LEARN

Arduino Introduction

With 8BitCADE

Compatible with:

8BitCADE XL

8BitCADE

FOUNDATION TUTORIALS

8BitCADE.com

Contents

Compiled & Written by the 8BitCADE Team
Support@8bitcade.com

Version 2

Introduction

Before we begin, I would like you to congratulate yourself! You've taken your first step into learning programming – with 8BitCADE! This tutorial will introduce you to the Arduino IDE and the basics of programming.

This booklet does not have to be read from the very beginning to the end. Once you know how to:

- install the Arduino IDE
- add Mr Blinky's Homemade Package to the Arduino IDE
- select the correct board settings
- load a sketch onto the 8BitCADE XL using the correct board settings

You can then essentially investigate each programming function one-by-one or use the booklet as a reference and go on to start programming your first game (learn section). What this booklet should do is give you a basic experience of the application of the basic programming language using your 8BitCADE XL Console.

The following **video tutorials** are also very useful in learning C/C++ language which is used for Arduino: https://www.youtube.com/playlist?list=PLPK2l9Knytg5s2dk8V09thBmNl2g5pRSr

Most of all, remember to have fun and enjoy!

8BitCADE Team

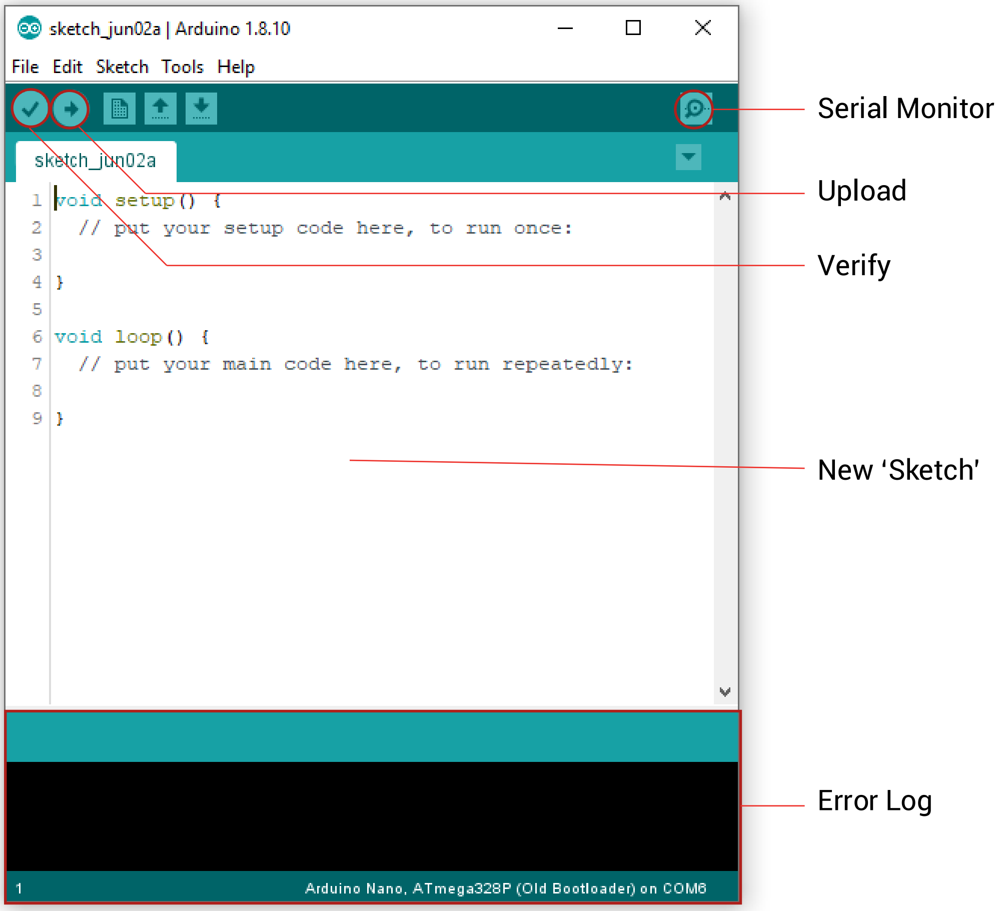
Brief Intro to the Arduino IDE

In this unit, we will be using the Arduino IDE software – a platform that allows us to program and upload our code to our consoles. You can download the software from the link below:

https://www.arduino.cc/en/main/software

Video tutorial How to install Arduino IDE: 8bitcade.com/learn/foundation

Follow the installation instructions and open Arduino – you should be met with a new blank Arduino file:


sketch_jun02a | Arduino 1.8.10
File Edit Sketch Tools Help
sketch_jun02a
Serial Monitor
Upload
Verify
New 'Sketch'
Error Log
Arduino Nano, ATmega328P (Old Bootloader) on COM6

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

}

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

}
```

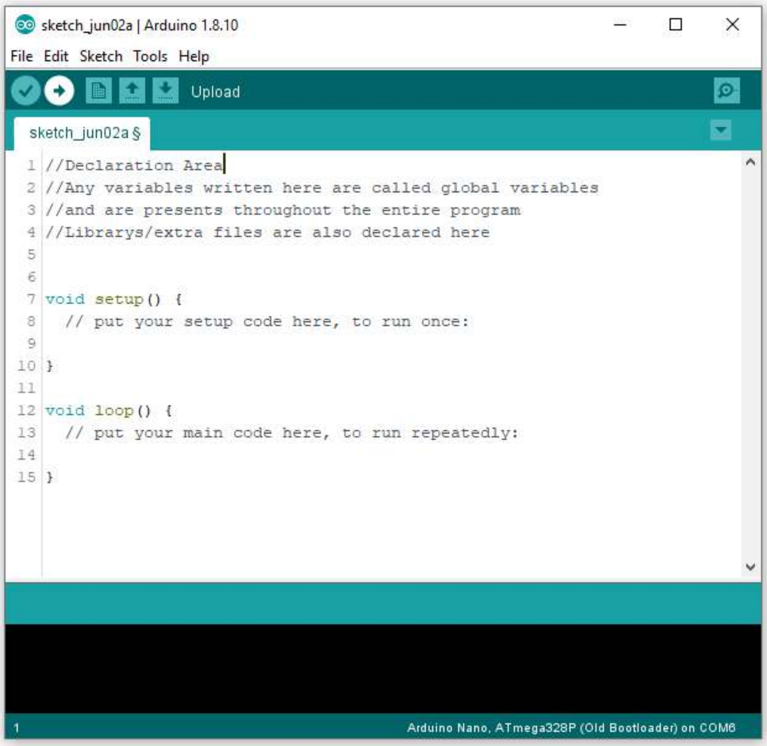
The 3 buttons we will be using the most are:

- Verify: Checks your code for errors and displays them in the error log
- Upload: Will compile your code and upload it to the connected board. Be sure to check your board settings with your console required settings (check your console Set up guide – more details in the next units)
- Serial Monitor: Displays serial data that is passed through between your computer and your Arduino board – serial transmission will be discussed in another tutorial.

Read more at https://www.arduino.cc/en/Guide/Environment

Libraries, void Setup() & void Loop()

The Arduino sketch is broken into 3 sections:


sketch_jun02a | Arduino 1.8.10

File Edit Sketch Tools Help

Upload

sketch_jun02a §

```
//Declaration Area
//Any variables written here are called global variables
//and are presents throughout the entire program
//Librarys/extra files are also declared here


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

}

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

}
```

Arduino Nano, ATmega328P (Old Bootloader) on COM6

Declaration Area: Is where you declare all of your variables and include all your files.

Void setup(): runs once when the program starts and is used to initialise variables, pin modes, libraries and more – we will go into what each of those means as we progress in this tutorial.

Void loop(): Any code here will loop continuously – this Is where your main code will go.

You cannot compile an Arduino sketch without using these two functions (void setup and loop) even if they are left blank, you must include them.

Libraries provide your sketches with extra functionality. An example of a library is "Wire.h". Think of libraries as cookbooks that we can include to utilize the different recipes that the library, or cookbook, provides. This allows us to use "functions" aka code, to improve our program – making it easier for us to write.

```
// use #include to include a library:
#include <Wire.h>
```

Creating a new sketch

When you first open Arduino, you create a new empty sketch – you can save this to any location and give it a name. If you wanted to create a new file, you would click: "**File**" > "**New**" and a new, empty sketch will appear.

Setting Up Your Console

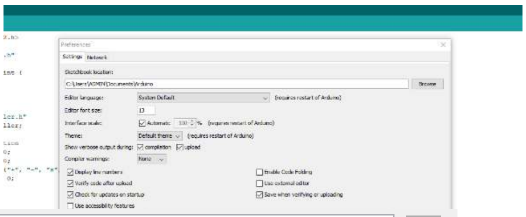
Before we get into programming, it is important that we set up our console first. This tutorial will guide you into setting up your console and give you a sketch to run to ensure it works correctly. Don't worry about what each line of code means, we will go over that in this in other tutorials. For now, set up your console. Have fun! It won't take long!

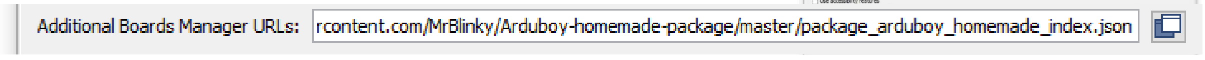
In this tutorial, we will go through the setup required to get the libraries and board settings we use for the 8BitCADE XL ready.

Board/Library Install

Before we write code, we need to include a specific library that will make programming your 8BitCADE much easier. The Arduboy2 Library. To begin this setup, we must first head on over to preferences in Arduino and add a link to allow us to access important board and library files. Firstly, click "**File**" on the top left taskbar of your screen, then click "preferences". A window like the one shown should appear. Where it says "Additional Boards Manager URLs" Click the icon

```
#include <Arduboy2.h>
Arduboy2 aboy;
```

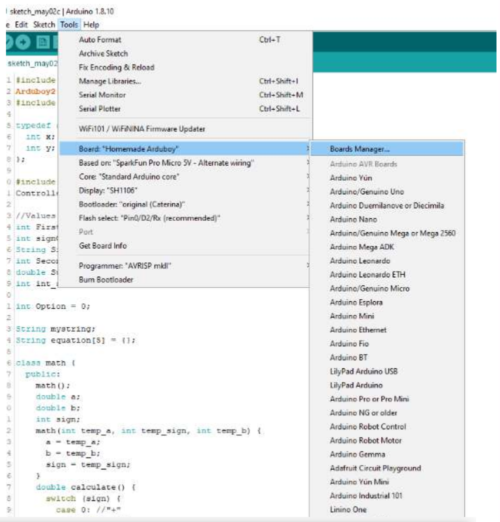

Preferences
Settings Network
Sketchbook location:
C:\Users\ADMIN\Documents\Arduino
Editor language: System Default (requires restart of Arduino)
Editor font size: 13
Interface scale: ☑ Automatic 100 % (requires restart of Arduino)
Theme: Default theme (requires restart of Arduino)
Show verbose output during: ☑ compilation ☑ upload
Compiler warnings: None
☑ Display line numbers
☑ Verify code after upload
☑ Check for updates on startup
☐ Use accessibility features
☐ Enable Code Folding
☐ Use external editor
☑ Save when verifying or uploading


Additional Boards Manager URLs: rcontent.com/MrBlinky/Arduboy-homemade-package/master/package_arduboy_homemade_index.json

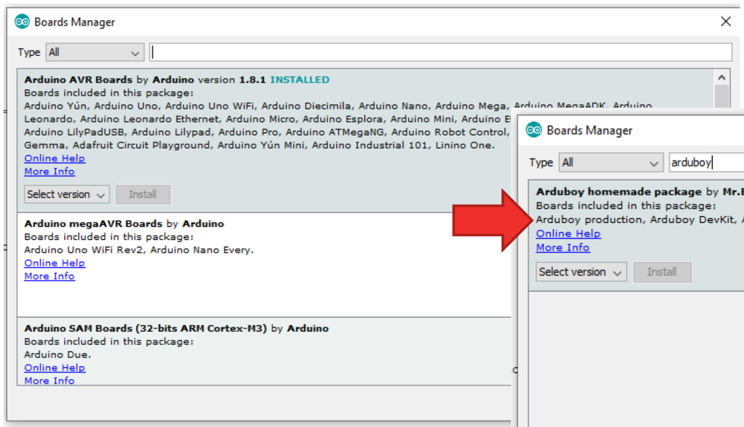
And type in, on a new line:

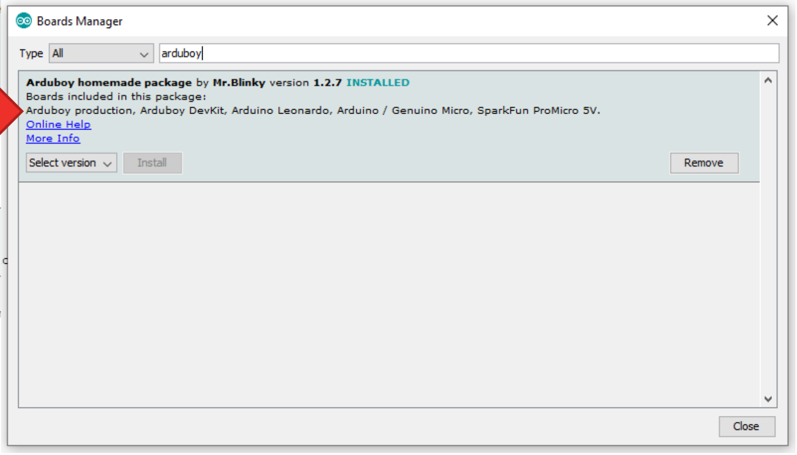
https://raw.githubusercontent.com/MrBlinky/Arduboy-homemade-package/master/package_arduboy_homemade_index.json

This will allow us to access the board and library information. Next, we need to install the board and all of its libraries. To do this, exit the current window and click "**Tools**" and select "**Board: […]**" > "**Boards Manager**"



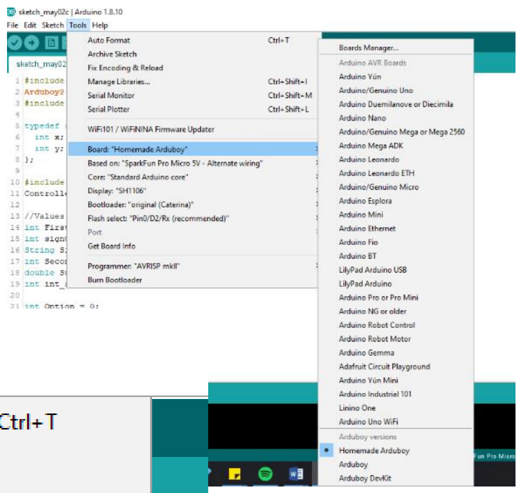
The below window should pop up (it will take some time as all your libraries are being checked/updated if need be).


Boards Manager
Type All
Arduino AVR Boards by **Arduino** version **1.8.1** INSTALLED
Boards included in this package:
Arduino Yún, Arduino Uno, Arduino Uno WiFi, Arduino Diecimila, Arduino Nano, Arduino Mega, Arduino MegaADK, Arduino Leonardo, Arduino Leonardo Ethernet, Arduino Micro, Arduino Esplora, Arduino Mini, Arduino E… Arduino LilyPadUSB, Arduino Lilypad, Arduino Pro, Arduino ATMegaNG, Arduino Robot Control, … Gemma, Adafruit Circuit Playground, Arduino Yún Mini, Arduino Industrial 101, Linino One.
Online Help
More Info
Select version Install
Arduino megaAVR Boards by **Arduino**
Boards included in this package:
Arduino Uno WiFi Rev2, Arduino Nano Every.
Online Help
More Info
Arduino SAM Boards (32-bits ARM Cortex-M3) by **Arduino**
Boards included in this package:
Arduino Due.
Online Help
More Info


Boards Manager
Type All arduboy
Arduboy homemade package by **Mr.Blinky** version **1.2.7** INSTALLED
Boards included in this package:
Arduboy production, Arduboy DevKit, Arduino Leonardo, Arduino / Genuino Micro, SparkFun ProMicro 5V.
Online Help
More Info
Select version Install Remove
Close

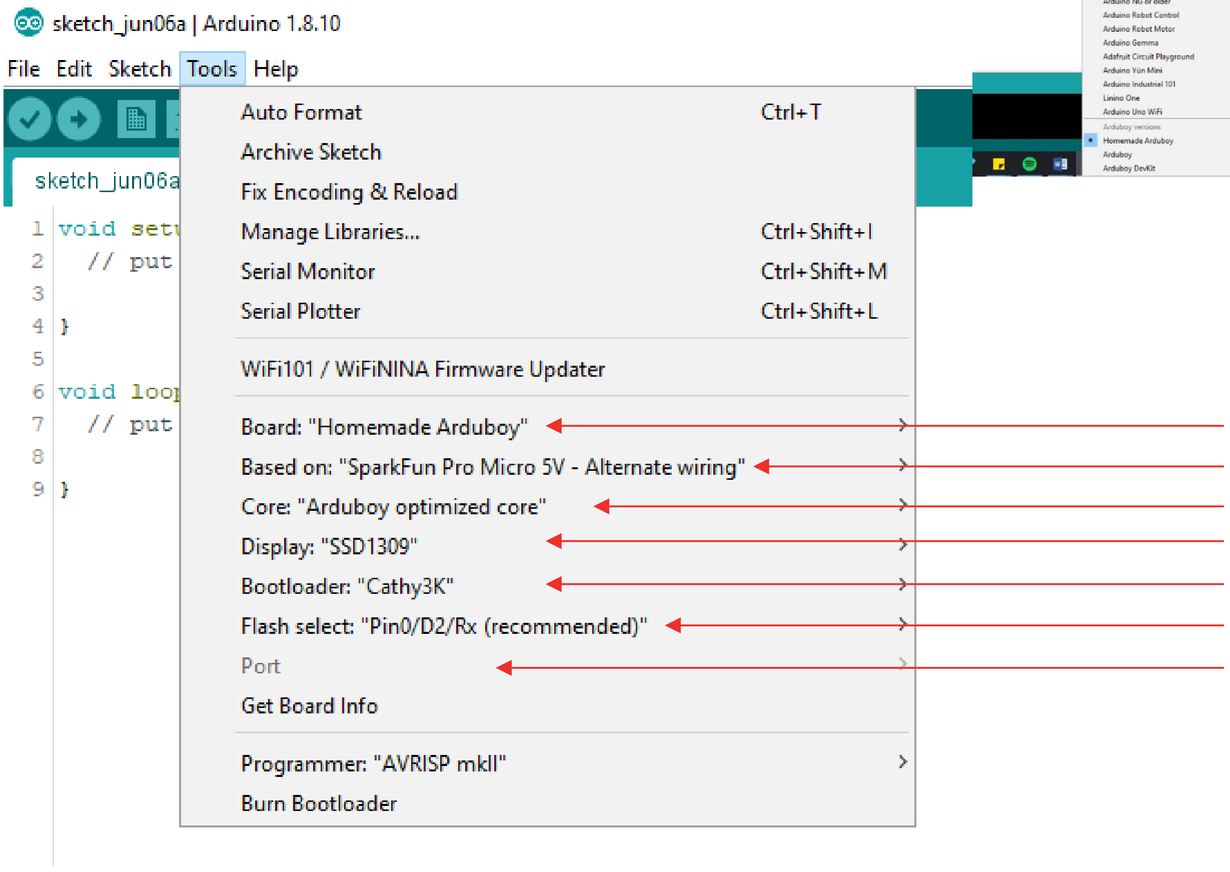
Next type in "**Arduboy**" and install the "**Arduboy Homemade Package**" By "**Mr Blinky**".

Next, we need to head on over to select the board we just downloaded. To do this head over to the toolbar and click "**Tools**" and select "**Board: […]**" > "**Home Made Arduboy**"



Board: "Homemade Arduboy"
Based on: "SparkFun Pro Micro 5V - Alternate wiring"
Core: "Standard Arduino core"
Display: "SH1106"
Bootloader: "original (Caterina)"
Flash select: "Pin0/D2/Rx (recommended)"
Programmer: "AVRISP mkII"

The next step is important, and you should double-check your settings. Check and change your board values to be exactly like the photo below:

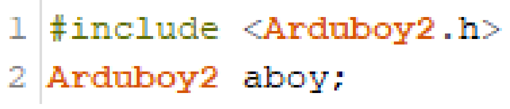

sketch_jun06a | Arduino 1.8.10
File Edit Sketch Tools Help
sketch_jun06a
void setu
// put
}
void loop
// put
}
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater
Port
Get Board Info
Burn Bootloader

Board: "Homemade Arduboy"
Based on: "SparkFun Pro Micro 5V - Alternate wiring"
Core: "Arduboy optimized core"
Display: "SSD1309"
Bootloader: "Cathy3K"
Flash select: "Pin0/D2/Rx (recommended)"
Programmer: "AVRISP mkII"

The port will be set to whatever "COM" your console is connected too.

Using the Library

Whenever you are using a library in Arduino, it's important to include it in your sketch. To include a library simply write:



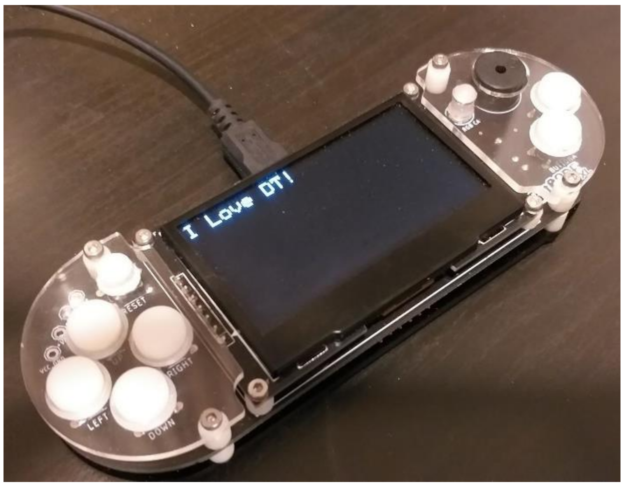
```
#include <Arduboy2.h>
Arduboy2 aboy;
```

#include <LibraryName.h>

In this case, we are including the ArduBoy2 Library, one of 6 library's that we can use. After including the Arduboy library, we can redefine its name. Instead of writing "Arduboy2.[Function]()" etc, we can type "aboy. [Function] ();"

Test Code

```
#include <Arduboy2.h>
Arduboy2 arduboy;

void setup() {
  // put your setup code here, to run once:
  arduboy.begin();
  arduboy.clear();
  arduboy.print("I Love DT!");
  arduboy.display();
}

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

}
```



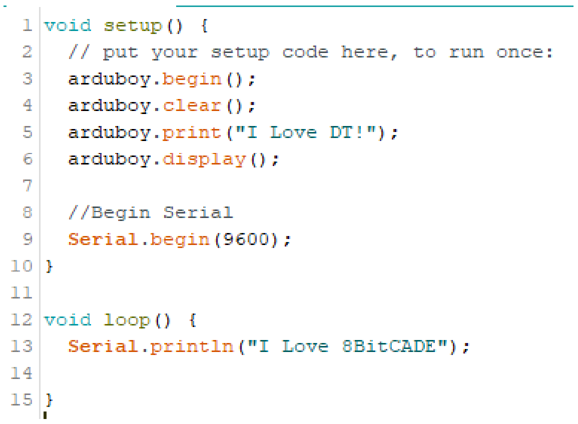
The above code should run without any errors and produce the above output. Be sure to run this to test that both your screen is working and that the library is correctly installed and that you can use Arduboy Functions correctly

Using Serial

Before we go any further – it's important to understand how to use a command called 'Serial', and many functions such as begin and print.

Serial is a way for our computer to communicate with our Arduino – all Arduino boards have at least one 'Serial Port' which consists of both an RX pin (used to receive data) and a TX pin (used to transmit data). To communicate with our Arduino, we must first use the **Serial.begin(Speed)** command – where we define the rate of bits per second at which we will transmit data, this is known as a baud rate – for those interested in more, check out this link here.

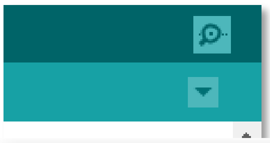
In our case, we will be using **9600**. Once the Serial has been initialized viz the being command, we can use various commands to transmit or receive data from our Arduino – in this case, we will use **Serial.print()** (or **Serial.println()** if you want to print on a new line) to display the data in our classes. Another use for serial is for debugging. Print variables to serial so you can analyse the variables and ensure they are the correct expected values.



```
void setup() {
  // put your setup code here, to run once:
  arduboy.begin();
  arduboy.clear();
  arduboy.print("I Love DT!");
  arduboy.display();

  //Begin Serial
  Serial.begin(9600);
}

void loop() {
  Serial.println("I Love 8BitCADE");

}
```

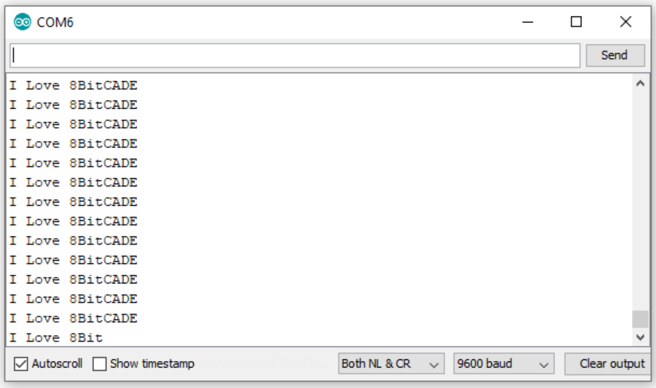
Write the code shown on the left. Here you can see how we use serial to print out a similar message to what we printed on the actual screen.

Now we printed to serial, but how do we even see this data? Where did we print it too? Well, we use something called "Serial Monitor" that can be accessed by clicking the icon shown on the right (this can only be opened if an Arduino is plugged in).



Open the serial monitor and check your monitor to mine on the next page!

Your monitor should look like this:



```
COM6
I Love 8BitCADE
I Love 8BitCADE
I Love 8BitCADE
I Love 8BitCADE
I Love 8BitCADE
I Love 8BitCADE
I Love 8BitCADE
I Love 8BitCADE
I Love 8BitCADE
I Love 8BitCADE
I Love 8BitCADE
I Love 8BitCADE
I Love 8BitCADE
I Love 8Bit
```

In the Serial Monitor, we have options such as:

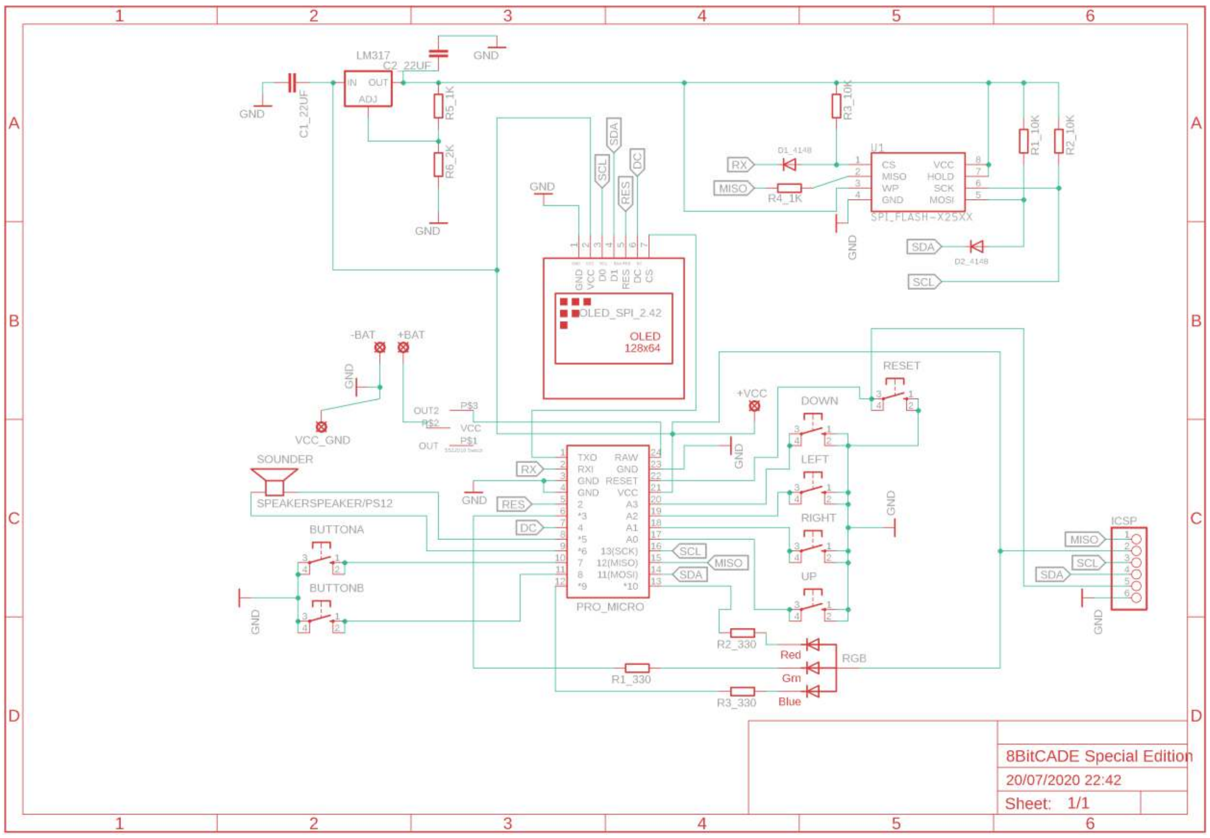
Toggle **AutoScroll**: Use this if the data being printed is being printed too fast and you need to stop and view a certain bit of data (via scrolling up and down)

Toggle **Timestamp**: Will allow you to see when something was printed to serial.

The clear output allows you to clear the current serial monitor.

Finally, we can adjust the Serial Monitor speed/baud rate– this MUST be the same value as the one specified in the Serial.begin().

8BitCADE XL Wiring Schematic for Reference



Introduction to the Arduino Language & Syntax

As previously stated, the basic structure of the Arduino programming language is fairly simple and runs in at least two parts. These two required parts, or functions, enclose blocks of statements.

```
void setup()
{
statements;
}

void loop()
{
statements;
}
```

Where setup() is the preparation, loop() is the execution. Both functions are required for the program to work.

The setup function should follow the declaration of any variables at the very beginning of the program. It is the first function to run in the program, is run only once, and is used to set pinMode or initialize serial communication.

The loop function follows next and includes the code to be executed continuously – reading inputs, triggering outputs, etc. This function is the core of all Arduino programs and does the bulk of the work.

setup()

The setup() function is called once when your program starts. Use it to initialize pin modes, libraries or begin serial. It must be included in a program even if there are no statements to run.

```
void setup()                //set-up function, only called once
{
pinMode(pin, OUTPUT);       // sets the 'pin' as  output
}
```

loop()

After calling the setup() function, the loop() function does precisely what its name suggests, and loops consecutively, allowing the program to change, respond, and control the Arduino board.

```
void loop()                 // loop function, called repeatedly.
{
```

```
digitalWrite(pin, HIGH);       // turns 'pin' on
delay(1000);                   // pauses for one  second (1000 milliseconds = 1 second)
digitalWrite(pin, LOW);        // turns 'pin' off
delay(1000);                   // pauses for one  second
}                              // these are curly braces
```

Commenting in Arduino

/… */ block comments*

Block comments, or multi-line comments, are areas of text ignored by the program and are used for large text descriptions of code or comments that help others understand parts of the program. They begin with /* and end with */ and can span multiple lines.

```
/*      this is an  enclosed  block  comment don't forget the closing comment  - they have   to
be  balanced!

*/
```

Because comments are ignored by the program and take no memory space they should be used generously and can also be used to “comment out” blocks of code for debugging purposes.

Note: While it is possible to enclose single line comments within a block comment, enclosing a second block comment is not allowed.

// line comments

Single line comments begin with // and end with the next line of code. Like block comments, they are ignored by the program and take no memory space.

```
// this is a single line comment
```

Single line comments are often used after a valid statement to provide more information about what the statement accomplishes or to provide a future reminder.

; Semicolon

A semicolon must be used to end a statement and separate elements of the program. A semicolon is also used to separate elements in a for loop.

```
int x  =  13;      // declares variable  'x' as  the integer  13
```

Note: Forgetting to end a line in a semicolon will result in a compiler error. The error text may be obvious, and refer to a missing semicolon, or it may not. If an impenetrable or seemingly illogical compiler error comes up, one of the first things to check is a missing semicolon, near the line where the compiler complained.

pinMode(pin, mode)

Used in void setup() to configure a specified pin to behave either as an INPUT or an OUTPUT.

```
pinMode(pin, OUTPUT);           // sets 'pin'  to output
```

Arduino digital pins default to inputs, so they don't need to be explicitly declared as inputs with pinMode(). Pins configured as INPUT are said to be in a high-impedance state.

There are also convenient 20KΩ pullup resistors built into the Atmega chip that can be accessed from software. These built-in pullup resistors are accessed in the following manner:

```
pinMode(pin,  INPUT_PULLUP);          // set 'pin'  to input and turn on  pullup resistors
```

Pullup resistors would normally be used for connecting inputs like switches. Notice in the above example it does not convert pin to an output, it is merely a method for activating the internal pull-ups.

Pins configured as OUTPUT are said to be in a low-impedance state and can provide 40 mA (milliamps) of current to other devices/circuits. This is enough current to brightly light up an LED (don't forget the series resistor), but not enough current to run most relays, solenoids, or motors.

Short circuits on Arduino pins and excessive current can damage or destroy the output pin, or damage the entire Atmega chip. It is often a good idea to connect an OUTPUT pin to an external device in series with a 470Ω or 1KΩ resistor.

{} Curly Braces

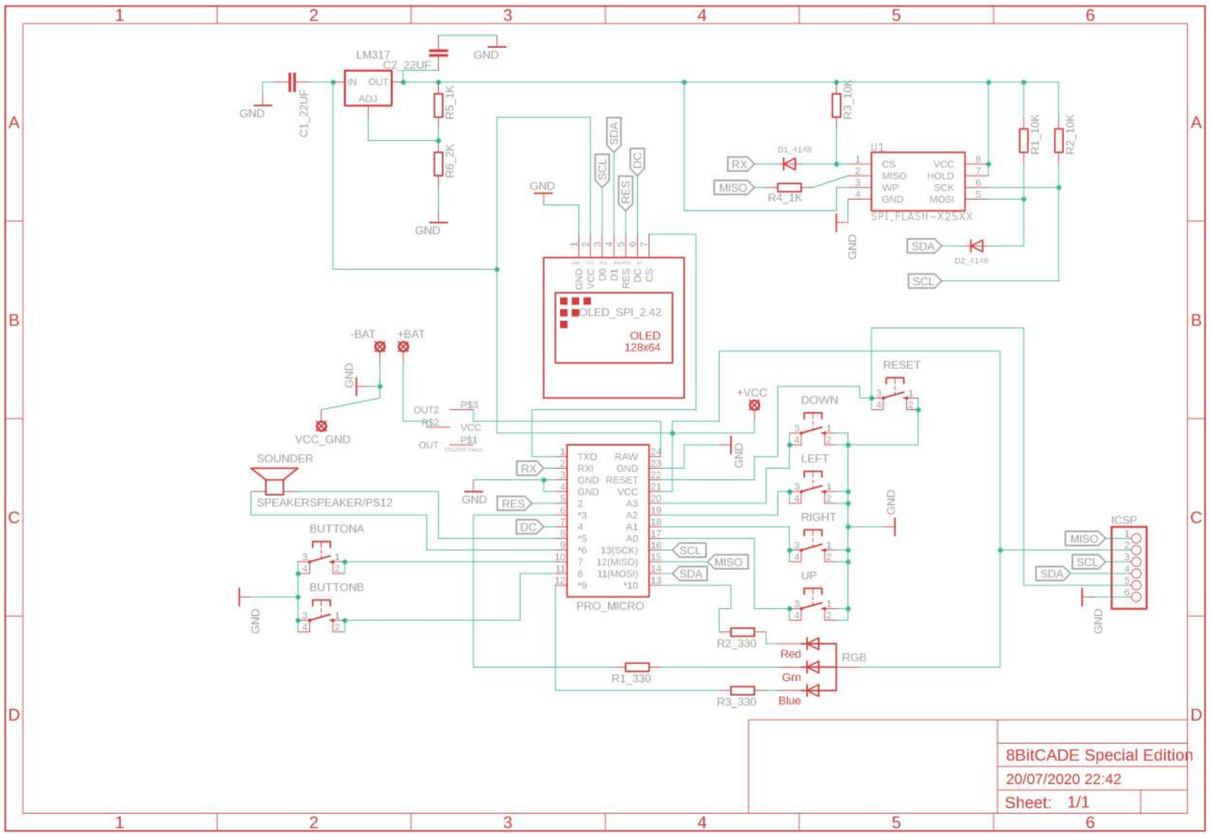
Curly braces (also referred to as just "braces" or "curly brackets") define the beginning and end of function blocks and statement blocks such as the void loop() function and the for and if statements.

```
type function()
{
statements;
}
```

An opening curly brace { must always be followed by a closing curly brace }. This is often referred to as the braces being balanced. Unbalanced braces can often lead to cryptic, impenetrable compiler errors that can sometimes be hard to track down in a large program.

The Arduino environment includes a convenient feature to check the balance of curly braces. Just select a brace, or even click the insertion point immediately following a brace, and its logical companion will be highlighted.

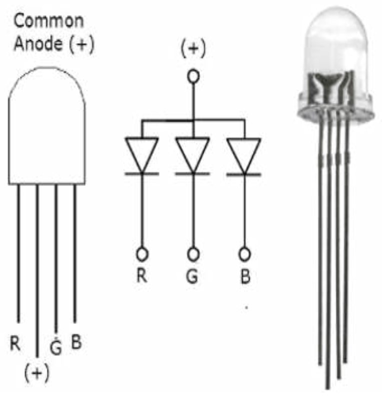
Example Sketch 3 Blink LED Circuit



RGB LED		**Arduino ProMicro Pins**	**Colour**
Pin 1	- SHORT LEG	D9 VIA 330 OHM RESISTOR	Blue
Pin 2	- SHORT LEG	D3 VIA 330 OHM RESISTOR	Green
Pin 3	+ LONG LEG	VCC pin on Pro Micro (Anode)	5V (USB) or 3.7-4.2V (lipo Batt)
Pin 4	- SHORT LEG	D10 VIA 330 OHM RESISTOR	Red

Description of the circuit.

An RGB LED is a package containing 3 separate LEDS coloured Red, Green and Blue. Your 8BitCADE XL provides voltage to your RGB LED from the VCC pin of your Pro Micro. If plugged into the USB connector and your PC, this voltage is about 5.0V. If voltage is provided by your lipo battery, then the voltage will be about 3.7-4.2 Volts. The voltage goes from VCC pin to the RGB LED + leg. When the circuit is operational (current flowing), the current goes through R,G or B LED and to 330 ohm resistors. These regulate the current flow through each individual LED to about 20mA to protect it from being destroyed. The LED cannot emit light unless a ground is provided to close the circuit and enable current to flow through it. The Pro Micro provides the ground signal internally (Atmega 32U4 chip) by switching a transistor on (electronic switch) based on instructions from your RGB LED sketch, providing a ground path, an internal connection to a ground pin (GND). This allows your RGB LED to light up!



Type in the following code into your Arduino IDE and upload it to your 8BitCADE XL and watch the RGB LED flash on and off by providing a ground path (LOW) to each leg of it. Please make sure you set the board up correctly BEFORE you upload the sketch, otherwise you can ‘Brick’ your console (stop it from taking anymore sketches without a special reset).

SKETCH CODE

```
/* Arduino IDE Console set-up 8BitCADE XL
Board: Homemade Arduboy
Based On: Sparkfun Promicro 5V Alternative Wiring
Core: Arduboy Optimised Core
Display: SSD1309
Bootloader: Cathy 3K
Flash Select:Pin0/D2/RX
Port:
*/
// blue LED is on pin 9, Red LED is on Pin 10, Green LED is on Pin 3

void setup()                    //set-up function, only called once
{
pinMode(3, OUTPUT);                     // sets the 'pin3' as  output
}

void loop()                     // loop function, called repeatedly.
{
digitalWrite(3, LOW);            // turns 'pin 3' to low, to ground, to switch on the LED
delay(1000);                    // pauses for one  second
digitalWrite(3, HIGH);          // turns 'pin 3' high, turns LED off as no ground is provided
delay(1000);                    // pauses for one  second
}
```

Key point:

In this sketch you can adjust the time delay (1000) to other values to check the effect on flashing. You can change the pin value in all the parts of the sketch from 3 to 9 or 10 to flash different colours – for red, type 10, for blue, type 9 & for green, type 3.

Example Sketch 4 Multiple Colours in a single LED

Type in the following code into your Arduino IDE and upload it to your 8BitCADE XL and watch the RGB LED flash different colours as we turn on each individual LED inside of the RGB LED (Note that this LED is made up of 3 LEDS – red, green & blue – here we are activating each individual LED. Read more in the 8BitCADE XL Make Guide Part Dictionary). Please make sure you set the board up correctly BEFORE you upload the sketch, otherwise you can 'Brick' your console (stop it from taking anymore sketches without a special reset).

SKETCH CODE

```
// blue LED is on pin 9, Red LED is on Pin 10, Green LED is on Pin
3
void setup()     //set-up function, only called once
{
pinMode(3, OUTPUT);     // sets the 'pin3' as  output
pinMode(9, OUTPUT);     // sets the 'pin3' as  output
pinMode(10, OUTPUT);     // sets the 'pin3' as  output


}


void loop()    // loop function, called repeatedly.
{
digitalWrite(3, LOW);    // turns 'pin 3' on by providing a ground
delay(1000);       // pauses for one  second
digitalWrite(3, HIGH);      // turns 'pin 3' off
delay(1000);     // pauses for one  second
digitalWrite(10, LOW);    // turns 'pin 10' on by providing a ground
delay(1000);       // pauses for one  second
digitalWrite(10, HIGH);      // turns 'pin 10' off
delay(1000);     // pauses for one  second
digitalWrite(9, LOW);     // turns 'pin 3' on by providing a ground
delay(1000);       // pauses for one  second
digitalWrite(9, HIGH);     // turns 'pin9' off
delay(1000);     // pauses for one  second
}
```

Key point:

In this sketch you can see each LED is turned on and off in a sequence. On is achieved by providing a ground path to the LED and not a voltage.

Functions

A function is a block of code that has a name and a block of statements that are executed when the function is called. The functions void setup() and void loop() have already been discussed.

Custom functions can be written to perform repetitive tasks and reduce clutter in a program. Functions are declared by first declaring the function type. This is the type of value to be returned by the function such as 'int' for an integer type function. If no value is to be returned the function type would be void. After type, declare the name given to the function and in parenthesis any parameters being passed to the function.

type functionName(parameters)

{

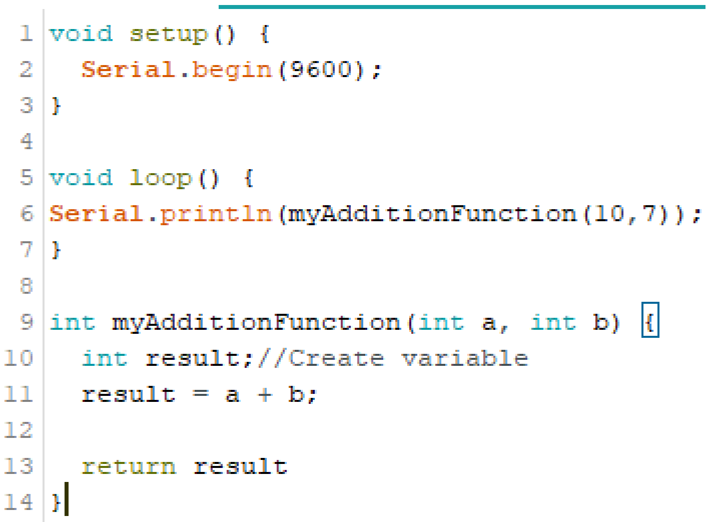
statements;

}

Example Sketch 5 My First Function

Let's create a simple program with a simple function:

SKETCH CODE

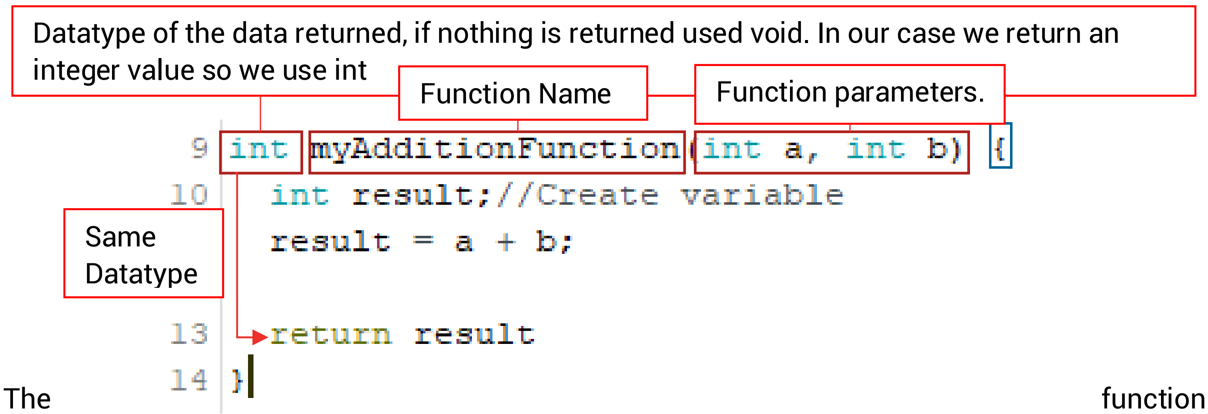


```
 1 void setup() {
 2   Serial.begin(9600);
 3 }
 4
 5 void loop() {
 6 Serial.println(myAdditionFunction(10,7));
 7 }
 8
 9 int myAdditionFunction(int a, int b) {
10   int result;//Create variable
11   result = a + b;
12
13   return result
14 }
```

Firstly, I set up serial in the void setup(). This allows us to test to see if our function works.

If we look at line 9, we can see I have defined a function called myAdditionFunction();

Functions have 3 main parts:

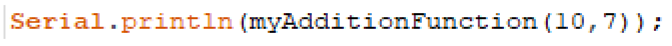


Datatype of the data returned, if nothing is returned used void. In our case we return an integer value so we use int
Function Name
Function parameters.
Same Datatype

```
 9 int myAdditionFunction(int a, int b) {
10   int result;//Create variable
     result = a + b;

13   return result
14 }
```

The function parameters are how we store data passed through when we create the function. In our case, it would be the two values we want to add together. Think of the parameters as local variables that only the function can use. In this mini piece of code, we add the parameters together, to store the result, we create a variable called result. As we create it in the function, every time we run the function, the old result will be overwritten.

We then use the return function to return the result. In this code, we can put the function inside of a Serial.println(); command, only because we have the return. Think of the return, as replacing the function with the value. So in this case, it would return the result of 10 + 7, meaning 17 would be printed to serial as seen below:



```
Serial.println(myAdditionFunction(10,7));
```

This is called “calling the function". As this function requires two parameters (a value of integer a and a value for integer b) we must place these inside the brackets.

Note that functions using void would leave the brackets empty e.g. myVoidFunction();

When we call the function myAdditionFunction, it looks in the code for a function definition with the same name. When it finds one, it places the parameters and runs the code.

Functions allow us to reuse code, makes code easier to read, reduces the length of the sketch as repeated lines of code are written once and makes the code more modular

Example Sketch 6 ‘Multiple LED's' using Function

Type in the following code into your Arduino IDE and upload it to your 8BitCADE XL and watch the LED flash the different colours.

SKETCH CODE

```
void setup() {
  pinMode(3, OUTPUT);                    // sets the 'pin3' as  output
  pinMode(10, OUTPUT);
  pinMode(9, OUTPUT);
}
void loop() {                            // loop function, called repeatedly.
  blink(3, 500); //Green // uses function name ‘blink’ with 2 parameters (pin number) and (delay)
  blink(10, 1000); //Red
  blink(9, 1500); //Blue
}
void blink(int pin, int waitTime) { //Function ‘blink’ integer ‘pin number’ and ‘waitTime’ (delay)
  digitalWrite(pin, LOW);          // pin number is taken from the integer pin in the function blink
  delay(waitTime);                 // delay is taken from the integer waitTime in the function blink
  digitalWrite(pin, HIGH);
  delay(waitTime);
}
```

Variables

A variable is a way of naming and storing a numerical value for later use by the program. As their namesake suggests, variables are numbers that can continually change as opposed to constants whose value never change. A variable needs to be declared and optionally assigned to the value needing to be stored. If the value of a variable never changes the programmer can add to the type of variable ‘const’. This will save memory in the programme.

The following code declares a variable name ‘greenLed’ and then assigns it the value 3 which is later used as pin 3. Because const is placed before the type of variable, it is expected that the value of this variable will not change throughout the program and because the variable is before the void setup() part of the program, the variable can be used by any part of the programme.

Example Sketch 7 ‘Multiple LED’s’ using Pin Variables

Type in the following code into your Arduino IDE and upload it to your 8BitCADE XL and watch the LEDS flash.

SKETCH CODE

```
const int redLED=10;    // const integer, a global variable that never changes, returns pin 10
const int greenLED=3;
const int blueLED=9;

void setup() {
  pinMode(greenLED, OUTPUT); // Pin number comes from variable greenLed which returns 10
  pinMode(redLED, OUTPUT);
  pinMode(blueLED, OUTPUT);
}

void loop() {
  blink(greenLED, 500); //Green
  blink(redLED, 1000); //Red
  blink(blueLED, 1500); //Blue
}

void blink(int pin, int waitTime) {
  digitalWrite(pin, LOW);
  delay(waitTime);
  digitalWrite(pin, HIGH);
  delay(waitTime);
}
```

Note: Variables should be given descriptive names, to make the code more readable. Variable names like tiltSensor or pushButton help the programmer and anyone else reading the code to understand what the variable represents. Variable names like var or value, on the other hand, do little to make the code readable and are only used here as examples. A variable can be named any word that is not already one of the keywords in the Arduino language.

All variables must be declared before they can be used. Declaring a variable means defining its data type, as in int, long, float, etc., setting a specified name, and optionally assigning an initial value. This only needs to be done once in a program but the value can be changed at any time using arithmetic and various assignments.

The following example declares that inputVariable is an int, or integer type, and that its initial value equals zero. This is called a simple assignment.

```
int  inputVariable =  0;  // a variable called inputVariable with the initial value of 0
                          //the data type is integer and can store data from 32,767 to -32,768.
```

Variable Scope

A variable can be declared at the beginning of the program before void setup(), locally inside of functions, and sometimes within a statement block such as for loops. Where the variable is declared determines the variable scope, or the ability of certain parts of a program to make use of the variable. A global variable is one that can be seen and used by every function and statement in a program. This variable is declared at the beginning of the program, before the setup() function. A local variable is one that is defined inside a function or as part of a for loop. It is only visible and can only be used inside the function in which it was declared. It is therefore possible to have two or more variables of the same name in different parts of the same program that contain different values. Ensuring that only one function has access to its variables simplifies the program and reduces the potential for programming errors.

The following example shows how to declare a few different types of variables and demonstrates each variable's visibility:

```
int value;        // 'value' is visible to any function
void setup()      // no  setup  needed
{
}
void loop()
{
for (int  i=0; i<20;)        // 'i' is only visible inside the for-loop i++;
{
}
float f;  // 'f' is only visible inside  loop
}
```

Introduction to Data Types

We must choose the correct data type for the type and size of data we want to store. An example is if we were to store pi, an infinite number:

(pi = 3.14159265358979323846…)

As an integer we would store: int pi = 3;

As a float we would store: float pi = 3.1415927;

As a double we would store: double pi = 3. 141592653589793;

Therefore, to create variables in our sketch, we use the below format:

DataType, VariableName = AssignedValue;

byte:

Byte stores an 8-bit numerical value without decimal points. They have a range of 0-

255.

byte someVariable = 180;   // declares 'someVariable' as a byte type

integers (int):

Integers are the primary datatype for storage of numbers without decimal points and store a 16-bit value with a range of 32,767 to -32,768.

int someVariable = 1500;   // declares 'someVariable' as an integer type

Note: Integer variables will roll over if forced past their maximum or minimum values by an assignment or comparison. For example, if x = 32767 and a subsequent statement adds 1 to x, x = x + 1 or x++, x will then rollover and equal -32,768.

long:

Extended size datatype for long integers, without decimal points, stored in a 32-bit value with a range of 2,147,483,647 to -2,147,483,648.

long someVariable = 90000; // declares 'someVariable' as a long type

float:

A datatype for floating-point numbers, or numbers that have a decimal point. Floating- point numbers have greater resolution than integers and are stored as a 32-bit value with a range of 3.4028235E+38 to -3.4028235E+38.

float someVariable = 3.14; // declares 'someVariable' as a floating-point type

Note: Floating-point numbers are not exact, and may yield strange results when compared. Floating point math is also much slower than integer math in performing calculations, so should be avoided if possible.

double:
double is similar to a float but can store more decimal places (64 bit value), therefore making it more accurate.

double num = 45.352 ;// declaration of variable with type double and initialize it with 45.352

character (Char):
A data type that takes up one byte of memory that stores a character value. Characters are written in single quotes like this: 'A' and for multiple characters, strings use double quotes: "ABC". However, characters are stored as numbers. You can see the specific encoding in the ASCII chart. This means that it is possible to do arithmetic operations on characters, in which the ASCII value of the character is used. For example, 'A' + 1 has the value 66, since the ASCII value of the capital letter A is 65.

Char chr_a = 'a' ;//declaration of variable with type char and initialize it with character a

Char chr_c = 97 ;//declaration of variable with type char and initialize it with character 97

string (string):
Is used to store anything, usually words. E.g. "you can store both characters and numbers such as 213.451" storing each letter or number in ASCII. Note while you can store a number as a string "312" you could not do maths with that number as it is a string data type. Hence using an integer.

char my_str[] = "Hello";

boolean (bool):
Is either true or false. It's similar to binary as it can be either 1 or 0.

boolean val = false ; // declaration of variable with type boolean and initialize it with false

If Statement

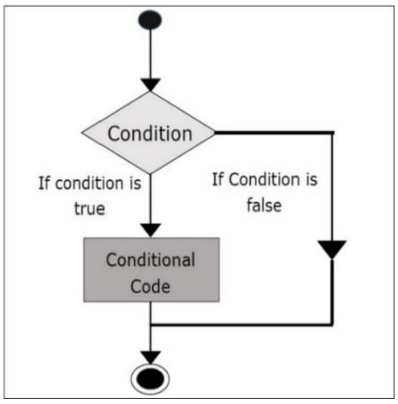
The if statement is very useful when you need to make decisions within code need to take place. It takes a condition in parenthesis and a statement or block of statements. If the condition is true then the statement or block of statements gets executed otherwise these statements are ignored.

Syntax block of statements in curly brackets

```
if (condition)
{
  block of statement(s)
}
```

Syntax single statement

```
if (condition)
  statement;
```



Example Sketch 8 Problem Solving Using If Statement

Programs use logic and algorithms to solve a problem. All programs run line by line and can be seen as recipes – with each statement being steps in creating the program. The process of writing programs can be divided into three basic sections:

- Flow Diagram: All programs can be written in a flow diagram style. It is a schematic of the program that displays the steps a program takes to achieve a goal.

In the programs we have

- Variables: Names that hold values – these can stay constant or change depending on the conditions.
- Algorithms: The recipes which take the necessary steps to achieve a goal.

An algorithm can be defined as a set of instructions or procedures, like a recipe, that will help to calculate or solve a problem.

Let's say we are coding a program that has to calculate the grade boundaries for an exam. A simple program that decides if a student has passed or if a student has failed – with 55% being the boundary. If the student has a result less then 55%, then the student has failed. If the student has a result greater than or equal to 55%, then the student has passed. Let's say we have a student called Daniel with a result of 67% - the below flow diagram shows how the basic program would work

1. Start
2. Enter students name
3. Enter students **grade**
4. **If** the **grade** is **less than 55%**, the student has **FAILED**
5. **If** the **grade** is **greater than or equal** to **55%,** the student has **PASSED**
6. End

This is known as an algorithm. We can see we also have two commands called start and stop to tell our program when to start and stop. Here each line would run line by line.

The most common statements to use for this are If statement.

REVIEW: ***If (this is TRUE) then RUN { THIS }***

If statements run certain parts of code depending on a condition, if the condition is true then the if statement will run the specified code, if it is false, it will not.

Previously we made a basic algorithm that checked if the student either passed or failed:

1. **If** the **grade** is **less than 55%**, the student has **FAILED**
2. **If** the **grade** is **greater than or equal** to **55%,** the student has **PASSED**

To write this in a code format would be:

```
If(StudentsGrade < 55){
        Result = ‘FAIL’;
}
If(StudentsGrade >= 55){
        Result = ‘PASS’;
}
```

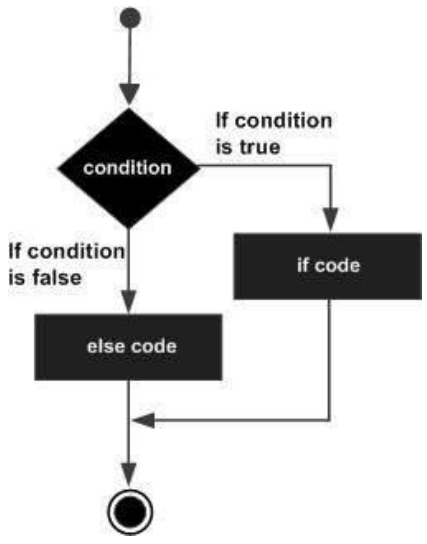
This might not make much sense at the moment, but note that the if statement follows the below format:

If (this is TRUE) then RUN { whatever code is in these brackets }

The (StudentsGrade < 55) is known as the condition of the if statement, for the if statement to run this must be true. Aka the variable StudentsGrade must be a value that is greater than 55 like 67 – if it isn't, the code does not run and misses the line “Result = ‘FAIL’;”. We use logic operations, a unit we will go into later, to create these conditions. < is a logic operation.

If else Statement

An if statement can be followed by an optional else statement, which executes when the expression is false.



Syntax

```
if (condition1) {
  // do Thing A
}
else if (condition2) {
  // do Thing B
}
else {
  // do Thing C
}
```

Conditions use simple logic (those familiar with logic gates will be familiar with this):

```
if ( x == y ) // if x is equal to y
if ( x != y ) // if x is not equal to y
if ( x < y ) // if x is smaller than y
if ( x > y ) // if x is larger than y
if ( x <= y ) // if x is smaller than or equal to y
if ( x >= y ) // if x is larger than or equal to y
```

Note that if the result of any of the logic "equations" is true, then the if statement will run -think of these like equations if the result of x == y is true, then the statement will run. Also notice how we use two equal signs for "equal to" – as writing one (x = y) would simply assign the value of y to x – be careful!

We can also utilize brackets and logical operators to expand our conditions:

```
if ( (x == y) and (x < y)) // if x == y AND x less then y
if ( (x == y) && (x < y)) // and can also be written with &&

if ( (x > 0) or (y > 0)) // if EITHER x less then 0 or y is less then 0
if ( (x >= y) || (x == y)) // Or can also be written by ||
```

AND returns true if the two inputs are both true – otherwise, it returns false:

True and true = true, anything else is false, aka **true and false = false or false and false = false**

OR returns true if one of the inputs is true – otherwise, it returns false:

false or false = false, anything else is true, aka **true or false = true** or **true or true = true**

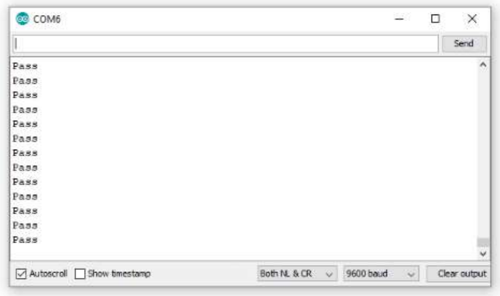
Example Sketch 9 Problem Solving Using If Else Statement

REVIEW: If (Condition) {Do Something} else {Do something else}

Let's create the code below in the Arduino IDE:

SKETCH CODE

```
int Grade = 68;
String Result;

void setup() {
  Serial.begin(9600);
}

void loop() {
  if (Grade >= 55) {
    Result = "Pass";
    Serial.println(Result);
  }
  else{
        Result = "Fail";
    Serial.println(Result);
  }
}
```



```
COM6
Pass
Pass
Pass
Pass
Pass
Pass
Pass
Pass
Pass
Pass
Pass
Pass
Pass
```

Firstly, we define the variables at the start of the code (before the void setup. For their grade score, we don't need to store any decimals, therefore, we use the (int) for integer, data type. To store the result, we want to store the words either Pass or Fail, therefore we use a string data type. Note two things: to declare a string you need a capital S & you don't need to give the variable a value, simply writing String Result; will create an empty variable that we can then assign a value to later.

We then begin serial in the void setup by typing Serial.begin(9600);

If statements use a condition, in this case, the condition is the result of Grade >= 55, aka comparing the value of the variable Grade with the value of 55, if it is greater than or equal to 55, the Result will be assigned the value "pass" – we then print the Result variable so we can see the change. Experiment by changing the Grade value and opening the serial monitor.

Conditions work on true or false bases – if the above condition is TRUE then the if statement will run (if you were to write if(true){} you will see how the if statement will always run. If you were to write if(false){} the if statement will never run).

What would the value result end up being?

If you said Pass, then you are correct! Because **68 IS** greater than **55** and that condition, therefore, *equals* **true** – therefore the variable Result gets overwritten with the value "pass".

For Statement

The for statement is used to repeat a block of statements enclosed in curly braces a specified number of times. An **increment counter** is often used to increment and terminate the loop.

There are three parts, separated by semicolons (;), to the for loop header:

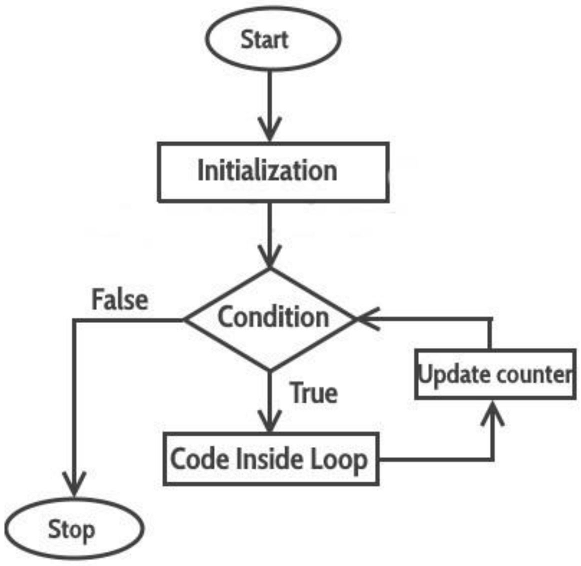
for (initialization; condition; expression)

{

doSomething;

}

The initialization of a local variable, or increment counter, happens first and only once. Each time through the loop, the following condition is tested. If the condition remains true, the following statements and expression are executed and the condition is tested again. When the condition becomes false, the loop ends.



Example Sketch 10 For Statement Loop

The following example starts the integer i at 0, tests to see if i is still less than 20 and if true, increments i by 1 and executes the enclosed statements:

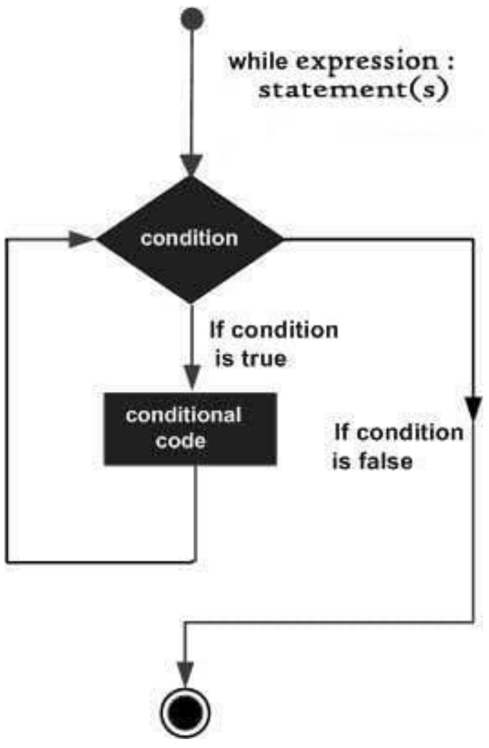
SKETCH CODE

```
const int redLED = 10; // const integer, a global variable that never changes, returns pin 10

void setup() { // put your setup code here, to run once:
pinMode(redLED, OUTPUT); // Pin number comes from variable redLED which returns 10
}

void loop() {

 for (int  i = 0; i < 20; i++) //  declares i, tests if less than 20, increments i by  1
 {
   digitalWrite(redLED, LOW); // turns pin 10  on
   delay(250);  // pauses for 1/4  second
   digitalWrite(redLED, HIGH);   // turns pin 10  off
   delay(250);
 }
}
```

Note: The C for loop is much more flexible than for loops found in some other computer languages, including BASIC. Any or all of the three header elements may be omitted, although the semicolons are required. Also the statements for initialization, condition, and expression can be any valid C statements with unrelated variables. These types of unusual for statements may provide solutions to some rare programming problems.

While Loop

while loops will loop continuously, and infinitely, until the expression inside the parenthesis becomes false. Something must change the tested variable, or the while loop will never exit. This could be in your code, such as an incremented variable, or an external condition, such as testing a sensor.



```
while  (someVariable ??  value)
{
doSomething;
}
```

The following example tests whether ‘someVariable' is less than 200 and if true executes the statements inside the brackets and will continue looping until ‘someVariable' is no longer less than 200.

```
while  (someVariable <  200) // tests if less  than 200
{
doSomething;  // executes enclosed statements
someVariable++;// increments variable  by  1
}
```

Example Sketch 11 While loop Traffic Lights

REVIEW: while(condition is true){run this, once ran, check condition is true if it is loop}

While statements loop through a certain part of code *while the condition is true.* It's important to note that, when the code is run, the code checks the condition again (hence being called a loop). An example would be traffic lights:

1. **While** the lights are **green**, let traffic **pass**
2. **While** the lights are **red**, **stop** traffic

To rewrite this in a code format would be:

```
while(lightState == Green){
        Traffic = 'PASS;
}
while(lightState == Red){
        Traffic = 'STOP;
}
```

This might not make much sense at the moment, but note that the statement follows the below format:

while (this is TRUE) then RUN { THIS } once run, check the condition again

Here we have another basic logic operation '==' which means equal to. So to translate, allow traffic to pass when the light state is equal to green. When this runs, before moving on to the next block of code it will recheck the condition – if the condition is true then it will loop through again. It will keep looping until the condition is false.

for (Current State, Condition, New State) RUN { THIS }

For statements loop through a certain part of code *for a specific amount of iterations/loop cycles.* An example would be flashing a LED (or light):

1. **When x < 5**
2. **Turn On light**
3. **Wait**
4. **Turn Off light**
5. **Wait**
6. **Increase X value by 1 (loop to 1)**

```
for(x=0; x < 5, x = x + 1){
        Turn Light On
        Wait
        Turn light off
        wait
}
```

To rewrite this in a code format would be:

This might not make much sense at the moment, but note that the if statement follows the below format:

for (Current State, Condition, New State) RUN { THIS }

Notice how in the loop code we don't increase the X value by one – this is as the for loop does it automatically via the 'new state' section of the for loop.

<u>**While VS For:**</u> A simple differentiator is: use WHILE loops when you don't know the number of loop iterations (aka let traffic through until a condition is met), while we use for loops for when we know the definite amount of loop iterations, e.g. flash the light FIVE times.

Let's look at the traffic light example:

SKETCH CODE

```
String TrafficLightColour = "Green";
bool LetTrafficPass;

void setup() {
  Serial.begin(9600);
}

void loop() {
  while (TrafficLightColour == "Green") {
    LetTrafficPass = true;
    Serial.print("Traffic is passing Through");
  }
  Serial.print("Traffic is NOT passing Through");
}
```

Firstly, we must define the variables we use:

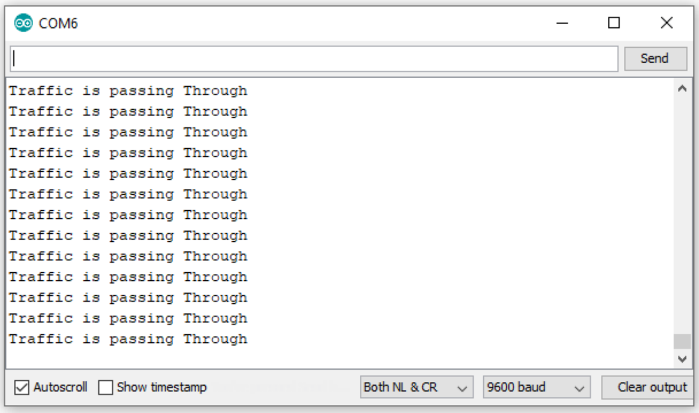
We create a string for the Traffic Light Colour, assigning it to be red.

We create a Boolean variable, either true or false, to control if traffic should pass or not.

We then begin serial in the void setup by typing Serial.begin(9600);

We then create a while loop, if the traffic light colour is green, then assign LetTrafficPass as true, and print too serial that traffic is currently passing.

Experiment by changing the Traffic light colour too red and see what happens in serial.

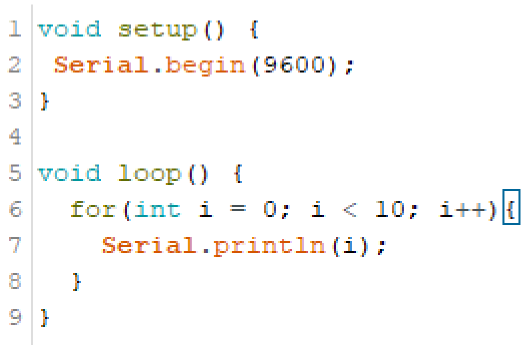


```
Traffic is passing Through
Traffic is passing Through
Traffic is passing Through
Traffic is passing Through
Traffic is passing Through
Traffic is passing Through
Traffic is passing Through
Traffic is passing Through
Traffic is passing Through
Traffic is passing Through
Traffic is passing Through
Traffic is passing Through
Traffic is passing Through
```

Let's create a simple program to count to 10

SKETCH CODE

- ***for (Current State, Condition, New State) { RUN THIS }***



```
void setup() {
 Serial.begin(9600);
}

void loop() {
  for(int i = 0; i < 10; i++){
    Serial.println(i);
  }
}
```

We then begin serial in the void setup by typing Serial.begin(9600);

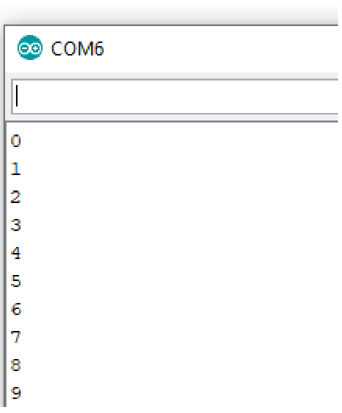
We then create a for loop with the 3 attributes (current State, condition, new state)

Int i = 0; is the start state, notice how we don't have to create this variable beforehand (define it before the void setup). This demonstrates, how we can create variables anywhere in the sketch, but if we want them to be 'global' (aka accessed anywhere in the program) then we need to define them before void setup().

i < 10; Is the condition, the for loop will run WHILE i is less than 10.

i++ is the same as writing i = i + 1 , meaning each iteration of the loop will increase the i value by one.

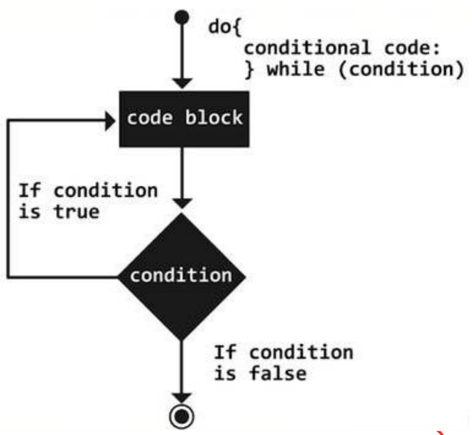
It will keep iterating until the condition is false. We then Serial print the i value so we can see this increase. Note that the serial result goes from 0 to 9 as we started on i = 0 and the for loop runs 10 times.



```
0
1
2
3
4
5
6
7
8
9
```

do… while

The do while loop is a bottom driven loop that works in the same manner as the while loop, with the exception that the condition is tested at the end of the loop, so the do loop will always run at least once.



do

{

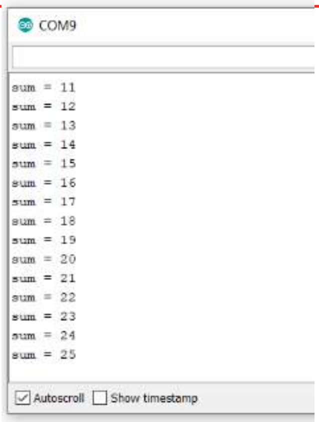
doSomething;

} while (someVariable ?? value);

SKETCH CODE

```
void setup() {
  int sum = 0;
 Serial.begin(9600);

  do {
    sum = sum + 1;
    Serial.print("sum = ");
    Serial.println(sum);
    delay(500);             // 500ms delay
  } while (sum < 25);
}
void loop() {
}
```

The sketch takes variable sum, adds 1 in a loop, prints the result in serial, pauses for 0.5 seconds and then continues to loop until the sum reaches 25.



```
sum = 11
sum = 12
sum = 13
sum = 14
sum = 15
sum = 16
sum = 17
sum = 18
sum = 19
sum = 20
sum = 21
sum = 22
sum = 23
sum = 24
sum = 25
```

digitalRead(pin)

Reads the value from a specified digital pin with the result either HIGH or LOW. The pin can be specified as either a variable or constant (0-13).

value = digitalRead(Pin);   // sets 'value' equal to the input pin

digitalWrite(pin, value)

Outputs either logic level HIGH or LOW at (turns on or off) a specified digital pin. The pin can be specified as either a variable or constant (0-13).

digitalWrite(pin, HIGH);// sets 'pin' to high

Example Sketch 12 digitalRead(pin) & digitalWrite(pin)

The following example reads a pushbutton connected to a digital input and turns on an LED connected to a digital output when the button has been pressed:

SKETCH CODE

```
/*
 Pushbutton sketch to test A0, A1, A2, A3, D7, D8
*/
const int Red_LED = 10;  // The red LED has a defined Arduino pin numbered 10
const int Green_LED = 3;
const int Blue_LED = 9;
const int inputUp_1 = A0; // Up button
const int inputRight_2 = A1; // Right button
const int inputLeft_3 = A2; // Left button
const int inputDown_4 = A3; //Down button
const int inputButA_5 = 7; // Button A
const int inputButB_6 = 8; // Button B

void setup() {
 pinMode(Red_LED, OUTPUT); // choose the pin for the LED
 pinMode(Green_LED, OUTPUT); // choose the pin for the LED
 pinMode(Blue_LED, OUTPUT); // choose the pin for the LED
 pinMode(inputUp_1, INPUT_PULLUP);
 pinMode(inputRight_2, INPUT_PULLUP);
 pinMode(inputLeft_3, INPUT_PULLUP);
 pinMode(inputDown_4, INPUT_PULLUP);
 pinMode(inputButA_5, INPUT_PULLUP);
 pinMode(inputButB_6, INPUT_PULLUP);

}
void loop() {
 int val1 = digitalRead(inputUp_1); // read input value
 if (val1 == LOW)
 {
  digitalWrite(Red_LED, LOW);
 } else
 {
  digitalWrite(Red_LED, HIGH);
 }

 int val2 = digitalRead(inputRight_2); // read input value
 if (val2 == LOW)
 {
  digitalWrite(Red_LED, LOW);
 } else
 {
  digitalWrite(Red_LED, HIGH);
 }

 int val3 = digitalRead(inputLeft_3); // read input value
 if (val3 == LOW)
 {
  digitalWrite(Red_LED, LOW);
 } else
 {
  digitalWrite(Red_LED, HIGH);
 }
```

```
  int val4 = digitalRead(inputDown_4); // read input value
  if (val4 == LOW)
  {
    digitalWrite(Red_LED, LOW);
  } else
  {
    digitalWrite(Red_LED, HIGH);
  }
  int val5 = digitalRead(inputButA_5); // read input value
  if (val5 == LOW)
  {
    digitalWrite(Green_LED, LOW);
  } else
  {
    digitalWrite(Green_LED, HIGH);
  }
  int val6 = digitalRead(inputButB_6); // read input value
  if (val6 == LOW)
  {
    digitalWrite(Blue_LED, LOW);
  } else
  {
    digitalWrite(Blue_LED, HIGH);
  }
}
```

analogRead(pin)

Reads the value from a specified analog pin with a 10-bit resolution. This function only works on the analog in pins. The resulting integer values range from 0 to 1023.

```
value =  analogRead(pin);        // sets 'value'  equal to 'pin'
```

Note: Analog pins unlike digital ones, do not need to be first declared as INPUT or OUTPUT.

analogWrite(pin, value)

Writes a analog value using hardware enabled pulse width modulation (PWM) to an output pin marked PWM. The value can be specified as a variable or constant with a value from 0-255.

```
analogWrite(pin, value);         // writes 'value'  to analog 'pin'
```

Depending on how your LED is wired up:

VCC switched: a value of 0 generates a steady 0 volts output at the specified pin, so the LED is off; a value of 255 generates a steady 5 volts output at the specified pin so the LED is fully on.

GND switched (same as 8BitCADE XL): a value of 0 generates a steady ground signal so the LED is fully on; a value of 255 generates no ground, so the LED is fully off.

For values in between **0 and 255**, the pin rapidly alternates between 0 and 5 volts – for VCC switched, the higher the value, the more often the pin is HIGH (5 volts). For example, a value of 64 will be 0 volts threequarters of the time, and 5 volts one quarter of the time; a value of 128 will be at 0 half the time and 255 half the time; and a value of 192 will be 0 volts one quarter of the time and 5 volts three-quarters of the time. Because this is a hardware function, the pin will generate a steady wave form after a call to analogWrite in the background until the next call to analogWrite (or a call to digitalRead or digitalWrite on the same pin). Analog pins unlike digital ones, do not need to be first declared as INPUT or OUTPUT.

Example Sketch 13 analogWrite Using PWM

The following example uses 'Ground Switched' LEDS so 1 = bright and 254 very dim!

analogWrite(led,0); // Fully on
analogWrite(led,255); // Fully off
analogWrite(led,1); // Very bright
analogWrite(led,254); // Very dim
analogWrite(led,128); // half brightness (half voltage 2.5V)

SKETCH CODE Manual Brightness

```
int led =  9;                          //  pin  9 blue led

void setup(){
  pinMode(led, OUTPUT);
}
 void loop(){
analogWrite(led,1);                    // 1(Bright) - 254 (Dim)
```

SKETCH CODE Auto Brightness

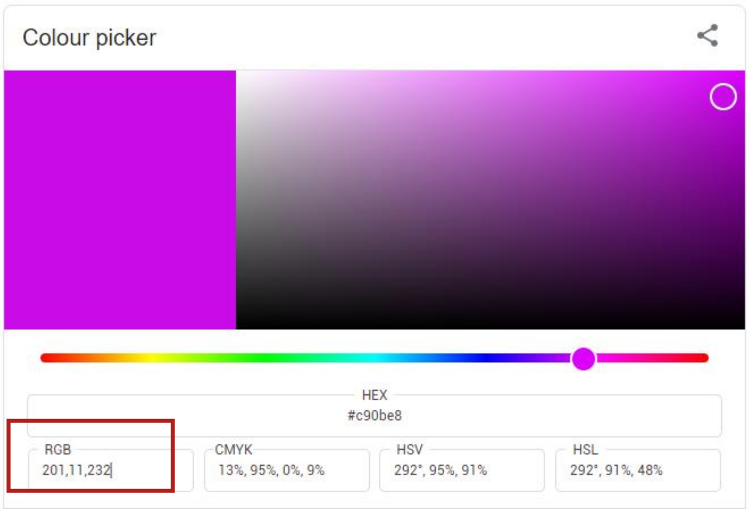
```
/*
 Fade
*/

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 time through the 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
}
```

Example Sketch 14 Creating Different Colours

The following example uses 'Ground Switched' LEDS so 1 = bright and 254 very dim!

We can further use this in combination with the 2 other built in LEDS in the RGB LED to create different colours – by carrying there brightness and turning different LEDS on at the same time. Let's take a look at an example:

NOTE: You can display any RGB colour on your LED, simply type in colour picker on google and the below 'app' will show up. Select the colour you wish to use and take the values of the RGB section (see RED box).



However, if you want to use this, you need to adjust the RGB values. We need to get the values and minus them by 255 as our values are inverted (where 255 = off and 0 = On) if we were to use this value directly, we would get a different colour.

For pink, the new value would be:

255-201 = 54, so Red = 54

255-11 = 244, so Green = 244

255-232 = 23, so Blue = 23

If you take a look at the code below, and check out the //pink line of code, youll see I have placed our calculated values inside. Upload the code and take a look! Be sure to search up your own colour and try it!

```
//Create Variables for the pins
const int Red_LED = 10;
const int Green_LED = 3;
const int Blue_LED = 9;

void setup() {
  //Set pins to OUTPUT to send data out
  pinMode(Red_LED, OUTPUT);
  pinMode(Green_LED, OUTPUT);
  pinMode(Blue_LED, OUTPUT);
}
void loop() {
  //Use our function to easily set each LED pins value
  RGB_LED(0, 255, 255); // Red
  delay(1000);
  RGB_LED(255, 0, 255); // Green
  delay(1000);
  RGB_LED(255, 255, 0); // Blue
  delay(1000);
  RGB_LED(54, 244, 23); // Pink
  delay(1000);
  RGB_LED(255, 0, 0); // Cyan
  delay(1000);
  RGB_LED(0, 255, 0); // Magenta
  delay(1000);
  RGB_LED(0, 0, 255); // Yellow
  delay(1000);
  RGB_LED(0, 0, 0); // White
  delay(1000);
}
//This function allows us to easily set the 3 LEDs inside of the RGB led values
//allowing us to get a variety of different colours
void RGB_LED(int Red_Val, int Green_Val, int Blue_Val)
 {
  analogWrite(Red_LED, Red_Val);
  analogWrite(Green_LED, Green_Val);
  analogWrite(Blue_LED, Blue_Val);
}
```

Further Reading on PWM

More information on PWM and other electronic signals can be found in the Make Guide, under the Parts Dictionary section.

delay(ms)

Pauses a program for the amount of time as specified in milliseconds, where 1000 equals 1 second.

```
delay(1000);     // waits for one  second
```

millis()

Returns the number of milliseconds since the Arduino board began running the current program as an unsigned long value.

```
value =  millis();          // sets ‘value’  equal to millis()
```

Note: This number will overflow (reset back to zero), after approximately 9 hours.

min(x, y)

Calculates the minimum of two numbers of any data type and returns the smaller number.

```
value =  min(value, 100);

// sets 'value'  to the smaller of 'value' or 100, ensuring that  it never gets above 100.
```

max(x, y)

Calculates the maximum of two numbers of any data type and returns the larger number.

```
value =  max(value, 100);

// sets 'value'  to the larger of  'value' or 100, ensuring that it is at least 100.
```

randomSeed(seed)

Sets a value, or seed, as the starting point for the random() function.

```
randomSeed(value);     // sets 'value'  as  the random  seed
```

Because the Arduino is unable to create a truly random number, randomSeed allows you to place a variable, constant, or other function into the random function, which helps to generate more random "random" numbers.

random(max) or random(min, max)

The random function allows you to return random numbers within a range specified by min and max values.

```
value =  random(100, 200);      // sets 'value'  to a  random number  between 100-200
```

Note: Use this after using the *randomSeed()* function.

Example Sketch 15 Random PWM

The following example creates a random value between 0-255 and outputs a PWM signal on a PWM pin equal to the random value: It does this for each pin (Red, Green & Blue) to create random colours.

SKETCH CODE

```
//Create Variables for the pins
const int Red_LED = 10;
const int Green_LED = 3;
const int Blue_LED = 9;

void setup() {
  //Set pins to OUTPUT to send data out
  pinMode(Red_LED, OUTPUT);
  pinMode(Green_LED, OUTPUT);
  pinMode(Blue_LED, OUTPUT);
}
void loop() {
  randomSeed(millis());   // sets millis() as  seed

  int randRed =  random(255);  // random  number  from 0-255
  int randGreen =  random(255);  // random  number  from 0-255
  int randBlue =  random(255);  // random  number  from 0-255

  analogWrite(Red_LED, randRed); // outputs PWM  signal
  analogWrite(Green_LED, randGreen); // outputs PWM  signal
  analogWrite(Blue_LED, randBlue); // outputs PWM  signal

  delay(500);     // pauses for half a  second
}
```

Example Sketch 16 Math Operations Using Serial

Another core part of Arduino is math operations. The four main operations are:

+ = addition

- = subtraction

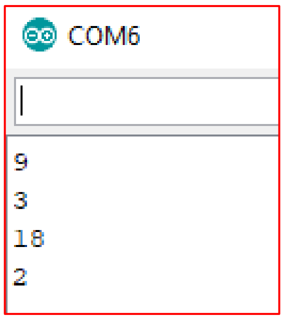
* = times

/ = divide

Create the following sketch and upload it to your 8BitCADE XL

SKETCH CODE

```
int a = 6;
int b = 3;

void setup() {
  Serial.begin(9600);
}

void loop() {
  Serial.println(a + b);// 6 + 3 = 9
  Serial.println(a - b);// 6 - 3 = 3
  Serial.println(a * b);// 6 x 3 = 18
  Serial.println(a / b);// 6 / 3 = 2
}
```

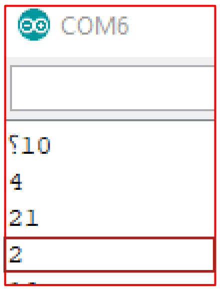


```
9
3
18
2
```

Explanation

Here we create two integers, one called ‘a’ that is 6, and one called ‘b’ that is ‘3’. We then begin serial in the void setup by typing ‘Serial.begin(9600);’ In the void loop, we then Serial print the four different operations. If we look at the serial monitor, you can see the result of these equations. Experiment by altering the a and b values or by using brackets to create more complex equations.

Note, we defined ‘a’ and ‘b’ as integers, if we were to reassign ‘a’ as ‘7’ and do a ‘/ b’ or ‘7/3’ which equals ‘2.3333333….’ recurring. However, when we serial print this we get 2. This is due to both ‘a’ and ‘b’ being integers - remember integers do not take decimal values, therefore if we wanted to show the recurring value, we would have to rewrite our program and change the datatype to something like double:



```
10
4
21
2
```

```
double a = 7;
double b = 3;
```

Now when we look at serial, we can see that we get a value of 2.33

```
2.33
```

We can utilize brackets to create more complex equations like ((a + b) / (a*b)) * (a – b)

More advanced functions consist of:

```
y = cos (x); //Cosinus
y = sin (x); //Sinus
y = sqrt (x); //Square Root
y = tan (x); //Tangent
y = log (x); //Logarithm in base e
y = log10 (x); //Logarithm in base 10
y = sq(x); //Square
```

Going Further with Arduino

You have completed the Arduino Basics! Now its time to go to your console-specific area and learn more about your console-specific code! Be sure to also check out the other foundation courses to broaden our knowledge.

Check out the Arduino reference page to learn about each function.

Check out our other tutorials at 8bitcade.com/learn We recommend:

1. 8Bit-Etch-A-Sketch
2. Calculator
3. Build your first game
4. Dino Smash
5. Space Invaders!

Thank you for following along with this tutorial. If you have any 8BitCADE XL related issues, please email us at 8BitCADE@support.com.