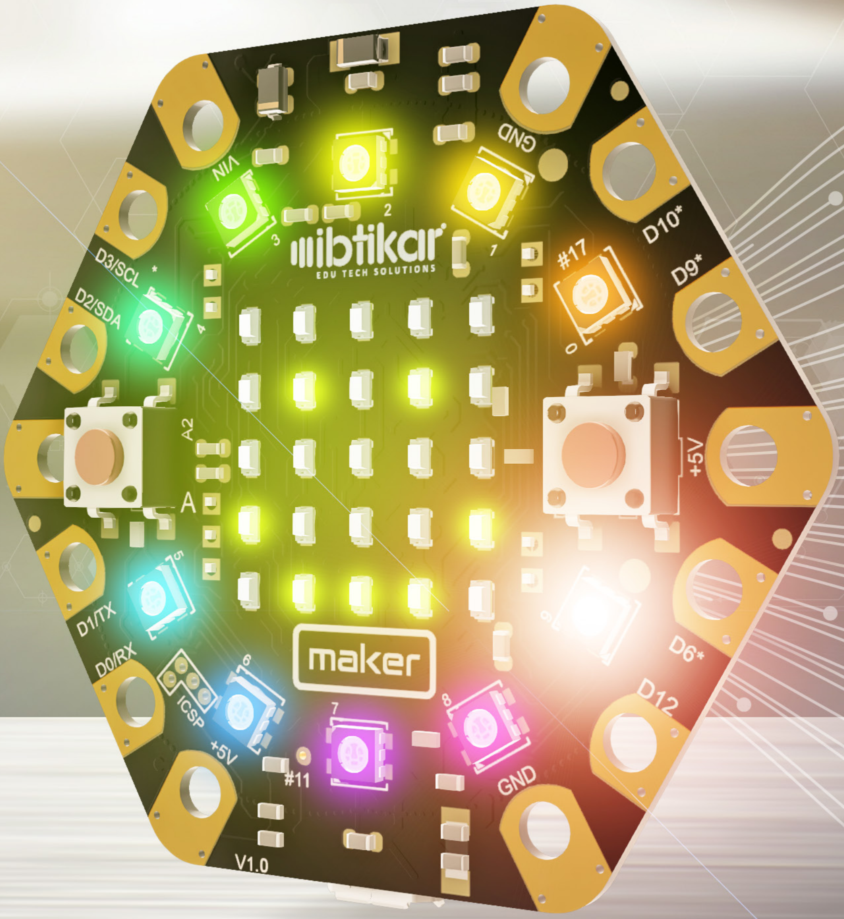
ibtikar®
EDU TECH SOLUTIONS

maker™ GUIDE

FOR ADVANCED USERS

ARDUINO™

python™



Maker Guide

For Advanced Users

Credits

All pictures showing the Ibtikar Maker board interfaced with other modules are created using Fritzing. Fritzing is an open-source hardware initiative that makes electronics accessible as a creative material for anyone. Ibtikar Maker has developed their own library but also uses material from libraries owned by others.

Authored by Mohannad Takrouri

ISBN: 978-9948-38-696-4

While the advice and information in this book is believed to be true and accurate at the date of publication, neither the author nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Ibtikar Edu Tech Solutions, and its dealers and distributors will not be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Ibtikar Edu Tech Solutions has endeavored to provide trademark information of the companies and products mentioned in this book by the appropriate use of capitals. However, Ibtikar Edu Tech Solutions cannot guarantee the accuracy of this information.

Contents

Getting Started with the Ibtikar Maker Board

Introduction to Microcontrollers

Definition

A **microcontroller** is a small computer on a single chip. Unlike your personal computer which can do different tasks at the same time, the microcontroller can only do one task at a time.

Did You Know?

The word **micro** means very small.

A microcontroller is usually **embedded** inside a system. You can think of it as the brain of a system.

If the system has an input unit for sensing the environment, a control unit for processing the received signals and an output unit for sending out information or to control an output device, then it can be called an **embedded system**.

Where Do You Find Embedded Systems?

Embedded systems control many devices in common use today. You can find microcontrollers in washing machines, microwave ovens, cars, elevators and any smart machine.

An example of an embedded system is the air conditioner unit that you use in your home and in your car. The air conditioner cools down the air by removing its heat. The main controller of this unit is an embedded computer system that senses the temperature of the room or car, compares it with the desired temperature you choose, then controls the cooling process to adjust the temperature.

One of the most well known microcontrollers, is the Arduino board. The Arduino microcontroller is an easy-to-use board with many ports for input and output that can be programmed to do certain jobs.

Programming the Microcontroller

What Is a Program?

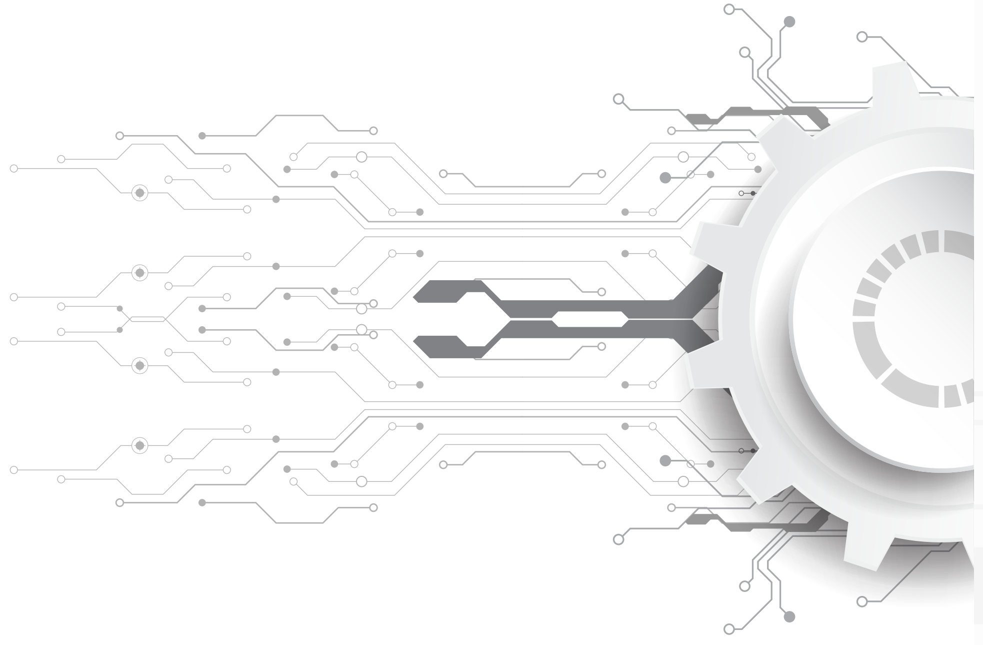
A **program** is a set of commands and instructions that can be downloaded to a computer or a microcontroller to do a specific task.

Computers have their own language which is based on two numbers, 0 and 1. It is very difficult for us as humans to write in this language. Because of this, programmers have created high level languages which allow us to write computer programs in a language that is like the language we understand. A **compiler** is then used to convert the program into the computer language. This makes it easy for us to modify and understand the programs that are written.

Types of Programming

Microcontrollers can be programmed using either a visual programming interface or a text-based one. In visual programming, you can use graphical elements to create programs. You can also drag and drop program elements, click, use menus, forms, dialogue boxes and so on. Behind each block of your program, there are tens or even hundreds of lines of code. This type of programming helps new users to easily understand programming.

The other type of programming is text-based. In text-based programming, you must understand the language syntax and rules. You can cut, copy, and paste your code which gives you more flexibility compared to dragging and dropping one block at a time.



The Ibtikar Maker Board

The Hardware

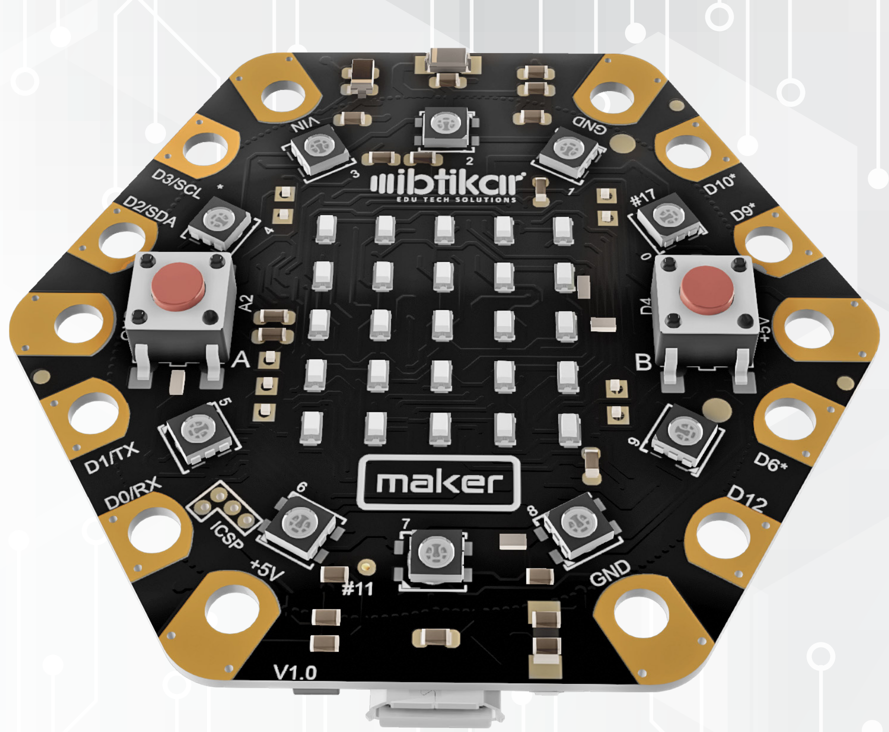
Size

The Ibtikar Maker board is a small, hexagon-shaped microcontroller board, designed and developed in the United Arab Emirates. The following two figures show the front and the back of the board.

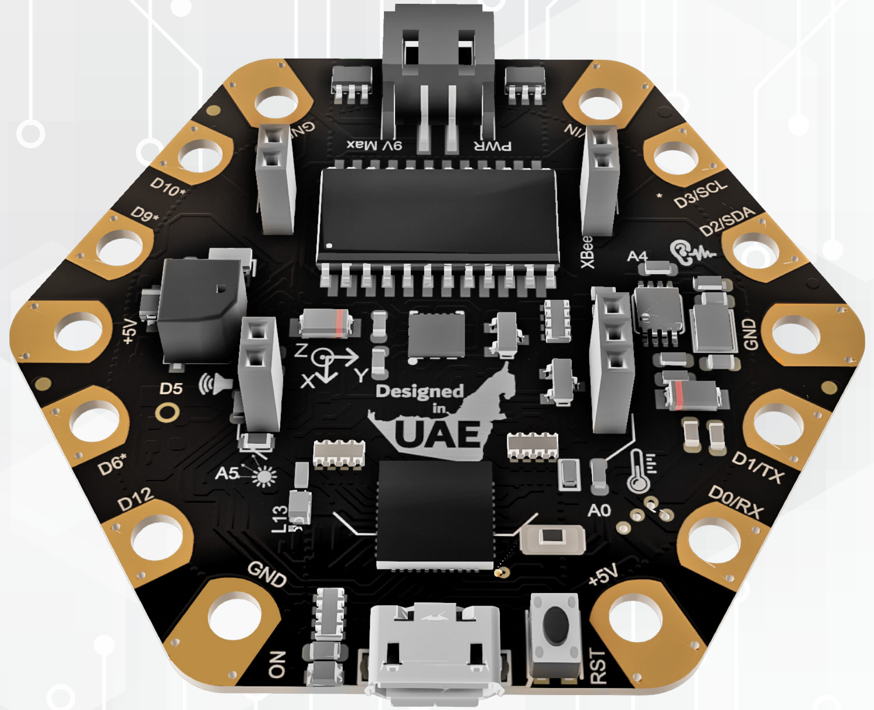
Features

The Ibtikar Maker board has the following features:

- 25 LEDs in grid formation
- 10 NeoPixel RGB LEDs
- 2 push buttons (left and right)
- temperature sensor
- ambient light sensor
- sound sensor
- mini speaker (magnetic buzzer)
- triple axis accelerometer
- 8 input/output pins of which 7 of them can act as capacitive touch inputs
- XBEE Socket to allow for Wi-Fi or Bluetooth expansion
- Arduino compatible microcontroller (ATmega32u4, 16 MHz crystal)



Front



Back

Front and Back

Before you start using the Ibtikar Maker board, it is important for you to know where each component is, and its purpose.

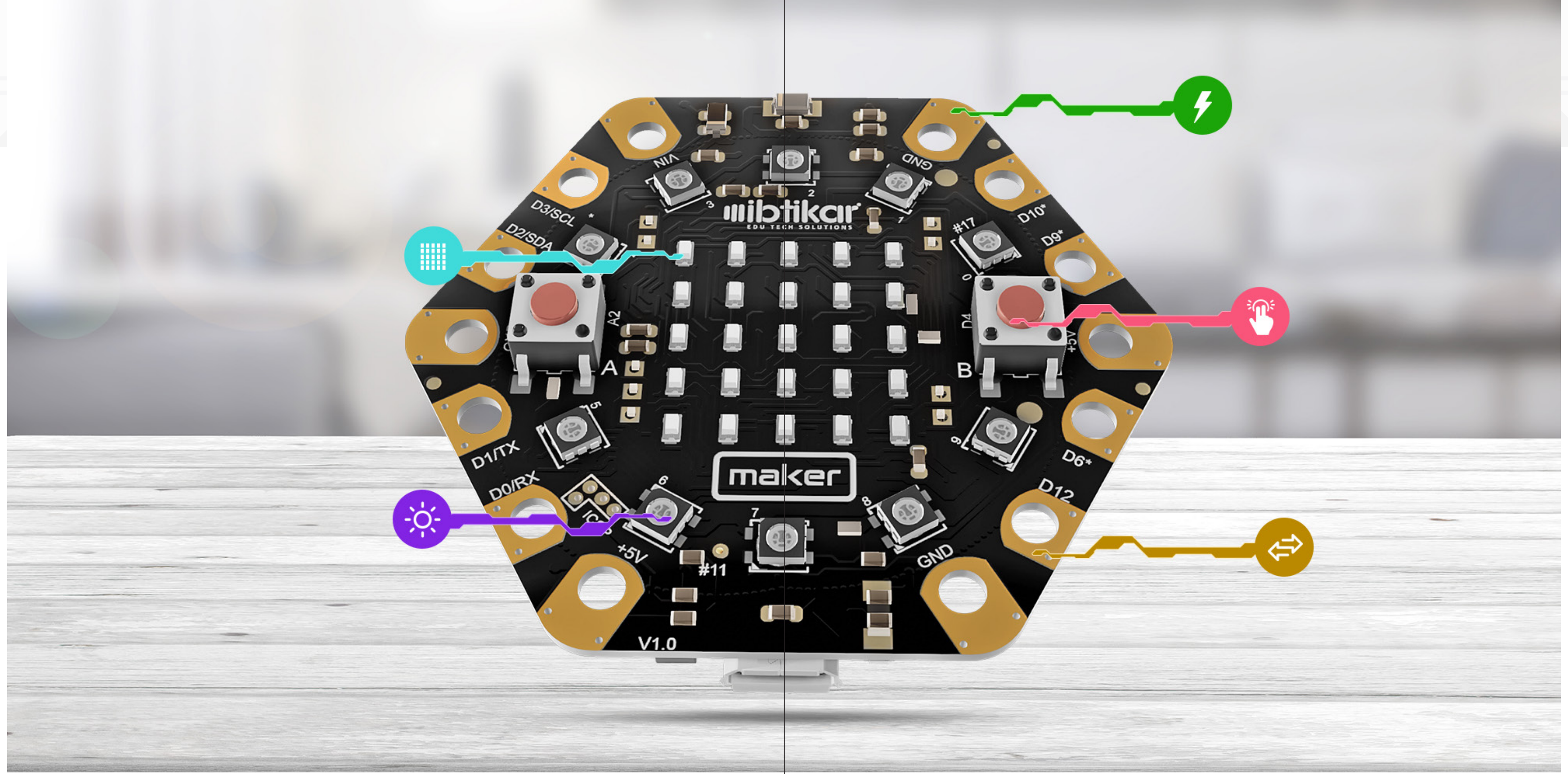
On the front side, there are the following components:

1. LED Grid

There are 25 LEDs which you can turn ON or OFF. Each LED can be controlled individually, which allows you to create patterns. For example, you can show letters, numbers, text, emojis or any other pattern you like.

Did you Know?

LED stands for Light Emitting Diode. An LED is not the same as the original light bulb invented by Thomas Edison. It has no filament or special wire that produces light when electricity passes through it. LEDs use advanced semiconductor material, the same material found inside computer chips. LEDs are better than traditional light bulbs. They last longer, are more robust and use much less power.



2. NeoPixels

The Ibtikar Maker has 10 RGB LEDs. **RGB** stands for **Red**, **Green** and **Blue** which are the basic colours. Unlike the 25 LEDs which cannot change their colour, the RGB LEDs can be programmed to show any colour by combining the three basic colours. Think of each pixel as three small LEDs combined, with each of these LEDs being a different colour (**Red**, **Green** and **Blue**).

3. Push Buttons

On the front of the board, there are two push buttons labelled, A and B. Button A is to the left of the board, while button B is to the right. These two buttons are user input components on your board. This means that you can program your board to detect when they are pressed or released. Pressed means 1 and released means 0.

4. Power Pads

These pads can supply power to external input or output modules. There are three **ground** pins, two **5V** pins and one **Vin** pin. The **Vin** pin has the same voltage as the one coming from the external battery connector.

5. Pin Pads

These pads can be interfaced with extra input and output modules allowing you to extend the capabilities of your Maker board. You can do this by adding another push button for example, or even a module like a motion or a flame sensor, both of which are not on the board. Some of the pins allow you to communicate between other boards or components. Others allow you to read and write analog and digital signals. These pins are explained in detail in the next section.

All these pads can be accessed either from the front or the back side of the board.

On the back side of the board, there are the following components:

1. Temperature Sensor

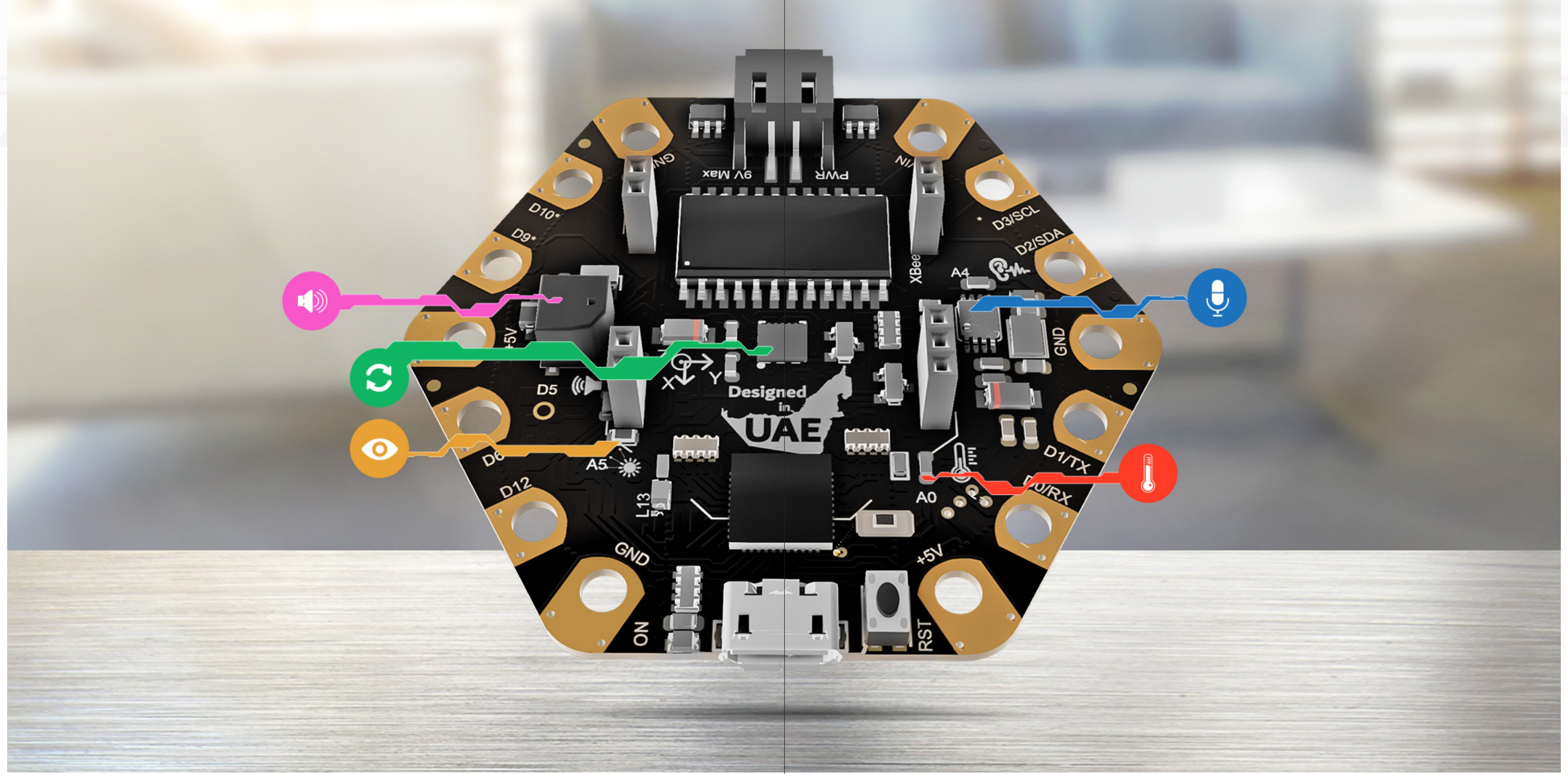
   This is an analog sensor which measures the board temperature.

2. Light Sensor

   This is an analog sensor which measures the ambient light in the environment. The analog light sensor gives values between 0 and 1023.

3. Buzzer (Mini Speaker)

   This is a magnetic buzzer which can play tones. By controlling the frequency going to the buzzer, you can generate different tones.



4. Sound Sensor

   This is an analog sound level sensor. You can use it to detect if there is a clap/sound near the board. The sensor reads values between 0 and 1023.

5. Triple-Axis Accelerometer

   This sensor is in the middle of the board. Accelerometers are used to measure acceleration, which is how fast something is speeding up or down. An accelerometer can measure static acceleration like gravity, which is useful in detecting tilt, like when your phone tilts.

Accelerometer can also detect dynamic acceleration: the sudden start or stop of an acceleration.

> Note
>
> The Ibtikar Maker accelerometer can sense the three axes (X, Y and Z). You can identify the positive and negative directions of each axis by looking at the small three-axis drawing next to the sensor.

6. XBEE Sockets

These sockets allow for Wi-Fi or Bluetooth expansion. This is useful if you want to wirelessly control your Maker or read from it. You can use your phone or tablet, for example, to connect to the Maker board.

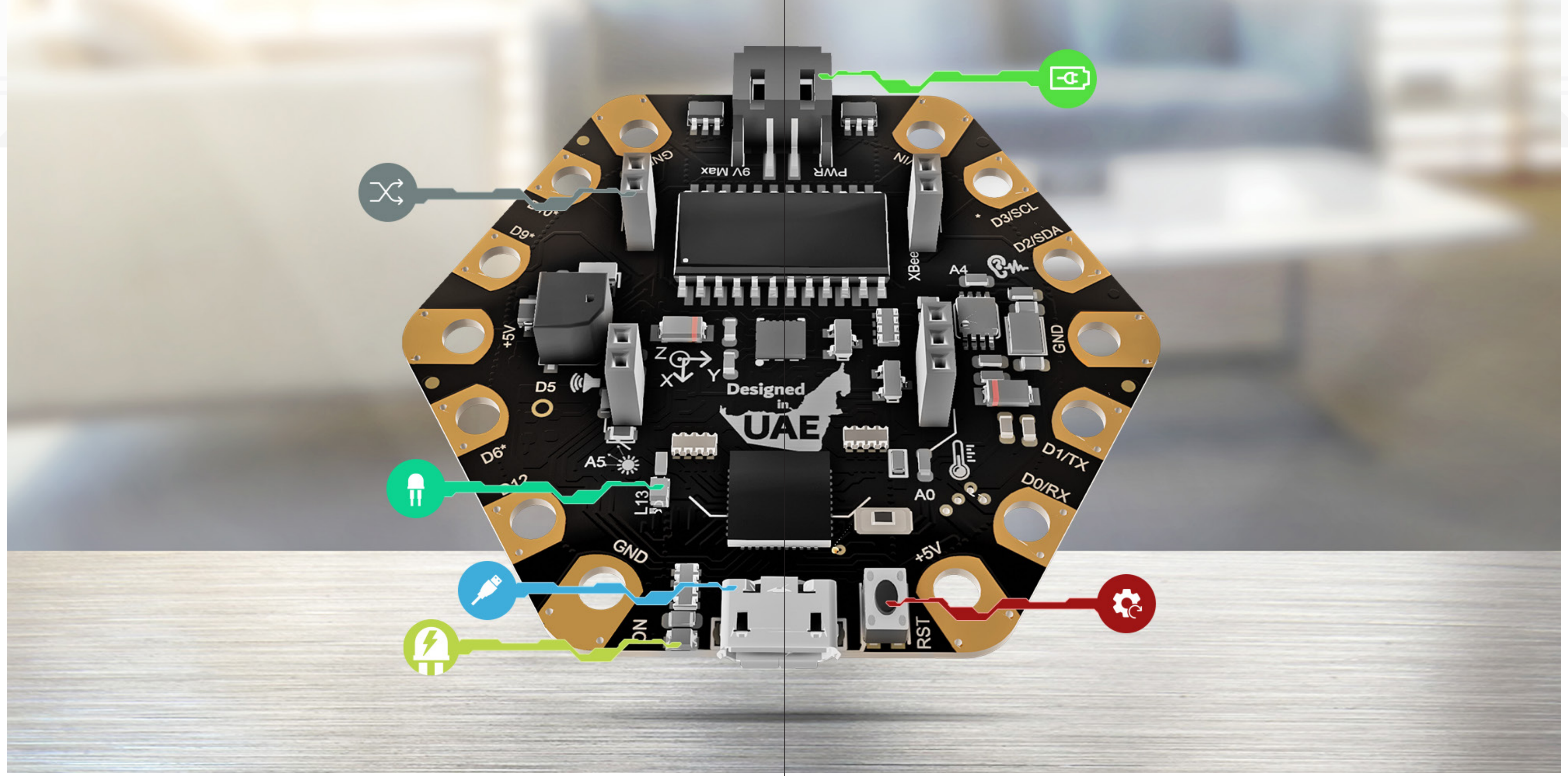
Note

The Wi-Fi/Bluetooth module is not included in the kit and can be purchased separately.

Beside each socket there are some fine, white lines drawn on the board. These lines help you to connect the Wi-Fi/Bluetooth module in the right orientation.

7. On-Board LED

This LED can be controlled by the user and is connected to pin 13.



8. Micro-USB Socket

This socket allows you to connect to your computer using a USB cable. You can then program your Maker board, send and receive data and power the board.

9. Power LED

This LED turns ON when your Maker is powered.

10. Reset Button

This button restarts or resets the board.

11. Battery Connecter

This connecter serves as an external power supply source. You use it to power your board when the USB cable is not connected. This can be extremely useful when you do not want your project to be attached to a computer all the time.

Note

The supply voltage should be between 6V to 9V. Supplying the board with a voltage different to that recommended, may cause the board to malfunction or even damage it.

Pin Configuration

The Maker board has multiple pins. Some of these pins are external (i.e. on the outer edge of the Maker) and some are internal, like the two buttons. It is important to know what each pin is connected to or capable of, to give you full control of your Maker board.

External Pins:

Pin	Function	Digital (I/O)	Analog (I)	Serial	I2C	PWM	Interrupt
D0/RX	Touch	0		RX			INT2
D1/TX		1		TX			INT3
D2	Touch	2			SDA		INT1
D3	Touch	3			SCL	*	INT0
D6	Touch	6	A7			*	
D9	Touch	9	A9			*	
D10	Touch	10	A10			*	
D12	Touch	12	A11				

Some of these pins are used for special applications. The table below shows the name of each category and its usage:

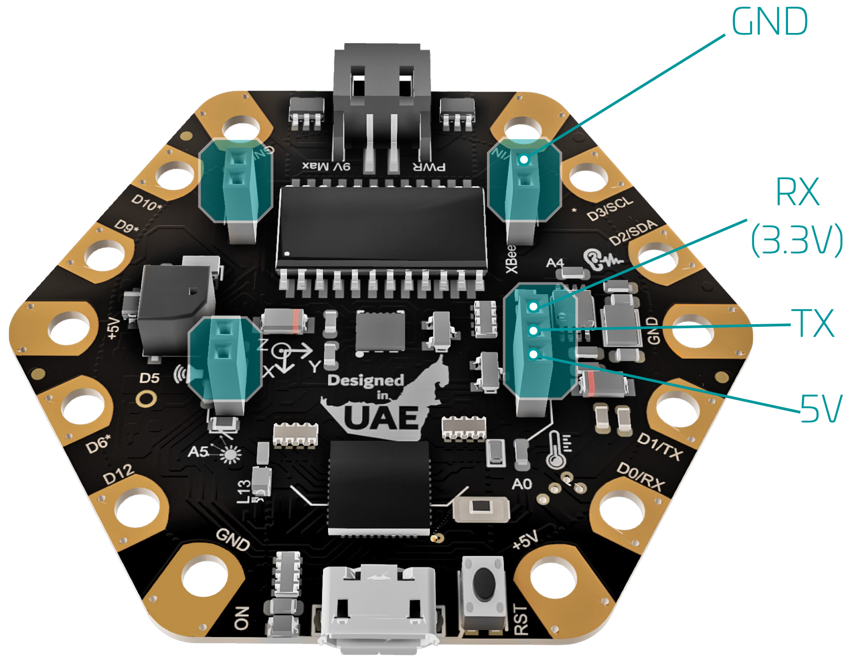
Pin Category	Usage
I2C	Inter-Integrated Circuit (I2C), or Two-Wire Interface. Using this protocol, many I2C devices can be connected on the same two wires. These pins are D2 and D3.
PWM	Pulse Width Modulation (PWM) is a technique used to gain analog results with digital means by switching signals between HIGH and LOW, thousands of times a second. Using this, you can simulate voltages between 0 and 5 volts.
Interrupt	An Interrupt makes the processor respond quickly to important events. When a certain signal is detected, the Interrupt interrupts whatever the processor is doing, and executes some code. Once that code has finished, the processor goes back to where it was originally.

Internal Pins:

Module	V0.0 Pins	V1.0 Pins
Bluetooth RX Socket	D1 (TX)	D1 (TX)
Bluetooth TX Socket	D0 (RX)	D0 (RX)
Left Button (A)	A1	A2
Right Button (B)	D4	D4
Buzzer	D5	D5
Temperature Sensor	A0	A0
Sound Sensor	A4	A4
Light Sensor	A5	A5
Built-in LED	D13	D13
NeoPixels	D17	D17
Touch Pads	D0, D2, D3, D6, D9, D10, D12	D0, D2, D3, D6, D9, D10, D12
	Control Pin: D30	Control Pin: D30
LED Grid	Pin 1: SPI-MISO	Pin 1: SPI-MOSI
	Pin 2: SPI-MOSI	Pin 2: A1
	Pin 3: SPI-SCK	Pin 3: SPI-SCK
Accelerometer	I2C-SDA (D2)	SPI-MISO
	I2C-SCL (D3)	SPI-MOSI
		SPI-SCK
	Interrupt INT (D7)	Interrupt INT (D7)
	Chip Select CS (D8)	Chip Select CS (D8)

Bluetooth/Wi-Fi Socket:

The pinout for the Bluetooth/Wi-Fi socket is shown in the figure. The rest of the pins are not connected and are used for the module stability when attached.



Maker Version

At the time of writing this guide, Maker has gone from version V0.0 to V1.0 with some improvements/modifications. Based on the Maker version you have, make sure you read this section. All changes are summarised below.

Sound Sensor

Maker V0.0 has a special configuration in that, instead of the value increasing when the sound sensor detects noise, the value decreases. This has now changed in Maker V1.0. When the sound sensor detects noise, the value increases.

Accelerometer

If you are using the accelerometer sensor in Maker V0.0, you should not touch or use touch pads 2 and 3 because they are sharing the same data lines (I2C). You can still use the other touch pads. Also, the sensor usually needs an offset to calibrate its values. The sensor gives raw output which is not mapped to the acceleration of gravity. The X axis points to the right while the Y axis points to the top.

Maker V1.0 has a new accelerometer where its communication lines are different from those on pin 3 and 4. The accelerometer is now using SPI pins. The sensor is also calibrated which means that you do not need to add an offset to the axis you want to read. The output of the sensor is in the unit of g (acceleration of gravity). The X axis points down while the Y axis points to the right.

Pin 3 and Pin 12 Silkscreen

In Maker V0.0, pin 12 has an asterisk * next to it indicating that it is a pulse width modulation (PWM) pin while pin 3 does not. In Maker V1.0, this has been fixed. Pin 3 has the asterisk while pin 12 does not.

Power Regulators

The power regulators on the V1.0 board are changed to allow for a higher current value from the external power supply.

Slight Changes in Wiring

Button A is now connected to pin A2 in V1.0 while it is connected to pin A1 in V0.0.

The LED Grid wiring changed as explained in the Pin Configuration table.

Power Options

There are several ways to power the Maker board such as, the USB cable, power connector, or the Vin pin. Choosing the power option depends on the application and whether you need to run the Maker board in a stationary place or within a movable machine.

USB Connector

The micro USB cable allows you to connect your Maker to the computer, which powers the board and allows you to send and receive data. The USB cable looks like the following:

External Power Supply

The other power option is to use an external supply between 5V to 9V. Based on which kit you get, you may have a 9V battery holder. The battery holder has an ON/OFF switch to make it easy to power the Maker, without the need to connect and disconnect the battery each time.

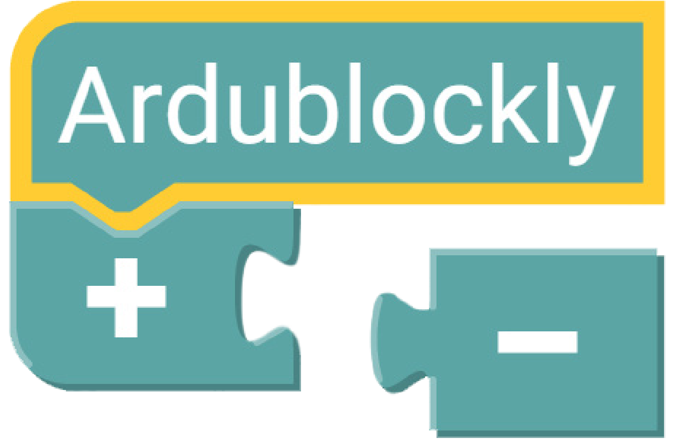
The Software

Arduino IDE

The Arduino Integrated Development Environment (Arduino IDE) allows you to write and upload codes to your Ibtikar Maker, using a text editor. This open source interface contains a message area, a toolbar with buttons to verify and upload your code and other common functions like create and save files. You can cut, copy, and paste your code.

Ardublockly

Ardublockly is a visual programming editor for Arduino. This interface is based on Google's Blockly. In visual programming, you can use graphical elements to create programs. You can also drag and drop program elements, click, use menus, forms, dialogue boxes and so on. Behind each block of your program, there are tens or even hundreds of lines of code. This type of programming helps new users to easily understand programming.



Python

Python is a widely used, high-level programming language for general-purpose programming, which was first released in 1991. It has a design philosophy which emphasises code readability, and a syntax (a set of rules) which allows programmers to show concepts in fewer lines of code, compared to other languages.

Python can be used to connect serially to your Maker board which allows you to use all the capabilities of Python installed on your computer, using a Python library called PySerial. Unlike Arduino IDE and Ardublockly which install the code, you write to Maker directly; PySerial allows you to send and receive data between your computer and the Maker.

To make this process easy, you can use the Ibtikar Serial (**IBSerial**) library which uses the pySerial with all the Maker functions and many Arduino commands implemented in Python. This library is intended to mirror the Maker Arduino functions and use them in Python.

In this manual, we will only be focusing on the Arduino programming environment with a brief introduction to Python. If you are interested to know more about using the visual interface Ardublockly, you can visit the Maker portal which contains the Maker-Ardublockly guide with other supporting activities and videos.

Creating Your First Program

Arduino Interface

Before you start programming your Maker for the first time, you must set it up. You need the Maker board, a USB cable, and a computer.

First, install the Arduino IDE from the Arduino official website. Choose the one that suits your machine and operating system. This software allows you to connect with the Maker board, write a code using the Arduino language, verify it, and then upload it to the Maker board.

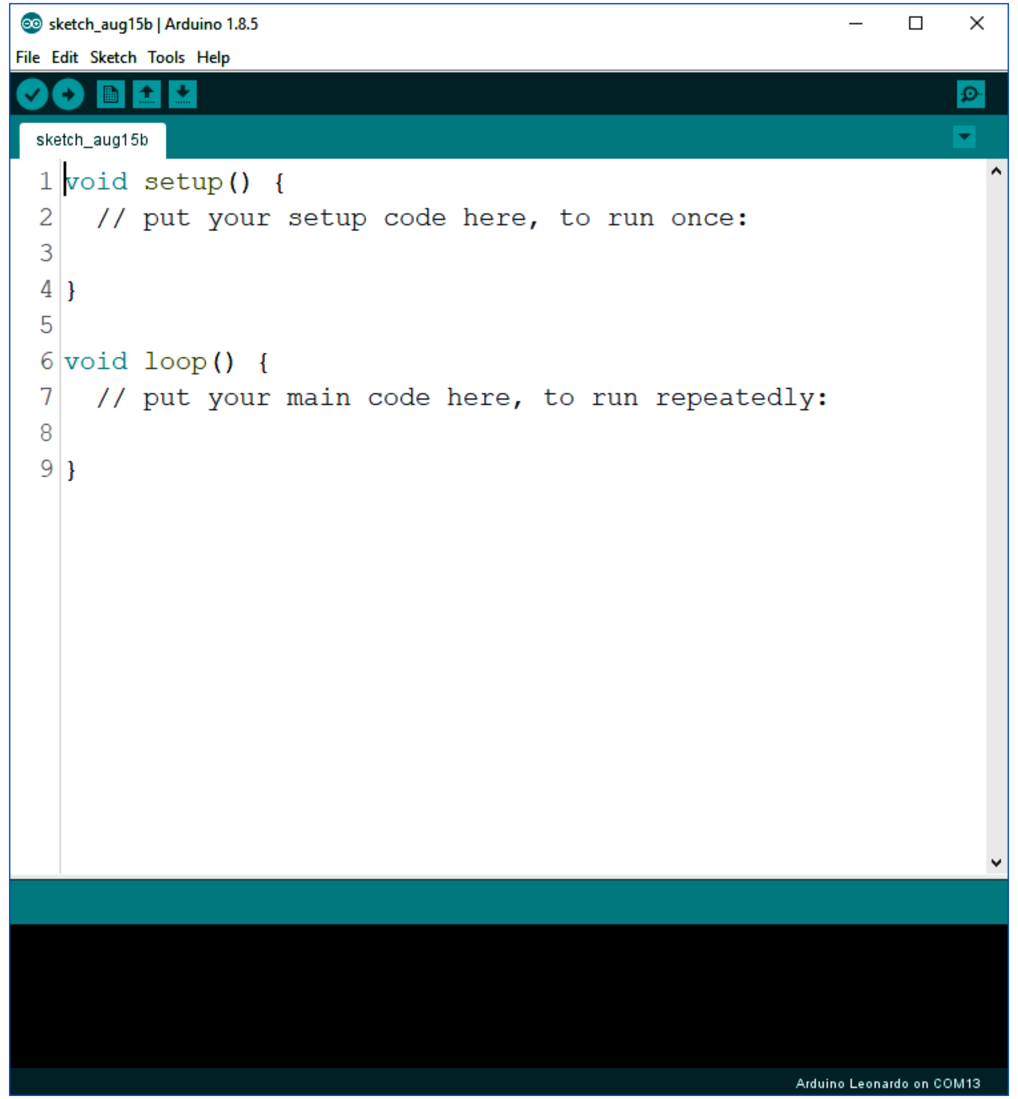
Note

If you have used Ardublockly, the visual programming interface for Maker, then you already have the Arduino IDE installed.

Connect the Maker board to the computer using the USB cable. Insert one end of the USB cable into the USB connecter and the other end into a USB socket on your computer. The power LED will turn ON.

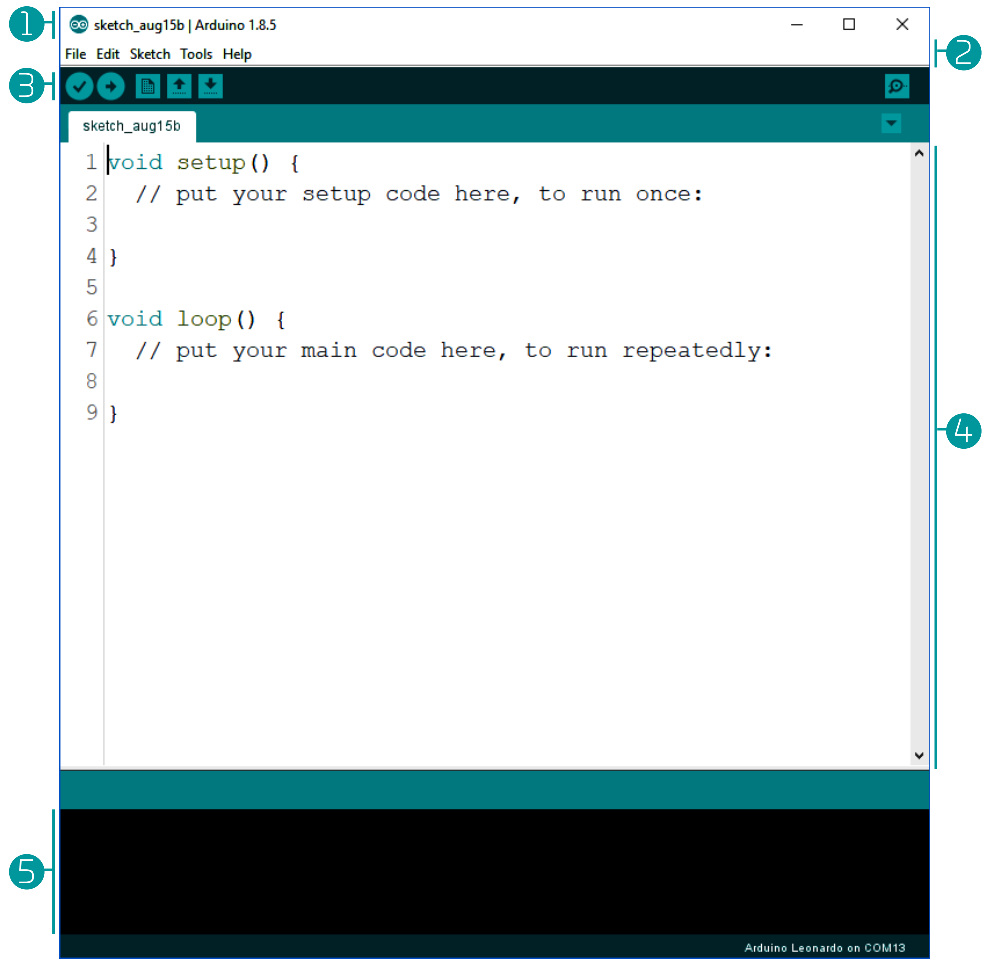
A message might pop up saying that the "*Device driver software was not successfully installed*". This means that the Maker board and your computer cannot see each other. For your computer to recognise the Maker board, it needs a proper introduction which is known as **Driver**. More details can be found in *Appendix 1: Arduino Driver Installation*.

Launch the Arduino IDE by double-clicking on the Arduino icon on the desktop and wait until the software starts. You should see the following window:


sketch_aug15b | Arduino 1.8.5
File Edit Sketch Tools Help
sketch_aug15b
Arduino Leonardo on COM13

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

The IDE is split into five parts as shown:


1
sketch_aug15b | Arduino 1.8.5
File Edit Sketch Tools Help
2
3
sketch_aug15b
4
5
Arduino Leonardo on COM13

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

These 5 sections are:

1. The **file information** section which has two titles: the **filename** and the **Arduino IDE version**.

2. The **menu bar** which holds five drop down menus: **File**, **Edit**, **Sketch**, **Tools**, and **Help**.

3. The **toolbar** consists of six buttons; five on the left side and one on the far right. These buttons give you easy access to the most frequently used functions. These buttons are:

   - The **Verify** button, the first button to the left, is used to check the code and make sure that it is free of mistakes.

   - The **Upload** button, the second button to the left, is responsible for uploading the code in the sketch file to the connected Maker board.

   - The **New** button, the button in the middle, will create a new blank sketch.

   - The **Open** button, the button with an arrow pointed up, will allow the user to open a stored sketch file.

   - The **Save** button, the button with an arrow pointed down, will allow the user to save the sketch file.

   - The **Serial Monitor**, the button on the far left, used to open the monitor display. The monitor will show all the serial data sent and received by the serial interface.

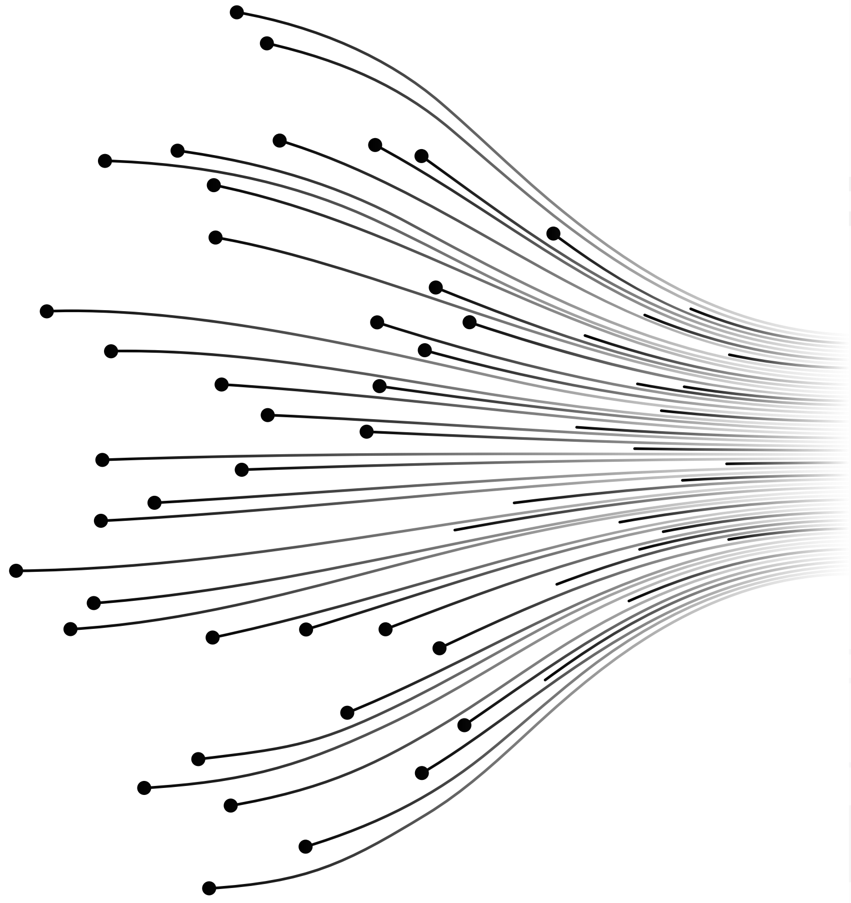
4. The **sketch** window with a tab on top, holding the sketch name. This is where you write your code.

5. The **message** window at the bottom of the program which shows the status and error messages to the user.

In the Arduino IDE window, you have two main sections where you write your code: the void setup() section and the void loop() section. The void setup() part of the code is executed when the device is initialised. It runs only once. It is used to declare the output and input of the device and other commands. The void loop() part of the code is executed after void setup(). Once initialised, it runs in a loop forever and it represents the actual job you need to do.

The Arduino files are called **sketches**, so the file name will start with the word sketch and be followed by the month and day. Arduino sketches are saved under the '*.ino*' extensions.

Now that you have installed the Arduino IDE, you are going to upload your first code.



Serial Monitor and Serial Plotter

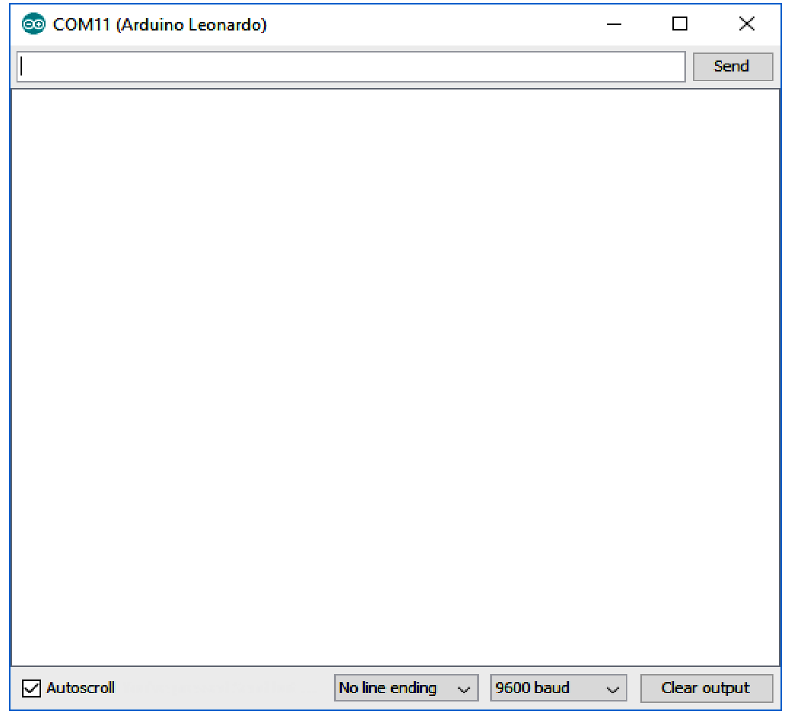
Serial Monitor

The Arduino IDE has a tool that can be very useful in debugging sketches or controlling Arduino using your computer's keyboard. This tool is called the Serial Monitor. It is a pop-up window that acts as a terminal allowing you to receive and send Serial Data. Serial Data is sent and received over two wires (Tx and Rx).

In the Maker board, the serial communication is interfaced with the USB allowing you to send and receive data directly using the USB. The Maker must be connected by USB to your computer to be able to use the Serial Monitor. The port which the Arduino board is connected to must also be selected from the same menu before you can use the tool.

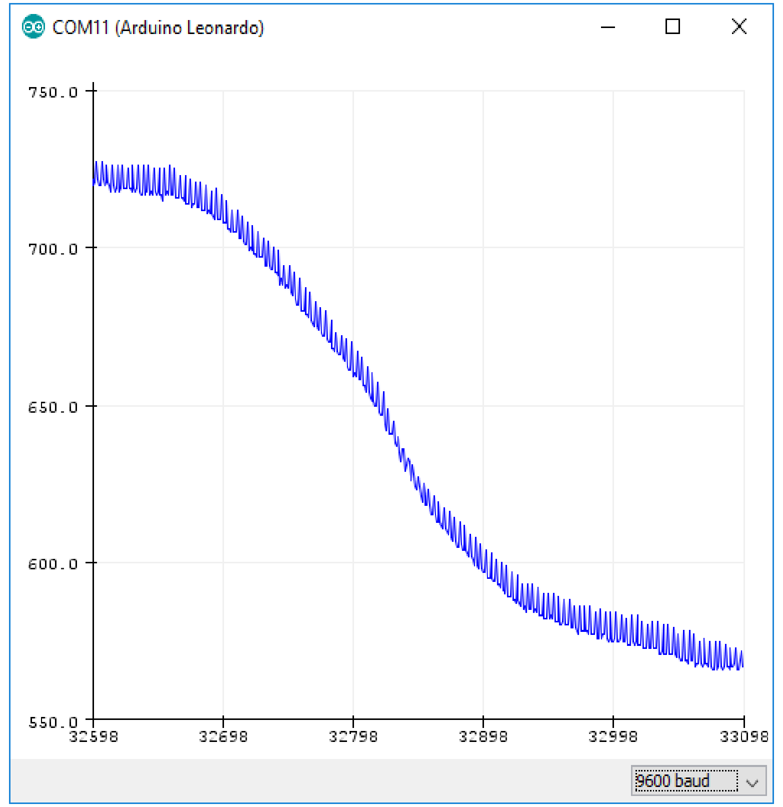
When you click on the Serial Monitor icon or when you open it from the **Tools** menu, the Serial Monitor window will pop up. This tool has:

- A small upper box. This is where you can type in characters and then hit Enter or click **Send**.
- A larger white area. This is where characters read by the Arduino board will be displayed.
- A pulldown that sets the **line ending** that will be sent to Arduino when you hit Enter or click **Send**. Based on your application, these options can be useful when you want to send data with or without a carriage return, for example.
- Another pulldown menu sets the Baud Rate for communications. It is important that this value matches the value you set in your sketch. Otherwise, characters will be unreadable. Some sketches or other applications may use a different Baud Rate.



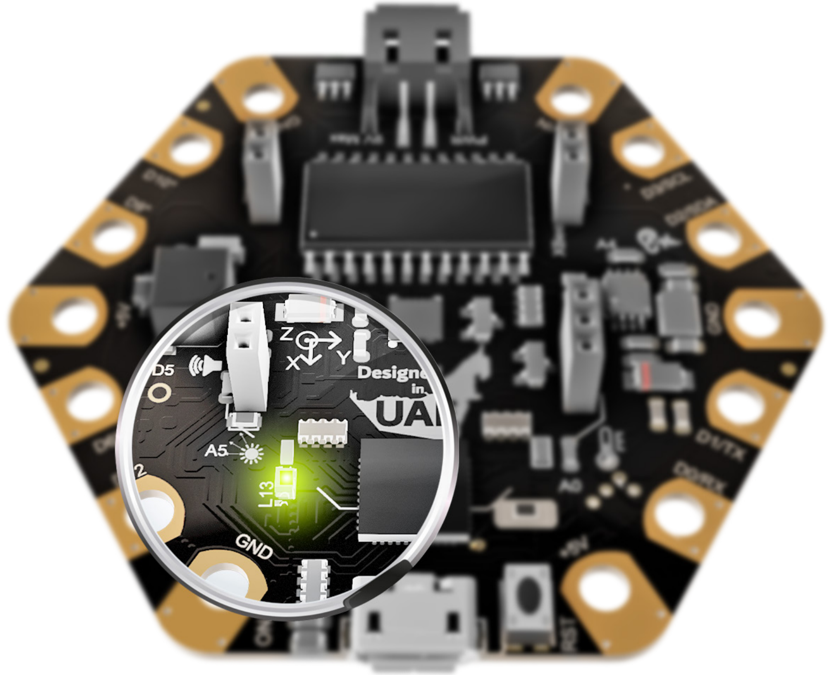
Serial Plotter

Another useful tool is the Serial Plotter. This tool is similar to the Serial Monitor but instead, it helps you visualise the data on the screen.



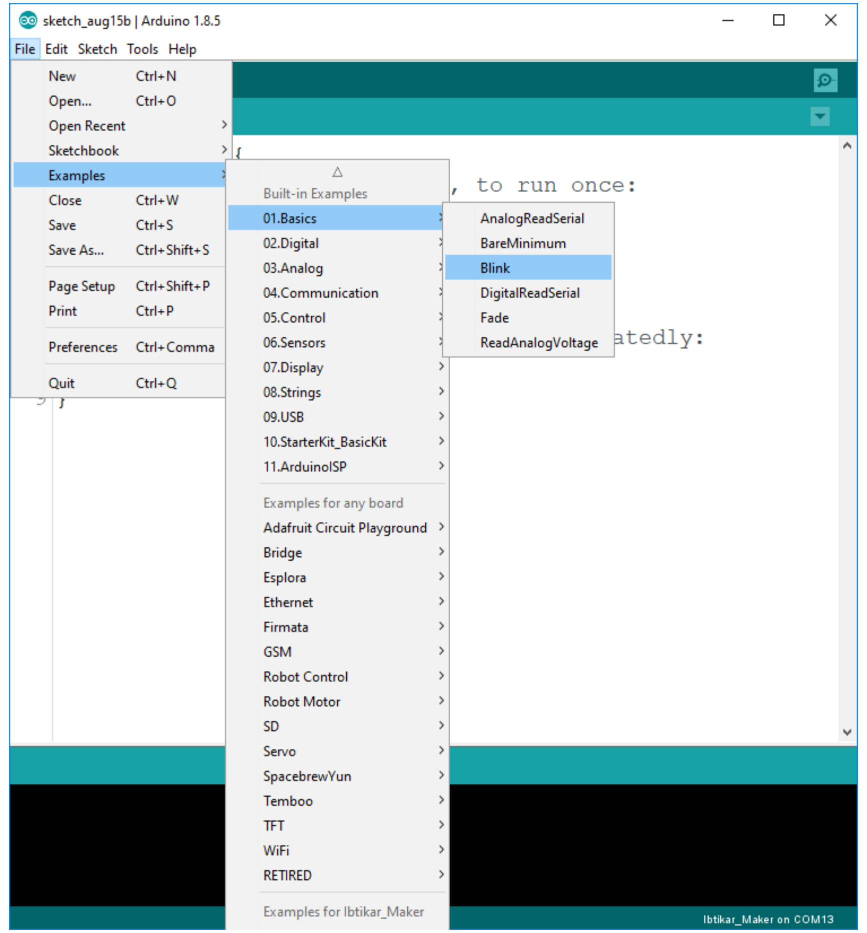
Your First Program

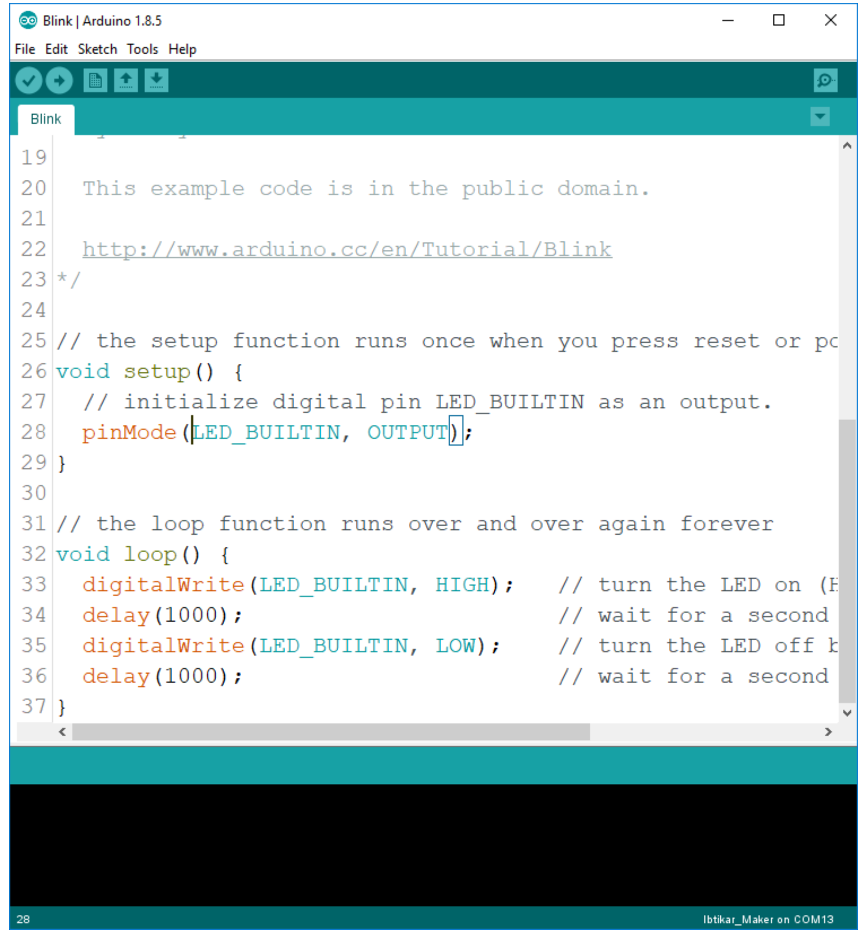
Here is a simple example of how you can use the Ibtikar Maker board and the IDE to do an action. This will let you test the board to see if it works normally. In this test, the built-in LED in the board will blink using a ready-made example code from the Arduino IDE. The built-in LED is called the **L13** LED, as shown in the figure.



Follow these steps:

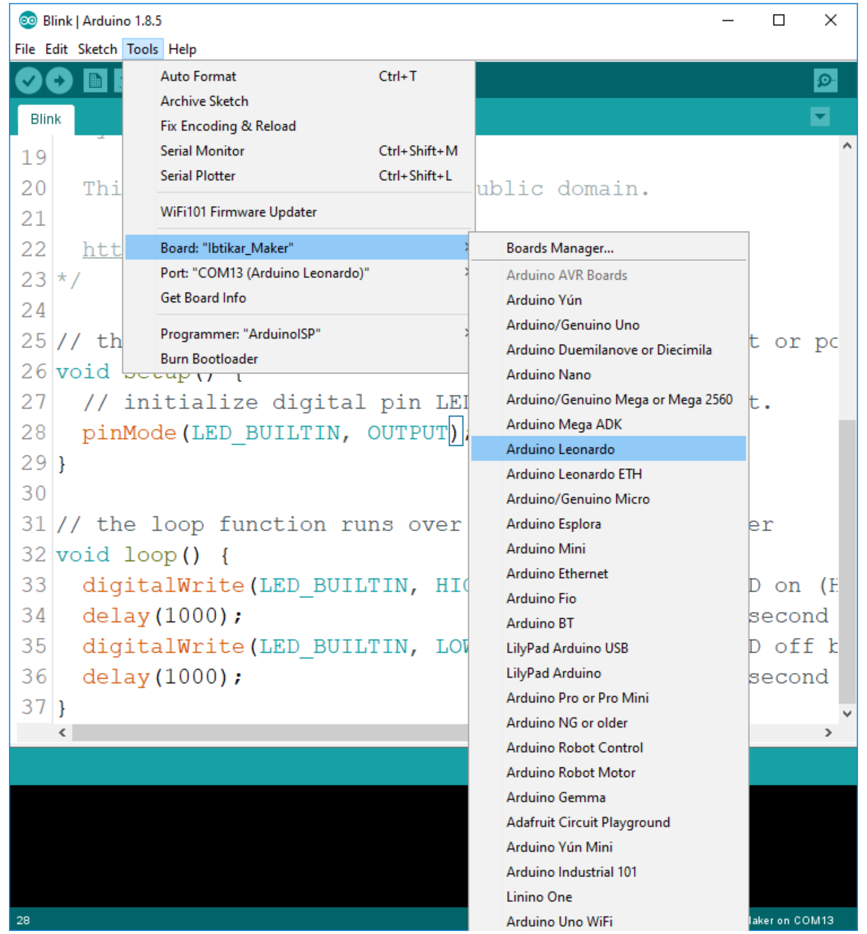
- Connect the board to your computer using the USB cable and your computer should recognise the board automatically
- Open the Arduino IDE
- Once it is open, go to File > Examples > 01.Basics > Blink. The Blink example will open as shown.



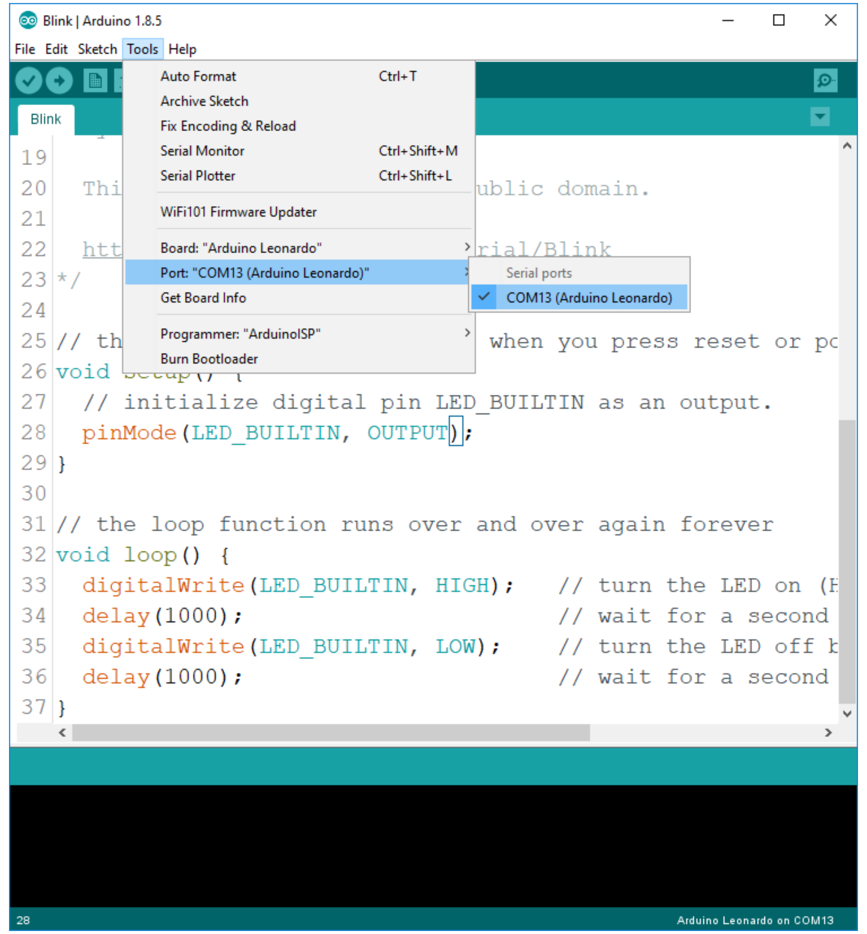


```

  This example code is in the public domain.

  http://www.arduino.cc/en/Tutorial/Blink
*/

// the setup function runs once when you press reset or pc
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (H
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off b
  delay(1000);                       // wait for a second
}
```

- Now, go to Tools and change the board to **Arduino Leonardo** and change the port whatever your board is connected to. In our case it is **COM13**.



Board: "Ibtikar_Maker"
Port: "COM13 (Arduino Leonardo)"



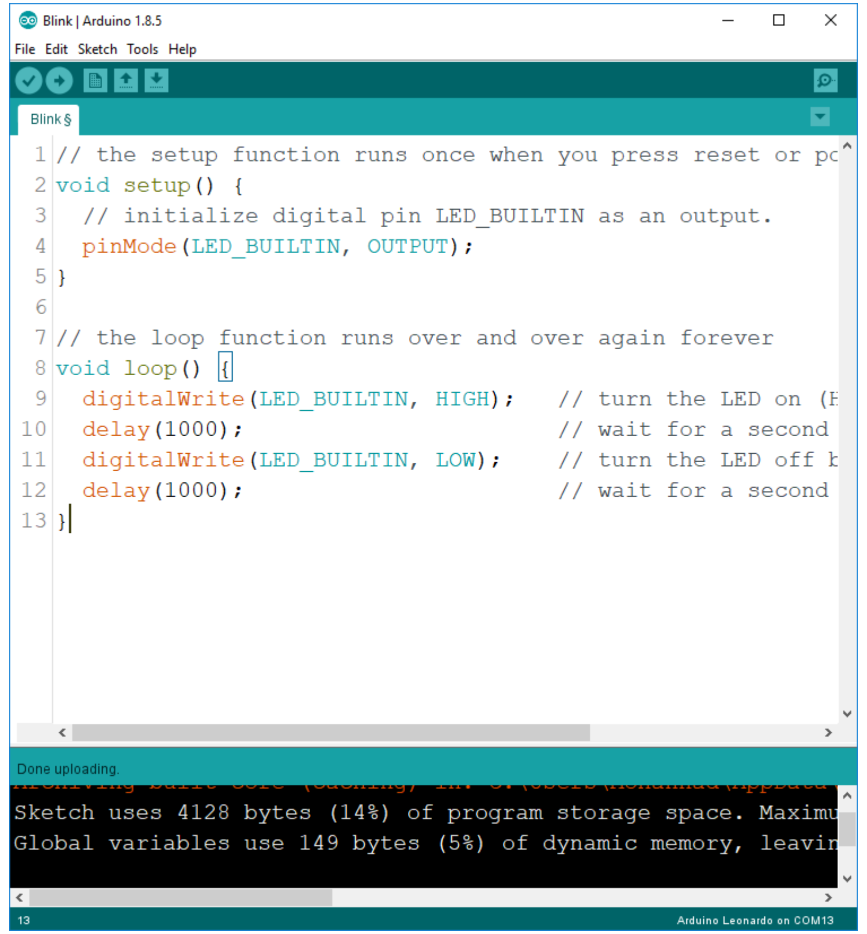
Board: "Arduino Leonardo"
Port: "COM13 (Arduino Leonardo)"
✓ COM13 (Arduino Leonardo)
Arduino Leonardo on COM13

```
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (H
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off b
  delay(1000);                       // wait for a second
}
```

- Click on Upload and wait a couple of seconds until the code is uploaded to your board. The message window should show *'Compiling sketch . . .'* and then it will change to *'Uploading'*. The RX and TX LEDs will start blinking as they show that the sketch file is being transmitted from the IDE to the Maker board.

- If you followed the steps correctly, you should see a message with '*Done uploading*' and the RX and TX LEDs will stop blinking.

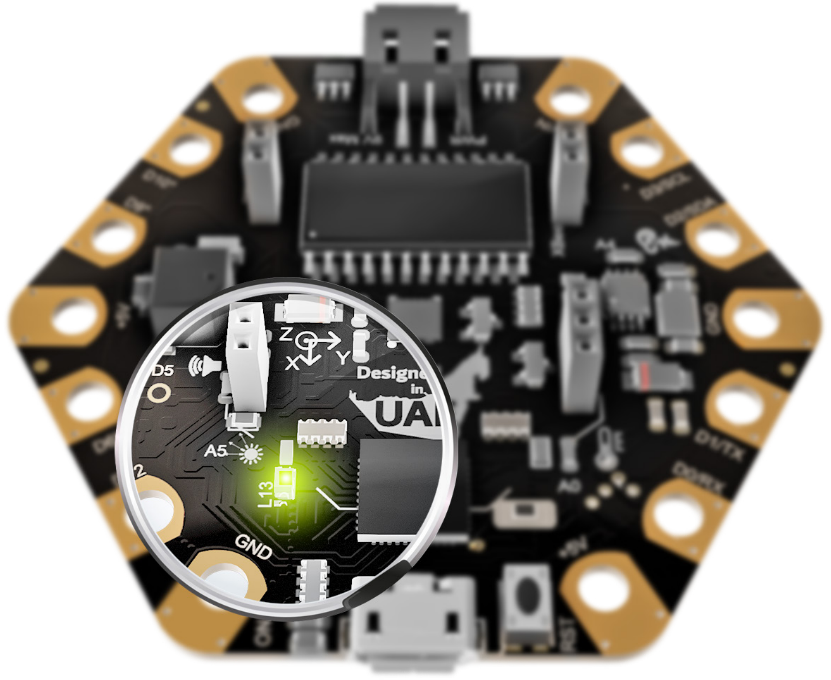
- The **L13** LED will now start blinking once every second.



```
// the setup function runs once when you press reset or pc
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (H
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off b
  delay(1000);                       // wait for a second
}
```

Done uploading.

```
Sketch uses 4128 bytes (14%) of program storage space. Maximu
Global variables use 149 bytes (5%) of dynamic memory, leavin
```



The previous test is a basic test, but if you see the LED blinking, you are sure that your board is detected, and the board name and port are selected correctly.

In the Arduino IDE, you can **Create**, **Save** and **Open** codes using the **Tools** section or from the **File** menu.

Installing the Maker Library

To start using any Arduino-compatible board, you have 2 options. You can either write the Arduino code from scratch yourself that deals with hardware directly or you can you use what is called a **library**. A library contains a premade code that most of the time is optimised and ready to be used.

Usually the second option makes it easy for you to start with. In Arduino, there are built-in libraries which you can use directly after installing the Arduino IDE. If you want to use the additional libraries, you will need to install them first. There are hundreds of additional libraries available on the Internet for you to download and test.

The Ibtikar Maker board has its own library which you can install and use. The library has lots of examples that walk you through the board features.

There are different ways of installing a library. You can visit the Arduino website for more details. It is a great website and you should always use it as a starting point. In this guide, we will focus on one of the methods mentioned there.

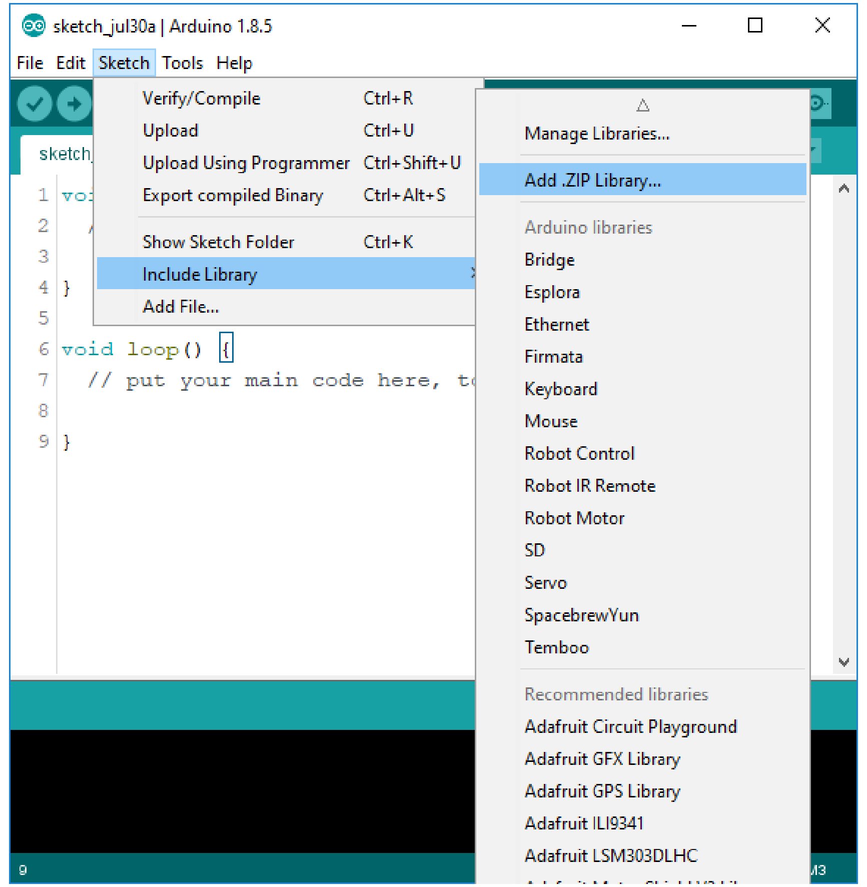
If the Arduino IDE was installed automatically on your device after installing Ardublockly, then the Maker library will already be installed. It is always good to ensure you have the latest library installed. This usually comes with lots of new useful functions and all bugs solved.

Importing the .zip Library

Libraries are often distributed as a ZIP file or folder. The name of the folder is the name of the library. For the Maker, there is a ZIP file called '**Ibtikar_Maker.zip**' which you can install from the Maker portal. The name usually contains the version number.

You do not need to unzip the file, leave it as is.

In the Arduino IDE, navigate to Sketch > Include Library > Add .ZIP Library. At the top of the drop-down list, select the option to ‘**Add .ZIP Library**’ as shown below.



```
voi
  /
}

void loop() {
  // put your main code here, t

}
```

Navigate to the file location, choose the ‘**Ibtikar_Maker.zip**’ and click open. The library now will be installed. You need to close the Arduino IDE and open it again for the library to show.

Testing the Maker Library

To test that you have successfully installed the library, you can go to the library examples and pick one. Read the first few lines of the code to see what the code will do. These lines are called the header of the code and are very useful to helps us humans understand the code.

In these examples, you only need to make sure which version of the board you are using. Look at the bottom left of the front side of your Maker. If you see **V1.0** or any other version number, then make sure you update the code with this number. If you do not see any number, then this means you are using **V0.0**.

Connect the Maker to your computer. Upload the sketch to the board and wait a couple of seconds. You should see the On-Board LED blink as the program is uploaded to the Maker. After that you should see the result of that code. Make sure the Maker response matches what is written in the header.

Before You Start

If you are new to the programming field, make sure you read this section before you start programming. This will give you a general overview of things you will need, to do the activities in this guide.

Logic and Conditional Statements

This category follows the Boolean Algebra system which is a branch of Algebra where the value of the variables has only two possible values; **True** or **False**.

Comparison Operators

The comparison operators compare the values on either side of them and decide the relation between them. They are also called relational operators. They take two inputs and they return **True** based on how these variables compare to each other.

Operator	Symbol in Arduino	Description
=	==	If the values of two operands are equal, then the condition becomes true.
≠	!=	If the values of two operands are not equal, then condition becomes true.
<	<	If the value of left operand is less than the value of right operand, then the condition becomes true.
≤	<=	If the value of the left operand is less than or equal to the value of the right operand, then the condition becomes true.
>	>	If the value of left operand is greater than the value of right operand, then the condition becomes true.
≥	>=	If the value of left operand is greater than or equal to the value of right operand, then the condition becomes true.

Boolean Operators

1. The **& & (logical and)** operator which returns **True** when both of its inputs are **True**.

2. The **|| (logical or)** operator which returns **True** when both inputs or one of them is **True**.

3. The **! (logical not)** operator which converts its input into the opposite. If the input originally is **True**, it will become **False** and vice versa.

Conditional Statements

Conditional statements perform different actions based on their Boolean conditions. There are different ways of doing this. One way is to use the **if** statement which you can configure, based on your need.

The **if** statement by itself is the simplest form of the conditional statements. If the condition is **True**, then you **do** something. For example; you have a number stored in a variable called **x** and you want to check the value of this variable. If the value is greater than or equal to zero, then you want to print the message '**x is greater than or equal to 0**'.

The other form of the conditional **if** statements, is the **if - else** form. This structure allows you to check the condition, and if it is **True** or not. If it is **True**, then you **do** something. **Else** you **do** another action.

For the same example, if the value of **x** is greater than or equal to zero, then you want to print the message '**x is greater than or equal to 0**'. Else, then print the message '**x is less than 0**'.

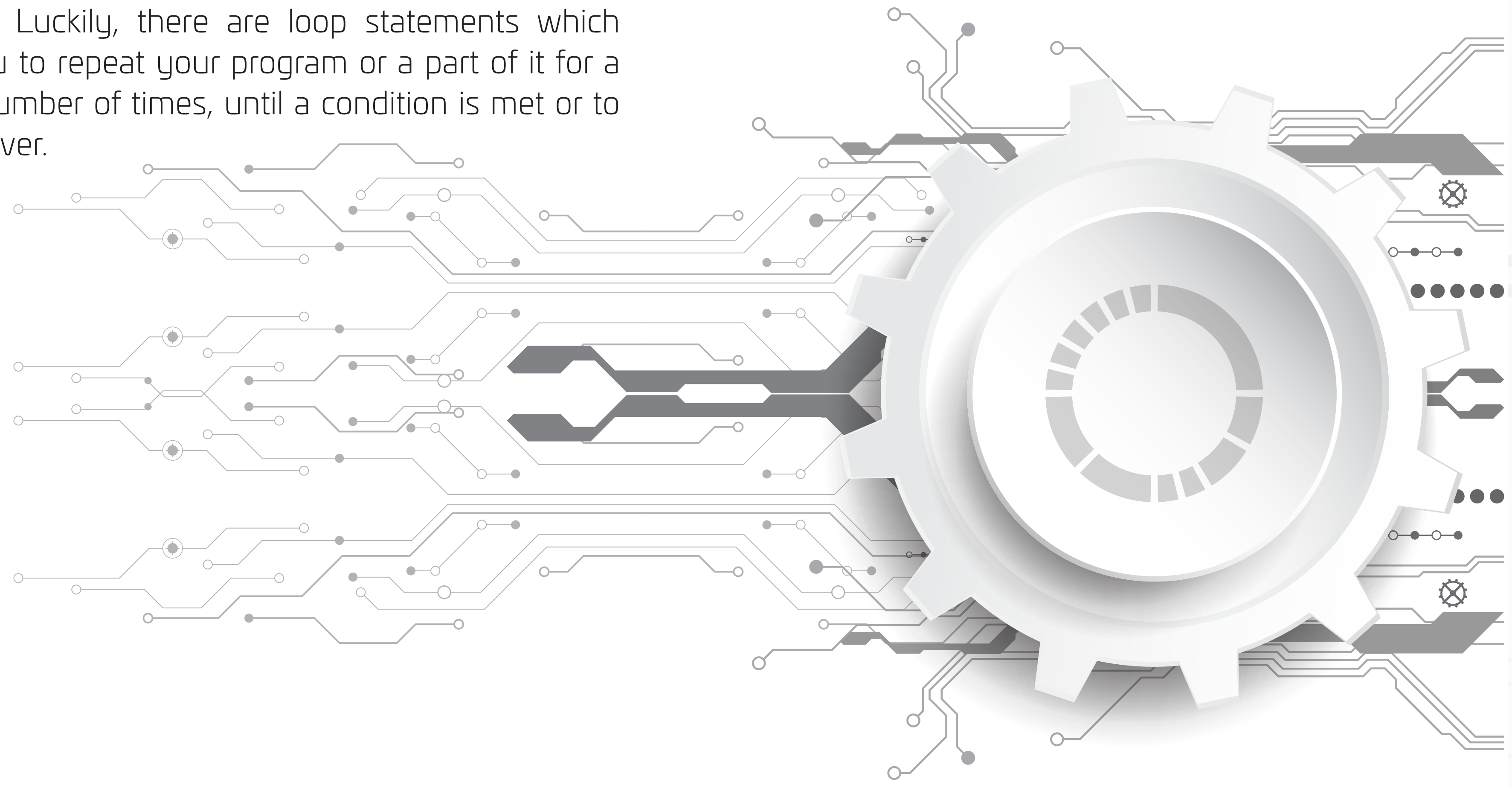
The third form of the conditional **if** statements, is the **if - else if - else** form. This structure allows you to check multiple conditions and do different actions.

For the same example, if the value of **x** is greater than zero, then print '**x is greater than 0**'. But if the value of **x** is less than zero, then print '**x is less than 0**'. Else, print '**x equals zero**' since it is the last case you may encounter.

For each **if** statement, you must start with the **if** and you can add as much as you want from the **else if** and at most, one **else** at the end.

Loops

Sometimes it is important to repeat a program without stopping. One way is to repeat the commands you need, but this does not make sense if your program is long or if you want the program to run without stopping. Luckily, there are loop statements which allow you to repeat your program or a part of it for a certain number of times, until a condition is met or to loop for ever.



Maker Activities

On-Board LED

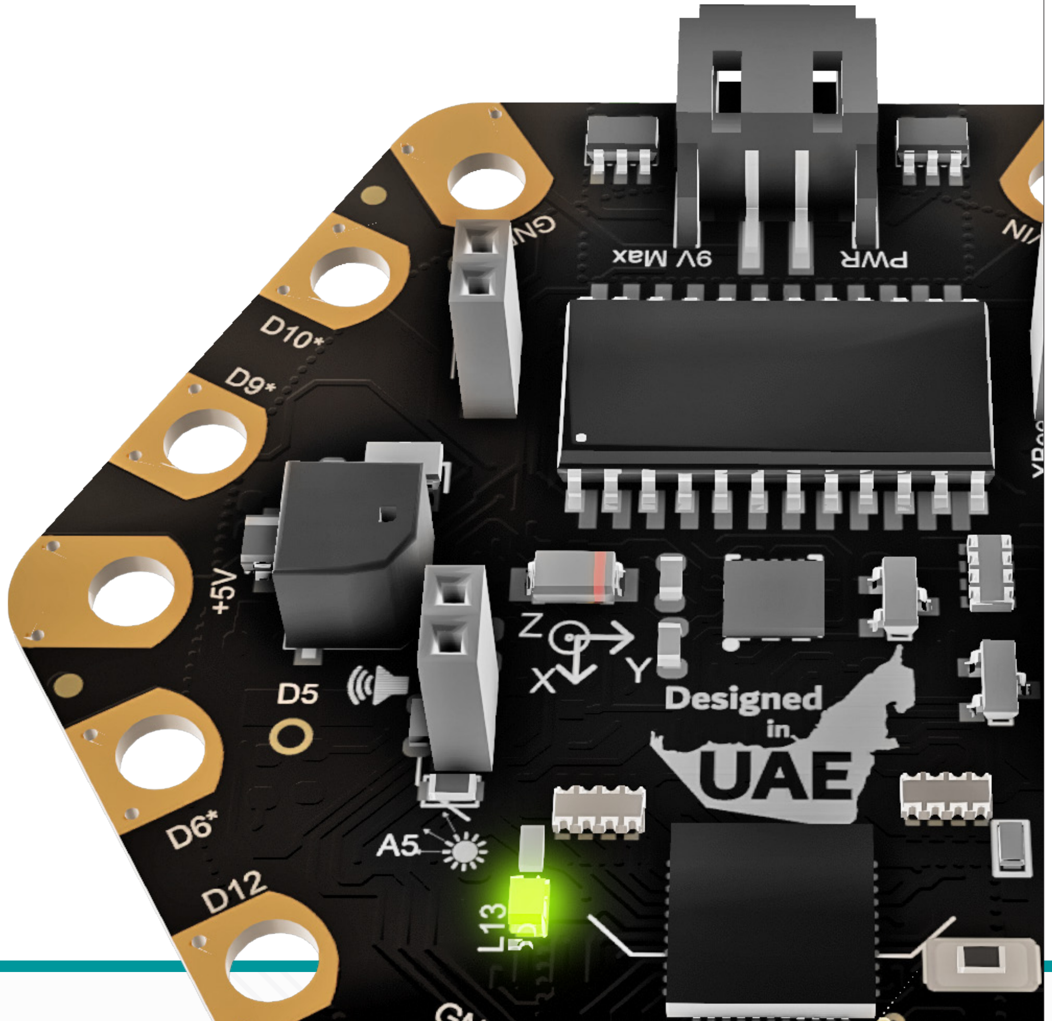
This LED has its own command in the Maker library. It can be controlled to be ON or OFF.

To turn the LED ON use the following command:

```
IBMaker.ledbuiltin(HIGH);
```

And to turn it OFF, use:

```
IBMaker.ledbuiltin(LOW);
```



Activity 1: LED ON

In this activity, the on-board LED will be ON forever. Forever means until you unplug the Maker, or the battery dies. You will first need to include the Maker library and initialise it in the setup function. Choose the board version which corresponds to the board you have.

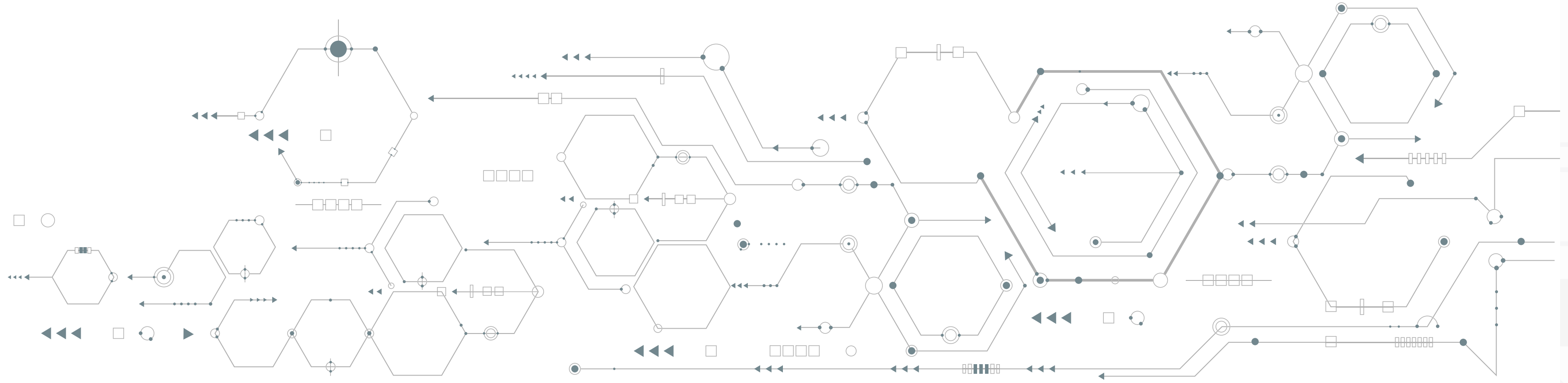
```
#include <Ibtikar_IBMaker.h>
void setup() {
  IBMaker.begin("V0.00");  // or "V1.00"
}
void loop() {
  IBMaker.ledbuiltin(HIGH);
}
```

Upload the program to your Maker board. The LED will be ON.

Activity 2: LED Blink

In this activity, the on-board LED will blink each second. You will need two LED commands and two wait commands. Arrange the commands as shown and upload the program. The LED will blink each second. This program is very similar to the Blink example you used to test the board.

```
#include <Ibtikar_IBMaker.h>
void setup() {
  IBMaker.begin("V0.00");  // or "V1.00"
}
void loop() {
  IBMaker.ledbuiltin(HIGH);
  delay(1000);
  IBMaker.ledbuiltin(LOW);
  delay(1000);
}
```



LED Grid

Maker can control the 25 LEDs at once or each one individually. In the Maker library, there are many commands available. Some of these commands allow you to display numbers, characters, scroll strings, or even draw shapes.

Activity 3: Draw Shapes

You can draw shapes on the Maker using different methods. For example, you can use:

- The **setLed** command to turn ON or OFF an individual LED by specifying the row and column of the LED followed by the state (HIGH or LOW).

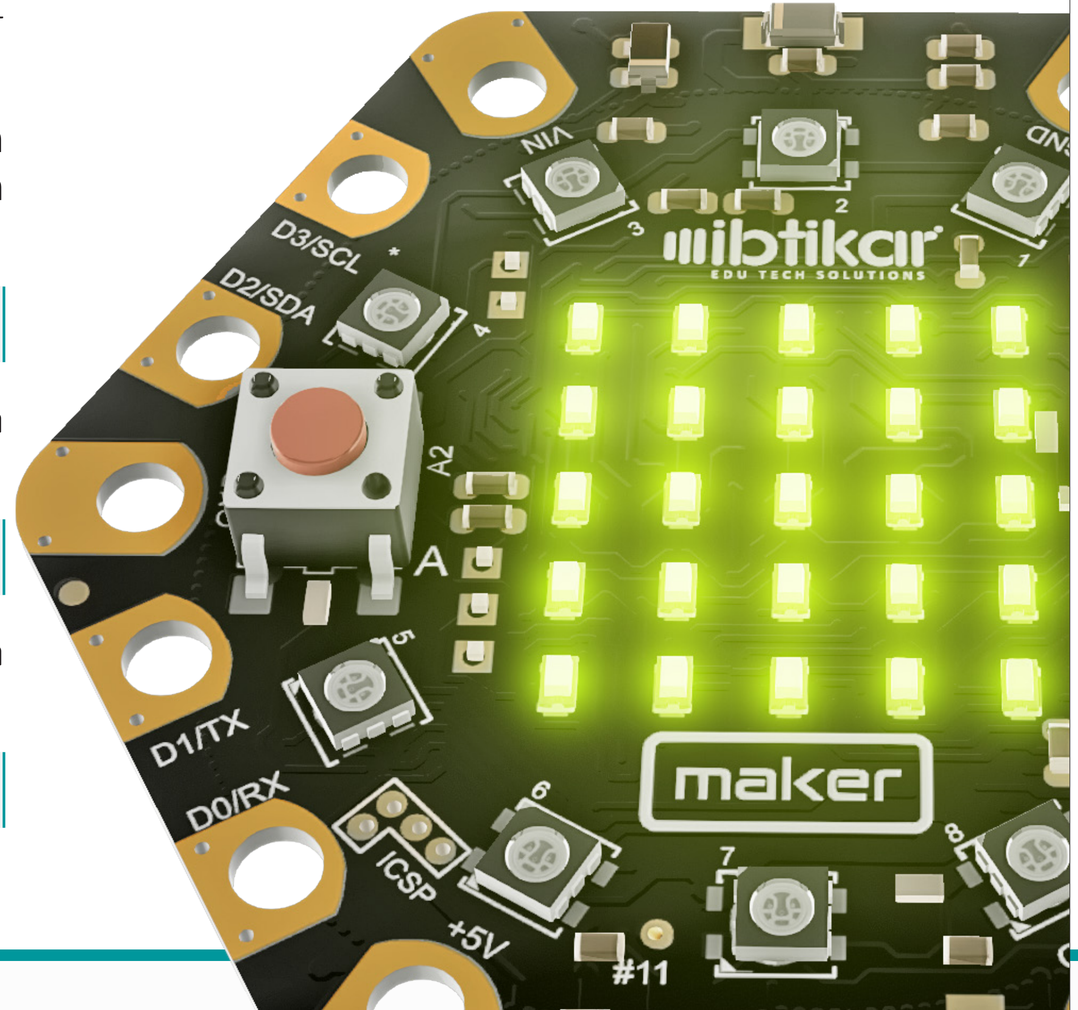
```
IBMaker.setLed(row, column, state)
```

- The **setRow** command to control the LEDs in a specific row at once.

```
IBMaker.setRow(row_number, 0bxxxxxxxx)
```

- The **setColumn** command to control the LEDs in a specific column at once.

```
IBMaker.setColumn(column_number, 0bxxxxxxxx)
```



Note

The second parameter in the **setRow** and **setColumn** commands is an 8-bit binary number. Since the Maker has a grid of 5X5, only the 5 most significant bit are important. The last three have no effect on the Maker grid. For example, to turn ON a complete row on the grid, the binary number should be 0b11111000.

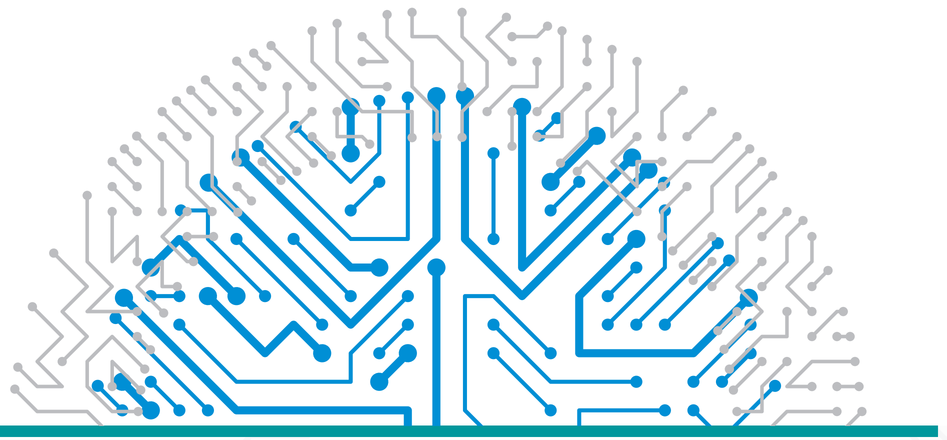
In the following activity, you will create a program to blink each LED individually using two for-loops, one for the rows and one for the columns.

Since you need to go through rows and columns at the same time, then you need two different variables, one for the rows and one for the columns. You can create a new variable by typing the type of the variable followed by the name of that variable.

Now, write the code as shown. Since the LED grid is 5 by 5 and the indexing starts from 0, then each loop should count from 0 to 4.

```
#include <Ibtikar_IBMaker.h>
int i;
int j;
void setup() {
  IBMaker.begin("V0.00");  // or "V1.00"
}
void loop() {
  for (i = 0; i <= 4; i++) {
    for (j = 0; j <= 4; j++) {
      IBMaker.setLed(i, j, HIGH);
      delay(100);
      IBMaker.setLed(i, j, LOW);
      delay(100);
    }
  }
}
```

If you swapped the inner and outer loops, then the LEDs pattern will be a vertical one instead of horizontal.



Activity 4: Display a Number

To display numbers on the Maker, you need the following command:

```
IBMaker.Leds_Num(0, duration);
```

This command accepts integer numbers (numbers without a fraction). It also accepts variables, so you can display a counter for example. In the first parameter, you specify the number you want to show. In the second parameter, you specify the scrolling step duration.

The following program will display numbers from 1 to 15 with a 100-millisecond scrolling duration.

```
#include <Ibtikar_IBMaker.h>
int i;
void setup() {
  IBMaker.begin("V0.00");   // or "V1.00"
}
void loop() {
  for (i = 1; i <= 15; i++) {
    IBMaker.Leds_Num(i, 100);
  }
}
```

Activity 5: Display a Character

To display a single character (either a letter, number or a symbol) on the Maker, you need the following command:

```
IBMaker.Leds_Char('', duration);
```

This command accepts a single character in its first parameter. In the second parameter, you specify the display duration.

This means that the character you want, will be displayed at once without scrolling for the duration you specify.

To see the difference, try the following program.

```
#include <Ibtikar_IBMaker.h>
void setup() {
  IBMaker.begin("V0.00");   // or "V1.00"
}
void loop() {
  IBMaker.Leds_Char('?', 1000);
  IBMaker.Leds_Char('4', 1000);
}
```

Once you upload it to the Maker, you will see the question mark symbol for 1 second then number 4 for another second. This will repeat forever.

Activity 6: Scroll a String

If you want to scroll a text, your name for example, you will need a different command called **Leds_Str**.

```
IBMaker.Leds_Str("", duration);
```

Note the difference between the single quotation in the **Leds_Char** command and the double quotation in the **Leds_Str** command. In the second parameter, you specify the scrolling step duration.

In the following example, we will scroll “Hello Maker” with a duration of 200ms for each scrolling step.

```
#include <Ibtikar_IBMaker.h>
void setup() {
  IBMaker.begin("V0.00");   // or "V1.00"
}
void loop() {
  IBMaker.Leds_Str("Hello Maker", 200);
}
```

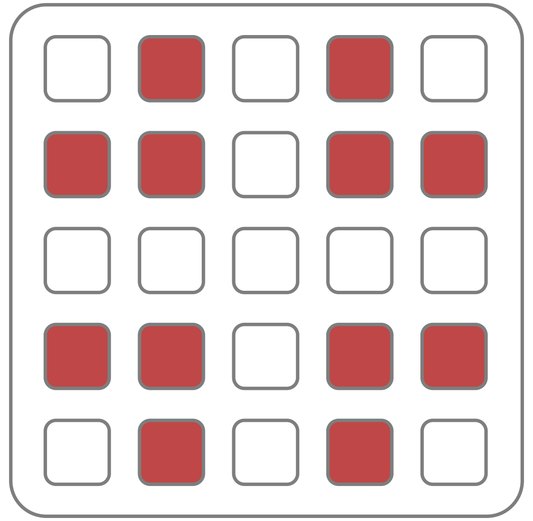
Upload the program to your Maker and check the result. Now change the code to make it scroll your name.

Activity 7: Move and Turn an LED

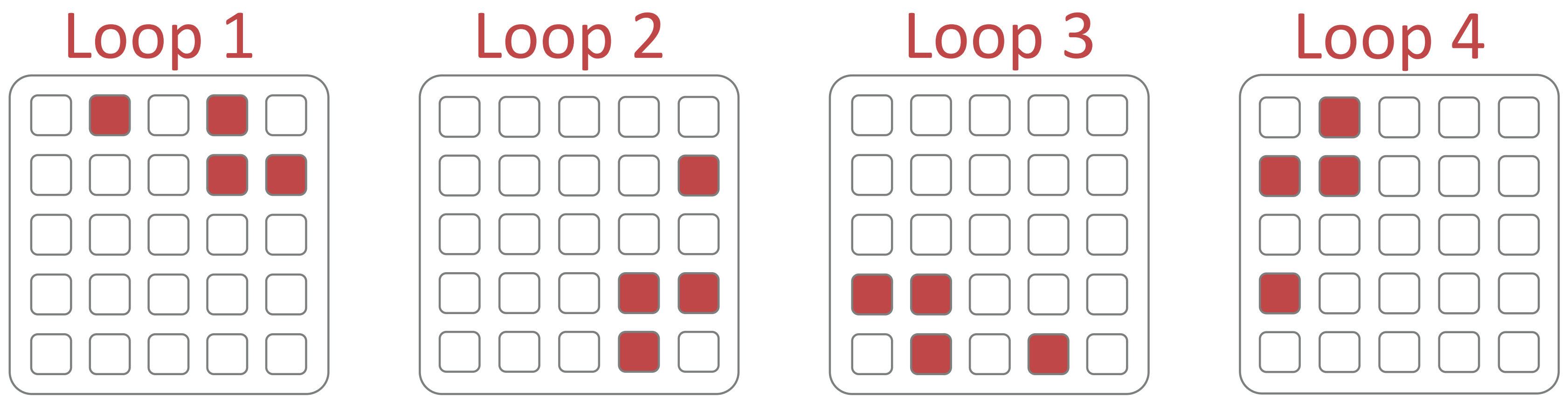
Since the 25 LEDs form a grid, you can consider them as a coordinate system. This means that you can create an origin, move an LED in a certain direction and turn clockwise or counter clockwise. This makes it easy to create complex patterns in a shorter code.

```
IBMaker.Create_Center(x, y)
IBMaker.Move(steps)
IBMaker.Turn(CW)
IBMaker.Turn(CCW)
```

In this activity, you will create the following pattern with one LED being ON at a time.



You can think of this as a pattern that repeats itself 4 times, as shown.



The origin should start from the point (0,1). Then the LED will move two steps, turn clockwise, move one step, turn counter clockwise and finally move one step. If the same pattern is repeated 4 times, you will end up with the required pattern. Since this process is very fast, a time delay is needed in each part. Your program should look like this.

Notice that to create the origin once, you place its command in the setup part.

```
#include <Ibtikar_IBMaker.h>
void setup() {
  IBMaker.begin("V0.00");  // or "V1.00"
  IBMaker.Create_Center(0, 1);}
void loop() {
  for (int count = 0; count < 4; count++){
    IBMaker.Move(2);
    IBMaker.Turn(CW);
    delay(250);
    IBMaker.Move(1);
    IBMaker.Turn(CCW);
    delay(250);
    IBMaker.Move(1);
    IBMaker.Turn(CW);
    delay(250);
}}
```

Activity 8: LEDs Brightness

The last two commands that control the LED grid are the **LEDBrightness** and **clearDisplay**. The brightness command allows you to change the brightness of the 25 LEDs at once. You can change the brightness level from 1 to 15. The clear command turns OFF all LEDs at once. This is useful if you made a pattern and then you want to turn the grid OFF without the need for turning OFF each LED individually.

```
IBMaker.clearDisplay(0x00)
IBMaker.LEDBrightness(level)
```

In this activity, you will increase the brightness from 1 to 15 in steps one at a time. Inside the for-loop, you will add the brightness command, turn all LEDs ON and wait for 100ms. Once the loop is finished, you will clear all the LEDs at once and wait for half a second. Your program should look like the following.

```
#include <Ibtikar_IBMaker.h>
int i;
void setup() {
  IBMaker.begin("V0.00");   // or "V1.00"
}
void loop() {
  for (i = 1; i <= 15; i++) {
    IBMaker.LEDBrightness(i);
    IBMaker.setRow(0, 0b11111000);
    IBMaker.setRow(1, 0b11111000);
    IBMaker.setRow(2, 0b11111000);
    IBMaker.setRow(3, 0b11111000);
    IBMaker.setRow(4, 0b11111000);
    delay(100);
  }
  IBMaker.clearDisplay(0x00);
  delay(500);
}
```

Note:

To avoid hurting your eyes do not look directly at the LED grid when the brightness is high.

Read the Buttons

The Maker has two buttons that you can read. These buttons are button A on the left side of the board and button B on the right side. In the Maker library, there are three commands that you can use, two of them check if the button is pressed or not while the third command returns the number of clicks you clicked on a button.

```
IBMaker.ButtonL()
IBMaker.ButtonR()
```

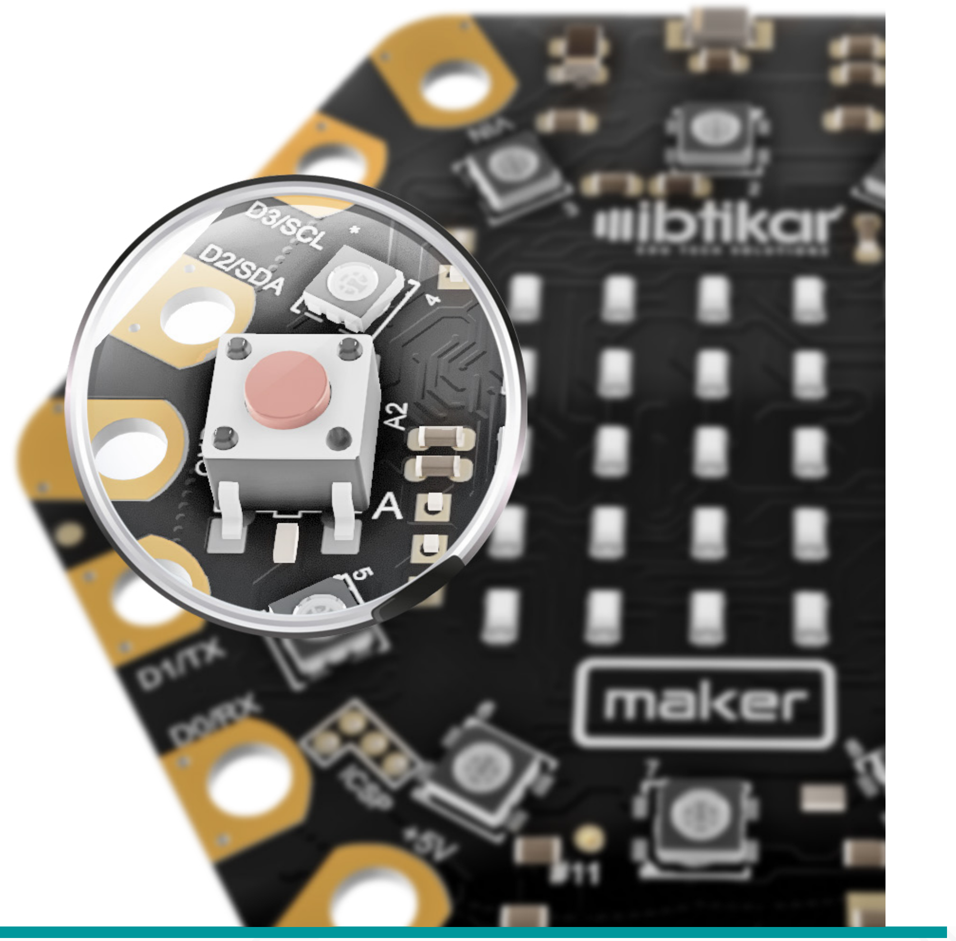
For the third command, you need to choose which button you want to read and the maximum number of clicks you want to reach. For example, if the value is 2, this means the command returns 0 if it is not clicked, 1 if it is clicked once and 2 if it is clicked twice. If you clicked three times, this will not be detected, and the maximum will be 2.

The following command will check the left button for the number of clicks you click:

```
IBMaker.ButtonCount(IBPIN_LEFT, count)
```

While this command will the check the right button for the same:

```
IBMaker.ButtonCount(IBPIN_RIGHT, count)
```



Activity 9: Read Both Buttons

In this activity, you will read both buttons and display a character on the LED grid. If the left button (or button A) is pressed, then 'A' will be displayed on the grid. Similarly, if the right button (or button B) is pressed, 'B' will be displayed on the grid. Try the following program and upload it to your Maker. Now, click on each button and notice the letter displayed on the LED grid.

```
#include <Ibtikar_IBMaker.h>
void setup() {
  IBMaker.begin("V0.00");  // or "V1.00"
}
void loop() {
  if (IBMaker.ButtonL()) {
    IBMaker.Leds_Char('A', 200);}
  if (IBMaker.ButtonR()) {
    IBMaker.Leds_Char('B', 200);}
}
```

Activity 10: Detect Multiple Clicks

In this activity, you will read how many times button A is pressed and you will display a happy face if you reach the maximum number of clicks. If you do not reach the maximum value, a sad face will be displayed.

Since the Button Count command will return different values based on how many times you press the button, then it is better to read the command and store its value in a variable. Then based on the value stored, you can make decisions.

Try the following program and upload it to your Maker and see the maximum number of clicks you reached.

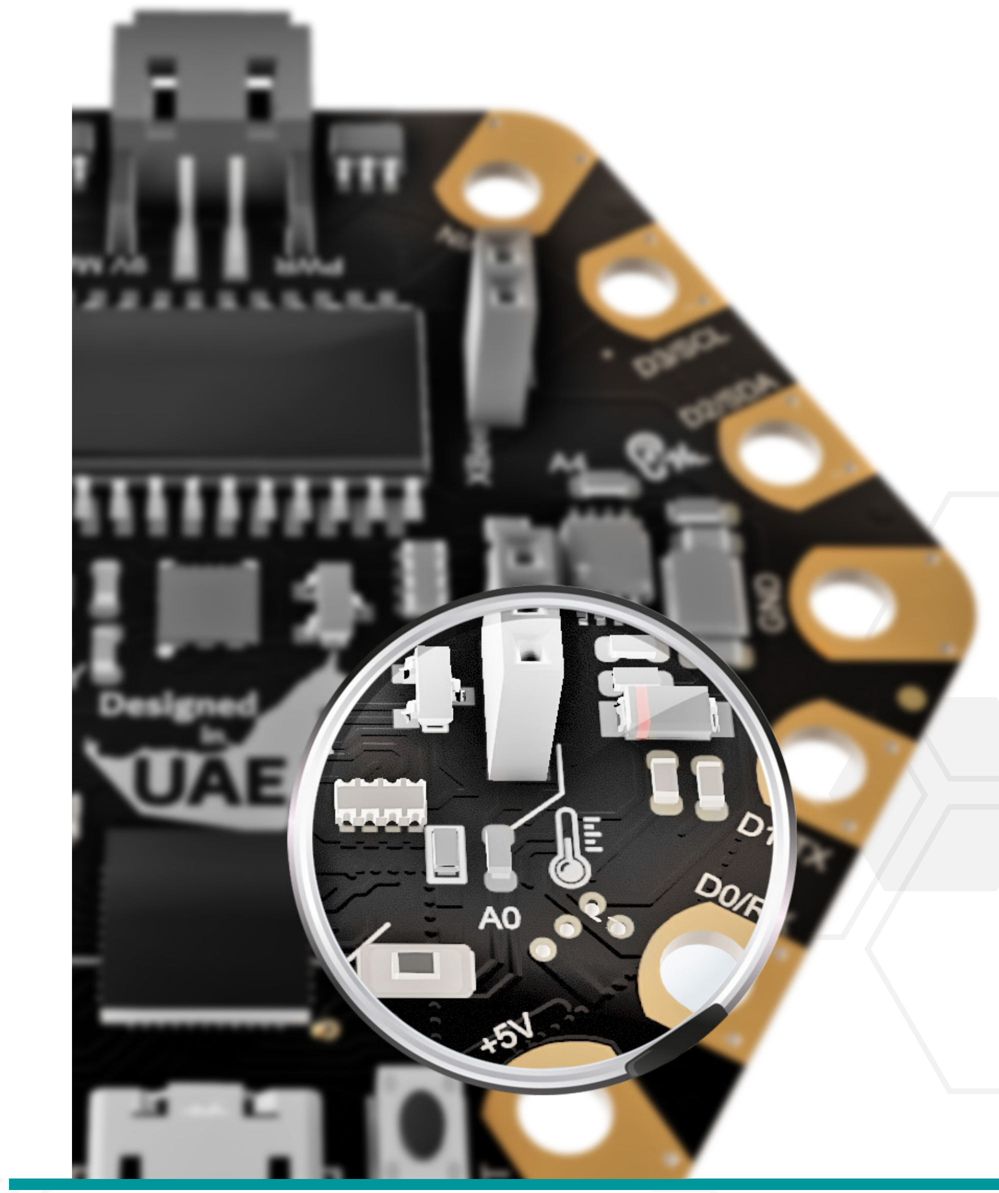
```
#include <Ibtikar_IBMaker.h>
int item;
void setup() {
  IBMaker.begin("V0.00");  // or "V1.00"
}
void loop() {
  item = IBMaker.ButtonCount(IBPIN_LEFT, 5);
  if (item == 5) {
    IBMaker.setRow(0, 0b00000000);
    IBMaker.setRow(1, 0b01010000);
    IBMaker.setRow(2, 0b00000000);
    IBMaker.setRow(3, 0b10001000);
    IBMaker.setRow(4, 0b01110000);
  } else {
    IBMaker.setRow(0, 0b00000000);
    IBMaker.setRow(1, 0b01010000);
    IBMaker.setRow(2, 0b00000000);
    IBMaker.setRow(3, 0b01110000);
    IBMaker.setRow(4, 0b10001000);}}
```

Activity 11: Detect Multiple Clicks (Optimised)

You may have noticed that even when you do not click button A at all, the sad face is still there. You can improve the code by adding another case to check when the number of counts is zero. It is good to add wait commands when you display the happy and sad faces. Try the following program.

```
#include <Ibtikar_IBMaker.h>
int item;
void setup() {
  IBMaker.begin("V0.00");  // or "V1.00"
}
void loop() {
  item = IBMaker.ButtonCount(IBPIN_LEFT, 5);
  if (item == 5) {
    IBMaker.setRow(0, 0b00000000);
    IBMaker.setRow(1, 0b01010000);
    IBMaker.setRow(2, 0b00000000);
    IBMaker.setRow(3, 0b10001000);
    IBMaker.setRow(4, 0b01110000);
    delay(500);
  } else if (item == 0) {
    IBMaker.clearDisplay(0x00);
  } else {
    IBMaker.setRow(0, 0b00000000);
    IBMaker.setRow(1, 0b01010000);
    IBMaker.setRow(2, 0b00000000);
    IBMaker.setRow(3, 0b01110000);
    IBMaker.setRow(4, 0b10001000);
    delay(500);
  }
}
```

Read the Temperature



The temperature sensor on the Maker can be used to measure the board temperature. The temperature command in the Maker library allows you to read the temperature in either the Celsius or Fahrenheit unit.

```
IBMaker.temperatureC()
IBMaker.temperatureF()
```

Activity 12: Display the Temperature

In this activity, you will scroll the temperature in both the Celsius and Fahrenheit units on the LED grid. Try the following program and upload it to your Maker.

```
#include <Ibtikar_IBMaker.h>
void setup() {
  IBMaker.begin("V0.00");  // or "V1.00"
}
void loop() {
  IBMaker.Leds_Num(IBMaker.temperatureC(), 200);
  IBMaker.Leds_Str("C ", 200);
  IBMaker.Leds_Num(IBMaker.temperatureF(), 200);
  IBMaker.Leds_Str("F ", 200);
}
```

Read Ambient Light

The light sensor on the Maker measures the ambient light. It returns a value between 0 to 1023 representing the light intensity level. Higher values mean the measured light level is high (there is light). You can find its command in the Maker library.

```
IBMaker.Sensor_Light()
```

Activity 13: Display the Ambient Light Value

In this activity, you will scroll the ambient light value on the LED grid. While you are testing, you can use a torch and direct it toward the sensor to notice the change. Try the following program and upload it to your Maker.

```
#include <Ibtikar_IBMaker.h>
void setup() {
  IBMaker.begin("V0.00");  // or "V1.00"
}
void loop() {
  IBMaker.Leds_Num(IBMaker.Sensor_Light(), 200);
}
```

Buzzer

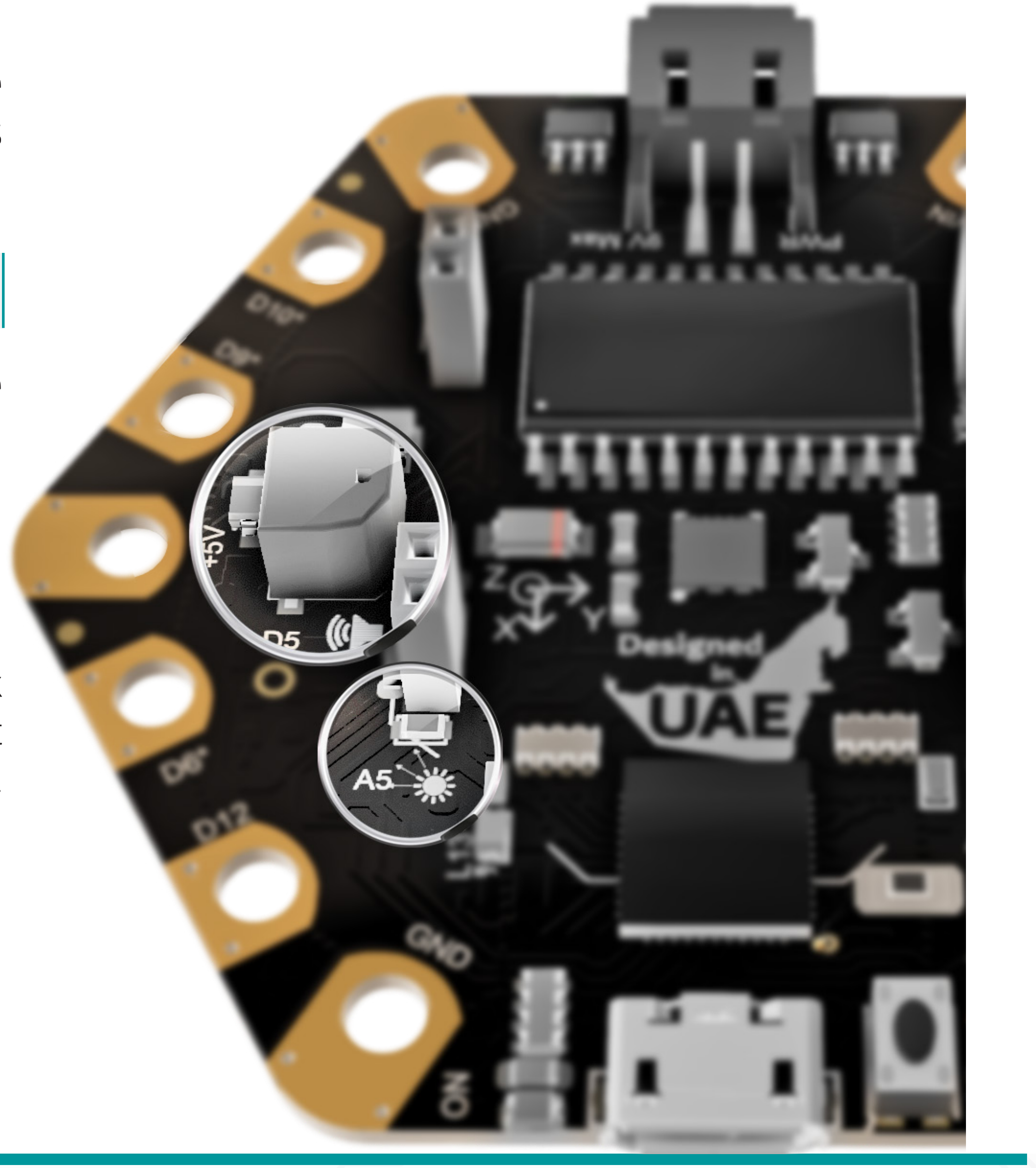
The magnetic buzzer generates tones by controlling the frequency and duration of the note needed. The buzzer has one command in the Maker library. This command by itself is a complete program.

```
IBMaker.playTone(100, 250)
```

The first parameter is the frequency in Hertz (Hz) while the second parameter is the duration in millisecond (ms).

Activity 14: Play Different Tones

Using 3 **playTone** commands, generate tones with different frequencies and fixed duration. Then fix the frequency and change the duration. Add a wait command after each note to have time to listen to it. In each case, upload the program to your Maker board and note the difference.



Your programs should look like these.

Program 1:

Variable frequency

Fixed Duration

```
#include <Ibtikar_IBMaker.h>
void setup() {
  IBMaker.begin("V0.00");  // or "V1.00"
  IBMaker.playTone(300, 250);
  delay(500);
  IBMaker.playTone(400, 250);
  delay(500);
  IBMaker.playTone(500, 250);
  delay(500);
}
void loop() {
}
```

Program 2:

Fixed frequency

Variable Duration

```
#include <Ibtikar_IBMaker.h>
void setup() {
  IBMaker.begin("V0.00");  // or "V1.00"
  IBMaker.playTone(300, 250);
  delay(500);
  IBMaker.playTone(300, 500);
  delay(500);
  IBMaker.playTone(300, 750);
  delay(500);
}
void loop() {
}
```

Can you tell the difference?

If you tried to upload these programs, but you decided to put the commands in the loop and the setup part, you will notice that it is not easy to identify between them. It may also be annoying to listen to them because they repeat forever. Therefore, the setup part is useful when you want to initialise variables or set properties of some components, like brightness for example.

Activity 15: Loop Tone Frequency

The **playTone** command accepts parameters as integer numbers or variables. In this activity, you will change the frequency from 100 to 2000 Hz with steps of 50 Hz, using a **for loop**. The duration will be fixed at 250 milliseconds. The frequency for loop and the tone commands will be played once in the code, so they are moved to the setup part. Your program should look like this.

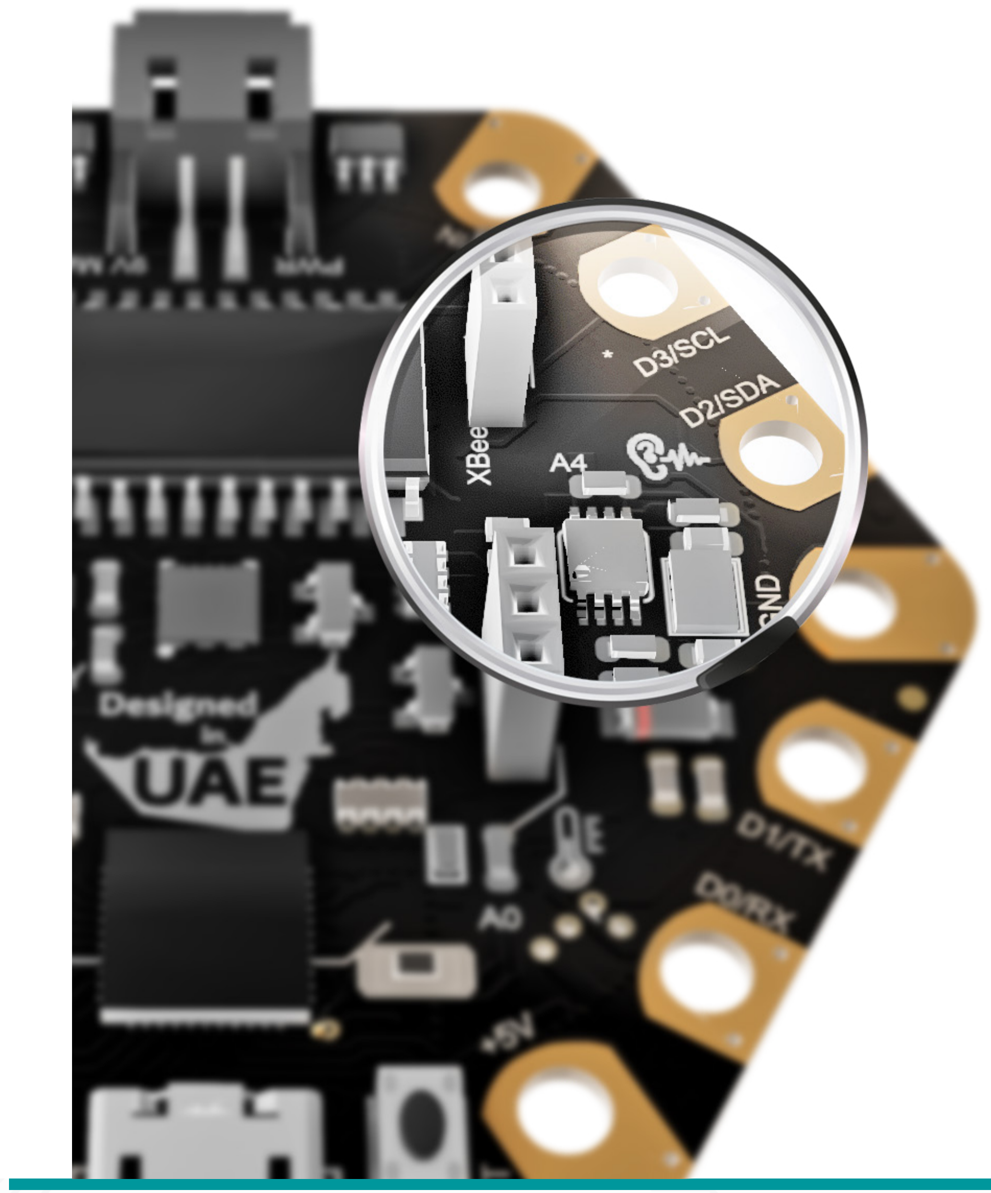
```
#include <Ibtikar_IBMaker.h>
int i;
void setup() {
  IBMaker.begin("V0.00");  // or "V1.00"
  for (i = 100; i <= 2000; i += 50) {
    IBMaker.playTone(i, 250);
  }
}
void loop() {
}
```

Try the program, upload it to your Maker and enjoy the music you have just made.

Note

To avoid damaging the buzzer, do not use higher frequency values (more than 2000Hz).

Read the Sound Level



The sound sensor on the Maker measures the sound intensity level. The sound sensor has one command in the Maker library. Even though it returns a value between 0 to 1023, the highest value it may reach, if you clap your hand for example, is around 400.

```
IBMaker.Sensor_Sound()
```

Note

Maker V0.0 has a special configuration in that, instead of the value increasing when the sound sensor detects noise, the value decreases. This should be considered when you create your code as explained in the next activity.

Activity 16: React to Sound

In this activity, you will detect the change in the sound level and play a tone. The tone frequency will depend on the sound level detected. You can pass the value of the detected sound directly to the **playTone** command.

You can even do more. You can cover a wider range of frequencies by mapping the sound range (0-400) to a frequency range (0-2000) Hz.

To avoid hearing noise when the frequency is low, you can put the **playTone** command inside an **if** statement so that you can hear a tone only when the value exceeds a threshold, say 100.

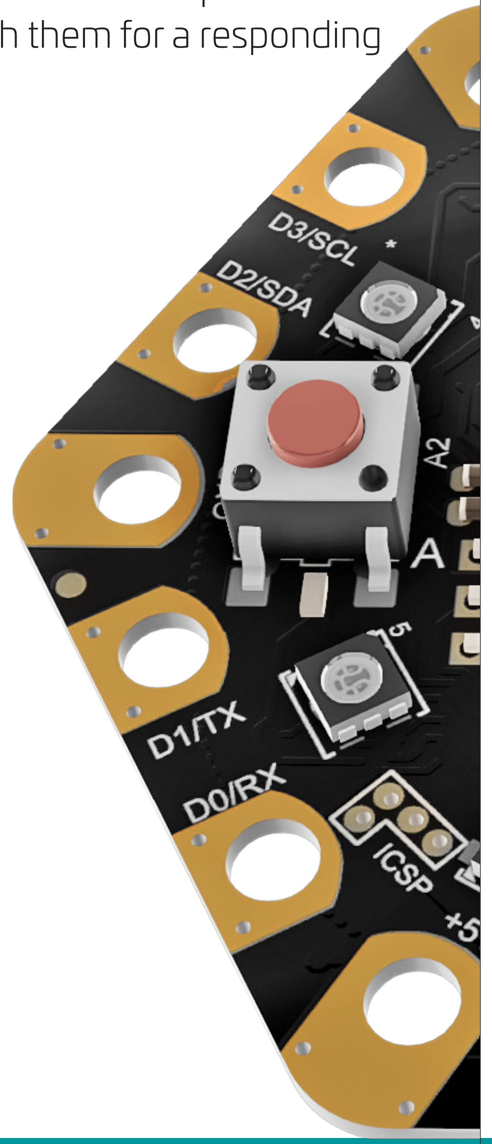
Try the following program, upload it to your Maker and clap your hand near the sound sensor. Did you hear anything? Now put your mouth close and blow air on the sensor. Did you hear different notes?

```
#include <Ibtikar_IBMaker.h>
int item;
void setup() {
  IBMaker.begin("V0.00");  // or "V1.00"
}

void loop() {
  // Use this for "V0.00"
  item = (map(IBMaker.Sensor_Sound(), 400, 0, 0, 2000));

  // Use this for "V1.00"
  // item = (map(IBMaker.Sensor_Sound(), 0, 400, 0, 2000));

  if (item > 350) {
    IBMaker.playTone(item, 200);
    delay(10);
  }
  delay(1);
}
```

Pin Pads

These 8 pin pads can be interfaced with extra input and output modules allowing you to extend the capabilities of your Maker board. On top of that, 7 of them, can be used as touch sensors. So, you can touch them for a responding action.

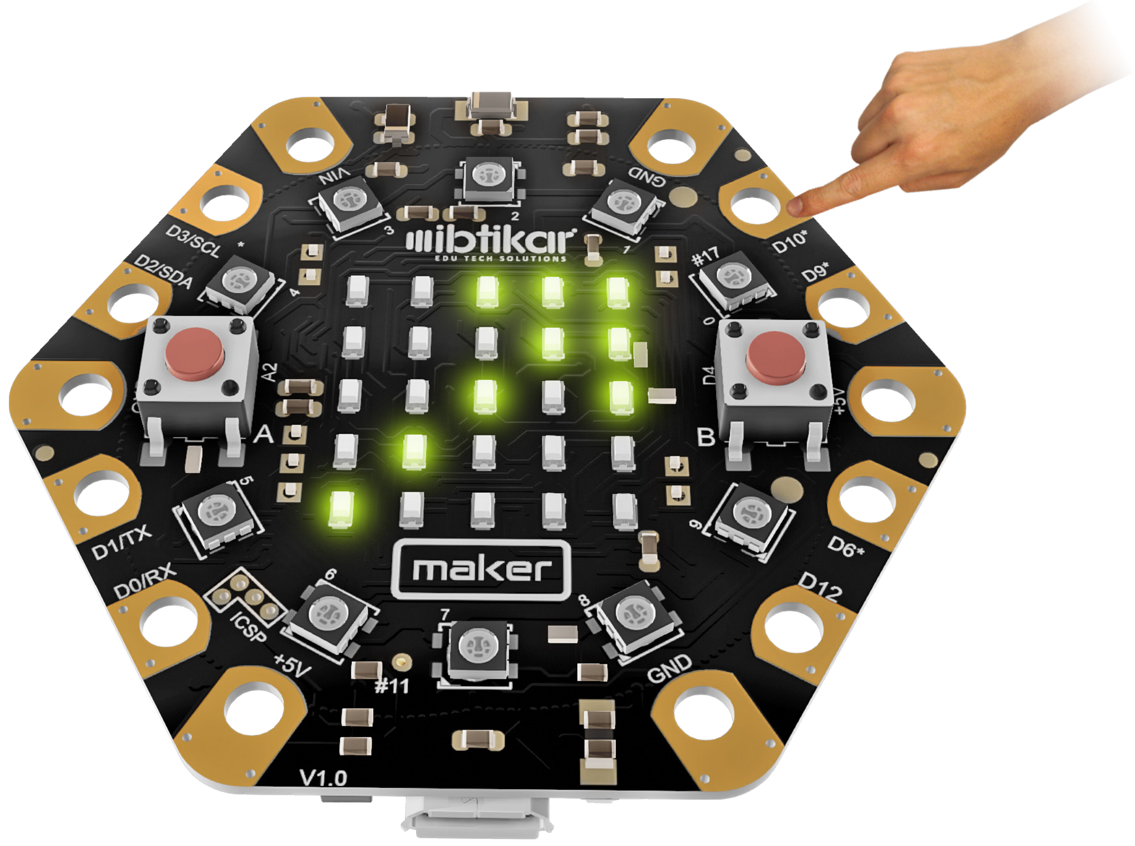
Pin	Touch	Digital	Analog	Serial	I2C	PWM	Interrupt
D0/RX	*	0		RX			INT2
D1/TX		1		TX			INT3
D2	*	2			SDA		INT1
D3	*	3			SCL	*	INT0
D6	*	6	A7			*	
D9	*	9	A9			*	
D10	*	10	A10			*	
D12	*	12	A11				



Activity 17: Detect Touch

In this activity, you will program the Maker to show an arrow, based on which touch pad you pressed. For example, if you touched pin 9 or pin 10, you should see an arrow toward these pins, as shown.

Each pin gives a value based on the capacitance. You will pick a threshold, say 200 and make the decisions based on that. Since the threshold will be the same for all touch pads, then you can use a variable and assign the value of say 200 to it. This will make it easier if you want to change the value later.



Try the following program and upload it to your Maker.

```
#include <Ibtikar_IBMaker.h>
int item;

void setup() {
  IBMaker.begin("V0.00");  // or "V1.00"
}

void loop() {
  item = 200;
  if (IBMaker.Touch(0) > item) {
    IBMaker.setRow(0, 0b00001000);
    IBMaker.setRow(1, 0b00010000);
    IBMaker.setRow(2, 0b10100000);
    IBMaker.setRow(3, 0b11000000);
    IBMaker.setRow(4, 0b11100000);  }
  if (IBMaker.Touch(2) > item || IBMaker.Touch(3) > item) {
    IBMaker.setRow(0, 0b11100000);
    IBMaker.setRow(1, 0b11000000);
    IBMaker.setRow(2, 0b10100000);
    IBMaker.setRow(3, 0b00010000);
    IBMaker.setRow(4, 0b00001000);  }
  if (IBMaker.Touch(6) > item || IBMaker.Touch(12) > item) {
    IBMaker.setRow(0, 0b10000000);
    IBMaker.setRow(1, 0b01000000);
    IBMaker.setRow(2, 0b00101000);
    IBMaker.setRow(3, 0b00011000);
    IBMaker.setRow(4, 0b00111000);  }
  if (IBMaker.Touch(9) > item || IBMaker.Touch(10) > item) {
    IBMaker.setRow(0, 0b00111000);
    IBMaker.setRow(1, 0b00011000);
    IBMaker.setRow(2, 0b00101000);
    IBMaker.setRow(3, 0b01000000);
    IBMaker.setRow(4, 0b10000000);  }
}
```

Note

Do not forget, pin 1 is not a touch pad.

Activity 18: Piano

In this activity, you will program the Maker to play a tone based on which touch pad you pressed. Try the following program and upload it to your Maker.

```
#include <Ibtikar_IBMaker.h>

int item;

void setup() {
  IBMaker.begin("V0.00");  // or "V1.00"
}

void loop() {
  item = 100;
  if (IBMaker.Touch(0) > item) {
    IBMaker.playTone(400, 250);
  }
  if (IBMaker.Touch(2) > item) {
    IBMaker.playTone(450, 250);
  }
  if (IBMaker.Touch(3) > item) {
    IBMaker.playTone(500, 250);
  }
  if (IBMaker.Touch(10) > item) {
    IBMaker.playTone(550, 250);
  }
  if (IBMaker.Touch(9) > item) {
    IBMaker.playTone(600, 250);
  }
  if (IBMaker.Touch(6) > item) {
    IBMaker.playTone(650, 250);
  }
  if (IBMaker.Touch(12) > item) {
    IBMaker.playTone(700, 250);
  }
}
```

NeoPixels

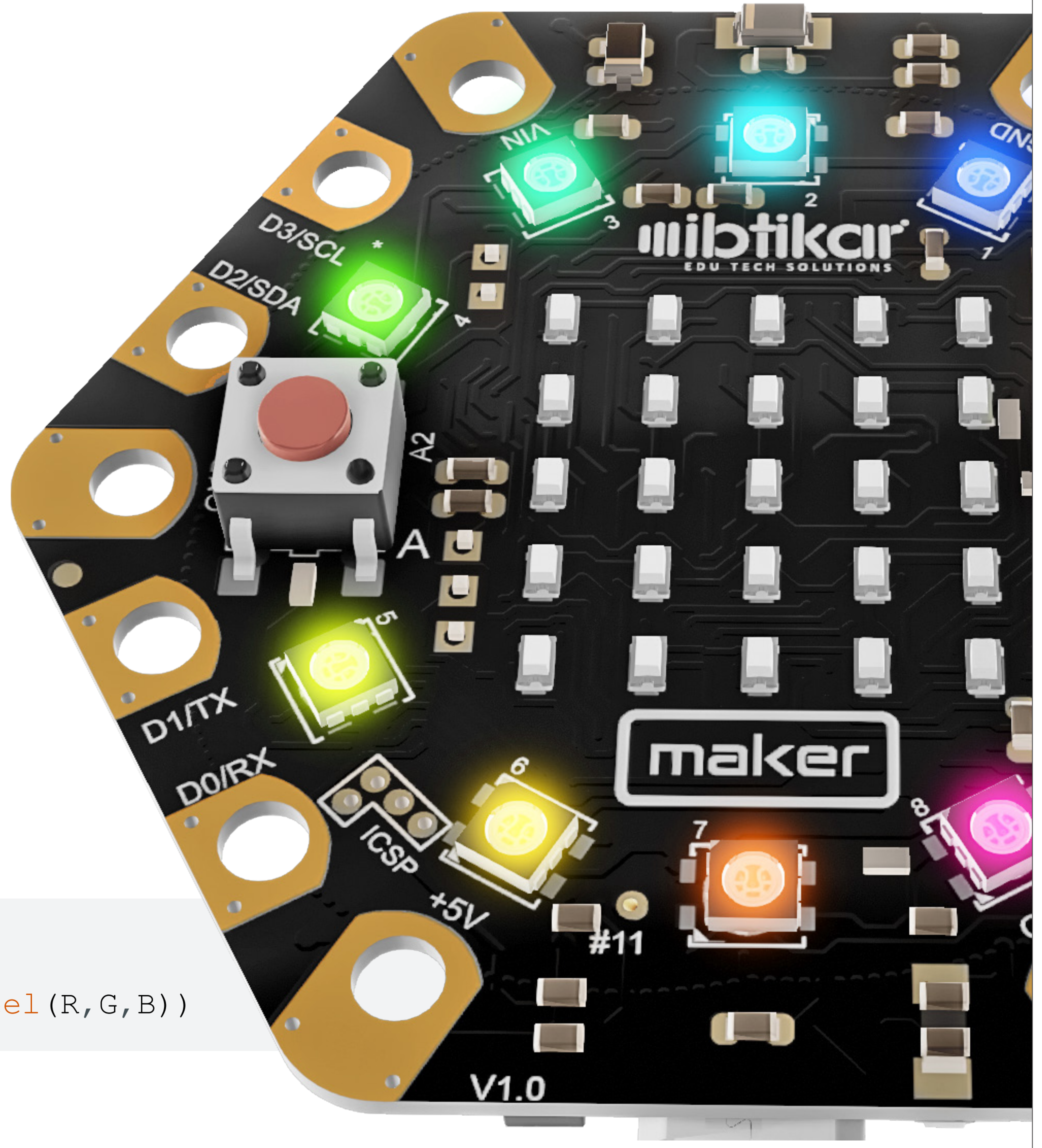
The Maker has 10 RGB LEDs. Unlike the 25 LEDs which cannot change their colour, the RGB LEDs can be programmed to show any colour by combining the three different colours. Think of each pixel as three small LEDs combined, and each LED has a different colour (**Red**, **Green** and **Blue**).

Like the LED grid commands, you can set the brightness of all NeoPixels using the **NeoBrightness** command and turn them all OFF using the **clear Pixels** command. The other command allows you to choose one of the NeoPixels and choose its colour, by specifying the RGB values.

Each NeoPixel has a number written next to it. This will help you identify which one you want to control.

The NeoPixels have the following commands in the Maker library:

```
IBMaker.clearPixels()
IBMaker.NeoBrightness(level)
IBMaker.setPixelColor(number, IBMaker.colorWheel(R,G,B))
```



Activity 19: Colour Wheel

In this activity, you will give each NeoPixel a different colour. Since there are 10 NeoPixels then you will need 10 lines of code. Try the following program and upload it to your Maker.

```
#include <Ibtikar_IBMaker.h>

void setup() {
  IBMaker.begin("V0.00");  // or "V1.00"
}

void loop() {
  IBMaker.setPixelColor(0, IBMaker.colorWheel(255, 255, 255));
  IBMaker.setPixelColor(1, IBMaker.colorWheel(255, 0, 0));
  IBMaker.setPixelColor(2, IBMaker.colorWheel(160, 200, 0));
  IBMaker.setPixelColor(3, IBMaker.colorWheel(255, 255, 0));
  IBMaker.setPixelColor(4, IBMaker.colorWheel(0, 255, 255));
  IBMaker.setPixelColor(5, IBMaker.colorWheel(255, 0, 255));
  IBMaker.setPixelColor(6, IBMaker.colorWheel(0, 255, 0));
  IBMaker.setPixelColor(7, IBMaker.colorWheel(128, 0, 128));
  IBMaker.setPixelColor(8, IBMaker.colorWheel(0, 128, 128));
  IBMaker.setPixelColor(9, IBMaker.colorWheel(0, 0, 128));
}
```

Did you see all the NeoPixels coloured?

Activity 20: Fading Sequence

In this activity, you will change the brightness of the NeoPixels using a **for-loop.** The following program starts by turning OFF all NeoPixels, then goes to the brightness **for-loop**. The brightness can be changed from 0 to 255. For each NeoPixel, you can choose whatever colour you prefer.

A 10ms delay between each brightness update is added. Once the loop is finished, the program will wait half a second before starting from the beginning.

Try the following program and upload it to your Maker.

```
#include <Ibtikar_IBMaker.h>
int i;
void setup() {
  IBMaker.begin("V0.00");  // or "V1.00"
}
void loop() {
  IBMaker.clearPixels();
  for (i = 0; i <= 255; i++) {
    IBMaker.NeoBrightness(i);
    IBMaker.setPixelColor(0, IBMaker.colorWheel(255, 0, 0));
    IBMaker.setPixelColor(1, IBMaker.colorWheel(255, 127, 0));
    IBMaker.setPixelColor(2, IBMaker.colorWheel(255, 255, 0));
    IBMaker.setPixelColor(3, IBMaker.colorWheel(127, 255, 0));
    IBMaker.setPixelColor(4, IBMaker.colorWheel(0, 255, 0));
    IBMaker.setPixelColor(5, IBMaker.colorWheel(0, 0, 255));
    IBMaker.setPixelColor(6, IBMaker.colorWheel(0, 255, 255));
    IBMaker.setPixelColor(7, IBMaker.colorWheel(255, 0, 255));
    IBMaker.setPixelColor(8, IBMaker.colorWheel(75, 0, 130));
    IBMaker.setPixelColor(9, IBMaker.colorWheel(255, 255, 255));
    delay(10);}
  delay(500);}
```

Activity 21: Looping LEDs

Sometimes it is easier to loop the NeoPixels number instead of adding 10 lines of code. This will help in making your program compact and easier to understand.

In this activity, you will set the 10 NeoPixels to a specific colour using 1 **NeoPixel** command and a **for-loop**. Try the following program and upload it to your Maker.

```
#include <Ibtikar_IBMaker.h>
int LED;
void setup() {
  IBMaker.begin("V0.00");  // or "V1.00"
}
void loop() {
  IBMaker.clearPixels();
  for (LED = 0; LED <= 9; LED++) {
    IBMaker.setPixelColor(LED, IBMaker.colorWheel(189, 71, 71));
    delay(250);
  }
}
```

Activity 22: Looping Colours and LEDs

What if you want to change the RGB values?

In this case, you can add a separate **for-loop** for each value. In the following program, you will change the value of the 10 NeoPixels at once. The three RGB loops will start from 0 to 255 with a step of 50. Each time the blue-colour loop finishes, the green-colour loop will change the value of **G** with a step of 50. And each time the green-colour loop finishes, the red-colour loop will change the value of **R** with a step of 50.

Try the following program and upload it to your Maker.

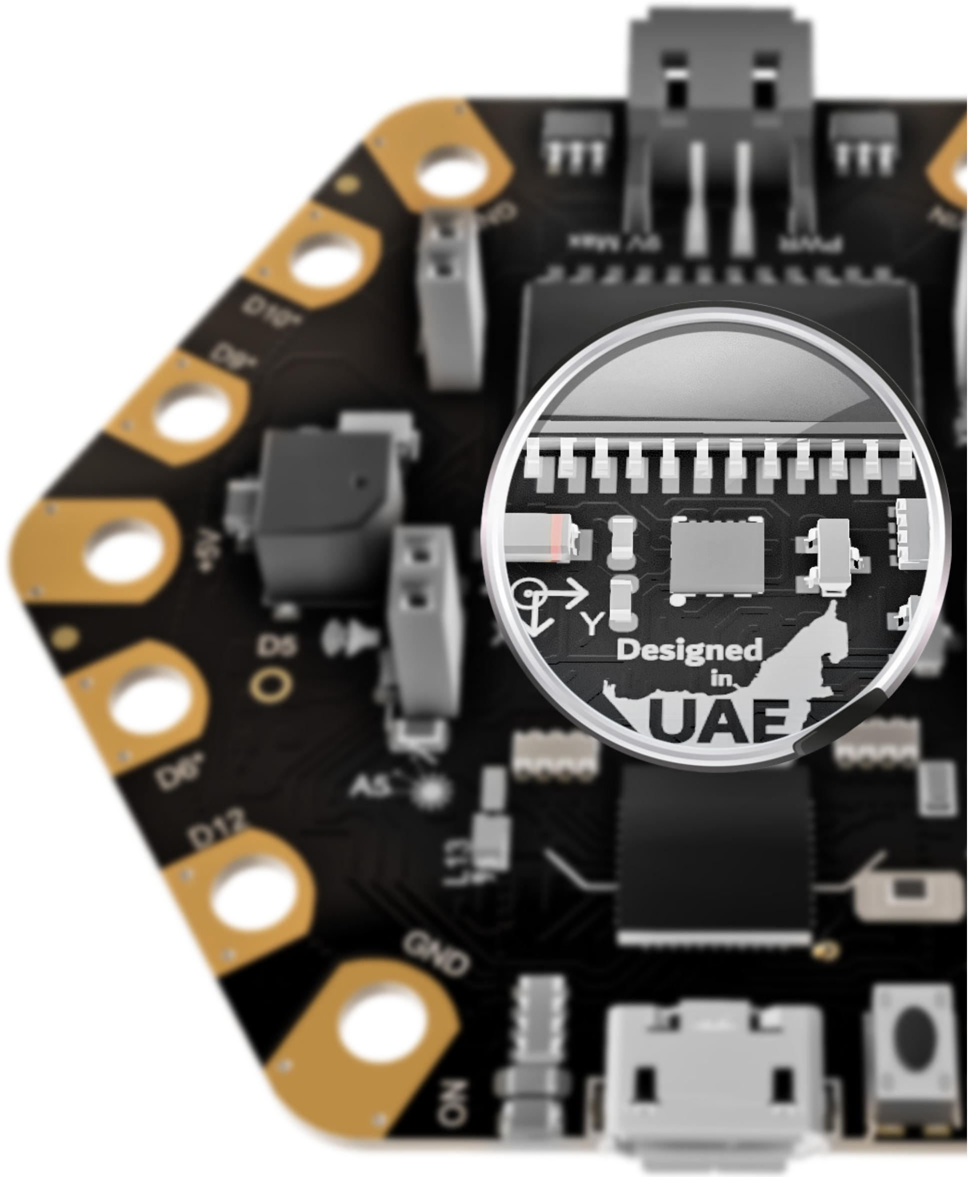
```
#include <Ibtikar_IBMaker.h>
int R;
int G;
int B;
int LED;
void setup() {
  IBMaker.begin("V0.00");  // or "V1.00"
}
void loop() {
  IBMaker.clearPixels();
  for (R = 0; R <= 255; R += 50) {
    for (G = 0; G <= 255; G += 50) {
      for (B = 0; B <= 255; B += 50) {
        for (LED = 0; LED <= 9; LED++) {
          IBMaker.setPixelColor(LED, IBMaker.colorWheel(R, G, B));
        }
        delay(100);
      }
    }
  }
}
```

You can reduce the step in each colour loop, but this will make your program take a longer time to loop over all the colours.

For each NeoPixel, how many colour combinations do you think exit?

Triple Axis Accelerometer

Accelerometers are used to measure acceleration, which is how fast something is speeding up or down. An accelerometer can measure static acceleration like gravity, which is useful in detecting tilt, like when your phone tilts. This sensor is in the middle of the board.



The accelerometer has many commands in the Maker library. The first one is used to enable or disable the sensor by passing the parameter true or false respectively.

```
IBMaker.Enable_ADXL(true);
```

Or

```
IBMaker.Enable_ADXL(false);
```

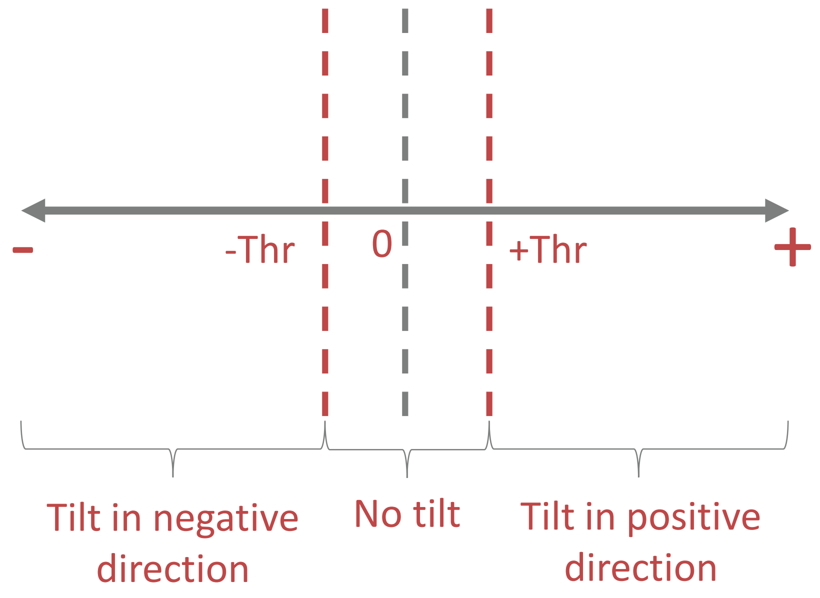
To read the acceleration value in each of the three axes (X, Y or Z), use these commands based on which axis you want.

```
IBMaker.motionX()
IBMaker.motionY()
IBMaker.motionZ()
```

The value of the accelerometer could be a positive number, a negative number or a zero. When the value of the acceleration in the X-axis is zero for example, this means the Maker is not tilted in that direction.

Sometimes, you may get a small positive or negative value even if the Maker is not tilted. To avoid this, you will need to measure this offset and compensate for it. A simple, yet effective method of finding the offset is to add or subtract a number from the measurement and check if this solves the issue. You can keep tuning the offset until you cancel its effect.

Another important topic is the sensitivity of the sensor. What if you want the board to detect the tilt after a certain value. This means when you tilt the board, you can create a threshold value, so if you exceed it, you are sure that the Maker is tilted as shown.



Note

Based on which version of the Maker you have, pay attention to the following:

If you are using the accelerometer sensor in Maker V0.0, you should not touch or use touch pads 2 and 3 because they are sharing the same data lines. If you touched them accidentally, the Maker will stop working and you will need to reset it using the Reset button on the back side. You can still use the other touch pads. Also, the sensor usually needs an offset to calibrate its values. The sensor gives raw output which is not mapped to the acceleration of gravity. Maker V1.0 has a new accelerometer where its communication lines are different from those on pin 3 and 4. This accelerometer is also calibrated which means that you do not need to add an offset to the axis you want to read. The output of the sensor is in the unit of m/s2.

Activity 23: Single Axis Tilt Detection

In this activity, you will create a program to detect the tilt in the direction of one axis. You will show the direction as an arrow (East or West) using the LED grid.

You first need to enable the accelerometer sensor, then define a threshold variable and set it to a certain value. Since the accelerometer returns float values, you need to define the variable as float as well.

In the main loop, read the sensor in the required direction. If there is an offset, you will need to compensate for it. Once the value is read and the offset is compensated for, you need to compare the measured value with the threshold value. If the value is greater than the positive threshold value, draw the arrow which corresponds to that direction. If it is smaller than the negative threshold value, draw the arrow which corresponds to the other direction. Otherwise, clear the LED grid to indicate that there is no tilt.

For Maker V0.0, the offset in the x axis is found to be around 34. The value may differ for your board. The chosen threshold is 25..

```
#include <Ibtikar_IBMaker.h>
float threshold;
float x;
void setup() {
  IBMaker.begin("V0.00");
  IBMaker.Enable_ADXL(true);
  threshold = 25.0;
}
void loop() {
  x = IBMaker.motionX() + 33.5;
  if (x > threshold) {
    IBMaker.setRow(0, 0b00100000);
    IBMaker.setRow(1, 0b00010000);
    IBMaker.setRow(2, 0b11111000);
    IBMaker.setRow(3, 0b00010000);
    IBMaker.setRow(4, 0b00100000);
  } else if (x < -1 * threshold) {
    IBMaker.setRow(0, 0b00100000);
    IBMaker.setRow(1, 0b01000000);
    IBMaker.setRow(2, 0b11111000);
    IBMaker.setRow(3, 0b01000000);
    IBMaker.setRow(4, 0b00100000);
  } else {
    IBMaker.clearDisplay(0x00);
  }
  delay(100);
}
```

For Maker V1.0, there is no need to compensate for the offset. The chosen threshold is 1. Based on the accelerometer used in this version, the X and Y axes direction is different from those in V0.0. This means to do the same activity as in V0.0, you need to read the value of the Y axis.

```
#include <Ibtikar_IBMaker.h>
float threshold;
float y;
void setup() {
  IBMaker.begin("V1.00");
  IBMaker.Enable_ADXL(true);
  threshold = 1.0;
}
void loop() {
  y = IBMaker.motionY();
  if (y > threshold) {
    IBMaker.setRow(0, 0b00100000);
    IBMaker.setRow(1, 0b00010000);
    IBMaker.setRow(2, 0b11111000);
    IBMaker.setRow(3, 0b00010000);
    IBMaker.setRow(4, 0b00100000);
  } else if (y < -1 * threshold) {
    IBMaker.setRow(0, 0b00100000);
    IBMaker.setRow(1, 0b01000000);
    IBMaker.setRow(2, 0b11111000);
    IBMaker.setRow(3, 0b01000000);
    IBMaker.setRow(4, 0b00100000);
  } else {
    IBMaker.clearDisplay(0x00);
  }
  delay(100);
}
```

Activity 24: Multiple Axis Tilt Detection (4 Directions)

In this activity, you will create a program to detect the tilt in the X and Y axes. You will show the direction as an arrow (North, East, South or West) using the LED grid. Like the previous activity, you will create a threshold value so if you exceed it, you are sure that the Maker is tilted. This threshold will be the same for both directions.

For Maker V0.0, the offset in the x axis is found to be around 34 and the offset in the y axis is found to be 85. These values may differ for your board. The chosen threshold is 25.

```
#include <Ibtikar_IBMaker.h>
float threshold;
float x;
float y;
void setup() {
  IBMaker.begin("V0.00");
  IBMaker.Enable_ADXL(true);
  threshold = 25.0;
}
void loop() {
  x = IBMaker.motionX() + 33.5;
  y = IBMaker.motionY() + 85;

  if (x > threshold) {
    IBMaker.setRow(0, 0b00100000);
    IBMaker.setRow(1, 0b00010000);
    IBMaker.setRow(2, 0b11111000);
    IBMaker.setRow(3, 0b00010000);
    IBMaker.setRow(4, 0b00100000);
    delay(100);
  } else if (x < -1 * threshold) {
    IBMaker.setRow(0, 0b00100000);
    IBMaker.setRow(1, 0b01000000);
    IBMaker.setRow(2, 0b11111000);
    IBMaker.setRow(3, 0b01000000);
    IBMaker.setRow(4, 0b00100000);
    delay(100);
  } else {
    IBMaker.clearDisplay(0x00);
  }
  if (y > threshold) {
    IBMaker.setRow(0, 0b00100000);
    IBMaker.setRow(1, 0b00100000);
    IBMaker.setRow(2, 0b10101000);
    IBMaker.setRow(3, 0b01110000);
    IBMaker.setRow(4, 0b00100000);
    delay(100);
  } else if (y < -1 * threshold) {
    IBMaker.setRow(0, 0b00100000);
    IBMaker.setRow(1, 0b01110000);
```

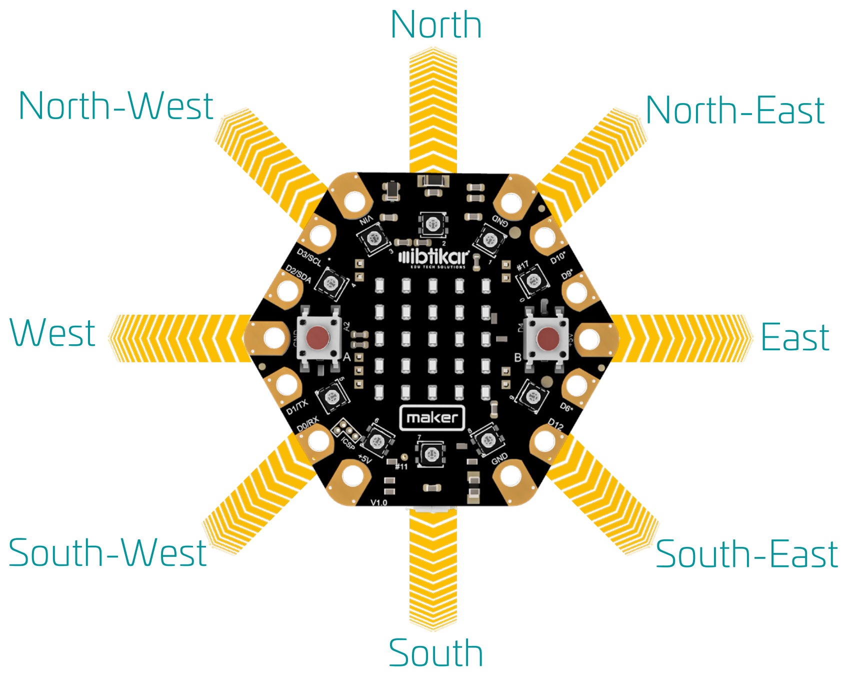
```
      IBMaker.setRow(2, 0b10101000);
      IBMaker.setRow(3, 0b00100000);
      IBMaker.setRow(4, 0b00100000);
      delay(100);
    } else {
      IBMaker.clearDisplay(0x00);
    }
  }
```

For Maker V1.0, there is no need to compensate for the offset. And like before, the chosen threshold is 1.

```
  #include <Ibtikar_IBMaker.h>
  float threshold;
  float x;
  float y;
  void setup() {
    IBMaker.begin("V1.00");
    IBMaker.Enable_ADXL(true);
    threshold = 1;
  }
  void loop() {
    x = IBMaker.motionX();
    y = IBMaker.motionY();
    if (x > threshold) {
      IBMaker.setRow(0, 0b00100000);
      IBMaker.setRow(1, 0b01110000);
      IBMaker.setRow(2, 0b10101000);
      IBMaker.setRow(3, 0b00100000);
      IBMaker.setRow(4, 0b00100000);
      delay(100);
    } else if (x < -1 * threshold) {
      IBMaker.setRow(0, 0b00100000);
      IBMaker.setRow(1, 0b00100000);
      IBMaker.setRow(2, 0b10101000);
      IBMaker.setRow(3, 0b01110000);
      IBMaker.setRow(4, 0b00100000);
      delay(100);
    } else {
      IBMaker.clearDisplay(0x00);
    }
    if (y > threshold) {
      IBMaker.setRow(0, 0b00100000);
      IBMaker.setRow(1, 0b00010000);
      IBMaker.setRow(2, 0b11111000);
      IBMaker.setRow(3, 0b00010000);
      IBMaker.setRow(4, 0b00100000);
      delay(100);
    } else if (y < -1 * threshold) {
      IBMaker.setRow(0, 0b00100000);
      IBMaker.setRow(1, 0b01000000);
      IBMaker.setRow(2, 0b11111000);
      IBMaker.setRow(3, 0b01000000);
      IBMaker.setRow(4, 0b00100000);
      delay(100);
    } else {
      IBMaker.clearDisplay(0x00);
    }
  }
```

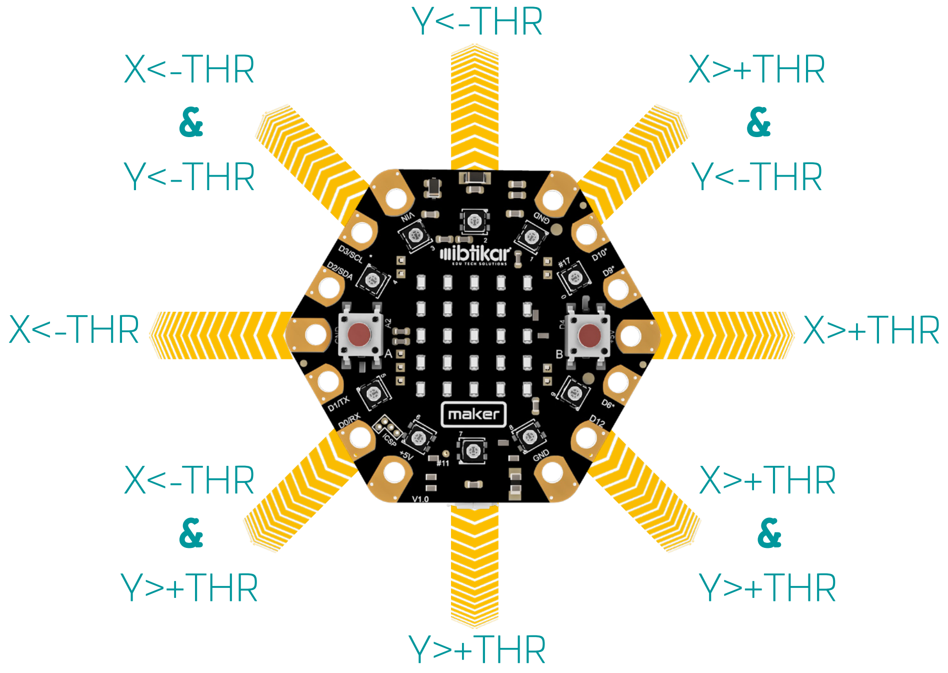
Activity 25: Multiple Axis Tilt Detection (8 Directions)

In this activity, you will create a program to detect the tilt in the X and Y axes. You will show the direction as an arrow (North, North-East, East, South-East, South, South-West, West or North-West) using the LED grid.



Like the previous activity, you will create a threshold value so if you exceed it, you are sure that the Maker is tilted. This threshold will be the same for both directions. For each direction, there is a different condition you need to check. These conditions depend on the value read from both X and Y axes.

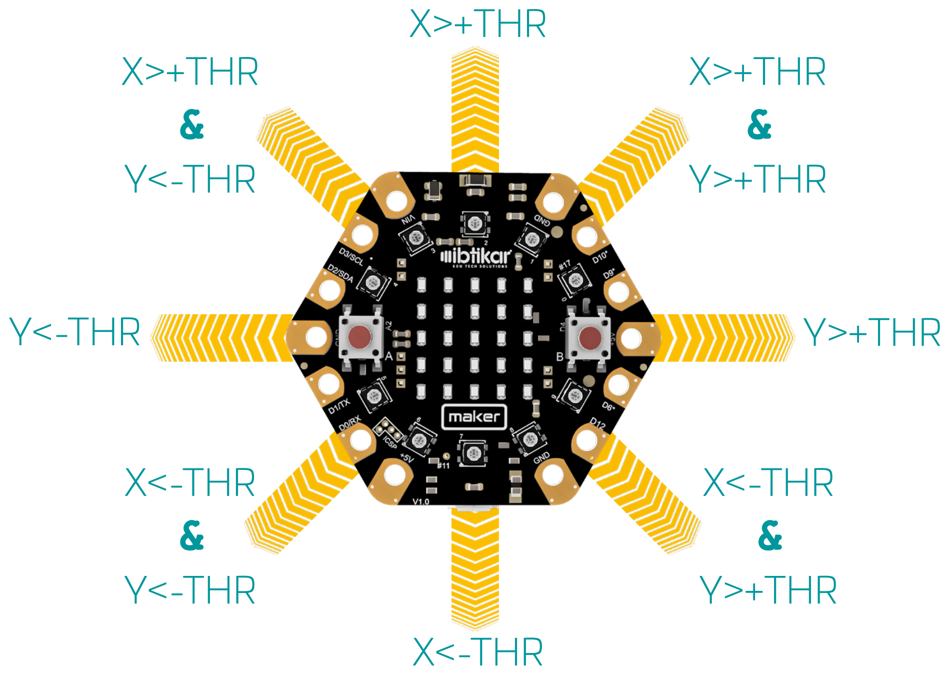
For Maker V0.0, the threshold and the offset values in the x axis and y axis are like before. The conditions are shown below based on the Accelerometer X and Y axes.



Your code will look like the following.

```
#include <Ibtikar_IBMaker.h>
float threshold;
float x;
float y;
void setup() {
  IBMaker.begin("V0.00");
  IBMaker.Enable_ADXL(true);
  threshold = 25.0;
}
void loop() {
  IBMaker.clearDisplay(0x00);
  x = IBMaker.motionX() + 33.5;
  y = IBMaker.motionY() + 85;
  if (x > threshold) {
    if (y > threshold) {
      IBMaker.setRow(0, 0b10000000);
      IBMaker.setRow(1, 0b01000000);
      IBMaker.setRow(2, 0b00101000);
      IBMaker.setRow(3, 0b00011000);
      IBMaker.setRow(4, 0b00111000);
    } else if (y < -1 * threshold) {
      IBMaker.setRow(0, 0b00111000);
      IBMaker.setRow(1, 0b00011000);
      IBMaker.setRow(2, 0b00101000);
      IBMaker.setRow(3, 0b01000000);
      IBMaker.setRow(4, 0b10000000);
    } else {
      IBMaker.setRow(0, 0b00100000);
      IBMaker.setRow(1, 0b00010000);
      IBMaker.setRow(2, 0b11111000);
      IBMaker.setRow(3, 0b00010000);
      IBMaker.setRow(4, 0b00100000);
    }
  } else if (x < -1 * threshold) {
    if (y > threshold) {
      IBMaker.setRow(0, 0b00001000);
      IBMaker.setRow(1, 0b00010000);
      IBMaker.setRow(2, 0b10100000);
      IBMaker.setRow(3, 0b11000000);
      IBMaker.setRow(4, 0b11100000);
    } else if (y < -1 * threshold) {
      IBMaker.setRow(0, 0b11100000);
      IBMaker.setRow(1, 0b11000000);
      IBMaker.setRow(2, 0b10100000);
      IBMaker.setRow(3, 0b00010000);
      IBMaker.setRow(4, 0b00001000);
    } else {
      IBMaker.setRow(0, 0b00100000);
      IBMaker.setRow(1, 0b01000000);
      IBMaker.setRow(2, 0b11111000);
      IBMaker.setRow(3, 0b01000000);
      IBMaker.setRow(4, 0b00100000);
    }
  } else {
    if (y > threshold) {
      IBMaker.setRow(0, 0b00100000);
      IBMaker.setRow(1, 0b00100000);
      IBMaker.setRow(2, 0b10101000);
      IBMaker.setRow(3, 0b01110000);
      IBMaker.setRow(4, 0b00100000);
    } else if (y < -1 * threshold) {
      IBMaker.setRow(0, 0b00100000);
      IBMaker.setRow(1, 0b01110000);
      IBMaker.setRow(2, 0b10101000);
      IBMaker.setRow(3, 0b00100000);
      IBMaker.setRow(4, 0b00100000);
    } else {
      IBMaker.clearDisplay(0x00);
    }
  }
  delay(100);
}
```

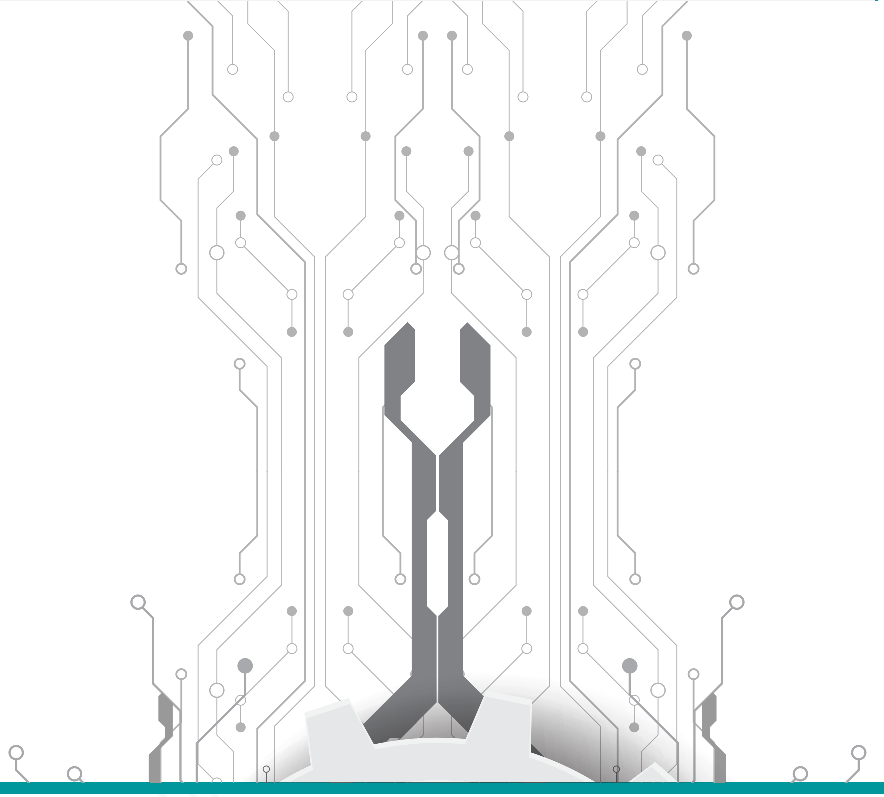
For Maker V1.0, the conditions are shown below based on the Accelerometer X and Y axes which are different to the Maker V0.0 axes.



Your code will look like the following.

```
#include <Ibtikar_IBMaker.h>

float threshold;
float x;
float y;

void setup() {
  IBMaker.begin("V1.00");
  IBMaker.Enable_ADXL(true);
  threshold = 1.0;
}

void loop() {
  IBMaker.clearDisplay(0x00);
  x = IBMaker.motionX();
  y = IBMaker.motionY();
  if (x > threshold) {
    if (y > threshold) {
      IBMaker.setRow(0, 0b00111000);
      IBMaker.setRow(1, 0b00011000);
      IBMaker.setRow(2, 0b00101000);
      IBMaker.setRow(3, 0b01000000);
      IBMaker.setRow(4, 0b10000000);
    } else if (y < -1 * threshold) {
      IBMaker.setRow(0, 0b11100000);
      IBMaker.setRow(1, 0b11000000);
      IBMaker.setRow(2, 0b10100000);
      IBMaker.setRow(3, 0b00010000);
      IBMaker.setRow(4, 0b00001000);
    } else {
      IBMaker.setRow(0, 0b00100000);
      IBMaker.setRow(1, 0b01110000);
      IBMaker.setRow(2, 0b10101000);
      IBMaker.setRow(3, 0b00100000);
      IBMaker.setRow(4, 0b00100000);
    }
  } else if (x < -1 * threshold) {
    if (y > threshold) {
      IBMaker.setRow(0, 0b10000000);
      IBMaker.setRow(1, 0b01000000);
      IBMaker.setRow(2, 0b00101000);
      IBMaker.setRow(3, 0b00011000);
      IBMaker.setRow(4, 0b00111000);
    } else if (y < -1 * threshold) {
      IBMaker.setRow(0, 0b00001000);
      IBMaker.setRow(1, 0b00010000);
      IBMaker.setRow(2, 0b10100000);
      IBMaker.setRow(3, 0b11000000);
      IBMaker.setRow(4, 0b11100000);
    } else {
      IBMaker.setRow(0, 0b00100000);
      IBMaker.setRow(1, 0b00100000);
      IBMaker.setRow(2, 0b10101000);
      IBMaker.setRow(3, 0b01110000);
      IBMaker.setRow(4, 0b00100000);
    }
  } else {
    if (y > threshold) {
      IBMaker.setRow(0, 0b00100000);
      IBMaker.setRow(1, 0b00010000);
      IBMaker.setRow(2, 0b11111000);
      IBMaker.setRow(3, 0b00010000);
```

```
      IBMaker.setRow(4, 0b00100000);
    } else if (y < -1 * threshold) {
      IBMaker.setRow(0, 0b00100000);
      IBMaker.setRow(1, 0b01000000);
      IBMaker.setRow(2, 0b11111000);
      IBMaker.setRow(3, 0b01000000);
      IBMaker.setRow(4, 0b00100000);
    } else {
      IBMaker.clearDisplay(0x00);
    }
  }
  delay(100);
}
```



Creating Your Own Functions

You may notice in the previous activities that sometimes the code gets long. And sometimes, it is difficult to remember some code syntax. This usually can be solved by creating a special function that encapsulates the lines of code and passes the parameters if needed.

In the following two activities, you will make your own functions to draw arrows and some other shapes like a smiley or sad face for example.

Activity 26: Draw Arrows

In this activity, you will create a function to draw the 8 arrows (North, North-East, East, South-East, South, South-West, West or North-West) using the LED grid.

A function called **drawArrow** takes an argument with the direction needed. It will then construct a two-dimensional (2D) array with all the binary values representing all the arrows. Each row of this matrix represents an arrow with a specific direction. Based on the parameter you pass; the if statement decides which row (arrow) to display. In the main loop, the arrows will be displayed one by one with a duration of 250 milliseconds.

You can see that the function you created can be reused in other programs, so you do not need to lose time writing it again.

```
#include <Ibtikar_IBMaker.h>
int Period = 250;

void setup() {
  IBMaker.begin("V0.00");  // or "V1.00"
}

void loop() {
  drawArrow("N");
  delay(Period);
  drawArrow("NE");
  delay(Period);
  drawArrow("E");
  delay(Period);
  drawArrow("SE");
  delay(Period);
  drawArrow("S");
  delay(Period);
  drawArrow("SW");
  delay(Period);
  drawArrow("W");
  delay(Period);
  drawArrow("NW");
  delay(Period);
}

void drawArrow(String Dir) {
  int j;
  byte dirArray[][5] = {
    {B00111000 , B00011000 , B00101000 , B01000000 , B10000000},
    {B11100000 , B11000000 , B10100000 , B00010000 , B00001000},
    {B00100000 , B01110000 , B10101000 , B00100000 , B00100000},
    {B10000000 , B01000000 , B00101000 , B00011000 , B00111000},
    {B00001000 , B00010000 , B10100000 , B11000000 , B11100000},
    {B00100000 , B00100000 , B10101000 , B01110000 , B00100000},
    {B00100000 , B00010000 , B11111000 , B00010000 , B00100000},
    {B00100000 , B01000000 , B11111000 , B01000000 , B00100000}
  };

  if (Dir == "NE")       // NorthEast
    j = 0;
  else if (Dir == "NW")  // NorthWest
    j = 1;
  else if (Dir == "N")   // North
    j = 2;
  else if (Dir == "SE")  // SouthEast
    j = 3;
  else if (Dir == "SW")  // SouthWest
    j = 4;
  else if (Dir == "S")   // South
    j = 5;
  else if (Dir == "E")   // East
    j = 6;
  else if (Dir == "W")   // West
    j = 7;

  for (int i = 0; i <= 4; i++) {
    IBMaker.setRow(i, dirArray[j][i]);
  }
}
```

Activity 27: Draw Shapes

In this activity, you will create a function to draw 6 shapes (a rectangle, a square, a diamond, a smiley face, a neutral face and a sad face) using the LED grid.

A function called **drawShape** takes an argument with the shape needed. It will then construct a two-dimensional (2D) array with all the binary values representing all the shapes. Each row of this matrix represents a certain shape. Based on the parameter you pass; the if statement decides which row (shape) to display. In the main loop, the three faces will only be displayed one by one with a duration of 500 milliseconds.

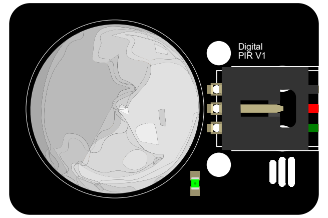
Like the previous activity, you can reuse this function in other programs, so you do not need write it again.

```
#include <Ibtikar_IBMaker.h>
int Period = 500;
void setup() {
  IBMaker.begin("V0.00");   // or "V1.00"
}

void loop() {
  drawShape(":)");
  delay(Period);
  drawShape(":|");
  delay(Period);
  drawShape(":(");
  delay(Period);
}
void drawShape(String Shape) {
  int j;
  byte shapeArray[][5] = {
    {B00000000 , B11111000 , B10001000 , B11111000 , B00000000},
    {B11111000 , B10001000 , B10001000 , B10001000 , B11111000},
    {B00100000 , B01010000 , B10001000 , B01010000 , B00100000},
    {B00000000 , B01010000 , B00000000 , B10001000 , B01110000},
    {B00000000 , B01010000 , B00000000 , B11111000 , B00000000},
    {B00000000 , B01010000 , B00000000 , B01110000 , B10001000}
  };
  if (Shape == "Rec")        // Rectangle
    j = 0;
  else if (Shape == "Sqr")   // Square
    j = 1;
  else if (Shape == "Dmd")   // Diamond
    j = 2;
  else if (Shape == ":)")    // Smiley Face
    j = 3;
  else if (Shape == ":|")    // Neutral Face
    j = 4;
  else if (Shape == ":(")    // Sad Face
    j = 5;

  for (int i = 0; i <= 4; i++) {
    IBMaker.setRow(i, shapeArray[j][i]);
  }
}
```

Adding More Sensors

Using the Maker touch pads and the crocodile cables that come with the Maker, you can interface more components. This means that if you need a sensor that does not come with the standard Maker board, say a motion sensor as an example, you can interface it easily to Maker.

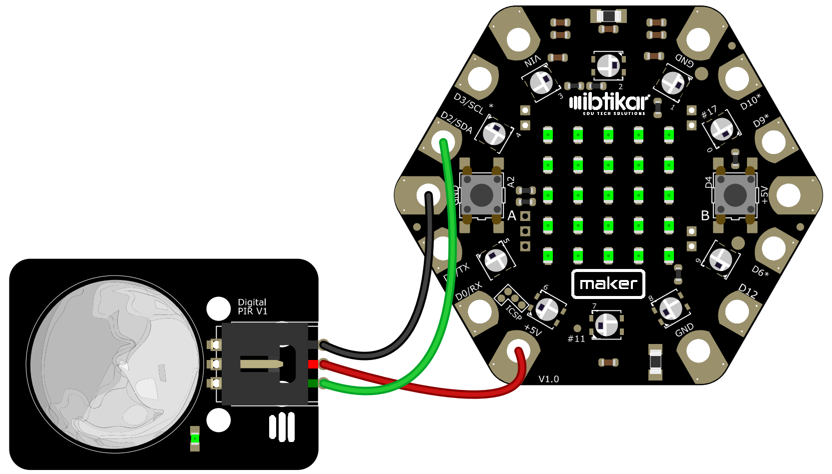
Activity 28: Infrared Motion Sensor



In this activity, an infrared motion sensor is interfaced to the Maker board and based on its digital input value, a message is displayed on the Serial Monitor.

A PIR sensor (Passive Infrared) gets activated when it is exposed to heat from bodies that emit energy, like humans and animals. The PIR sensor compares the new signal with the previous signal. A change between the readings, means there is motion.

Using the crocodile-to-female cables, connect the sensor to the Maker as shown.



Upload the code to the Maker board.

```
int PIR_pin = 2;
void setup() {
  Serial.begin(9600);
  pinMode(PIR_pin, INPUT);
}

void loop() {
  int PIR_state = digitalRead(PIR_pin);
  if (PIR_state == 1)
   Serial.println("Motion is detected!!");
  else
   Serial.println("Nothing is moving.");
  delay(100);
}
```

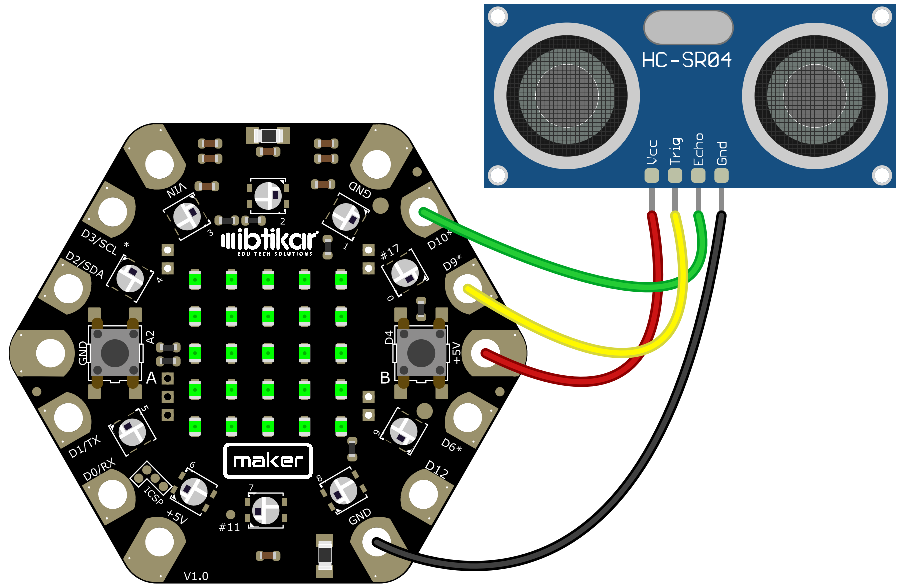
After uploading the code to your Maker, open the Serial Monitor. Test the motion sensor by passing your hand over the top of the sensor (do not touch the sensor). The PIR sensor updates its output signal based on the variations in the infrared signal from the objects. Now keep your hand stationary over the sensor and notice the difference.

Activity 29: Ultrasonic Sensor

In this activity, an ultrasonic sensor is interfaced to the Maker board and based on its digital input value representing the distance, a message is displayed on the Serial Monitor showing the distance in centimeters. The LED grid will also display a face based on the distance. If the distance is less than 30cm, a happy face will be displayed. If the distance is between 30 and 50cm, a neutral face will be displayed. If the distance is greater, a sad face will be displayed.

The HC-SR04 ultrasonic sensor uses sonar to determine distance to an object. The transmitter (trig pin) sends a high-frequency sound signal. When the signal hits an object, it is reflected and the transmitter (echo pin) receives it. By measuring the time it takes the signal to reach the object and goes back to the sensor and by knowing the sound speed in the air, the distance can be found.

Using the crocodile-to-female cables, connect the sensor to the Maker as shown.



Upload the code to the Maker board.

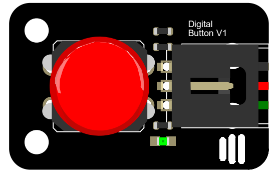
Put an object close to the sensor and see the output. Now, start moving the object away from the sensor and notice the change in the Serial Monitor and on the LED grid.

```
#include <Ibtikar_IBMaker.h>

// defines pins numbers
const int trigPin = 9;
const int echoPin = 10;

void setup() {
  IBMaker.begin("V0.00");   // or "V1.00"
  Serial.begin(9600);       // Starts the serial communication
  pinMode(trigPin, OUTPUT); // Sets the trigPin as an Output
  pinMode(echoPin, INPUT);  // Sets the echoPin as an Input
}

void loop() {
  int distance = Distance_in_CM(trigPin, echoPin);
  Serial.print("Distance: ");
  Serial.println(distance);

  if (distance <= 30)
    drawShape(":)");
  else if (distance > 30 && distance <= 50)
    drawShape(":|");
  else if (distance > 50)
    drawShape(":(");

delay(20);
}

void drawShape(String Shape) {
  int j;
  byte shapeArray[][5] = {
    {B00000000 , B01010000 , B00000000 , B10001000 , B01110000},
    {B00000000 , B01010000 , B00000000 , B11111000 , B00000000},
    {B00000000 , B01010000 , B00000000 , B01110000 , B10001000}
};

  if (Shape == ":)")       // Smiley Face
    j = 0;
  else if (Shape == ":|")  // Neutral Face
    j = 1;
  else if (Shape == ":(")  // Sad Face
    j = 2;

  for (int i = 0; i <= 4; i++) {
    IBMaker.setRow(i, shapeArray[j][i]);
  }
}

int Distance_in_CM(int trig, int echo) {
  digitalWrite(trig, LOW);   // Clears the trigPin
  delayMicroseconds(2);
  // Sets the trig on HIGH state for 10 micro seconds
  digitalWrite(trig, HIGH);
  delayMicroseconds(10);
  digitalWrite(trig, LOW);
  // Reads the echo, returns the sound wave travel time in microseconds
  long duration = pulseIn(echo, HIGH);
  int distance = duration * 0.034 / 2;   // Calculating the distance
  return distance;
}
```

Using an Interrupt

An Interrupt makes the processor respond quickly to important events. When a certain signal is detected, the Interrupt interrupts whatever the processor is doing, and executes some code. Once that code has finished, the processor goes back to where it was originally. The interrupt service routine (ISR) should be as short as possible with no input variables or returned values. This means all changes must be made on global variables.

Activity 30: Interrupt the Text Scrolling

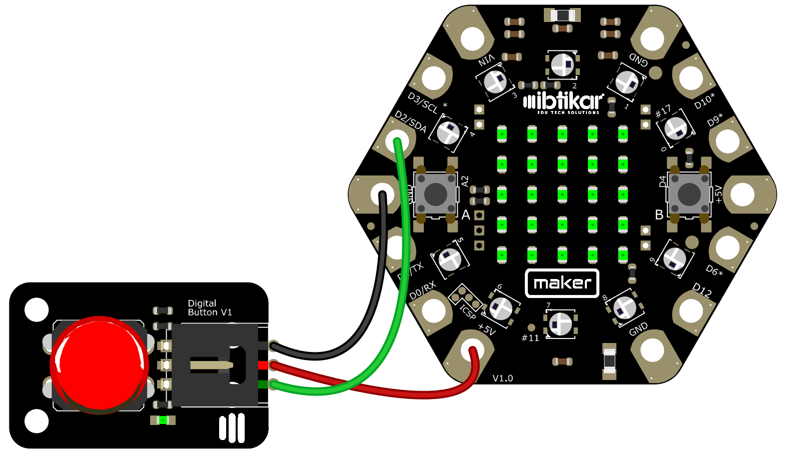


To understand the interrupt concept, program your Maker board to scroll a long text and then try to press an external button module. The button should beep the buzzer. If you hear a beep during the text scrolling process, then you have succeeded. Otherwise, the button you have does not interrupt the scrolling process and no interrupt has happened.

The reason you need an external button instead of the two on-board buttons, is to connect it to one of the interrupt pins on the Maker board. The following table shows the pin-interrupt mapping.

Pin	Interrupt
D0/RX	INT2
D1/TX	INT3
D2	INT1
D3	INT0

Using the crocodile-to-female cables, connect the sensor to the Maker as shown.



Upload the following code to the Maker board. This code has no interrupt.

```
#include <Ibtikar_IBMaker.h>

const int buttonPin = 2;        // the number of the pushbutton pin
int buttonState = 0;            // variable for reading the pushbutton status

void setup() {
  IBMaker.begin("V0.00");       // or V1.00
  pinMode(buttonPin, INPUT);
}

void loop() {
  IBMaker.Leds_Str("This text is ", 100);
  IBMaker.Leds_Str("scrolling to ", 100);
  IBMaker.Leds_Str("test the interrupt", 100);
  IBMaker.Leds_Str("activity", 100);

  buttonState = digitalRead(buttonPin);
  if (buttonState) {
    IBMaker.playTone(400, 250);
  }
}
```

After uploading the code to your Maker, click the button and check if you hear a beep. You will notice that it may take 30 seconds to hear a beep if you were lucky to press the button in the correct time.

Now, upload this code to the Maker which has an interrupt (INT1) on pin 2.

```
#include <Ibtikar_IBMaker.h>

const int buttonPin = 2;        // the number of the pushbutton pin
volatile int buttonState = 0;   // variable for reading the pushbutton status

void setup() {
  IBMaker.begin("V0.00");       // or V1.00
  pinMode(buttonPin, INPUT);
  attachInterrupt(1, pin_ISR, CHANGE);
}

void loop() {
  IBMaker.Leds_Str("This text is ", 100);
  IBMaker.Leds_Str("scrolling to ", 100);
  IBMaker.Leds_Str("test the interrupt", 100);
  IBMaker.Leds_Str("activity", 100);
}

void pin_ISR() {
  buttonState = digitalRead(buttonPin);
  if (buttonState) {
    IBMaker.playTone(400, 250);
  }
}
```

After uploading the code to your Maker, click the button and check if you hear a beep. You will notice that the buzzer beeps at any time even during the text scrolling.

Activity 31: Accelerometer Interrupts

In addition to measuring the acceleration in the three axes as in Activity 31, the accelerometer can also generate an interrupt when certain events happen. These events can be a single tap or double tap on the board, activity/inactivity of the sensor, or a free fall of the board.

In Maker V0, the type of the accelerometer is ADXL345 and the Ibtikar Maker library uses a special library for this sensor created by *SparkFun Electronics*. This accelerometer can generate interrupts for single or double taps or for activity/inactivity of the sensor.

In the following code, you can choose the interrupt you like. You can change the sensor range or the taps threshold which controls the sensitivity of the tap. You can even choose which axis to detect taps or activity on it.

```
#include <Ibtikar_IBMaker.h>

int state = 0;

void setup() {
  delay(2000);
  IBMaker.begin("V0.00");
  IBMaker.Enable_ADXL(true);
  Serial.begin(9600);

  ADXL_Config();
}

void loop() {
  ADXL_ISR();
  delay(10);
}
```

```
int ADXL_ISR() {
  byte interrupts = IBMaker.adxl.getInterruptSource();
  int state_old = state;
  if (IBMaker.adxl.triggered(interrupts, ADXL345_DOUBLE_TAP)) {
    state = 1;
    Serial.println("*** Double Tap - Maker V0 ***");
  }
  else if (IBMaker.adxl.triggered(interrupts, ADXL345_SINGLE_TAP)) {
    state = 2;
    Serial.println("*** Single Tap - Maker V0 ***");
  }
  else if (IBMaker.adxl.triggered(interrupts, ADXL345_ACTIVITY)) {
    state = 3;
    Serial.println("*** Activity - Maker V0 ***");
  }
  else if (IBMaker.adxl.triggered(interrupts, ADXL345_INACTIVITY)) {
    state = 4;
    Serial.println("*** Inactivity - Maker V0 ***");
  }
  if (state_old == state) {
    state = 0;
  }
}
void ADXL_Config() {
  IBMaker.adxl.powerOn();  // Power on the ADXL345

  // The range settings:2g,4g,8g or 16g
  // Higher val = wider measurement range
  // Lower val = greater sensitivity
  IBMaker.adxl.setRangeSetting(16);

  // Detect activity in all the axes(X, Y, Z):(1 = ON, 0 = OFF)
  IBMaker.adxl.setActivityXYZ(1, 1, 1);
  // Set activity threshold (0-255):62.5mg/increment
  IBMaker.adxl.setActivityThreshold(75);

  // Detect inactivity in all the axes(X, Y, Z):(1 = ON, 0 = OFF)
  IBMaker.adxl.setInactivityXYZ(1, 1, 1);

  // Set inactivity threshold (0-255):62.5mg/increment
  IBMaker.adxl.setInactivityThreshold(75);

  // How many seconds of no activity is inactive
  IBMaker.adxl.setTimeInactivity(8);

  // Detect taps in the directions turned ON(X, Y, Z) (1 = ON, 0 = OFF)
  IBMaker.adxl.setTapDetectionOnXYZ(0, 0, 1);

  // Set values for what is considered a TAP or a DOUBLE TAP (0-255)
  IBMaker.adxl.setTapThreshold(130);           // 62.5 mg per increment
  IBMaker.adxl.setTapDuration(15);             // 625 µs per increment
  IBMaker.adxl.setDoubleTapLatency(80);        // 1.25 ms per increment
  IBMaker.adxl.setDoubleTapWindow(200);        // 1.25 ms per increment

  // Setting all interupts to take place on INT1 pin
  IBMaker.adxl.setImportantInterruptMapping(1, 1, 1, 1, 1);

  // Turn on Interrupts for each mode (1 == ON, 0 == OFF)
  IBMaker.adxl.InactivityINT(1);
  IBMaker.adxl.ActivityINT(1);
  IBMaker.adxl.doubleTapINT(1);
  IBMaker.adxl.singleTapINT(1);
}
```

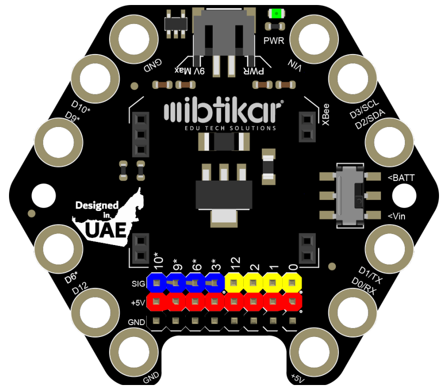
In Maker V1, the accelerometer can generate interrupts for single or double taps or a free fall of the board. The type of the accelerometer is LIS3DH and the Ibtikar Maker library uses a special library for this sensor created by *Adafruit Industries*.

In the following code, you can detect one of the three interrupt types by choosing the mode value. Based on which mode you choose; the interrupt will be configured. You can change the sensor range or the taps threshold which controls the sensitivity of the tap.

```
#include <Ibtikar_IBMaker.h>

void setup() {
  Serial.begin(9600);
  IBMaker.begin("V1.00");
  IBMaker.Enable_ADXL(true);

  IBMaker.setAccelRange(LIS3DH_RANGE_2_G);   // 2, 4, 8 or 16 G!

  // Choose the AXL interrupt mode:
  // 0 = turn off click detection & interrupt
  // 1 = single click only interrupt output
  // 2 = double click only interrupt output, detect single click
  // 3 = free fall only interrupt output, free fall
  // Adjust threshold, higher numbers are less sensitive
  int mode = 1;

  byte CLICKTHRESHHOLD = 100;

  if (mode == 1) {
    IBMaker.setAccelTap(mode, CLICKTHRESHHOLD);
    attachInterrupt(digitalPinToInterrupt(IBPIN_ACCE_INTE_V1), tapTime_single, CHANGE);
  } else if (mode == 2) {
    IBMaker.setAccelTap(mode, CLICKTHRESHHOLD);
    attachInterrupt(digitalPinToInterrupt(IBPIN_ACCE_INTE_V1), tapTime_double, CHANGE);
  } else if (mode == 3) {
    IBMaker.setAccelTap(mode);
    attachInterrupt(digitalPinToInterrupt(IBPIN_ACCE_INTE_V1), tapTime_freefall, CHANGE);
  } else {
    IBMaker.setAccelTap(0);
  }
}

void loop() {
}

void tapTime_single(void) {
  Serial.println("*** Single Tap - Maker V1 ***");
  IBMaker.clear_event();
}
void tapTime_double(void) {
  Serial.println("*** Double Tap - Maker V1 ***");
  IBMaker.clear_event();
}
void tapTime_freefall(void) {
  Serial.println("*** Free Fall - Maker V1 ***");
  IBMaker.clear_event();
}
```

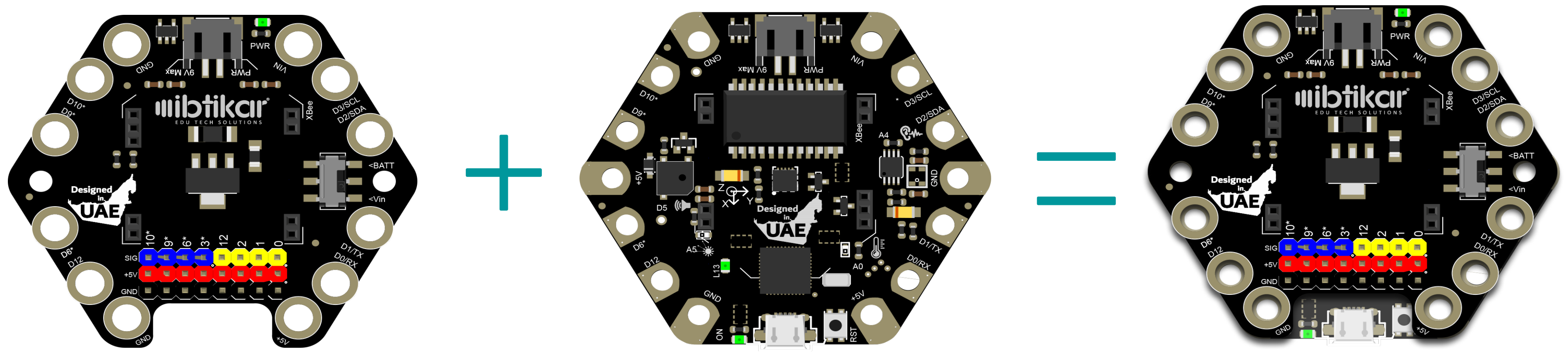
Maker Servo/Expansion Shield

Why Do We Need It?



While building any electronic system, you might need to add other input and output modules to your system. Sometimes the module you need to add, like a big servo motor, requires more power than what the Maker can supply. Supplying the servo motor from the Maker directly can burn the board.

To make it easy to connect these modules to the Maker board, you need to use the Maker Servo/Expansion Shield. This shield works as a bridge between the Maker and the other modules and allows to you supply them with an external power supply.



After connecting the Maker Expansion Shield onto the back of the Maker board you can use the other modules easily (input and output). Input is when the microcontroller receives a signal from an input module (a sensor), and output is when the microcontroller sends a signal to an output module (an actuator).

On the shield, there are different pin categories: the red and black pins stand for the 5V and ground respectively. The yellow and blue pins are explained in the following table.

Pin	Digital (I/O)	Analog (I)	Analog-PWM (O)
D0	0		
D1	1		
D2	2		
D3	3		*
D6	6	A7	*
D9	9	A9	*
D10	10	A10	*
D12	12	A11	

Dealing with Digital Signals

All the pins on the shield can be used as digital I/O pins based on the defined pin mode in the code, input or output. These pins can be defined directly using their numbers in the IDE code as input or output.

Dealing with Analog Signals

For the analog I/O, there are specific pins used to send signals and others used to receive. Unlike the digital pins, the same pin cannot be used for sending or receiving analog signals.

Pins D6, D9, D10, D12 are used to read analog signals as A7, A9, A10 and A11 respectively while the pins included within the PWM category are used to send a signal that gives an analog behaviour to the actuator. These are D3, D6, D9 and D10.

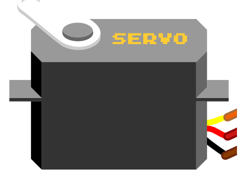
Powering the Shield

The shield has a sliding switch that allows you to either power the board from the external connecter of the shield or from the external connector of the Maker itself. **The power from the micro USB is not connected to the 5V pins on the shield.** This means that even if the Maker board is running via the USB cable, there is still no power in the shield regardless of the slide switch position.

If you want to use some of the modules which do not require a lot of power, you can use a female-to-crocodile cable and connect the female pin to one of the 5V pins on the shield. The other side of the cable connects to the 5V pad on the Maker (near button B). Do this at your own risk.

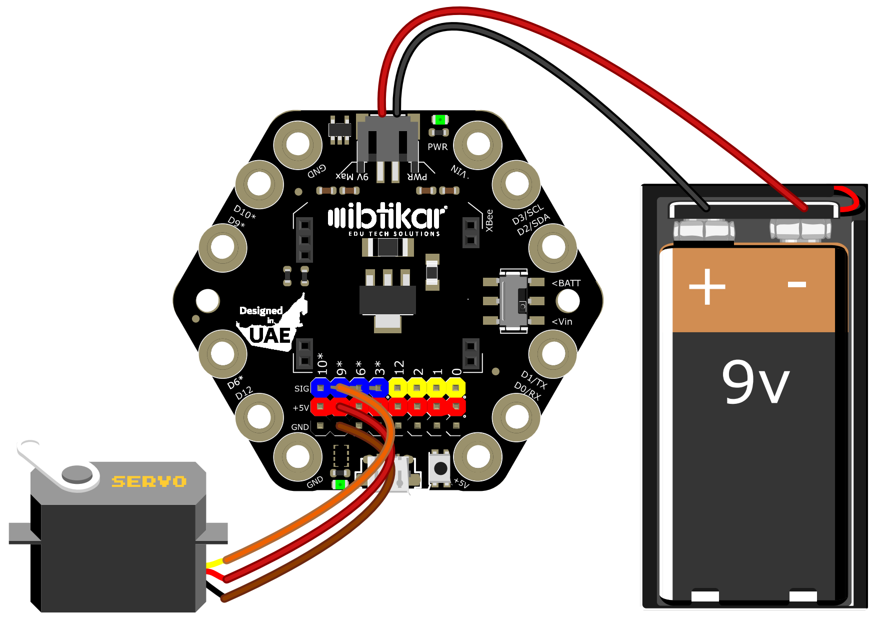
Activities

Activity 32: 9g Micro Servomotor



A servomotor is a rotary actuator that allows for exact rotary positioning. It consists of an electric motor with a feedback system to detect the position. The servo motor that comes in the Ibtikar Discovery Kit has a limited range of 0-180 degrees of rotation.

Connect the servo motor to the Maker as shown.

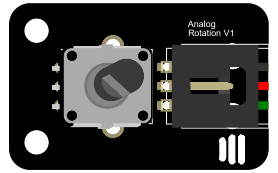


Upload the code to the Maker board.

```
#include <Servo.h>
Servo myservo;

int pos = 0;

void setup() {
  myservo.attach(9);
}

void loop() {
  for (pos = 0; pos <= 180; pos += 1) {
    myservo.write(pos);
    delay(20);
  }
  for (pos = 180; pos >= 0; pos -= 1) {
    myservo.write(pos);
    delay(20);
  }
}
```

After uploading the code to your Maker, the servo motor will start rotating from 0-180 degrees and then go back to 0 degrees.

Activity 33: Analog Rotation Sensor

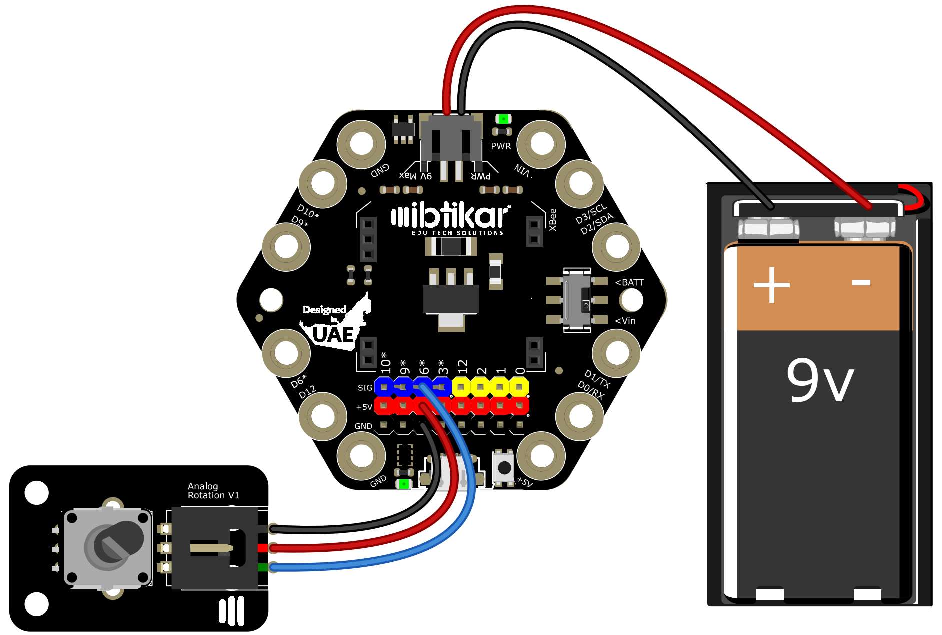


In this activity, we will interface an analog rotation sensor to the Maker board and display the analog input value on the Serial Monitor. Since this is a sensor with an analog value, then it must be interfaced with one of the Maker analog input pins.

An analog rotation sensor (or a potentiometer) consists of a fixed value resistor and a rotating wiper. Manipulating the wiper will divide the fixed value resistor into two parts. Its main use is as a variable resistor or voltage divider.

If you recall the table with all the pin pads functions, you can see that 4 pins can be used to read an analog sensor. These are D6, D9, D10 and D12. It is important to use the correct pin name when dealing with analog since D6 is A7 and D12 is A11.

Connect the potentiometer to the Maker as shown.

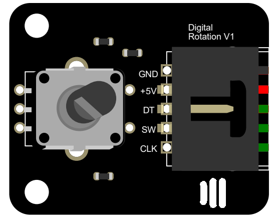


Upload the code to the Maker board.

```
void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
}
void loop() {
  int sensorValue = analogRead(A7);
  float voltage = sensorValue * (5.0 / 1023.0);

  // print out the value you read:
  Serial.print(sensorValue);
  Serial.print(" , ");
  Serial.println(voltage);

  delay(10);  // delay between reads for stability
}
```

After uploading the code to your Maker, open the Serial Monitor, then rotate the analog rotation sensor. The Serial Monitor displays two values with a comma separating them. The first value is the value of the rotational sensor read by the analog to digital converter (ADC) module in the Maker which varies between 0 and 1023. The second value is the mapped voltage with respect to the ADC value since the 0 read by the ADC corresponds to 0V and the maximum value which is 1023 corresponds to 5V.

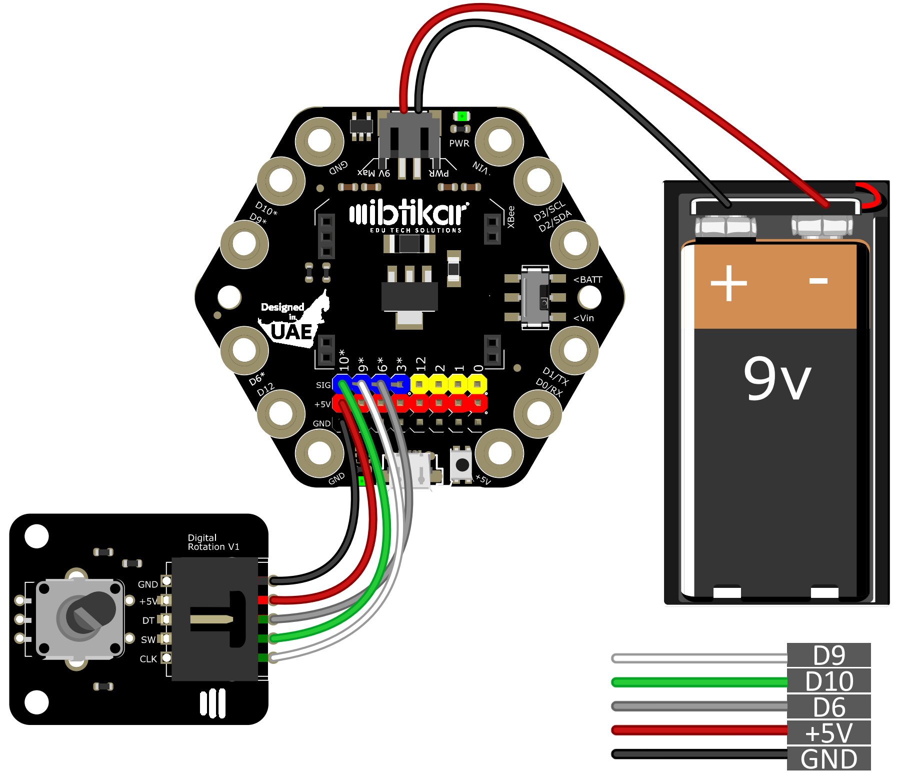
Activity 34: Digital Rotation Sensor



In this activity, a digital rotation sensor (encoder) is interfaced to the Maker board and based on its digital input value, a counter will increment or decrement its value on the Serial Monitor.

A digital rotation sensor (or an encoder) is an electro-mechanical device that senses the rotary motion (position and speed for instance) from an axis and then sends an electrical signal to the microcontroller.

Connect the encoder sensor to the Maker as shown.

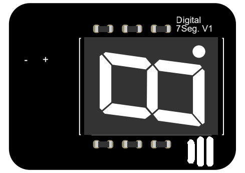


Upload the code to the Maker board.

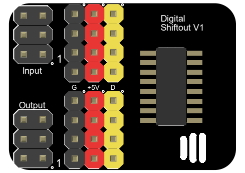
```
void setup() {
  pinMode(6, INPUT);
  pinMode(9, INPUT);
  pinMode(10, INPUT);
  Serial.begin(9600);
}

int count = 0;
boolean A, B;

void loop() {
  if (digitalRead(10) == HIGH) {
    count = 0;
    Serial.println(count);
    delay(100);
  }

  ReadEnc();
  while (A == 1 && B == 0) {
    ReadEnc();
    if (A == 1 && B == 1) {
      count--;
      Serial.println(count);
      break;
    }
  }

  while (B == 1 && A == 0) {
    ReadEnc();
    if (A == 1 && B == 1) {
      count++;
      Serial.println(count);
      break;
    }
  }
}

void ReadEnc (void) {
  A = digitalRead(6);
  B = digitalRead(9);
  delay(1);
}
```

After uploading the code to your Maker, open the Serial Monitor, then rotate the digital rotation sensor clockwise and counter clockwise. The Serial Monitor displays the value of the rotational sensor. You can reset the value of the counter to 0 by pressing the knob itself.

Activity 35: 7-Segment & Shift-Out Modules



7-Segment



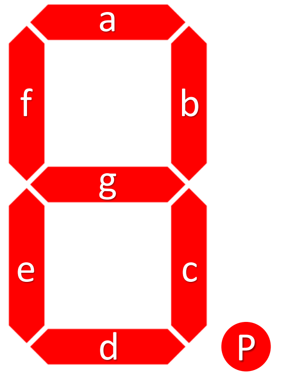
Shift-Out

The 7-segment display has 8 embedded LEDs combined into one package as one digital digit; 7 for the segments and 1 for the decimal point. To control all LEDs, you need 8 pins from the Maker. Luckily, you can use the shift-out module.

The shift-out module has an electronic chip that controls a certain number of parallel outputs using binary signals. It can control many devices with less input wires. The module can control up to 8 outputs using only 3 inputs which makes it an ideal solution to control the 7-segment display with 3 wires instead of 8. The shift-out module outputs a Byte of data (8 Bits), where each bit can either be 0 or 1.

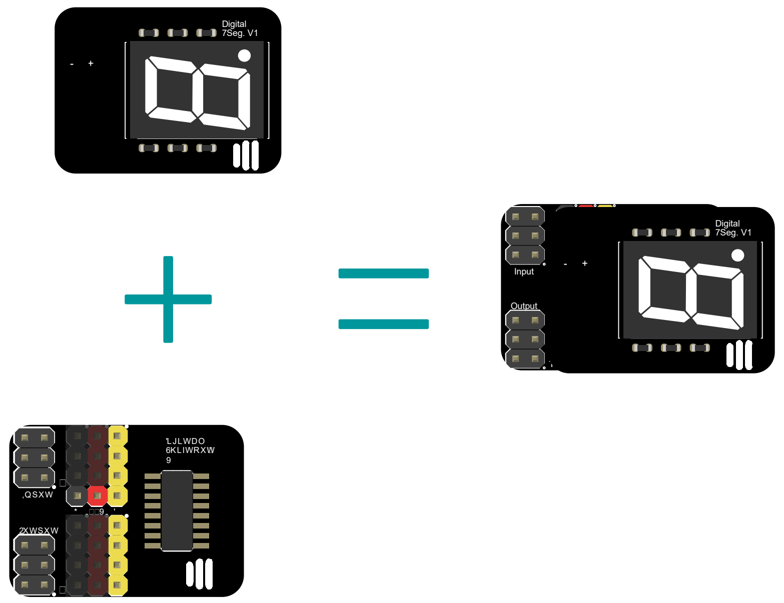
The shift-out module has an input socket and an output socket. The input socket will be connected to Maker. If it happens that you have another 7-segment and shift-out module, you can drive them using the same pins by connecting the output socket of the first one to the input socket of the second one.

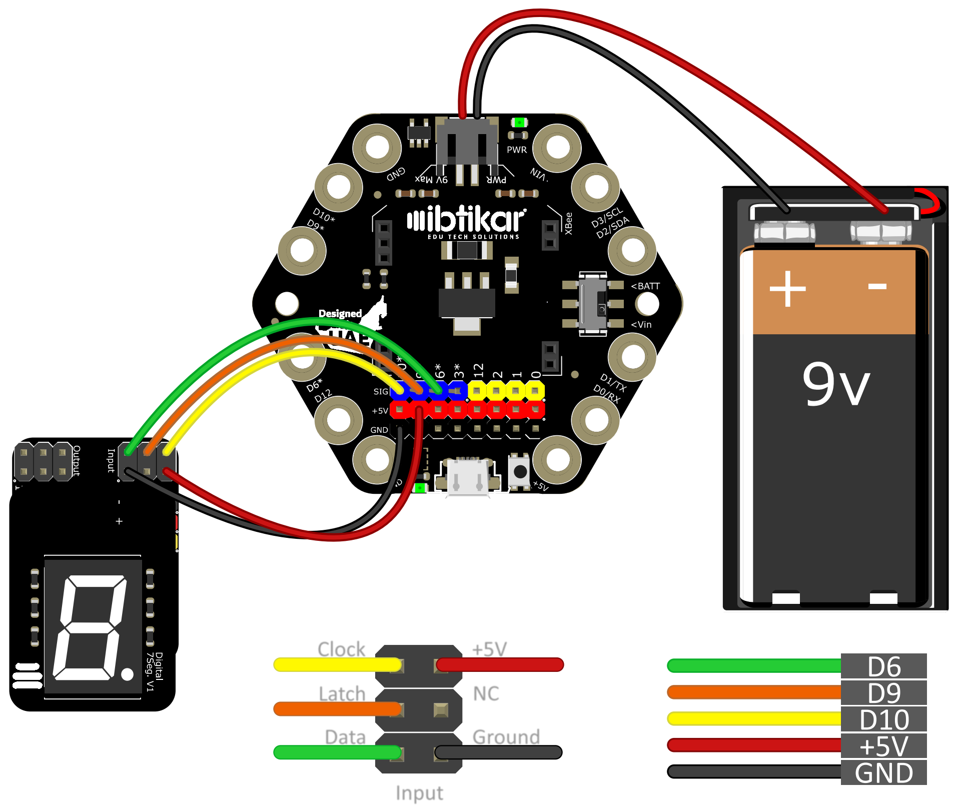
In the 7-segment display, each LED receives its signal from the shift-out module in a binary form 0 or 1. For example, if the displayed number is 1 it means there are only two LEDs which are ON since the number 1 has two segments only (segments b and c).



The Ibtikar 7-segment which comes in the Discovery Kit has a special hardware connection that makes the 8 embedded LEDs work when sending logic 0 to them not 1. This connection is called the Common Anode. This means if number 1 is to be displayed, the binary representation will be 11111001 since the order of the 8 segments which needs to be sent is Pgfedcba.

Attach the 7-segment module on top of the shift-out module. The 7-segment module pins must match the coloured pins of the shift-out module as shown.





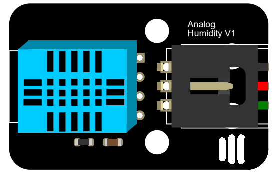
Use the jumper wires to connect the shift-out module to the Maker Expansion shield per the following input pin assignment: Wire the 3 input signals to any digital pin on the shield. For the power lines use any VCC and GND pin on the shield.

Upload the code to the Maker board.

```
int clockPin = 10;
int latchPin = 9;
int dataPin = 6;

byte numberArray[] = {
  B11000000, B11111001, B10100100, B10110000, B10011001,
  B10010010, B10000010, B11111000, B10000000, B10011000  };

void setup() {
  pinMode(latchPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
}

void loop() {
  for (int i = 0; i <= 9; i++) {
    digitalWrite(latchPin, LOW);
    // shift the bits out:
    shiftOut(dataPin, clockPin, MSBFIRST, numberArray[i]);
    // turn on the output so the LEDs can light up:
    digitalWrite(latchPin, HIGH);
    delay(500);
  }
}
```

After uploading the code to your Maker, the 7-Segment module will display the numbers from 0 to 9 with a delay of half a second. This will repeat forever.

Activity 36: Temperature and Humidity Sensor (DHT 11)

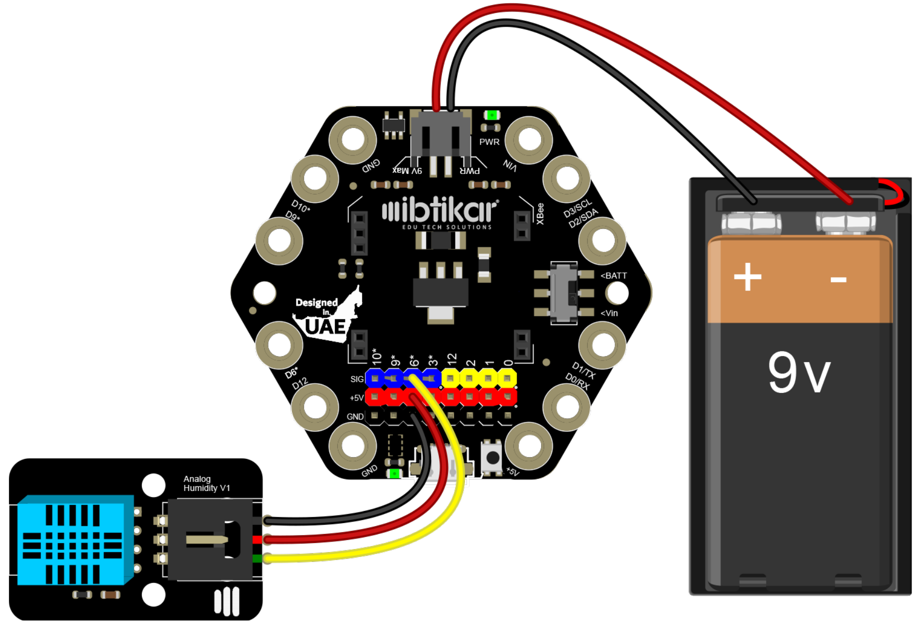


This sensor is used for measuring the temperature and the amount of the humidity in the air. The sensor sends the two values at the same time. The sampling rate for this sensor is 1 Hz or 1 reading per second.

Before using this module, you need to add its library. Without the library, you cannot use this sensor as it is not included by default in the Arduino IDE.

To import this library to the Arduino IDE, go to Sketch > Include Library > Add .ZIP Library ... > browse your computer files then select the library .ZIP extension (DHT-lib.zip). This library can be downloaded from different sites on the internet.

Connect the temperature and humidity sensor to the Maker as shown.



Upload the code to the Maker board.

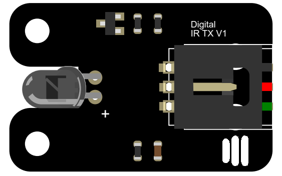
```
#include <dht.h>
dht DHT;
#define DHT11_PIN 6

void setup() {
  Serial.begin(9600);
  while (!Serial);
  Serial.println("Humidity (%),\tTemperature (C)");
}

void loop() {
  DHT.read11(DHT11_PIN);
  Serial.print(DHT.humidity, 1);
  Serial.print(",\t");
  Serial.println(DHT.temperature, 2);
  delay(1000);
}
```

Once you download this code to your Maker, open the Serial Monitor to read the temperature and humidity values. Blow air on the sensor and notice the change in the values. You also can use the Serial Plotter which is a very useful tool in such situations when you want to see the overall change for a measured value or values.

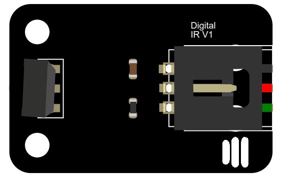
Activity 37: Infrared (IR) Kit-Part 1

IR stands for Infra-Red: It is a light that cannot be seen by the human's eye; it has a wavelength higher than our eyes' ability to detect. This wavelength can be detected by an IR sensor/receiver that decodes the light pulses into digital patterns of 0's and 1's.

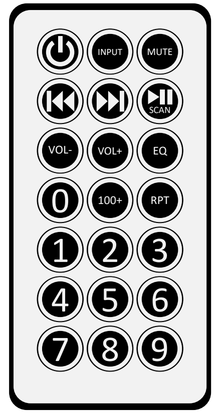
The IR kit consists of three main items; the transmitter, the receiver and the remote control. The transmitter has an IR LED light, that emits a series of IR digital pulses as a set of Highs and Lows at a certain frequency.



IR Transmitter



IR Receiver



Remote Control

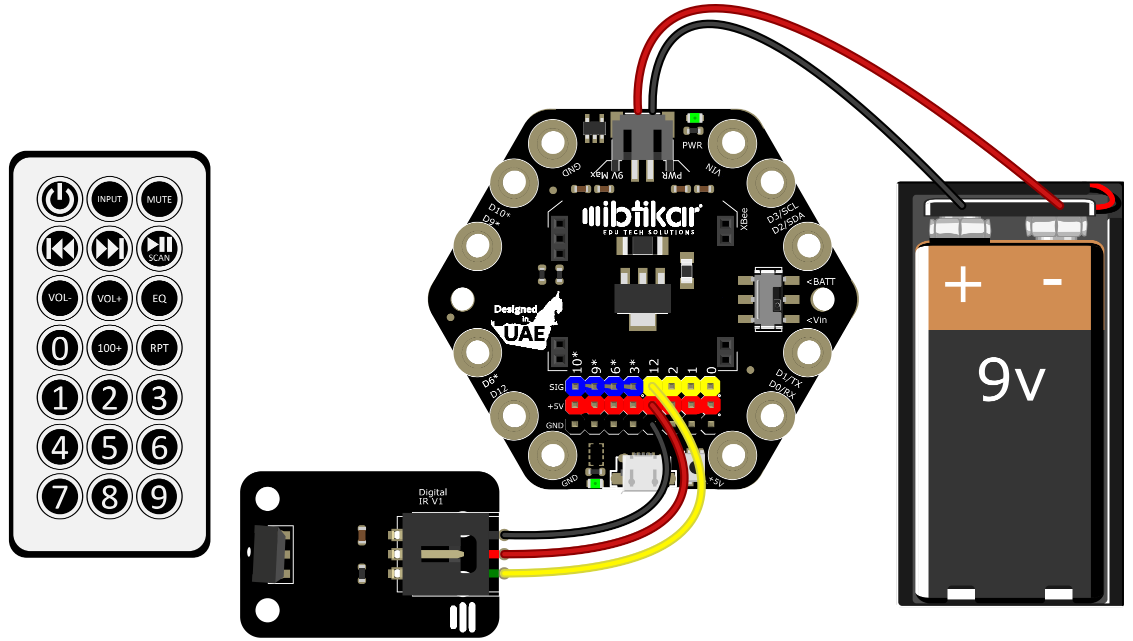
The remote control has an IR LED as well but with buttons. Each button on the remote control sends a different digital pattern of 0's and 1's that can be generated by an integrated circuit.

The IR receiver module decodes the pulses received from the IR transmitter (either the transmitter module itself or the remote control), into a digital series of 0's and 1's which can be used to perform a certain task.

Before using these modules, you need to add its library. Without the library, you cannot use these modules as it is not included by default in the Arduino IDE.

To import this library to the Arduino IDE, go to Sketch > Include Library > Add .ZIP Library ... > browse your computer files then select the library .ZIP extension (IRremote-2.2.3.zip). This library can be downloaded from different sites on the internet.

Connect the IR receiver sensor to the Maker as shown.



Upload the code to the Maker board.

```
#include <IRremote.h>

int RECV_PIN = 12;

IRrecv irrecv(RECV_PIN);
decode_results results;

void setup() {
  Serial.begin(9600);
  irrecv.enableIRIn();  // Start the receiver
}
```

```
void loop() {
  if (irrecv.decode(&results)) {  // check if any key is pressed
    String stringOne = String(results.value, HEX);
    Serial.println(stringOne);
    irrecv.resume();  // Receive the next value
  }
}
```

After uploading the code to your Maker, open the Serial Monitor, then press the keys of the remote control while you are pointing it toward the IR receiver. You should see HEX values similar to what you see below. If you are using a different remote control, you will see different values.

⏻	INPUT	MUTE	⏮	⏭	⏯ SCAN	VOL-	VOL+	EQ	100+	RPT
ffa25d	ff629d	ffe21d	ff22dd	ff02fd	ffc23d	ffe01f	ffa857	ff906f	ff9867	ffb04f

0	1	2	3	4	5	6	7	8	9
ff6897	ff30cf	ff18e7	ff7a85	ff10ef	ff38c7	ff5aa5	ff42bd	ff4ab5	ff52ad

If you are using the remote control for the first time, make sure you remove the small plastic piece that is attached to the battery compartment.

Now, use the following code to control the built-in LED, NeoPixels and buzzer using the remote control. In this activity, seven buttons are only used but you can use the buttons you prefer.

```
#include <IRremote.h>
#include <Ibtikar_IBMaker.h>

int RECV_PIN = 12;

IRrecv irrecv(RECV_PIN);
decode_results results;

void setup() {
  IBMaker.begin("V0.00");  // or V1.00
  IBMaker.clearPixels();
  irrecv.enableIRIn();     // Start the receiver
}

void loop() {
  if (irrecv.decode(&results)) {  // check if any key is pressed
    String stringOne = String(results.value, HEX);
    irrecv.resume();              // Receive the next value

         if (stringOne == "ff6897")  All_Neo(0);                   //0 key
    else if (stringOne == "ff30cf")  All_Neo(1);                   //1 key
    else if (stringOne == "ff18e7")  All_Neo(2);                   //2 key
    else if (stringOne == "ff7a85")  All_Neo(3);                   //3 key
    else if (stringOne == "ffa857")  IBMaker.ledbuiltin(HIGH);     //VOL+ key
    else if (stringOne == "ffe01f")  IBMaker.ledbuiltin(LOW);      //VOL- key
    else if (stringOne == "ff906f")  IBMaker.playTone(500, 250);  //EQ key
  }
}

void All_Neo(int CLR) {
  int R = 0, G = 0, B = 0;
  if (CLR == 1)
    { R = 160;  G = 200;  B =   0; }  // Yellow
  else if (CLR == 2)
    { R =   0;  G =   0;  B = 255; }  // Blue
  else if (CLR == 3)
    { R = 128;  G =   0;  B = 128; }  // Pink

  for (int i = 0; i < 10; i++)
    IBMaker.setPixelColor(i, IBMaker.colorWheel(R, G, B));
}
```

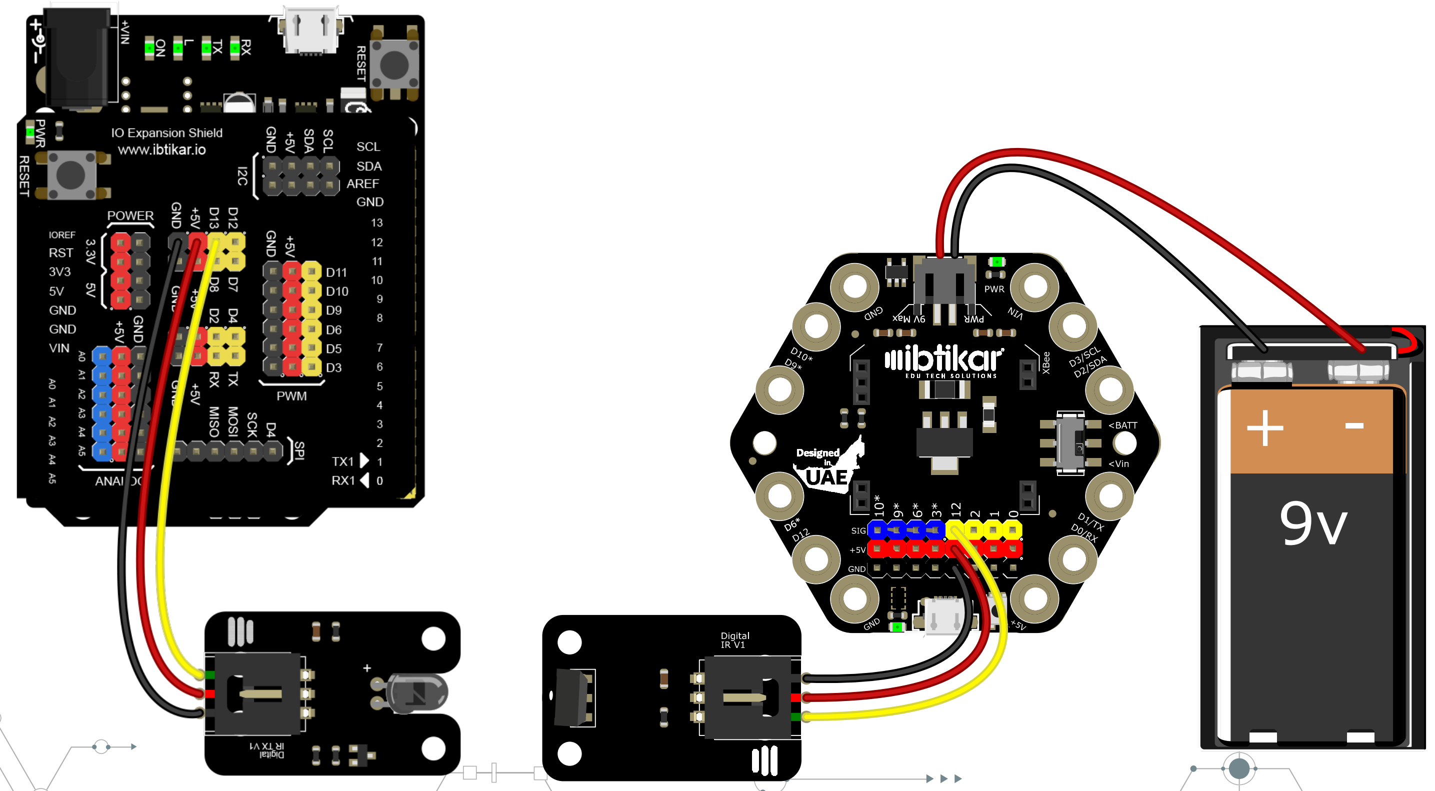
Once you click the **VOL+** button, the built-in LED on the Maker board will be ON. Click the **VOL-** button to turn it OFF. The **EQ** button plays a tone. Buttons **0**, **1**, **2**, and **3** control the NeoPixels.

Similarly, you can use any other key to do different tasks by checking its HEX value.

Activity 38: Infrared (IR) Kit-Part 2

To test the transmitter and receiver, you will need 2 boards. The first board is the Maker and it has the exact same code from part 1. The second board is an Arduino Leonardo board and it will have a code that sends the HEX values which turns the LED ON and OFF. The IR sender module must be connected to pin 13 as it is internally defined for the Arduino Leonardo in the IR library. If you are using an Arduino Uno, then you must connect the module to pin 3.

Connects the boards as shown.



Upload the code to the Arduino Leonardo board.

```
#include <IRremote.h>
IRsend irsend;

void setup() {
}

void loop() {
  irsend.sendNEC(0xffa857, 32);
  delay(1000);
  irsend.sendNEC(0xffe01f, 32);
  delay(1000);
}
```

After uploading the codes to your Maker and Arduino boards, the built-in LED will turn ON and OFF each second. If it is not blinking, make sure the modules are facing each other.

Since the same HEX values from the remote control are used, you can use the remote control as well. Cover the IR sender module with your hand. This will make the LED stop blinking. You can now control it manually using the remote control as in part 1. If you click the **VOL+** button, the built-in LED will be ON and if you click the **VOL-** button it will turn it OFF. You can use the remote-control buttons **0** to **3** to control the NeoPixels as well.

Maker Speaks Python

What is Python?

Python is a widely used high-level programming language for general-purpose programming, which was first released in 1991. It has a design philosophy which emphasizes code readability, and a syntax (a set of rules) which allows programmers to show concepts in fewer lines of code compared to other languages.

This programming language is famous for being delimited by indentation rather than being delimited by curly braces. Indenting a line is like adding an opening curly brace, and de-denting is like a closing curly brace.

Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are both available without charge for all major platforms and can be freely distributed.

Python is used in many application domains. Some of these applications are:

- Scientific and Numeric Computing.
- Web and Internet Development.
- Education: for teaching programming, both at the introductory level and in more advanced courses.
- Software Development: Python is often used as a support language for software developers, for testing and for many other tasks.

For more information or to install the latest version of Python, you can visit the Python official website: www.python.org

For tutorials on Python, you can visit this great website as well: www.learnpython.org

All the following activities have been tested on the latest Python release. At the time of writing this guide, the latest Python version is **3.7.1rc1**.

To be able to do the coming activities, you will need to install some packages to the code. If these packages are not already installed to your Python, you will need then to install them yourself. Installing packages to Python is well documented online.

Make sure you have the following packages before you start:

- pyserial
- numpy
- scipy
- matplotlib
- drawnow

Python and the Ibtikar Maker Board

Using a Python library called **pySerial**, Python can be used to connect serially to your Maker board which allows you to use all the capabilities of Python installed on your computer. Unlike Arduino IDE and Ardublockly which install the code and write it to Maker directly; PySerial allows you to send and receive data between your computer and the Maker. This means that the Maker should be programmed with a certain Arduino program allowing it to send and receive data serially.

To make this process easy, you can use the Ibtikar Serial (**IBSerial**) library which uses the pySerial with all the Maker functions and many Arduino commands implemented in Python. This library is intended to mirror the Maker Arduino functions and use them in Python. On the Arduino side, the Ibtikar Serial library can be used to program Maker with code that allows for the communication between your Python code and the Arduino Maker library.

To start using the Maker with Python, follow these steps:

1. Ensure you have Python and Arduino IDE installed on your computer.

2. If the libraries are not already installed, copy the **Ibtikar_IBMaker** and **Ibtikar_IBSerial** libraries to the Arduino IDE library folder.

3. Copy the **IBserial.py** to the Python library folder.

4. Upload the **SerialMaker.ino** sketch to your Maker board using the Arduino IDE.

5. Write your Python program to control Maker the way you like. You can always start from the **blank.py** file which contains the header name and serial configurations.

This is what the **blank.py** file looks like.

```
# **************************************************
import time
from IBSerial import *
import IBSerial as IBMaker

## open the COM port
IBMaker.Open_Port("COM13","115200")

## map pinMode
IBMaker.PinMAP("MAKER")

IBMaker.begin("V0.00")       ## or "V1.00"
## ***************************************


## Write your code here


## ***************************************
## close the COM port
IBMaker.Close_Port()
## ***************************************
```

In this file, there are a couple of things you must pay attention to. These are:

- The COM port number may differ based on which COM the board is connected to.
- The Baud rate is 115200 which matches the one in the **SerialMaker.ino** sketch.
- The PinMAP function maps the Maker pins to allow you to use them as digital or Analog and to read or write to them.
- Based on which version of Maker you have, you need to specify the number. This function is similar to the one in the Arduino activities.
- The last line is to close the COM port once you finish the instructions.

Activity 39: LED Blink in Python

This activity shows how to blink the built-in LED in Python and compare it with the one in Arduino. All functions related to Maker are made similar to the Arduino ones.

First recall how to blink the built-in LED in Arduino (Activity 2).

```
#include <Ibtikar_IBMaker.h>

void setup() {
  IBMaker.begin("V0.00");  // or "V1.00"
}

void loop() {
  IBMaker.ledbuiltin(HIGH);
  delay(1000);
  IBMaker.ledbuiltin(LOW);
  delay(1000);
}
```

Now, try this code to blink the built-in LED in Python. But first, do not forget to upload the **SerialMaker.ino** sketch to your board.

```
## ***************************************
import time
from IBSerial import *
import IBSerial as IBMaker

## open the COM port
IBMaker.Open_Port("COM13","115200")

## map pinMode
IBMaker.PinMAP("MAKER")

IBMaker.begin("V0.00")      ## or "V1.00"
## ***************************************

while(1):
  IBMaker.ledbuiltin(HIGH)
  time.sleep(1)
  IBMaker.ledbuiltin(LOW)
  time.sleep(1)
```

The COM port in this example is 13 which may differ from the one your Maker is connected to. The main code is almost the same as the one in Arduino except for the delay function. In Python, the delay has a function called **sleep** which is defined in the **time** library. This function accepts the sleep time in the unit in seconds unlike the Arduino **delay** function which accepts the value in milliseconds.

Since the program will enter the while loop and stay there forever, there is no need to add the part where you close the COM port.

Points to Keep in Mind

- Activity 1 to Activity 28, Activity 33 and Activity 35 are implemented in the exact same way the original Arduino activities are implemented.
- Some Arduino commands are implemented in the Python **IBSerial** library to make it easier for you, such as the **map** function in Activity 16 and the **shiftOut** function in Activity 35.
- For some activities where you need to import an Arduino specific library or implement a function directly in Arduino, you will need to include the extra required library or functions in the **SerialMaker.ino** sketch and upload them to the Maker first. These activities are Activity 29, Activity 32, Activity 36.
- Since the Python **IBSerial** library depends on the serial communication, you should expect a short delay. Based on the activity you are trying to do; the effect of the delay will vary. For most of the activities, the delay effect is negligible. In Activity 10 and 11 the delay only slightly affects the button counting but in Activity 34 some counts are missing due to this delay which makes the activity unreliable.
- Because of the way the **IbSerial** library is implemented and due to the time in which this guide is being authored, passing interrupts from Arduino to Python are not supported. Hence, Activity 30 and Activity 31 are not implemented.
- Due to the size of the IRremote library, Activity 37 and Activity 38 could not be implemented on the Maker.
- The codes for all these activities, including the different versions of **SerialMaker.ino** sketch, are available for download, hence there is no point in repeating them in this guide.

Real-time Logging using Maker and Python

In this section, Python will be used to log real-time data from the Maker board. This will show the power of using Python with Maker. In Python, you can plot the data received, add a timestamp to it and save it to a file. In addition to the **IBSerial** and **time** packages, you will need to have **pyserial**, **numpy**, **matplotlib** and **drawnow**.

Activity 40: Accelerometer Datalogging

In this activity, the three axes of the Maker accelerometer are read, as explained earlier in this guide. The timestamp is added using the **datetime** package. The plot is created using the **matplotlib** and **drawnow** packages. The **numpy** package is needed to handle arrays, and append and pop samples, to and from each axis array.

- The following block of code shows all the packages needed. Make sure to choose the correct COM port and version of the Maker board.

```
## ***************************************
import time
from IBSerial import *
import IBSerial as IBMaker

## open the COM port
IBMaker.Open_Port("COM13","115200")

## map pinMode
IBMaker.PinMAP("MAKER")

IBMaker.begin("V0.00")        ## or "V1.00"
## ***************************************

import matplotlib.pyplot as plt
from datetime import datetime
from drawnow import *
import numpy as np
## ***************************************
```

- First, enable the accelerometer and specify the number of samples needed. Say we want to read 100 samples, this means we will need to define the arrays where the values will be stored and initialise them with zero.

```
IBMaker.Enable_ADXL(true)

samples = 100

X_values = []
Y_values = []
Z_values = []

#pre-load dummy data
for i in range(0,samples+1):
    X_values.append(0)
    Y_values.append(0)
    Z_values.append(0)
```

To print values to a text file, you need to open the file first by specifying its name and then specify the way you are planning to work on the file. The letter 'a' means append. This means that when you write a new value, it will be appended to the end of the file rather than replacing the old value.

If the file did not exist before, it will be created then opened. Once the file is opened, you write the string values you want. In this activity, we will write the sample number, the current date and time, the accelerometer values and finally the end of line character. All these values are separated by commas. The following block of code shows this function.

```
def printToFile(SAMPLE,TIME,X,Y,Z):
    dataFile = open('dataFile.txt', 'a')
    dataFile.write(str(SAMPLE)+','+TIME+','+str(X)+','
                   +str(Y)+','+str(Z)+'\n')
    dataFile.close()
```

The following function creates the plot window by dividing the plot into three subplots (3 rows and 1 column). For each subplot, one of the accelerometer axes is plotted with a specific style, colour, line width and marker size.

```
def plotValues():
    plt.subplot(3,1,1)
    plt.plot(X_values, 'o-', label='X', linewidth=2, markersize=5,
             color='cornflowerblue')
    plt.ylabel('X Value')
    plt.title('Maker Serial Data')
    plt.grid(True)
    plt.legend(loc='upper left')
    plt.xlim(0, samples)
    plt.ylim(-15, 15)

    plt.subplot(3,1,2)
    plt.plot(Y_values, 'o-', label='Y', linewidth=2, markersize=5,
             color='crimson')
    plt.ylabel('Y Value')
    plt.grid(True)
    plt.legend(loc='upper left')
    plt.xlim(0, samples)
    plt.ylim(-15, 15)

    plt.subplot(3,1,3)
    plt.plot(Z_values, 'o-', label='Z', linewidth=2, markersize=5,
             color='orange')
    plt.ylabel('Z Value')
    plt.xlabel('Samples')
    plt.grid(True)
    plt.legend(loc='upper left')
    plt.xlim(0, samples)
    plt.ylim(-15, 15)
```

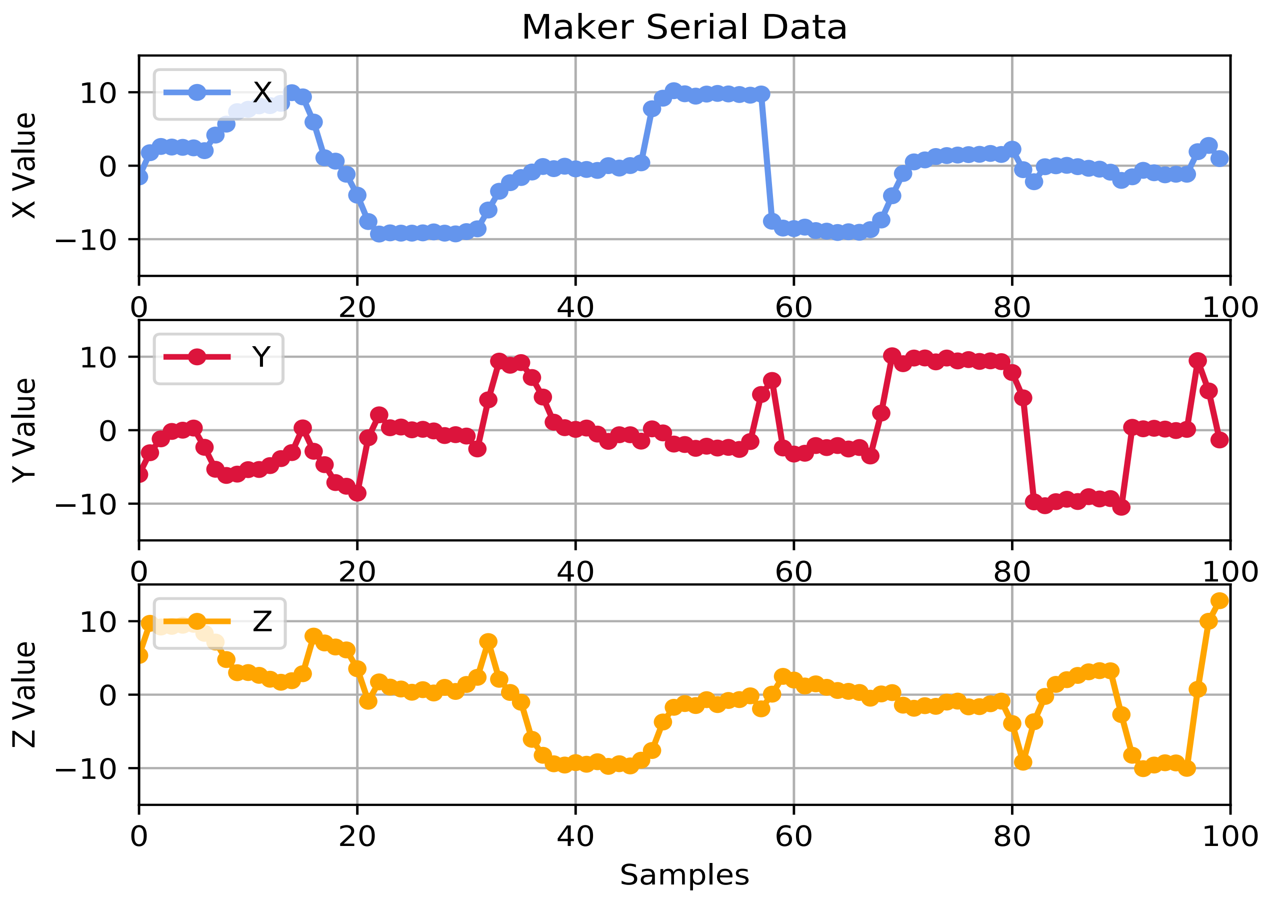
The plot has a title, labels for plot axes, legends to show the signal name and grids for each plot. You can customise all these properties yourself. The documentation for each package is the best place to start. They are all well documented with lots of examples.

- The main loop contains the commands to read the accelerometer and the current time and date. These values are then passed to the **printToFile** function. The arrays have a fixed size, so the new values will be appended at the end and the first values will be popped out, which will keep the array size the same. Then update the plot using the **drawnow** function. Once the loop is finished, the COM port will be closed.

```
for i in range(0,samples):
    x = IBMaker.motionX()
    y = IBMaker.motionY()
    z = IBMaker.motionZ()

    Mytime = datetime.now().strftime('%Y-%m-%d,%H:%M:%S.%f')[:-3]
    printToFile(i,Mytime,x,y,z)

    X_values.append(x)
    X_values.pop(0)

    Y_values.append(y)
    Y_values.pop(0)

    Z_values.append(z)
    Z_values.pop(0)

    drawnow(plotValues)
## *************************************
IBMaker.Close_Port()
## *************************************
```

After combining all these parts and running the code, the plot window will open. You will see the three plots moving to the left. Rotate your Maker and notice the change in the values. Once finished, check the folder where your Python script is saved. You will find a new file called **dataFile.txt** with all the values saved.

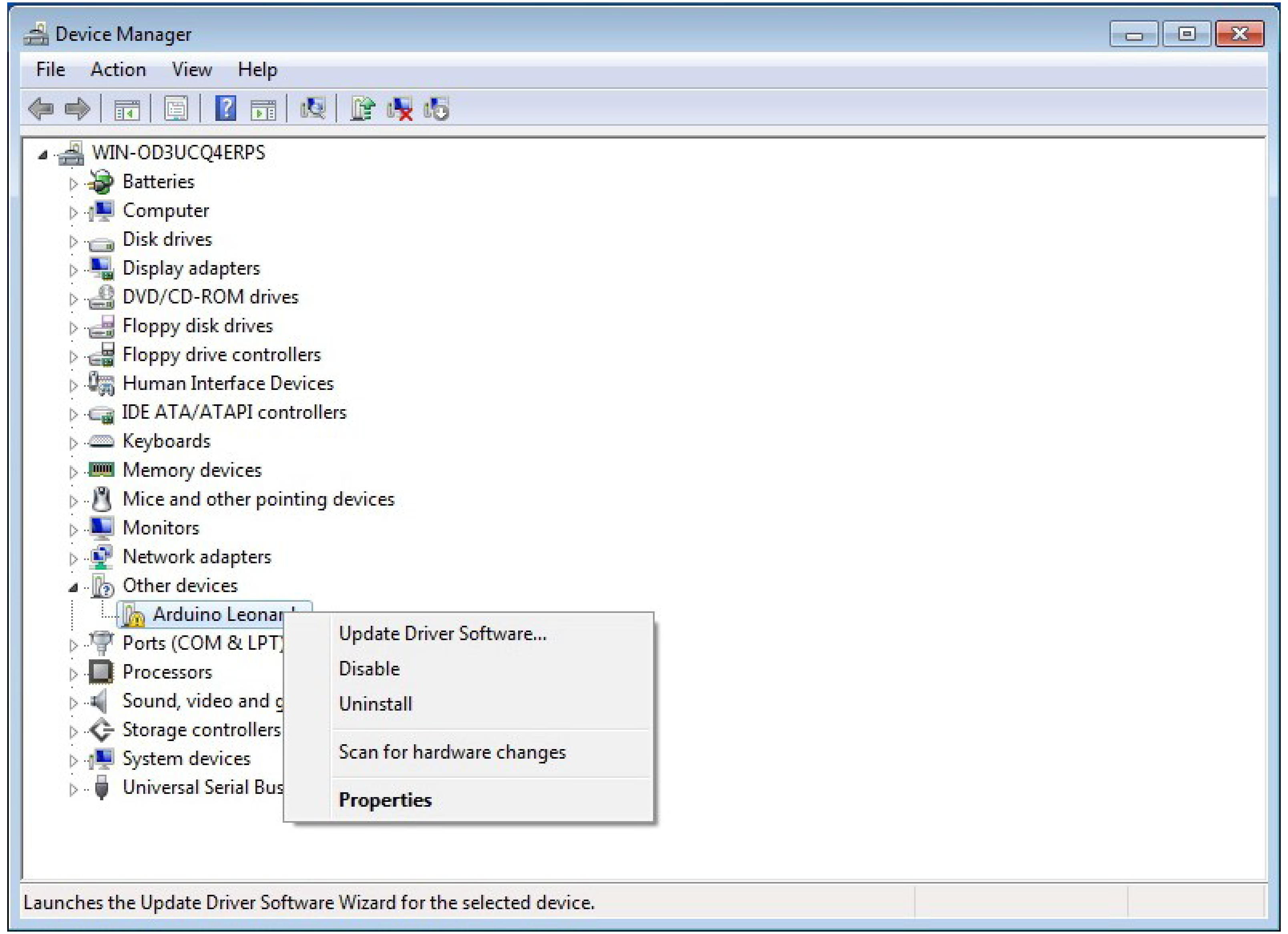
This is how the plot looks.



Appendix 1: Arduino Driver Installation

To install the Arduino board driver to the computer, follow these steps:

- Click on the *Start Menu* and then write *Control Panel*.
- Open it and then navigate to *System and Security*.
- Click on *System* and on the left panel click on *Device Manager*.
- A window will pop up.
- Under *Other Devices*, you will see an icon with a yellow hazard sign named *Arduino Leonardo*.
- Right click on the icon named *Arduino Leonardo*.
- Then click *Update Driver Software*.
- Browse to the driver folder where the Arduino IDE is installed and click *next*.
- Your computer will install a proper *Driver* for your board.



Launches the Update Driver Software Wizard for the selected device.

Appendix 2: Maker Commands List

Arduino-Specific Functions

```
#include <Ibtikar_IBMaker.h>
```

```
void setup()
```

```
void loop()
```

```
Serial.begin(Baud_rate)
```

```
Serial.print(Value_to_print)
```

```
Serial.println(Value_to_print)
```

```
delay(time_in_milliseconds)
```

Note:

Activity 31 contains more advanced accelerometer commands which you can refer to.

Python-Specific Functions

```
import time
from IBSerial import *
import IBSerial as IBMaker
```

```
IBMaker.Open_Port("COM_No","115200")
```

```
IBMaker.PinMAP("MAKER")
```

```
IBMaker.Close_Port()
```

```
time.sleep(time_in_seconds)
```

```
## This is how you write comments
```

Advanced Commands

```
IBMaker.Echo_ON()
```

```
IBMaker.Echo_OFF()
```

```
IBMaker.Debug(1)
```

```
IBMaker.Debug(0)
```

```
IBMaker.Help()
```

```
TEST_WRITE(command,[expected_reply],["OK","ERROR"])
```

Common Functions

Initialisation

```
IBMaker.begin("V0.00")
```

```
IBMaker.begin("V1.00")
```

Built-in LED

```
IBMaker.ledbuiltin(state)
```

LED Grid

```
IBMaker.setLed(row, column, state)
```

```
IBMaker.setRow(row_number, 0bxxxxxxxx)
```

```
IBMaker.setColumn(column_number, 0bxxxxxxxx)
```

```
IBMaker.Leds_Num(0, duration)
```

```
IBMaker.Leds_Char('', duration)
```

```
IBMaker.Leds_Str("", duration)
```

```
IBMaker.Create_Center(x, y)
```

```
IBMaker.Move(steps)
```

```
IBMaker.Turn(CW)
```

```
IBMaker.Turn(CCW)
```

```
IBMaker.clearDisplay(0x00)
```

```
IBMaker.LEDBrightness(level)
```

Push Buttons

```
IBMaker.ButtonL()
```

```
IBMaker.ButtonR()
```

```
IBMaker.ButtonCount(IBPIN_LEFT, number_of_counts)
```

```
IBMaker.ButtonCount(IBPIN_RIGHT, number_of_counts)
```

Temperature Sensor

```
IBMaker.temperatureC()
```

```
IBMaker.temperatureF()
```

Light Sensor

```
IBMaker.Sensor_Light()
```

Buzzer

```
IBMaker.playTone(frequency, duration)
```

Sound Sensor

```
IBMaker.Sensor_Sound()
```

Touch Pads

```
IBMaker.Touch(pad_number)
```

pad_number: 0, 2, 3, 6, 9, 10 or 12

Accelerometer

```
IBMaker.Enable_ADXL(true)
```

```
IBMaker.Enable_ADXL(false)
```

```
IBMaker.motionX()
```

```
IBMaker.motionY()
```

```
IBMaker.motionZ()
```

NeoPixels

```
IBMaker.clearPixels()
```

```
IBMaker.NeoBrightness(level)
```

```
IBMaker.setPixelColor(pixel_number, IBMaker.colorWheel(R, G, B))
```

Read/Write Digital & Analog

```
pinMode(pin_number,mode)
```

```
analogWrite(pin_number,value)
```

```
analogRead(pin_number)
```

```
digitalWrite(pin_number,value)
```

```
digitalRead(pin_number)
```

maker™ GUIDE

FOR ADVANCED USERS

ARDUINO™

python™