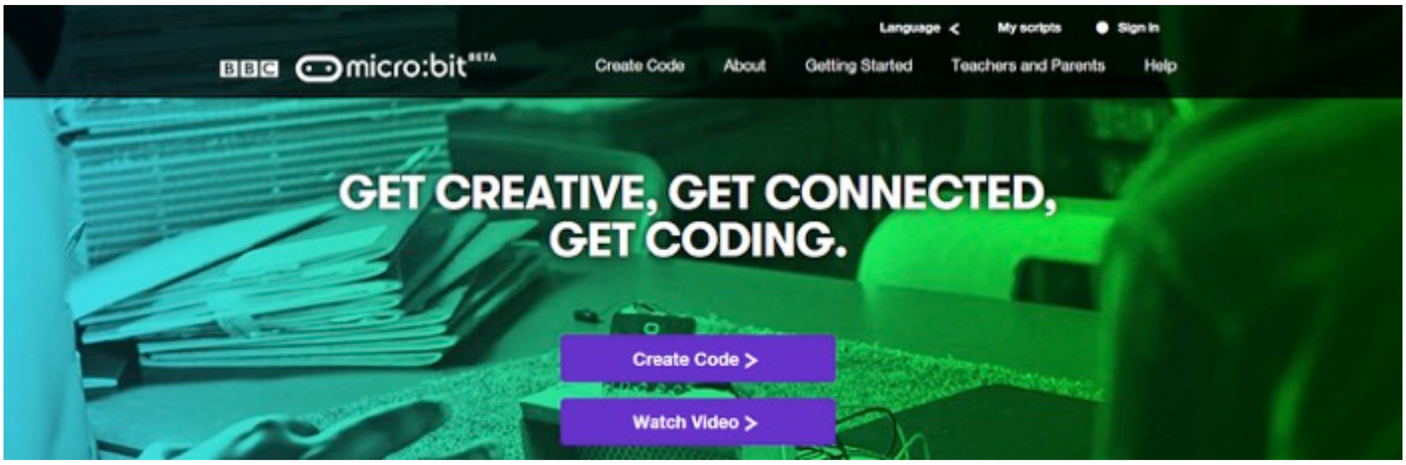
Make your very own micro:monster (advanced)

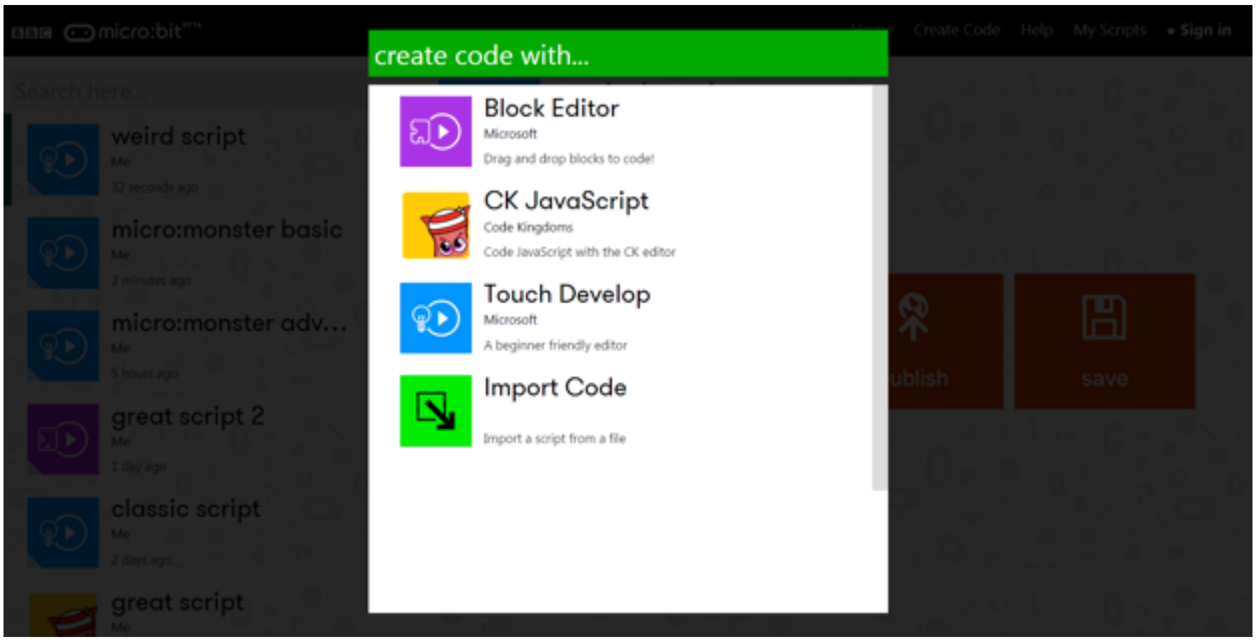
Step 1: Import the code

Download the hex file from our Live Lessons website by clicking on the **micro:monster basic hex file** link.

Firstly, select **'My scripts'** on the top navigation on the micro:bit website (**www.microbit.co.uk**), and choose **'Create code'**.



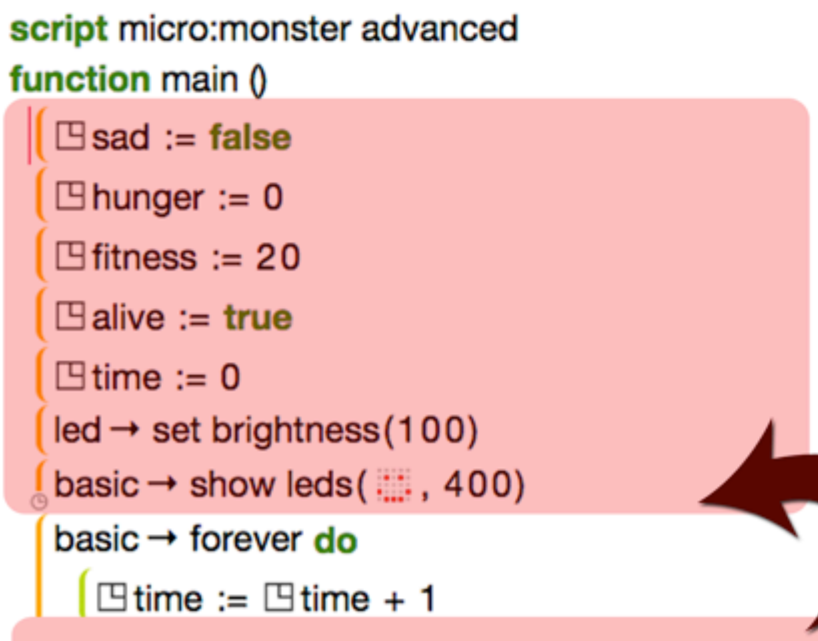
Choose **'Import Code'** and upload the hex file that you've downloaded from the Live Lessons website.



The code for your micro:monster should now appear in your code window.

Hit **'run'** to see it in action on the simulator, or plug in your micro:bit, hit '**compile**' and drag your hex file onto your micro:bit to try out your micro:monster.

Step 2: Understanding the code



```
script micro:monster advanced
function main ()
  sad := false
  hunger := 0
  fitness := 20
  alive := true
  time := 0
  led → set brightness(100)
  basic → show leds( , 400)
  basic → forever do
    time := time + 1
```

How your micro:monster looks at the start

This is where you determine how your micro:monster looks when you first power up.

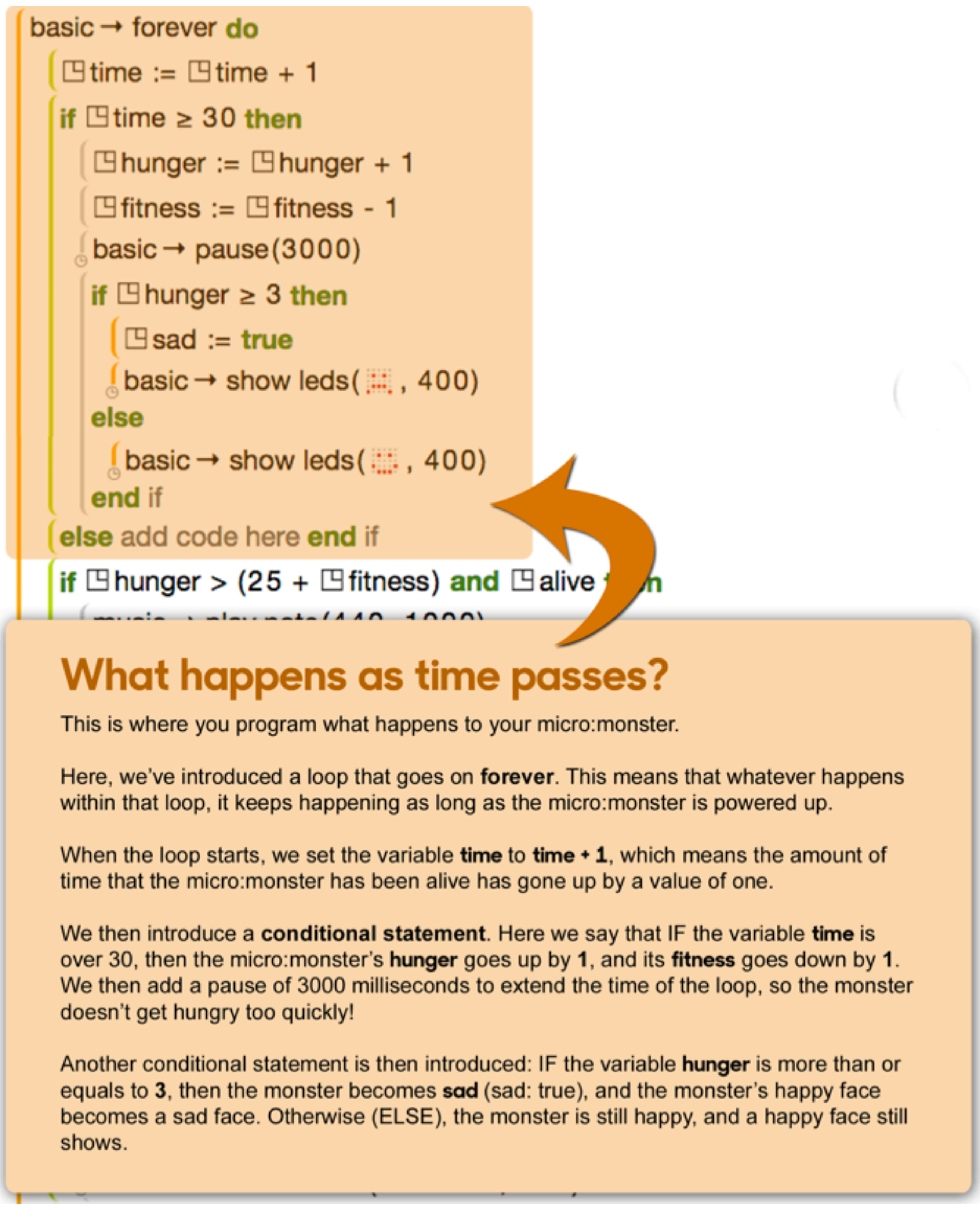
Here, we've set four variables:

alive	this is a variable that calls for a Boolean data type (**true** or **false**)
sad	this is a variable that calls for a Boolean data type (**true** or **false**)
hunger	this is a variable that calls for an integer data type (**numbers**)
fitness	this is a variable that calls for an integer data type (**numbers**)
time	this is a variable that calls for an integer data type (**numbers**)

We've written that the micro:monster is happy when you first start up (so **sad**: **false**), it's not hungry (so **hunger**: **0**) and it's also got an average fitness level (so **fitness**: **20**).

We've also said that the micro:monster is brand new - so it's been alive for zero amount of time (**time**: **0**). Of course, your micro:monster is also **alive**.

Because it's not sad, the micro:monster has a smiley face, so we've drawn a smiley face using the LEDs.



```
basic → forever do
    time := time + 1
    if time ≥ 30 then
        hunger := hunger + 1
        fitness := fitness - 1
        basic → pause(3000)
        if hunger ≥ 3 then
            sad := true
            basic → show leds( , 400)
        else
            basic → show leds( , 400)
        end if
    else add code here end if
    if hunger > (25 + fitness) and alive t n
```

What happens as time passes?

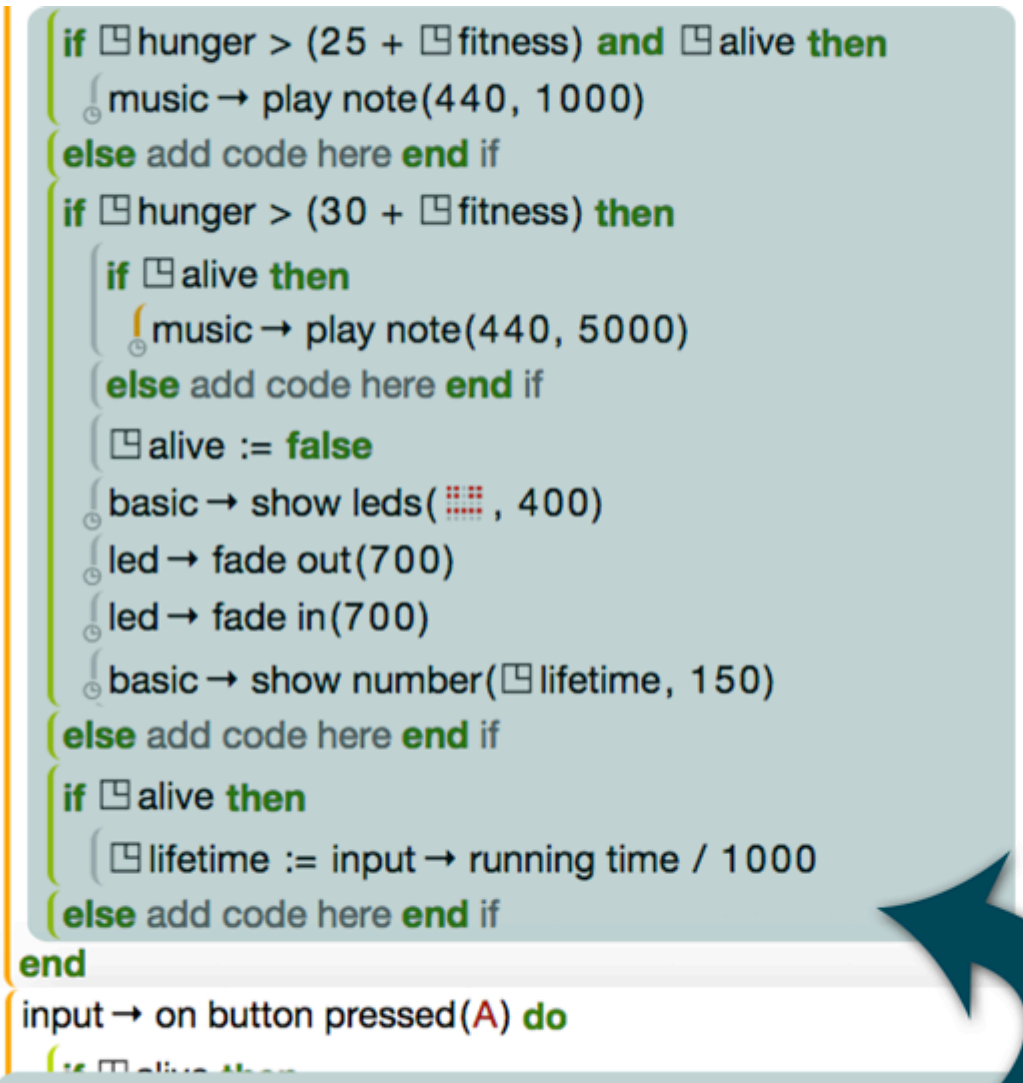
This is where you program what happens to your micro:monster.

Here, we've introduced a loop that goes on **forever**. This means that whatever happens within that loop, it keeps happening as long as the micro:monster is powered up.

When the loop starts, we set the variable **time** to **time + 1**, which means the amount of time that the micro:monster has been alive has gone up by a value of one.

We then introduce a **conditional statement**. Here we say that IF the variable **time** is over 30, then the micro:monster's **hunger** goes up by **1**, and its **fitness** goes down by **1**. We then add a pause of 3000 milliseconds to extend the time of the loop, so the monster doesn't get hungry too quickly!

Another conditional statement is then introduced: IF the variable **hunger** is more than or equals to **3**, then the monster becomes **sad** (sad: true), and the monster's happy face becomes a sad face. Otherwise (ELSE), the monster is still happy, and a happy face still shows.



```
if hunger > (25 + fitness) and alive then
  music → play note(440, 1000)
else add code here end if
if hunger > (30 + fitness) then
  if alive then
    music → play note(440, 5000)
  else add code here end if
  alive := false
  basic → show leds( , 400)
  led → fade out(700)
  led → fade in(700)
  basic → show number(lifetime, 150)
else add code here end if
if alive then
  lifetime := input → running time / 1000
else add code here end if
end
input → on button pressed(A) do
  if alive then
```

Life and death

This is where you program what happens to your micro:monster when you don't take good care of it.

Here, we've said that if the micro:monster gets hungry and unfit **(hunger > (25 + fitness)**, you are given a warning tone, at the frequency of 440hz and for 1000ms.

If you micro:monster gets too hungry and too unfit **(hunger > (30 + fitness)**, then it dies (**alive**: **false**). A face displays to show that it's dead, and the LEDs fade out, before fading back in again to show how long you've managed to keep it alive.

We've also introduced a final IF condition that allows the program to sound a flatline tone (440hz for 5000ms) before it dies.

The final IF condition in this sequence allows you to see in seconds how long you've kept your micro:monster alive for. It states that if your micro:monster is **alive**, then the monster's **lifetime** (another variable) is the program's running time in seconds.

Taking care of your micro:monster

This is where you program actions to help make your micro:monster less or more hungry, which in turn decides whether your monster is happy or sad.

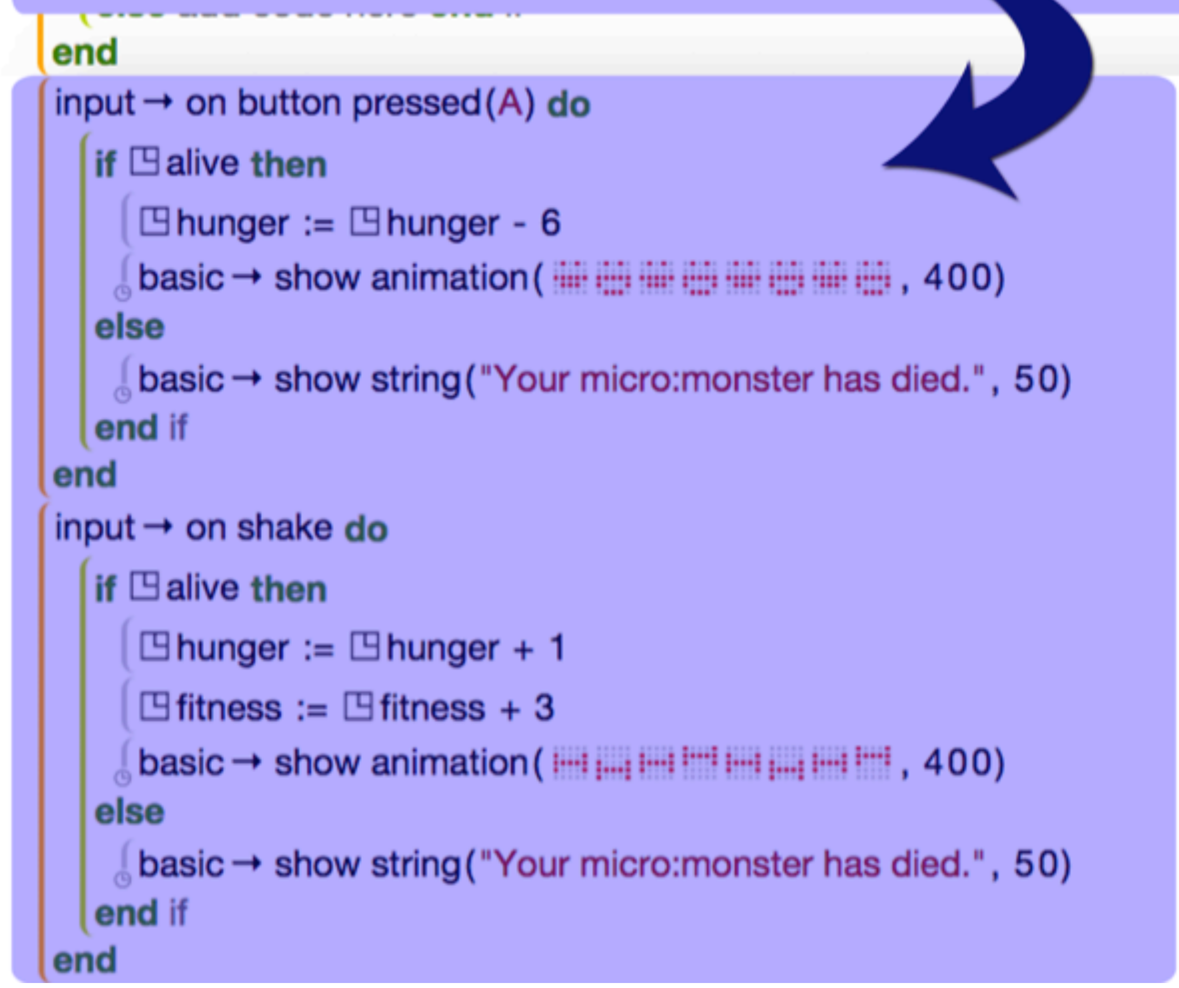
Here we've said that when you press button A, you 'feed' the micro:monster, and its **hunger** goes down (**hunger - 6**). The feeding action is displayed using a short animation, where the monster opens and closes its mouth. So when your micro:monster is **sad**, you can press button A to feed it and try and make it happy again.

If your micro:monster is dead (**alive**: **false**), then a message string is shown, simply saying "Your micro:monster has died.".

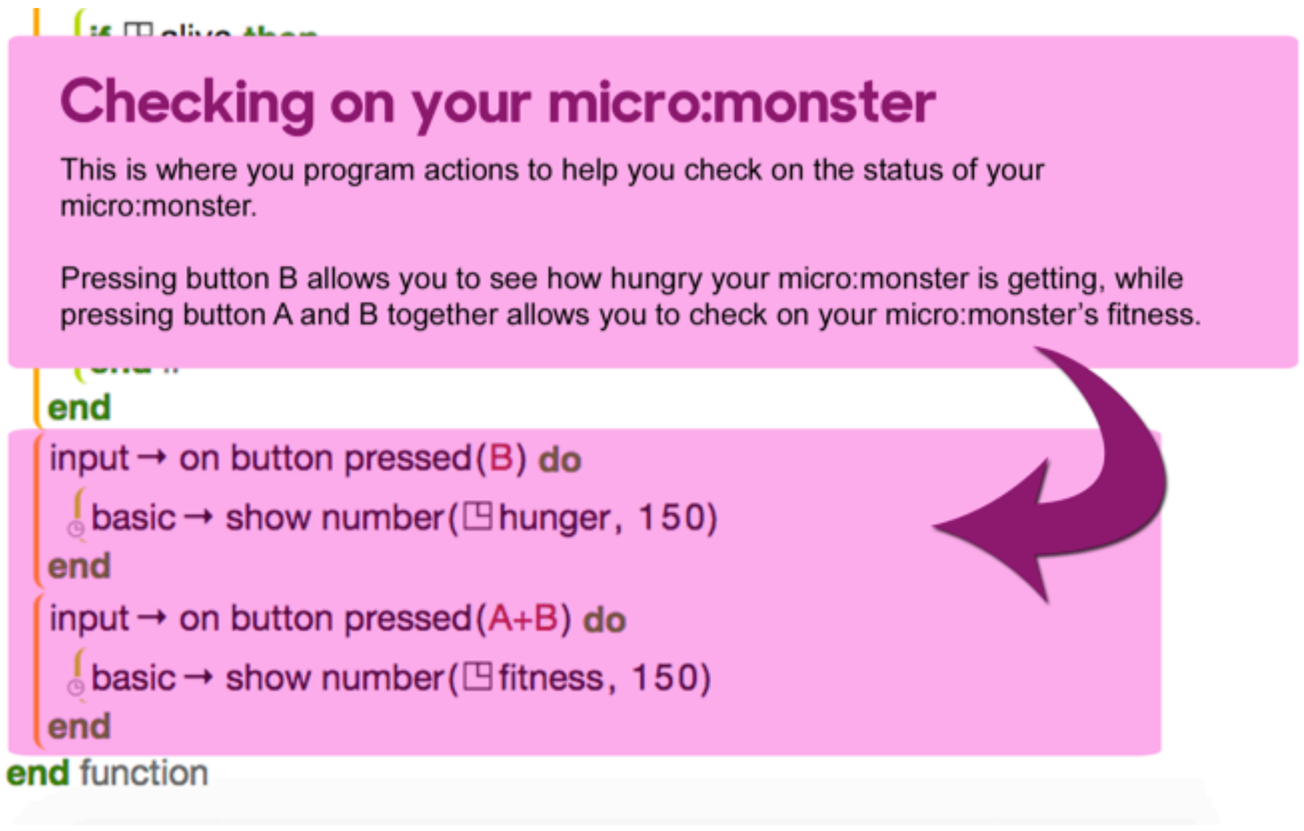
To 'exercise' your micro:monster and bring its **fitness** up (**fitness + 3**), you can shake your micro:bit. However, when you exercise, you also get hungry, so its hunger also goes up a little (**hunger + 1**).

Therefore, you need to make sure that while your micro:monster gets fitter, it also gets well fed, so it doesn't get too hungry.

If your micro:monster is dead (**alive**: **false**), then a message string is shown, simply saying "Your micro:monster has died.".



```
end
input → on button pressed(A) do
  if alive then
    hunger := hunger - 6
    basic → show animation( , 400)
  else
    basic → show string("Your micro:monster has died.", 50)
  end if
end
input → on shake do
  if alive then
    hunger := hunger + 1
    fitness := fitness + 3
    basic → show animation( , 400)
  else
    basic → show string("Your micro:monster has died.", 50)
  end if
end
```



Checking on your micro:monster

This is where you program actions to help you check on the status of your micro:monster.

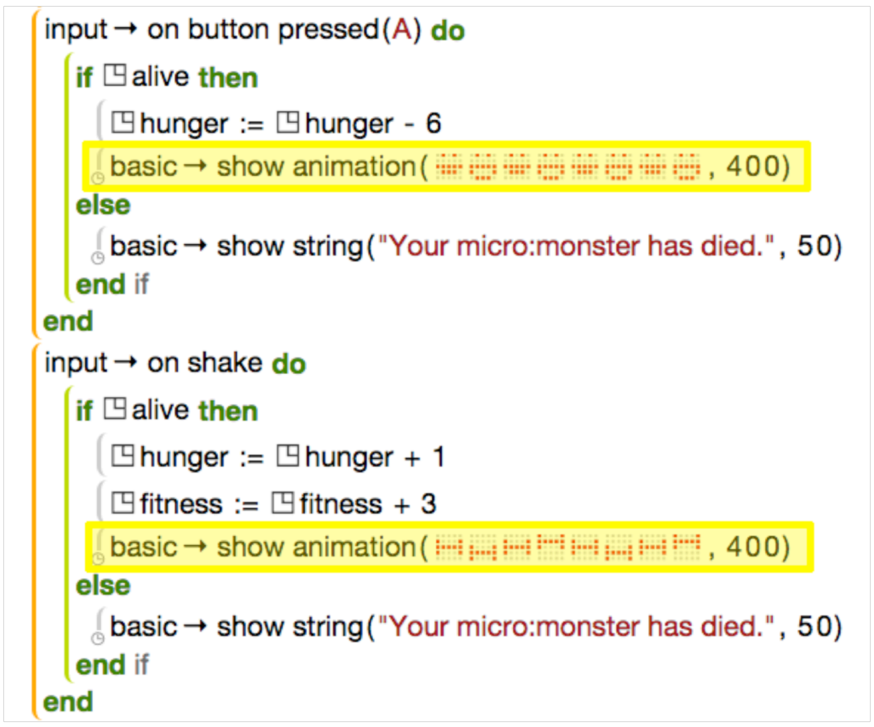
Pressing button B allows you to see how hungry your micro:monster is getting, while pressing button A and B together allows you to check on your micro:monster's fitness.

```
end
input→ on button pressed(B) do
  basic→ show number(hunger, 150)
end
input→ on button pressed(A+B) do
  basic→ show number(fitness, 150)
end
end function
```

Step 3: Modifying the code

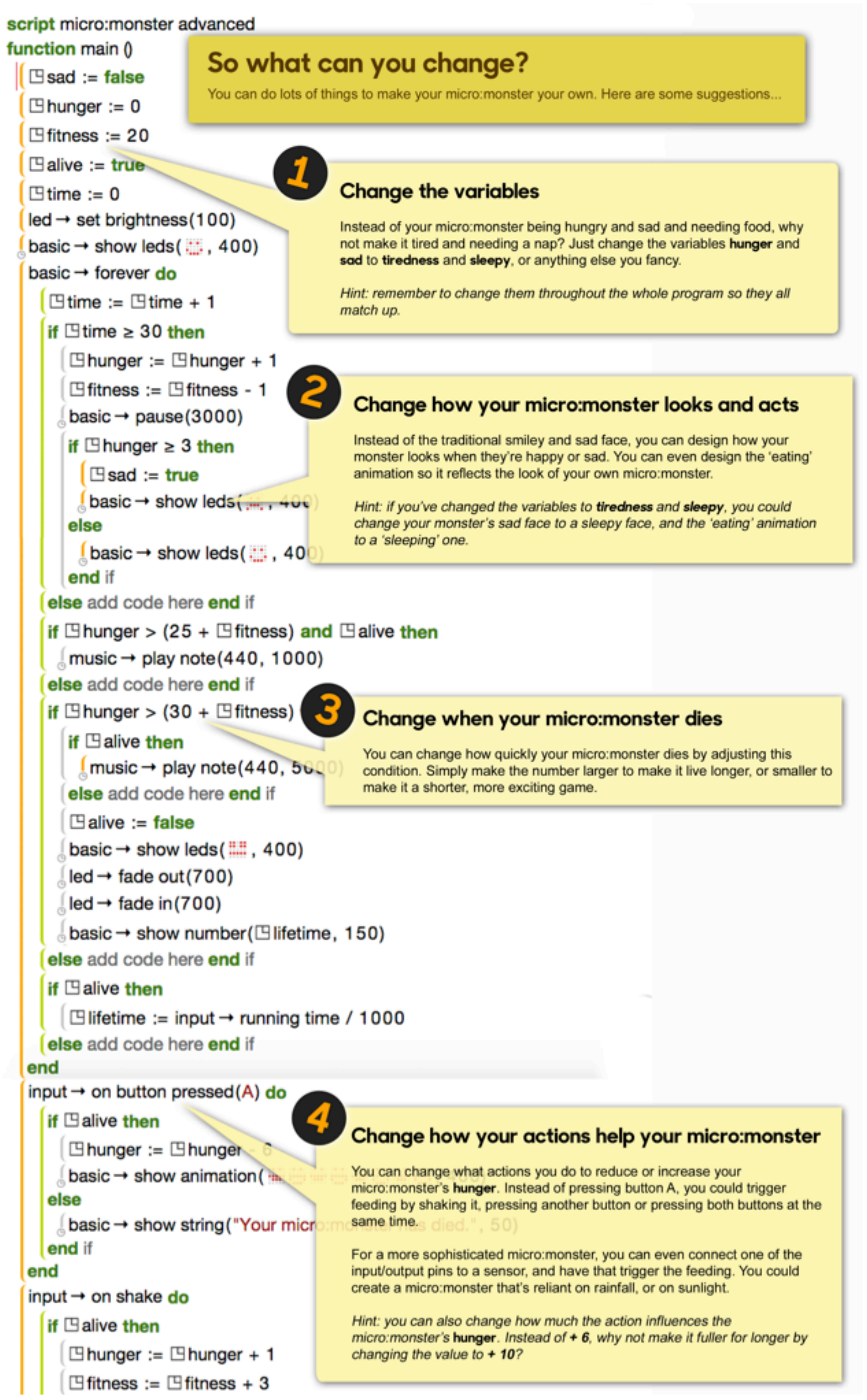
There are lots of things you can do to adapt your micro:monster and make it your own.

In the Live Lesson, we want you to make your own animation for eating and exercising. Click on these lines of code in the editor and change which LEDs light up to create your own animations:



```
input→ on button pressed(A) do
  if alive then
    hunger := hunger - 6
    basic→ show animation(, 400)
  else
    basic→ show string("Your micro:monster has died.", 50)
  end if
end
input→ on shake do
  if alive then
    hunger := hunger + 1
    fitness := fitness + 3
    basic→ show animation(, 400)
  else
    basic→ show string("Your micro:monster has died.", 50)
  end if
end
```

If you like, you can also alter different parts of the code.



```
script micro:monster advanced
function main ()
  sad := false
  hunger := 0
  fitness := 20
  alive := true
  time := 0
  led → set brightness(100)
  basic → show leds( , 400)
  basic → forever do
    time := time + 1
    if time ≥ 30 then
      hunger := hunger + 1
      fitness := fitness - 1
      basic → pause(3000)
      if hunger ≥ 3 then
        sad := true
        basic → show leds( , 400)
      else
        basic → show leds( , 400)
      end if
    else add code here end if
    if hunger > (25 + fitness) and alive then
      music → play note(440, 1000)
    else add code here end if
    if hunger > (30 + fitness)
      if alive then
        music → play note(440, 5000)
      else add code here end if
      alive := false
      basic → show leds( , 400)
      led → fade out(700)
      led → fade in(700)
      basic → show number(lifetime, 150)
    else add code here end if
    if alive then
      lifetime := input → running time / 1000
    else add code here end if
end
input → on button pressed(A) do
  if alive then
    hunger := hunger - 6
    basic → show animation(
  else
    basic → show string("Your micro:monster has died.", 50)
  end if
end
input → on shake do
  if alive then
    hunger := hunger + 1
    fitness := fitness + 3
```

So what can you change?

You can do lots of things to make your micro:monster your own. Here are some suggestions...

1 Change the variables

Instead of your micro:monster being hungry and sad and needing food, why not make it tired and needing a nap? Just change the variables **hunger** and **sad** to **tiredness** and **sleepy**, or anything else you fancy.

Hint: remember to change them throughout the whole program so they all match up.

2 Change how your micro:monster looks and acts

Instead of the traditional smiley and sad face, you can design how your monster looks when they're happy or sad. You can even design the 'eating' animation so it reflects the look of your own micro:monster.

Hint: if you've changed the variables to ***tiredness*** *and* ***sleepy****, you could change your monster's sad face to a sleepy face, and the 'eating' animation to a 'sleeping' one.*

3 Change when your micro:monster dies

You can change how quickly your micro:monster dies by adjusting this condition. Simply make the number larger to make it live longer, or smaller to make it a shorter, more exciting game.

4 Change how your actions help your micro:monster

You can change what actions you do to reduce or increase your micro:monster's **hunger**. Instead of pressing button A, you could trigger feeding by shaking it, pressing another button or pressing both buttons at the same time.

For a more sophisticated micro:monster, you can even connect one of the input/output pins to a sensor, and have that trigger the feeding. You could create a micro:monster that's reliant on rainfall, or on sunlight.

Hint: you can also change how much the action influences the micro:monster's ***hunger****. Instead of* ***+ 6****, why not make it fuller for longer by changing the value to* ***+ 10****?*

Test, play and show us what you’ve done

Now that you’ve made your very own micro:monster, click ‘**run**’ to test it on the simulator and ‘**compile**’ to see it working on your micro:bit.

Click ‘**export**’ to save off your code and send it to us at **live.lessons@bbc.co.uk**. You could see your micro:monster featured on our **micro:bit Live Lesson** in February.