BBC Micro: bits – Starfall

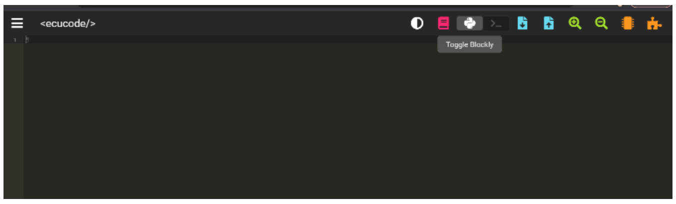
ECU Makerspace Group

Written by Sam Blizard and Leanne Goh

Getting started

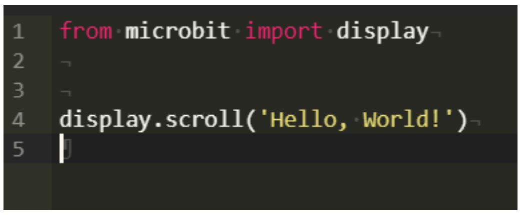
Let's do a practice run just using the Micro: bit LED's. Once you have access to the computers, follow through this guide. If your computer is not on the makerspace website then use this link to gain access. **Please use Google Chrome to access this website (makerspace.ecu.edu.au/code)**

Click on **'Toggle Blockly'** to switch to Python code.



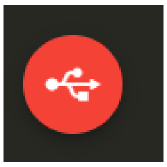
Step One – Print your name:

Click on to the first line and import 'display' from 'microbit', then display a message using display.scroll(" ").

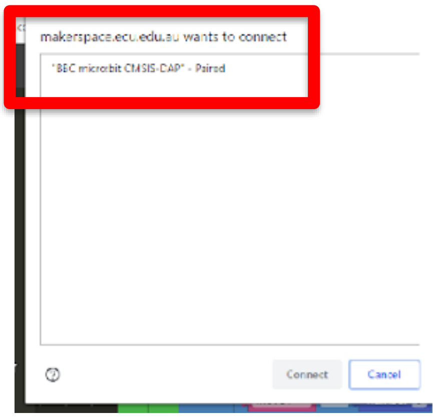


```
from microbit import display


display.scroll('Hello, World!')

```

Step Two - Now that you have the scroll message on the page, write your name and upload it to the Micro: Bit. Do this by following these instructions:



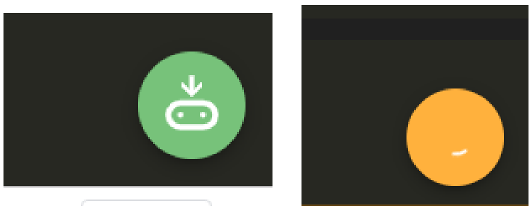
Click on the red button then a box comes up inviting you to pair the micro: bit. Click on the "BBC Micro: bit CMSIS-DAP"



makerspace.ecu.edu.au wants to connect

"BBC microbit CMSIS-DAP" - Paired

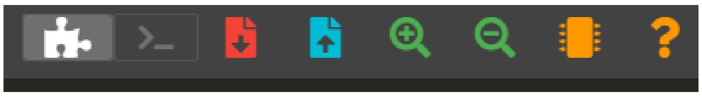
Once this has been completed click on the green button to download. An orange button will appear as it is downloading.



Congratulations! Remember to delete this scroll message before starting the next section.

Troubleshooting:

Updating the firmware on the Micro: bit

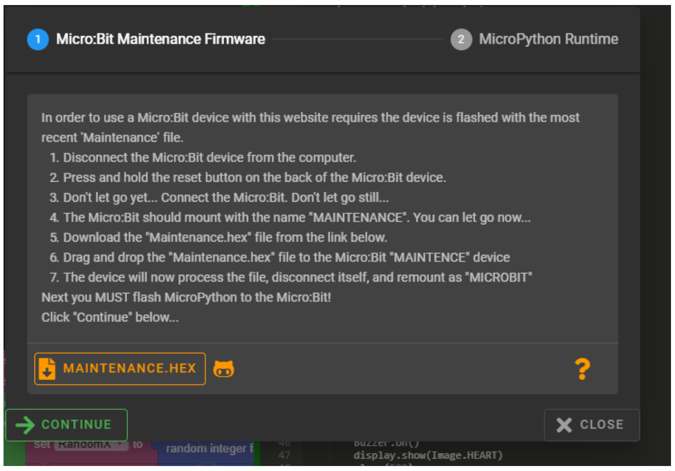


Click on the orange shape which appears 'update micro: bit'. Follow set instructions on the **Maintenance Firmware** guide.

BBC Micro: bits – Starfall

ECU Makerspace Group

Written by Sam Blizard and Leanne Goh



In order to use a Micro:Bit device with this website requires the device is flashed with the most recent 'Maintenance' file.

1. Disconnect the Micro:Bit device from the computer.
2. Press and hold the reset button on the back of the Micro:Bit device.
3. Don't let go yet... Connect the Micro:Bit. Don't let go still...
4. The Micro:Bit should mount with the name "MAINTENANCE". You can let go now...
5. Download the "Maintenance.hex" file from the link below.
6. Drag and drop the "Maintenance.hex" file to the Micro:Bit "MAINTENCE" device
7. The device will now process the file, disconnect itself, and remount as "MICROBIT"

Next you MUST flash MicroPython to the Micro:Bit!
Click "Continue" below...

If the micro: bit appears not to be responding. Unplug the micro: bit, press the reset button at the back of the micro: bit and repeat this process. If the micro: bit continues to fail, swap it for another one!

Now let's learn how to write a program so we can play our own Starfall game.

First, we need to import microbit and the random module, so that we can control the microbit.

```python
from microbit import *
import random

```

In the next step, we need to set the starting position of the player, which is represented by a pixel. For now, we set it to 0.

```python
player_x = 0
```

Now it is time to add stars to our game! To make sure that the stars are unique, they each need to have their own version of the y coordinate. The best way to do this is by creating a list of lists. Each star will be a list of lists and each list will contain the x and y coordinates for each star.

```python

stars = [[4, 3], [3, 2], [4, 1], [3, 0]]

```

As we will be using the buttons (button a and button b) on the microbit, we need to set up variables, so we can later control whether a button was pressed. For now, we set the variables to 0.

```python
presses_a = 0
presses_b = 0

```

To keep track of our score, we need to create a variable that holds the score. We want to make sure that no matter how many stars we add to the game, we will always have the correct score that is based on how many stars move up – 1 score for each star that moves up.

```python
score = 0 - len(stars)
```

Next, we need to set the speed with which the stars move. Here, we set it to 300, but you can reduce the speed to make it a little easier.

```python
speed = 300

```

BBC Micro: bits – Starfall

ECU Makerspace Group

Written by Sam Blizard and Leanne Goh

We want to make sure that if we re start the game, the stars from previous rounds are removed, so we need to clear the display.

Also, so that we can keep track of whether the game has been lost, we create a game over variable that we set to False. We don’t want to start playing the game and lose immediately!

```
display.clear()

game_over = False
```

As we want to play the game until we lose or decide that we don’t want to play anymore, we are going to add a loop, which keeps the game running until we lose or quite the game. Make sure that you place the next parts of your code within the while True loop.

```
while True:

```

At the moment, we cannot control our player – so need to change that. First, we need to check if button was pressed AND we need to check whether the player is on the edge – if the player was on the edge and we moved it, would fall out of the screen.

Once we know that button a has been pressed and the player is not on the edge of the screen, we move the player to the left.

```
while True:

    if button_a.was_pressed() and player_x > 0:
        player_x -= 1

```

Now we repeat the same for button b, so that we can also move the player to the right.

```
    if button_b.was_pressed() and player_x < 4:
        player_x += 1

```

Now our player can move to the left and right!

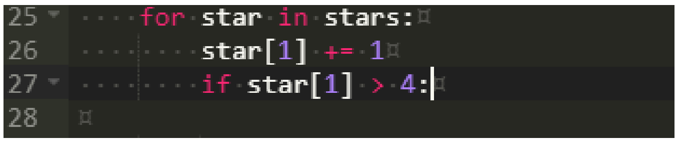
Next, we are going to add the stars.

As we want to loop over our stars to display them all at the same time (instead of one by one), we are going to create a loop. This also ensures that the logic for the starts applies to ALL stars at any time.

We just need to make sure that the next parts of the code for the stars is all placed within this loop.

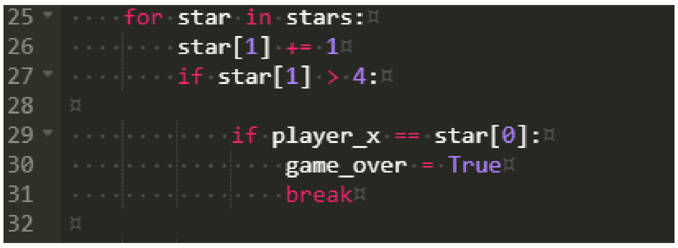
```
    for star in stars:
```

We want to make sure that for each start, the y-coordinates increment by 1, which will make the star move down from the top of the display. Also, we need to check whether a star has reached the bottom of the display.



```
    for star in stars:
        star[1] += 1
        if star[1] > 4:

```

The game is game over when the player collides with a star at the bottom of the display. Lets implement this. If the player and a star have the same coordinates, we set the game over variable to true and break out of the loop – Game over!



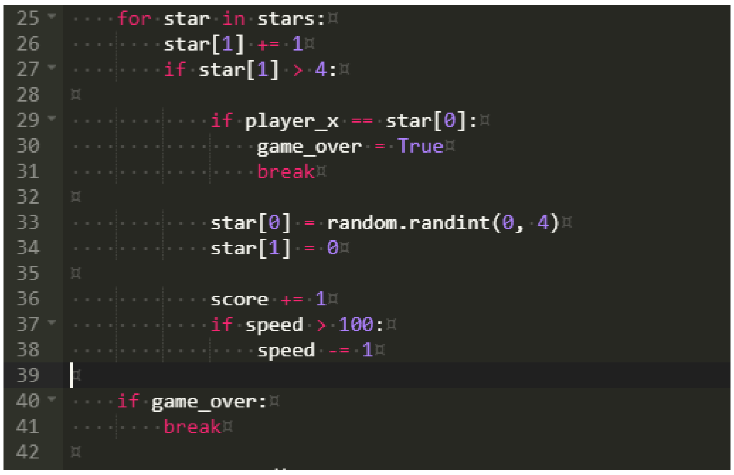
```
    for star in stars:
        star[1] += 1
        if star[1] > 4:

            if player_x == star[0]:
                game_over = True
                break

```

In the next step, we will create random x coordinates to the stars, so that we don’t get exactly the same set of starts every time we play.
We also set the y coordinate of the star to 0, so the star is placed at the top of the display when we start the game.

We’ll also update the score, so every time a star reaches the bottom of the display without having collided with the player, we increment the score by 1.

We also need to check whether the current speed of the game is greater than 100, so we can control the game difficulty. If the speed is above 100, the game automatically decreases the speed by 1.

After we have done that, we just need to check whether the game over variable is set to True. If it is set to true, it means that a collision has occurred and the game ends.



```
    for star in stars:
        star[1] += 1
        if star[1] > 4:

            if player_x == star[0]:
                game_over = True
                break

            star[0] = random.randint(0, 4)
            star[1] = 0

            score += 1
            if speed > 100:
                speed -= 1

    if game_over:
        break

```

Next, we set up a couple for things for our game, before it is ready to be played.

First, we clear the display to make sure that it resets every time we start a game.

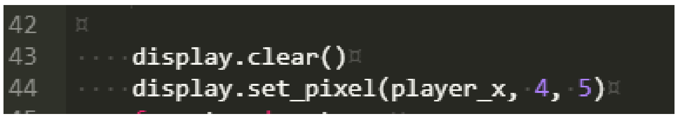
We also want to make sure that the position of the player is on the bottom row (4) and is set to

BBC Micro: bits – Starfall

ECU Makerspace Group

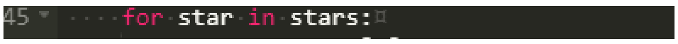
Written by Sam Blizard and Leanne Goh

brightness 5. 0 means that the pixel is dark and we would not be able to see it.



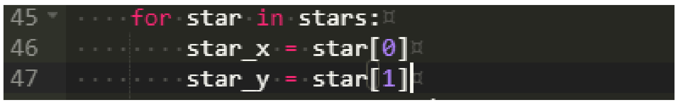
```python

    display.clear()
    display.set_pixel(player_x, 4, 5)
```

Next, we need to iterate over our star list again to make sure the same applies to all stars



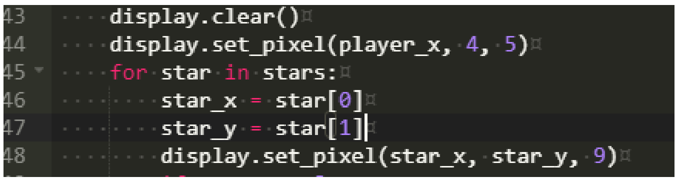
```python
    for star in stars:
```

Within that loop, we want to retrieve the x and y coordinates for each star, so that we can reference it later.



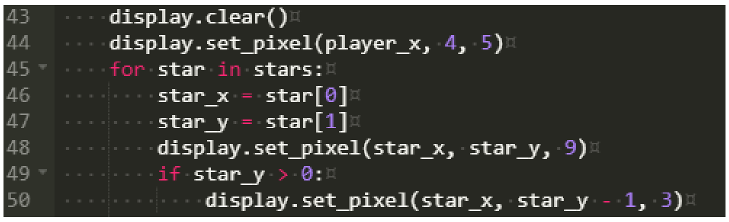
```python
    for star in stars:
        star_x = star[0]
        star_y = star[1]
```

Next, we need to set the brightness for the pixel of each start to 9. Nine is as bright as it gets. Remember, if you set it to 0, you make the stars invisible, and if you set it to 5 (which is the same brightness as the player), you can no longer distinguish the star from the player.



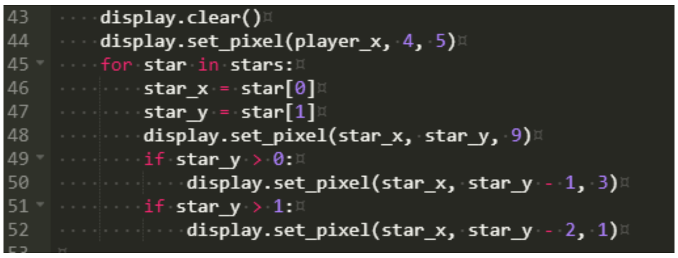
```python
    display.clear()
    display.set_pixel(player_x, 4, 5)
    for star in stars:
        star_x = star[0]
        star_y = star[1]
        display.set_pixel(star_x, star_y, 9)
```

Next, we want to check whether the star is not at the top row of the display. If it is not at the top row, we set a pixel one row above the stars current position with a brightness level of 3. This will make it look like there is a fading trail behind the falling star.



```python
    display.clear()
    display.set_pixel(player_x, 4, 5)
    for star in stars:
        star_x = star[0]
        star_y = star[1]
        display.set_pixel(star_x, star_y, 9)
        if star_y > 0:
            display.set_pixel(star_x, star_y - 1, 3)
```

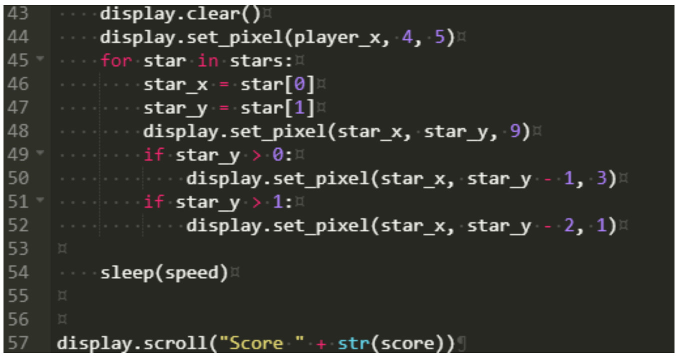
If the star two or more rows above the bottom row of the display, we set a pixel two rows above the stars position with a brightness level of 1. This way we can add another fading trail above the star, which makes the trail appear longer.



```python
    display.clear()
    display.set_pixel(player_x, 4, 5)
    for star in stars:
        star_x = star[0]
        star_y = star[1]
        display.set_pixel(star_x, star_y, 9)
        if star_y > 0:
            display.set_pixel(star_x, star_y - 1, 3)
        if star_y > 1:
            display.set_pixel(star_x, star_y - 2, 1)
```

Once we have done this, we introduce sleep (a delay) into the game, which controls the speed of the game.

Lastly, we just need to print the score to the display to show the final score!



```
    display.clear()
    display.set_pixel(player_x, 4, 5)
    for star in stars:
        star_x = star[0]
        star_y = star[1]
        display.set_pixel(star_x, star_y, 9)
        if star_y > 0:
            display.set_pixel(star_x, star_y - 1, 3)
        if star_y > 1:
            display.set_pixel(star_x, star_y - 2, 1)

    sleep(speed)


display.scroll("Score " + str(score))
```

Congratulations, you have finished your own “Starfall” game!