Teacher:

Class:

Crack the Code

Revised to reflect the 2019 implementation of the Technology Mandatory Years 7-8 Syllabus

Teacher workbook

Copy and paste the following binary code into a translator to see who wrote this unit. You can skip the binary code if you are using a screen reader.

01010100 01101000 01101001 01110011 00100000 01110101 01101110 01101001 01110100 00100000 01110111 01100001 01110011 00100000 01110111 01110010 01101001 01110100 01110100 01100101 01101110 00100000 01100010 01111001 00111010 00100000 01000100 01100001 01101110 01101001 01100101 01101100 01101100 01100101 00100000 01010011 01100001 01110101 01101110 01100100 01100101 01110010 01110011 00101100 00100000 01000001 01101100 01100101 01111000 01100001 01101110 01100100 01100101 01110010 00100000 01010011 01110100 01100101 01110111 01100001 01110010 01110100 00100000 01100001 01101110 01100100 00100000 01001010 01101111 01101000 01101110 00100000 01010111 01100001 01101100 01101100 01100001 01100011 01100101 00101110

Acknowledgements

The resources for the ‘Crack the Code’ unit of work were developed by the NSW Department of Education, Learning and Teaching Directorate for use by TAS teachers in NSW. They are developed to meet the requirements of the NESA Technology Mandatory Years 7-8 Syllabus 2017.

The materials were developed by:

* Danielle Saunders, Elderslie High School
* Alexander Stewart, Carlingford High School
* John Wallace, Carlingford High School
* Dan Rytmeister, Learning and Teaching Directorate.

The following people and organisations contributed to the consultation and development process of the materials:

* Museum of Applied Arts and Sciences (MAAS)
* Peter Davis, Normanhurst Boys High School
* Andrew Murray, Cranebrook High School
* Kylie Rytmeister, East Hills Girls Technology High School
* Greig Tardiani, Information Technology Directorate

The sample code required for this unit of work can be downloaded from <https://education.nsw.gov.au/curriculum/tas/s4-5/resources>

For more information contact:

Dan Rytmeister

TAS Advisor

NSW Department of Education

[dan.rytmeister@det.nsw.edu.au](mailto:dan.rytmeister@det.nsw.edu.au)

July 2018 - Version 3.2

Contents

[Crack the Code – Unit overview 4](#_Toc514103890)

[The design and production process 9](#_Toc514103891)

[Control technologies 10](#_Toc514103892)

[Microcontrollers 13](#_Toc514103893)

[PRP #01: Plug Run Play – Blink 23](#_Toc514103894)

[Binary – What you need to know 32](#_Toc514103895)

[PRP #02: Digital input – Button 34](#_Toc514103896)

[Input components or sensors 44](#_Toc514103897)

[PRP #03: Analog input – potentiometer and Light Dependant Resistor (LDR) 45](#_Toc514103898)

[PRP #04: Digital output – buzzer 56](#_Toc514103899)

[Managing your design project 67](#_Toc514103900)

[Understanding electronic circuits and Arduino 78](#_Toc514103901)

[Components of circuits / control technologies 79](#_Toc514103902)

[Electronic Connection Ready Reckoner 87](#_Toc514103903)

[Crack the code – Marking criteria 92](#_Toc514103904)

[Troubleshooting and debugging your code 94](#_Toc514103905)

Crack the Code – Unit overview

Design situation

As an adolescent in the 21st Century you are surrounded by technology. You are living in an era where there are gizmos and gadgets that can do all sorts of things to help you with organisation, communication and entertainment. Did you know that electronic devices, even simple ones like a TV remote control, use a computer program to tell them what to do? What if you could control what an electronic device does? Well you can by writing your own simple computer program which is referred to as coding. A computer program is referred to as ‘code’.

In this unit of work, you will learn about coding and how to use it in the creation of a variety of projects that are known as control technologies. You will use the skills learnt throughout the unit in a design project which is outlined in the ‘Design brief’. By completing the Plug Run Play (PRP) activities in this booklet you should gain enough experience to complete the following design brief.

Design brief

Utilising your skills in control technologies and coding you are to design, produce and evaluate an alarm/alert system to inform you if someone has broken into something you own or care for. This may include bedrooms, boxes, treasure chests etc.

Constraints

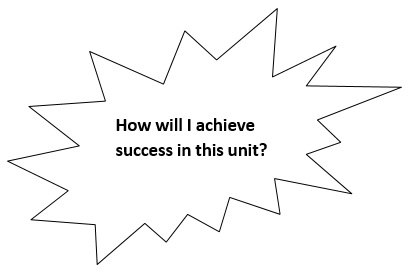
Your ability to develop, assemble and code a control technology within your budget, ability and time frame. Your control technology must:

* Use a microcontroller e.g. Arduino
* Include working inputs (sensors) and outputs (actuators)
* Vary outputs depending on input data (branching)

Analysis of design brief and constraints

Describe exactly what you need to do in the space below.

Criteria for success



Write your criteria for success on the lines below.

Glossary

Many of the following words will gather more meaning to you as you work through the coding PRPs in this booklet.

Complete the following table with definitions as you progress through the unit.

| Word | Definition |
| --- | --- |
| actuator | A component that will move in your circuit. For example a servo motor in a remote control car controls the steering wheel direction. |
| analogue input | Allows an input voltage on Arduino anywhere between 0 and 5 volts, as opposed to digital which is off or on (0 or 5 only). |
| analogue output | Allows an output voltage on Arduino anywhere between 0 and 5 volts, as opposed to digital which is off or on (0 or 5 only). |
| analogRead | Command converts the input voltage range, 0 to 5 volts, to a digital value between 0 and 1023. This is done by a circuit inside the microcontroller called an analog-to-digital converter or ADC |
| analogWrite | Writes an analog value to a pin. Can be used to light a LED at varying brightnesses or drive a motor at various speeds. Range is between 0 and 255. |
| Arduino | A small and easy to use computer board.  A programmable microcontroller board. |
| board | A surface that electronic components and parts are attached to. Can be made of various materials but they must not conduct. |
| branching | An instruction in a computer program or algorithm that causes different actions to be performed depending on specified conditions. Flow charts are examples of branching. |
| camel case | Using capital letters to join words that have **been joined together.** |
| coding | Another word for computer programming. |
| component | Electronic parts that on their own generally have no function. When combined into a circuit they can perform a task such as being a night light. |
| conductive | Able to transfer electricity/electronic charge through itself. E.g. copper wire. |
| control technology | Electronic devices that use inputs to create outputs. |
| declaration | Means defining the type, and optionally, setting an initial value (initialising the variable) of a variable. |

| Word | Definition |
| --- | --- |
| digital input | An input that is only on or off, it is not able to gradually turn on or off. |
| digital output | Can only be switched on and off, not able to be changed to a % of the full “on” value. Usually pins 0-13 on an Arduino Uno. Pins with pulse wave modulation (PWM) can simulate a % value by switching on and off really fast. |
| digitalRead | To receive a HIGH or LOW signal from a pin that has been made an input. |
| digitalWrite | To send a HIGH or LOW signal from a pin that has been made an output. |
| electrical circuit | A path that allows electricity to flow. |
| false | Not true. No electrical current present. Also known as OFF or LOW or 0. |
| hardware | The physical components of the electrical circuit and microcontroller (Arduino). |
| HIGH | Also known as ON or True or 1. It means that there is electrical current present (up to 5V for Arduino Uno). |
| Input, Processing and Output (IPO) Chart | A simple map of the input and output components of a control system. These make it easier to understand and design the control system you are working with or building. |
| Integer | A whole number |
| Light dependant resistor (LDR) | A component that varies resistance depending on the amount of light on it. |
| Light emitting diode (LED) | Light Emitting Diode – a component that emits light and only lets electrical current pass in one direction. They are available in a wide range of colours. |
| LOW | No electrical current present. Also known as OFF or False or 0. |
| microcontroller | A small programmable computer that is able to run code. |
| multimeter | A tool to measure a wide range of electrical values (including amps, continuity, resistance and volts) on circuits and components. |
| piezo buzzer | Makes a buzzing sound with electrical current applied. They are not a speaker but still output sound. |

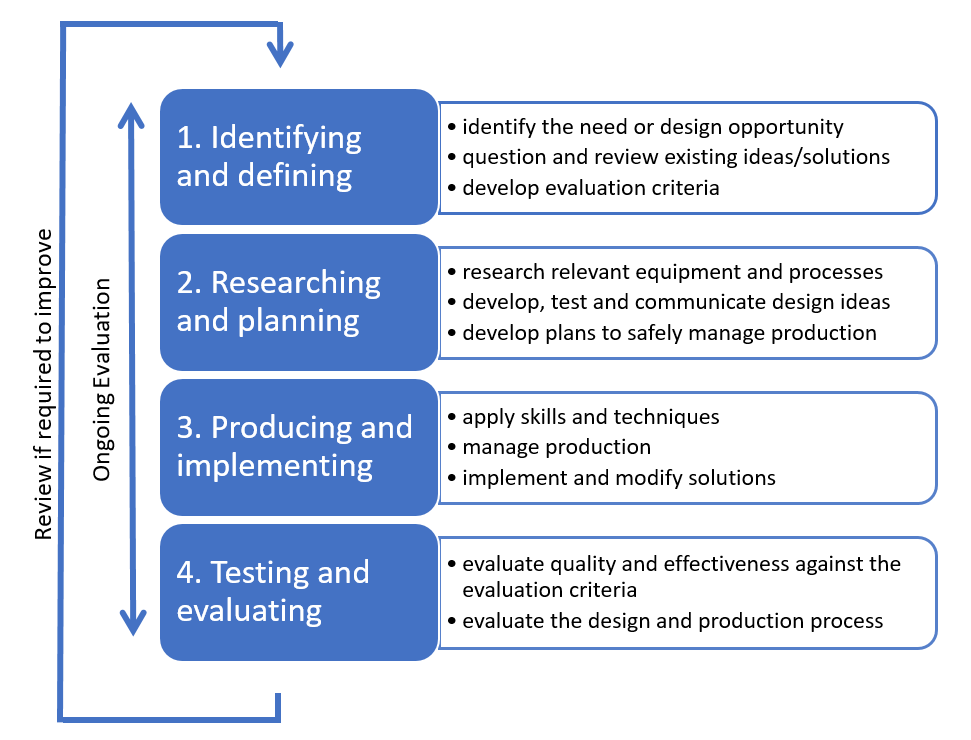
| Word | Definition |
| --- | --- |
| port | A connection to the Arduino that sensors or actuators to ‘plug in’. |
| potentiometer | A resistor that can have its resistance changed by turning a knob or moving a slide, just like on a car radio or home stereo system. |
| power supply | This supplies the electrical circuit with power. Comes in a large range of options from small batteries (like in watches/fitbits), to power packs that plug into power points. |
| pseudocode | Writing in plain English line by line (step by step) what you want the computer program to do. |
| resistor | A component that reduces the amount of current (amps) in a circuit. |
| sensor | A device that can be connected to the microcontroller that detects a change in its environment providing data to the microcontroller. |
| shield | A shield fits onto the microcontroller and is connected via the analogue and digital ports that run along the side of the Arduino board. The Thinkershield is an example. |
| sketch | This is the name Arduino give their code or programs. |
| tactile switch | A switch operated by a human. |
| true | Also known as ON or HIGH or 1. It means that there is electrical current present (up to 5V for Arduino Uno). |
| void loop | This section of code which runs after void setup runs forever or until the power runs out. |
| void setup | This section of code gets the Arduino prepared for the code that is going to run continuously and is always run before the void loop. Without this part being defined you will get errors as the Arduino board will not know how to execute the code. |
| Volts or voltage | The units of potential or “pressure or force” that electrical current wants to flow at. |

The design and production process

Throughout your study of Technology, you will learn about design processes and how to apply them. You will explore different types of design processes and learn how to apply them in your design project.

The design and production process:

* Involves a sequence of organised steps which provide a solution to design needs and opportunities
* May take a few seconds or minutes, such as when you select what clothes to wear, or may take years as in the case with the design of a motor vehicle
* May involve one person or may involve many people
* May be simple or complex, depending on the task
* Involves the designer questioning (or evaluating) throughout the process



The sequence (or steps) in design processes may vary depending on design Does your solution perform as it should?

Throughout the process there should be ongoing evaluation and a review if required to improve the outcomes.

Control technologies

Control Technologies are all around us. They are in our homes, classrooms, shopping centres and even out streets! But what are they and how do they work?

Identify four examples of control technology systems in the space below.

Traffic lights

Climate control air conditioning in cars

iPads / tablets / mobile phones

Dishwasher

Tickle me Elmo

The following YouTube clip, [LOL](https://www.youtube.com/watch?v=65maMPzLFjg) [Elmo](https://www.youtube.com/watch?v=65maMPzLFjg) (<https://www.youtube.com/watch?v=65maMPzLFjg>), will show you how ‘Tickle Me Elmo’ works. After watching it, complete the following questions in the space provided.

What makes Elmo move?

Electrical motors controlled by a microcontroller inside the toy make it move.

What happens when you press the button on Elmo’s hands or feet?

the sensor in Elmo's hand is a switch that tells the microcontroller that it has been pressed. The microcontroller is programmed to provide power to the motors that make it move.

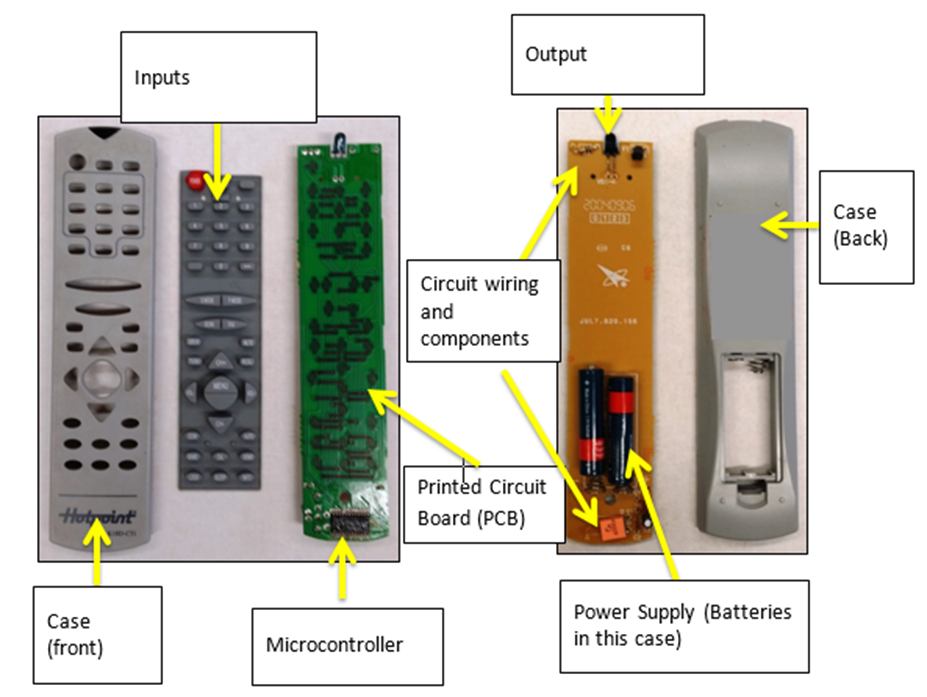
How do you think Elmo knows to move/vibrate when the button has been pushed?

When the sensors detect a change the microcontroller detects this and is programmed to make a decision about what to do which in this case is activate the motors

Remote control for a television

Here is a TV remote control that has been pulled apart. Using the following word bank, label the components used.

* Case (back)
* Case (front)
* Circuit, wiring and components
* Inputs
* Microcontroller
* Outputs
* Power supply (batteries)
* Printed circuit board (PCB)



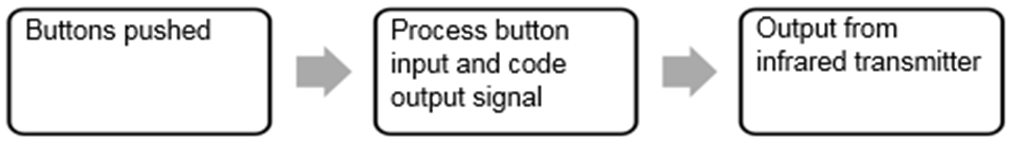
There are three core components of all control technology systems. Identify them below:

Input device (sensor)

Processor (decision maker)

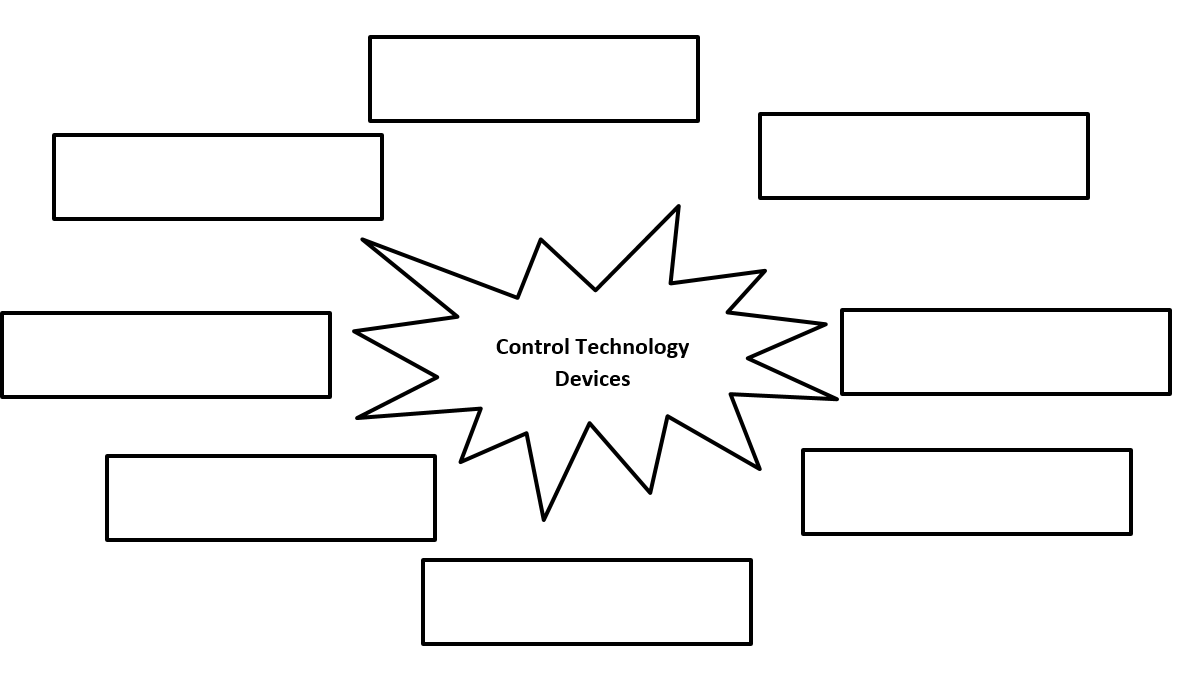
Output device (actuator)

Complete the Input, Processing and Output (IPO) chart below for the remote control.



Defining a control technology

Brainstorm as many control technology devices as you can below.



How do control technologies actually work? Write your own definition below then share with the class.

Microcontrollers



Word bank

* computer
* controller
* Independent
* inputs
* integrated circuits
* logic
* microchips
* outputs
* Power supply
* programmable
* programmable
* programmed
* receives
* simple
* sketch
* USB cable

What is a microcontroller?

A microcontroller is a **programmable** device that **receives** data from **inputs** in order to produce one or more **outputs**.

A microcontroller is made up of small **microchips** that contain microscopic circuits called **integrated** **circuits** that can be programmed to carry out **simple** functions. Once programmed, there is no need to use a **computer** to operate them.

Another name for a Microcontroller is a: **P programmable L logic C controller** (PLC)

How does it work?

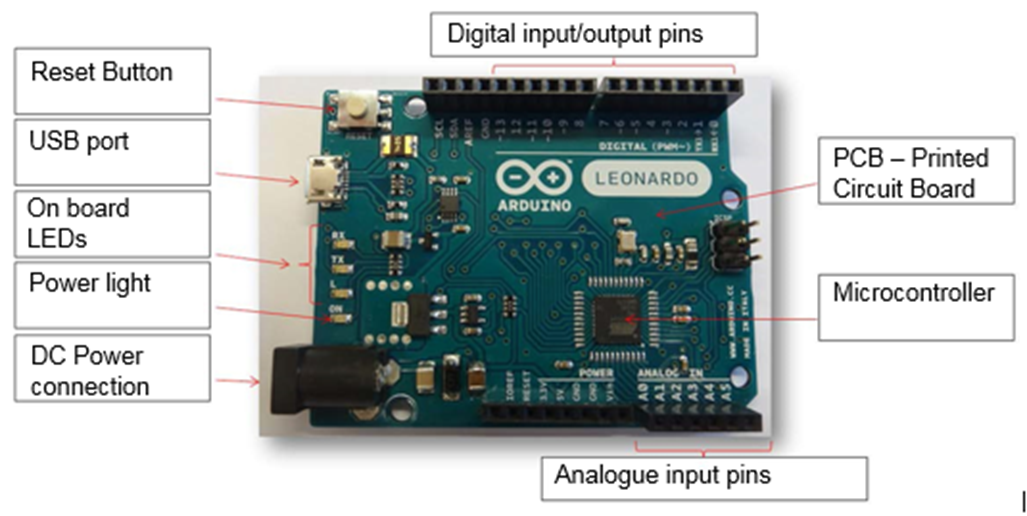
A microcontroller can only work if it has been **programmed**. The program which is called a **sketch** in Arduino is written on a computer. The program is then uploaded to the microcontroller chip via a **USB cable** connection.

The microcontroller can then be disconnected from the computer and an alternate **Power supply** can be connected. This allows the microcontroller to be an **Independent** control technology system

The Arduino microcontroller

Arduino is a company that develops microcontrollers and the software required to program them ([www.arduino.cc](http://www.arduino.cc/)). During this unit you will be using Arduino software and a compatible microcontroller board to create your own control technology.

Look at your microcontroller and discuss as a class the parts of the board and their functions. Label the diagram below using the work bank at the bottom of the page.



Word bank

* Analogue input pins
* Digital input/output pins
* DC Power connector
* Microcontroller
* On-board LED
* Power light
* Printed Circuit Board (PCB)
* Reset button
* USB port

MAAS ThinkerShield

Answer the following questions in the space provided.

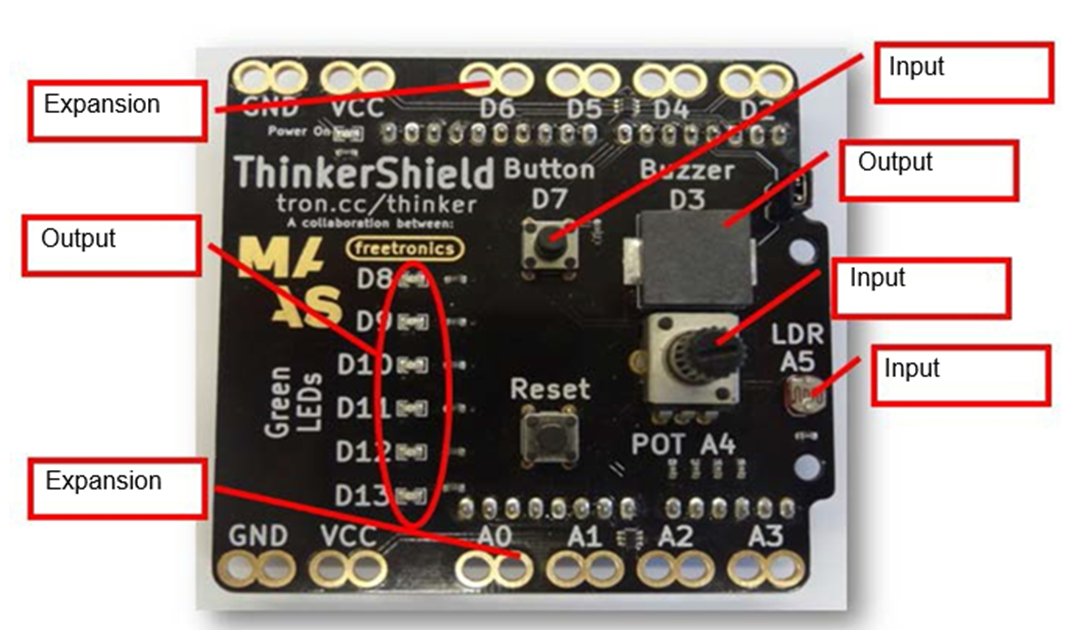
What is a ‘shield’?

**Shields are used with Microcontrollers to create an easy-to-use interface for developing control technology systems. The shield fits onto the microcontroller and is connected via the analogue and digital ports that run along the side of the Arduino board. (Show how they fit together)**

The ‘MAAS ThinkerShield’ was developed by the ‘Museum of Applied Arts and Sciences’ Sydney (formerly the Powerhouse Museum), Australia. It contains a variety of on-board components to make it easy to learn how to ‘code’ without having to learn how to do electronics.

Activity

Identify each of the components on the ‘ThinkerShield’ as either input, output or expansion.



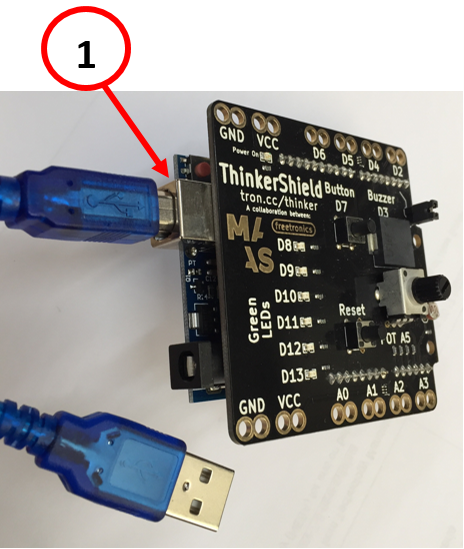
Word bank

* Input
* Output
* Expansion

Arduino software setup

How to connect the Arduino board to the computer

1. Using a USB cable, connect the Arduino Uno board to computer.



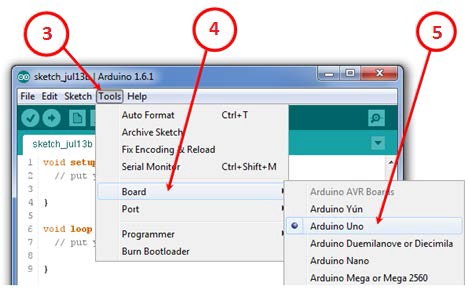
1. If successful, the power light on the ThinkerShield will turn on.



Setting the Arduino board type

You will need to make the Arduino board known to the application so you can upload the Blink sketch to the board.

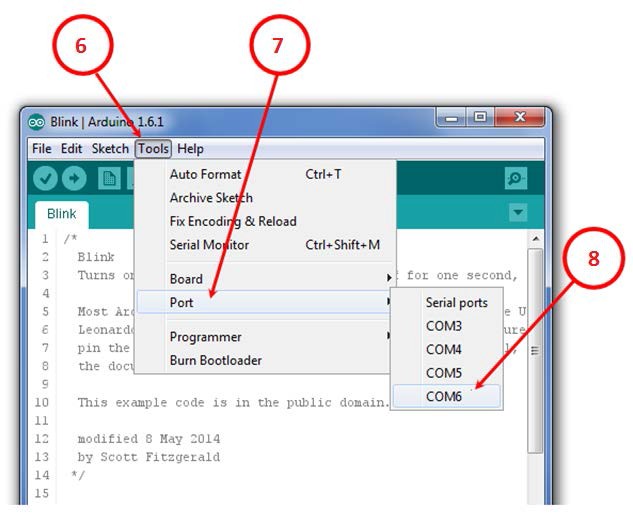
1. Click on the ‘Tools’ drop down menu.
2. Click on ‘Board’ option. Another menu list appears.
3. Locate the Arduino board type and click on it. A blue button will appear in front of the selection you made and the menus will automatically disappear.



© [ARDUINO LLC 2016](https://www.arduino.cc/) - CC-BY-SA 3.0

Setting the port number

1. Click on the ‘Tools’ drop down menu.
2. Click on the ‘Port’ option.
   1. On Windows: There is usually only one option to choose but if there is more than one then it will most likely be the COM with the highest number. There is no harm in guessing wrong so if it doesn’t work, try the next one. Another way to find the correct port is to take note of the options available, disconnect your Arduino board and re-open the menu; the entry that disappears should be the Arduino board.
   2. On Macintosh: This should be something with /dev/tty.usbmodem in it. There are usually two of these; select either one.
3. Click on the ‘Port’ number the Arduino is connected to. A tick will appear in front of the selection made and the menus will automatically disappear.
4. Each time you connect the Arduino you will need to check the settings are correct or an error may occur.



© [ARDUINO LLC 2016](https://www.arduino.cc/) - CC-BY-SA 3.0

Safety considerations

List some general safety issues to consider when using a microcontroller.

**It runs on electricity so no drinks or other liquids, otherwise it may short circuit.**

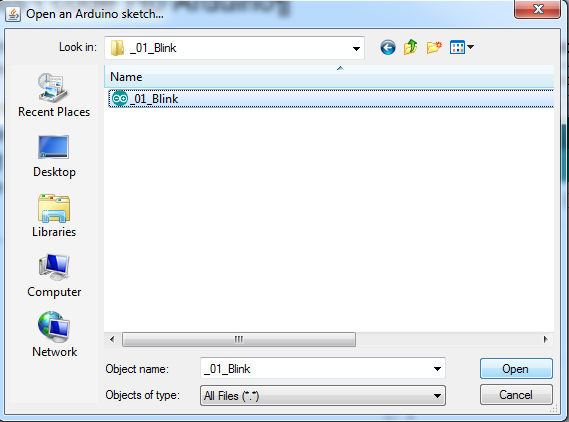
**You can create a “short” if you put it on conductive surfaces, so only put it down on non-conductive bench and table tops.**

Loading sketches (‘code’) to Arduino

Locate the sketch “\_01\_Blink” that has been provided by your teacher. (an Arduino program is called a ‘sketch’). You can also download the sample sketch from the [Crack the Code resources page](https://schoolsnsw.sharepoint.com/sites/Stage4Coding/Shared%20Documents/Crack%20the%20Code%20resources%20page). (<https://education.nsw.gov.au/teaching-and-learning/curriculum/key-learning-areas/tas/s4-5/resources/crack-the-code>).

To load the sketch either double click on the file or navigate to it by follow these instructions:

1. Click on the ‘File’ drop menu.
2. Click on ‘Open…’.
3. Locate and select ‘\_01\_Blink’.
4. Click on Open.



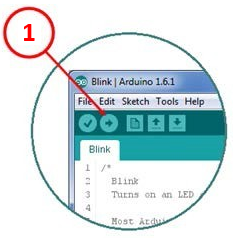
© [ARDUINO LLC 2016](https://www.arduino.cc/) - CC-BY-SA 3.0

Uploading the sketch

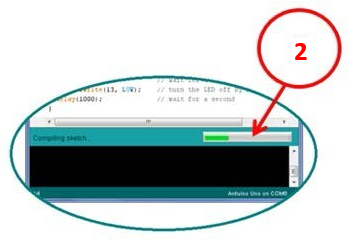
Now that you have the sketch on the computer, you will need to upload it to the Arduino so it can use the program. Every time you change the sketch on the computer, you will need to upload to the Arduino so it has the latest version of your code.

When you click the upload button, the IDE **checks it for syntax errors** and then **compiles it into machine language** before **sending it to the Arduino through the USB cable**. Compiling is like translating into another language.

1. Press the ‘Upload’ toggle button on the toolbar (the top left corner of the window).

 © [ARDUINO LLC 2016](https://www.arduino.cc/) - CC-BY-SA 3.0

1. You should see a bar indicating the progress of the upload near the lower right corner of the Arduino integrated development environment (IDE).

 © [ARDUINO LLC 2016](https://www.arduino.cc/) - CC-BY-SA 3.0

1. The lights labelled TX and RX on the Arduino board will be blinking and on the ThinkerShield the D13 LED will be flashing while the data transfer takes place. TX and RX represent transmission of data and receiving data.

 © [ARDUINO LLC 2016](https://www.arduino.cc/) - CC-BY-SA 3.0

1. If the upload is successful, the IDE will display the message ‘*DONE UPLOADING’.*

 © [ARDUINO LLC 2016](https://www.arduino.cc/) - CC-BY-SA 3.0

What should happen when the Arduino is running?

If the sketch is uploaded successfully then you should see a blinking light on the ThinkerShield at D12.

If it is working, then you should have successfully retrieved a sketch (Arduino code), compiled it and uploaded it onto a microcontroller.



Blink sketch

If you do not have the code samples then you can copy and paste the code below into the Arduino IDE or you can type it in yourself.

| The blink sketch |
| --- |
| void setup() { // the setup function runs once when you press reset or power the board  pinMode(12, OUTPUT); // initialize digital pin 12 as an output.  }  void loop() { // the loop function runs over and over again forever  digitalWrite(12, HIGH); // turn the LED on (HIGH is the voltage level)  delay(1000); // wait for a second  digitalWrite(12, LOW); // turn the LED off by making the voltage LOW  delay(1000); // wait for a second  } |

Note:

Arduino boards have a built in LED on pin 13.We have used pin 12 instead of 13 in our sample sketch. This is because pin 13 is also used as the data transmission LED on the ThinkerShield and when data is transferring between the computer and Arduino you might think the code is working incorrectly.

Some keywords are written in capitals like OUTPUT, HIGH and LOW. These are reserved words that Arduino understands and they have a special meaning.

Program layout and structure for Arduino

Arduino programs can be divided into three main parts: ‘structure’, ‘values’ (variables and constants), and ‘functions’. A list of words used in Arduino language can be found in the Arduino Help menu or on the [Arduino reference](https://www.arduino.cc/en/Reference/HomePage) webpage.

Syntax – The spelling and grammar of coding

When coding, the spelling and grammar (syntax) you use must be absolutely correct. If you have any aspect of the code incorrect the computer will not know what to do, or will do something you don’t expect.

Camel case is used when words are joined together and use a capitalised letter to show separation. Commands in code are case sensitive. Camel case has been used for the name of the ThinkerShield.

Computers only do exactly what they have been told to do in the code!

Comments in Code and Programs

When coding, programmers will often add comments next to the lines of code. These comments are not read by the computer and are only there to help people reading the code.

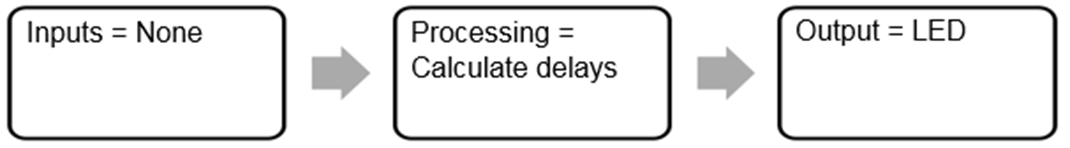
In the Arduino language, comments start with **//**. The software knows to ignore everything on the line after the //. Comments help you remember what each part of the program does, especially tricky bits and also are great if other programmers are helping you or adding to your code. Not every line needs a comment, however the examples in this unit have many comments to help explain what each line is doing. Good programmers write quite a few comments in their programs.

PRP #01: Plug Run Play – Blink

Now that you have uploaded your first sketch and watched blink happen we will now go deeper into the sketch. In the plug run plays (PRPs) you will upload basic examples of sketch and then modify them to complete challenges and will become familiar with coding terminology, syntax and structure. Each PRP introduces different coding features and the activities should be signed off by a partner or your teacher when finished.

Activity: Input Processing and Output (IPO) chart

In the spaces below, write what you think the inputs (sensors) and outputs (actuators) are for the blink sketch.



Pseudocode activity

In plain English, how would you write out a program (code) to make the classroom lights blink on and off every ten seconds?

| Step | Pseudocode |
| --- | --- |
| 1 | Turn light switch on |
| 2 | Wait ten seconds |
| 3 | Turn light switch off |
| 4 | Wait ten seconds |
| 5 | Repeat indefinitely |
| 6 |  |
| 7 |  |

Write the pseudocode for the blink sketch in the table below. You can add lines if you need more space

Activity: Making connections

Open the ‘\_01\_Blink’ sketch again. Compare the pseudocode you wrote on the previous page for turning the light on and off to the actual ‘Blink’ sketch and answer the following questions in the space provided:

1. Are there any similarities?

The commands are in the same order and processes are similar

1. Do they follow similar steps or sequence?

**Yes but language is different**

Activity: Understanding the code/sketch

| Command | What does it mean? |
| --- | --- |
| pinMode(12, OUTPUT); | The mode or function of pin 12 is to be an output, it will send current out of the Arduino board to what is connected to this pin. In this project it is an LED on pin 12 of the ThinkerShield. |
| void loop() { | This is the start of the code that will run continuously in a “loop” until the power is turned off or other code is uploaded to the Arduino. |
| digitalWrite(12, HIGH); | Turn pin 12 to “high” or turn the electrical flow on. |
| delay(1000); | Delay means to wait for the time written in brackets. The unit for time is millisecond.  E.g. (1000) = 1 second, (10000) = 10 seconds. |
| digitalWrite(12, LOW); | Turn pin 12 to “low” or turn the electrical flow off. |
| delay(1000); | Delay means to wait for the time written in brackets. The unit for time is millisecond. |

Have a look at the code and see if you can work out what the following commands might mean in the sketch.

Label the components of the Arduino sketch and program



© [ARDUINO LLC 2016](https://www.arduino.cc/) - CC-BY-SA 3.0

Word bank

* Command
* Comments
* Curly brace close
* Curly brace open
* Debug area
* Serial Monitor
* Semi colon
* Sketch title
* Status bar
* Upload
* Verify command
* Void setup

For each of the following aspects of programming, outline their function in the table below.

| Component | Function |
| --- | --- |
| Commands | Are an instruction given in a program that tells the computer to do something specific. The main ones we will work with are keywords. |
| Semicolon ; | To tell Arduino it is at the end of that line of code. |
| Void setup | This section of code initially runs once when the program is run, and is used to tell Arduino what to expect its input/output pins to be doing or connected to. |
| Void loop | This section of code runs over and over until the power is switched off or the program has an ending statement. |
| Curly braces { } | These begin and end sections of code and Arduino needs them to recognise “paragraph structure” in your program. |
| Comments | Are put into the program to remind the programmer or tell someone else looking at the program what is happening in that line or section of code. They are not compiled or ‘read’ by the compiler. |
| Serial monitor | To allow you to see what the program is doing “in the background” even if your circuit doesn’t appear to be working. Serial monitor is a good way to check your code is correct/working. |

Arduino language

Complete the following passage about the Arduino language using the word bank provided. Some of the missing words have been started for you.

Word bank

* Code
* Computers
* Development
* Environments
* Interactive
* Nothing
* Logical
* Pseudocode

Microcontrollers are mini **computers**. Microcontrollers and computers use a special type of programming language referred to as **code** so that they know what to do, how to do it and when to do it. Without code the microcontroller does **nothing**.

Different I **Integrated** D **Development** E **Environments** (IDEs) will use different programming languages. Just like people in different countries around the world use different languages to communicate.

P **Pseudocode** is the term for writing programming code in plain English. It involves writing step-by-step explicit instructions in a **logical** sequence.

Flowcharts / Branching Diagrams

A flowchart (also called a branching diagram) is a visual representation of the processes in a computer program or algorithm. It shows how different actions are performed depending on specified conditions. Coders often use flow charts to plan complex programs. Label the flow chart symbols below and explain what they are used for.

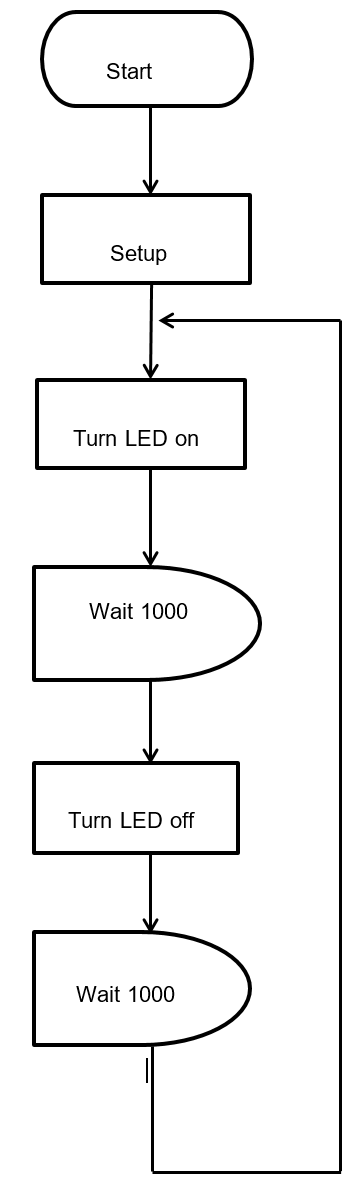
This shape is used to represent the start or end of a flowchart.

Rectangles represent a process where something happens in the program.

Diamonds are used to represent when a decision is made in the program.

This bullet shape is used to represent a delay in the program.

Draw the flow chart for the blink sketch below:



PRP #01: Blink – Challenges for you

Use the tables below to write the pseudocode for each challenge solution.

Change how long the LED flashes on for

| Step | Pseudocode |
| --- | --- |
| 1 | Turn LED 12 on |
| 2 | Wait 3 seconds |
| 3 | Turn LED 12 off |
| 4 | Wait 1 second |
| 5 |  |
| 6 |  |

Change how long the LED is off for.

| Step | Pseudocode |
| --- | --- |
| 1 | Turn LED 12 on |
| 2 | Wait 1 second |
| 3 | Turn LED 12 off |
| 4 | Wait 3 seconds |
| 5 |  |
| 6 |  |

Make two LEDs flash on and off at the same time.

| Step | Pseudocode |
| --- | --- |
| 1 | Turn LED 12 on |
| 2 | Turn LED 8 on |
| 3 | Wait 1 second |
| 4 | Turn LED 12 off |
| 5 | Turn LED 8 off |
| 6 | Wait 1 second |

Make the LEDs “chase” each other along from pin 12 to 9. Going on and off in alternate patterns.

| Step | Pseudocode |
| --- | --- |
| 1 | Turn LED 12 on |
| 2 | Wait 1 second |
| 3 | Turn LED 12 off |
| 4 | Turn LED 11 on |
| 5 | Wait 1 second |
| 6 | Turn LED 11 off (NOT WHOLE PSEUDOCODE) |

Make the LEDs come on in pairs or other patterns that you have thought of.

| Step | Pseudocode |
| --- | --- |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |

Get a partner or your teacher to check and tick if you have been successful in the following challenges.

| Challenge | Achieved? |
| --- | --- |
| Change how long the LED flashes for. |  |
| Change how long the LED is off for. |  |
| Make two LEDs flash on and off at the same time. |  |
| Make the LEDs “chase” each other along from pin 12 to 9. Going on and off in alternate patterns. |  |
| Make the LEDs come on in pairs or other patterns that you have thought of. |  |

Reflection of PRP#01

Answer these questions by completing the table:

| Task | Completed |
| --- | --- |
| Were you successful at getting the LED to blink? |  |
| Were you able to change the code and get the blink rate to change? |  |
| Were you able change the code and get the LEDs to flash in the pattern that you wanted? |  |

Write two things you have learnt so far about coding using Arduino in the space below.

Describe how you could improve your coding skills in the space below.

Teacher feedback

Binary – What you need to know

Complete the following passage about binary numbers in computing using the word bank provided. Some of the missing words have been started for you.

Word bank

* Binary
* C/C++
* Computer
* Digital
* Efficient
* Every
* Translator
* Language
* One
* Programming languages
* Sketch
* Syntax
* Theoretically
* Zero

Computers have different **programming languages**, you may have heard of some of them such as C, HTML, Python or Scratch.

Regardless of what language you program in they all convert to **binary** when running through the **computer**. This is because computer processors only work at base level as “on” or “off”.

They are a **digital** machine and only have **zero** or **one** (off or on) as a state to calculate. **Every** computer, regardless of its age or specification works on binary at its core. You need to understand that binary is like a counting system that can tell a computer what to do.

All a programming **language** does is make the binary system easier to understand and use by acting as an **interpreter** between plain human speak and computer speak. It is **theoretically** possible to program a computer using binary only, but this would be an enormous task and it is much more **efficient** using a programming language instead. These languages have dictionaries of commands, **syntax** and terminology preloaded for you to use.

This is what we have already been doing with the Blink **sketch** in Arduino. The language that Arduino uses is **C/C++.**

Computers also use binary to store and send data. This is because they are electronic and turn electricity on (one) or off (zero) to represent the data. Sometimes the terms high and low are used to represent on and off. There are other ways to store the ones and zeros that a computer reads, but we won’t go into that here. To learn more, search the internet for “data storage”.

Each digit is called a bit. This will be a 1 or 0.

Computers store bits in blocks of 8. Each block of eight bits is called a byte.

Normally people count in base ten. In binary we count in base two.

| 27s | 26s | 25s | 24s | 23s | 22s | 2s | Units | In Base 10 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 128s | 64s | 32s | 16s | 8s | 4s | 2s | 1s | Calculations |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 2 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | (2+1) = 3 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 4 |
| 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | (32+8+2) = 42 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | (128+64+32+16+8+4+2+1) = 255 |

Here are some larger numbers:

| 27s | 26s | 25s | 24s | 23s | 22s | 2s | Units | In Base 10 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 128s | 64s | 32s | 16s | 8s | 4s | 2s | 1s | Calculations |
| 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | (32+8+2) = 42 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | (128+64+32+16+8+4+2+1) = 255 |

Each byte represents a number between 0 and 255. There are 256 numbers in total but computers consider zero to be the first counting number so the highest number is 255.

You might recognise some of these numbers from “computer talk”.

The cover on this booklet has a series of binary numbers. Each group of eight bits represents a number or letter for the computer. Copy the text and paste it into a binary translator on the internet. It will tell you who wrote this unit of work.

**All data used by a computer is stored in binary.**

Use an internet binary translator to write a message in binary in the space below.

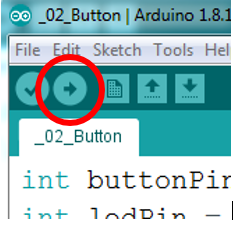
PRP #02: Digital input – Button

Locate and load the sketch “\_02\_Button” that has been provided by your teacher. This sketch uses the button to control an LED on the ThinkerShield.



Uploading the sketch

1. Open the sketch \_02\_Button.



© [ARDUINO LLC 2016](https://www.arduino.cc/) - CC-BY-SA 3.0

1. Click the ‘Upload Program’ button to run the sketch. Click the upload button to load the sketch onto the Arduino.

Activity: IPO chart of the button sketch

Complete the IPO chart to control a Light Emitting Diode (LED) with a button.

IPO chart of the button sketch
Input = Button. Processing = If button on, then turn on LED
Output = LED.

Sketch for PRP #02 – Button (switch)

This sketch uses a button that is either on or off. That is, produces a binary input.

| The button sketch |
| --- |
| int buttonPin = 7; // the number of the pushbutton pin  int ledPin = 12; // the number of the LED pin  int buttonState = 0; // variable for reading the pushbutton status  void setup() {  pinMode(ledPin, OUTPUT); // initialise the ledPin as an output:  pinMode(buttonPin, INPUT); // initialise the buttonPin as an input:  }  void loop() {  buttonState = digitalRead(buttonPin); // read the state of the pushbutton value:  if (buttonState == HIGH) { // check if the pushbutton is pressed.  digitalWrite(ledPin, HIGH); // turn LED on if the buttonState is HIGH:  } else {  digitalWrite(ledPin, LOW); // turn LED off if the buttonState is LOW:  }  } |

if statements

The PRP #2 sketch uses an ‘if statement’.

The ‘if statement’ is used to allow the program to make a decision based on certain conditions such as the state of an input. These commands can be known as conditionals.

An example could be a night light. The processor is making the decision to turn on a light if it is dark.

The if statement checks whether the condition in the brackets is true or false.
If it is true the operations between the next set of curly braces will be performed. If not the sketch will perform the operations between the curly brackets after the Else command.

Comparison operators

| Symbol | Meaning |
| --- | --- |
| == | Equal to |
| != | Not equal to |
| < | Less than |
| > | Greater than |
| <= | Less than or equal to |
| >= | Greater than or equal to |

Note: In most computer languages we use a single = to assign a value to something.

We have to use a double == to compare two values.

Variables – Using the integer “int” command

There are shortcuts in coding that tell the computer what to do with a minimum of commands/lines of code.

At the start of the blink sketch this line runs in the setup section between the {} curly braces: “pinMode (12, OUTPUT)”;

You can give pin 12 (or any integer) a name in the code/sketch by using the int command. In this case it makes it easier to remember what is hooked up to your circuit or ThinkerShield when you are coding. The command you can use in your code/sketch is the “int” command. An example of how to code/sketch this is below. You will notice the rest of the code is basically the same as blink, except that instead of the pin number being in the loop, the name you gave the pin is there. It is good practice to use names that make sense or give clues to the components that are attached to the Arduino. So in this example instead of just using “12” we have used “LED” to tell anyone reading the program that it is an LED.

Using the int command also allows us to change a pin connection easily. If we needed to use pin 8 for the LED instead of 12, we would only need to change it in the int command rather than all the way through the sketch.

The section before the setup is called the declarations as it is where we declare that a word (variable) is equal to an integer (whole number).

When declaring a variable, the word (string) you use cannot be one recognised by the Arduino software, so if it changes colour, you need to use a different word.

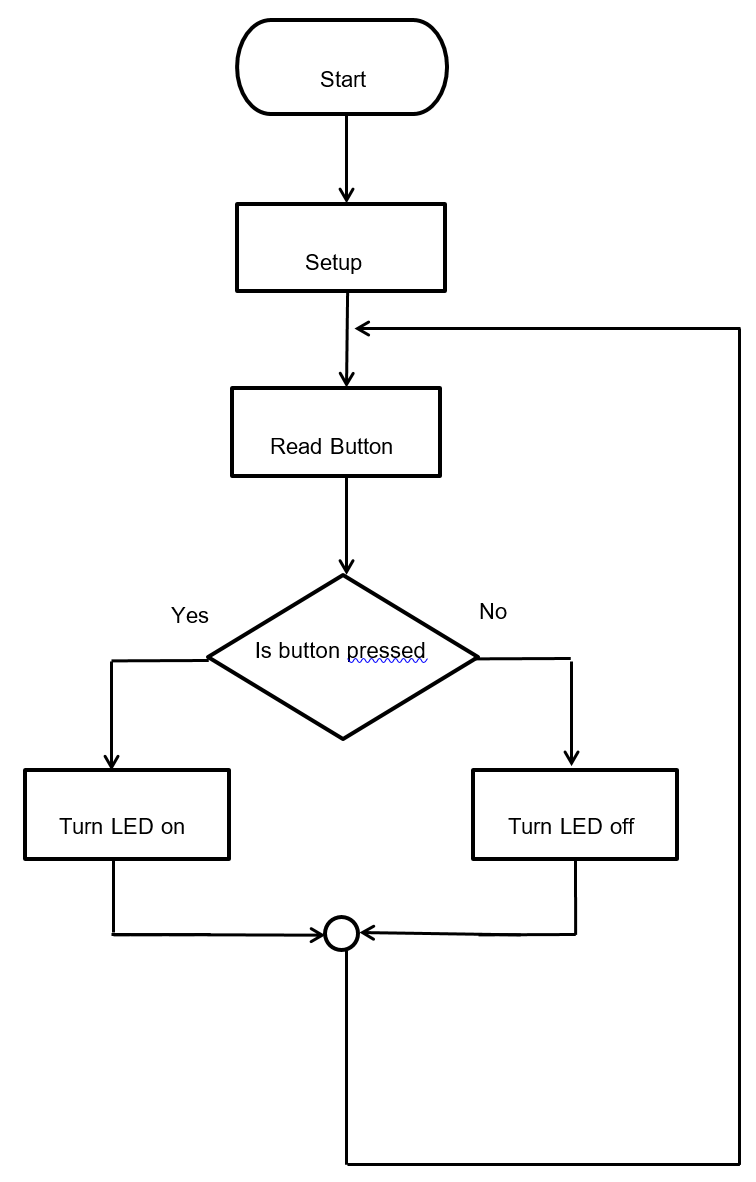
| The int command: ‘\_03\_int’ |
| --- |
| int LED = 12; // tell Arduino that pin 12 is going to be called “LED” for the following sketch  void setup() {  pinMode(LED, OUTPUT); // initialize digital pin 12 as an output.  }  void loop() { // the loop function runs over and over again forever  digitalWrite(LED, HIGH); // turn the LED on (HIGH is the voltage level)  delay(1000); // wait for a second  digitalWrite(LED, LOW); // turn the LED off by making the voltage LOW  delay(1000); // wait for a second  } |

Pseudocode activity

Write out the pseudocode that turns on the LED when the button is pressed and turns off when it is released. The table below is to get you started, it may have too many lines or it may not have enough!

| Step | Pseudocode |
| --- | --- |
| 1 | Check if the button is pressed on or off |
| 2 | If the button is not pressed on then do nothing |
| 3 | If the button is pressed on then turn on the light |
| 4 | When the button is not pressed on then turn off the light |
| 5 | Repeat indefinitely |
| 6 |  |
| 7 |  |
| 8 |  |
| 9 |  |
| 10 |  |
| 11 |  |
| 12 |  |
| 13 |  |

Draw a flowchart for the button sketch below.



Activity: Making connections

Compare the pseudocode above to your PRP#02 Button sketch and complete the following questions in the space provided.

Are there any similarities between the pseudocode and sketch?

Do they follow similar steps or sequence?

PRP #02: Button – Challenges for you

Reverse the switch action so the button/switch does the opposite to what it did in the original code/sketch.

Write out the pseudocode to the challenge in the table below before you attempt to code it. (Keep going on a separate sheet if you need more space). Use the flowchart to help solve the problem.

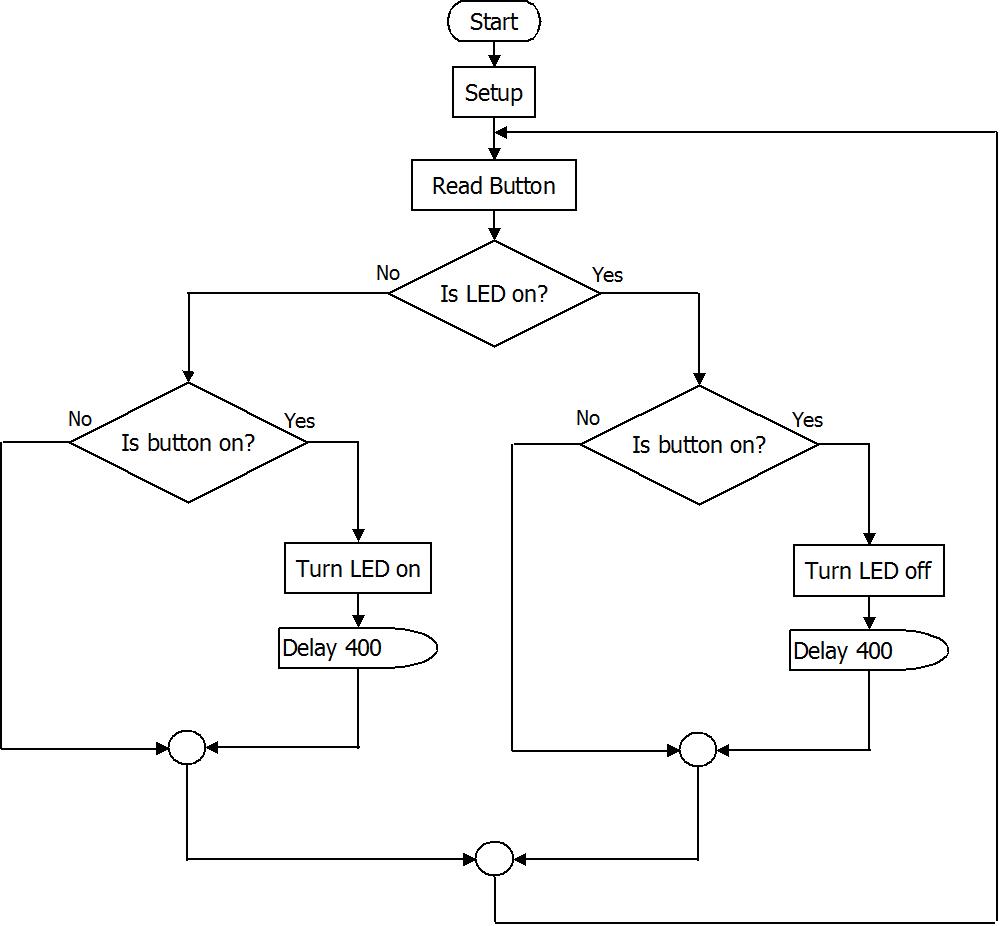
| Step | Pseudocode |
| --- | --- |
| 1 | Check if the button is pressed on or off |
| 2 | If the button is not pressed on then turn on the LED |
| 3 | If the button is pressed on then turn off the LED |
| 4 | When the button is not pressed on then turn on the LED |
| 5 | Repeat indefinitely |
| 6 |  |
| 7 |  |
| 8 |  |

Make the LED stay on for 3 seconds before turning off when the button is released. Write out the pseudocode to the challenge in the table below before you attempt to code it.

| Step | Pseudocode |
| --- | --- |
| 1 | Check if the button is pressed on or off |
| 2 | If the button is not pressed on then turn LED off |
| 3 | If the button is pressed on then turn the LED on |
| 4 | Wait three seconds |
| 5 | Repeat indefinitely |
| 6 |  |
| 7 |  |
| 8 |  |

Turn the button into a “toggle switch” where the LED stays on when the button is first pressed and then turns off when the button is pressed again. (This is a very difficult challenge that you may not be able to solve quickly).

The flowchart below could help you write your pseudocode and sketch to complete the challenge.



Write out the pseudocode to the above challenge in the table below before you attempt to code it. (Keep going on a separate sheet if you need more space).

| Step | Pseudocode |
| --- | --- |
| 1 | If button is off |
| 2 | Do nothing |
| 3 | If LED is on and button is on |
| 4 | Turn LED off |
| 5 | If LED is Off and Button is on |
| 6 | Turn LED on |
| 7 | Repeat |
| 8 |  |

PRP #02: Button – Challenges check off

Get a partner or your teacher to check and tick if you have been successful in the following challenges.

| Challenge | Achieved? |
| --- | --- |
| Reverse the switch action so the button/switch does the opposite to what it did in the original code/sketch. i.e. get the LED to always be on unless the button is pressed |  |
| Make the LED stay on for 3 seconds before turning off when the button is released. |  |
| Turn the button into a “toggle switch” where the LED stays on when the button is first pressed and then turns off when the button is pressed again. |  |

Reflection of PRP#02

Answer these questions by completing the table:

| Task | Completed |
| --- | --- |
| Were you successful at getting the LED to come on when the button is pressed? |  |
| Were you able to change the code and get the LED to always be on unless the button is pressed? |  |
| Were you able change the code and get the LED to stay on for three seconds after the button is pressed? |  |
| Were you able to turn the button into a “toggle switch” where the LED stays on when the button is first pressed and then turns off when the button is pressed again? |  |

Write two things you have learnt so far about coding using Arduino and the ThinkerShield in the space below.

Teacher feedback

Input components or sensors

An input component is also known as a sensor as it sends a signal when it detects change in its environment, like human senses detect change in their environment. There are many different sensors that are used in different automatic control systems such as:

* Buttons/switches
* motion sensors
* light sensors
* sound sensors
* level sensors
* pressure sensors
* thermal sensors
* mechanical sensors

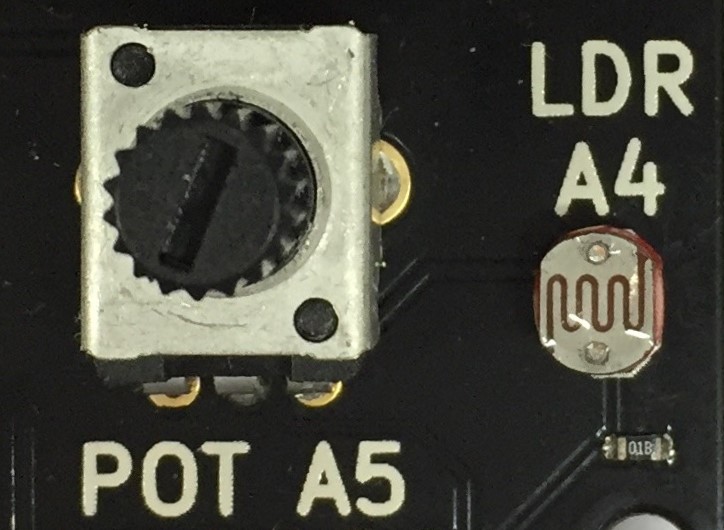
Activity

For each of the inputs listed above, list two examples of common items that use these types of sensors in the table below.

| Input/Sensor component | Example of a common item that uses the component |
| --- | --- |
| Switch | Air conditioner in a car |
| Motion sensor | Security light on a house |
| Light sensor | Driveway lighting for a house |
| Sound sensor | Sound activated children’s toy |
| Level sensor | A fuel tank in a car |
| Pressure sensor | Digital weight scales |
| Thermal sensor | Weather monitoring station |
| Mechanical sensor (potentiometer) | Volume control knob on a home theatre system |

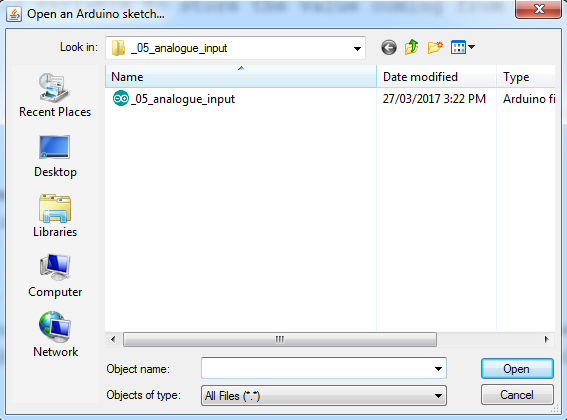
PRP #03: Analog input – potentiometer and Light Dependant Resistor (LDR)

Follow the instructions below to load the sketch called “\_05\_analog\_Input”. This sketch will allow you to use a potentiometer or a Light Dependant Resistor (LDR) to control the LED on the ThinkerShield. A potentiometer which is marked as POT A5 on the ThinkerShield is also known as a variable resistor. This basically means that by turning the knob the voltage is increased and decreased. The LDR works in a similar way, however the voltage increases and decreases depending on the amount of light present.



Uploading the sketch

Open the sketch ‘\_05\_analogue\_input’



© [ARDUINO LLC 2016](https://www.arduino.cc/) - CC-BY-SA 3.0.

Activity: Input Processing and Output (IPO) chart

Complete the IPO chart to control the Light Emitting Diode (LED) with the potentiometer.

IPO chart. What are the inputs and outputs to control a Light emitting diode with a potentiometer.
Input = Potentiometer.
Processing = read potentiometer and use to adjust delay.
Output = Blink rate of LED.


Sketch for PRP #03: AnalogInput

Below is a copy of the \_05\_analog\_input sketch.

| \_05\_analog\_input sketch |
| --- |
| int sensorPin = A5 // select the input pin for the potentiometer on the ThinkerShield  int ledPin = 12; // select the pin for the LED – 12 on the ThinkerShield  int sensorValue = 0; // variable to store the value coming from the sensor  void setup() {  pinMode(ledPin, OUTPUT); // declare the ledPin as an OUTPUT  Serial.begin(9600); // tell the software to begin transmitting the value given below using serial communication  }  void loop() {  sensorValue = analogRead(sensorPin); // read the value from the sensor  digitalWrite(ledPin, HIGH); // turn the ledPin on  delay(sensorValue); // stop the program for some time  digitalWrite(ledPin, LOW); // turn the ledPin off  delay(sensorValue); // stop the program for some time  // Serial.println("The sensor value is " + sensorValue);  // in serial monitor write “The sensor value is”  Serial.print("The sensor value is: "); // put the sensor value in serial monitor after the wording given in line above this.  Serial.println(sensorValue);  } |

.

Pseudocode activity

Write out the pseudocode in the table below to make the LED blink faster and slower depending on how far you turn the potentiometer “up” and “down”.

| Step | Pseudocode |
| --- | --- |
| 1 | Read the potentiometer (possible values from the potentiometer that can be returned varies between 0 and 1023 inclusive). |
| 2 | Turn on the LED |
| 3 | Wait for a delay time according to the value from the potentiometer |
| 4 | Turn off the LED |
| 5 | Wait for a delay time according to the value from the potentiometer |
| 6 | Repeat the above steps |
| 7 |  |
| 8 |  |
| 9 |  |

Activity: Making connections

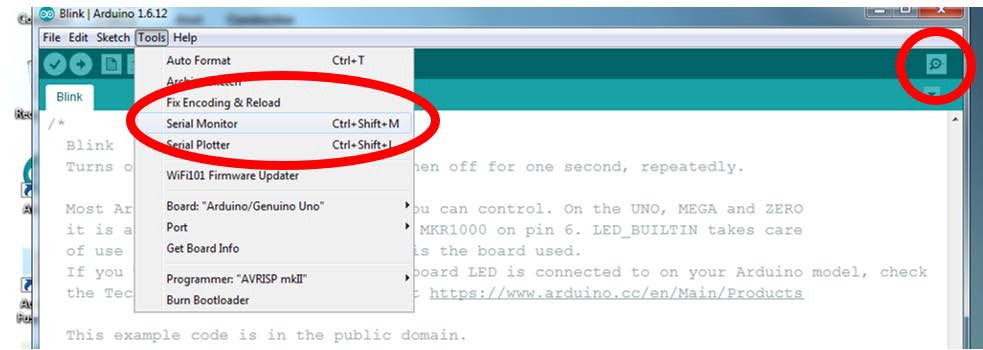
Compare your pseudocode to your \_05\_Analog\_Input sketch and complete the following questions in the space provided below.

Are there any similarities?

Do they follow similar steps or sequence?

Serial monitor

When running programs your code may not appear to be working properly. There is a screen in Arduino software known as serial monitor that can indicate what is happening in the program and allow you to check if your program functions properly. This is where to find it (there are two places circled):



© [ARDUINO LLC 2016](https://www.arduino.cc/) - CC-BY-SA 3.0

Once you have opened serial monitor nothing happens unless you add code into your sketch telling the IDE what to display data on the serial monitor. The sketch \_05\_analog\_input has the code required to use the serial monitor. Run the serial monitor with the sketch on the Arduino to see what happens as you turn the potentiometer.

The commands you need are here:

Serial.begin (9600);

This line is in the Setup. The 9600 refers to the communication speed between the Arduino and the computer. It must match the number in the bottom of the serial monitor window.

Serial.println (sensorValue);

This line prints the value in the brackets () to the computer via the USB cable. In this case it is the value that was read from the potentiometer. After printing the value to the screen, it starts a new line.

Serial.print (“TEXT”);

This line prints the value in the brackets () to the computer via the USB cable but does not start a new line so it will print across the screen instead of down. As the word TEXT is in quotation marks, it will print the word TEXT on the monitor.

Below is a simple sketch that demonstrates the use of the Serial Monitor. It reads the value on the potentiometer, assigns this value to “potValue” and prints this value to the serial monitor.

| Sketch |
| --- |
| int potSensorPin = 5; // The potentiometer is connected to analog pin 5  int dlyTime = 500; // 1/4 of a second delay between readings  int potValue; // variable to store the value read from the potentiometer  void setup() // this function runs once when the sketch starts up  {  Serial.begin(9600); // setup serial communication  }  void loop() {  potValue = analogRead(potSensorPin);  Serial.println(potValue); // Send sensor value to serial monitor  delay(dlyTime); // Pause for dlyTime milliseconds  } |

PRP #03: Analog input – Challenges for you

Change the code to get multiple LEDs to flash. Write out the pseudocode in the table below before you attempt to code it.

| Step | Pseudocode |
| --- | --- |
| 1 | Read the potentiometer (possible values from the potentiometer that can be returned varies between 0 and 1023 inclusive). |
| 2 | Turn on: LED 9, LED 10, LED 11, LED 12 |
| 3 | Wait for a delay time according to the value from the potentiometer |
| 4 | Turn off: LED 9, LED 10, LED 11, LED 12 |
| 5 | Wait for a delay time according to the value from the potentiometer |
| 6 | Repeat the above steps |
| 7 |  |
| 8 |  |

Reverse the potentiometer action so the blink rate speeds up and slows down in the opposite direction to the original sketch. Write out the pseudocode in the table below before you attempt to code it.

| Step | Pseudocode |
| --- | --- |
| 1 | Read the potentiometer (possible values from the potentiometer that can be returned varies between 0 and 1023 inclusive). |
| 2 | Turn on the LED |
| 3 | Wait for a delay time of 1023 minus value from the potentiometer |
| 4 | Turn off the LED |
| 5 | Wait for a delay time of 1023 minus value from the potentiometer |
| 6 | Repeat the above steps |
| 7 |  |
| 8 |  |

Use the POT to make different LEDs light up as you turn the potentiometer. Write out the pseudocode below before you attempt to code it.

| Step | Pseudocode |
| --- | --- |
| 1 | Read the potentiometer (possible values from the potentiometer that can be returned varies between 0 and 1023 inclusive). |
| 2 | If potentiometer value is greater than 0 and less than 255 |
| 3 | Turn LED 9 on |
| 4 | If potentiometer value is greater than 255 and less than 512 |
| 5 | Turn LED 10 on |
| 6 | If potentiometer value is greater than 512 and less than 1023 |
| 7 | Turn LED on 11 |
| 8 | Repeat the above steps |

Edit the code to use the LDR as an input device that turns the LED on when it gets dark (like a night light). Hint: Use the serial monitor to indicate the value of the LDR at different light levels

| Complete a flowchart in the space below to represent your algorithm for the challenge before writing the pseudocode.  Flow chart for LDR night light |
| --- |

Write out the pseudocode in this table.

| Step | Pseudocode |
| --- | --- |
| 1 | Read the LDR (possible values from the LDR that can be returned varies between 0 and 1023 inclusive). |
| 2 | Is LDR Value less than 300 (Adjust for light level) |
| 3 | If true turn LED on |
| 4 | If false turn LED off |
| 5 | Repeat |
| 6 |  |
| 7 |  |
| 8 |  |

PRP #03: Analog Input – Challenges for you

Get a partner or your teacher to check and tick if you have been successful in the following challenges.

| Challenge | Achieved? |
| --- | --- |
| Change the code and get multiple LEDs to flash. |  |
| Reverse the potentiometer action so the blink rate speeds up and slows down in the opposite direction to the original sketch. |  |
| Use the POT to make the LED scroll through from one to the next as you turn the dial. |  |
| Edit the code to use the LDR as an input device that turns the LED on when it gets dark (like a night light). |  |

Reflection of PRP #03:

Answer these questions by completing the table:

| Task | Completed |
| --- | --- |
| Were you successful in getting multiple LEDs flash when the potentiometer was turned? |  |
| Were you able to reverse the potentiometer action so the blink rate speeds up and slows down in the opposite direction to the original sketch |  |
| Were you able to use the POT to make the LED scroll through from one to the next as you turn the dial |  |
| Were you able to make a night light? |  |

Write two things below that you have learnt so far about coding using Arduino and the ThinkerShield.

Teacher feedback

Outputs

An output component is also known as an actuator as it performs a task or action once it has been given a command. Actuators may activate another system, communicate something to the user or physically perform a task. There are many different actuators that are used in different automatic control systems including:

* light
* sound
* Motion

Activity

For each of the outputs listed above, list two examples of common items that use these types of actuator components in the table below.

| Input/Sensor component | Example of a common item that uses the component |
| --- | --- |
| Light | Low fuel warning light in a car, light to indicate that an oven has reached temperature. |
| Sound | Warning buzzers on a car reversing system, alarm to indicate that a 3D printer is not working properly. |
| Motion | School room alarm system. |

PRP #04: Digital output – buzzer

This PRP will help you to code the Arduino to play tones.



Activity: IPO chart

Identify the inputs and outputs of this IPO chart.

IPO chart.
Input = None.
Processing = calculate frequency / length.
Output = buzzer sound.

Upload the sketch

| Buzzer Sketch: ‘\_07\_Buzzer1’ |
| --- |
| int buzzerPin = 7; //Declares the value of buzzerPin to be 7  void setup()  {  pinMode(buzzerPin, OUTPUT); // Initialise the buzzerPin as an output.  }  void loop()  {  tone(buzzerPin, 600, 30); delay(150); // Play a tone with the buzzer. frequency 600, duration 30ms then delay 150ms.  } |

A note about cutting and pasting of code

There is plenty of code available on the internet that can be used to complete your projects. The skill is knowing how to manipulate what you find so it will do what you want. Often with simple programs you’ll spend longer debugging than if you wrote the program yourself. When you first start coding it is good practice to type programs out manually rather than cut and paste so you can get proficient at terminology, syntax and the structure of a language.

Pseudocode Activity

Write out the pseudocode to make the buzzer sound below.

| Step | Pseudocode |
| --- | --- |
| 1 | Send electrical current to the buzzer on pin 3 for 30 milliseconds |
| 2 | Wait 150 milliseconds |
| 3 | Repeat |
| 4 |  |
| 5 |  |
| 6 |  |
| 7 |  |
| 8 |  |
| 9 |  |
| 10 |  |

A sketch to make the buzzer change tone like an alarm

| Buzzer Sketch 2: ‘\_08\_Buzzer2’ |
| --- |
| int lowTone = 400;  int highTone = 600;  const int ALARMSOUNDER = 3;  void setup() {  pinMode(ALARMSOUNDER, OUTPUT); }  void loop(){  tone(ALARMSOUNDER, lowTone,500); //Play lowTone frequency for half a second  delay(500);  tone(ALARMSOUNDER, highTone,500); //Play highTone frequency for half a second  delay(500);  } // end loop() |

The example below uses a ‘for’ command. This is beyond the scope of the unit but has been included as an extension activity.

| Buzzer Sketch 3: ‘09\_Buzzer3’ |
| --- |
| int i = 200;  const int ALARMSOUNDER = 3;  void setup() {  pinMode(ALARMSOUNDER, OUTPUT); }  void loop(){  for(i = 200; i < 700; i = i + 2) { // sound a range of tones from low to high,  tone(ALARMSOUNDER, i,10); //Play tone changing every 10 millisecond  delay(10); }  for(i = 701; i > 200; i = i - 2) { // reverse the above range of tones (high to low),  tone(ALARMSOUNDER, i,10); //Play tone changing every 10 milliseconds  delay(10); }  } // end loop() |

PRP#04: Buzzer – Challenges for you

Change the code and get the buzzer to play different tones (two or more). The Arduino examples under “Digital > Tone” will help you work it out.

Write out the pseudocode to the challenge in this table before you attempt to code it.

| Step | Pseudocode |
| --- | --- |
| 1 | Play a frequency of 600 for half a second |
| 2 | Wait half a second |
| 3 | Play a frequency of 400 for half a second |
| 4 | Wait half a second |
| 5 | Repeat |
| 6 |  |

Code the buzzer to only come on when the button is pressed.

| Complete a flowchart to represent your algorithm for the challenge before attempting the pseudocode.  flow chart for buzzer button sketch |
| --- |

Write out the pseudocode to the challenge in this table before you attempt to code it.

| Step | Pseudocode |
| --- | --- |
| 1 | Check if button is pressed |
| 2 | Is button on? |
| 3 | If true play tone for 10ms |
| 4 | If false do nothing |
| 5 | Repeat |
| 6 |  |
| 7 |  |
| 8 |  |

Code the buzzer to start playing when the button is pressed once and then stop once the button is pressed once again.

| Complete a flowchart in this space to represent your algorithm for the challenge before attempting the pseudocode.  Flow chart for toggle buzzer sketch |
| --- |

Write out the pseudocode to the challenge in this table before you attempt to code it.

| Step | Pseudocode |
| --- | --- |
| 1 | Read button |
| 2 | Is button on and tone on? |
| 3 | If true do nothing |
| 4 | If false play tone for 20ms |
| 5 | Is button on and tone off? |
| 6 | If true play tone 20ms |
| 7 | If false do nothing |
| 8 | repeat |

Code the buzzer to play a familiar tune e.g. imperial march from star wars, Pokémon go theme.

Write out the pseudocode to the challenge in this table before you attempt to code it

| Step | Pseudocode |
| --- | --- |
| 1 | SEE SAMPLE SKETCH AT BACK OF BOOK |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |
| 7 |  |
| 8 |  |

PRP#04: Buzzer – Challenges for you to check

Get a partner or your teacher to check and tick if you have been successful in the following challenges.

| Challenge | Achieved? |
| --- | --- |
| Change the code and get the buzzer to play different tones (two or more). The Arduino examples under “Digital > Tone” will help you work it out. |  |
| Code the buzzer to only come on when the button is pressed. |  |
| Code the buzzer to start playing when the button is pressed once and then stop once the button is pressed once again. |  |
| Code the buzzer to play a familiar tune e.g. imperial march from Star Wars. |  |

Reflection of PRP #04:

Answer these questions by completing the table:

| Task | Completed |
| --- | --- |
| Did you successfully get the buzzer to play a sound? |  |
| Were you successful at changing the code and getting the buzzer to play different tones (two or more)? |  |
| Were you able to code the buzzer to only come on when the button is pressed? |  |
| Were you able to code the buzzer to start playing when the button is pressed once and then stop once the button is pressed once again? |  |
| Were you able to code the buzzer to play a familiar tune e.g. imperial march from star wars? |  |

Write two things you have learnt so far about coding using Arduino and the ThinkerShield in the space below.

Teacher feedback

Other commands in Arduino – Extension work

Below are brief explanations of some Arduino commands used in some of the sketches above. These are extension activities that will further develop your coding skills and knowledge. More information about these commands can be found in the Arduino reference at <https://www.arduino.cc/reference/en/>

analogWrite

One of the commands for Arduino that was used in the code above is analogWrite. This command can be used by the Arduino to control an analogue output, in this case, the brightness of an LED. In the \_01\_blink sketch we used digitalWrite to turn the LED on or off but in order to control the brightness we need to use analogWrite.

analogWrite (ledPin, fadeValue);

In this example fadeValue is a number between 0 and 255 and describes how bright or intense the LED is to shine.

const

The command const is short for constant. A constant is value assigned to a string that a programmer does not want to be changed. For example you could create a constant called Pi and give it the value 3.14, we would not want it to change to 5.23.

for loops

The for command is a looping command that won’t run forever like the void loop command. The for loop runs a certain number of times and then stops. In the example above – Buzzer Sketch 3, the for loop starts at 200 and continues while its value is less than 700. It will increment (count) up by a value of 2 each time it is ‘looped’.

millis

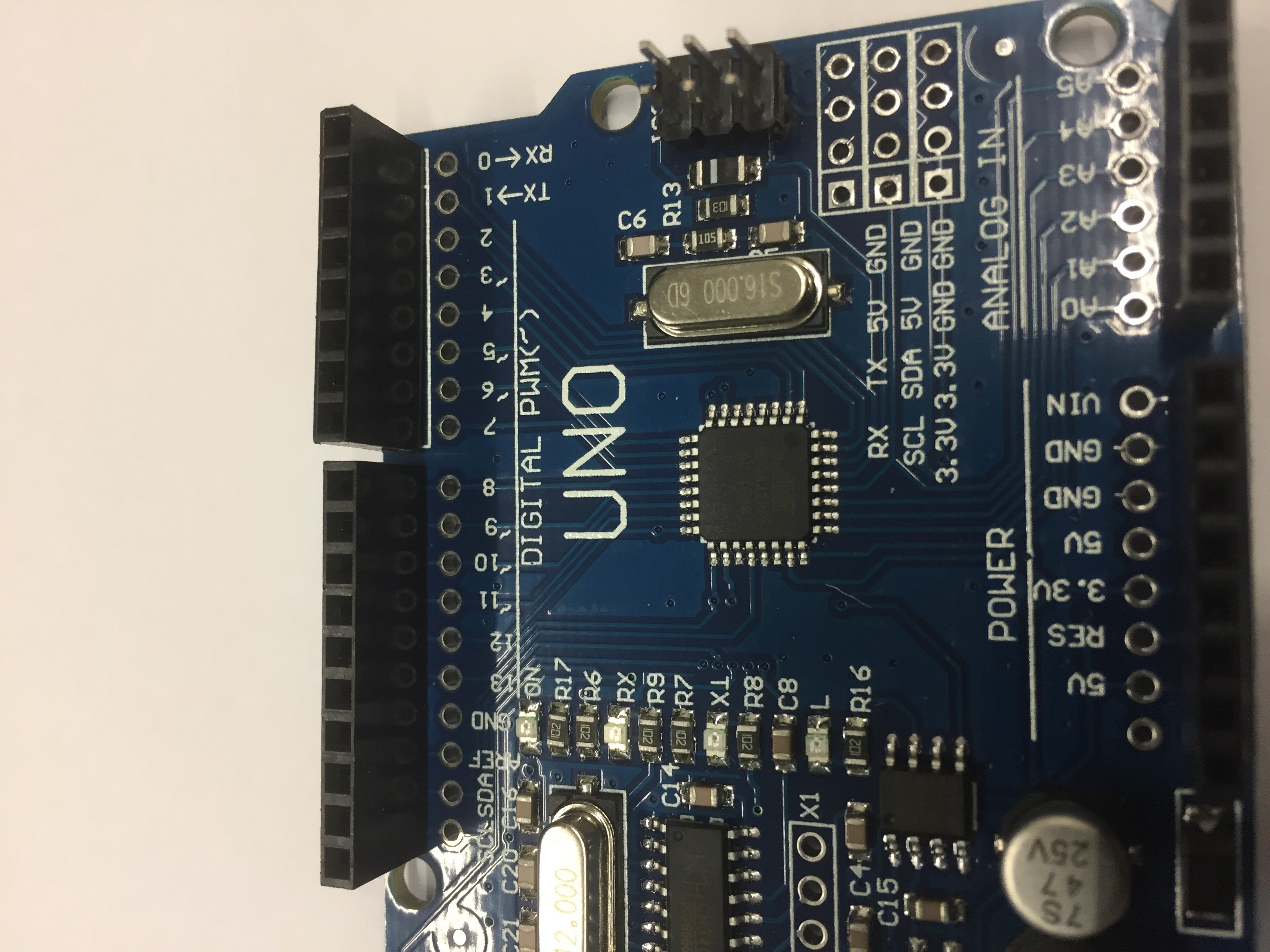
The millis command gives the number of milliseconds since the Arduino began running the program. This could be a useful command to use in the alarm code to provide time delays. Use the reference to find the correct syntax for the millis command.

Pulse Width Modulation

LEDs work differently to traditional light globes. Traditional globes can be dimmed by reducing the voltage supply. For example if a halogen globe has 12 volts applied it will be very bright but if the voltage is reduced to 6 volts it will be much dimmer. This is an analogue dimming system.

LEDs are dimmed by turning them on and off quickly. So fast that you cannot see it flashing. The LED turns on and off for different times to get different brightness. For example if the LED is on and off for the same amount of time (on for a millisecond, off for a millisecond) the light will be dim. If it is on twice as long as it is off it will be brighter. If it is on all the time it is at its brightest. This digital dimming system behaves like an analogue system. It is called Pulse Width Modulation (PWM).

The ports on the Arduino that have a ~ symbol next to them can use PWM. The PWM is controlled with a number between 0 and 255. These are pins 3, 5, 6, 9, 10 and 11.



Upload the sketch called \_04\_fade to the Arduino to see how it works and have a tinker.

| More of the int command: ‘\_04\_fade’ |
| --- |
| int led = 9; // the PWM pin the LED is attached to  int brightness = 0; // how bright the LED is  int fadeAmount = 5; // how many points to fade the LED by  void setup() { // the setup routine runs once when you press reset:  pinMode(led, OUTPUT); // declare pin 9 to be an output:  }  void loop() { // the loop routine runs over and over again forever  analogWrite(led, brightness); // set the brightness of pin 9:  brightness = brightness + fadeAmount; // change the brightness for next loop:  if (brightness <= 0 || brightness >= 255) {  fadeAmount = -fadeAmount; // reverse the direction of the fading at the ends of the fade  }  delay(30); // wait for 30 milliseconds to see the dimming effect  } |

Enhance the quality of the code

There are many “shortcuts” in programming that can perform functions quickly for you, rather than having to write out in long form what you want to happen. These shortcuts are sub programs that the computer uses to work out what to do. Some ones to get started with are:

* Arrays - <https://www.arduino.cc/en/Reference/Array>
* For Loop - <https://www.arduino.cc/en/Reference/For>
* While Loop - <https://www.arduino.cc/en/Reference/While>

Change some of the variable values

For each of the PRP activities you have been challenged to change the basic program to do something different. A lot of the challenges involved changing variables. Sometimes changing variables can make a program behave very differently. For example the buzzer code in PRP#04 changes variables to play different tunes. More information is available here: <https://www.arduino.cc/en/Reference/VariableDeclaration>

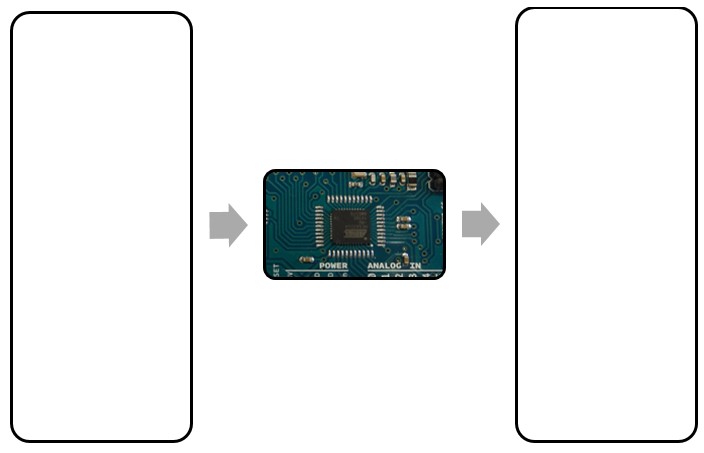
Managing your design project

Complete the Know Want Learned (KWL) chart below. Review the following to plan what you need to do to achieve success in this project.

* Design brief
* Constraints
* Criteria for Success

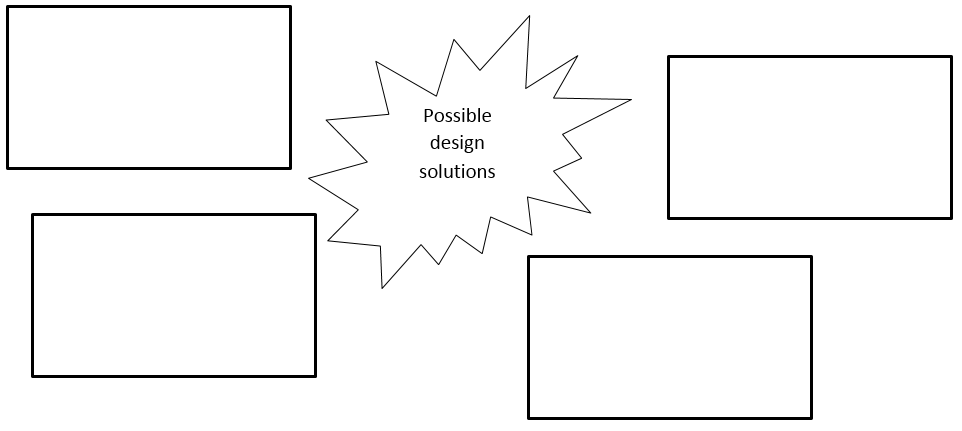
| What do I know already? | What do I want to know? | What I have learned? |
| --- | --- | --- |
|  |  |  |

Consider the components of an alarm system. List the potential sensors and actuators that could be used based on what you have already learnt in this unit on the IPO chart below.



Brainstorm of possible design solutions

After reviewing the design brief and what you already know, brainstorm possible ideas for your ‘Crack the Code’ design project.



Time/action plans

Identify in logical order, the activities you need to complete and when you intend to complete them. You will evaluate each action, outlining the reasons for it being complete or incomplete when you expected. Use the table below for your planning.

| Action/Activity | Time of expected completion | Ongoing evaluation |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Finance planning/costs

Your school fees provide the consumable items (things you use) required to complete your project. In the table below, identify the consumable items you will need for the project and any additional items you may wish to acquire yourself for the project.

| Consumable items you will be provided | Additional items you may wish to provide |
| --- | --- |
|  |  |

List websites where you can source project components in the space provided.

Generating and developing ideas

You must develop at least 4 basic control technology designs. These ideas must include:

* An IPO chart that identifies the input and output components.
* An end-use application

You need to consider your knowledge and understanding of coding and how long it will take you to write and test the program. You can use the internet to assist you with your design development if needed.

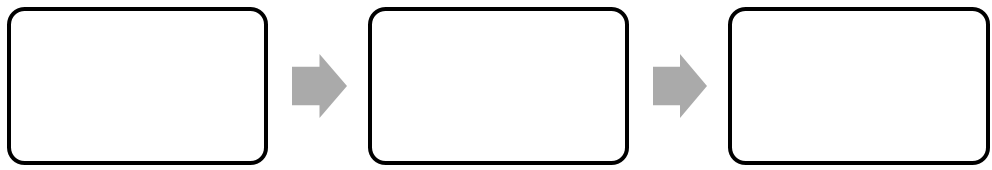
Each design must be evaluated using ‘Plus, Minus and Interesting (PMI)’.

* P (plus) = the good points in your design, its strengths
* M (minus) = the bad points, the designs weaknesses
* I (interesting) = things in your design that you want to remember for later or include in other design solutions.

Design 1

| Design name | End-use application/s |
| --- | --- |
|  |  |

IPO chart of the control system:



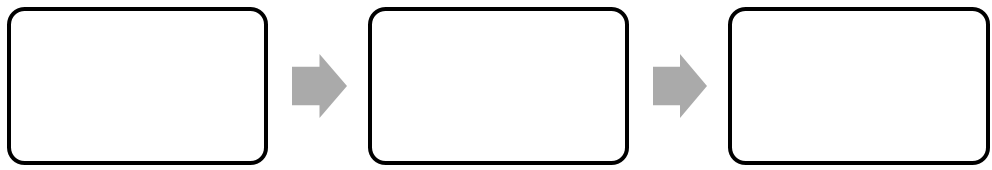
Evaluation:

| P | M | I |
| --- | --- | --- |
|  |  |  |

Design 2

| Design name | End-use application/s |
| --- | --- |
|  |  |

IPO chart of the control system:



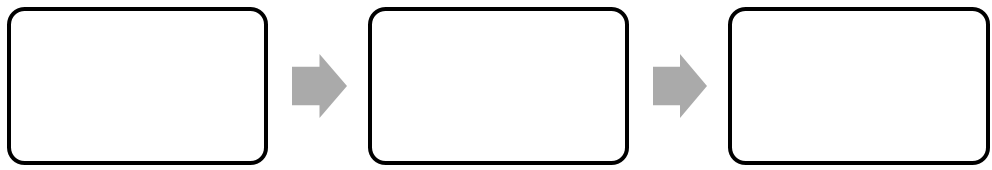
Evaluation:

| P | M | I |
| --- | --- | --- |
|  |  |  |

Design 3

| Design name | End-use application/s |
| --- | --- |
|  |  |

IPO chart of the control system:



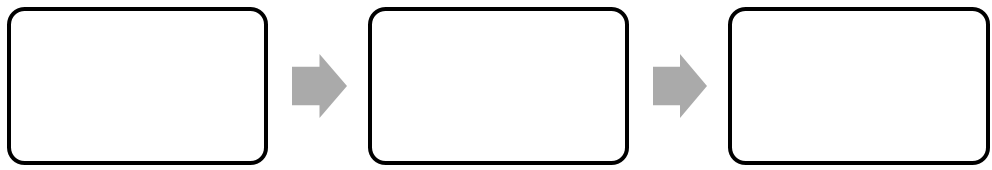
Evaluation:

| P | M | I |
| --- | --- | --- |
|  |  |  |

Design 4

| Design name | End-use application/s |
| --- | --- |
|  |  |

IPO chart of the control system:



Evaluation:

| P | M | I |
| --- | --- | --- |
|  |  |  |

Testing of ideas

Now that you have your ideas and have checked them against the design brief and criteria for success, you will need to decide on 2 of these ideas to trial in class.

These design ideas will be trialled and evaluated on:

You will need to present your completed flowchart and code in your folio (this can be a screenshot or you can copy and paste the code into a word processing software program) and complete evaluations for each idea.

Selection

Complete the PMI for the two design ideas that you have tested. Consider what worked and what didn’t work as well as reflecting on the design brief and your criteria for success.

Idea 1

| P | M | I |
| --- | --- | --- |
|  |  |  |

Idea 2

| P | M | I |
| --- | --- | --- |
|  |  |  |

Which design idea best meets the design brief and end-use application? Justify your choice below by making links to the Design Brief and ‘Criteria for Success’.

Developing your code

Draw a flowchart for your final design

Write the pseudocode for your sketch in the space provided.

| Step | Pseudocode |
| --- | --- |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |
| 7 |  |
| 8 |  |

Final design production

Once you have developed the code for your chosen control system design and tested it successfully using the ‘MAAS ThinkerShield’, you will need to construct the control system using the Arduino board and electronic components.

In order to do this, you will need to learn about basic electronics and electronic components that can be used with the Arduino board. The following pages give a brief overview of how to do this. There are plenty of online resources available if you do an internet search.

The final section of this workbook goes over producing your design solution, steps of production, evaluation and the marking criteria.

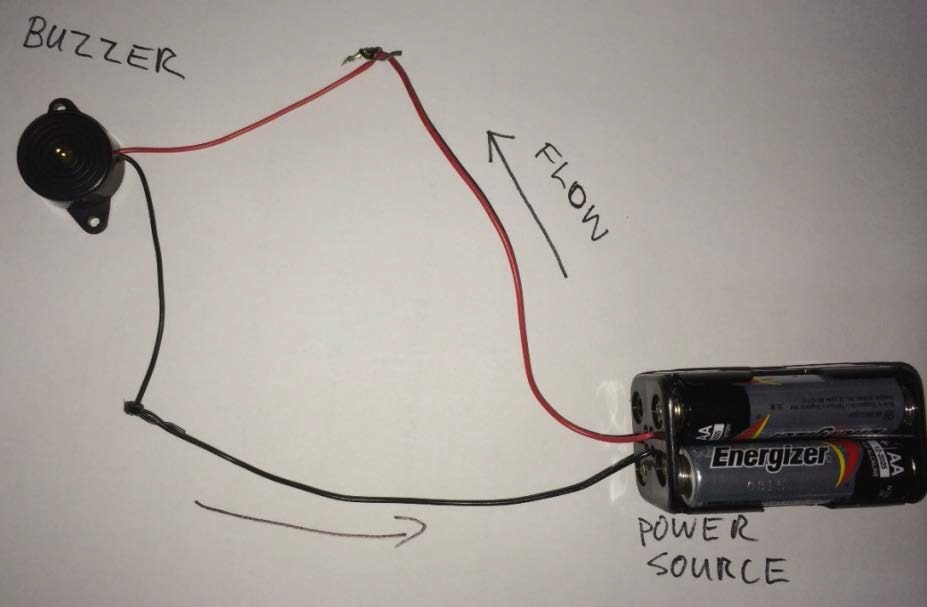
Understanding electronic circuits and Arduino

Here are some basic things you need to know about electronics when building your design solution. We haven’t explained why, just what you need to know. The why explanation would take another unit of work.

Basic electronic circuit theory

For an electronic circuit to work the electricity must be able to follow a path back to “home” (ground/earth) from where it started. Think of it like a hose connected to a tanks and pump that pumps back into itself. If the hose breaks or splits the water runs out and doesn’t refill the tank – the electrical circuit has to be “closed” so that the electrical flow can go all the way from start to finish. Below is a simple example of a continuous buzzer circuit.

The electricity can flow the whole way round the circuit from the batteries through the red wires to the buzzer and back through the black wires to the batteries again.



If the wires aren’t connected or they are broken the buzzer will not make a noise as the electricity can’t make it all the way around the circuit.

The components (electronic parts) used in Arduino projects are generally low powered (we measure power using watts in electronics). Generally most components you buy from retailers that are listed for Arduino are matched and should not be a problem.

Components of circuits / control technologies

Circuits are made up of several components which perform different functions depending on the way they are assembled. Look at the circuit components below and complete the following:

Label each component using the word bank provided and indicate whether the function is a ‘sensor’, ‘actuator’ or ‘other component’.

* Battery power supply
* Breadboard
* Copper tape
* Jumper leads
* Keypad
* Light dependent resistor (LDR)
* Light emitting diode (LED)
* Piezo buzzer
* Push button switch
* Resistor
* Toggle switch



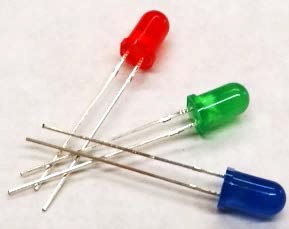
* Label: Push button switch
* Function: **Sensor**



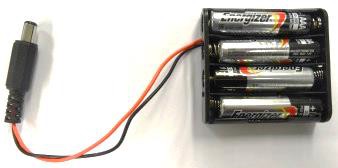
* Label: **Piezo buzzer**
* Function: Actuator



* Label: **Keypad**
* Function: **Sensor**



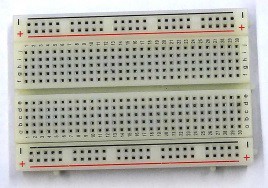
* Label: **Light Emitting Diodes LEDs**
* Function: **Actuator**



* Label: **Battery power supply**
* Function: **Other**



* Label: **Jumper Leads**
* Function: **Other**



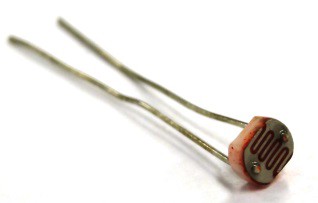
* Label: **Breadboard**
* Function:  **Other**



* Label: **Toggle Switch**
* Function: **Actuator**



* Label: **Resistor**
* Function: **Other**



* Label:   **Light Dependent Resistor LDR**
* Function: **Sensor**

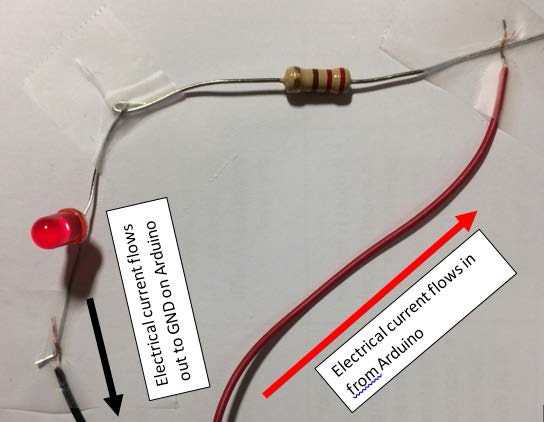


* Label: **Copper Tape**
* Function:   **Other**

Components

Light Emitting Diodes – LEDs

LEDs are small lights that must be connected to a 220Ω resistor in series (in line), otherwise the LED will eventually pop (stop working). The picture below shows you how to do this.



A circuit must be closed (complete with no breaks or missing links) for electrical current to flow from the start to the end of the circuit. If there is a break in the circuit (open circuit) the electricity can’t make it to the other end. Similar to a garden hose having a break or not being connected to the tap. In the picture above if the resistor and LED weren’t connected the LED would not light up.

The usual method of joining electrical wires for these types of projects is soldering. In the time we have this is challenging to learn and do. In the section circuit assembly are a few other ways to join wires and components together to form a closed circuit.

The LED will only work if the current is flowing from the anode (positive leg) to the cathode (negative leg). If the LED is in the wrong way round, just switch the connections around and it should start working.

Resistors

As the name implies resistors “resist” or restrict the flow of current (electricity). They come in a range of values, the ones needed in this booklet are easily obtained and you can research how to read resistor values on the Internet.

Tools, equipment and safety concerns with electronics and control technologies

Below is a series of images common tools and equipment that you will be using in this unit of work. Label each item and outline the use and safety concerns.



* Name: **Long Nose pliers**
* Use: **To bend component legs into shapes.** 
  + **To insert wires into narrow gaps**.
* Safety concerns: **Pinching of fingers**.

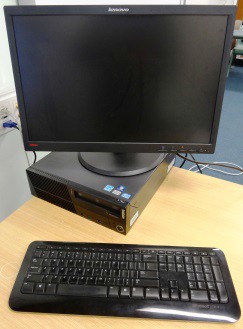


* Name: **Bull/flat nose pliers**
* Use: **To insert wires into receivers.**
  + General holding of small parts while working on them.
  + Twisting or wires / components to join.
* Safety concerns: Pinching of fingers.



* Name: **Side cutters**
* Use: **To trim wire and component legs to length before circuit assembly.**
* Safety concerns: **Pinching of fingers.**

**Protect eyes from airborne off cuts with eye protection.**



* Name: **Computer**
* Use: **To develop your code and check before uploading to Arduino.**

Safety concerns:



* Name: **USB Cable**
* Use: **To power Arduino while testing.**

**To transmit data from the computer to the Arduino.**

* Safety concerns:

Circuit assembly

Joining electrical components and wires so that electricity can flow is necessary to complete the design brief. Your code for your design solution may be correct and work on the ‘ThinkerShield’. If you haven’t made the circuit up correctly (with no breaks or short circuits) it may not work on your design solution.

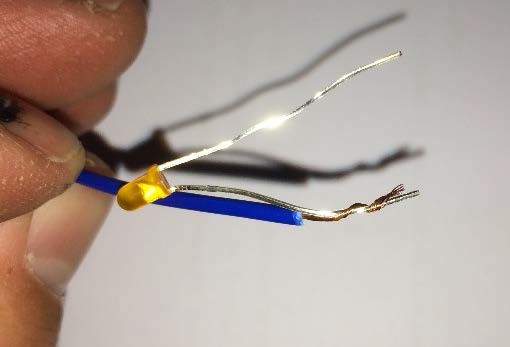
There are many ways to connect components and wires, some simple ways that you can easily use are below. Some of them are more permanent than others.

Twist and heat shrink

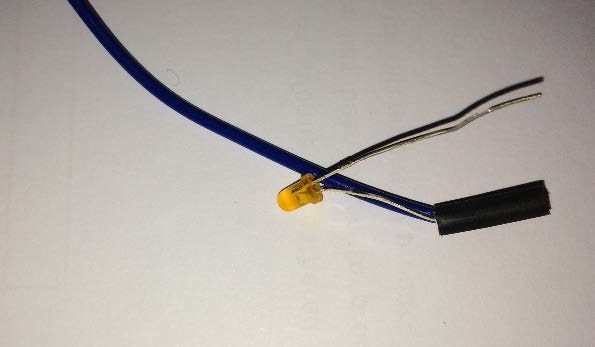
Strip ends of wires using a wire stripper or very carefully use the wire cutting part of pliers.



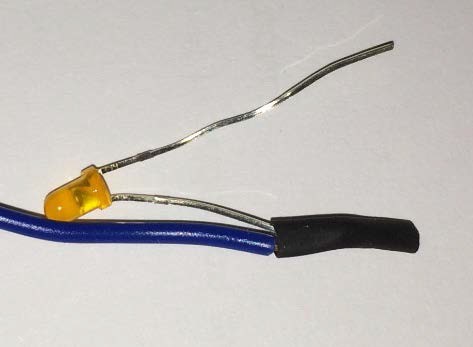
Twist wires and/or component ends together. You may need to use some bull nose pliers to do this.



Cut off a length of heat shrink and slide over your new join.



Use a heat gun or hair dryer to shrink the heat shrink which will hold the join together.

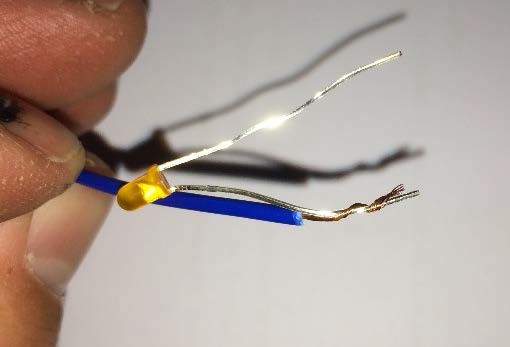


Twist and tape

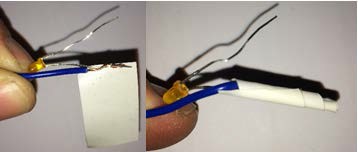
Strip ends of wires using a wire stripper or very carefully use the wire cutting part of pliers.



Twist wires and/or component ends together. You may need to use some pliers to do this.



Cut off a length of electrical tape and twist around to bind ends together.



Jumper wires

These are readily available and have the ends completed with push together terminals in a range of configurations – female to female; male to female and male to male. Depending on what components you are connecting these may be the best option. The Arduino board is best connected with a jumper wire from the output pins and back to the GND pin. After that you can easily use jumper wires, alligator clips discussed later or other joining techniques.

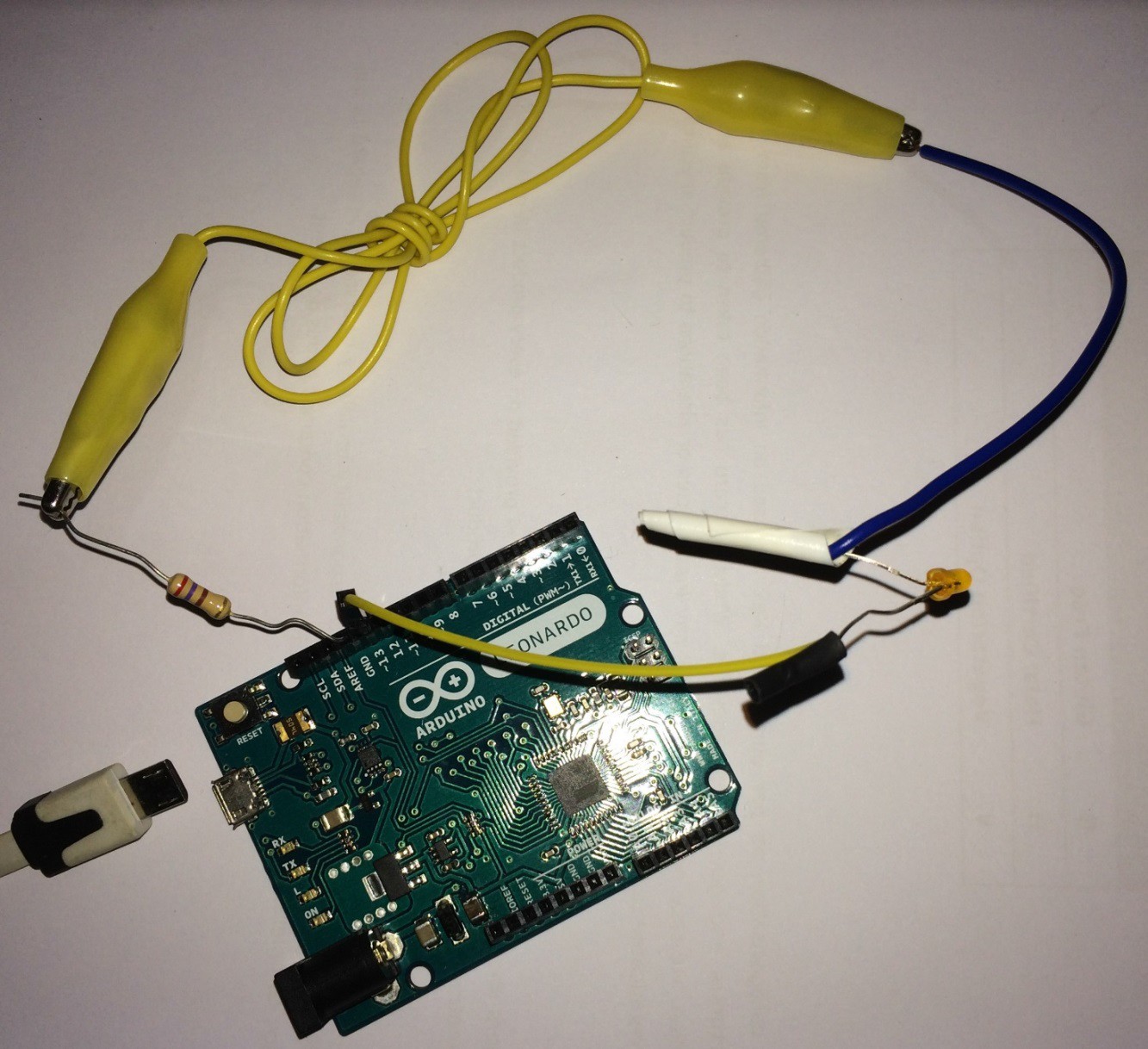
To / from Arduino board.



Connecting to a LED (female jumper lead) leg.



Alligator Clips with Other Connection Methods



Use the ready reckoner to learn how to connect each component.

Troubleshooting/Debugging Arduino

* Check that commands change colour when you type them. If a command has not changed colour it may be spelt incorrectly.
* Some commands are written in camel case. This is where the words are joined but use a capital for the second word. Examples include:
  + pinMode
  + digitalWrite.
* Make sure you use the semi colon (;) at the end of each instruction.
* Exceptions include:
* void setup () { o void loop () { o if (condition) {
* Check that commands are in the correct spot, there are lots of rules about the order things go. Remember that computers only do what they are told line by line!
* Simple things such as a USB cable not plugged in fully can stop things working properly.
* Sometimes when you check your sketch, the error is in the line before the highlighted line.
* For every opening bracket or brace { ( , there must be a closing one. ) }.Putting the cursor on a brace will highlight its partner

Electronic Connection Ready Reckoner

A quick guide to which resistors go with which components, you can find plenty more on the internet. All images created in Fritzing, available at – <http://fritzing.org/home/>

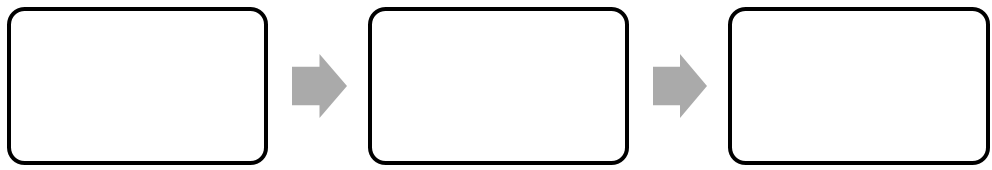
| Component | Resistor | Fritzing |
| --- | --- | --- |
| LED (Light Emitting Diode) | 220Ω   * 4 Band 220Ohm resistor - Four band colours red, red, and brown.   + Red   + Red   + Brown   + Gold * 5 band 220Ohm resistor - Five band colours red, red, black and black.   + Red   + Red   + Black   + Black   + Gold | Wiring for a LED on breadboard: An LED and 220Ohm resistor are in series between the output pin and ground. |
| Buzzer | * None | Wiring for buzzer on a breadboard: the red wire connects directlt to the output pin and the black wire connects directly to ground. |
| Potentiometer  (variable resistor) | * None | Wiring for a potentiometer on breadboard: one side of the potentiometer connects to the 5Volt supply and the opposite end connects to the ground pin. The centre connection goes to the analogue input pin. |

| Component | Resistor | Fritzing |
| --- | --- | --- |
| LDR (Light Dependant Resistor) | 10kΩ   * 4 Band 10kOhm resistor - Four band colours brown, black and orange.   + Brown   + Black   + Orange   + Gold * 5 band 10kOhm resistor - Five band colours brown, black, black and red.   + Brown   + Black   + Black   + Red   + Gold | Wiring for a LDR on breadboard: one side of the LDR connects to the 5Volt supply and the opposite end connects to both the ground pin via a 10kOhm resistor and to the analogue input pin. |
| Button | 10kΩ   * 4 band 10kOhm resistor - Four band colours brown, black and orange.   + Brown   + Black   + Orange   + Gold * 5 band 10kOhm resistor - Five band colours brown, black, black and red.   + Brown   + Black   + Black   + Red   + Gold | Wiring for button on a breadboard requires the following connections: one side of the switch connects to the 5Volt supply, the other switch leg connects to both the input pin and a 10kOhm resistor that is connected to ground. |
| Button and Resistor | 220Ω  220Ohm resistor - Four band colours red, red, and brown.  220Ohm resistor - Five band colours red, red, black and black.  10kΩ  10kOhm resistor - Four band colours brown, black and orange.  10kOhm resistor - Five band colours brown, black, black and red. | Wiring for both a pushbutton switch and a LED: The 5Volt supply is connected to one side of the switch, The other switch connection goes to both a 10kOhm resistor and the input pin. The other end of the resistor is connect ed ground pin. The ground connection also connects a 220Ohm resistor in series with an LED that is connected to the output pin. |

Producing Your Design Solution

You have decided on your final solution to the design brief and have now learnt how to construct basic circuits using an Arduino microcontroller. In order to complete your project, you will need to assemble the electronic components and the Arduino microcontroller to complete your control system. Complete the template below to assist in your planning.

Your final product needs to be housed in an appropriate environment so it can be presented to your class teacher e.g. in a box. Your teacher may also ask you to take a short video clip of your control system working in its intended environment e.g. drawer or door.



Materials List

Planning for Production and Ongoing Evaluation

Plan how you will go about producing and evaluating your control system and how well you performed each step.

Complete the action plan below to help you plan and evaluate each step of the process.

| Production step | Ongoing evaluation |
| --- | --- |
| E.g. connect input component to breadboard and Arduino | E.g. It was tricky to connect the cables into the breadboard and to the components. I need to ensure I handle them with care to avoid them breaking |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Final evaluation

Reflect on your learning with a thinking focus using Bloom’s Taxonomy. Think about the whole semester’s work when answering the questions in this table.

| Question | Answer |
| --- | --- |
| Remembering  What facts about control technology programming do you remember the most from studying Crack the Code?  What aspects of the unit were the most enjoyable and least enjoyable for you? |  |
| Understanding  What is it important to be able to program control technologies to perform set tasks? |  |
| Applying  Do you think that you will use programming the future?  How and when would you use this skill? |  |
| Analysing  What are some things you did well and what things you would do differently? |  |
| Creating  Tell us exactly how you would create the project differently in the future.  What would you do more, less and differently and why? |  |
| Evaluating  What was the most worthwhile part of ‘crack the code’ for you? |  |
| Other comments |  |

Crack the code – Marking criteria

| Focus area | Outstanding 18-20 | High 15-17 | Sound 11-14 | Basic 6-10 | Limited 0-5 |
| --- | --- | --- | --- | --- | --- |
| Project management | Time/action and finance planning is extensive.  Project is completed within time frame and financial limitations.  Flowcharts follow a clearly logical sequence that solve the brief. | Time/action and finance planning appropriate for the design project.  Flowcharts follow a clearly logical sequence with minor errors that solve the brief. | Time/action and finance planning has some appropriate aspects to the design project.  Flowcharts follow a logical sequence with some errors that attempt to solve the brief. | Time/action and/or finance planning has some relevance.  Flowcharts follow a sequence with some errors that may/may not solve the brief. | Time/action and/or finance plan has limited connection to the design project.  Flowcharts have little or no relevance to the brief. |
| Coding and function | Coding is error free and works in the design project.  Evidence of error checking and/or tinkering/adjustment present.  Design project functions as intended to fulfil the brief. | Coding is error free and works with the MAAS.  ThinkerShield.  Evidence of error checking and/or tinkering/adjustment present.  Design project partially functions as intended to fulfil the brief. | Coding appears correct, though does not perform as intended.  Some evidence of error checking and/or tinkering/ adjustment present.  Design project has little function as intended to fulfil the brief. | Coding has errors or is missing aspects to perform intended function.  Little evidence of error checking and/or tinkering/adjustment present.  Design project appears to not function as intended. | Coding is minimal and/or non-existent.  No evidence of error checking or tinkering/adjusting of the code.  Design project does not function. |
| Physical electronics | Project is housed in an aesthetically pleasing and appropriate case/enclosure.  Electronic circuit appears to be fault free and well- constructed. | Project is housed in an aesthetically pleasing and appropriate case/enclosure.  Electronic circuit appears to be fault free. | Project is housed in an aesthetically pleasing and/or appropriate case/enclosure.  Electronic circuit appears to be fault free. | Project housing is in a partially completed aesthetically pleasing and/or appropriate case/enclosure.  Electronic circuit appears to have identified faults. | Project is not housed but a circuit is present.  Circuit has faults that are not identified. |
| Final evaluation | Evaluation is detailed, objective and descriptive outlining areas of success and areas for improvement if you made the project again and why. | Evaluation is descriptive outlining areas of success and areas for improvement if you made the project again and why. | Evaluation outlines areas of success and areas for improvement if you made the project again. | Evaluation outlines some areas of success and/or areas for improvement. | Evaluation mentions some areas of success or for improvement. |

Teacher comment

Troubleshooting and debugging your code

Check that the correct board type and port are selected in the tools menu.

Go back to the pseudocode and flowchart. Are the instructions in the correct order? Computers only do what they are told line by line!

Make sure you use the semi colon (;) at the end of each line.

* Exceptions include:
  + void setup () {
  + void loop () {
  + if (condition) {

For every opening bracket or brace ( {, there must be a closing one. ) }.

* Putting the cursor on a brace will highlight its partner.

Check that commands change colour when you type them. If a command has not changed colour it may be spelt incorrectly.

Some commands are written in camel case. This is where the words are joined but use a capital for the second word. Examples include:

* pinMode
* digitalWrite.

Check simple things such whether the USB cable is fully plugged in. Sometimes the cable can be faulty. Swap with another cable to check.

Is the ThinkerShield plugged in firmly to the Arduino including all the legs in the correct positions?

Sometimes when you verify your sketch, the error is in the line before the highlighted line.

Do the pin numbers in the code match where devices are plugged in?

Sample Code

This section contains samples of code that students may come up with to achieve some of the PRP challenge exercises. Generally there is more than one solution to a coding problem. Most of these solutions are coded “the long way” to show students the step by step logical thinking nature required to write effective programs. Students who are capable may use functions mentioned in the differentiation section of the teacher information book to save space/lines in their program. You should be able to copy and paste these samples, and they should work on the Thinkershield straight away. Internet searches and tinkering/adjusting the code to work on the Thinkershield is recommended to assist students in solving the PRP challenges.

| **\_01\_Blink** |
| --- |
| void setup() { // the setup function runs once when you press reset or power the board  pinMode(12, OUTPUT); // initialise digital pin 12 as an output.  }  void loop() { // the loop function runs over and over again forever  digitalWrite(12, HIGH); // turn the LED on (HIGH is the voltage level)  delay(1000); // wait for a second  digitalWrite(12, LOW); // turn the LED off by making the voltage LOW  delay(1000); // wait for a second  } |

| **\_01\_Blink Challenge 1 – Change time on** |
| --- |
| void setup() { // the setup function runs once when you press reset or power the board  pinMode(12, OUTPUT); // initialise digital pin 12 as an output.  }  void loop() { // the loop function runs over and over again forever  digitalWrite(12, HIGH); // turn the LED on (HIGH is the voltage level)  delay(2000); // wait for 2 seconds  digitalWrite(12, LOW); // turn the LED off by making the voltage LOW  delay(1000); // wait for a second  } |

| **\_01\_Blink Challenge 2 – Change time off** |
| --- |
| void setup() { // the setup function runs once when you press reset or power the board  pinMode(12, OUTPUT); // initialise digital pin 12 as an output.  }  void loop() { // the loop function runs over and over again forever  digitalWrite(12, HIGH); // turn the LED on (HIGH is the voltage level)  delay(1000); // wait for a second  digitalWrite(12, LOW); // turn the LED off by making the voltage LOW  delay(2000); // wait for 2 seconds  } |

| **\_01\_Blink Challenge 3 – Two flashing** |
| --- |
| void setup() { // the setup function runs once when you press reset or power the board  pinMode(12, OUTPUT); // initialise digital pin 12 as an output.  pinMode(8, OUTPUT); // initialise digital pin 8 as an output.  }  void loop() { // the loop function runs over and over again forever  digitalWrite(12, HIGH); // turn the LED on (HIGH is the voltage level)  digitalWrite(8, HIGH); // turn the LED on (HIGH is the voltage level)  delay(1000); // wait for a second  digitalWrite(12, LOW); // turn the LED off by making the voltage LOW  digitalWrite(8, LOW); // turn the LED off by making the voltage LOW  delay(1000); // wait for a second  } |

| **\_01\_Blink Challenge 4 – Chasing LEDs** |
| --- |
| void setup() { // the setup function runs once when you press reset or power the board  pinMode(9, OUTPUT); // initialise digital pin 9 as an output.  pinMode(10, OUTPUT); // initialise digital pin 10 as an output.  pinMode(11, OUTPUT); // initialise digital pin 11 as an output.  pinMode(12, OUTPUT); // initialise digital pin 12 as an output.  }  void loop() { // the loop function runs over and over again forever  digitalWrite(9, HIGH); // turn the LED on (HIGH is the voltage level)  delay(500); // wait for half a second  digitalWrite(9, LOW); // turn the LED on (HIGH is the voltage level)  digitalWrite(10, HIGH); // turn the LED on (HIGH is the voltage level)  delay(500); // wait for half a second  digitalWrite(10, LOW); // turn the LED on (HIGH is the voltage level)  digitalWrite(11, HIGH); // turn the LED on (HIGH is the voltage level)  delay(500); // wait for half a second  digitalWrite(11, LOW); // turn the LED on (HIGH is the voltage level)  digitalWrite(12, HIGH); // turn the LED on (HIGH is the voltage level)  delay(500); // wait for half a second  digitalWrite(12, LOW); // turn the LED on (HIGH is the voltage level)  } |

| **\_01\_Blink Challenge 5 – Pattern LEDs** |
| --- |
| void setup() { // the setup function runs once when you press reset or power the board  pinMode(8, OUTPUT); // initialise digital pin 8 as an output.  pinMode(9, OUTPUT); // initialise digital pin 9 as an output.  pinMode(10, OUTPUT); // initialise digital pin 10 as an output.  pinMode(11, OUTPUT); // initialise digital pin 11 as an output.  pinMode(12, OUTPUT); // initialise digital pin 12 as an output.  pinMode(13, OUTPUT); // initialise digital pin 13 as an output.  }  void loop() { // the loop function runs over and over again forever  digitalWrite(8, HIGH); // turn the LED on (HIGH is the voltage level)  digitalWrite(13, HIGH); // turn the LED on (HIGH is the voltage level)  delay(500); // wait for half a second  digitalWrite(8, LOW); // turn the LED on (HIGH is the voltage level)  digitalWrite(13, LOW); // turn the LED on (HIGH is the voltage level)  digitalWrite(9, HIGH); // turn the LED on (HIGH is the voltage level)  digitalWrite(12, HIGH); // turn the LED on (HIGH is the voltage level)  delay(500); // wait for half a second  digitalWrite(9, LOW); // turn the LED on (HIGH is the voltage level)  digitalWrite(12, LOW); // turn the LED on (HIGH is the voltage level)  digitalWrite(10, HIGH); // turn the LED on (HIGH is the voltage level)  digitalWrite(11, HIGH); // turn the LED on (HIGH is the voltage level)  delay(500); // wait for half a second  digitalWrite(10, LOW); // turn the LED on (HIGH is the voltage level)  digitalWrite(11, LOW); // turn the LED on (HIGH is the voltage level)  digitalWrite(9, HIGH); // turn the LED on (HIGH is the voltage level)  digitalWrite(12, HIGH); // turn the LED on (HIGH is the voltage level)  delay(500); // wait for half a second  digitalWrite(9, LOW); // turn the LED on (HIGH is the voltage level)  digitalWrite(12, LOW); // turn the LED on (HIGH is the voltage level)  } |

| **\_02\_Button** |
| --- |
| int buttonPin = 2; // the number of the pushbutton pin  int ledPin = 12; // the number of the LED pin  int buttonState = 0; // variable for reading the pushbutton status  void setup() {  pinMode(ledPin, OUTPUT); // initialise the ledPin as an output:  pinMode(buttonPin, INPUT); // initialise the buttonPin as an input:  }  void loop() {  buttonState = digitalRead(buttonPin); // read the state of the pushbutton value:  if (buttonState == HIGH) { // check if the pushbutton is pressed.  digitalWrite(ledPin, HIGH); // turn LED on if the buttonState is HIGH:  } else {  digitalWrite(ledPin, LOW); // turn LED off if the buttonState is LOW:  }  } |

| **\_02\_Button Challenge 1 – Reverse action** |
| --- |
| int buttonPin = 2; // the number of the pushbutton pin  int ledPin = 12; // the number of the LED pin  int buttonState = 0; // variable for reading the pushbutton status  void setup() {  pinMode(ledPin, OUTPUT); // initialise the ledPin as an output:  pinMode(buttonPin, INPUT); // initialise the buttonPin as an input:  }  void loop() {  buttonState = digitalRead(buttonPin); // read the state of the pushbutton value:  if (buttonState == LOW) { // check if the pushbutton is pressed.  digitalWrite(ledPin, HIGH); // turn LED on if the buttonState is HIGH:  } else {  digitalWrite(ledPin, LOW); // turn LED off if the buttonState is LOW:  }  } |

| **\_02\_Button Challenge 2 – LED stays on for 3 seconds** |
| --- |
| int buttonPin = 2; // the number of the pushbutton pin  int ledPin = 12; // the number of the LED pin  int buttonState = 0; // variable for reading the pushbutton status  void setup() {  pinMode(ledPin, OUTPUT); // initialise the ledPin as an output:  pinMode(buttonPin, INPUT); // initialise the buttonPin as an input:  }  void loop() {  buttonState = digitalRead(buttonPin); // read the state of the pushbutton value:  if (buttonState == HIGH) { // check if the pushbutton is pressed.  digitalWrite(ledPin, HIGH); // turn LED on if the buttonState is HIGH:  delay (3000); //Wait 3 seconds  } else {  digitalWrite(ledPin, LOW); // turn LED off if the buttonState is LOW:  }  } |

| **\_02\_Button Challenge 3 – Toggle switch 1** |
| --- |
| int buttonPin = 7; // the number of the pushbutton pin  int ledState = 0;  int ledPin = 12; // the number of the LED pin  int buttonState = 0; // variable for reading the pushbutton status  void setup() {  pinMode(ledPin, OUTPUT); // initialise the ledPin as an output:  pinMode(buttonPin, INPUT); // initialise the buttonPin as an input:  }  void loop() {  digitalWrite (ledPin,ledState); //Set the LED to the value of ledState  buttonState = digitalRead(buttonPin); // read the state of the pushbutton value:  if (ledState==1) //Check if LED is on  {  if (buttonState == 1) //check if the pushbutton is pressed.  {ledState=0; //if LED is on and button is pushed, set ledState to off  delay (400); } //Wait for button to stop bouncing  }else{ //if LED is off  if (buttonState ==1) // check if the pushbutton is pressed.  {ledState =1; //if LED is on and button is pushed, set ledState to on  delay (400); } //Wait for button to stop bouncing  }  } |

| **\_02\_Button Challenge 3 – Toggle switch 2** |
| --- |
| int buttonPin = 7; // the number of the pushbutton pin  int ledState = 0;  int ledPin = 12; // the number of the LED pin  int buttonState = 0; // variable for reading the pushbutton status  void setup() {  pinMode(ledPin, OUTPUT); // initialise the ledPin as an output:  pinMode(buttonPin, INPUT); // initialise the buttonPin as an input:  }  void loop() {  digitalWrite (ledPin,ledState); //Set the LED to the value of ledState  buttonState = digitalRead(buttonPin); // read the state of the pushbutton value {  if (buttonState==1 && ledState==1) { //If button is on and led is on then ...  ledState=0; //set ledState to off  delay (300); } //Wait for button to stop bouncing  buttonState=digitalRead(buttonPin); //read button  if (buttonState==1 && ledState==0) { //If button is on and led is off then ...  ledState =1; //if LED is on and button is pushed, set ledState to on  delay (300); } //Wait for button to stop bouncing  } |

| **\_05\_analogue\_input** |
| --- |
| int sensorPin = A5; // select the input pin for the potentiometer on the ThinkerShield  int ledPin = 12; // select the pin for the LED – 12 on the ThinkerShield  int sensorValue = 0; // variable to store the value coming from the sensor  void setup() {  pinMode(ledPin, OUTPUT); // declare the ledPin as an OUTPUT  pinMode(sensorPin, INPUT); // declare the sensorPin as an INPUT  Serial.begin(9600); //tell the software to begin transmitting the value given below  }  void loop() {  sensorValue = analogRead(sensorPin); // read the value from the sensor  digitalWrite(ledPin, HIGH); // turn the ledPin on  delay(sensorValue); // stop the program for some time  digitalWrite(ledPin, LOW); // turn the ledPin off  delay(sensorValue); // stop the program for some time  Serial.print("The sensor value is: "); // in serial monitor write “The sensor value is: ”  Serial.println(sensorValue); // put the sensor value in serial monitor after the wording given in line above this.  } |

| **\_05\_analogue\_input - Challenge 1 – Multiple LEDs** |
| --- |
| int sensorPin = A5; // select the input pin for the potentiometer on the ThinkerShield  int ledPin = 12; // select the pin for the LED – 12 on the ThinkerShield  int ledPin2 = 11; // select the pin for the LED – 11 on the ThinkerShield  int sensorValue = 0; // variable to store the value coming from the sensor  void setup() {  pinMode(ledPin, OUTPUT); // declare the ledPin as an OUTPUT  pinMode(ledPin2, OUTPUT); // declare the ledPin2 as an OUTPUT  pinMode(sensorPin, INPUT); // declare the sensorPin as an INPUT  Serial.begin(9600); //tell the software to begin transmitting the value given below  }  void loop() {  sensorValue = analogRead(sensorPin); // read the value from the sensor  digitalWrite(ledPin, HIGH); // turn the ledPin on  digitalWrite(ledPin2, HIGH); // turn the ledPin2 on  delay(sensorValue); // stop the program for some time  digitalWrite(ledPin, LOW); // turn the ledPin off  digitalWrite(ledPin2, LOW); // turn the ledPin2 off  delay(sensorValue); // stop the program for some time  Serial.print("The sensor value is: "); // in serial monitor write “The sensor value is: ”  Serial.println(sensorValue); // put the sensor value in serial monitor after the wording given in line above this.  } |

| **\_05\_analogue\_input - Challenge 2 – Reverse action** |
| --- |
| int sensorPin = A5; // select the input pin for the potentiometer on the ThinkerShield  int ledPin = 12; // select the pin for the LED – 12 on the ThinkerShield  int sensorValue = 0; // variable to store the value coming from the sensor  void setup() {  pinMode(ledPin, OUTPUT); // declare the ledPin as an OUTPUT  pinMode(sensorPin, INPUT); // declare the sensorPin as an INPUT  Serial.begin(9600); //tell the software to begin transmitting the value given below  }  void loop() {  sensorValue = analogRead(sensorPin); // read the value from the sensor  digitalWrite(ledPin, HIGH); // turn the ledPin on  delay(1023 - sensorValue); // stop the program for some time  digitalWrite(ledPin, LOW); // turn the ledPin off  delay(1023 - sensorValue); // stop the program for some time  Serial.print("The sensor value is: "); // in serial monitor write “The sensor value is: ”  Serial.println(sensorValue); // put the sensor value in serial monitor after the wording given in line above this.  } |

| **\_05\_analogue\_input - Challenge 3 – Night light** |
| --- |
| int sensorPin = 4; // the number of the LDR pin  int ledPin = 12; // the number of the LED pin  int lightLevel = 0; // variable for reading the LDR  void setup() {  pinMode(ledPin, OUTPUT); // initialise the ledPin as an output:  pinMode(sensorPin, INPUT); // initialise the sensorPin as an input:  Serial.begin(9600); //setup the serial monitor  }  void loop() {  lightLevel = analogRead(sensorPin); // read the state of the LDR value:  Serial.println(lightLevel); //Send value of lightLevel to the serial monitor  if (lightLevel < 500) { // check if is dark.  digitalWrite(ledPin, HIGH); // turn LED on if dark:  } else {  digitalWrite(ledPin, LOW); // turn LED off if light:  }  } |

| **\_05\_analogue\_input Challenge 4 – Chasing LEDs 1** |
| --- |
| int sensorPin = A5; // select the input pin for the potentiometer on the ThinkerShield  int sensorValue = 0; // variable to store the value coming from the sensor  void setup() { // the setup function runs once when you press reset or power the board  pinMode(8, OUTPUT); // initialise digital pin 8 as an output.  pinMode(9, OUTPUT); // initialise digital pin 9 as an output.  pinMode(10, OUTPUT); // initialise digital pin 10 as an output.  pinMode(11, OUTPUT); // initialise digital pin 11 as an output.  pinMode(12, OUTPUT); // initialise digital pin 12 as an output.  pinMode(13, OUTPUT); // initialise digital pin 13 as an output.  pinMode(sensorPin, INPUT); // declare the sensorPin as an INPUT  }  void loop() { // the loop function runs over and over again forever  sensorValue = analogRead(sensorPin); // read the value from the sensor  digitalWrite(8, HIGH); // turn the LED on (HIGH is the voltage level)  delay(sensorValue); // wait for half a second  digitalWrite(8, LOW); // turn the LED on (HIGH is the voltage level)  digitalWrite(9, HIGH); // turn the LED on (HIGH is the voltage level)  delay(sensorValue); // wait for half a second  digitalWrite(9, LOW); // turn the LED on (HIGH is the voltage level)  digitalWrite(10, HIGH); // turn the LED on (HIGH is the voltage level)  delay(sensorValue); // wait for half a second  digitalWrite(10, LOW); // turn the LED on (HIGH is the voltage level)  digitalWrite(11, HIGH); // turn the LED on (HIGH is the voltage level)  delay(sensorValue); // wait for half a second  digitalWrite(11, LOW); // turn the LED on (HIGH is the voltage level)  digitalWrite(12, HIGH); // turn the LED on (HIGH is the voltage level)  delay(sensorValue); // wait for half a second  digitalWrite(12, LOW); // turn the LED on (HIGH is the voltage level)  digitalWrite(13, HIGH); // turn the LED on (HIGH is the voltage level)  delay(sensorValue); // wait for half a second  digitalWrite(13, LOW); // turn the LED on (HIGH is the voltage level)  } |

| **\_05\_analogue\_input Challenge 4 – Chasing LEDs 2** |
| --- |
| int sensorPin = A5; // select the input pin for the potentiometer on the ThinkerShield  int sensorValue = 0; // variable to store the value coming from the sensor  void setup() { // the setup function runs once when you press reset or power the board  pinMode(8, OUTPUT); // initialise digital pin 8 as an output.  pinMode(9, OUTPUT); // initialise digital pin 9 as an output.  pinMode(10, OUTPUT); // initialise digital pin 10 as an output.  pinMode(11, OUTPUT); // initialise digital pin 11 as an output.  pinMode(12, OUTPUT); // initialise digital pin 12 as an output.  pinMode(13, OUTPUT); // initialise digital pin 13 as an output.  pinMode(sensorPin, INPUT); // declare the sensorPin as an INPUT  }  void loop() { // the loop function runs over and over again forever  sensorValue = analogRead(sensorPin); // read the value from the sensor  digitalWrite(8, HIGH); // turn the LED on (HIGH is the voltage level)  delay(sensorValue); // wait for half a second  digitalWrite(8, LOW); // turn the LED on (HIGH is the voltage level)  digitalWrite(9, HIGH); // turn the LED on (HIGH is the voltage level)  sensorValue = analogRead(sensorPin); // read the value from the sensor  delay(sensorValue); // wait for half a second  digitalWrite(9, LOW); // turn the LED on (HIGH is the voltage level)  digitalWrite(10, HIGH); // turn the LED on (HIGH is the voltage level)  sensorValue = analogRead(sensorPin); // read the value from the sensor  delay(sensorValue); // wait for half a second  digitalWrite(10, LOW); // turn the LED on (HIGH is the voltage level)  digitalWrite(11, HIGH); // turn the LED on (HIGH is the voltage level)  sensorValue = analogRead(sensorPin); // read the value from the sensor  delay(sensorValue); // wait for half a second  digitalWrite(11, LOW); // turn the LED on (HIGH is the voltage level)  digitalWrite(12, HIGH); // turn the LED on (HIGH is the voltage level)  sensorValue = analogRead(sensorPin); // read the value from the sensor  delay(sensorValue); // wait for half a second  digitalWrite(12, LOW); // turn the LED on (HIGH is the voltage level)  digitalWrite(13, HIGH); // turn the LED on (HIGH is the voltage level)  sensorValue = analogRead(sensorPin); // read the value from the sensor  delay(sensorValue); // wait for half a second  digitalWrite(13, LOW); // turn the LED on (HIGH is the voltage level)  } |

| **\_07\_Buzzer1** |
| --- |
| int buzzerPin = 3; //Declares the value of buzzerPin to be 3  void setup(){  pinMode(buzzerPin, OUTPUT); // Initialise the buzzerPin as an output.  }  void loop(){  tone(buzzerPin, 600, 30); delay(150); // Play a tone with the buzzer. frequency 600, duration 30ms then delay 150ms.  } |

| **\_07\_Buzzer1 Challenge 1 – two tones** |
| --- |
| int buzzerPin = 3; //Declares the value of buzzerPin to be 3  void setup(){  pinMode(buzzerPin, OUTPUT); // Initialise the buzzerPin as an output.  }  void loop(){  tone(buzzerPin, 600, 500); delay(500); // Play a tone with the buzzer. frequency 600, duration 30ms then delay 150ms.  tone(buzzerPin, 300, 500); delay(500); // Play a tone with the buzzer. frequency 600, duration 30ms then delay 150ms.  } |

| **\_07\_Buzzer Challenge 2 – Button Tone** |
| --- |
| int buttonPin = 7; // the number of the pushbutton pin  int buzzerPin = 3; //Declares the value of buzzerPin to be 3  int buttonState = 0; // variable for reading the pushbutton status  void setup() {  pinMode(buzzerPin, OUTPUT); // initialise the ledPin as an output:  pinMode(buttonPin, INPUT); // initialise the buttonPin as an input:  }  void loop() {  buttonState = digitalRead(buttonPin); // read the state of the pushbutton value:  if (buttonState == HIGH) { // check if the pushbutton is pressed.  tone(buzzerPin, 600, 50); delay(50); // Play a tone with the buzzer. frequency 600, duration 50ms then delay 50ms.  } } |

| **\_07\_Buzzer Challenge 3 – Toggle Tone** |
| --- |
| int buttonPin = 7; // the number of the pushbutton pin  int buzzerState = 0;  int buzzerPin = 3; //Declares the value of buzzerPin to be 3  int buttonState = 0; // variable for reading the pushbutton status  void setup() {  pinMode(buzzerPin, OUTPUT); // initialise the ledPin as an output:  pinMode(buttonPin, INPUT); // initialise the buttonPin as an input:  }  void loop() {  if (buzzerState == 1)  {  tone(buzzerPin, 600, 50); delay(50);} // Play a tone with the buzzer. frequency 600, duration 50ms then delay 50ms.  buttonState = digitalRead(buttonPin); // read the state of the pushbutton value:  if (buzzerState==1) //Check if LED is on  {  if (buttonState == 1) //check if the pushbutton is pressed.  {buzzerState=0; //if LED is on and button is pushed, set ledState to off  delay (400); } //Wait for button to stop bouncing  }else{ //if LED is off  if (buttonState ==1) // check if the pushbutton is pressed.  {buzzerState =1; //if LED is on and button is pushed, set ledState to on  delay (400); } //Wait for button to stop bouncing  }  } |

| **\_07\_Buzzer Challenge 4 – A tune** |
| --- |
| int buzzerPin = 3;  void setup(){  pinMode(buzzerPin, OUTPUT);}  void loop(){  tone(buzzerPin, 392, 1000); delay(1000); tone(buzzerPin, 587, 1000); delay(1000);  tone(buzzerPin, 523, 250); delay(250); tone(buzzerPin, 493, 250); delay(250);  tone(buzzerPin, 440, 250);delay(250); tone(buzzerPin, 783, 1000); delay(1000);  tone(buzzerPin, 587, 500); delay(500);tone(buzzerPin, 523, 250); delay(250);  tone(buzzerPin, 493, 250); delay(250);tone(buzzerPin, 440, 250); delay(250);  tone(buzzerPin, 783, 1000); delay(1000);tone(buzzerPin, 587, 500);delay(500);  tone(buzzerPin, 523, 220); delay(250);tone(buzzerPin, 493, 250); delay(250);  tone(buzzerPin, 523, 250);delay(250);tone(buzzerPin, 440, 1000);delay(1000);  tone(buzzerPin, 293, 500); delay(500);tone(buzzerPin, 293, 250); delay(250);  tone(buzzerPin, 392, 1000); delay(1000);tone(buzzerPin, 587, 1000);delay(1000);  tone(buzzerPin, 523, 250); delay(250);tone(buzzerPin, 493, 250);delay(250);  tone(buzzerPin, 440, 250);delay(250); tone(buzzerPin, 783, 1000);delay(1000);  tone(buzzerPin, 587, 500); delay(500);tone(buzzerPin, 523, 250); delay(250);  tone(buzzerPin, 493, 250); delay(250);tone(buzzerPin, 440, 250); delay(250);  tone(buzzerPin, 783, 1000); delay(1000);tone(buzzerPin, 587, 500); delay(500);  tone(buzzerPin, 523, 250); delay(250);tone(buzzerPin, 493, 250); delay(250);  tone(buzzerPin, 523, 250); delay(250);tone(buzzerPin, 440, 1000); delay(1000);  tone(buzzerPin, 293, 500);delay(500); tone(buzzerPin, 293, 250);delay(250);  tone(buzzerPin, 329, 750); delay(750);tone(buzzerPin, 329, 250); delay(250);  tone(buzzerPin, 523, 250); delay(250);tone(buzzerPin, 493, 250); delay(250);  tone(buzzerPin, 440, 250);delay(250); tone(buzzerPin, 392, 250); delay(250);  tone(buzzerPin, 392, 250); delay(250);tone(buzzerPin, 440, 250); delay(250);  tone(buzzerPin, 493, 250); delay(250);tone(buzzerPin, 440, 500); delay(500);  tone(buzzerPin, 329, 250); delay(250);tone(buzzerPin, 370, 500); delay(500);  tone(buzzerPin, 293, 500); delay(500);tone(buzzerPin, 293, 250);delay(250);  tone(buzzerPin, 329, 750); delay(750);tone(buzzerPin, 329, 250); delay(250);  tone(buzzerPin, 523, 250); delay(250);tone(buzzerPin, 493, 250);delay(250);  tone(buzzerPin, 440, 250);delay(250); tone(buzzerPin, 392, 250);delay(250);  tone(buzzerPin, 587, 500); delay(500);tone(buzzerPin, 440, 1000);delay(1000);  tone(buzzerPin, 293, 500); delay(500);tone(buzzerPin, 293, 250); delay(1250);  } |

| **\_10\_Alarm\_Basic** |
| --- |
| int sensorPin = 7; //set the sensor to connect to pin 7  int flashPin = 13; //an LED that lights up when the alarm sounds  int buzzerPin = 3; //Set buzzer to connect to pin 3  int Status = 0; //set arm value to 0, 0=exit delay, 1=armed, 2=triggered.  int sensor = 0; //set sensor value to 0  void setup() {  // initialize digital pins as output and input.  pinMode(sensorPin, INPUT);  pinMode(flashPin, OUTPUT);  pinMode (buzzerPin, OUTPUT);  }  void loop() {  if (Status == 0) //check to see if this is the first time through the loop. arm=0  { //the sketch will do all the commands between this { and the } if arm-0  delay (3000); //this delay gives you 5 seconds to close the box  tone(buzzerPin,725,40); //this gives a short beep to indicate it is armed  delay (200); //short gap  tone(buzzerPin,725,40); //another short beep gives it a double beep sound.  Status=1; //sets the value of arm to 1 so it doesn't do this loop again  }  if (Status==1){  sensor = digitalRead(sensorPin); //this reads the sensorpin value  }  if (sensor==0) //this tests to see if the box has been opened.  {  Status=2;  }  if (Status == 2)  {  digitalWrite(flashPin,1); //turns on the led  tone(buzzerPin,725,1000);  delay (1000);  tone(buzzerPin,330,1000);  delay (1000);  }  } //end of loop so the sketch goes back to the beginning of the loop. |

| \_10\_Alarm\_Complex |
| --- |
| /\*complex alarm code includes entry and exit delays, status report to serial monitor, Status leds and control button.  Pin configuration works on ThinkerShield  Each press changes mode  set threshold ONLY in standby mode.  LED 13 lights when threshold (Pot) value is 50 plus half the LDR value.  adjust pot so SetLED is just on in the light.  SetLED should be off when in dark.  \*/  int potPin = 5; //potPin is connected to a potentiometer and adjusts the threshold for alarm trigger  int potValue = 0;  int LDRPin = 4; //LDR Pin is connected to an LDR to detect openning drawer  int LDRValue = 0;  int Status = 0; //Status of alarm 0=standby, 1=exit delay, 2=armed, 3=exit delay, 4=triggered  int buzzerPin = 3; //connect to buzzer  int buttonPin = 7; //connected to button and controls alarm status  int buttonValue = 0;  int standbyLEDPin = 8; //indicator LEDs  int exitLEDPin = 9;  int armedLEDPin = 10;  int entryLEDPin = 11;  int triggeredLEDPin = 12;  int SetLEDPin = 13; //indicates if potValue is <LDRValue in standby mode.  void setup() { // Setup inputs, outputs and serial monitor  pinMode(potPin,INPUT);  pinMode(LDRPin,INPUT);  pinMode(buttonPin,INPUT);  pinMode(buzzerPin,OUTPUT);  pinMode (standbyLEDPin,OUTPUT);  pinMode (exitLEDPin,OUTPUT);  pinMode (armedLEDPin,OUTPUT);  pinMode (entryLEDPin,OUTPUT);  pinMode (triggeredLEDPin,OUTPUT);  pinMode (SetLEDPin,OUTPUT);  Serial.begin(9600);}  void loop() {  if(Status ==0){ //Run this section if in standby mode  digitalWrite(exitLEDPin,LOW); //other LEDs off  digitalWrite(armedLEDPin,LOW);  digitalWrite(entryLEDPin,LOW);  digitalWrite(triggeredLEDPin,LOW);  digitalWrite(standbyLEDPin,HIGH); //standby LED on  potValue = analogRead(potPin); //read values of pot and LDR  LDRValue = analogRead(LDRPin);  Serial.print("STANDBY "); //Print values and status to serial monitor  Serial.print("Threshold level: ");  Serial.print(potValue);  Serial.print(" Light level: ");  Serial.println(LDRValue);  if(potValue<(50+(LDRValue/2))){ //Turn on LED13 if potvalue is half LDR value plus 50.  digitalWrite(SetLEDPin,HIGH);  }else{  digitalWrite(SetLEDPin,LOW);}  buttonValue = digitalRead(buttonPin); //Check if button is pressed to move status into exit delay mode.  if(buttonValue == 1){  Status = 1;  delay(400);}}  if(Status==1){ //Run this section if in exit mode  digitalWrite(standbyLEDPin,LOW); //other LEDs off  digitalWrite(armedLEDPin,LOW);  digitalWrite(entryLEDPin,LOW);  digitalWrite(triggeredLEDPin,LOW);  digitalWrite(SetLEDPin,LOW);  digitalWrite(exitLEDPin,HIGH); //exit LED on  Serial.println ("EXIT DELAY"); //print EXIT DELAY to serial monitor  delay(3000); //wait 3 seconds then chirp twice  tone(buzzerPin,600,20);  delay(200);  tone(buzzerPin,600,20);  Status = 2;} //change status to armed  if(Status == 2){ /Run this section if in armed mode  digitalWrite(standbyLEDPin,LOW); //other LEDs off  digitalWrite(exitLEDPin,LOW);  digitalWrite(entryLEDPin,LOW);  digitalWrite(triggeredLEDPin,LOW);  digitalWrite(SetLEDPin,LOW);  digitalWrite(armedLEDPin,HIGH); //armed LED on  Serial.print ("ARMED "); //print ARMED to serial monitor  LDRValue = analogRead(LDRPin); //Read LDR value and test if triggered  Serial.print("Threshold level: ");  Serial.print(potValue);  Serial.print(" Light level: ");  Serial.println(LDRValue);  if(potValue<LDRValue){  Status = 3; }} //change status to exit mode  if(Status == 3){ //Run this section if in entry mode  digitalWrite(standbyLEDPin,LOW); //other LEDs off  digitalWrite(exitLEDPin,LOW);  digitalWrite(armedLEDPin,LOW);  digitalWrite(triggeredLEDPin,LOW);  digitalWrite(SetLEDPin,LOW);  digitalWrite(entryLEDPin,HIGH); //entry LED on  Serial.println ("ENTRY DELAY"); //print ENTRY DELAY to serial monitor  for(int count=0;count<=300;count++){ //this loops 300 times to test for reset button press  buttonValue = digitalRead(buttonPin);  f(buttonValue == 1){  delay(200);  Status = 0;  count = 299;  }else{  delay(10);}}}  if(Status==3){ //sets status to triggered mode  Status = 4;}  if (Status==4){ //Run this section if in triggered mode  digitalWrite(standbyLEDPin,LOW); //other LEDs off  digitalWrite(exitLEDPin,LOW);  digitalWrite(armedLEDPin,LOW);  digitalWrite(entryLEDPin,LOW);  digitalWrite(SetLEDPin,LOW);  digitalWrite(triggeredLEDPin,HIGH); //triggered LED on  Serial.println ("TRIGGERED"); //print TRIGGERED to serial monitor  tone(buzzerPin,600,50); //play alarm tone  buttonValue = digitalRead(buttonPin);  if(buttonValue == 1){ //test for reset button. If pressed set atatus to standby  delay(200);  Status = 0;}}} |