

Welcome to the labs!

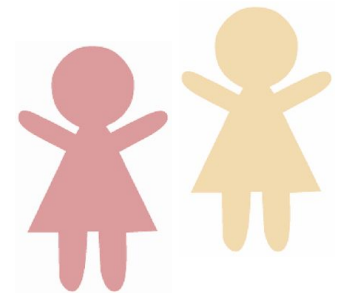
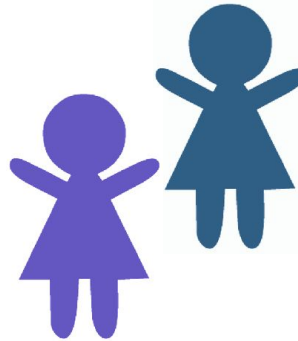
Bop It! - Micro:Bit

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

bit.ly/gpn-2019-4

You can see:

- These **slides** (to take a look back or go on ahead).
- A digital copy of your **workbook**.
- Help bits of text you can **copy and paste**!

There's also links to places where you can do more programming!

Tell us you're here!

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

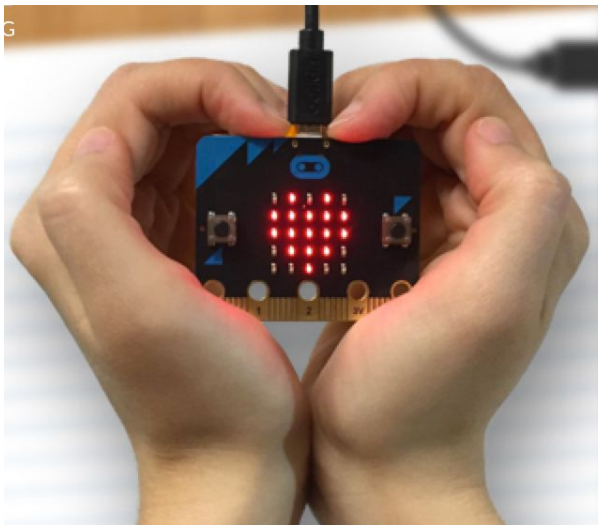
Today's project!

Bop It! - Micro:Bit

Micro:Bits - IRL

Today we have real life MicroBits to use!

But sad 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 are about \$25 .

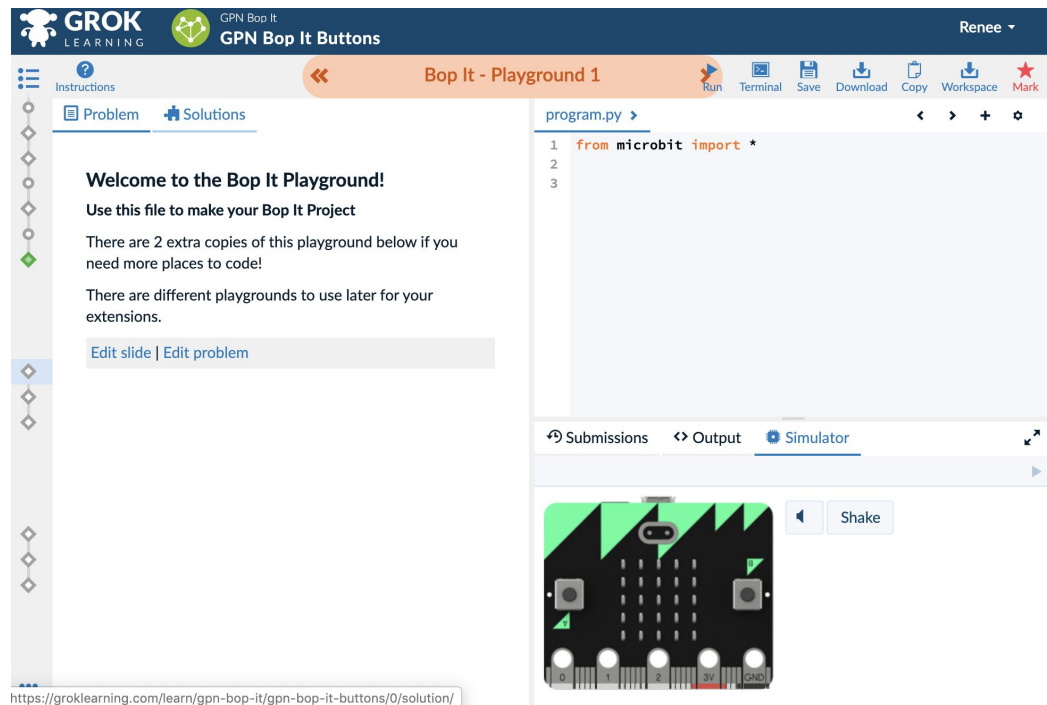
Find out where to buy them at the bottom of this page

<https://groklearning.com/microbit/>

Micro:Bits - Digital

We also have an emulator in Grok Learning! Which you can use after the workshop! 🎉

The emulator is a fast way to test the code without downloading it. Use it while you're still working on your code. And then try it in real life.



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 some extra!

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

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!

SALAD INSTRUCTIONS

Programming is like a recipe!

Computers do EXACTLY what you say, every time.

Which is great if you give them a good recipe!

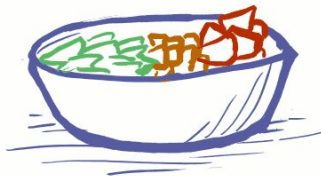
1) GET A LETTUCE HEAD, A CARROT, A TOMATO, A KNIFE, AND A BOWL



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 you get it out of order....

A computer wouldn't know this recipe was wrong!

SALAD INSTRUCTIONS

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



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



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



4) MIX THE CONTENTS OF THE BOWL



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/thing has strengths!



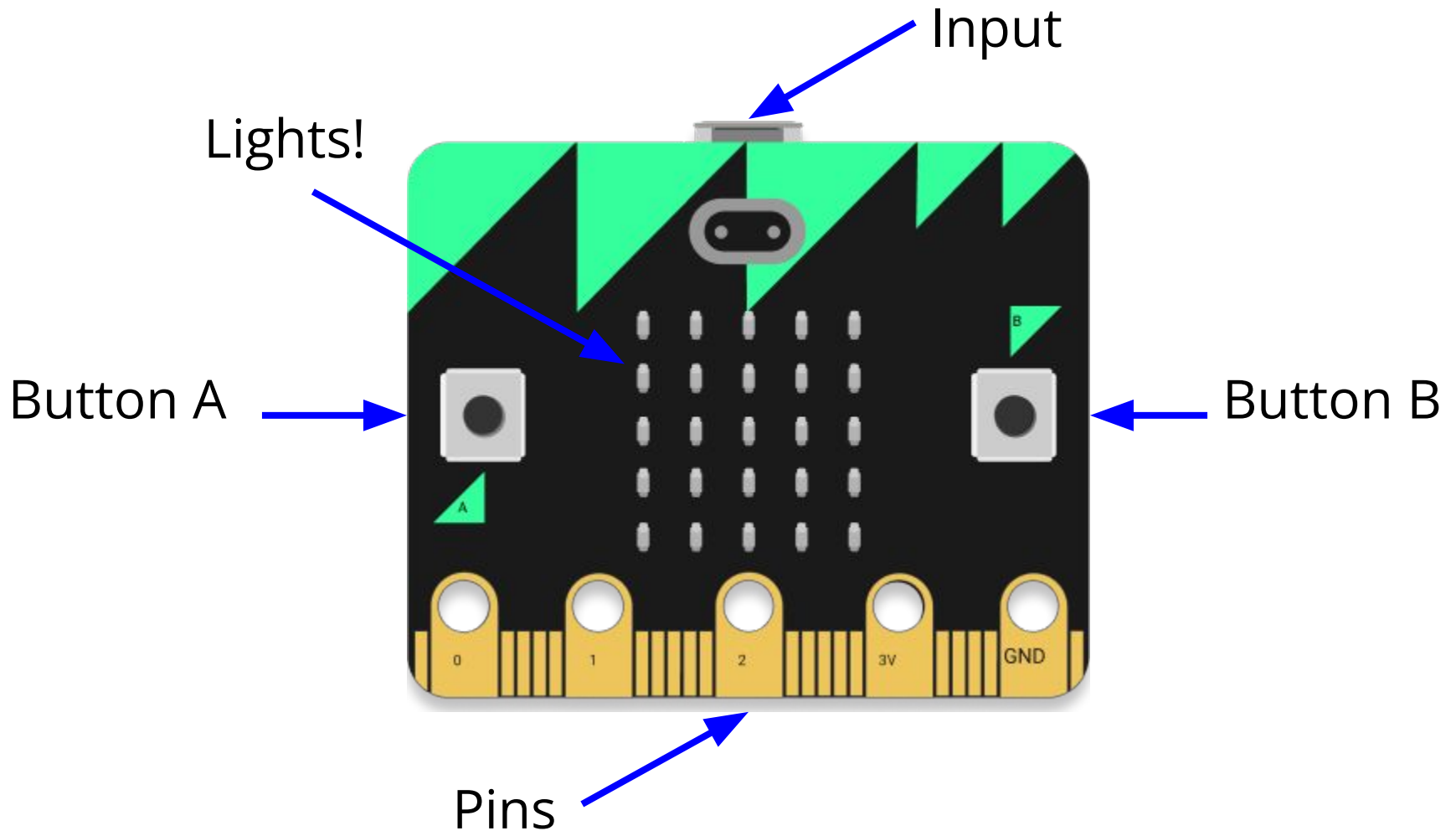
- Understand instructions despite:
 - Spelling mistakes
 - Typos
 - Confusing parts
- Solve problems
- Tell computers what to do
- Get smarter every day



- Does exactly what you tell it
- Does it the same every time
- Doesn't need to sleep!
- Will work for hours on end!
- Get smarter when you tell it how

Intro to Micro:Bit

What is a Micro:Bit?



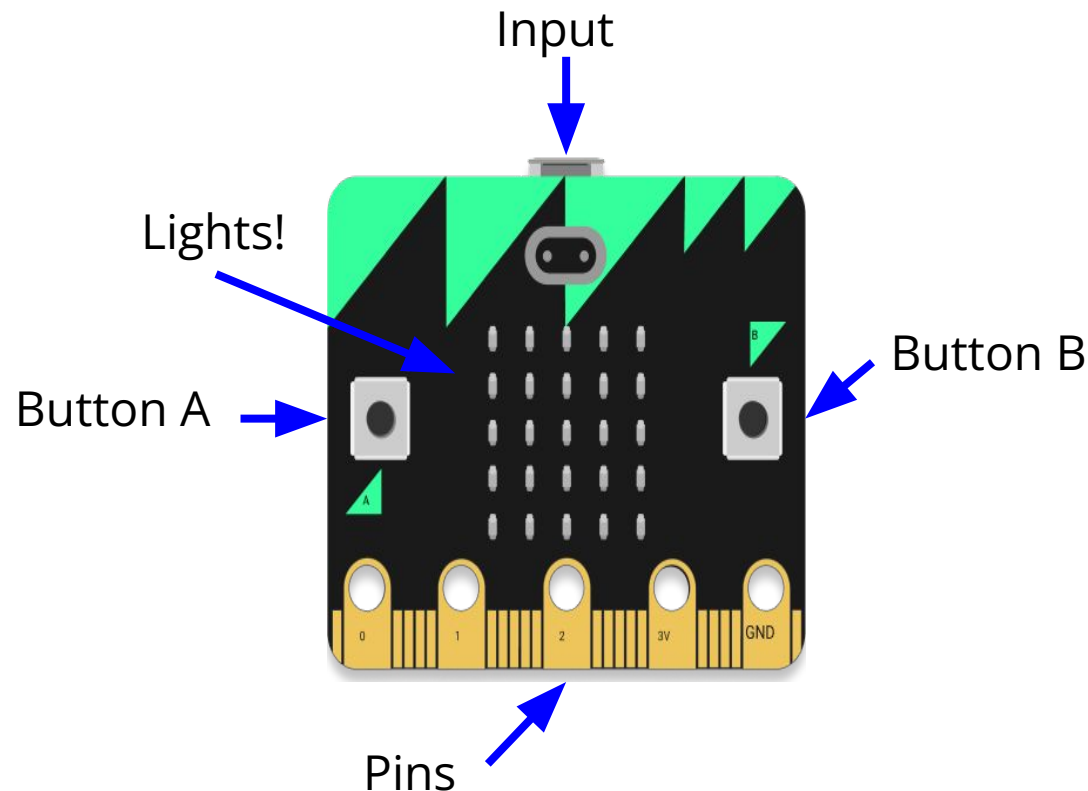
What do the different bits do?

Input: This is how we get code onto our Micro:Bit and tell it what to do!

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

Lights: Each of these is a little light that we can turn on. When we turn them on in different patterns we can make images!

Pins: These let us connect the Micro:Bit to other devices like extra buttons



How do we write code for it?

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

Because they have buttons, lights and other cool stuff we need to make sure that we tell Python that we want the extra stuff for Micro:Bits. We do this using this line of code:

```
from microbit import *
```

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

Using Grok Learning!

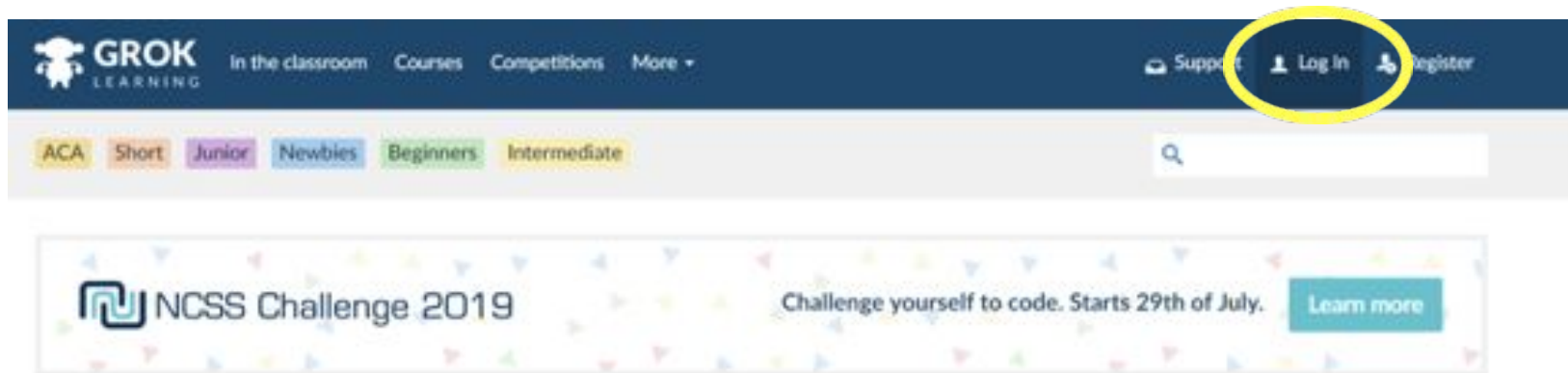
Today we will be using Grok Learning to program our Micro:Bits.

Grok has a great digital Micro:Bit which makes learning how to program them really easy!



Getting to Grok!

Go to groklearning.com



**Log in with the email address you signed up to
GPN with**

Getting into the GPN Workshop

Next go to your profile name and click “Workshops”



***You will be asked for a workshop code.
Our code is **gpn-syd-bop-it*****

A screenshot of a web form titled 'Welcome'. It contains the instruction 'If you are at a workshop and have a workshop code, enter it here:'. Below this is a text input field containing the code 'gpn-syd-bop-it'. At the bottom of the form is a dark blue button labeled 'Next'.

You can use your **school name or **Girls' Programming Network - University of Sydney** as your institution**

GPN MicroBit Playground

Once you're in the workshop click on the GPN Micro:Bit Playground



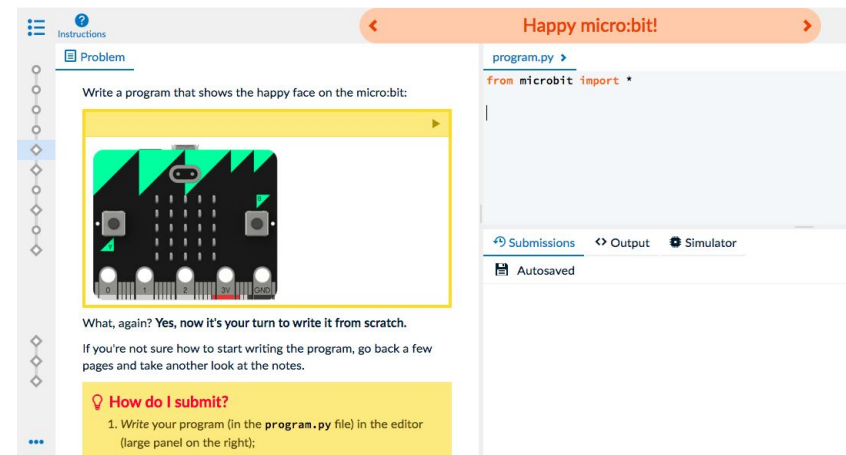
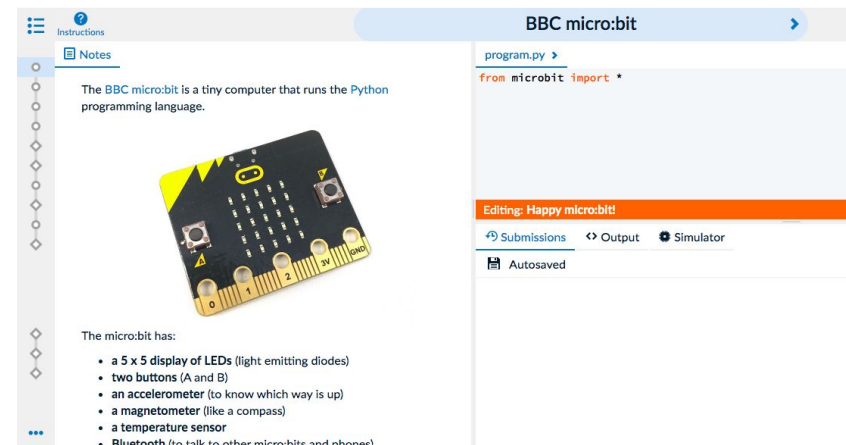
Slides and Problems

The first part of the workshop today we will be learning about Micro:Bits using Grok!

Grok has 2 different types of pages: slides and problems!

Slides look like this and teach you about the Micro:Bit

Problems look like this and they are your chance to practice what you've learned

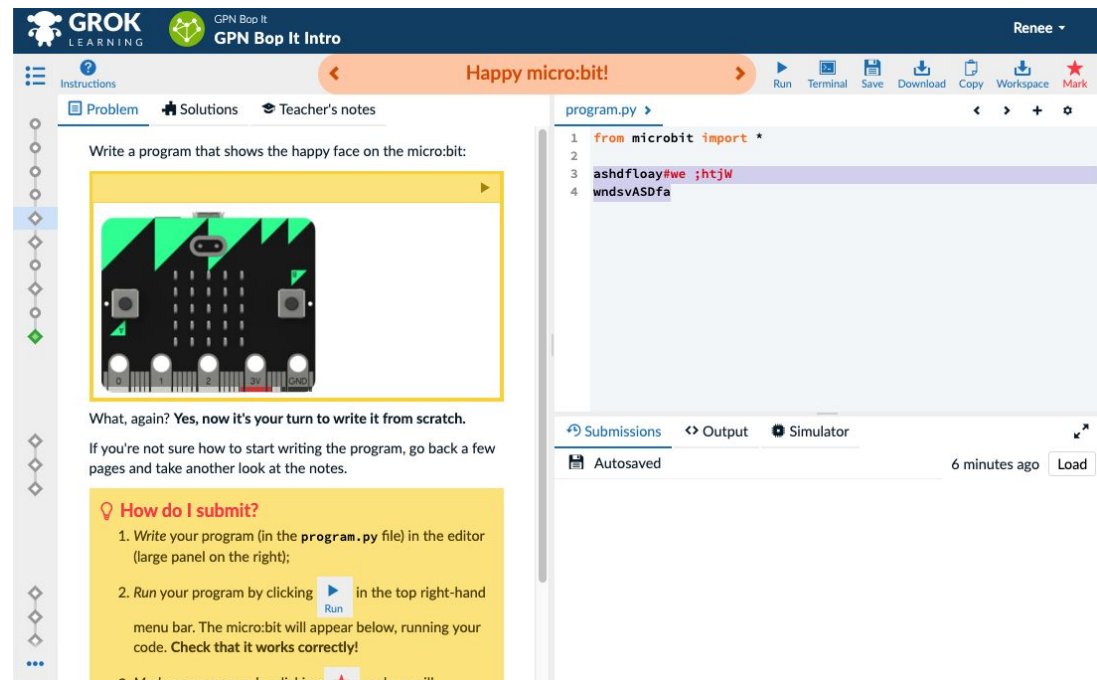


Let's do the first activity together

Let's start by making
and error!

Type some random
things into the code
box!

Press run!



The screenshot shows the GROK Learning GPN Bop It! interface. The main task is to write a program that shows the happy face on the micro:bit. The code editor on the right contains the following Python code:

```
1 from microbit import *
2
3 ashdflay#we ;htjW
4 wndsvASDfa
```

The interface also includes a sidebar with navigation options (Instructions, Problem, Solutions, Teacher's notes), a top bar with a 'Happy micro:bit!' message, and a bottom bar with 'Submissions', 'Output', and 'Simulator' tabs. The 'Submissions' tab shows an 'Autosaved' status from 6 minutes ago.

Let's do the first activity together

We can't see the error easily on the microbit!!
Let's click on **output** to see the error printed out.

The image shows two side-by-side screenshots of the Grok Learning GPN Bop It Intro interface, illustrating the process of running a program and viewing the output.

Left Screenshot: The interface shows the "Happy micro:bit!" page. The code editor displays the following code:

```
1 from microbit import *
2
3 ashdfloayfwe ;htjW
4 wndsvASDfa
```

The "Output" tab is selected, showing a simulated micro:bit display with a green background and a small robot icon. A blue arrow points from the "output" text in the text above to the "Output" tab in the interface.

Right Screenshot: The interface shows the same code, but the "Output" tab now displays a traceback error message:

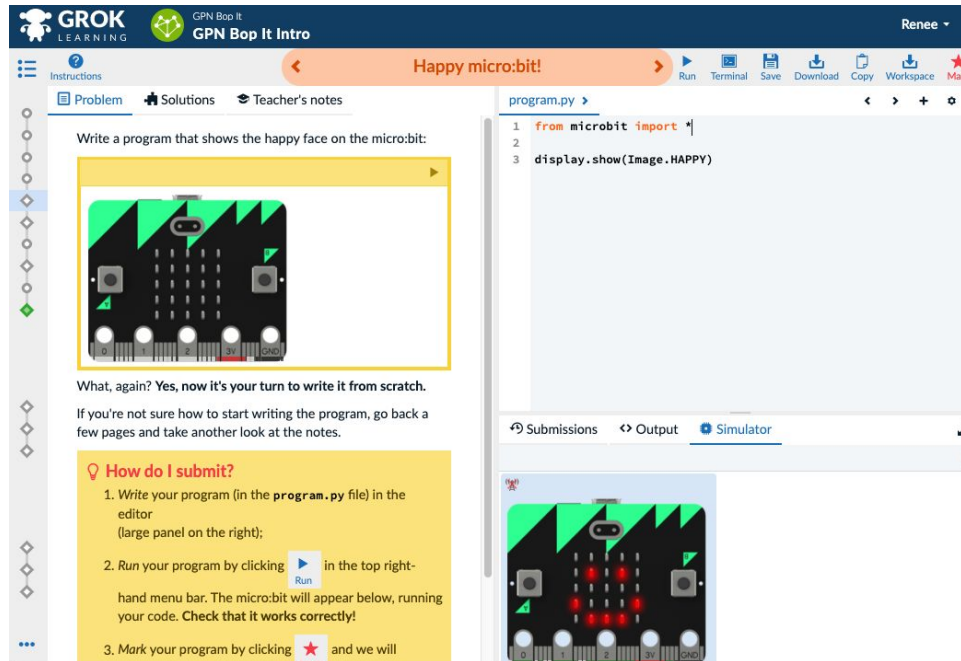
```
Traceback (most recent call last):
  File "__main__", line 3, in <module>
    ashdfloayfwe ;htjW
NameError: name 'ashdfloayfwe' is not defined
MicroPython v1.7.9-gb020eb on 2016-09-14; micro:bit with nRF51822
>>>
```

A blue arrow points from the "Output" tab in the left screenshot to the "Output" tab in the right screenshot, indicating the transition from a successful run to an error state.

Let's do the first activity together

Ok let's do it for real this time!

Look back at slide 2 to see how!



The screenshot shows the GROK Learning GPN Bop It Intro interface. The top navigation bar includes the GROK Learning logo, the title "GPN Bop It GPN Bop It Intro", and a user profile "Renee". Below the navigation bar, there are tabs for "Instructions", "Problem", "Solutions", and "Teacher's notes". The "Problem" tab is active, displaying the task: "Write a program that shows the happy face on the micro:bit:". Below the task, there is a visual representation of a micro:bit with a green background and a black face. The "Solutions" tab is also visible, showing a list of instructions: "1. Write your program (in the program.py file) in the editor (large panel on the right);", "2. Run your program by clicking [Run] in the top right-hand menu bar. The micro:bit will appear below, running your code. Check that it works correctly!", and "3. Mark your program by clicking [Star] and we will". The "Teacher's notes" tab is also visible, showing a list of instructions: "1. Write your program (in the program.py file) in the editor (large panel on the right);", "2. Run your program by clicking [Run] in the top right-hand menu bar. The micro:bit will appear below, running your code. Check that it works correctly!", and "3. Mark your program by clicking [Star] and we will". The "Program" tab is active, showing a code editor with the following code:

```
1 from microbit import *
2
3 display.show(Image.HAPPY)
```

 The "Simulator" tab is also visible, showing a visual representation of a micro:bit with a green background and a black face.

Using a Micro:Bit IRL

It's fun to mess around with the Micro:Bit in Grok but it's also really fun to see your code on a Micro:Bit in real life!

To get your code from Grok onto your Micro:Bit:

1. **Plug** your Micro:Bit into your computer
2. Click the **Download** button in Grok to download your code
3. **Drag** the downloaded .hex file onto your Micro:Bit (like you would with a usb)
4. **Wait for the red light** at the back to stop flashing and the code should be running!
5. If you want your code to start again from the beginning, press the **"reset"** button on the back



Onto the project!

Once you've done all the intro slides and problems it's time to work on our GPN Workbook of the day!

After the last problem there are a bunch of empty problem slides! This is where you will be writing your project code for today.

You can use the digital MicroBit to test your code and when you're happy with it, you can download it and put it on your real MicroBit!

The Display

Your Micro:Bit has a display! It is the 5 by 5 grid of little red LEDs on the front! You can do some cool stuff with the display like:

Scroll the words “Hello World” across the display

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

Show an image, like a happy face!

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

Project Time!

Let's get started!

Let's try use it in our project!
Try to do Part 0 - 1

The tutors will be around to help!

Variables, lists and random!

Storing things for later!

In our game we might have things we want to remember for later!

For example, a score or the list of moves.

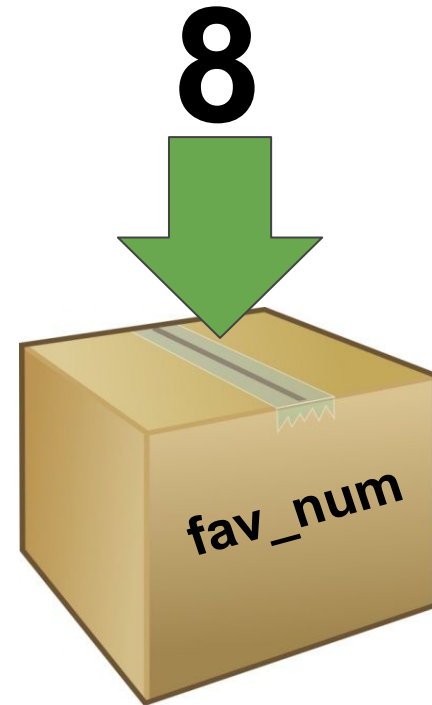
We might even want to change these things throughout the game (like increasing the score)

No Storing is Boring!

Computers remember things in "**variables**"

Variables are like putting things into a **labeled cardboard box**.

Let's make our favourite number 8 today!



Variables

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



$$\text{fav_num} - 6 \\ \Rightarrow 2$$

$$\text{fav_num} + 21 \\ \Rightarrow 29$$

$$\text{fav_num} * 2 \\ \Rightarrow 16$$

$$\text{fav_num} / 2 \\ \Rightarrow 4$$

Variables

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



$$\text{fav_num} - 6 \\ \Rightarrow 2$$

$$\text{fav_num} + 21 \\ \Rightarrow 29$$

$$\text{fav_num} * 2 \\ \Rightarrow 16$$

$$\text{fav_num} / 2 \\ \Rightarrow 4$$

Using variables

You set variables using one = symbol

You can update it by doing the same

Can you guess what each `print` will do?

```
>>> x = 3
>>> print(x)

>>> print(x + x)

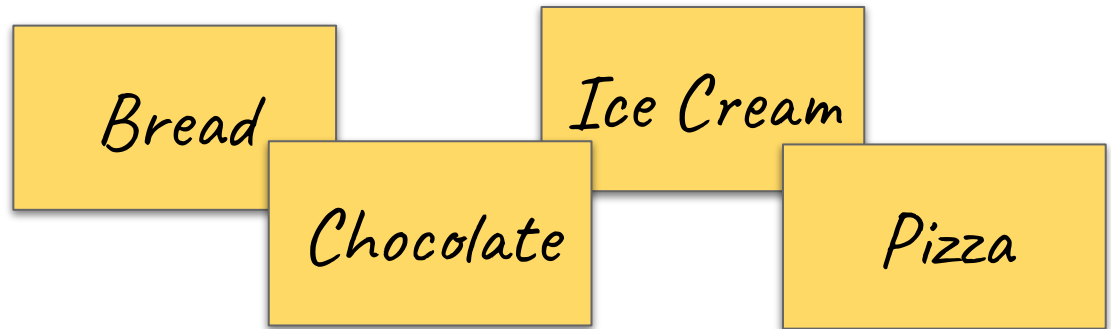
>>> y = x
>>> print(y)

>>> y = y + 1
>>> print(y)
```


Storing lists of things

When we go shopping, we write down what we want to buy!

But we don't store it on lots of little pieces of paper!



We put it in one big shopping list!

- Bread
- Chocolate
- Ice Cream
- Pizza

Lists

It would be annoying to store it separately when we code too!

```
>>> shopping_item1 = "Bread"  
>>> shopping_item2 = "Chocolate"  
>>> shopping_item3 = "Ice Cream"  
>>> shopping_item4 = "Pizza"
```

So much repetition!!

Instead we use a python list!

```
>>> shopping_list = ["Bread", "Chocolate", "Ice Cream",  
"Pizza"]
```

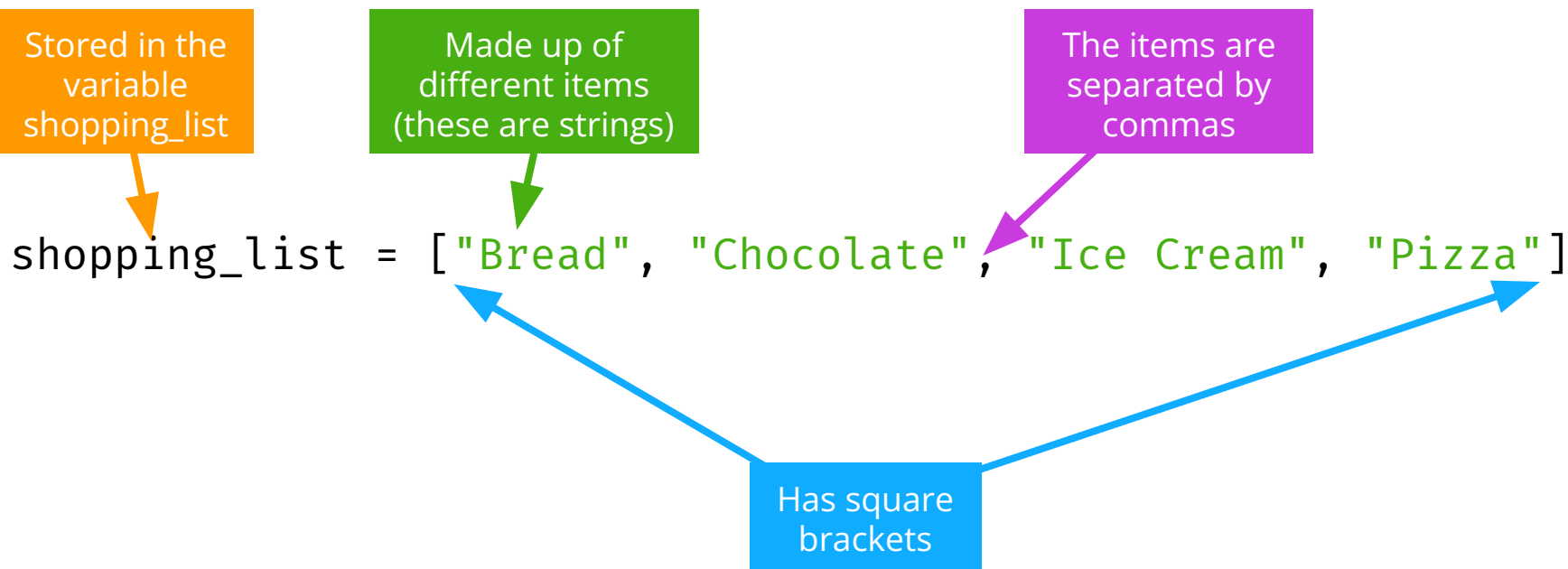
You can put (almost) anything into a list

- You can have a list of **integers**

```
>>> primes = [1, 2, 3, 5, 11]
```
- You can have **lists** with mixed **integers** and **strings**

```
>>> mixture = [1, 'two', 3, 4, 'five']
```
- But this is almost never a good idea! You should be able to treat every element of the **list** the same way.

List anatomy

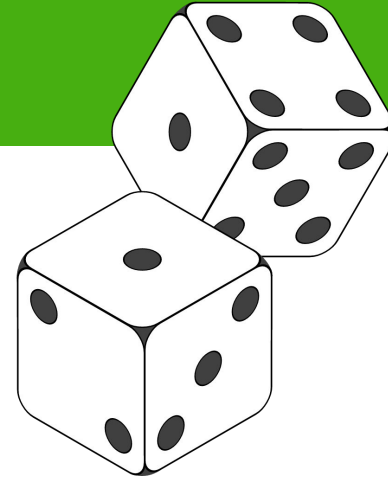


That's so random!

There's lots of things in life that are up to chance or random!



Python lets us **import** common bits of code people use! We're going to use the **random** module!



We want the computer to be random sometimes!



Using the random module

Let's choose something randomly from a list!

This is like drawing something out of a hat in a raffle!

Try this!



1. Import the random module!

```
>>> import random
```

2. Copy the shopping list into IDLE

```
>>> shopping_list = ["Bread", "Chocolate", "Ice Cream",  
                    "Pizza"]
```

3. Choose randomly! Try it a few times!

```
>>> random.choice(shopping_list)
```

Using the random module

You can also assign your random choice to a variable

```
>>> import random
>>> shopping_list = ["Bread", "Chocolate", "Ice Cream",
                    "Pizza"]
>>> random_food = random.choice(shopping_list)
>>> print(random_food)
```



Project Time!

Raaaaaaaaaandom! Can you handle that?

Let's try use it in our project!
Try to do Part 2

The tutors will be around to help!

If Statements

Conditions!

Conditions let us make 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

Try typing these into IDLE!

`5 < 10`

`3 + 2 == 5`

`5 != 5`

`"Dog" == "dog"`

`"D" in "Dog"`

`"Q" not in "Cat"`

Booleans (True and False)

Python has some special comparisons for checking if something is **in** something else. **Try these!**

```
>>> "A" in "AEIOU"  
>>> "Z" in "AEIOU"  
>>> "a" in "AEIOU"
```

```
>>> animals = ["cat", "dog", "goat"]  
>>> "banana" in animals  
>>> "cat" in animals
```

Booleans (True and False)

Python has some special comparisons for checking if something is **in** something else. **Try these!**

True

"A" in "AEIOU"

False

"Z" in "AEIOU"

False

"a" in "AEIOU"

False

"banana" in animals

True

"cat" in animals

```
>>> animals = ["cat", "dog", "goat"]
```

Conditions

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

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

What do you think happens?

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

Conditions

How about a different number???

```
fave_num = 9000  
if fave_num < 10:  
    print("that's a small number")
```



Conditions

Find out if it's **True**!

```
fave_num = 9000  
if False:  
    print("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 9000
- And it's not **True** that 9000 is less than 10
- So it is **False**!

Conditions

How about a different number???

```
fave_num = 9000  
if fave_num < 10:  
    print("that's a small number")
```



What do you think happens?

```
>>>
```

Conditions

How about a different number???

```
fave_num = 9000  
if fave_num < 10:  
    print("that's a small number")
```



What do you think happens?

```
>>>
```



Nothing!

If statements

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

This line ...

... controls this line

If statements

Actually

```
fave_num = 5
if fave_num < 10:
    print("that's a small number")
    print("and I like that")
    print("A LOT!!")
```

This line ...

... controls anything below it
that is indented like this!

If statements

```
fave_num = 5
if fave_num < 10:
    print("that's a small number")
    print("and I like that")
    print("A LOT!!")
```

What do you think happens?

```
>>>
```

If statements

What do you think happens?

```
fave_num = 5
if fave_num < 10:
    print("that's a small number")
    print("and I like that")
    print("A LOT!!")
```

```
>>> that's a small number
>>> and I like that
>>> A LOT!!
```

If statements

```
word = "GPN"  
if word == "GPN":  
    print("GPN is awesome!")
```

What happens?

If statements

```
word = "GPN"  
if word == "GPN":  
    print("GPN is awesome!")
```

What happens?

```
>>> GPN is awesome!
```

If statements

```
word = "GPN"  
if word == "GPN":  
    print("GPN is awesome!")
```

What happens?

```
>>> GPN is awesome
```

But what if we
want something
different to
happen if the
word isn't "GPN"

Else statements

else
statements
means something
still happens if
the **if** statement
was **False**

```
word = "Chocolate"
if word == "GPN":
    print("GPN is awesome!")
else:
    print("The word isn't GPN :(")
```

What happens?

Else statements

else
statements
means something
still happens if
the **if** statement
was **False**

```
word = "Chocolate"  
if word == "GPN":  
    print("GPN is awesome!")  
else:  
    print("The word isn't GPN :(")
```

What happens?

```
>>> The word isn't GPN :(
```

Elif statements

elif

Means we can
give specific
instructions for
other words

```
word = "Chocolate"
if word == "GPN":
    print("GPN is awesome!")
elif word == "Chocolate":
    print("YUMMM Chocolate!")
else:
    print("The word isn't GPN :(")
```

What happens?

Elif statements

elif

Means we can
give specific
instructions for
other words

```
word = "Chocolate"
if word == "GPN":
    print("GPN is awesome!")
elif word == "Chocolate":
    print("YUMMM Chocolate!")
else:
    print("The word isn't GPN :(")
```

What happens?

```
>>> YUMMM Chocolate!
```

Practice Time!

1. Create a new file, call it weather.py
2. Copy this code into your file

```
weather = input("What is the weather? ")  
if weather == "raining":
```

3. Add a third line to make it print a special message, but only if the user says "raining"
4. Run your code! Try typing in **raining**, try typing in **sunny**
5. BONUS! Add an else statement, to print a non-rainy message!

Practice Time!

1. Create a new file, call it weather.py
2. Copy this code into your file

```
weather = input("What is the weather? ")  
if weather == "raining":  
    print("Take an umbrella!")
```

3. Add a third line to make it print a special message, but only if the user says "raining"
4. Run your code! Try typing in **raining**, try typing in **sunny**
5. BONUS! Add an else statement, to print a non-rainy message!

Project Time!

You now know all about **if** and **else**!

See if you can do Part 3

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:
    print("i is " + str(i))
    i = i + 1
```

Introducing ... while loops!

What do you think this does?

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

```
i is 0
i is 1
i is 2
>>>
```

Introducing ... while loops!

Stepping through a while loop...

Introducing ... while loops!

One step at a time!

```
◆ i = 0  
  while i < 3:  
    print("i is " + str(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:
    print("i is " + str(i))
    i = i + 1
```

MY VARIABLES

i = 0

Introducing ... while loops!

One step at a time!

Print!

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

```
i is 0
```

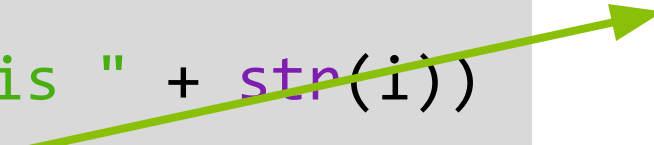
MY VARIABLES

```
i = 0
```

Introducing ... while loops!

One step at a time!

```
i = 0
while i < 3:
    print("i is " + str(i))
    ◆ i = i + 1
```



MY VARIABLES

~~i = 0~~
i = 1

UPDATE
TIME!

```
i is 0
```

Introducing ... while loops!

One step at a time!

Take it
from the
top!

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

```
i is 0
```

MY VARIABLES

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


Introducing ... while loops!

One step at a time!

i is less
than 3!

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

MY VARIABLES

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

```
i is 0
```

Introducing ... while loops!

One step at a time!

Print!

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

```
i is 0
i is 1
```


MY VARIABLES

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

Introducing ... while loops!

One step at a time!

```
i = 0
while i < 3:
    print("i is " + str(i))
    ◆ i = i + 1
```



MY VARIABLES

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

UPDATE
TIME!

```
i is 0
i is 1
```

Introducing ... while loops!

One step at a time!

Take it
from the
top!

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

```
i is 0
i is 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:
    print("i is " + str(i))
    i = i + 1
```

MY VARIABLES

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

```
i is 0
```

```
i is 1
```

Introducing ... while loops!

One step at a time!

Print!

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

```
i is 0
i is 1
i is 2
```


MY VARIABLES

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

Introducing ... while loops!

One step at a time!

```
i = 0
while i < 3:
    print("i is " + str(i))
    ◆ i = i + 1
```



MY VARIABLES

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

```
i is 0
i is 1
i is 2
```

UPDATE
TIME!

Introducing ... while loops!

One step at a time!

Take it
from the
top!

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

```
i is 0
i is 1
i is 2
```

MY VARIABLES

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


Introducing ... while loops!

One step at a time!

3 IS NOT
less than
3!

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

We are
done
with this
loop!

```
i is 0
i is 1
i is 2
```

MY VARIABLES

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

Introducing ... while loops!

Initialise the loop variable

Loop condition

Code to repeat

```
i = 0
while i < 3:
    print("i is " + str(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:
    print("i is " + str(i))
```

What happens when.....

What happens if we forget to update the loop variable?

```
i = 0
while i < 3:
    print("i is " + str(i))
```

[illegible]

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:  
    print("Are we there yet?")
```

Give me a break!

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

```
number = 0
while number != 42 :
    number = input("Guess a number: ")

    if number == "I give up":
        print("The number was 42")
        break

    number = int(number)
```

Continuing on

How about if I wanna skip the rest of the loop body and loop again? We use **continue** for that!

```
number = 0
while number != 42 :
    number = input("Guess a number: ")

    if not number.isnumeric():
        print("That's not a number!")
        print("Try again")
        continue

    number = int(number)
```

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** second?

10,500

Project Time!

while we're here:

Try to do Part 4!

The tutors will be around to help!

Micro:Bit Buttons

Buttons!

Your MicroBit has 2 buttons: Button A and Button B

You can use this code to check whether or not a button is pressed:

```
button_a.is_pressed()
```

```
button_b.is_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.is_pressed():  
    display.show(Image.HAPPY)  
  
if button_b.is_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

The Micro:Bit shows a Sad face

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

Project Time!

Does that press your buttons?

Try to do Part 5 and 6!

The tutors will be around to help!

Tell us what you think!

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