



School of Computing, Engineering and Intelligent Systems

Introduction to Programming concepts

What is programming?

Programming on a computer is a little bit like writing instructions. When you write **instructions**, you may want to tell the **reader** (in this case the computer) to execute certain tasks in a specific order. This is the first step of understanding any programming language. All tasks are executed sequentially.

For example:

1. Open this booklet
2. Go to the website: <https://makecode.microbit.org/#editor>
3. Start to program
4. Finish

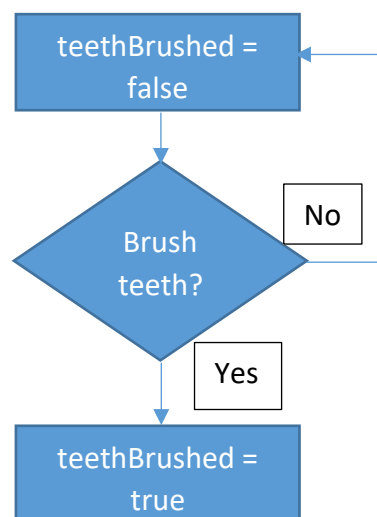
We don't start at 4 and work backward.

We perform tasks everyday without thinking about the order of sequence, or the routine but imagine you're getting ready in the morning, a part of that is brushing your teeth. What we can do is break the routine of getting ready into smaller routines and we can even go further and break brushing your teeth down into smaller routines or **sub routines**. That is parts of a sequence that combine to make a sequence.

We'll go through some programming basics as we go through this analogy.

We'll start with a **flow chart** and a **variable**. A **variable** is a specific space to hold specific information, for example your name ("**String**") or your age (**integer**). In this case we want to use a variable to store the information on whether or not you've brushed your teeth, we'll use a variable called a **Boolean**. A **Boolean** is **True** or **False**, there is no in between.

Usually, we set the variable to a value, in this case brushing teeth = false. Now we want to store the value in a **variable**, and we use a very specific spelling convention called **Camel Case**, this is writing without punctuation and usually starting lowercase using capitals to break the word up. Variables should be named appropriately, so we will call our variable brushing teeth as teethBrushed = false.



So, now we've covered **variables** we are going to cover the last two main concepts, **decisions** and **iterations/loops**. In the above flow chart, we simply "ask" whether the user has brushed their teeth, and give two outputs yes or no. We do this all the time in programming, usually in the form of an if else statement. We will cover this in more detail later on.

But brushing your teeth can have a number of steps:

Here's an example:

- 1) Get Toothbrush.
- 2) Rinse Toothbrush.
- 3) Get Toothbrush.
- 4) Apply toothpaste.
- 5) Begin Brushing.
- 6) Rinse with water.
- 7) Repeat until 2 minutes are up.



Iterations refer to the repetition of specific code. For example, when you brush your teeth, you keep brushing for 2 minutes or until your teeth are clean then you would be done.

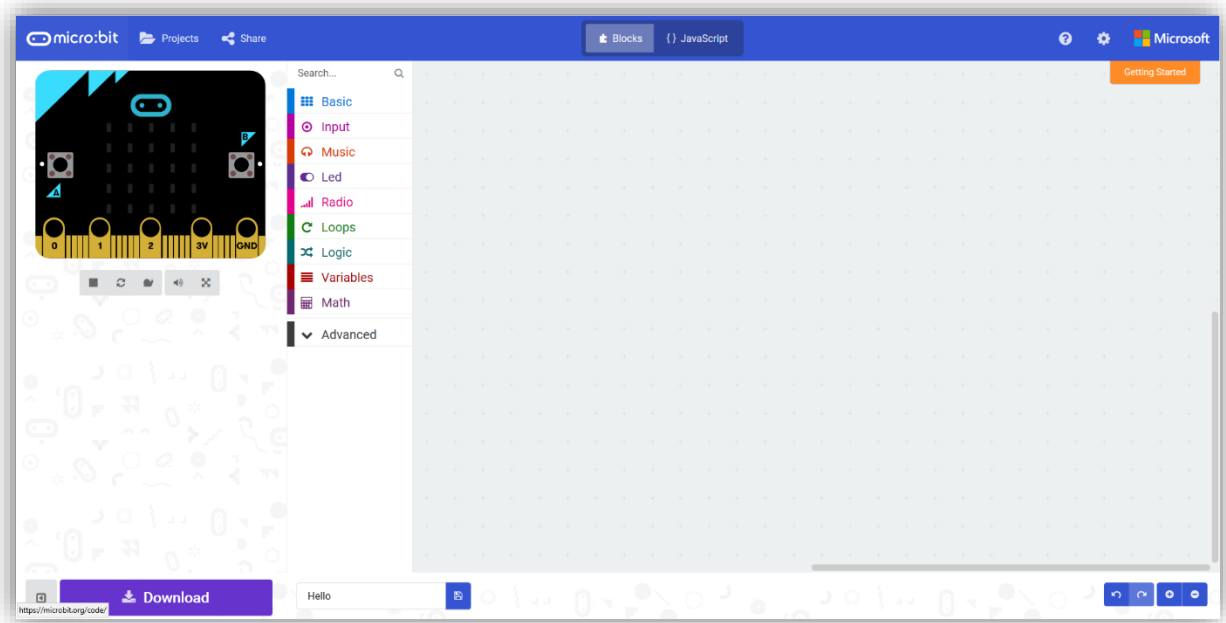
This is an iteration loop. In a way, the brushing of your teeth can be considered a loop. A loop will be continually repeated until a condition is met, in this case it would be until your teeth are clean or you've spent 2 minutes brushing.

When it comes to loops and iteration, a condition or an end point must be specified, otherwise the repetition will be infinite. So, the loop is the continual brushing of your teeth for 2 minutes. At the end of the loop teeth are now brushed. If you go to ask if teeth are brushed the variable will now be true: `teethBrushed = true`. Therefore, you don't repeat

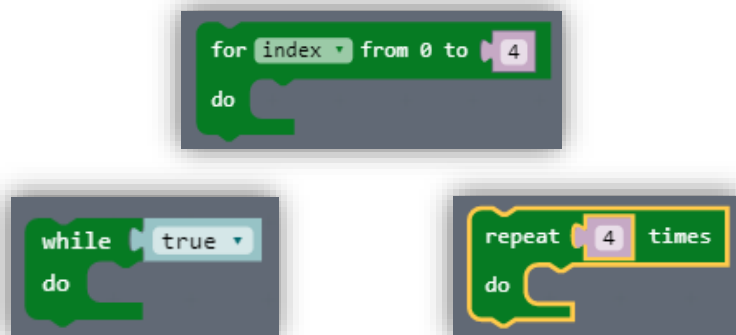
This is an algorithm. A sequence of steps/instructions that can be followed to complete a task.

In programming loops are important. It requires less code to be written, this is to reduce complexity as it isn't as time consuming for the programmer. The code can still be clearly read, and the overall performance is better as there is less code to run through when the program is to be executed. Loops can be a number of loops, a set amount of time or can run forever, however, there should be a clause to cancel the loop, otherwise it'll run infinitely i.e. an infinite loop. There are different loops that can be used in JavaScript – while, repeat and for.

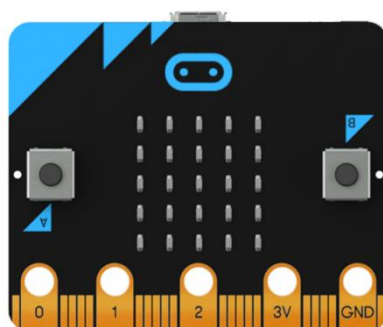
So, let's look at **Microbit**:

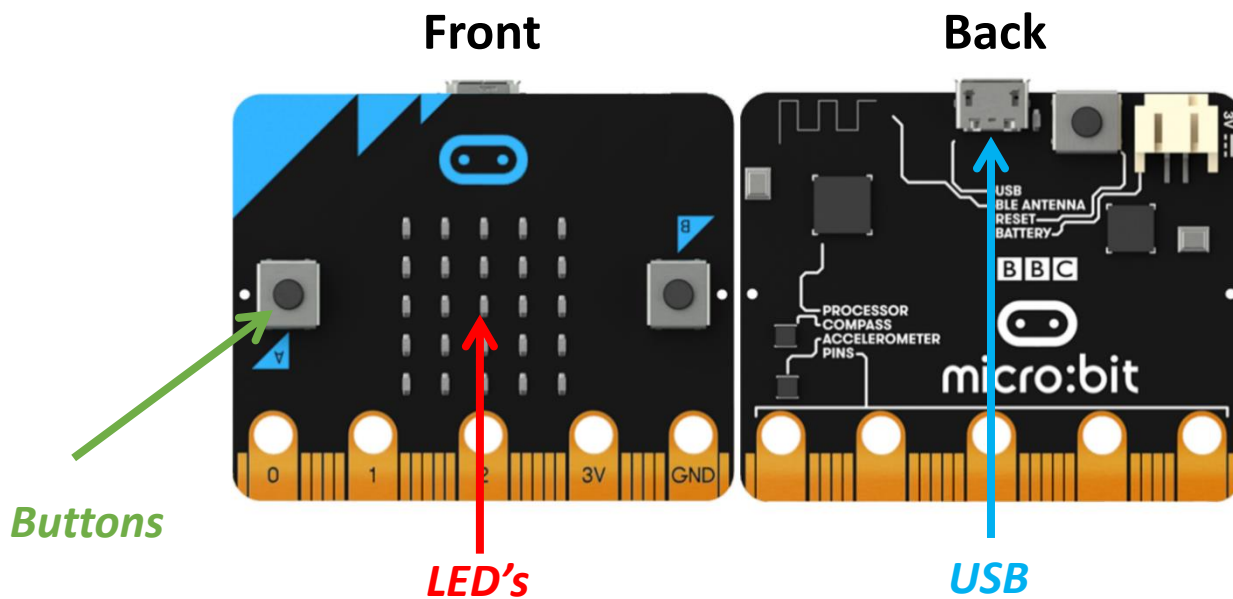


This is our **Microbit** editor. On the left we see an emulator, an emulator is a computer program set up to replicate hardware, in this case our **Microbit**. On the right of this we have code blocks, as we click on each we can give the Microbit different instructions:



And we have our **Microbit**:



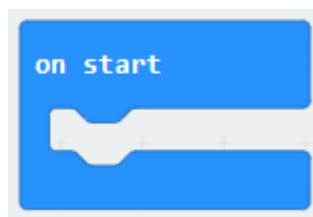


Block Editor for BBC Micro:Bit

An event in programming is an action, this action can happen due to the user whether it is clicking something or pressing a key or it can happen due to something else.

An event handler is an algorithm which will execute once a particular event occurs, this particular algorithm is written by the programmer and they decide what this algorithm will do when the event occurs.

An example of an event handler is the 'on start' block of code.

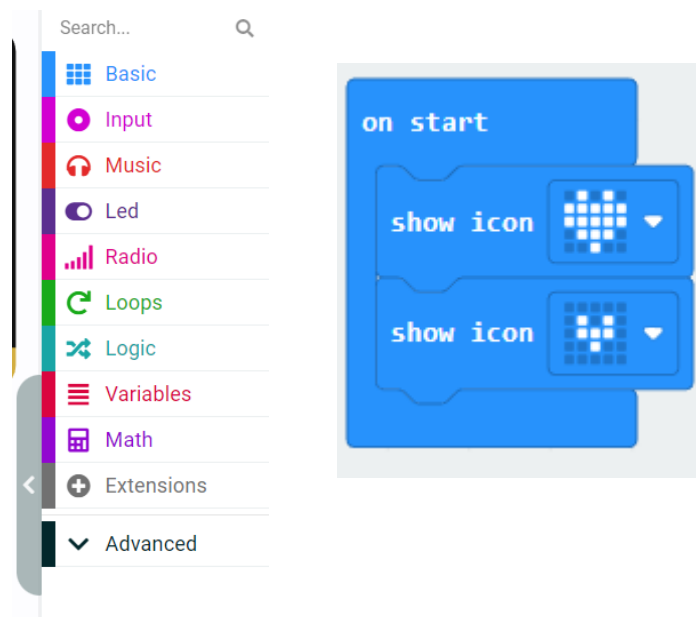


This simply tells the BBC Micro:Bit that when it starts, it must execute any code which follows this code block.

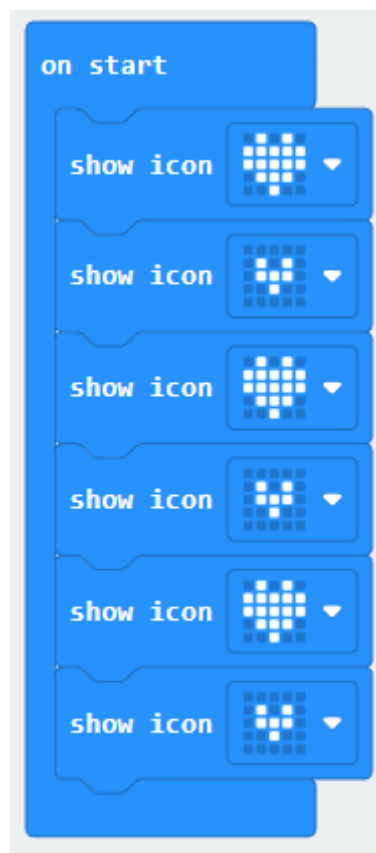
We also have our first loop, a forever loop :



Let's look at the difference between these. On the left hand side of the code editor we can see the emulator, this is used to test our code and a panel which allows us to access code blocks. Go to basic show icon, and choose a big heart and a small heart.

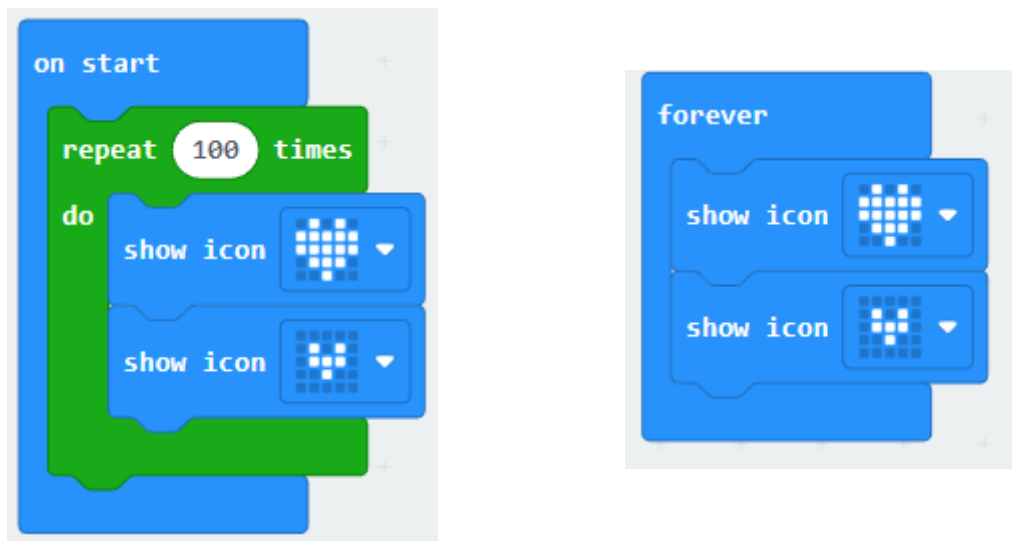


This code tells the Microbit to show these icons in sequence when the Microbit starts for the first time. If we want to do this 3 times we might do something like this:



But what if we want to do this 100 times?

It might get time consuming, we can simplify this with a loop. Click on the loops tab, and choose the repeats for **X** number of times, change the **X** to 100. If we wanted this to be the only thing our Microbit does we can use the forever loop.



We should only use one of these event handlers at a time.

So, let's get started. Combine everything we have done so far to do the following exercises.

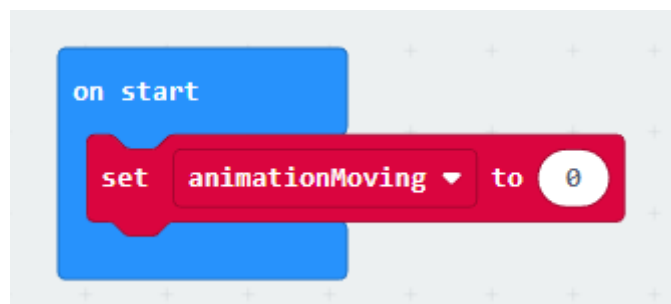
Simple Loop Task

Step 1

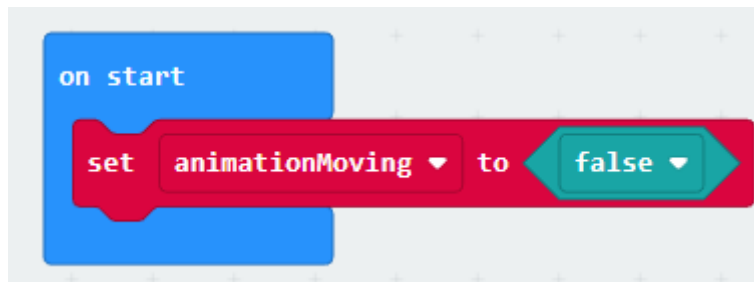
To start off with, we'll get the Micro:Bit to display an animation on its LED's by making use of one of the loops available in the Blocks editor.

This animation will be triggered by an event, for example a button press. This requires user input which can be used to carry out an action to provide an appropriate output.

So, start by going to the 'basic' section and drag in an 'on start' block of code. Then go to the variables section, you may have to 'Make a variable' call this variable "**animationMoving**".



Here we're setting a **variable**, go to the **Logic** section and under **Boolean** drag in a 'false' block and place this block on the "0" this will change your variable type. So, we are setting a variable to false i.e. the animation is not moving. It should look like this:



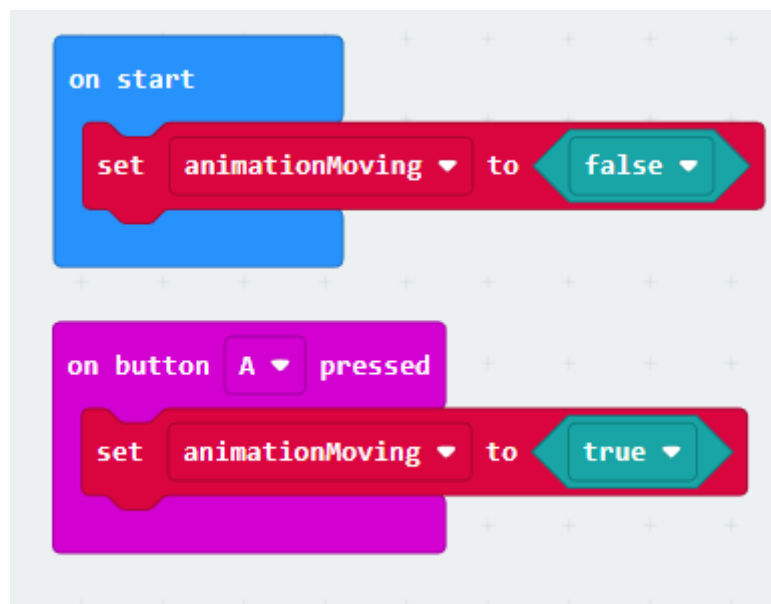
Step 2

Next, go to the 'input' section on the left side of the Blocks editor, these code blocks are concerned with input from either a user or from the features of the Micro:Bit.

Select the 'on button a pressed' block and drag this in. This is an event handler, any code which follows it will be executed when A is pressed.

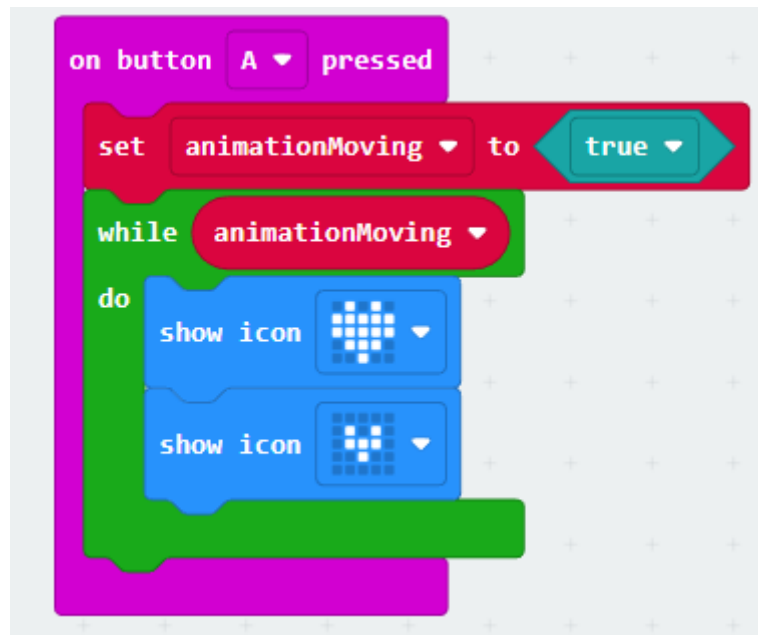


Now we are telling the **Microbit** that when the A button is pressed we want to change our variable from false to true. So, go to **Variables**, select set animation moving and replace the 0 to true, or copy and paste the first block and place it in the Button A pressed block:



Now that the variable can be changed, we can keep adding to our blocks in order to make our **Microbit** code more complicated.

Next, a 'while' loop is needed so that the animation will only show when the variable is true.



This means that while animationMoving is true, any code blocks following will be executed as many times as needed.

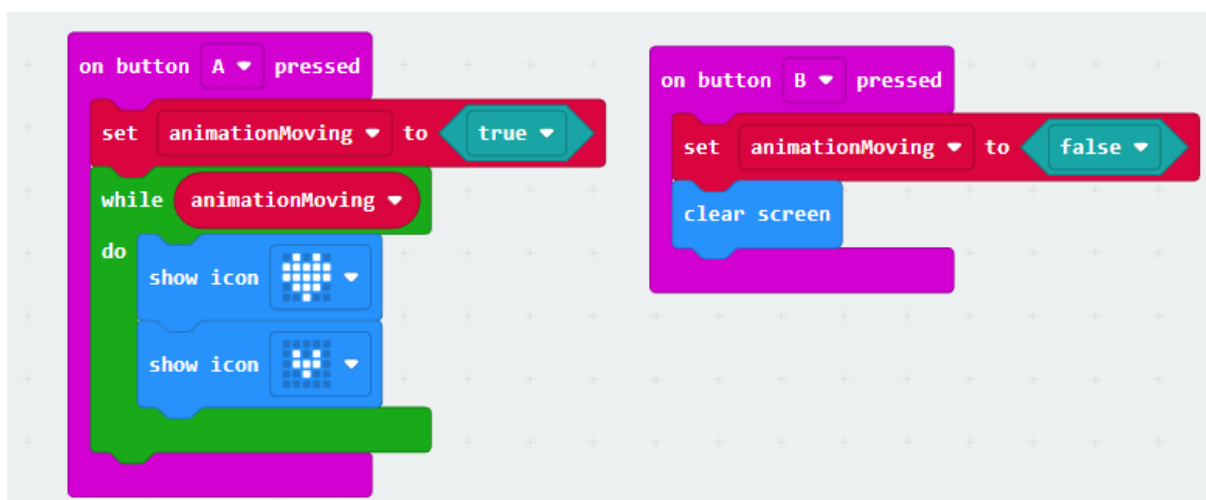
Step 3

The **Microbit** now needs to be told what to display when the event is triggered. Now that the animation happens, you need to tell it when to switch off.

So, when button B is pressed, it must turn off.

Using the 'on button a pressed' block in the 'input section', drag this in and then click the dropdown arrow to change it to 'on button b pressed'.

Now you need to go to the variable section, use the 'set item to' block again, and select your 'animationMoving' variable. Then set this to False. You can then use the clear screen block in basics to reset the **Microbit**:



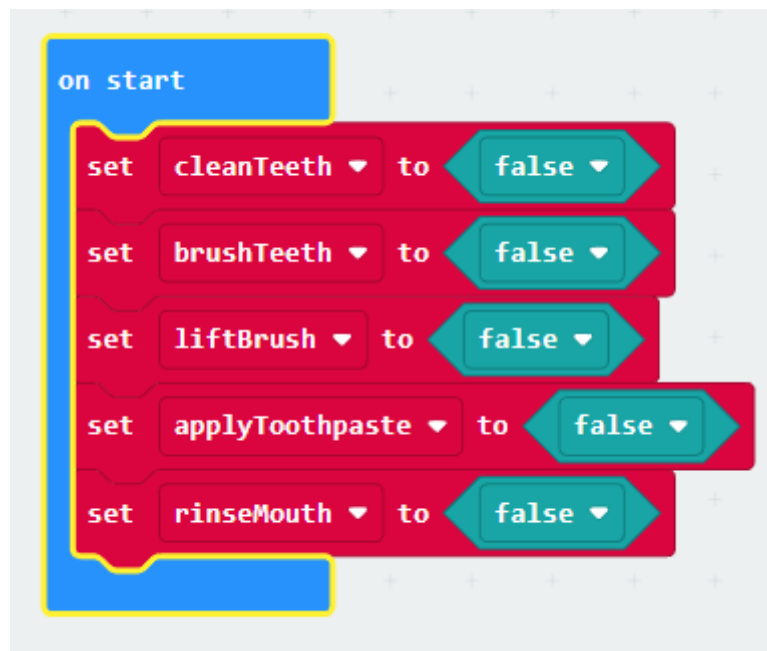
Activity 1

Let's apply our original example of brushing teeth, we'll get the **Microbit** to display an unhappy face until we brush our teeth, then we'll change to a happy face when we brush our teeth to do this, we will be using event handlers & loops.

Step 1

We start off by initializing a few variables. We do this because we want to use certain event changes i.e. variable changes, to trigger or cause certain events occur. At this point the unhappy face shows that the Microbit is ready but is idle. So, use the 'On start' block of code again, then draw a face using the 'show LED's' block or use the 'show icon' block instead.

Now, a number of variables need to be created similar to last time which will indicate when certain animations should display. The Variables will be cleanTeeth, rinseMouth, brushTeeth, liftBrush and applyToothpaste. Change the 0 in each.



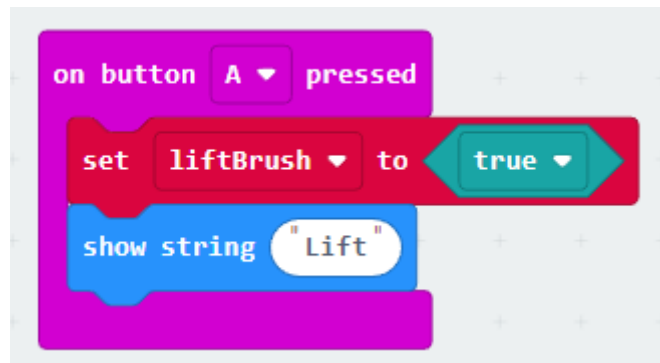
Step 2

Now event handlers need to be created. Button presses and movements will be used to create the animation of brushing the teeth.

Drag in 'on button pressed'. For this button, selecting button A will trigger lifting the toothbrush animation. So, we'll change the value of the variable 'liftBrush' to true, this means that the toothbrush has been picked up, and to indicate the user has lifted the brush user a message an animation is shown.

A while loop can be used, so that while the brush has been lifted, the toothbrush icon will be shown with a message along with it.

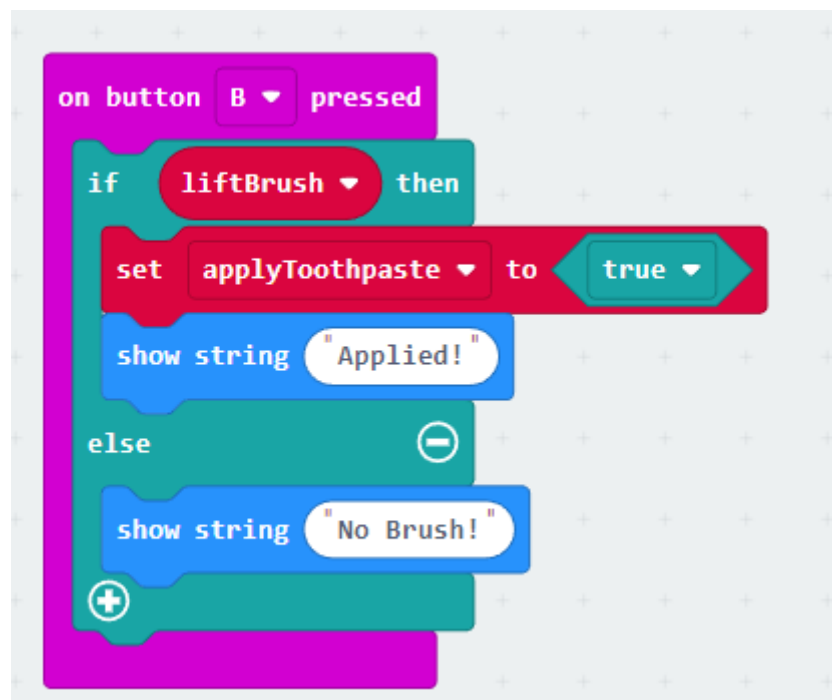
So, in programming terms: while the brush is lifted, Show a String message.



Step 3

The next thing to do when brushing your teeth is to apply toothpaste, then begin brushing.

1. To do this, drag in another 'on button pressed' event handler. This time set it to button B. This will be the apply toothpaste button.
2. The code within this must check if the toothbrush is in the user's hand.
3. So an 'if' statement is needed. If the user does have the toothbrush, then it must show an animation and or message when the user presses B to apply the toothpaste.



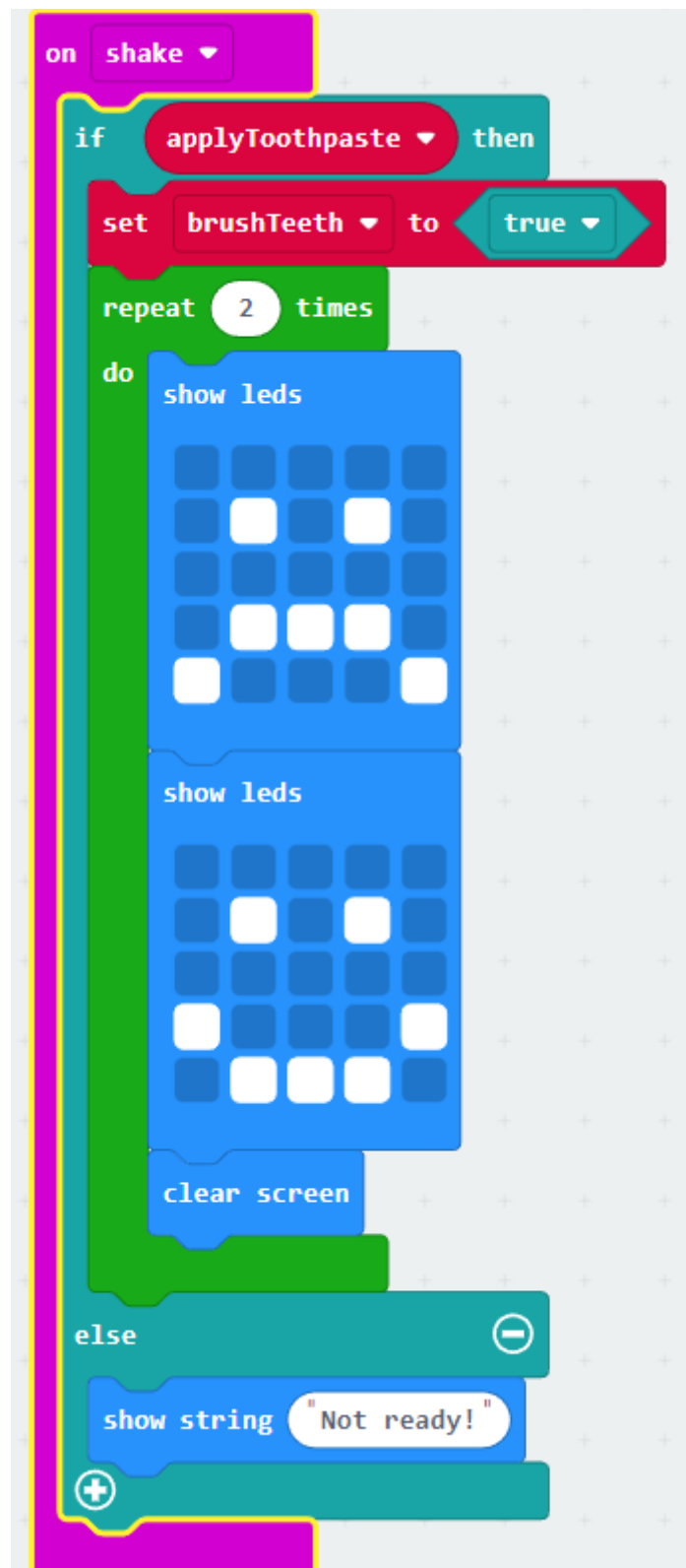
Step 4

Next, we need to brush the teeth.

This time use an 'on shake' is needed.

Similar, to the first animation, the animation must only be displayed when the variable is true. So go to the variables section and drag in 'set item to' under the 'on shake' event handler. Then change the variable to 'brushTeeth' and set this to true. When this is true then the

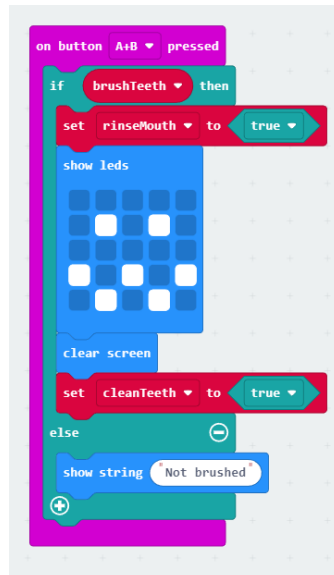
animation must be carried out multiple times. So, create an animation that mimics a face changing to imitate the teeth being brushed.



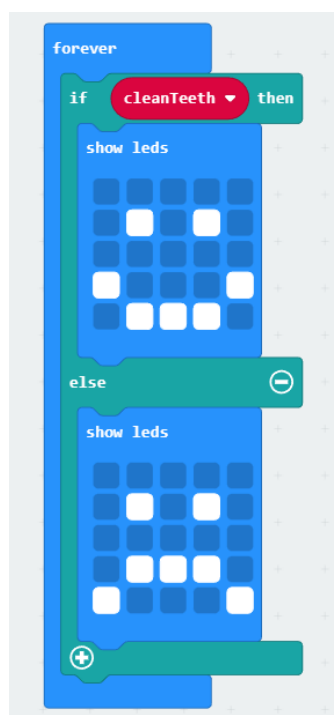
Step 5

Next, we want to rinse and complete.

This time an 'on button a pressed' is needed and must be changed to 'on button A+B pressed'. Each time we start a new task we check the previous task has been completed and usually we indicate to the user by way of feedback:



When all these steps are complete we set the clean teeth to true. Now, we want to know if our teeth are cleaned, when we started up, we checked the variable and displayed a sad face. We now want to show a happy face, so we check the status and show our animation accordingly, i.e. if teeth are clean smiley face and if they aren't sad face. Use the forever loop, to always check if the variable is true or false, you can then change the face:



Activity 2

Introduction

This activity will focus on the use of Condition statements. We have used these in the last practical but we focused on performing task A to change a variable which would allow us to perform task B and so on. Conditions may occur whilst a program is being executed and will perform actions based off of these conditions being met or not. For example imagine an automated heating system, when the temperature gets too cold the heater turns on, when it gets too hot the heater turns off. We would measure the temperature at all times in this instance (forever loop), when it gets too cold we'd execute code to turn on the heater.

Objectives

- Understand Conditional Statements and why we use them in programs.
- Learn how to use Conditional Statements by making use of the logic blocks.
- Determine what actions will be carried out based on conditions being met.

Introduction

With Programs, you have to tell the computer exactly what to do, because you can't always tell the computer exactly when to execute something, we use conditional statements to automate the process for us i.e., only to do them if certain conditions have been met.

You are probably familiar with conditions already. For example, if an exam is 1 hour long and you reach the 1 hour mark then you can leave the exam hall. So, an easy way to put it is IF (condition is met) THEN (Execute action). This is only one example and there are many more.

If the condition is not met we can tell the computer to do something else. Referring to our previous example, if the hour mark is reached then you can leave else continue working. Putting this into programming terms – IF (condition is met) THEN (Execute action) ELSE (Execute separate action).

We refer this to an If Else statement, if the temperature is below 16 turn on heater else heater is off. We can add to this, If the temperature is below 16 turn on heater, else if the temperature is above 28 turn on fan else turn off fan and heater.

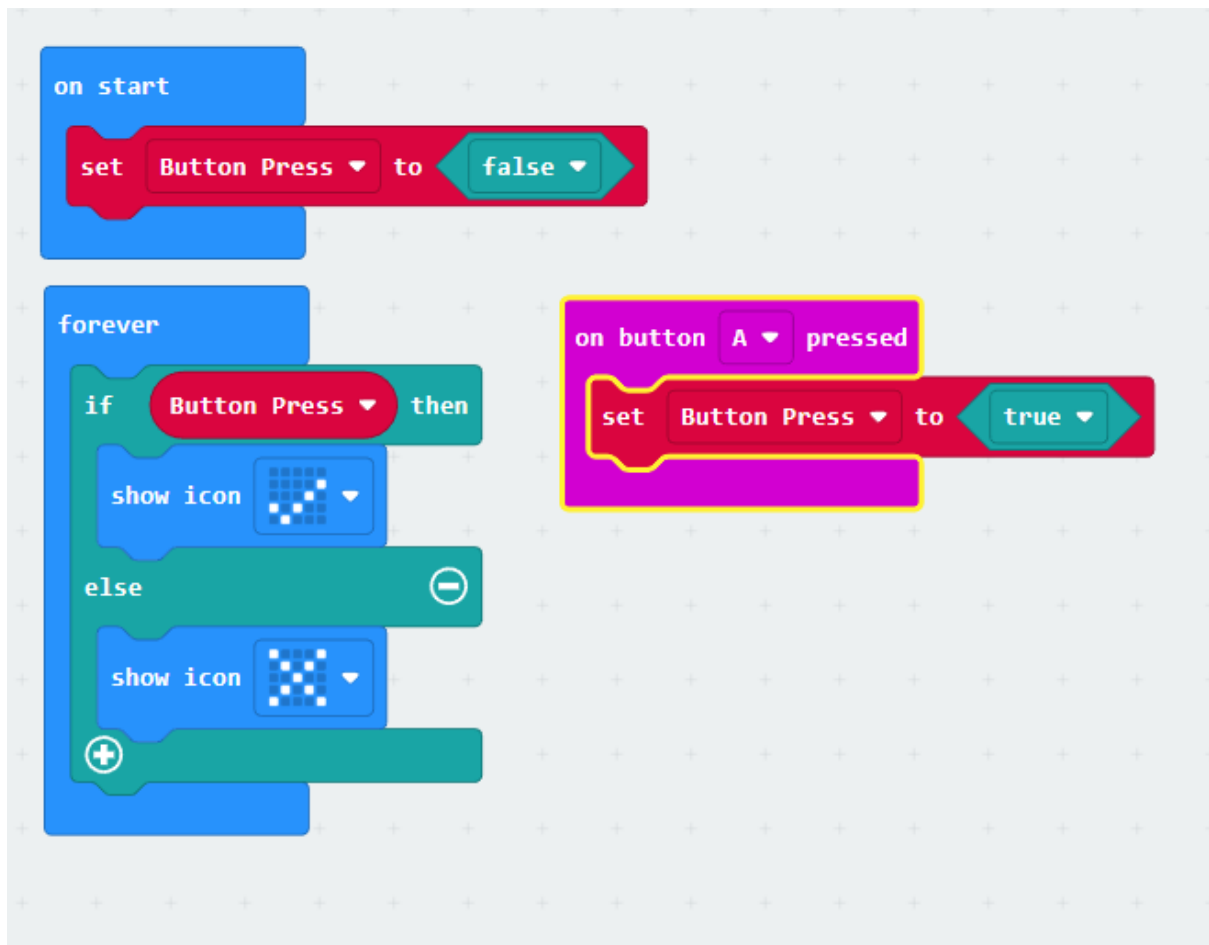
The ELSE statement is typically a default, it may refer to basically do nothing, or may tell the user something doesn't work and a condition has to be met. We'll see this later on.

Conditions

To start we usually have a variable that we are looking at, when we add a condition should we create it to observe the variable and carry out a specific action. A simple example would be to make the **Microbit** display an image or message when a button is pressed.

This requires User input, drag in a forever block, followed by an 'if then' statement, an icon and a 'button A is pressed input block'.

The **Microbit** should display an icon anytime that the button A is pressed. So, forever – if button A is pressed then show icon. Now we need to initialise the variable as false when the **Microbit** is started for the first time.



Next we'll introduce a counter. We'll also be looking at using Java script. We use coding blocks, but we can also use a programming language, in this case JavaScript, to write our programs. We click on the button:



And it transforms our code:

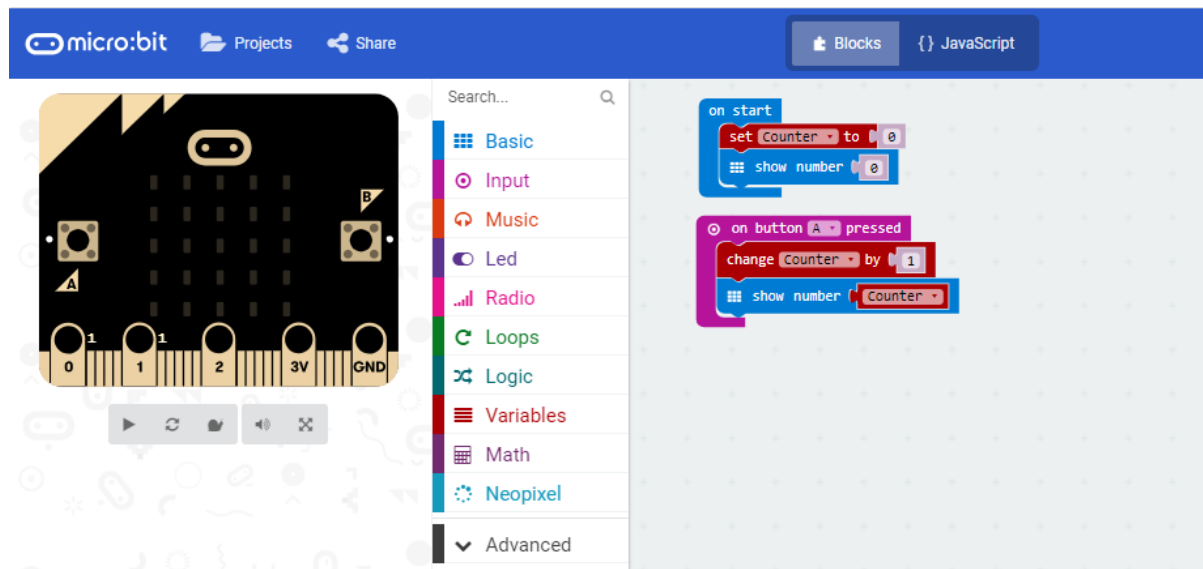
```
1  input.onButtonPressed(Button.A, function () {
2      Button_Press = true
3  })
4  let Button_Press = false
5  Button_Press = false
6  basic.forever(function () {
7      if (Button_Press) {
8          basic.showIcon(IconNames.Yes)
9      } else {
10         basic.showIcon(IconNames.No)
11     }
12 })
13
```

It looks more complicated but, the structure is still there. If you have experience of a coding language, try writing your project in JavaScript. If you feel more comfortable with using the code blocks editor, that's also fine, being aware of this will help you understand programming language or syntax.

Activity 3 - Coin Toss

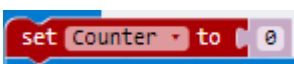
Step 1

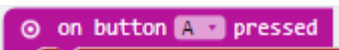
Using the Blocks, create a variable called counter. Go to the Variable section and drag in set item to 0 and change the variable name to counter. Once this is done, go to the input section and drag in 'on button a pressed'. This is an event handler, which will tell the computer to do something when the event occurs by using the code that follows it.

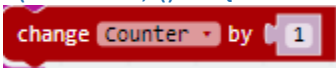



JavaScript:

```
let Counter = 0
input.onButtonPressed(Button.A, () => {
  Counter += 1
  basic.showNumber(Counter)
})
Counter = 0
basic.showNumber(0)
```

 - initialising counter





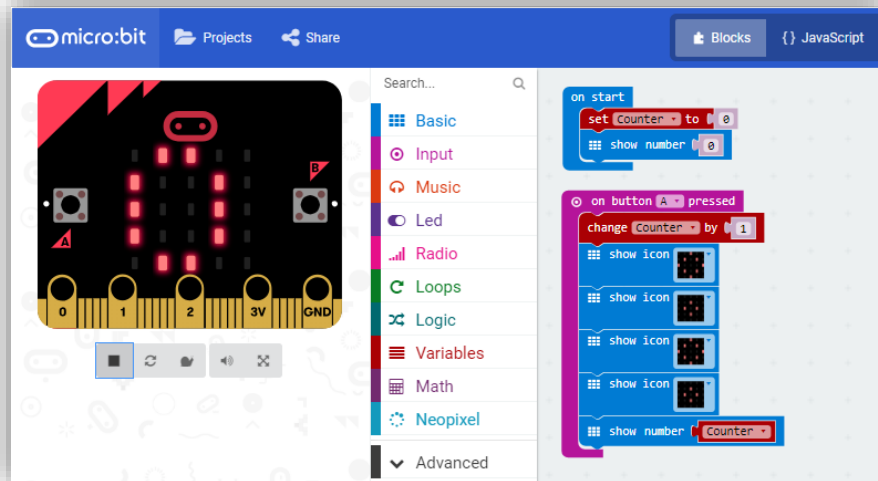


The **Microbit** should count each time a button has pressed and display it at the end of each action being performed. So, first of all, you are initialising the counter so that it'll keep track of button presses.

Step 2

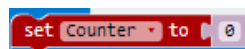
Now you're going to create a coin toss. Using the counter to count the number of tosses, and use an if statement to generate whether the coin lands on heads, or whether it lands on tails.

Firstly, it should display an animation indicating the coin has been flipped.

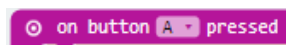


JavaScript:

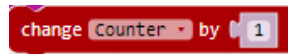
let Counter = 0



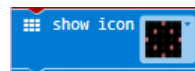
input.onButtonPressed(Button.A, () => {



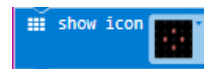
Counter += 1



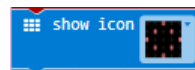
basic.showIcon(IconNames.Diamond)



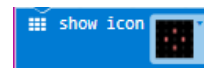
basic.showIcon(IconNames.SmallDiamond)



basic.showIcon(IconNames.Diamond)



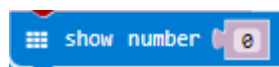
basic.showIcon(IconNames.SmallDiamond)



})

Counter = 0 - setting counter to 0

basic.showNumber(Counter)

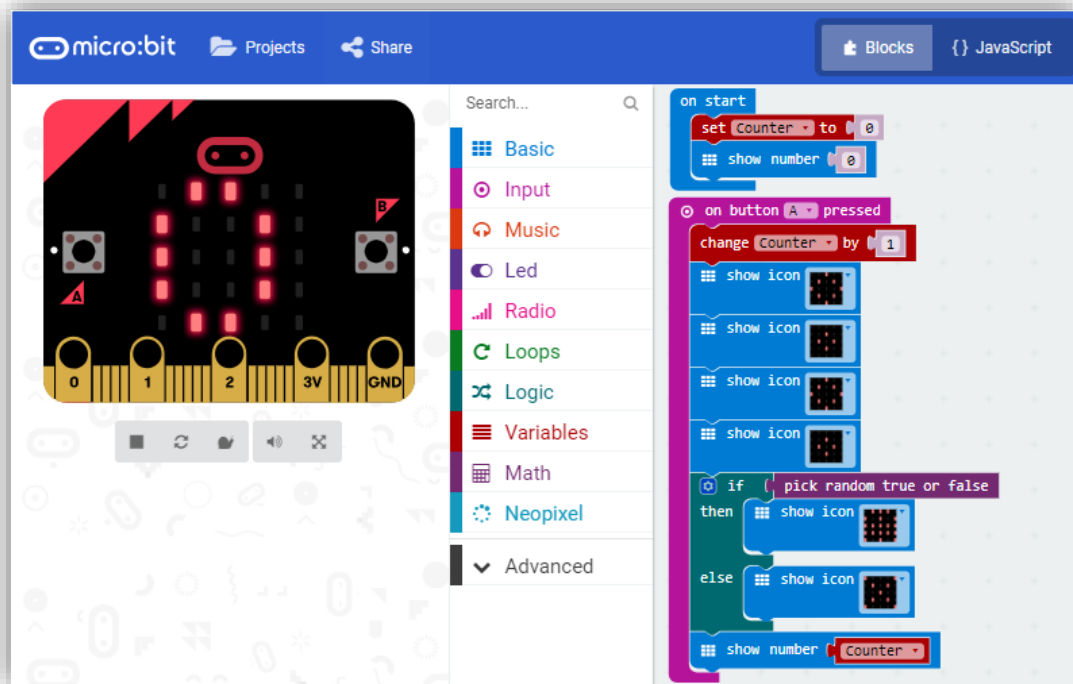


Step 3

For the last part we need to generate the 2 possible outcomes, heads or tails. Now we introduce the conditional statement.

Drag in the 'if then else' statement below the icons being shown, then we'll use the pick random true or false and attach that to the 'if' statement. If it equals true then one output will be produced, click the blue settings icon and add in an else statement so that if the outcome is false then the other outcome will be produced.

True and false are Boolean values.



JavaScript:

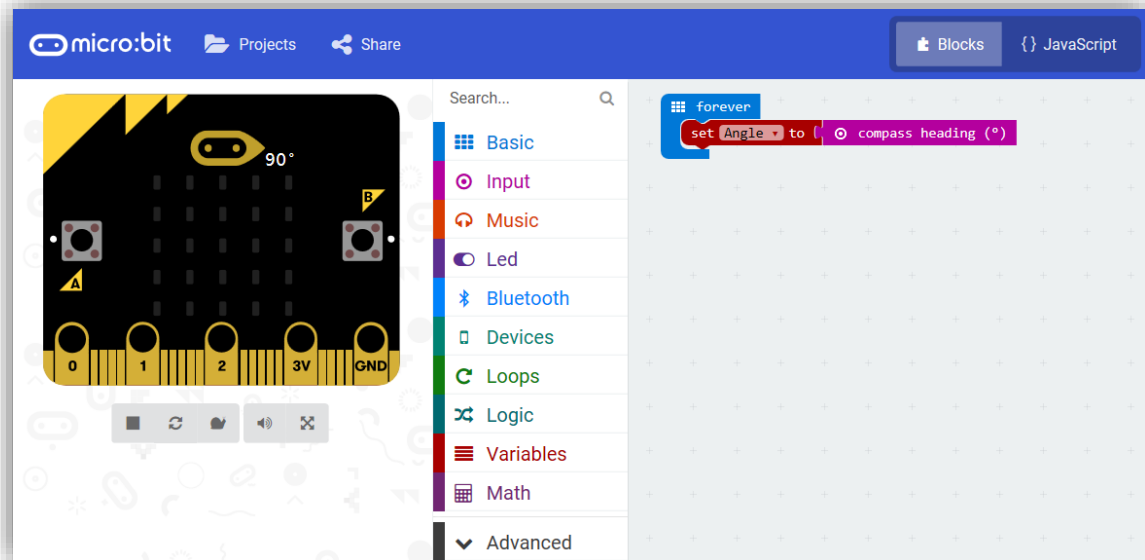
let Counter = 0

```
input.onButtonPressed(Button.A, () => {  
  Counter += 1  
  
  basic.showIcon(IconNames.Diamond)  
  basic.showIcon(IconNames.SmallDiamond)  
  basic.showIcon(IconNames.Diamond)  
  basic.showIcon(IconNames.SmallDiamond)  
  
  if (Math.randomBoolean()) {  
    basic.showIcon(IconNames.Ghost)  
  } else {  
    basic.showIcon(IconNames.No)  
  }  
  
  basic.showNumber(Counter)  
})  
Counter = 0  
basic.showNumber(0)
```

Activity 4 - Compass

Step 1

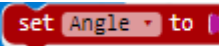
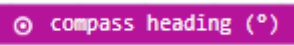
Now, you're going to use some of the built in hardware on the **Microbit** to create a compass. First of all, drag in forever, following by set item to. Attach them and rename this variable to angle. Then go to the input category and drag in compass heading and attach it to your variable. This will mean the variable is equal to whatever the compass reading is.



JavaScript:

`let Angle = 0`  - first set to 0, just to initialise the variable.

`basic.forever(() => {`  - tells Micro:Bit to always carry out the code that follows

`Angle = input.compassHeading()`   - Now assigns compass reading to the variable.

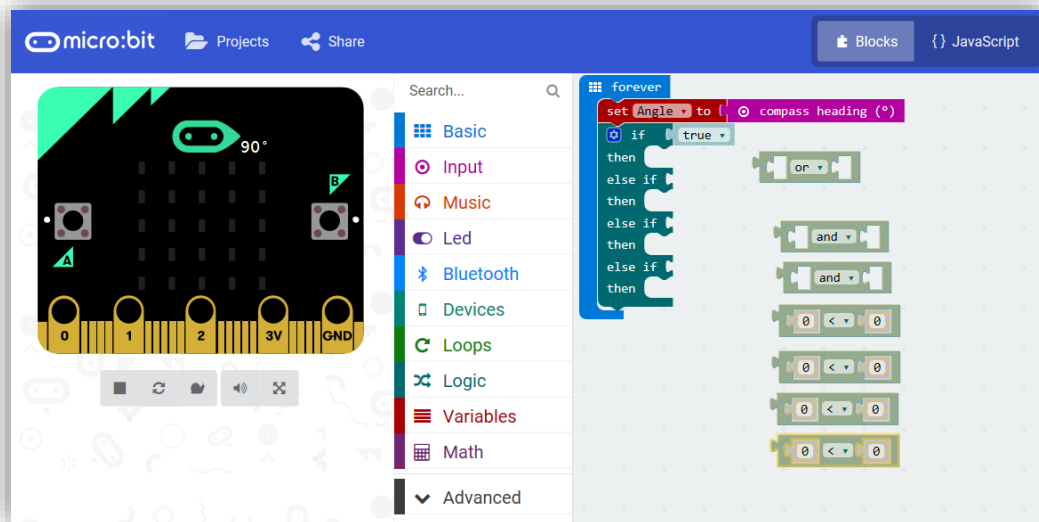
`})`

Step 2

The **Microbit** needs to display the direction that it is facing. So, you need to use the conditional statements with maths to get it to show these directions if the conditions are being met.

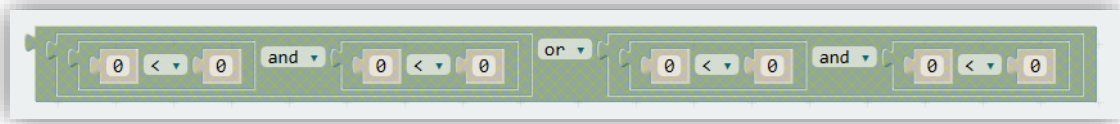
First go to logic and drag in an 'if then' statement.

Go to the logic category, drag in an 'or' block as well as 2 'and' blocks and 4 '>' blocks.



Step 3

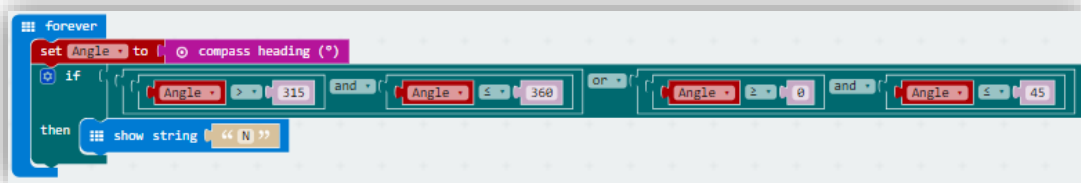
Join these blocks together by attaching the 'and' blocks to the inside of the 'or' block, then insert the '>' blocks inside the 'and' blocks. This allows us to set a condition now based on what we put inside this statement.



Step 4

We need to calculate at what degrees a certain direction will be shown by the LED's. So we use the angles variable to determine this.

You need to set it to show North between certain boundaries, if the compass is not within the boundaries, it will show another direction if it is between the other directions boundaries.

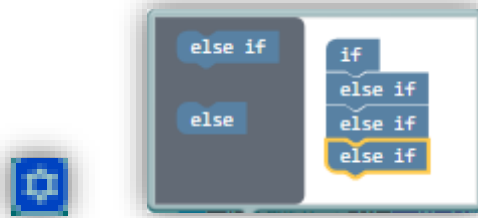


With these boundaries, the LED's will display N if the compass is at an angle between 315 and 45.

Step 5

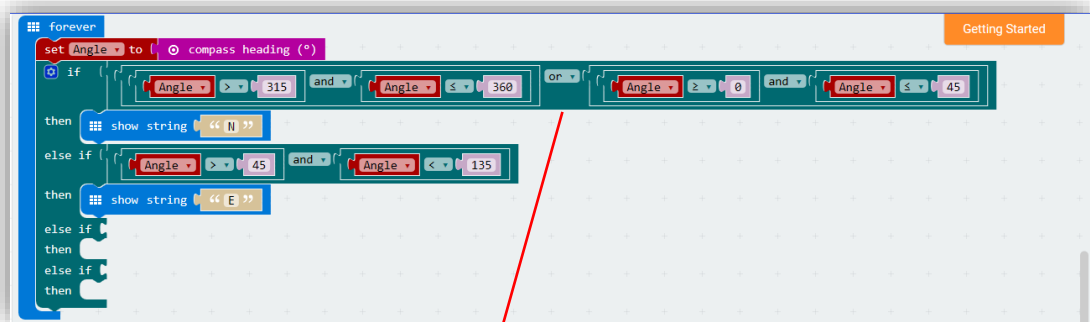
Now it needs to display other directions.

First, to add other conditions to the 'if' statement, click the blue settings icon at the top left of the block, next to where it says 'if'.



Then drag in 3 'else if' by attaching them to the 'if' on the right panel which appears. Then hit the blue icon again to close this.

Next you need it to display the other directions it will face. You'll need to decide the boundaries which will be used to determine what letter to show. See how the 'E' for east is done below.



JavaScript:

```
let Angle = 0
```

```
basic.forever(() => {
```

```
  Angle = input.compassHeading()
```

```
  if (Angle > 315 && Angle <= 360 || Angle >= 0 && Angle <= 45) {
```

```
    basic.showString("N")
```

```
  } else if (Angle > 45 && Angle <= 135) {
```

```
    basic.showString("E")
```

```
  }}
```

