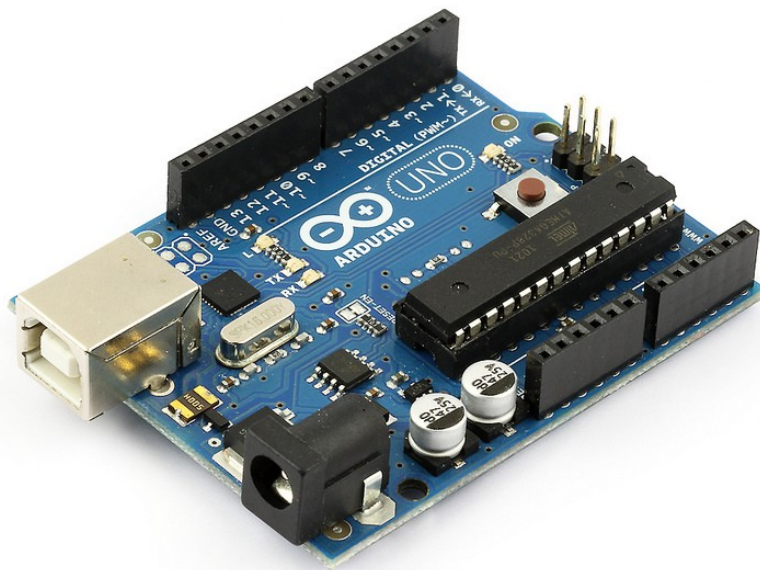
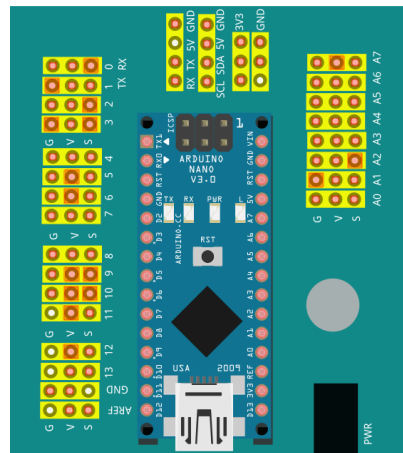


# Arduino For Beginners



# Session 1 – Basic Input and Output

Your arduino is mounted onto an expansion shield. This connects all the arduino's outputs to a lot of connection pins

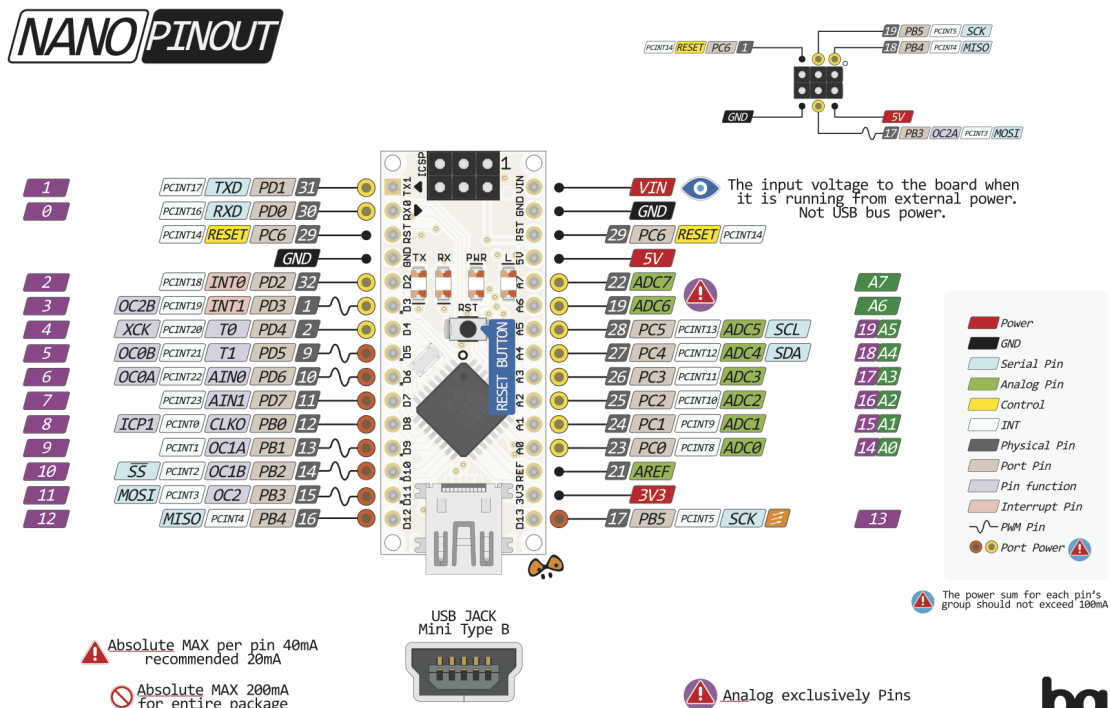


fritzing

Your board has:

- 14 digital input / output pins (labelled 0 to 13)
- 8 analogue input / output pins (labelled A0 to A7)
- A USB port for uploading sketches
- A power port for connecting to 5V to 9V DC (4AA batteries give 6V)
- Other specialised ports

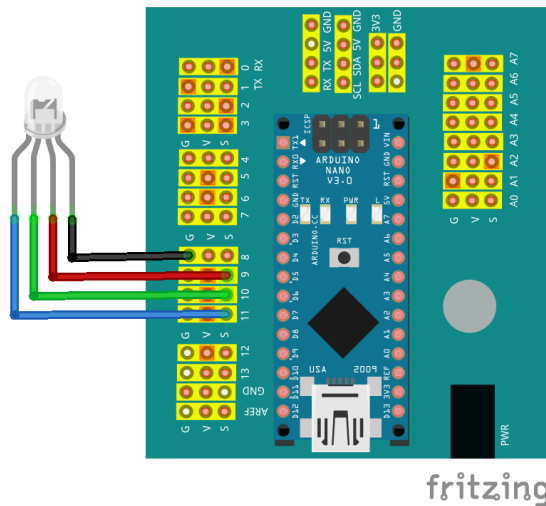
The possible connections are as shown below



## Program 1 – Hello World

In normal programming languages, the first program traditionally prints Hello World to a screen. With micro-controllers, the first program usually just makes an LED flash.

Using 4 connecting wires setup the following circuit using your arduino and an LED



The LED pin marked – needs to connect to a GND pin  
The LED pin marked R needs to connect to signal pin 9

In the arduino IDE, type the following code:

```
#define LEDpin 9

void setup ()
{
  pinMode(LEDpin, OUTPUT);
}

void loop ()
{
  digitalWrite(LEDpin, HIGH);
  delay(2000);
  digitalWrite(LEDpin, LOW);
  delay(1000);
}
```

Now click the tick to **verify** your code and if all is OK, click the arrow to **upload** it to your board and run the code.

## How it works:

```
#define LEDpin 9
```

This line tells the arduino that any time we use the **constant name** LEDpin we want the arduino to do something with digital pin 9.

```
void setup ()
{
  pinMode(LEDpin, OUTPUT);
}
```

This setup **function** which all programs must have sets the pin for the LED into output mode. Other options are INPUT and INPUT\_PULLUP (used later)

```
void loop ()
{
  digitalWrite(LEDpin, HIGH);
  delay(2000);
  digitalWrite(LEDpin, LOW);
  delay(1000);
}
```

This is the main part of the arduino program. DigitalWrite() sends data (either high (5V) or low (0V)) to the pin. delay() pauses the arduino program for the given number of milliseconds (2000 milliseconds equals 2 seconds). When the program gets to the end of the loop, it returns to the beginning and starts again.

## Challenges

- Modify the code so that the LED flashes blue instead of red.
- Modify the code so that the LED flashes quickly.
- Modify the code so that the LED flashes red, then green, then blue. (Hint change the defines to the following)

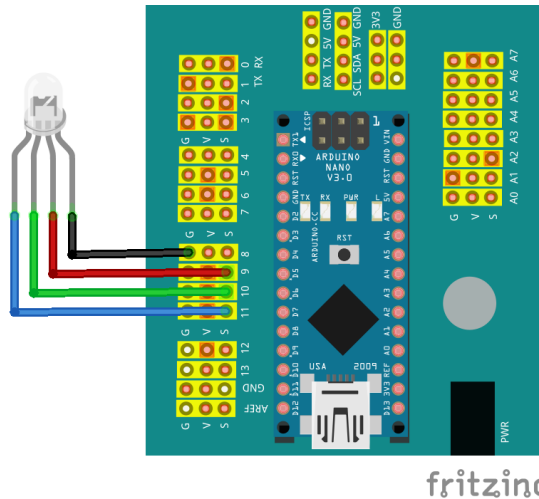
```
#define REDpin 9
#define GREENpin 10
#define BLUEpin 11
```

## Program 2 – Colour Mixing

Now that we can control an LED as either on or off, we can start to mix colours together to make new colours.

To save a lot of typing, this program also uses a function to control the output colours

Using 4 connecting wires setup the following circuit using your arduino and an LED



- The LED pin marked – needs to connect to a GND pin
- The LED pin marked R needs to connect to signal pin 9
- The LED pin marked G needs to connect to signal pin 10
- The LED pin marked B needs to connect to signal pin 11

In the arduino IDE, type the following code:

```
#define Rpin 9
#define Gpin 10
#define Bpin 11

void setup ()
{
  pinMode(Rpin, OUTPUT);
  pinMode(Gpin, OUTPUT);
  pinMode(Bpin, OUTPUT);
}

void loop ()
{
  setColour(Rpin,Gpin,Bpin, 'W');
  delay(1000);
  setColour(Rpin,Gpin,Bpin, 'R');
  delay(1000);
  setColour(Rpin,Gpin,Bpin, 'Y');
  delay(1000);
  setColour(Rpin,Gpin,Bpin, 'G');
  delay(1000);
  setColour(Rpin,Gpin,Bpin, 'C');
  delay(1000);
  setColour(Rpin,Gpin,Bpin, 'B');
  delay(1000);
  setColour(Rpin,Gpin,Bpin, 'M');
  delay(1000);
  setColour(Rpin,Gpin,Bpin, 'K');
  delay(1000);
}
```

```
void setColour(int R,int G,int B,char C)
{
  switch (C){
    case 'K':
      digitalWrite(R,LOW);
      digitalWrite(G,LOW);
      digitalWrite(B,LOW);
      break;
    case 'W':
      digitalWrite(R,HIGH);
      digitalWrite(G,HIGH);
      digitalWrite(B,HIGH);
      break;
    case 'R':
      digitalWrite(R,HIGH);
      digitalWrite(G,LOW);
      digitalWrite(B,LOW);
      break;
    case 'G':
      digitalWrite(R,LOW);
      digitalWrite(G,HIGH);
      digitalWrite(B,LOW);
      break;
    case 'B':
      digitalWrite(R,LOW);
      digitalWrite(G,LOW);
      digitalWrite(B,HIGH);
      break;
    case 'C':
      digitalWrite(R,LOW);
      digitalWrite(G,HIGH);
      digitalWrite(B,HIGH);
      break;
    case 'M':
      digitalWrite(R,HIGH);
      digitalWrite(G,LOW);
      digitalWrite(B,HIGH);
      break;
    case 'Y':
      digitalWrite(R,HIGH);
      digitalWrite(G,HIGH);
      digitalWrite(B,LOW);
      break;
  }
}
```

Notice the use of single quotes ' ' around values that need to be sent to the arduino as text **characters**. HINT, make good use of copy and paste when entering this code!!

Now click the tick to **verify** your code and if all is OK, click the arrow to **upload** it to your board and run the code.

## How it works:

The define section and the setup section are similar to before however the loop section now calls a new function called setColour().

SetColour needs 4 arguments; three integers for the Red Green and Blue pins of the LED and a character that sets the colour (**R**ed, **G**reen, **B**lue, **C**yan, **M**agenta, **Y**ellow, **W**hite, or black**K**.)

Inside the setColour function, we use a **switch** routine to run the code in one **case** block depending on the value of variable C.

```
switch (C){
  case 'K':
    digitalWrite(R, LOW);
    digitalWrite(G, LOW);
    digitalWrite(B, LOW);
    break;
  case 'W':
    // ...
}
```

We can add as many cases to this function as we like although we can only get 8 colours from our LED as it is currently set up.

## Challenges

- Modify the code so that the colour changes happen in a shorter time so that the colours start to blend together more.
- Change the order of the colour changes

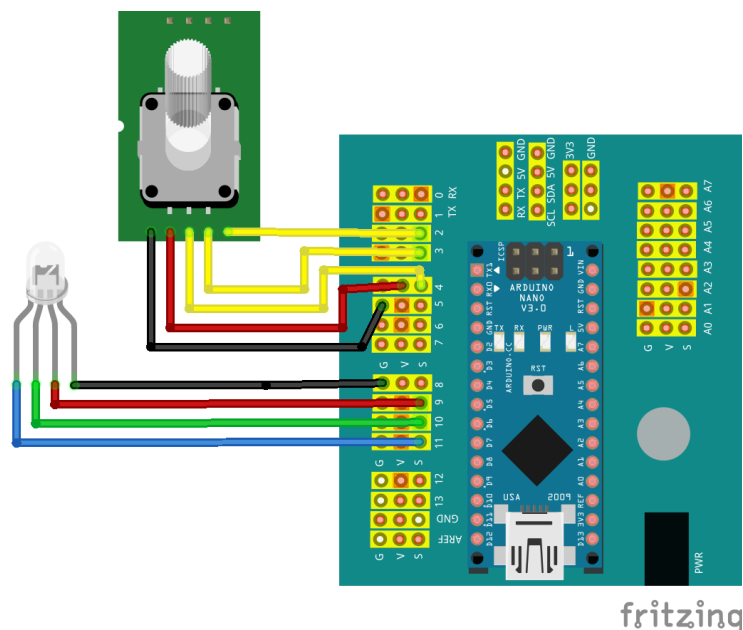
## Program 3 – Colour Control

Now that we can control the colour of an LED as either on or off, we can add a rotary encoder to provide a level of control over the LED.

A rotary encoder is like a digital switch. It has 5 connections. 5V and GND supply power to the switch. SW is connected to GND when the encoder is pushed in; we can use this as an on-off switch. CLK is either high or low depending on internal circuitry, BUT it will always change if the encoder is moved. DT is wired similarly to CLK but will change first if the encoder is turned anticlockwise or change second if the encoder is turned clockwise. We can therefore use these two connections to check if the encoder is moving or not.

This program uses the same `setColour()` function as Program 2, so that can be copied and pasted. Two global variables **state** and **colour** are used to hold the current values of the encoders settings and **CLKlast** is used to hold the last value of CLK.

Using 9 connecting wires setup the following circuit using your arduino, an LED, and a rotary encoder.





In the arduino IDE, type the following code:

```
#define Rpin 9
#define Gpin 10
#define Bpin 11
#define CLKpin 2
#define DTpin 3
#define SWpin 4

int CLKlast;
bool State=0;
volatile int Colour=0;
const char Colours[8]="RYGCBMW";

void setup ()
{
    pinMode(Rpin, OUTPUT);
    pinMode(Gpin, OUTPUT);
    pinMode(Bpin, OUTPUT);
    pinMode(CLKpin, INPUT);
    pinMode(DTpin, INPUT);
    pinMode(SWpin, INPUT_PULLUP);
    CLKlast=digitalRead(CLKpin);
    attachInterrupt (digitalPinToInterrupt (CLKpin), ISR_1, CHANGE);
}

void loop ()
{
    if(State==0)
    {
        setColour (Rpin,Gpin,Bpin, 'K');
    }else{
        setColour (Rpin,Gpin,Bpin,Colours[Colour]);
    }
    if(digitalRead(SWpin)==LOW)
    {
        State=!State;
        delay(200);
    }
}

void ISR_1()
{
    int a=digitalRead(CLKpin);
    if(digitalRead(DTpin)!=a)
    {
        if(Colour<(sizeof(Colours)-1))
        {
            Colour++;
        }
    }else{
        if(Colour>0)
        {
            Colour--;
        }
    }
}

void setColour(int R,int G,int B,char C)
{
    switch (C){
        case 'K':
            digitalWrite(R,LOW);
            digitalWrite(G,LOW);
            digitalWrite(B,LOW);
            break;
        case 'W':
            digitalWrite(R,HIGH);
            digitalWrite(G,HIGH);
            digitalWrite(B,HIGH);
            break;
    }
}
```

```
    case 'R':  
        digitalWrite(R, HIGH);  
        digitalWrite(G, LOW);  
        digitalWrite(B, LOW);  
        break;  
    case 'G':  
        digitalWrite(R, LOW);  
        digitalWrite(G, HIGH);  
        digitalWrite(B, LOW);  
        break;  
    case 'B':  
        digitalWrite(R, LOW);  
        digitalWrite(G, LOW);  
        digitalWrite(B, HIGH);  
        break;  
    case 'C':  
        digitalWrite(R, LOW);  
        digitalWrite(G, HIGH);  
        digitalWrite(B, HIGH);  
        break;  
    case 'M':  
        digitalWrite(R, HIGH);  
        digitalWrite(G, LOW);  
        digitalWrite(B, HIGH);  
        break;  
    case 'Y':  
        digitalWrite(R, HIGH);  
        digitalWrite(G, HIGH);  
        digitalWrite(B, LOW);  
        break;  
    }  
}
```

Notice the use of the double equals sign == in the loop's if statements. This compares the values rather than changing them. Also note the != which translates as not equals.

Now click the tick to **verify** your code and if all is OK, click the arrow to **upload** it to your board and run the code.

## How it works

The variables section has become a little more complicated this time

```
int CLKlast;  
bool State=0;  
volatile int Colour=0;  
const char Colours[8]="RYGCBMW";
```

The volatile declaration tells the arduino that the value of Colour might be changed by something other than the main program loop, in this case it is an interrupt. The use of square brackets [] in the declaration of Colours sets this up as an **array** (or list) of characters not as a single word. The individual letters (to be passed to our setColour() function) can be accessed using square brackets. Colours[0] = R, Colours[5] = M. As this is an array of characters, it needs to be declared 1 space longer than the string, hence the 8 even though there are only 7 characters.

The setup section has also got longer

```
void setup ()  
{  
  pinMode(Rpin, OUTPUT);  
  pinMode(Gpin, OUTPUT);  
  pinMode(Bpin, OUTPUT);  
  pinMode(CLKpin, INPUT);  
  pinMode(DTpin, INPUT);  
  pinMode(SWpin, INPUT_PULLUP);  
  CLKlast=digitalRead(CLKpin);  
  attachInterrupt(digitalPinToInterrupt(CLKpin),ISR_1, CHANGE);  
}
```

Three pins for the rotary encoder have been declared as inputs and the switch input has been also declared as a pullup input. This means that when nothing is pressed, the arduino sees 5V on this pin. Our button, then connects to GND so the arduino sees 0V as meaning pressed.

The CLKpin is also attached to an interrupt. The Arduino nano has 2 interrupts on pins 2 and 3 that can monitor sensors for changes without stopping the main loop from running. In this case, if the arduino sees a CHANGE on the CLKpin it will immediately stop what it is doing and go and run the function ISR\_1() (Interrupt Service Routine 1). Once this has run it jumps back to the main program.

The main loop function is fairly easy to follow.

```
void loop ()  
{  
  if(State==0)  
  {  
    setColour(Rpin,Gpin,Bpin,'K');  
  }else{  
    setColour(Rpin,Gpin,Bpin,Colours[Colour]);  
  }  
  if(digitalRead(SWpin)==LOW)  
  {  
    State=!State;  
    delay(200);  
  }  
}
```

The first if statement checks the value of the variable State. If it is off, then it sets the colour to black, if it is on, it sets it to one of the colours in the string stored earlier as Colours.

The second if statement checks the value of the switch input to see if it is pressed. If it is being pressed when it checks, then the value of state is reversed from a 0 to a 1 or vice-versa. The !

Means “not” or “opposite”. The short delay “debounces” the switch so that the LED doesn’t just flash repeatedly.

When the arduino sees a changing value on the CLK pin, it calls the Interrupt Service Routine

```
void ISR_1()
{
    int a=digitalRead(CLKpin);
    if(digitalRead(DTpin)!=a)
    {
        if(Colour<(sizeof(Colours)-1))
        {
            Colour++;
        }
    }else{
        if(Colour>0)
        {
            Colour--;
        }
    }
}
```

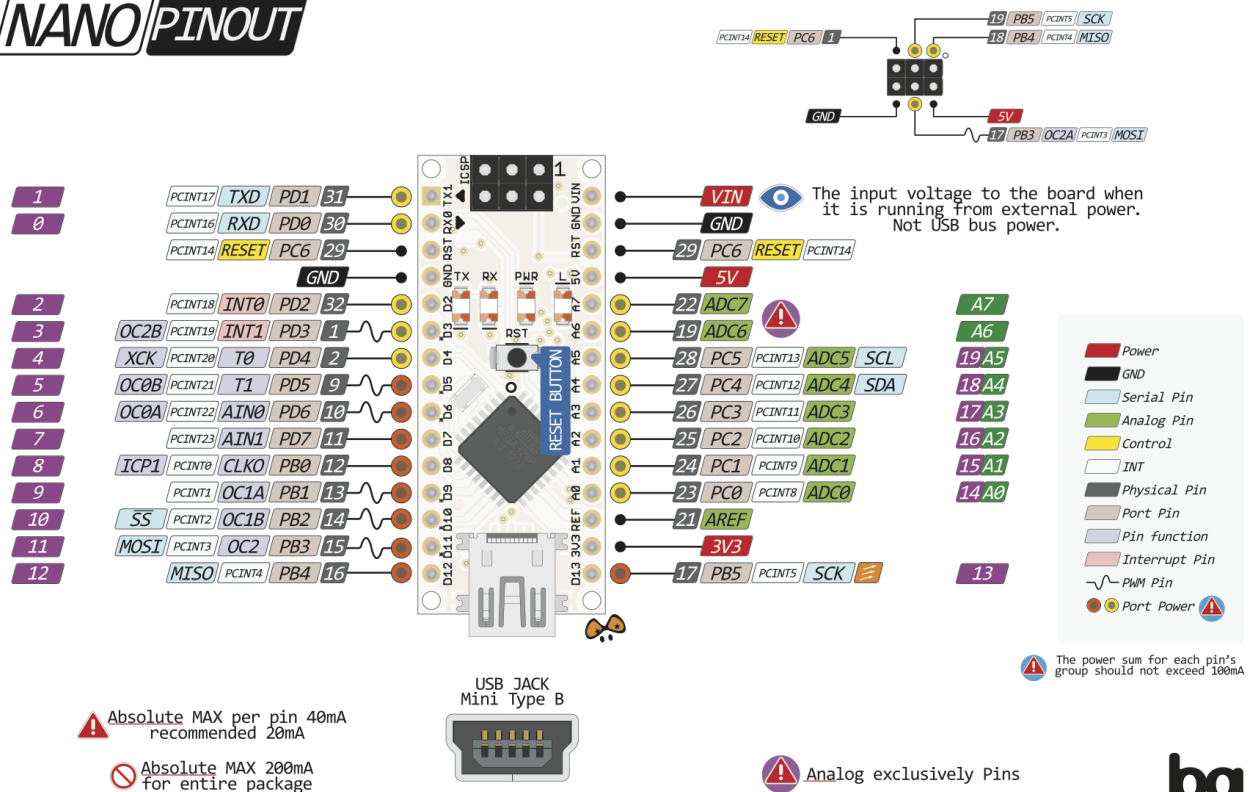
This routine already knows that the CLK pin has changed so stores the current value in local variable a. It then checks the DT pin. If the DT pin has not yet changed, then the encoder has been turned clockwise and Colour should be increased by 1. If the DT pin has already changed, i.e. it has the same value as CLK, then the encoder went anticlockwise and Colour should be decreased by 1. the ++ and -- make these increments and decrements. Because Colour has to stay between 0 and 6 (for the seven colours of our string) it is necessary to check that the current value is not already at the extremes before making the changes. If it is already at the endpoints, then the change is ignored.

## Session 2 – Variable Output

Six of the pins on your arduino are capable of switching on and off very rapidly. This is called pulsed width modulation or PWM. This can be used to produce an average output somewhere between off and on using the `analogWrite()` function.

This can be used to fade LED's, move a servo motor or control the speed of a small motor (so long as you don't connect it directly)

### NANO PINOUT

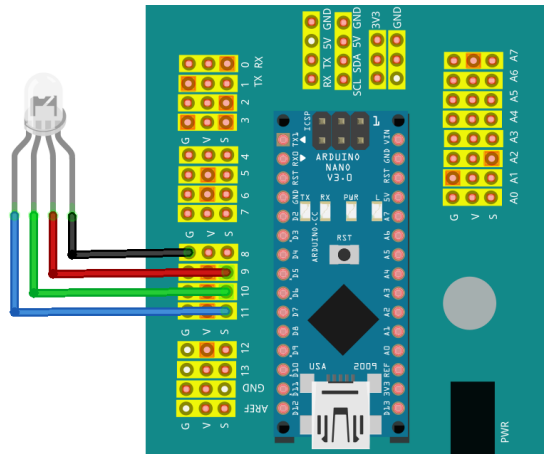


Pins 3, 5, 6, 9, 10 and 11 are capable of PWM

## Program 4 – Improved Colour Control 1

Eight colours is good, but more is better.

Using 4 connecting wires setup the following circuit using your arduino and an LED



fritzing

The LED pin marked – needs to connect to a GND pin

The LED pin marked R needs to connect to signal pin 9

The LED pin marked G needs to connect to signal pin 10

The LED pin marked B needs to connect to signal pin 11

In the arduino IDE, type the following code:

```
#define Rpin 9
#define Gpin 10
#define Bpin 11

void setup()
{
  pinMode(Rpin, OUTPUT);
  pinMode(Gpin, OUTPUT);
  pinMode(Bpin, OUTPUT);
}

void loop()
{
  for(int i=0; i<=255; i+=5)
  {
    analogWrite(Bpin,i);
    delay(50);
  }
  for(int i=255; i>=0; i-=5)
  {
    analogWrite(Bpin,i);
    delay(50);
  }
}
```

Notice that `analogWrite` takes two values; a pin to modulate and a value between 0 and 255. The for loop is a common control mechanism and usually uses a local variable called i, j, k etc to keep track of the number of loops.

Now click the tick to **verify** your code and if all is OK, click the arrow to **upload** it to your board and run the code.

## How it Works

The **for loops** in the main loop function perform most of the control here. The first loop ramps up the PWM time from 0 to 255 in steps of 5 and the second ramps it back down again.

The same effect can be achieved using a while loop as below:

```
int i=0;

void setup()
{

}

void loop()
{
  while(i<=255)
  {
    analogWrite(Rpin,i);
    delay(25);
    i++;
  }
  while(i>=0)
  {
    analogWrite(Rpin,i);
    delay(25);
    i--;
  }
}
```

This while loop can be more effective when you need to change multiple variables within a loop.

## Challenges:

- Modify the code so that the LED fades on and off in cyan.
- Place the code causing the flashing within this loop:

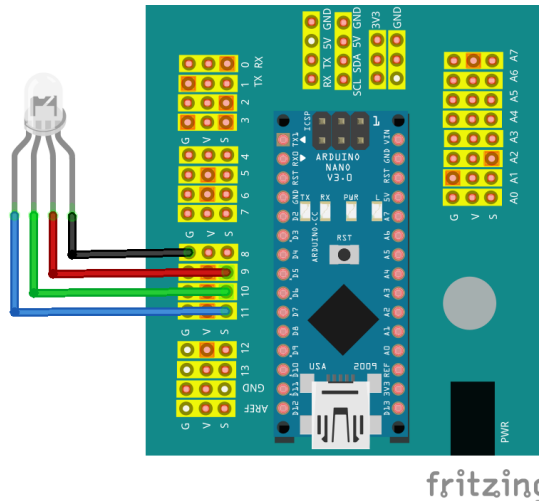
```
for(pin=9; pin<=11; pin++){
}
```

- Now modify the code so that the LED fades red, then green, then blue.

## Program 5 – Improved Colour Control 2

Now it is time to modify our earlier code so that the setColour() function can be more flexible.

Using 4 connecting wires setup the following circuit using your arduino and an LED



- The LED pin marked – needs to connect to a GND pin
- The LED pin marked R needs to connect to signal pin 9
- The LED pin marked G needs to connect to signal pin 10
- The LED pin marked B needs to connect to signal pin 11

In the arduino IDE, type the following code:

```
#define Rpin 9
#define Gpin 10
#define Bpin 11

void setup ()
{
  pinMode(Rpin, OUTPUT);
  pinMode(Gpin, OUTPUT);
  pinMode(Bpin, OUTPUT);
}

void loop ()
{
  setColour(Rpin,Gpin,Bpin, 'W');
  delay(1000);
  setColour(Rpin,Gpin,Bpin, 'R');
  delay(1000);
  setColour(Rpin,Gpin,Bpin, 'B');
  delay(1000);
  setColour(Rpin,Gpin,Bpin, 'G');
  delay(1000);
  setColour(Rpin,Gpin,Bpin, 'O');
  delay(1000);
  setColour(Rpin,Gpin,Bpin, 'B');
  delay(1000);
  setColour(Rpin,Gpin,Bpin, 'R');
  delay(1000);
  setColour(Rpin,Gpin,Bpin, 'K');
  delay(1000);
}

void setColour(int R,int G,int B,char C)
{
  int Red=0;
  int Green=0;
```



```

int Blue=0;
switch (C){
  case 'W':
    Red=255;
    Green=255;
    Blue=255;
    break;
  case 'K':
    break;
  case 'R':
    Red=255;
    break;
  case 'G':
    Green=255;
    break;
  case 'B':
    Blue=255;
    break;
  case 'O':
    Red=200;
    Green=100;
    break;
}
LedSet (R,G,B,Red,Green,Blue);
}

void LedSet(int Rpin,int Gpin,int Bpin, int Rval, int Gval, int Bval)
{
  analogWrite(Rpin,Rval);
  analogWrite(Gpin,Gval);
  analogWrite(Bpin,Bval);
}

```

Notice that the case statements only set the colours that are not zero. This is because the function assigns Red, Green and Blue to 0 at each calling. Also note that setColour() calls another function LedSet(). This means that you can use the preset colours of setColour or create your own colour using LedSet.

Now click the tick to **verify** your code and if all is OK, click the arrow to **upload** it to your board and run the code.

## Challenges

- Add your own colours to the setColour() function.
- Write a function that takes a letter and then fades the LED on that colour.

```

void fadeLED(char C)
{
}

```

- Extend you function to include a variable that controls the speed of the fade.

```

void fadeLED(char Col, int Step)
{
}

```

## Program 6 – Improved Colour Control 3

Having one LED is good, but 8 is better! However, trying to simply add more LED's quickly gets limiting. The arduino could only support 2 RGB LEDs if you want to control their brightness!

Neopixels are a solution to this problem. These are a series of LED's with on board chips to control them. A single data line can then control the whole string of LEDs.

Connect your strip of 8 NeoPixels to pin 12 of the arduino

DIN → S, GND → G, 4-7V → V

In the arduino IDE,

- Click Sketch → Include Library → Manage Libraries
- Search for NeoPixels
- Click on Adafruit\_NeoPixel
- Click Install

Now type the following code:

```
#include <Adafruit_NeoPixel.h>

#define Dpin 12
#define NumPix 8

Adafruit_NeoPixel strip = Adafruit_NeoPixel(NumPix, Dpin, NEO_GRB +
NEO_KHZ800);

void setup()
{
  strip.begin();
}

void loop()
{
  chase(strip.Color(255, 0, 0));
  chase(strip.Color(0, 255, 0));
  chase(strip.Color(0, 0, 255));
}

void chase(long c)
{
  for(int i=0; i<strip.numPixels()+3; i++)
  {
    strip.setPixelColor(i, c);
    strip.setPixelColor(i-3, 0);
    strip.show();
    delay(100);
  }
}
```

Notice that the line `Adafruit_NeoPixel strip = Adafruit_NeoPixel(NumPix, Dpin, NEO_GRB + NEO_KHZ800);` creates a new type of object called an `Adafruit_neoPixel` and calls it **strip**. The `NEO_GRB + NEO_KHZ800` are needed to define the type of neopixels being used.

Now click the tick to **verify** your code and if all is OK, click the arrow to **upload** it to your board and run the code.

## How it Works

The library produces a long string of data that it sends out down the data pin when `strip.show()` is called.

The first pixel takes the 3 bytes of data it wants (its Green value, Red value and Blue Value) and then sends the rest of the data down the line.

`Strip.setPixelColor(n,R,G,B)` sets the *n*th pixel to colour RGB (values between 0 and 255)

`strip.Color(R,G,B)` creates a long variable type that can then be used to set colours the quick way later.

```
long magenta=strip.Color(255,0,255);
for(i=0;i<strip.numPixels;i++)
{
    strip.setPixelColor(i,magenta);
}
strip.show()
```

## Challenges

- Write a function that builds up the number of pixels slowly rather than chasing them
- Produce a rainbow on your NeoPixels.

## Session 3 – Driving Other Outputs

Having LEDs changing colour is all well and good, but other outputs are useful too.

The servo library is a common output library

A servo is a motor that changes its position based on a PWM signal. Sending a value of 1500 sets it to centre. 1000 turns it 90 degrees acw and 2000 turns it 90 degrees cw. The servo library makes things easier by converting these PWM signals into degrees from 0 to 180.

### Program 7 – Servo Control

Connect a servo to pin 11 of your arduino.

```
#include <Servo.h>

#define servoPin 11

Servo servo_1;
int pos = 0;

void setup()
{
  servo_1.attach(servoPin);
}

void loop()
{
  for (pos = 0; pos <= 180; pos += 1)
  {
    servo_1.write(pos);
    delay(15);
  }
  for (pos = 180; pos >= 0; pos -= 1)
  {
    servo_1.write(pos);
    delay(15);
  }
}
```

Notice the new type of object Servo. When the servo is setup it is names servo\_1 and attached to pin 11. multiple servos can obviously be attached to multiple pins.

Now click the tick to **verify** your code and if all is OK, click the arrow to **upload** it to your board and run the code.

If this works, it can be merged with the rotary encoder code from earlier

```
#include <Servo.h>

#define servoPin 11
#define CLKpin 2
#define DTpin 3
#define SWpin 4

Servo servo_1;
volatile int pos = 0;
int CLKlast;

void setup ()
{
  servo_1.attach(servoPin);
  pinMode(CLKpin, INPUT);
  pinMode(DTpin, INPUT);
  CLKlast=digitalRead(CLKpin);
  attachInterrupt (digitalPinToInterrupt (CLKpin),ISR_1, CHANGE);
}

void loop ()
{
  servo_1.write(pos);
  delay(15);
}

void ISR_1()
{
  int a=digitalRead(CLKpin);
  if(digitalRead(DTpin)!=a)
  {
    if(pos<180)
    {
      pos++;
    }
  }else{
    if(pos>0)
    {
      pos--;
    }
  }
}
```

Now click the tick to **verify** your code and if all is OK, click the arrow to **upload** it to your board and run the code.

The rotary encoder should now control the position of the servo.

## Challenges

- The rotary encoder has 30 indents on a full rotation. Adjust the ISR so that one turn of the encoder gives a full 180 degrees of motion
- Add a touch sensor so that the servo only moves when the sensor is pressed.

```
#define touchpin 6
void setup()
{
  pinMode(TouchPin, INPUT)
}

void loop()
{
  if(digitalRead(TouchPin)==HIGH)
  {
    //switch pressed so do stuff
  }else{
    //switch not pressed so do other stuff
  }
}
```

- A Wii nunchuck can be connected to the arduino using the SDA and SCL ports at the end of the arduino. The redwire on the nunchuck cable is GND and the white is SCL
  - Search the libraries for the WiiChuck library
  - include the following libraries

```
#include <Nunchuck.h>
#include <Wire.h>
```

- setup the nunchuck as follows

```
nunchuck.begin();
nunchuck.addMap(new Nunchuck::joyX(5,200,128,10));
nunchuck.addMap(new Nunchuck::joyY(6,200,128,10));
nunchuck.addMap(new Nunchuck::buttonZ(7,200,128,10));
nunchuck.addMap(new Nunchuck::buttonC(8,200,128,10));
```

- Get data from the nunchuck as follows

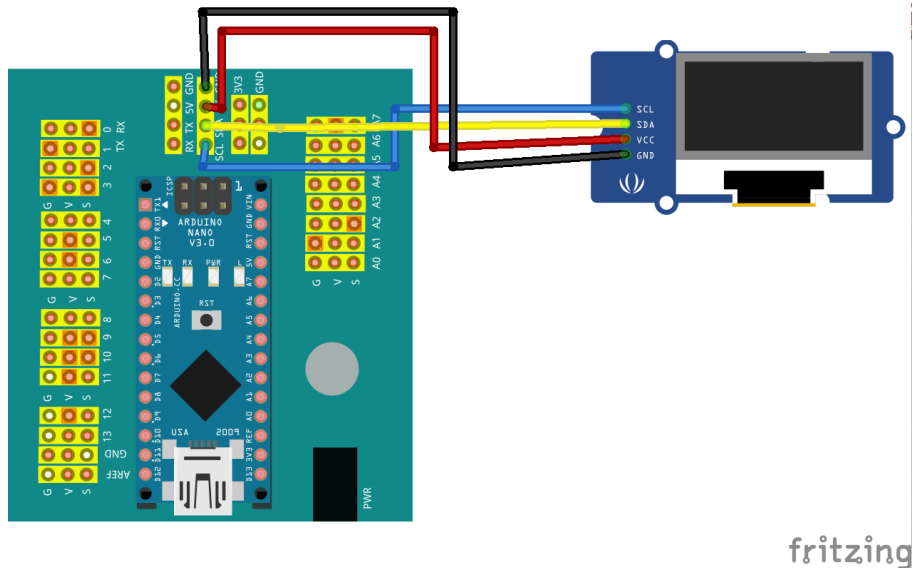
```
nunchuck.readData();
//Then use one of these functions
nunchuck.getJoyX()
nunchuck.getJoyY()
nunchuck.getRollAngle()
nunchuck.getPitchAngle()
nunchuck.checkButtonC()
nunchuck.checkButtonZ()
```

- Modify the Rotary encoder code to use the nunchuck to move the servo

## Program 8 – Driving an OLED Screen

A screen can also be connected to the arduino using the SDA and SCL pins. If you want to avoid using the port at the end, pins A4 and A5 are also connected to SDA and SCL respectively.

Connect the OLED screen as follows.



One library is needed to drive the OLED screen. Download this as before

U8glib library

Now enter the following code into the arduino IDE

```
#include "U8glib.h"

U8GLIB_SH1106_128X64 u8g(U8G_I2C_OPT_NONE);

int yPos = 0;

void setup()
{
  // flip screen, if required
  //u8g.setRot180();
  u8g.setFont(u8g_font_unifont);
  u8g.setColorIndex(1);
}

void loop()
{
  u8g.firstPage();
  do {
    draw();
  } while( u8g.nextPage() );
  //delay(1000);
  if(yPos < 83)
  {
    yPos++;
  }else{
    yPos = 0;
  }
}

void draw()
{
  u8g.drawStr( 0, yPos, "Hello World");
}
```

Try changing the text to something else or printing the output of the nunchuck.

Try drawing circles or discs with:

```
u8g.drawCircle(X,Y,Radius)
u8g.drawDisc(X,Y,Radius)
u8g.drawEllipse(X,Y,Rx,Ry)
u8g.drawFilledEllipse(X,Y,Rx,Ry)
```

or squares with:

```
u8g.drawFrame(X,Y,width,height)
u8g.drawBox(X,Y,width,height)
u8g.drawRBox(X,Y,width,height,radius)
u8g.drawRFrame(X,Y,width,height,radius)
```

or lines with:

```
u8g.drawLine(X1,Y1,X2,Y2)
```

or triangles (to build polygons) with:

```
u8g.drawTriangle(X1,Y1,X2,Y2,X3,Y3)
```

or pixels with:

```
u8g.drawPixel(X1,Y1)
```

or text with:

```
u8g.drawStr(X1,Y1,"text")
```