

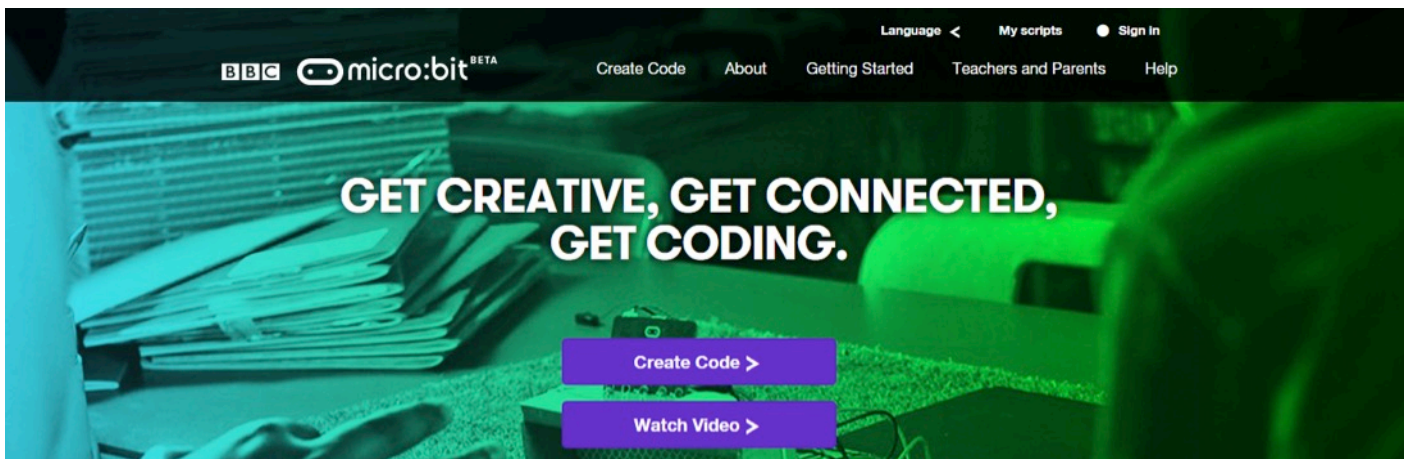


Make your very own micro:monster (basic)

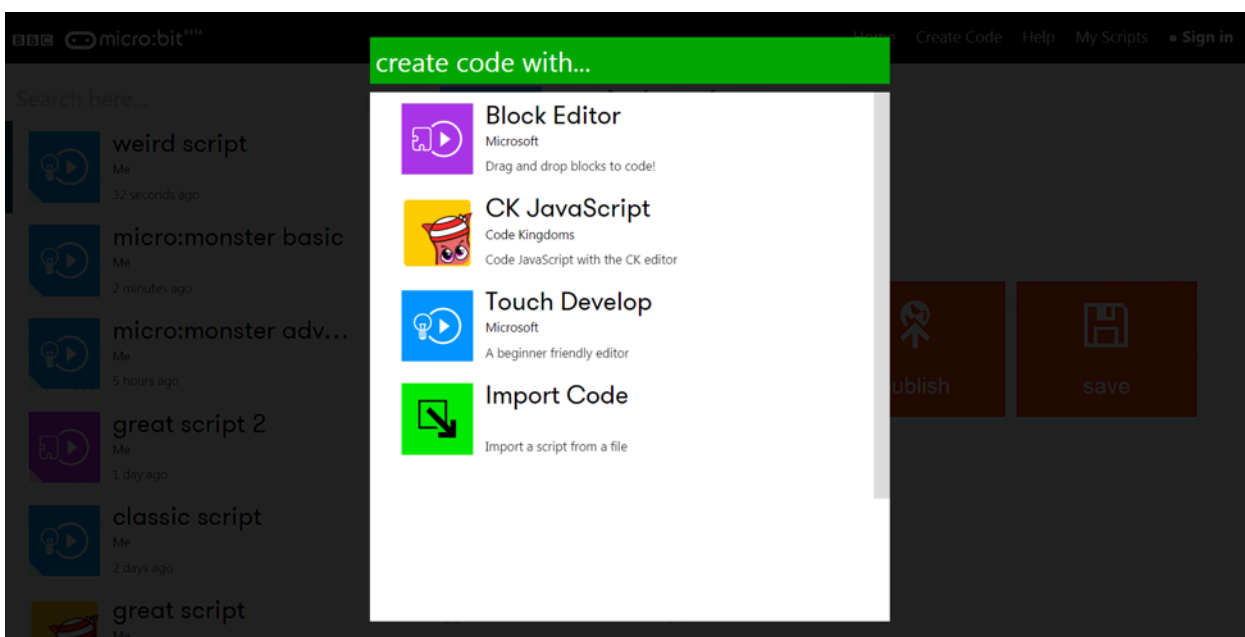
Step 1: Import the code

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 basic

function main ()

```
{ alive := true
  sad := false
  hunger := 0
  time := 0
  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**)
- 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**), and that it's not hungry (so **hunger: 0**). 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 → show animation(📺📺📺📺📺📺📺📺, 400)
```

```
basic → pause(100)
```

end

```
input → on button pressed(B) do
```

```
basic → show number(📺 hunger, 150)
```

end

end function



my scripts



run main



compile



undo

Search code...

script micro:monster basic

function main ()

{ □ alive := **true**

{ □ sad := **false**

{ □ hunger := 0

{ □ time := 0

basic → show leds(📺 , 400)

basic → forever **do**

{ □ time := □ time + 1

if □ time ≥ 30 **then**

{ □ hunger := □ hunger + 1

basic → pause(6000)

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 > 30 **then**



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. We then add a pause of 6000 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.



my scripts



run main



compile



undo

Search code...

script micro:monster basic

function main ()

```

{ alive := true
{ sad := false
{ hunger := 0
{ time := 0

```

Game over

If you don't keep your micro:monster happy and feed it regularly, it meets with a sad demise.

In this block of code, we've introduced a conditional statement that says IF your monster's **hunger** is more than **30**, then the **alive** variable is set to **false**.

Your micro:monster is now dead. A 'dead' face is programmed to appear on the LED screen, before the screen fades out.

You will have to reset your micro:bit in order to revive your micro:monster.

```

    end if
  else add code here end if
  if hunger > 30 then
    alive := false
    basic → show leds( [sad face], 400)
    led → fade out(1000)
  else add code here end if
end
input → on button pressed(A) do
  hunger := hunger - 3
  basic → show animation( [happy faces], 400)
  basic → pause(100)
end
input → on button pressed(B) do
  basic → show number(hunger, 150)
end
end function

```





my scripts



run main



compile



undo

Search code...

script micro:monster basic

function main ()

```

{ alive := true
{ sad := false
{ hunger := 0
{ time := 0
basic → show leds( [LED], 400)
basic → forever do
{ time := time + 1
if time ≥ 30 then
{ hunger := hunger + 1

```

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** - 3). 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.

We've also allowed you to check how hungry your micro:monster is, and how much you have to feed it, by pressing button B.

```

{ else add code here end if
end

```

```

input → on button pressed(A) do
{ hunger := hunger - 3
basic → show animation( [LED], 400)
basic → pause(100)
end
input → on button pressed(B) do
basic → show number(hunger, 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
  (
    hunger := hunger - 3
    basic → show animation( , 400)
    basic → pause(100)
  )
end
```

If you like, you can also alter different parts of the code to help make your micro:monster more personal to you.

You can make it easier or harder to take care of your micro:monster, and can also change the way your micro:monster looks and acts.

Have a look at the instructions on the next page to see what you can do.

So what can you change?

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

```

script micro:monster basic
function main ()
  ☐ alive := true
  ☐ sad := false
  ☐ hunger := 0
  ☐ time := 0
  basic → show leds(☐, 400)
  basic → forever do
    ☐ time := ☐ time + 1
    if ☐ time ≥ 30 then
      ☐ hunger := ☐ hunger + 1
      basic → pause(6000)
      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 > 30 then
      ☐ alive := false
      basic → show leds(☐, 400)
      led → fade out(1000)
    else add code here end if
  end
  input → on button pressed(A) do
    ☐ hunger := ☐ hunger - 3
    basic → show animation(☐, 400)
    basic → pause(100)
  end
  input → on button pressed(B) do
    basic → show number(☐ hunger, 150)
  end
end function

```

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 programme 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 fact 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 + 3, 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.