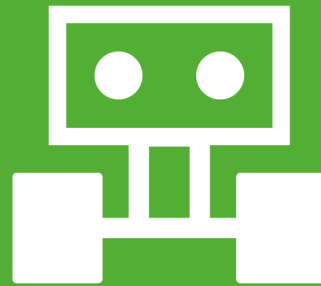




The Digital Skills Standard

# ICDL Digital Student Robotics

Syllabus 1.0



**Learning Material – SAMPLE LESSON**  
(micro:bit)

Copyright ICDL Foundation 2022. All rights reserved. Reproducing, repurposing, or distributing this courseware without the permission of ICDL Foundation is prohibited.

ICDL Foundation, ICDL Europe, ICDL, ECDL and related logos are registered business names and/or trademarks of ECDL Foundation.

This courseware may be used to assist candidates to prepare for the ICDL Foundation Certification Programme as titled on the courseware. ICDL Foundation does not warrant that the use of this courseware publication will ensure passing of the tests for that ICDL Foundation Certification Programme.

Any and all assessment items and/or performance-based exercises contained in this courseware relate solely to this publication and do not constitute or imply certification by ICDL Foundation in respect of the ICDL Foundation Certification Programme or any other ICDL Foundation test. This material does not constitute certification and does not lead to certification through any other process than official ICDL Foundation certification testing.

The official courseware content is written in English. ICDL Foundation shall not be responsible for any errors or omissions contained in translated or other variations of this courseware.

In order to take the official test for the related ICDL Foundation Certification Programme, Candidates using this courseware must be registered with the National Operator. Without a valid registration, the test(s) cannot be undertaken and no certificate, nor any other form of recognition, can be given to a candidate. Registration can be undertaken at an Accredited Test Centre.

Tool and application-specific details are correct as of Sept 2022. Robot kit and the related applications are subject to frequent update and change

This is a sample Lesson from the ICDL BCS Robotics Level 1 course. It can be used without a micro:bit or robot using the makecode coding environment. <http://makecode.microbit.org>

The sample lesson illustrates the student's learning materials provided by the course. This lesson can be used by the students themselves or used by a teacher.

For more details - <https://www.bcs.org/deliver-and-teach-qualifications/teachers-schools-and-colleges/teach-our-digital-literacy-robotics-and-coding-qualifications/teach-bcs-level-1-robotics-award/>

Computing at School Robotics page - <https://www.computingschool.org.uk/cas-robotics-teacher-support-page>

## ICDL Robotics

Robots are an increasingly common feature of modern life. As well as being used in factories and workplaces, robots can now be seen on the streets or in our homes. ICDL Robotics introduces the basic principles of robotics and covers the assembly, programming, and control of a simple robot.

When you have successfully finished this module, you will be able to:

- Understand key concepts relating to robots and robotics systems, and identify examples of robots.
- Identify the main parts of a robot and their function, including microcontrollers, actuators, sensors, and power sources.
- Understand the elements of a simple control system, and test a control system.
- Understand basic programming concepts, and create and execute a programme in a visual programming language.
- Set up a robot, implement robotic motion, and control a robot in an environment.

### What are the benefits of this module?

The ICDL Robotics module has been developed to support students who want to learn more about building and programming robots. Once you have taken this module, you will have the skills and knowledge needed to explore the potential of robots further, creating and innovating with them.

You will also be able to certify in ICDL Robotics. To get certified, you need to pass a test and demonstrate certain skills in using a robot.

For details of the specific areas of the ICDL Robotics syllabus covered in each section of this book, refer to the ICDL Robotics syllabus map at the end of the book.

# ICDL ROBOTICS

- LESSON 1 - ROBOTIC CONCEPTS.....** ERROR! BOOKMARK NOT DEFINED.
- LESSON 2 - ROBOTIC PARTS AND COMPONENTS..** ERROR! BOOKMARK NOT DEFINED.
- LESSON 3 - AN INTRODUCTION TO VISUAL PROGRAMMING.....** ERROR! BOOKMARK NOT DEFINED.
- LESSON 4 - LOCOMOTION AND THE ACTUATOR SYSTEM**ERROR! BOOKMARK NOT DEFINED.
- LESSON 5 - THE SIMPLE CONTROL SYSTEM .....** ERROR! BOOKMARK NOT DEFINED.
- LESSON 6 - TESTING A SIMPLE CONTROL SYSTEM** ERROR! BOOKMARK NOT DEFINED.
- LESSON 7 - PROGRAM CREATION .....**5
- LESSON 8 - PROGRAM ELEMENTS AND CONTROLS**ERROR! BOOKMARK NOT DEFINED.
- LESSON 9 - LOGIC OPERATORS.....** ERROR! BOOKMARK NOT DEFINED.
- LESSON 10 - ASSEMBLING A ROBOT.....** ERROR! BOOKMARK NOT DEFINED.
- LESSON 11 - IMPLEMENTING ROBOTIC MOTION.....** ERROR! BOOKMARK NOT DEFINED.
- LESSON 12 - IMPLEMENTING ROBOTIC CONTROLS** ERROR! BOOKMARK NOT DEFINED.
- LESSON 13 - AUTOMATING A ROBOT I .....** ERROR! BOOKMARK NOT DEFINED.
- LESSON 14 - AUTOMATING A ROBOT II....** ERROR! BOOKMARK NOT DEFINED.
- LESSON 15 - ROBOTIC CONTROL IN AN ENVIRONMENT .**ERROR! BOOKMARK NOT DEFINED.
- LESSON 16 - ETHICS IN THE USE OF ROBOTS.....** ERROR! BOOKMARK NOT DEFINED.

LESSON 7 -  
PROGRAM CREATION

After completing this lesson, you should be able to:

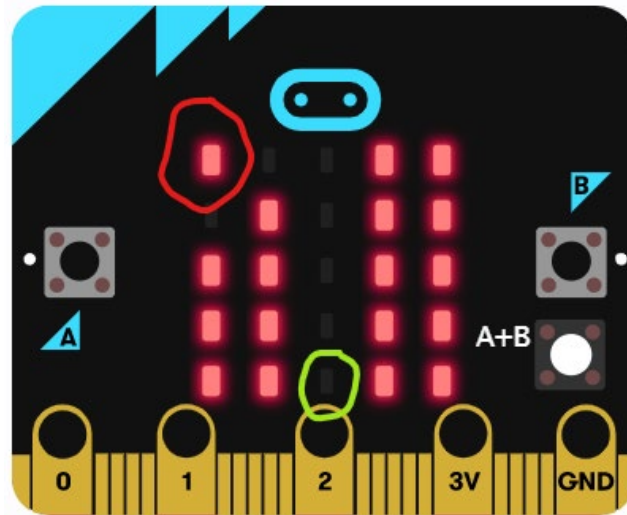
- Recognise typical activities in program creation
- Understand the basic elements of a program
- Use an Events block in a program like: when.
- Understand how a flowchart can be used to present the steps in a solution

Sample Lesson

---

## Scenario

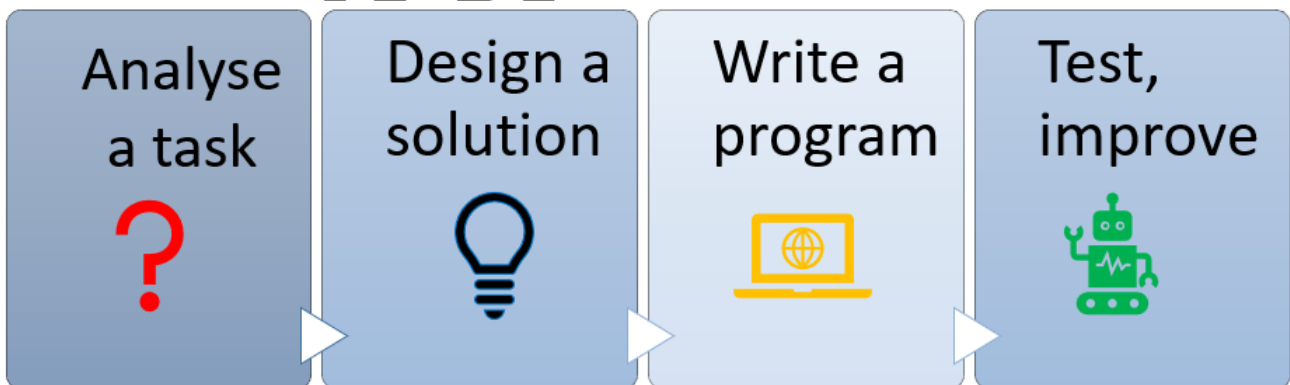
In a simple game, we want to 'move' an LED through a simple route on the micro:bit's LED display. From the red-circled position to the green-circled position.



Working on this 2-dimensional stage, what are the steps needed so that the LED moves to the centre of the display and then 'moves' to the centre of the bottom row on the display.

## Concepts

The following are some typical activities that can guide us in program creation:



### **Analyse a Task**

We will come across different problems when we are trying to get a robot to carry out a task. Before we write a program to solve these problems and achieve a given task, we need to think carefully about it. Because we are applying action to the movement of the LED, we can call this a Sprite. A Sprite is an object, usually in a game, that are visual objects that are controlled and interact in the

game. In some coding languages you can create your own sprites, Such as Scratch.

We do not need that level of complexity with microcontroller, so on the micro:bit a single LED illuminated represents a sprite.

Despite this limitation, how we use code to control the position of the sprite using a micro:bit is similar to other programming languages.

In the given scenario, the task is to have the LED sprite:

- start at the top left-hand corner of the display
- move to the middle of the micro:bit display
- lower itself to the bottom of the micro:bit display.

The task given in this scenario can be completed entirely in the **Makecode** application. Before we create a program to control the LED sprite, we will need to ask a few questions:

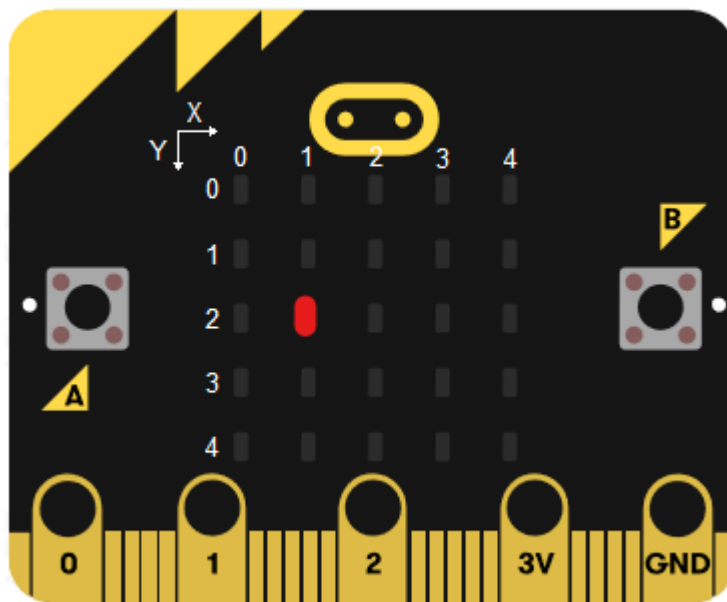
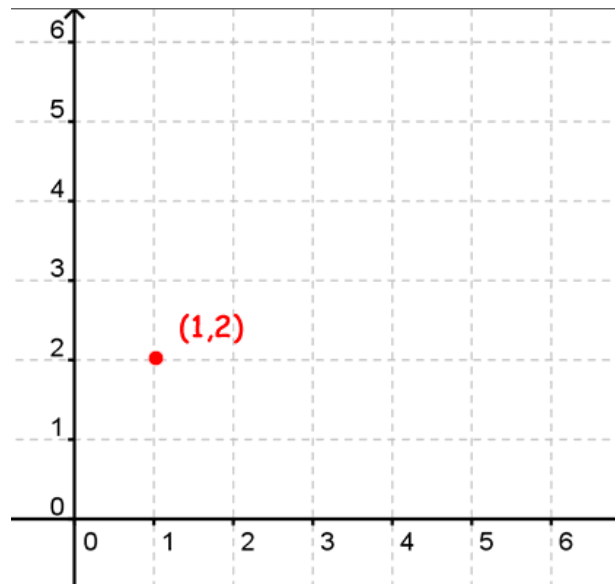
- What is the position of the starting point, the middle of the top row of the display, and the middle of the top row of the display?
- How can we instruct the LED Sprite to move to the final endpoint?
- How can we instruct the LED Sprite to 'move' smoothly?

These questions help us to think about the problems present. We can then consider the resources available to us and seek out possible solutions. This process is known as **Analysis of a Task**.

To Identify the position of a Sprite, we use co-ordinate systems.

A basic coordinate grid has two axes, an x-axis that runs horizontally, and a y-axis which runs vertically. Each axis is divided into equal parts. Where they intersect is the coordinate value. The X value is first. The starting point or origin is on the bottom left.

However, the micro:bit coordinate system is slightly different. It only has a coordinate grid of 5 x 5, giving 25 different positions. The Axes are numbered 0 - 4 and the origin (0,0) starts in the top left corner.



## Concepts

### Design a solution

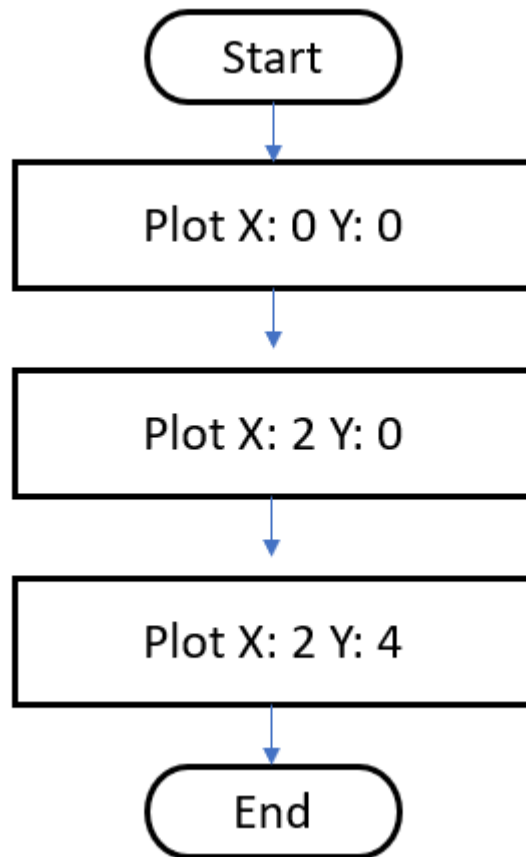
Identify the three coordinates that we can use to place the LED Sprite as stated by our problem. We now need to design a solution.

- Position a LED sprite at **X: 0 , Y: 0** (0,0) to start
- Move the LED sprite to **X: 2 , Y: 0** (2,0)
- Move the LED sprite to **X: 2 , Y: 4** (2,4)



## Steps

We can present these instructions in order graphically by using a flow chart.



A flowchart makes use of common shapes as symbols for the instructions and connecting the related steps with lines.

Some common symbols include:

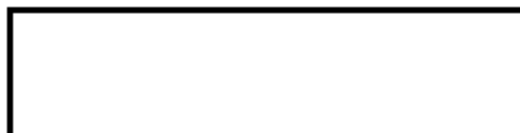
- **Start or Stop**

The algorithm starts at the Start box and runs until it reaches a Stop box.



- **Process**

These boxes contain simple instructions





## Concepts

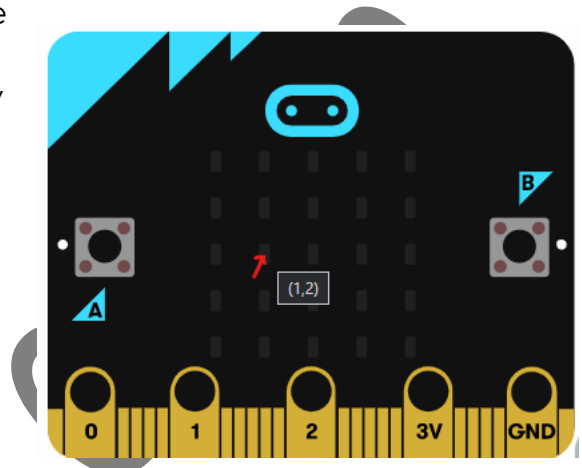
### Write a program

With a simple solution in mind we can create a very simple program that 'moves' the sprite around the display. Moving from the top left, to top middle to bottom middle. Just by changing the coordinates.



### Steps

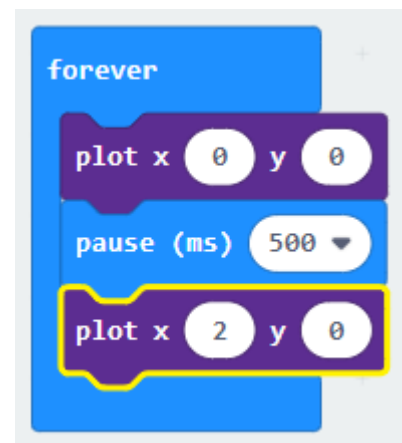
1. Create new project in Makecode. Name your project **Lesson 7 coordinates**.
2. To find out the co-ordinate value of any position on the LED Display, in makecode, hover your mouse over an LED to give you it's co-ordinate value.



3. Drag a **forever** block (an example of an event command), into the makecode code building area. From the **LED** command list drag a **plot x .. y** block and place inside the **forever** block. Check the micro:bit simulator that the correct LED is illuminated.



4. The values for x and y do not need to be changed here as this is our sprite's starting point. Next we need to add a short time delay before we plot the next position, otherwise the LED will all seemingly appear at the same time. From the **Basic** command list drag a **pause** command and change the value to 500ms. From the **LED** command list drag and drop the another **plot x...y** block and change the value to the next position we want the LED to appear which is (2,0). Again, Check the microbit simulator that the correct LEDs are illuminated.



5. Add another **pause** 500ms block and a plot **x .. y** with the final co-ordinates which will be (2,4). Finally add another **pause** block and from the **Basic** command list a clear screen command. Your final code should look like this.

```

forever
  plot x 0 y 0
  pause (ms) 500
  plot x 2 y 0
  pause (ms) 500
  plot x 2 y 4
  pause (ms) 500
clear screen

```

## Concepts

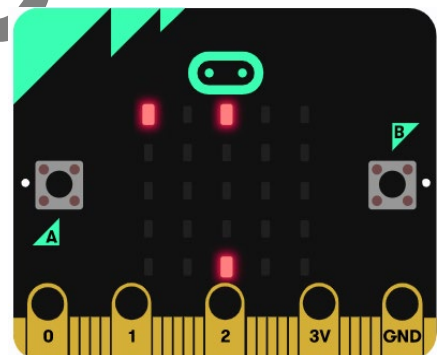
### Test and Improve the Program

When the program is ready, we test it by running the event specified in the program. In the micro:bit simulator you should see the LEDs illuminating in sequence.

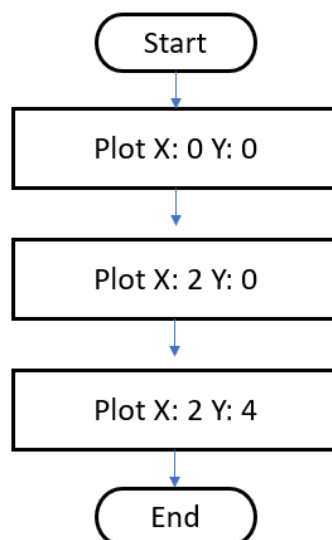
Download the code to your micro:bit and see how the program works on the micro:bit.

You will notice that the LEDs stay illuminated once their position has been plotted.

A way to improve this program would be if the display only showed a single LED Sprite moving around the display.



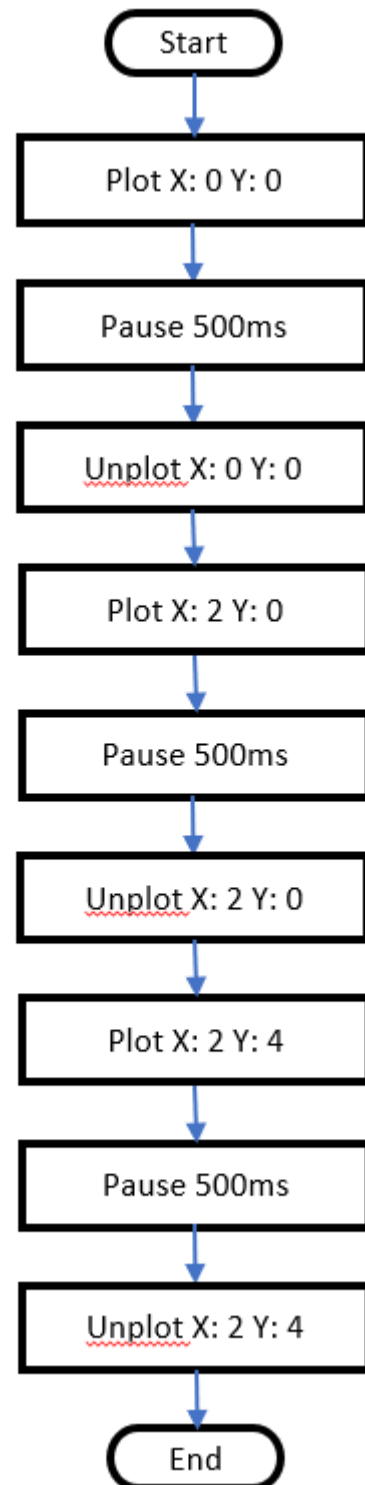
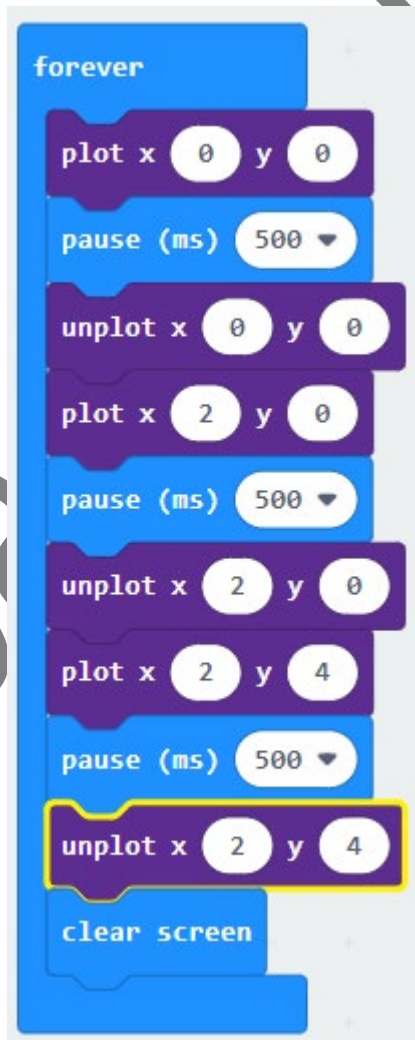
If we revisit the original design solution flowchart and see where we can make improvements.



To improve the program we want the LED sprite to appear, then appear to move to the next position. To do this we need to plot the LED Sprite position, wait for a short time and then unplot the LED sprite position, before plotting the Sprite's next position.

So we can change our design, so our flowchart looks like this.

- This program illuminates the LEDs at the correct co-ordinates. But we can improve the program so that it gives a better animated performance and looks as though the sprite is 'moving'. To do this we need to turn the LED off after a period of time once it has been plotted. The **LED** command list has an **unplot x...y** command block. Drag this block into your program and place after each pause command, with the value of the previous plot command. Your program should look like this.



In the micro:bit simulator you will see the LED now appearing to 'move'.  
OPTIONAL - Download your program to the micro:bit to see how it looks on the micro:controller.

## Concepts

### Using different event commands to execute different instructions

We have learnt that we can change the position of an LED sprite on a micro:bit display by changing its co-ordinates, also called manipulating the X and Y variables of the co-ordinates.

In our first program we used the simple event command **Forever** which means the code will run until the microcontroller is turned off or restarted.

This project uses the information from the previous project and what we learnt about the gyroscope sensor and movement of the micro:bit in the X and Y axis in Lesson 6. So when we move the micro:bit the LED sprite moves in that direction.

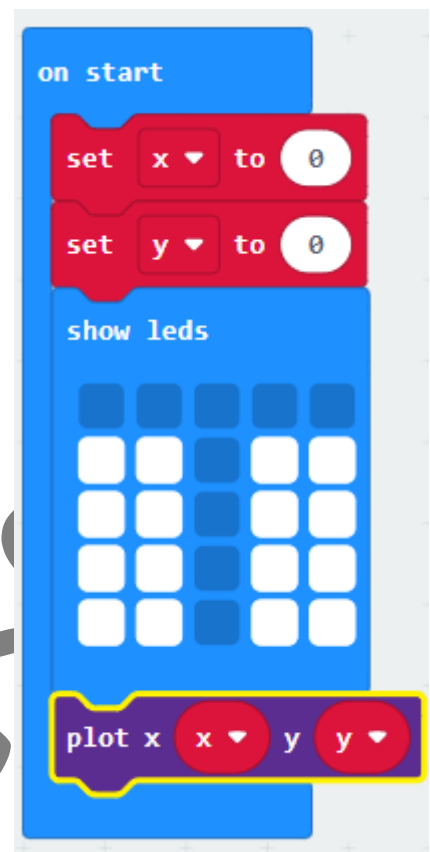
Create a new project in Makecode. Name your project **Lesson 7 coordinates tilt**. We are going to use what we learnt in the previous lesson about the gyroscope.

First we need to understand how we can move LED sprite around the display.

1. Drag an **on start** event block from the **Basic** command list to the code building area. The on start command is an example of an **event** block. This means that the commands inside the block will only be excited once the event has happened. In this case the event is when the micro:bit starts up.
2. Next create two variables called **x** and **y**. Drag from the **variable** command list the **set x to** and drag another and change to **set y to**. Both values need to remain at 0 as the starting coordinate for the sprite will be (0,0). You would change this if you wanted the starting point to be elsewhere on the display.



- From the **Basic** command list drag a **show LEDs** block and place it in the on start block. Click on the block to create a pathway and obstacle for the sprite to follow and avoid. Next from the LED command block list drag a **plot x... y** command block , then drag the **x and y variables** to replace the corresponding zeroes. Your code should look like this, and the LED at co-ordinate (0,0) in the micro:bit simulator should be illuminated.

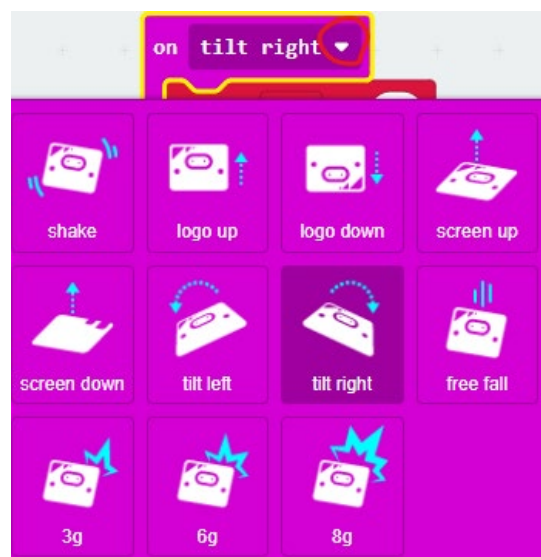


- We now need to create a way to change the x and y values so that the sprite will move left and right and up and down. We will use the gyroscope sensor to achieve this.

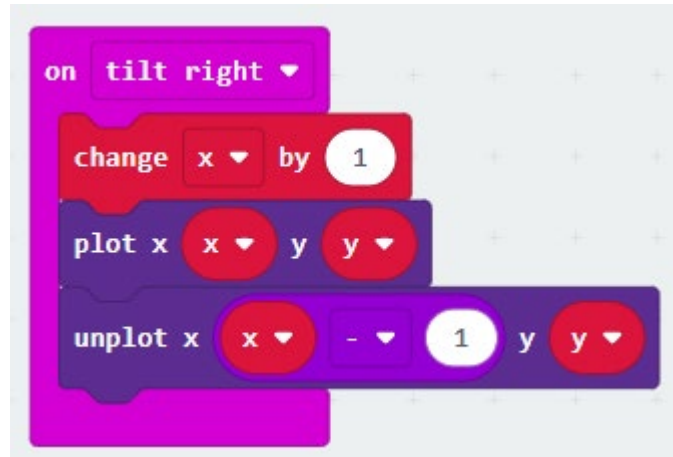
- We will use another **event** block , which will execute the code inside it based on the type and direction of movement on the micro:bit

- First let's make the sprite move left and right. From the **Input** command list drag into the code building area the **event** block **on shake**. Click the little white arrow to get further options.

- Select **tilt right** . Next drag into this block a **change X by** block from the Variables command list. This will change the value of the X co-ordinate every time we tilt the micro:bit to the right.



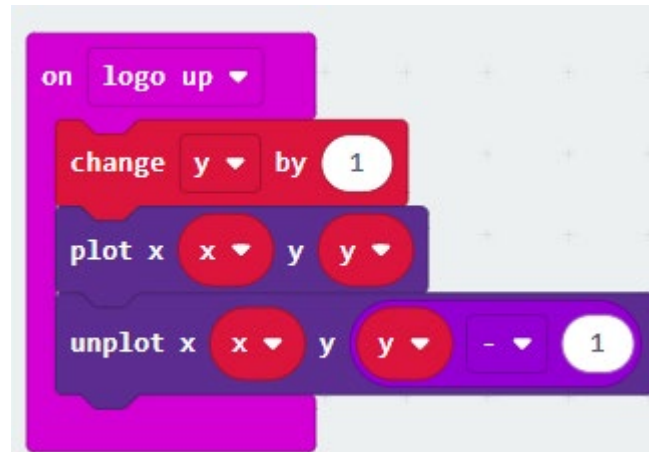
8. We now need to plot the new co-ordinate value for the LED sprite. Drag a **plot x...y** command from the **LED** command list and drag the variables **X** and **Y** into this block. To make the animation run smoothly, we need to unplot the previous co-ordinate value. To do this drag an **unplot x...y** block into your code. From the **Math** command drag a calculation command into the **x** coordinate space. Drag the **X** variable into this block and change the calculation to - (minus) and the value to 1. This will unplot the previous coordinate. Giving the impression that the LED sprite is moving. This block of code should look like this.



9. Repeat the above section, but do the opposite. So choose **tilt Left** in the **Event** block. We need to change the value of **x by -1** and we need to **unplot x to x+1**.
10. Moving your mouse left and right over the micro:bit simulator will allow you to see your code working. You can also download it to your micro:bit to test further.
11. Now we can control the movement in the X axis i.e. left and right. We need to be able to control the movement in the Y axis i.e. up and down.
12. We will repeat the above steps. With some changes. From the **Input** command list drag into the code building area an **event** command **on shake** command. Click the little white arrow to get further options.
13. This time select **Logo Up**. Next from the drag into this block a **change Y by** block from the Variables command list. This will change the value of the Y co-ordinate every time we tilt the micro:bit upwards.
14. We now need to plot the new co-ordinate value for the LED sprite. Drag a **plot x...y** command from the **LED** command list and drag the variables **X**



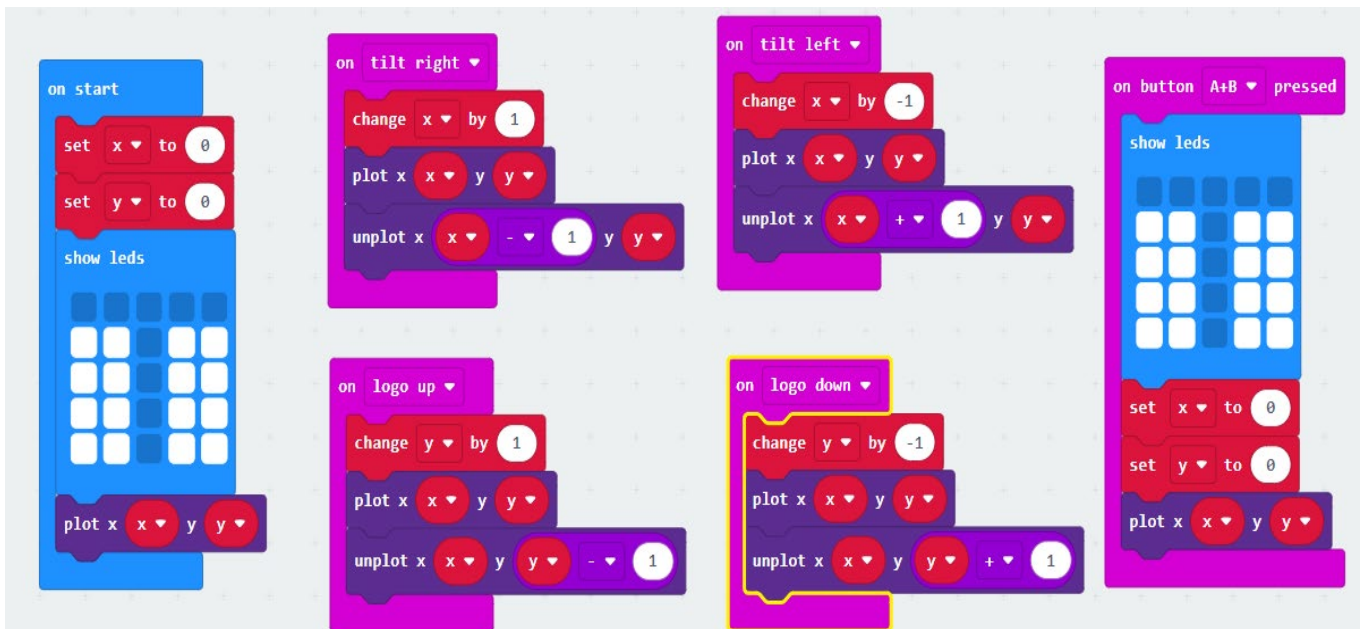
and **Y** into this block. To make the animation run smoothly, we need to unplot the previous co-ordinate value. To do this drag a **unplot x...y** block into your code. From the **Math** command drag a calculation command into the **Y** coordinate space. Drag the **Y** variable into this block and change the calculation to - (minus) and the value to 1. This will unplot the previous co-ordinate.



This is exactly what we did previously, but this time we are manipulating the **Y** variable.

15. Repeat the above section, but do the opposite. So choose **Logo down** we need to change the value of **y by -1** and we need to **unplot y to y+1**.
16. Moving your mouse up and down over the micro:bit simulator will allow you to see your code working. You can also download it to your micro:bit to test further.
17. We can control the movement in the X axis .i.e. left and right, and in the Y axis i.e. up and down. By tilting the micro:bit in different directions. The LED sprite will follow the movements.  
Finally drag an **on button....pressed** block from the **Input** command. Change it to **A+B** and copy the code from the **on start block** this will allow you to reset your display by pressing the A and B buttons together. Your final code should look like this.





Download your code to your micro:bit to try it out.



### Practice the Concept:

1. Create a flowchart for the previous program. How will it show that the code has multiple start points?
2. Can you use other event blocks from the INPUT commands list in your program to control the sprite? Do they all work? Can you redesign your program?

## REVIEW EXERCISE

1. List in order the typical activities in program creation:

1.	
2.	
3.	
4.	

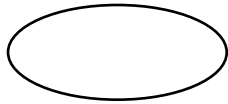

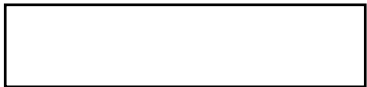
2. Which one of the following is **not** a way to analyse a task?

- a. Asking questions.
- b. Seek out available resources.
- c. Seek out possible solutions.
- d. Writing Instructions.

3. Which one of the following is used to present a solution graphically?

- a. A program.
- b. Flowchart.
- c. Questions.
- d. Instructions.

4. Match the following symbols used in a flowchart to their names given in the table:

a	
b	
c	

Start or Stop	
Process	
Connecting line	

5. Give an example of a control that can be added to a program so that we can observe clearly the different motions of a Sprite as they are carried out in sequence.

---

Sample Lesson