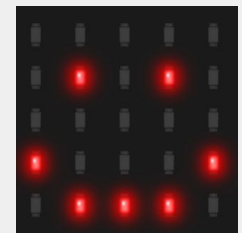
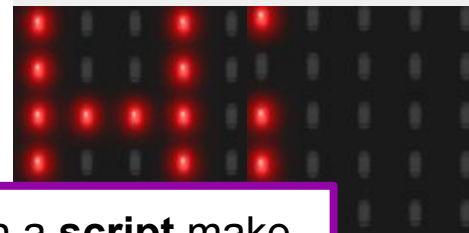
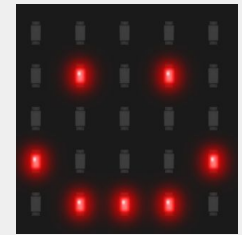
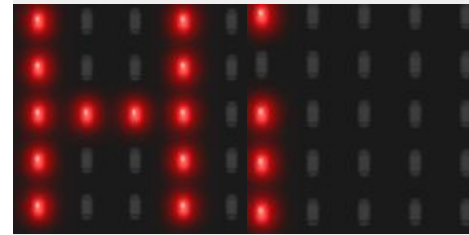
```
def say_hello():  
    display.scroll("Hi")  
    display.show(Image.HAPPY)
```

```
say_hello()  
say_hello()
```

When using a function in a **script** make sure you define the function first.

It doesn't matter if you call it from inside another function though!

Which will do this!

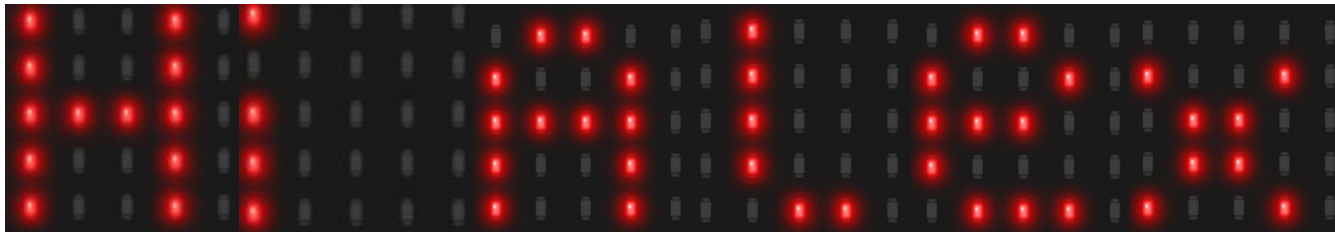


# Functions often need extra information

Functions are more useful if we can change what they do

We can do this by giving them arguments (aka parameters)

```
def hello(person):  
    display.scroll('Hi ' + person)  
  
hello('Alex')
```

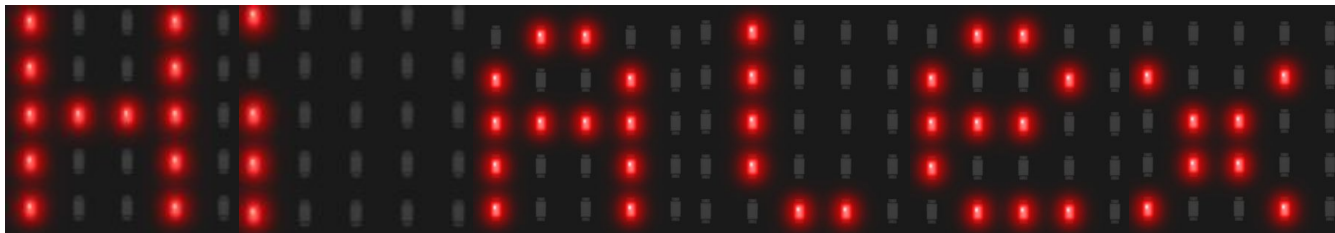


# Functions often need extra information

Functions are more useful if we can change what they do

We can do this by giving them arguments (aka parameters)

```
def hello(person):  
    display.scroll('Hi ' + person)  
  
hello('Alex')
```



Here, we give the hello() function a name

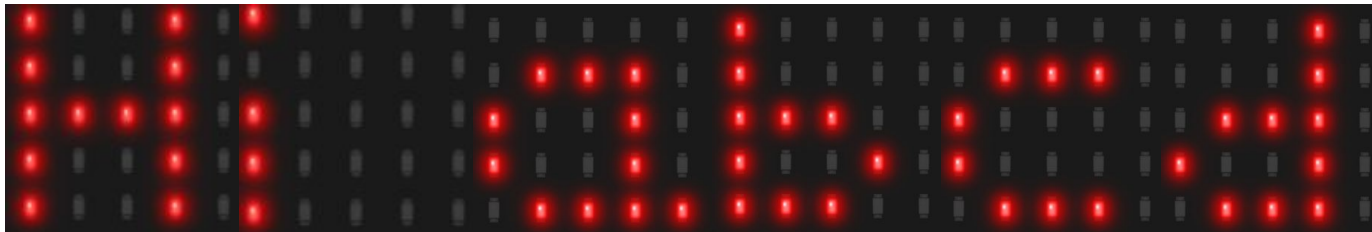
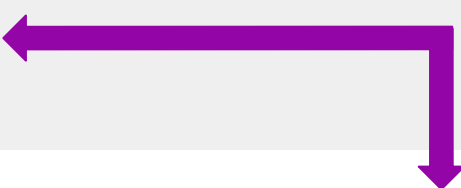
Any string will work



# Functions often need extra information

Functions are more useful if we can change what they do  
We can do this by giving them arguments (aka parameters)

```
def hello(person):  
    display.scroll('Hi ' + person)  
  
hello('abcd')
```



# Functions can take multiple arguments

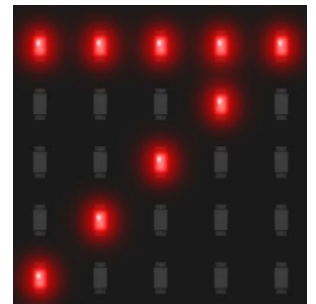
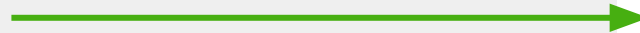
Often we want to work with multiple pieces of information.

You can actually have as many parameters as you like!

This function takes two numbers, adds them together and prints the result.

```
def add(x, y):  
    display.scroll(x + y)
```

```
add(3, 4)
```



# Arguments stay inside the function

The arguments are not able to be accessed outside of the function declaration.

```
def hello(person):  
    display.scroll('Hello, ' + person + '!')  
  
display.scroll(person)  
Traceback (most recent call last):  
File "<stdin>", line 1, in <module>  
NameError: name 'person' is not defined
```



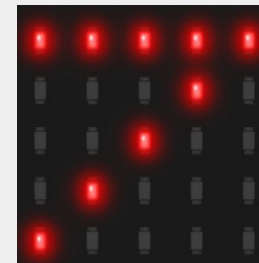


# Variables stay inside the function

Neither are variables made inside the function. They are **local variables**.

```
def add(x, y):  
    z = x + y  
    display.scroll(z)
```

```
add(3, 4) →
```



```
display.scroll(z)
```

```
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
NameError: name 'z' is not defined
```



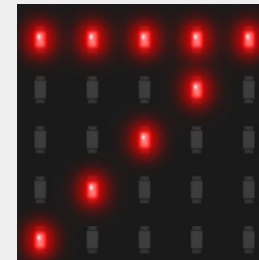
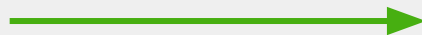
# Global variables are not affected

Changing a variable in a function **only changes it *inside* the function.**

```
z = 1
```

```
def add(x, y):  
    z = x + y  
    display.scroll(z)
```

```
add(3, 4)
```



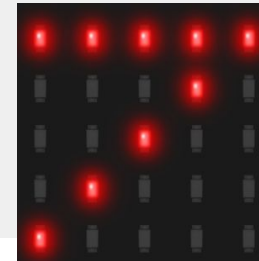
# Global variables are not affected

Changing a variable in a function **only changes it *inside* the function.**

```
z = 1
```

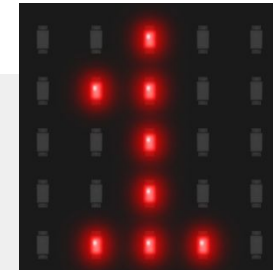
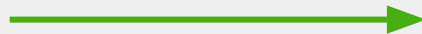
```
def add(x, y):  
    z = x + y  
    display.scroll(z)
```

```
add(3, 4)
```

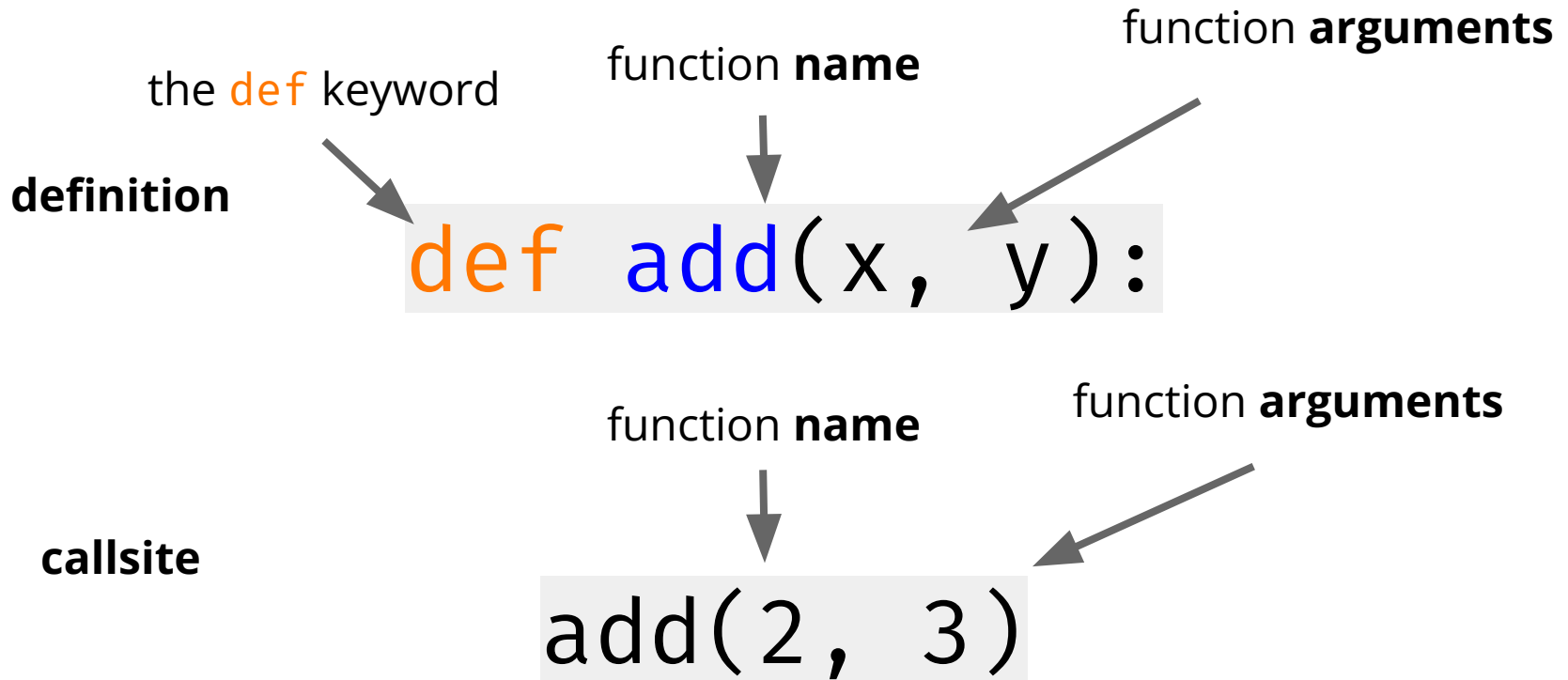


What's the value of z now?

```
display.scroll(z)
```

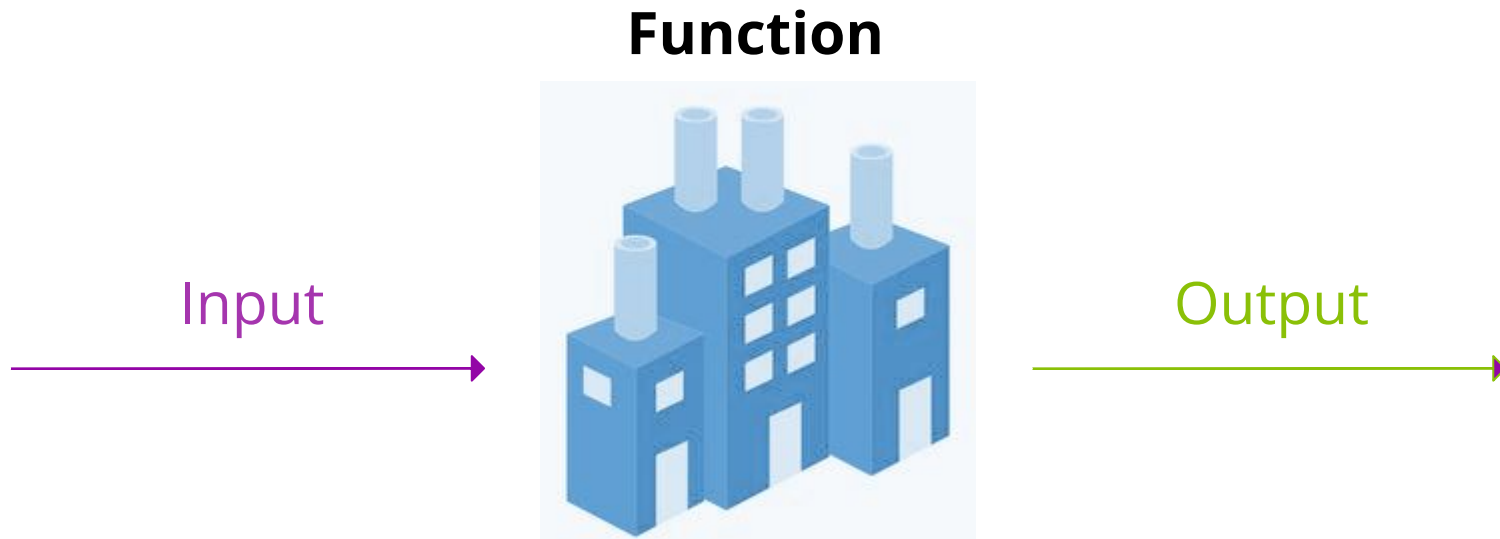


# Recap: A function signature



# Return Keyword

Remember functions are like a factory



But when we want our output to be sent to another factory we need to use the keyword: **return**

# Return Keyword

Here's an example

```
def get_name():  
    return "Alex"  
  
def hello()  
    name = get_name()  
    display.scroll("Hi " + name)
```

What's displayed?

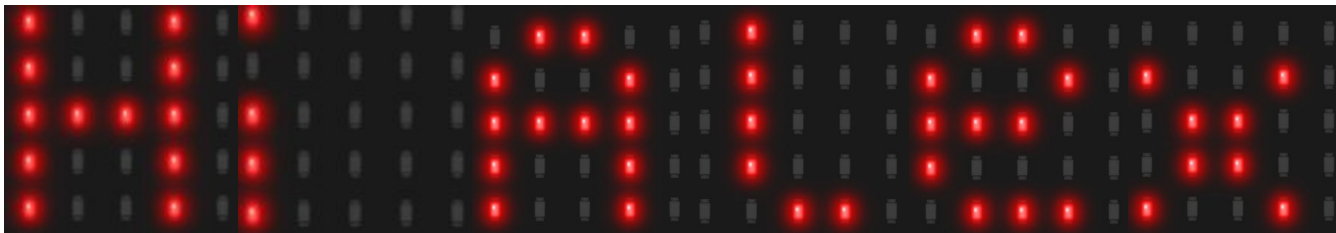


# Return Keyword

Here's an example

```
def get_name():  
    return "Alex"  
  
def hello()  
    name = get_name()  
    display.scroll("Hi " + name)
```

What's displayed?



# Project time!

Now go be functional.

**Do the next part of the project!**

**Try to do Part 4-6**

The tutors will be around to help!





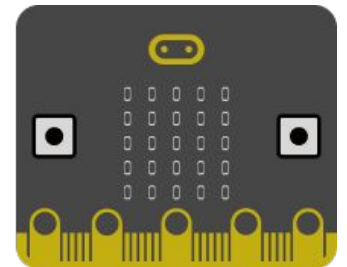
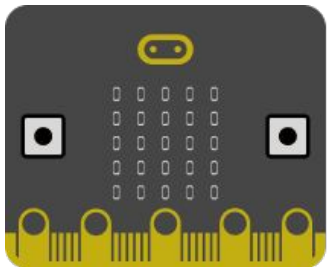
# Micro:Bit Radio



# We can use the radio to talk to each other!

All of your Micro:bits have the ability to send and receive radio messages. We are going to use this to make our Micro:bits communicate.

To send radio messages, our Micro:bits send out special invisible light waves at different times to symbolise a series of 1s and 0s, which other Micro:bits can then translate into words and information.



# Radio

Your Micro:Bit can send messages to other Micro:Bits using radio waves!

It only takes a few lines of code to make this work!

1. We have to tell the Micro:Bit that we want to use the radio:

```
import radio
```

2. We need to turn the Radio on:

```
radio.on()
```

3. We need to send a message:

```
radio.send("Hello World")
```

4. We want to receive a message:

```
message = radio.receive()
```



# Radio Groups

We need to set our radio to communicate on a certain group, otherwise all our Micro:Bits will try to talk to each other! This will get confusing for the Micro:Bit.

After you turn the radio on, set the group channel!

```
radio.config(group=100)
```

Your tutors will give you a group number to use.



# Radio Example

What do you think this code does?

Micro:Bit 1

```
import radio

radio.on()
radio.config(group=100)

while True:
    if button_a.is_pressed():
        radio.send("Hello!")

    if button_b.is_pressed():
        radio.send("World!")
```

Micro:Bit 2

```
import radio

radio.on()
radio.config(group=100)

while True:
    message = radio.receive()
    if message:
        display.scroll(message)
```

Why do you think it's important to check the message?



# Project Time!

**Let's use our MicroBit!**  
**Try Parts 7 & 8 of your Workbook!**

The tutors will be around to help!

