Lesson 1

An Introduction To Arduino

01

DFROBOT
DRIVE THE FUTURE

What is Arduino ?

Arduino is an open-source hardware and software platform designed for computer programmers, industrial artists, professionals and those interested in developing interactive devices and applications specific to an interactive development environment.

Arduino can receive input signals from various sensors and inputs. By controlling light sources, motors, or other actuators, Arduino can change the surrounding environment. Programs for the microcontroller on the Arduino board are written in Arduino's programming language (based on "Wiring" - an open source framework for microcontrollers) and run in the Arduino development environment (based on "Processing" - an open source programming language and integrated development environment).
Arduino is able to run independently or communicate with software running on a computer (for instance, Flash, Processing and MaxMSP). The open-source Arduino IDE, which is free to download, makes it easy for you to write code, upload it to the board and come up with your own interactive devices.

What can you do with Arduino?:

- Make a line-following robot
- Make a fluffy toy that lights up
- Make your phone ring when you receive an e-mail
- Make a Metroid-style arm cannon
- Make a coffee maker that sounds an alarm when your coffee is ready
- Make device that can record your heart rate when you ride your bike

All of these things can be achieved with Arduino!

The Birth of Arduino

Arduino started in a small picturesque town in northern Italy as as a project for students at the Interaction Design Institute Ivrea. Members of Arduino's core development team were Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino, David Mellis and Nicholas Zambetti.

Massimo Banzi's students often complained that they couldn't find a cheap and easy-to-use microcontroller. In the winter of 2005, Banzi mentioned this issue while talking with David Cuartielles, a Spanish chip engineer who was a visiting scholar to Massimo's university. They decided to design their own circuit board and brought Banzi's student David Mellis into this project to design the programming language for their board. Mellis finished the source code within two days. It took them another three days to etch the circuit board. They named it Arduino.

Now, anyone is able to make magic with Arduino, even without knowledge of computer programming. With Arduino, you can make a device respond to its0 environment, make flashing light shows

Why "Arduino"?

Ivrea is famous for the story of an oppressed king: In 1002 AD, King Arduino took the crown of Italy; however, in 1004 AD, he was dethroned by King Henry II of Germany.

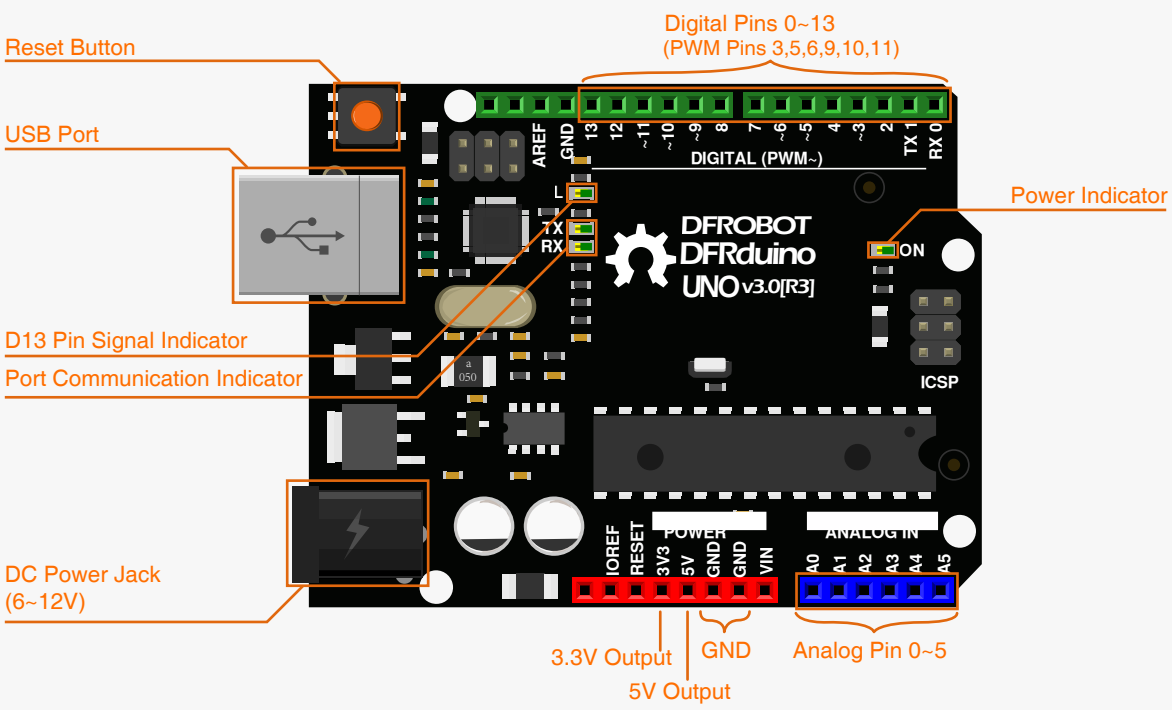
There is now a bar named "de Re Arduino" on Cobblestone Street in Ivrea which was opened in commemoration of King Arduino. Massimo Banzi was a regular customer there, and he fondly named his open-source hardware platform after it.

Introduction to DFRduino

In these lessons we will use DFRobot's own clone of an Arduino Uno board: "DFRduino Uno". DFRduino operates in exactly the same way as an Arduino Uno. To save confusion with names, let's just call it the microcontroller. Let's examine its features:

Sections with annotations below are parts that will be regularly used. Digital pins and analog pins marked on the diagram are what we call I/O (input/output). Digital pins are numbered from 0 to 13. Analog pins are numbered from 0 to 5.

Voltage supply is also shown on the diagram below. DFRduino can be powered either by a USB connection or through a 6 ~ 12V DC supply via the barrel jack connector. Also integrated in to the board are four LED lights and a reset button. The LED light marked with "ON" is the power indicator, which will be on once power is connected. The LED light marked with "L" is an indicator for digital pin 13 which will be discussed in the next section. TX (transmit) and RX (receive) are indicator lights for serial communication. When we upload a program, these two lights will blink rapidly, showing that data is being transmitted and received between the board and your computer.



First Use

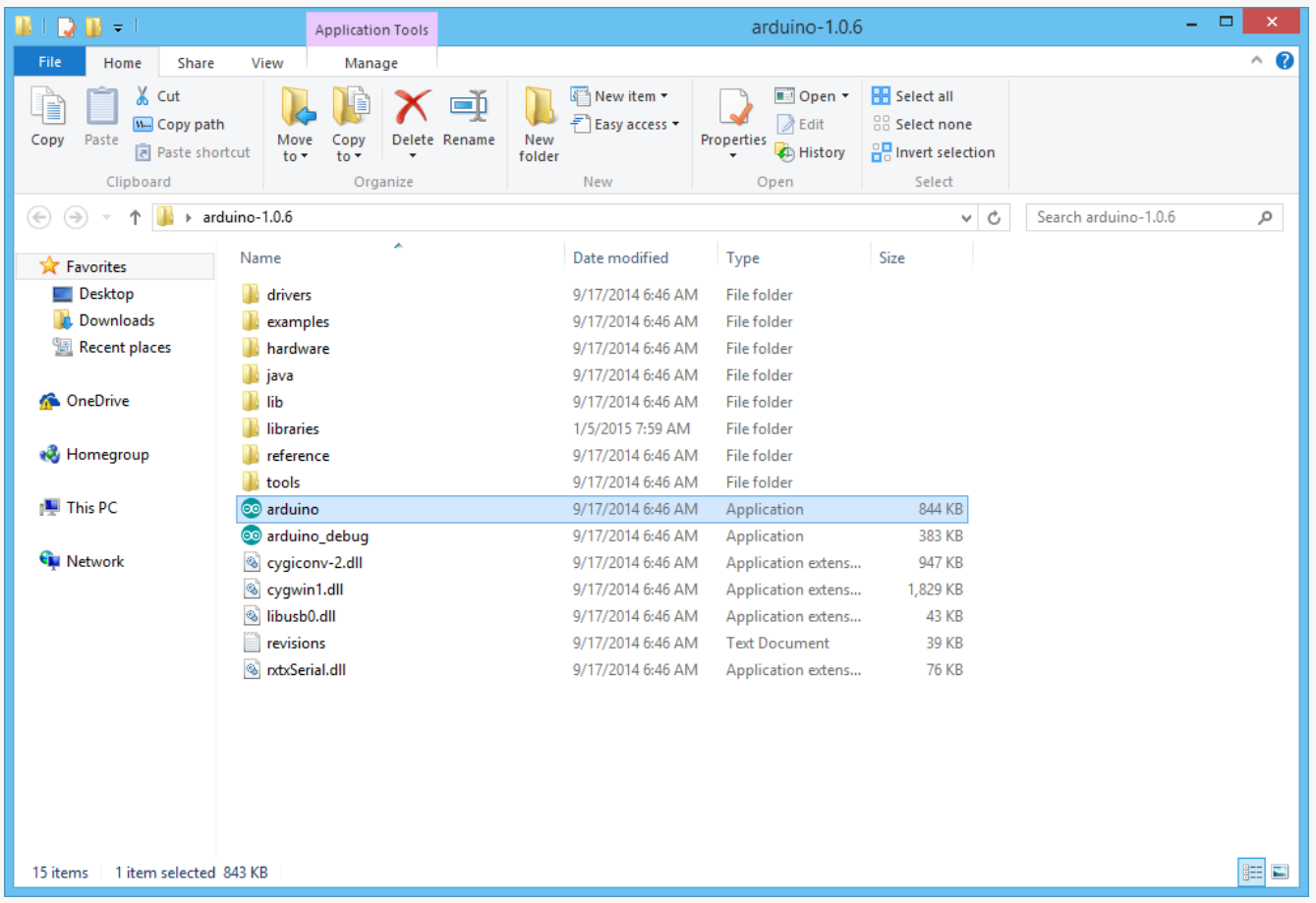
1. Download Arduino IDE

You can download Arduino IDE from this website:

http://www.arduino.org/software

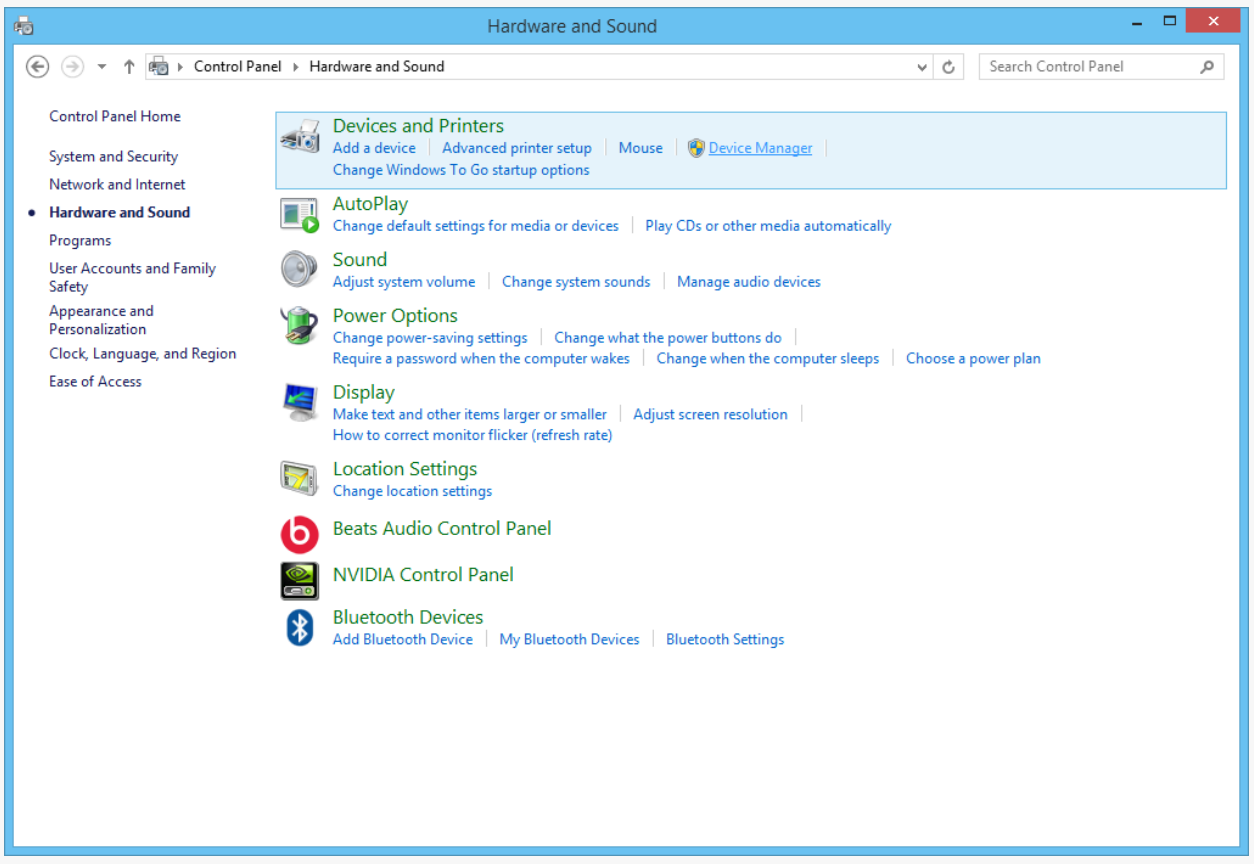
For Windows users, please click "Windows (ZIP file)". For Mac and Linux users, please select the corresponding link for your operating system.

Once downloaded, extract the files to a directory of your choice. Once extracted, open the directory. It should look like the image below:

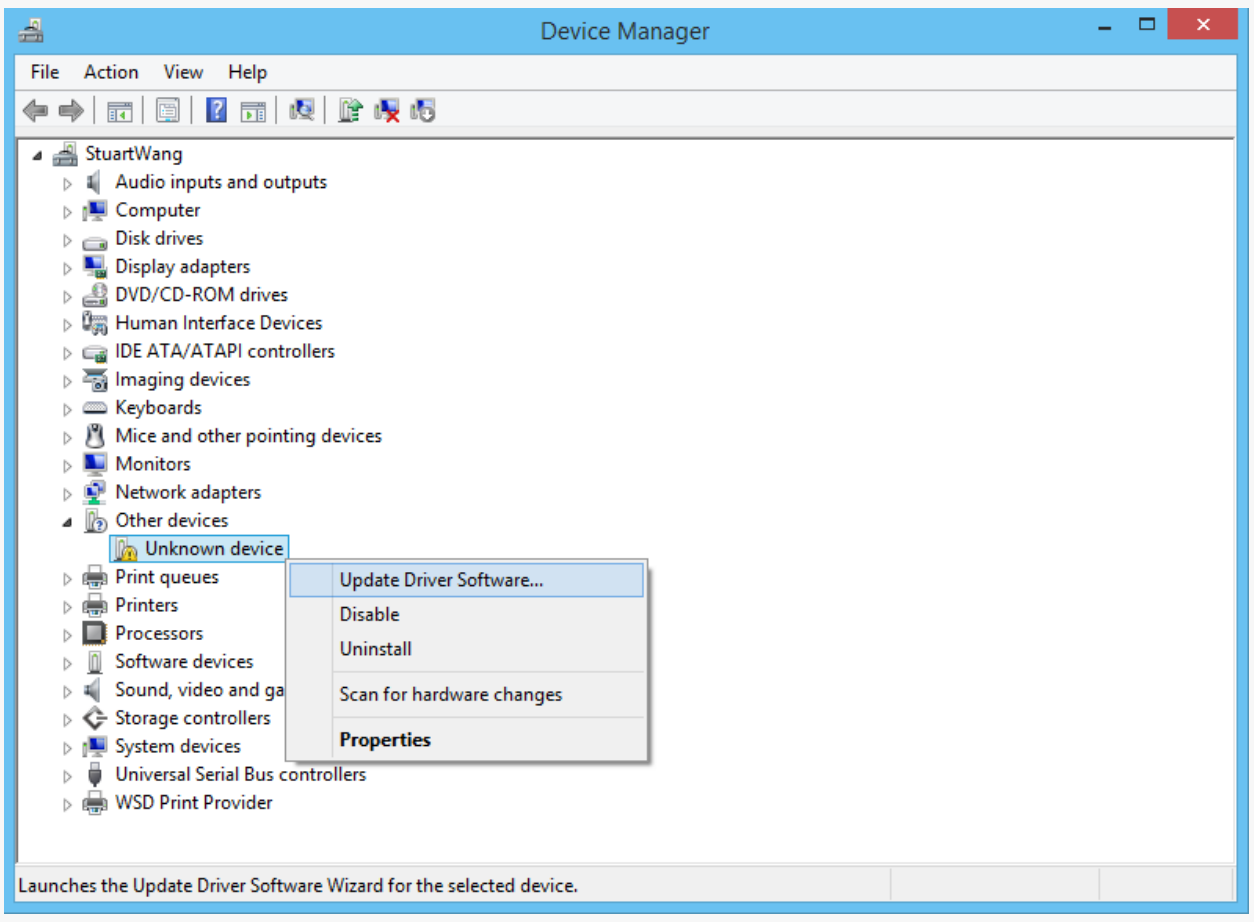


2. Install Drivers

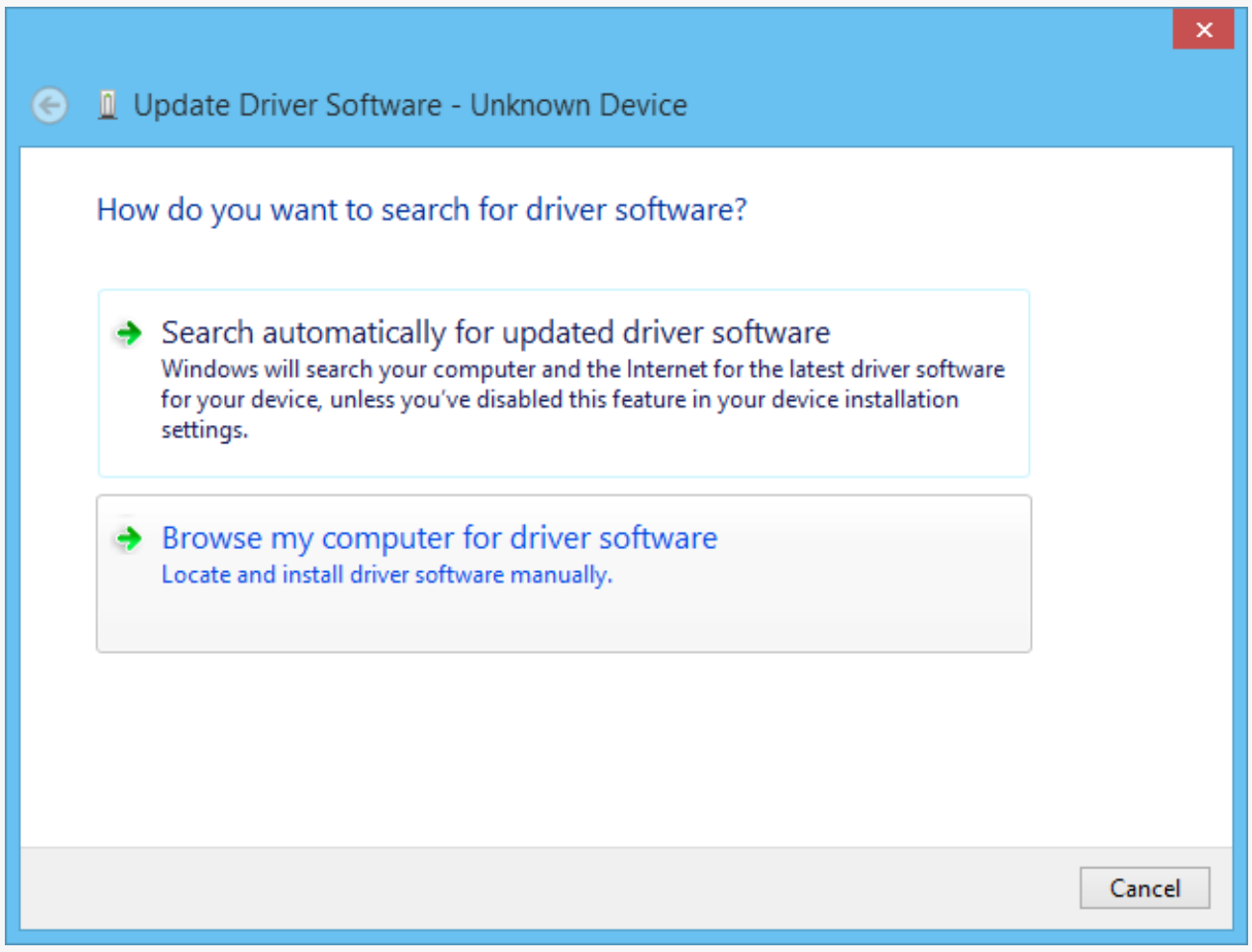
Connect the microcontroller to your computer using a USB cable. Once connected, the board's power indicator light will turn on.



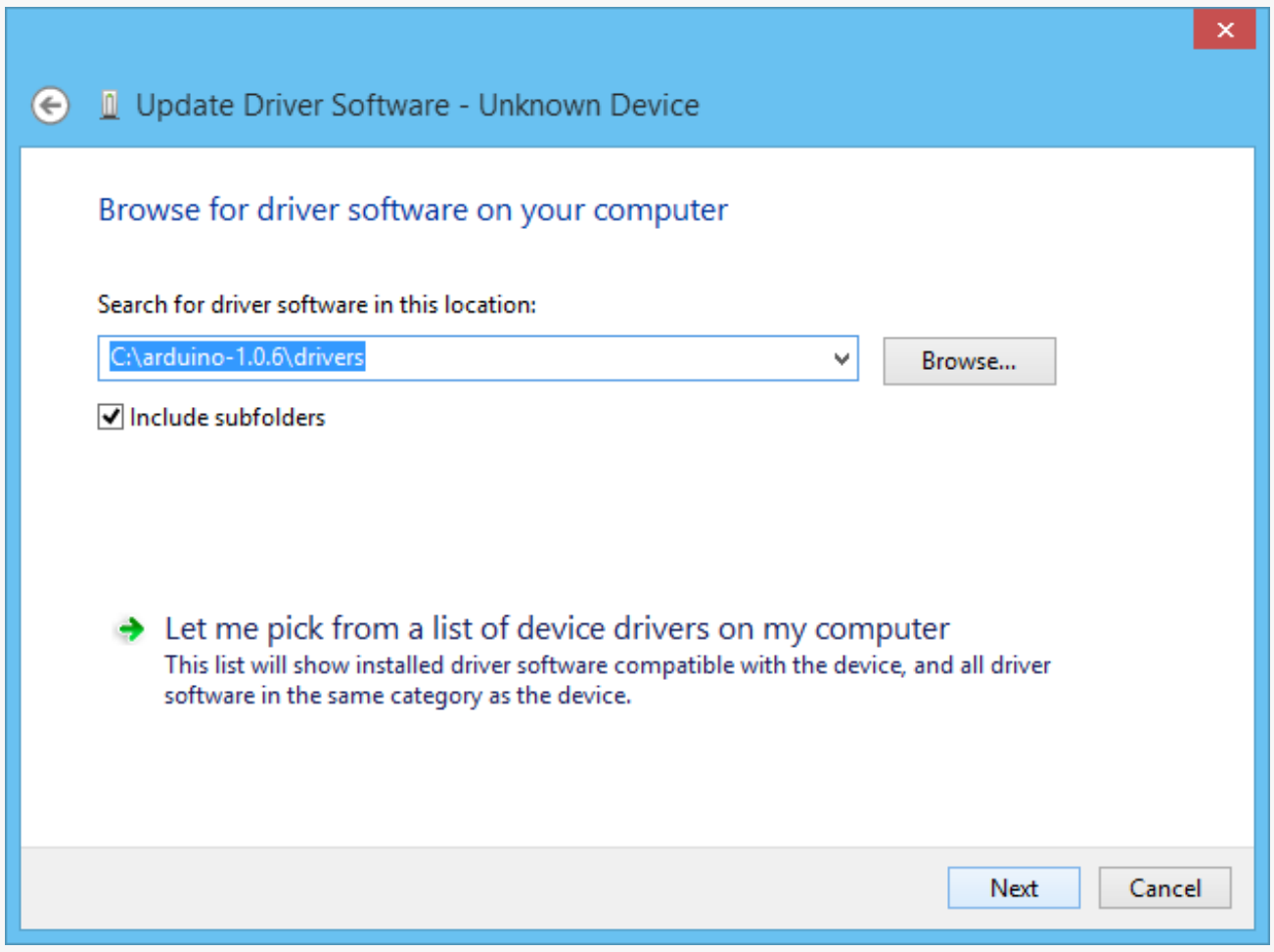
Open "Control Panel" and navigate to "Device Manager" to set up the drivers.



You will then see a dialog box pop up. Select the second item “Manually Search for Drivers”. A browse window will appear that requires you to point it to the correct directory.



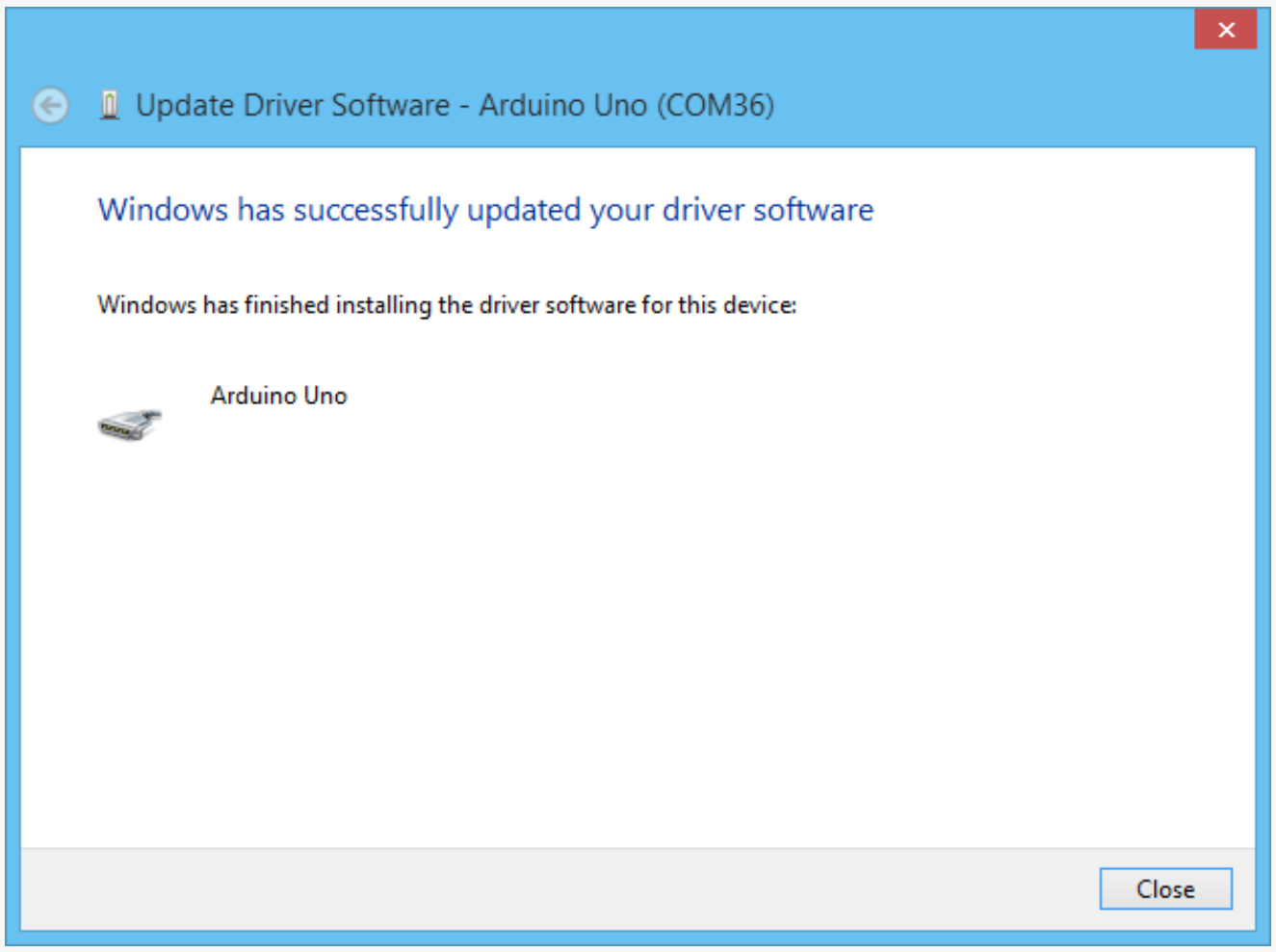
Navigate the browse dialog to your Arduino directory. Inside this directory, there is a subfolder called “drivers” where the Arduino drivers are stored. Select this directory and then click “Next”.



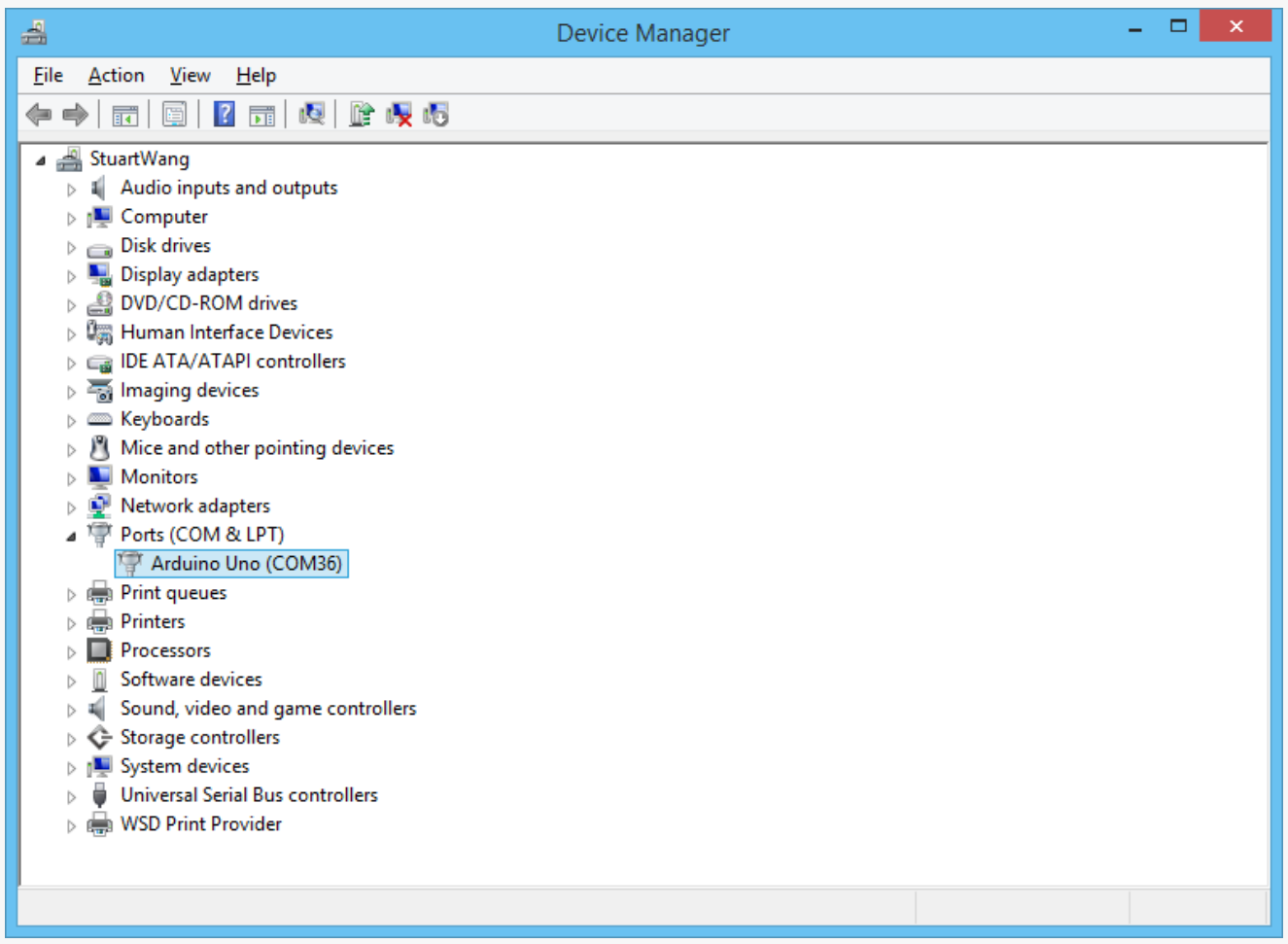
Hopefully this window will appear. This shows that drivers have been successfully installed!

If you encounter issues, check this website for help:

https://www.arduino.cc/

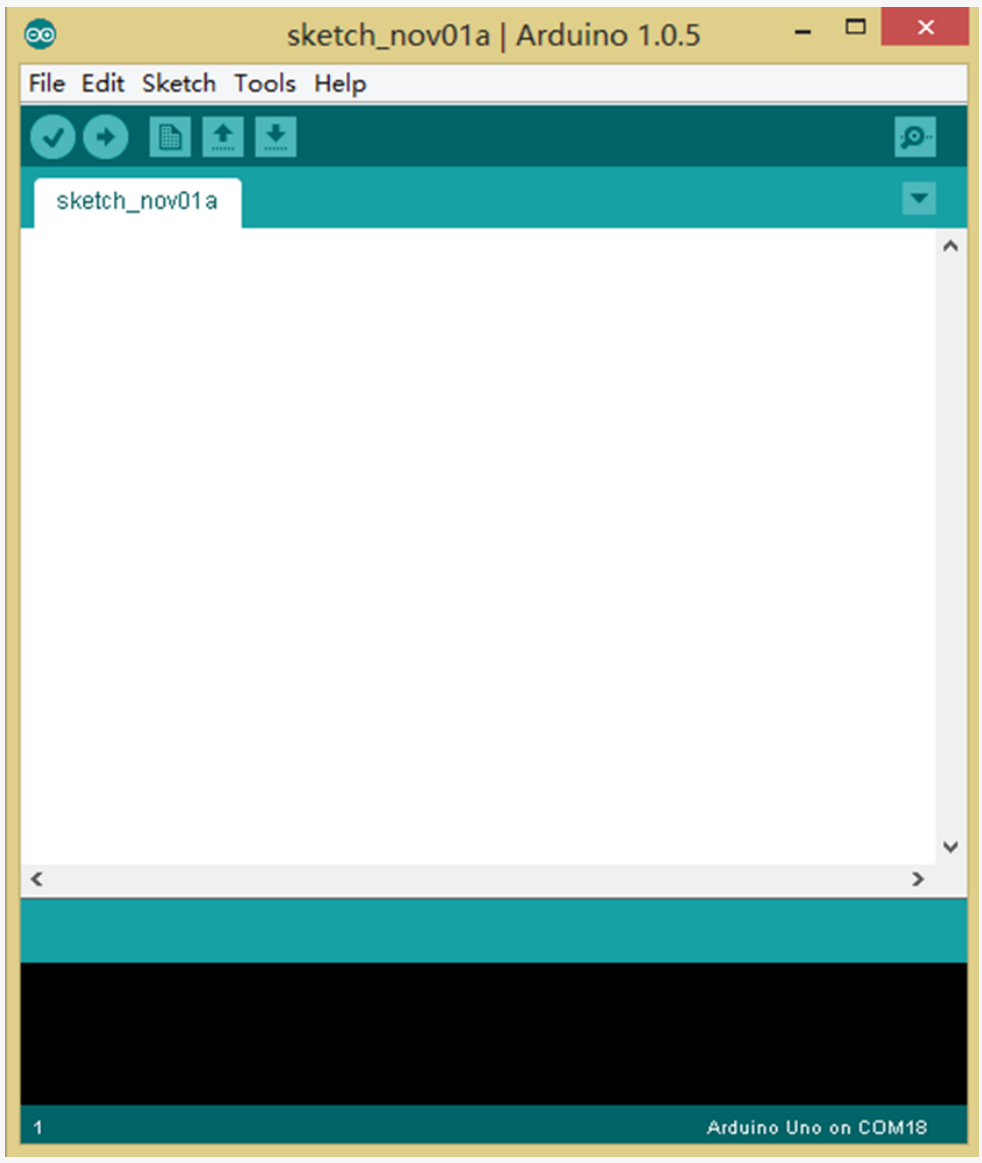


Return to “Device Manager”. The computer will have assigned a serial port to the microcontroller (your computer will show it as Arduino Uno). The serial port will vary depending on your computer, but should appear as “COM”, followed by a number.

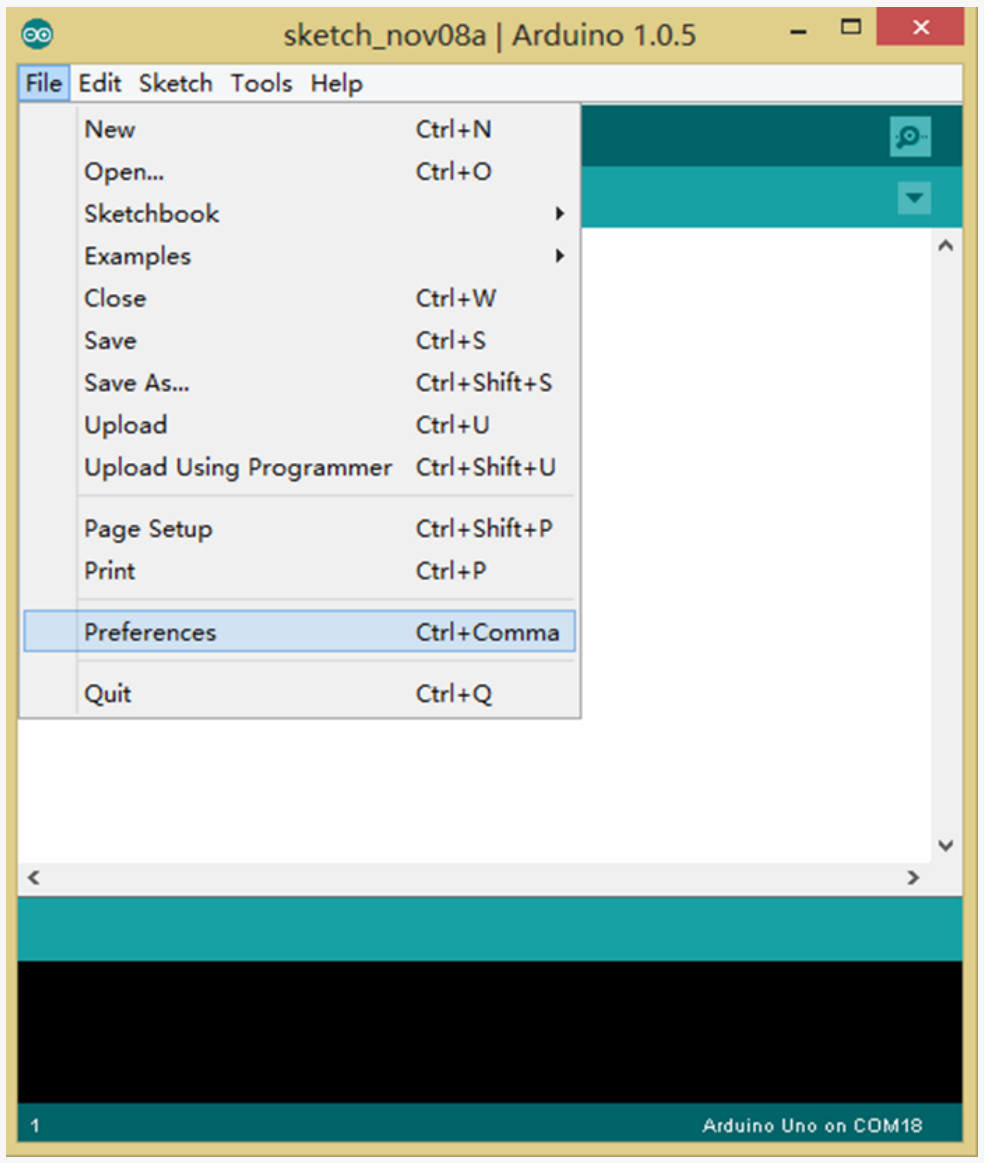


3.Introduction to Arduino IDE

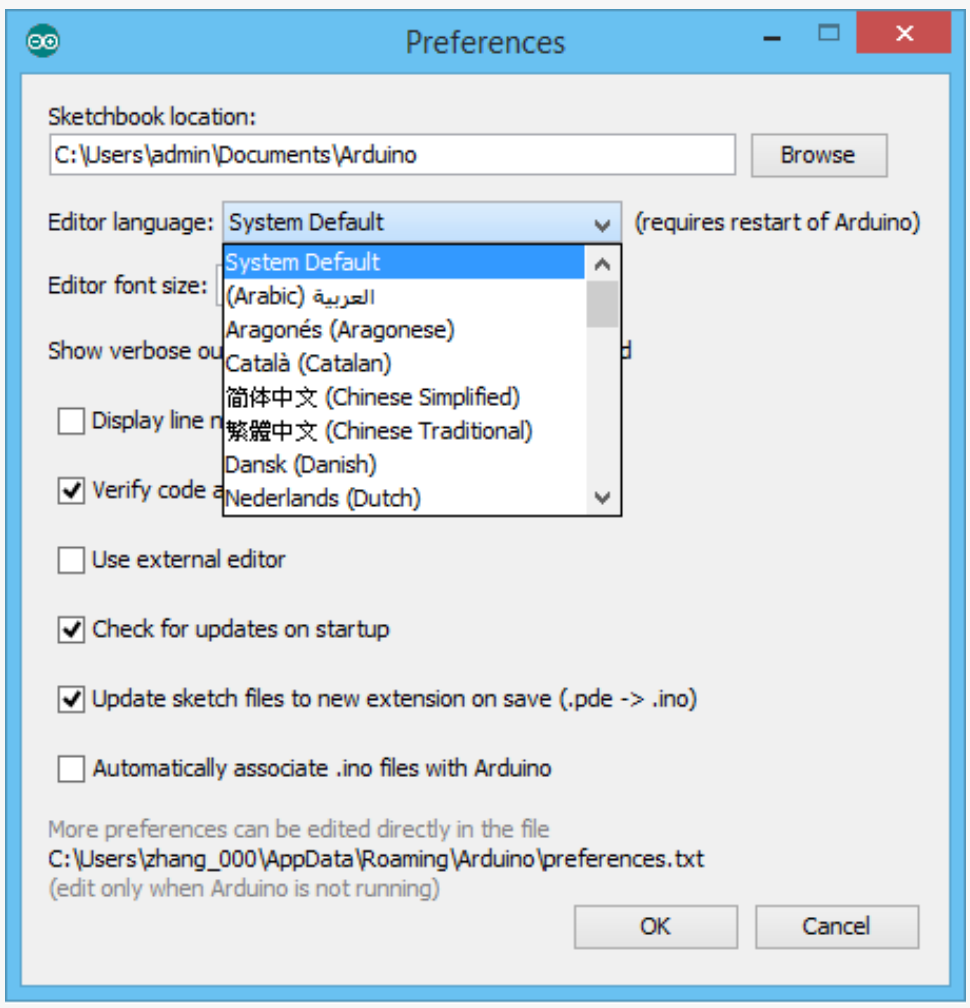
Inside your Arduino directory, open “Arduino.exe”. The application will open and the code editing interface will appear.



If you wish to change the language of the interface, select “File” and then “Preferences” to open preferences.

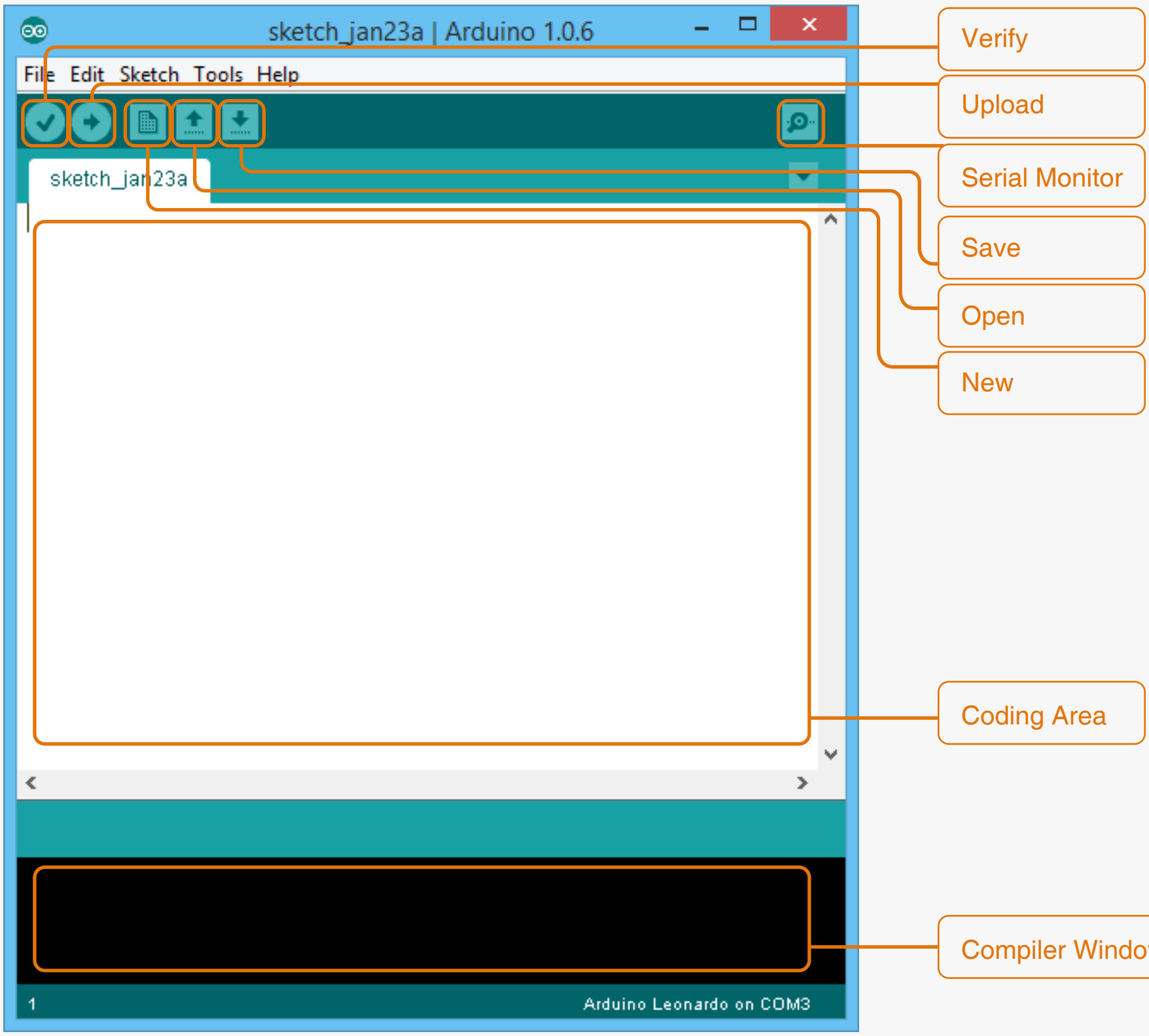


The dialog box shown below will pop up. Select “Editor Language”, choose your language and then click OK.



Close Arduino IDE and then reopen it for the changes to take effect.

We will use Arduino IDE a lot in these tutorials, so lets learn about its features:

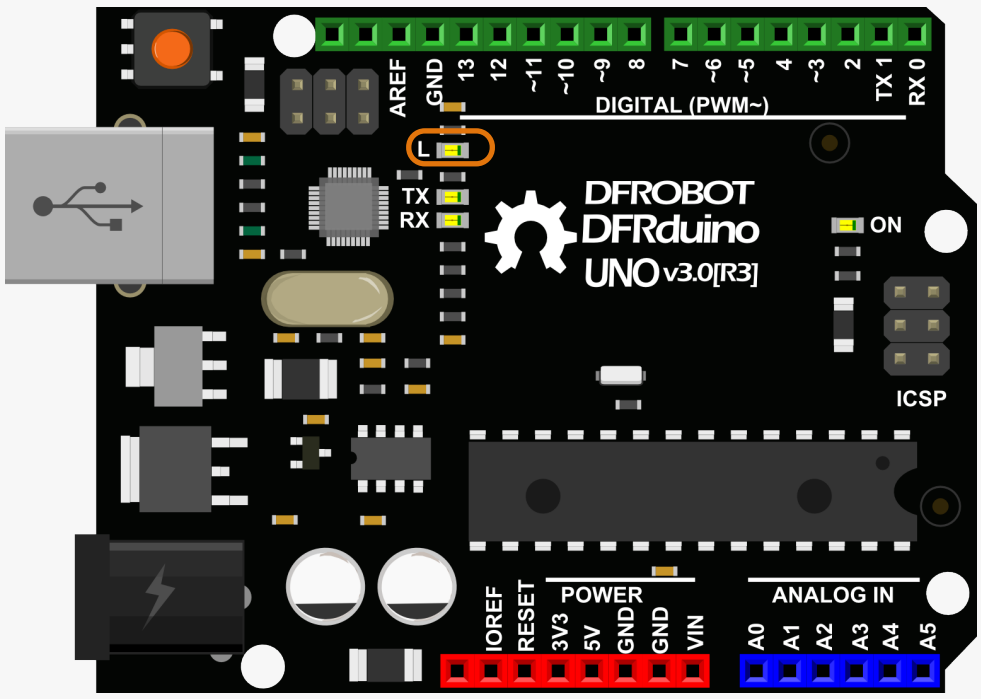


Arduino IDE allows you to edit and upload programs to your microcontroller. Arduino IDE calls programs “sketches”, but in these tutorials, we will simply say “program” or “code”, rather than “sketch”, but they really mean the same thing.

The interface shown in the picture above is where you’ll write and upload code within Arduino IDE. The main white area is where you input code. We use the C programming language to program the microcontroller, which we will discuss in more detail in later chapters. When you press “verify” or “upload” , the code instructions you have written will be translated in to machine language by a piece of software called a “compiler” so that the microcontroller can understand it. This process is called “compiling” or “verifying”. Messages shown in the black area will show information about the compiling and upload process.

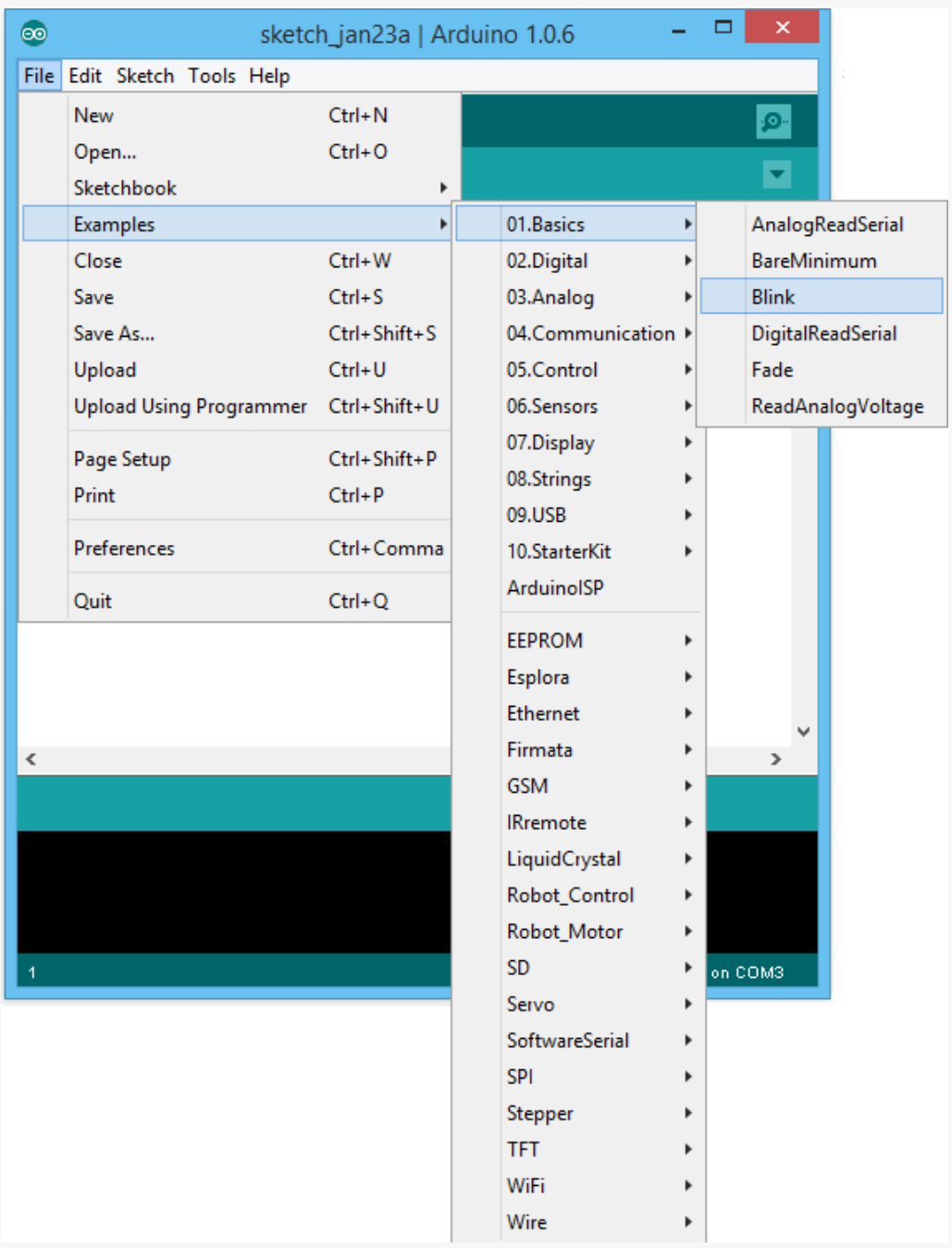
4. Upload a Blink Program

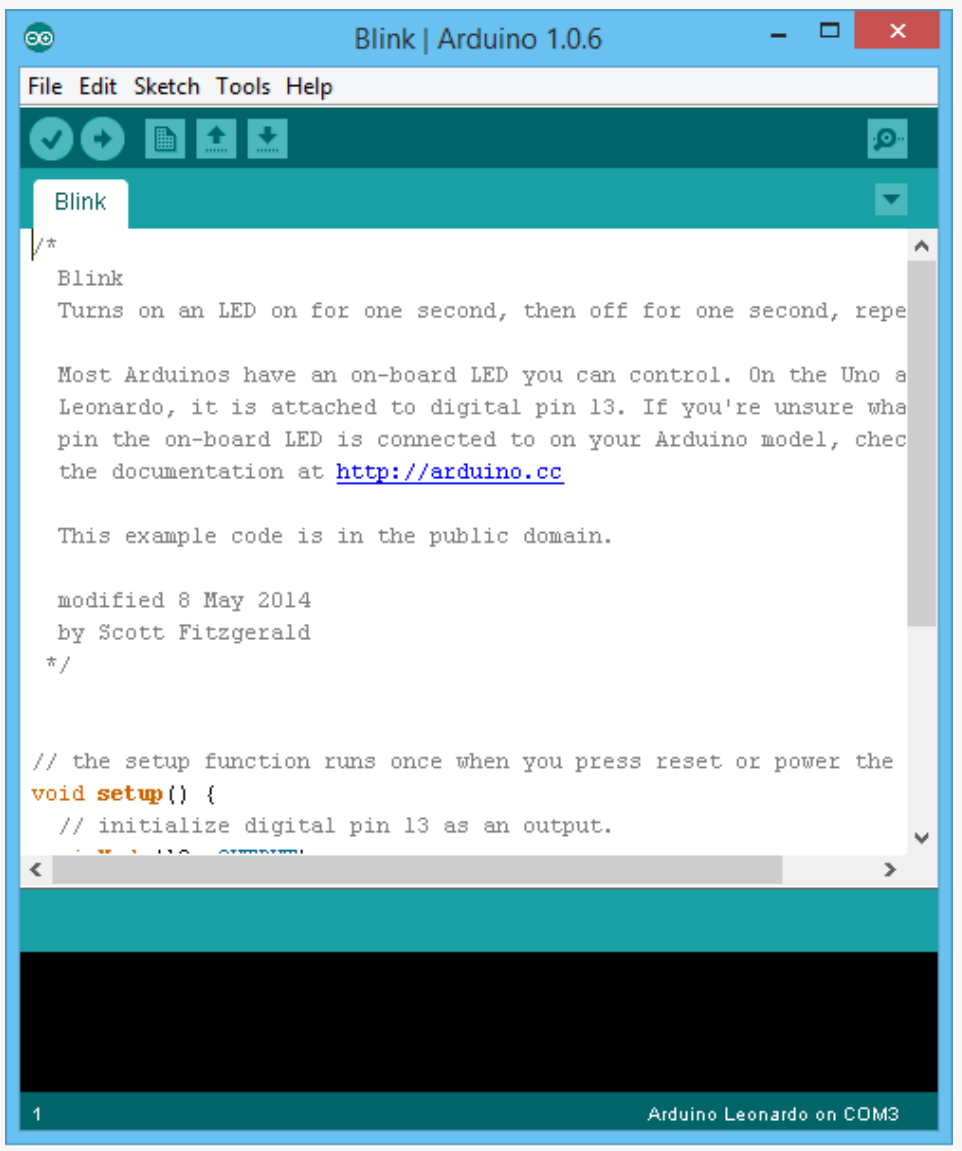
Let’s upload a simple program to become familiar with the uploading process and to test the microcontroller. This code will turn the “L” LED light on the microcontroller on and off. The location of the “L” LED is circled in the diagram below.



Connect your computer to the microcontroller via the USB.

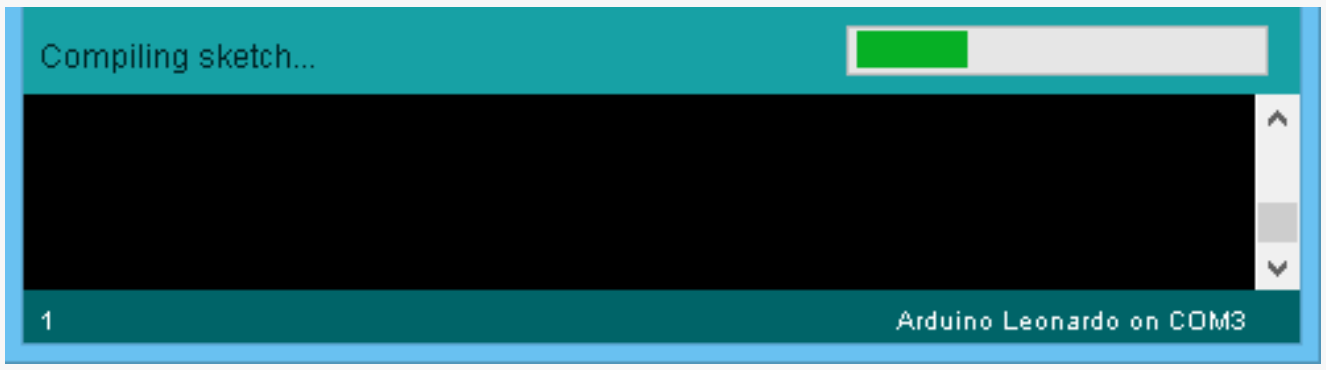
In the Arduino IDE, go to `File > Examples > 01. Basics > Blink`.



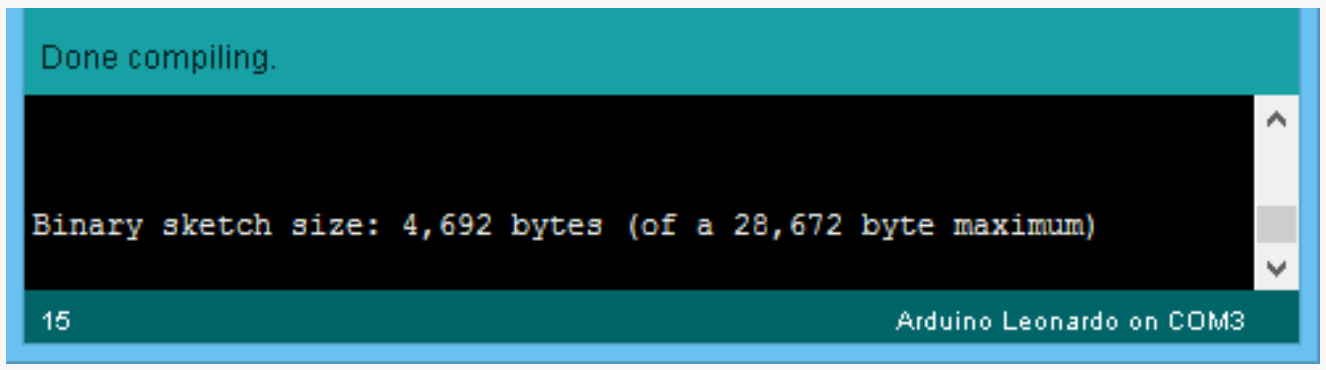


As this is an example program, there will be no syntax errors in the code. When we write code ourselves we can click “Verify” to check if there are any syntax errors. Try it now.

The code is run through the compiler. This green bar shows the compiling process.



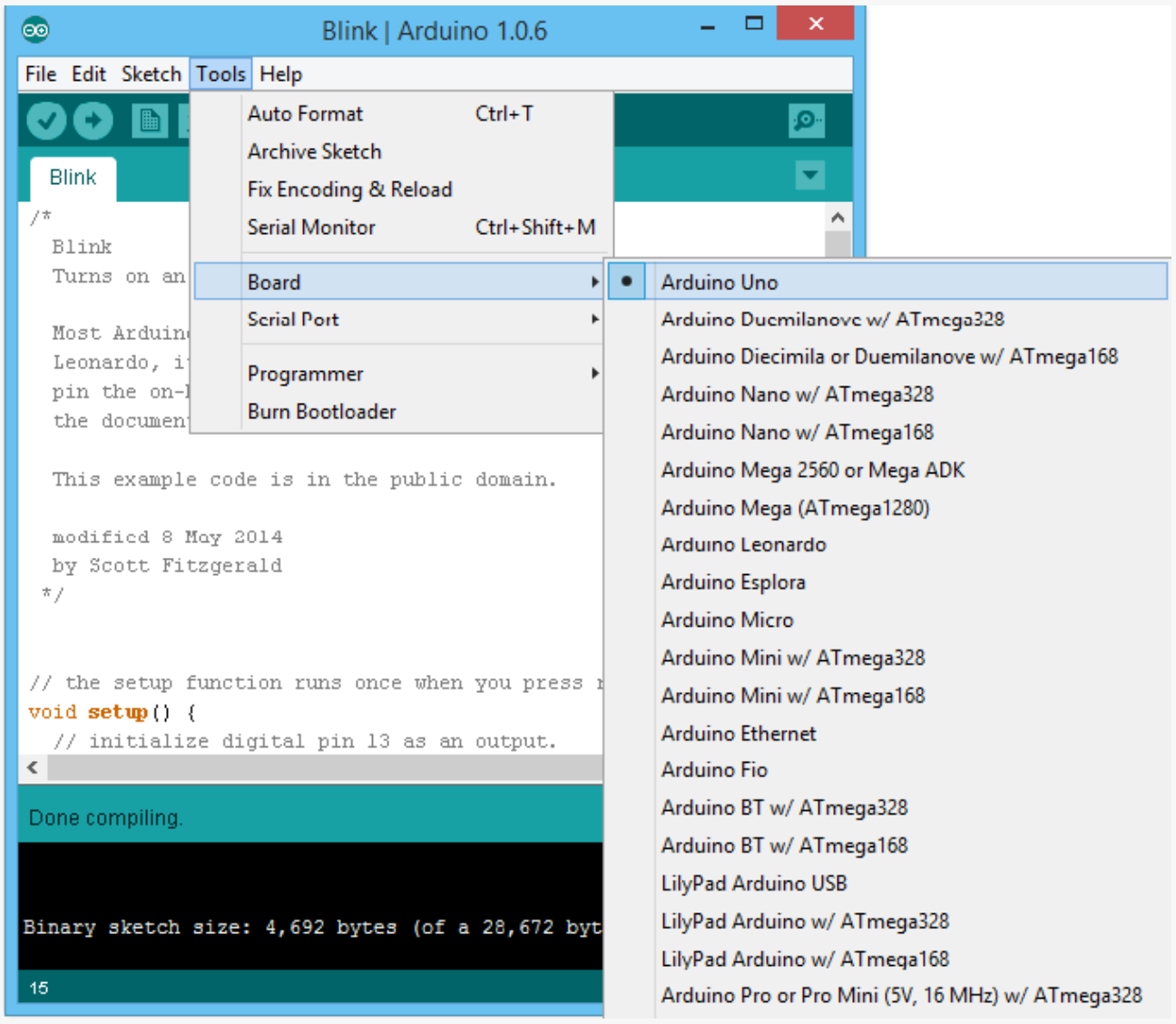
Hopefully you will see this message:



Before code is uploaded to the microcontroller, the IDE will automatically verify it.
If there are any errors in your code, a big ugly orange message box will appear above the compiler window indicating a problem, and your code can not upload until it is fixed!

Before we can upload a program, we need to tell the computer where to send it. In Arduino IDE, we need to select which type of Arduino board we are using as well as the serial port that the board is connected to.

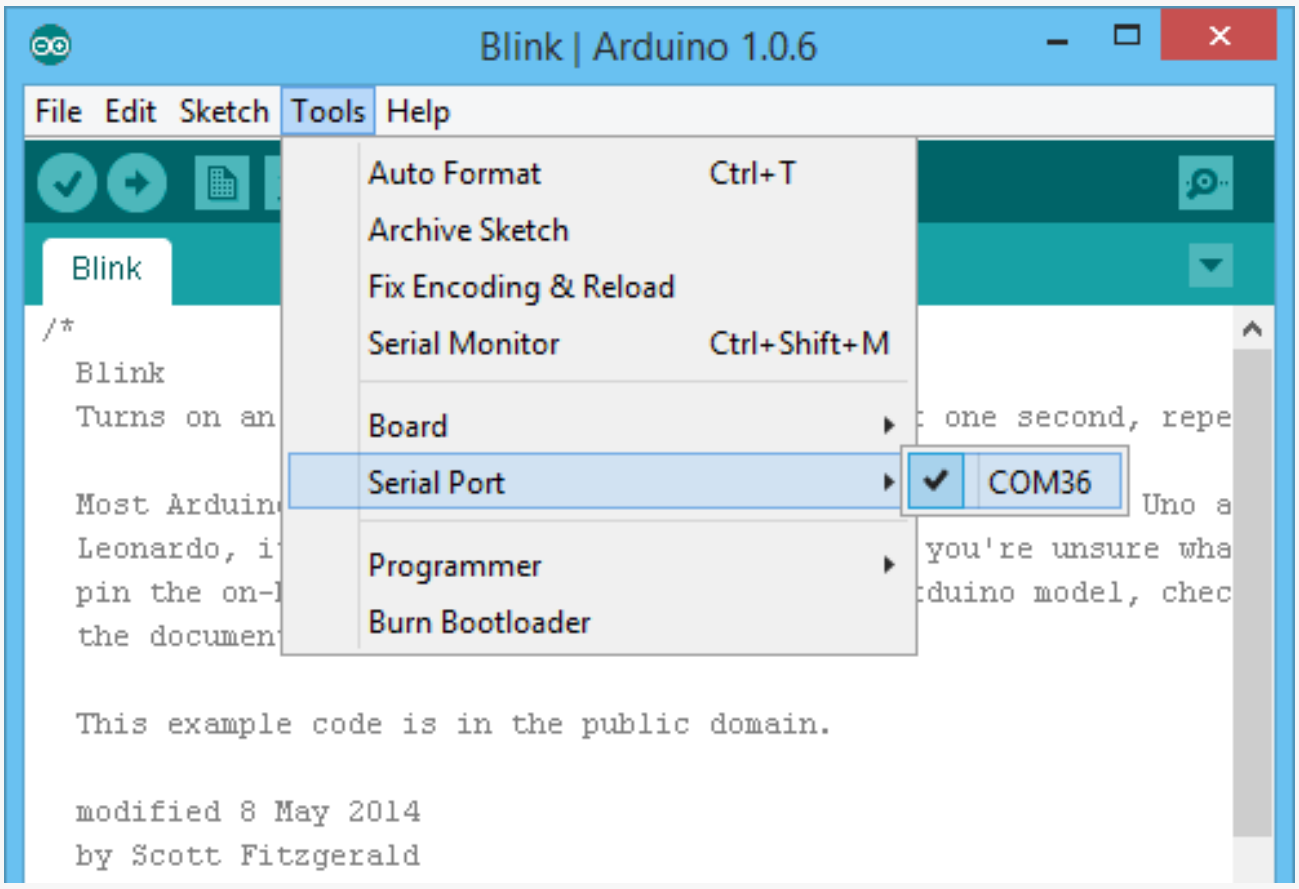
Navigate to `Tools > Board` and then select Arduino UNO from the list. That's it!



(There are many different types of Arduino board of all shapes and sizes. Some have more RAM and more I/O pins and can hold larger programs. We are using an Arduino Uno variant, so we must select Arduino Uno from the boards list.)

Now select the serial (COM) port. Navigate to `Tools > Port` and your available serial ports will be listed. Usually this will be "COM" followed by a number. Select the available port.

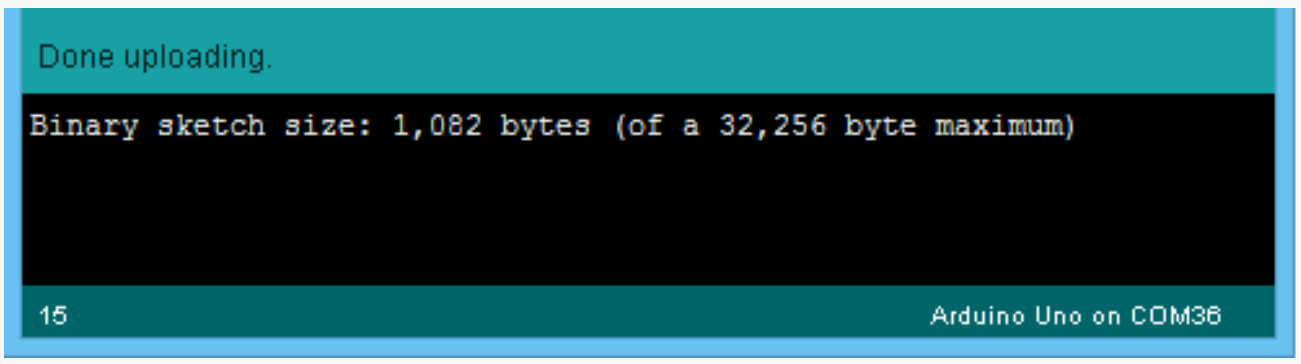
In our example the board is connected to the computer on COM36.



We should now be able to upload code to the microcontroller.

Click upload in Arduino IDE. If all is well, the code will verify and then upload to the microcontroller.

When the upload has completed successfully, the microcontroller will automatically reset and the code will execute. Is the light blinking?



This concludes the blink tutorial.

To review, there are three fundamental steps to upload code to your microcontroller:

1. Verify code
2. Select board and COM port
3. Upload!

If you have any questions or comments about this tutorial, or any other tutorials in this guide, feel free to contact us on the DFRobot forum:
http://www.dfrobot.com/forum/

The DFRobot forum is a community where hackers and makers can share their exciting ideas. Come and visit to get ideas about what you can make with your DFRduino microcontroller, or just share what you've done following these tutorials.
We love your feedback!

Lesson 2

How Does a Device “Think”?

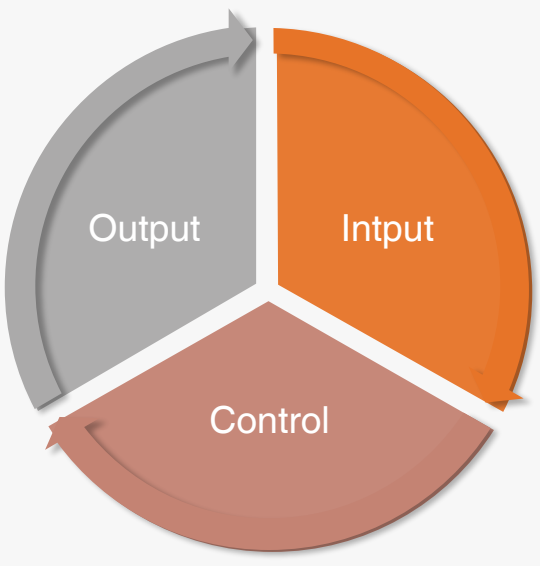
02

DFROBOT
DRIVE THE FUTURE

Simple Automatic Control Devices

Any device that includes a microcontroller can be considered a simple automatic control device. There are three essential constituent parts for a simple automatic control device:

- An **input** unit for collecting signals
- An **output** unit for sending out signals
- A **control** unit for processing the signals received (such as a microcontroller).



Let’s draw an analogy between an automatic control device and a person. In a person, signals are sensed through sight, hearing, taste, touch and smell, and then processed by the brain which then outputs responses to each. In this case, the output signal would be the actions of that person.

A microcontroller can output responses as well. Output responses include sound, light and movement (DC Motors, LEDs, servos etc.)

Here is another way to explain it: imagine someone says “hello” to you, and you immediately reply “hello” in return. In this case, your ear is the input unit; your brain is the control unit and your mouth is the output unit.

How can we carry out this whole process (receiving signals, processing signals and then sending out signals) using our microcontroller?

As well as our microcontroller, we simply need an acoustic sensor and a buzzer. The acoustic sensor “hears” a sound, the microcontroller receives a signal and then outputs a signal to the buzzer for it to buzz. In this case the acoustic sensor is the input unit; the microcontroller is the control unit and the buzzer is the output unit.

Think!

Can you tell which components in the kit can be used as input units and which can be used as output units?

Input Units - Sensors

A sensor (also known as a transducer) is a physical unit whose purpose is to sense or detect the characteristics of its environment (such as light, temperature or moisture levels) and then transmit this data to another device.

Control Unit - DFRduino Uno

The microcontroller is the control unit. Think of it as the brain of your device.

Sensor Pins

The three categories of sensor pins are as follows:

- Digital Pin
- Analog Pin
- Protocol Pin (Digital)

A protocol pin is also a kind of digital pin. I2C, Serial and SPI are frequently used digital pins.

Output Units - Actuators

There are many different types of actuators. An actuator is a type of device that is responsible for moving or controlling a system or mechanism. It is also the mechanism by which a control system acts upon an environment. It may convert electrical energy into motion, sound or light. A buzzer or speaker are actuators that output sound.

The Relationship Between Programs and Hardware

The input unit, control unit and output unit mentioned above are all hardware. In the context of our person analogy, hardware is the body of our device. However, the brain is much more important as it produces ideas and then controls actions every person takes. Code here functions as the mind of a person. Both body and mind are indispensable to a person.

The Difference between Digital & Analog Signals

Digital Signals:

A digital signal has two states: `HIGH` or `LOW`.

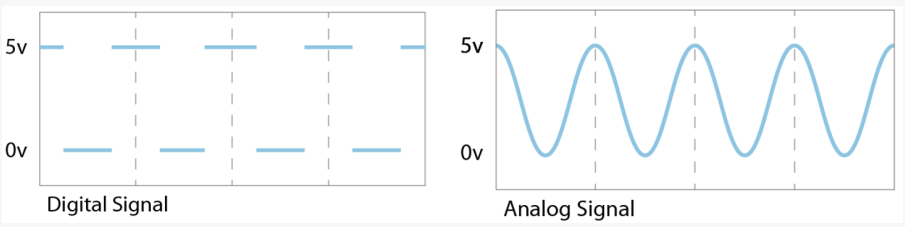
`HIGH` is a 5V signal and represents “1” (or on).

`LOW` is a 0V signal and represents 0 (or off).

Analog Signals:

An analog signal has a range of values.

The analog pins of your microcontroller can have between 0V and 5V is mapped to a range between 0 and 1023. For instance, 0 is mapped as 0V; 1023 is mapped as 5V and 512 is mapped as 2.5V.



Digital Signals & Analog Signals in the World of Electronics

The input unit, the microcontroller and the output unit communicate by signals, which in turn are processed by code. How do input units and controllers communicate with each other? How do controllers communicate with the output units? To answer the above questions, we first need to understand two concepts: digital signals and analog signals.

“Digital” & “Analog” in a DFRobot Kit

Here are two ways to tell whether sensors in your DFRobot Kit are digital or analog:

(1) If a sensor has a green wire it uses digital signal; if a sensor has a blue wire it uses analog signal.

(2) “A” or “D” may be marked on the sensor’s board. “D” represents “digital” and “A” represents “analog”.

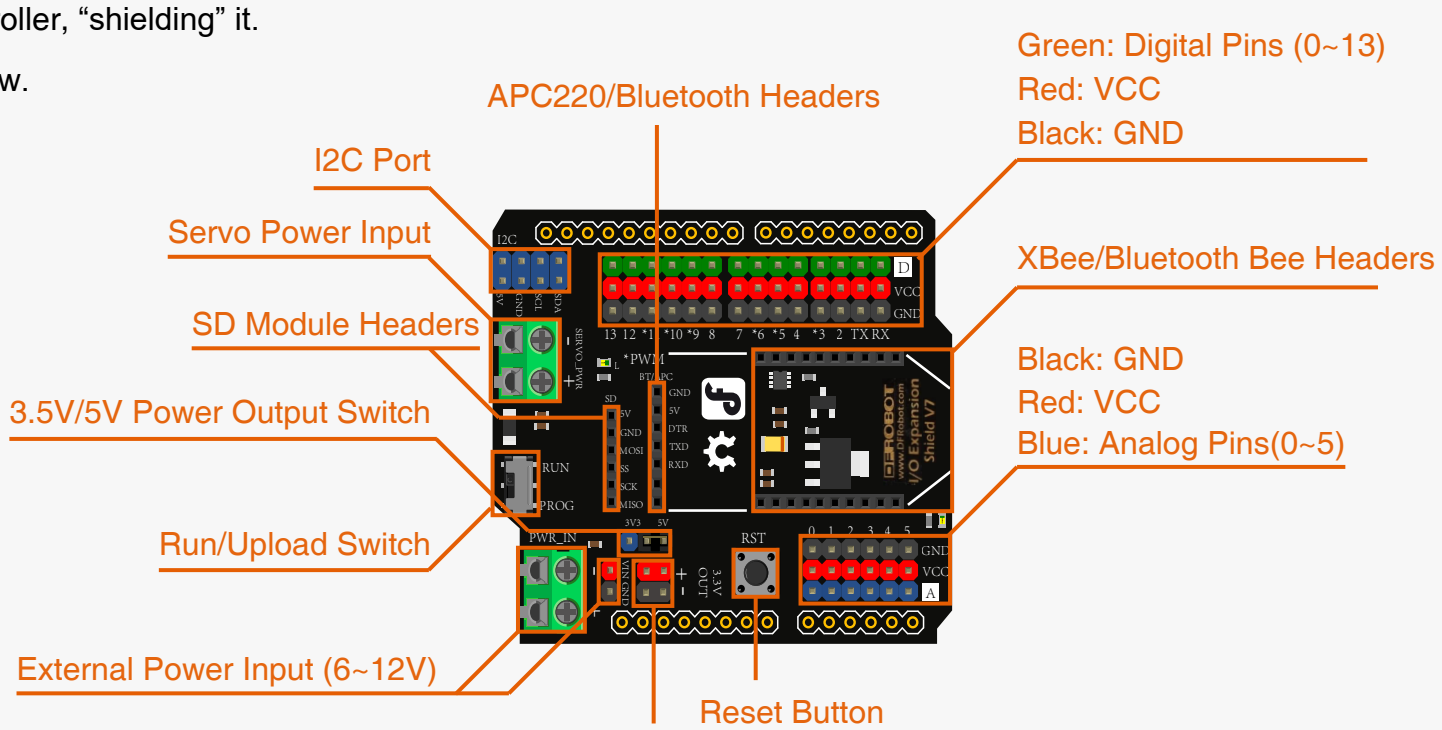
I/O Expansion Shield V7.1

Let’s take a look at the DFRobot I/O Expansion Shield.

This shield gives us lots of extra pins to connect sensors to.

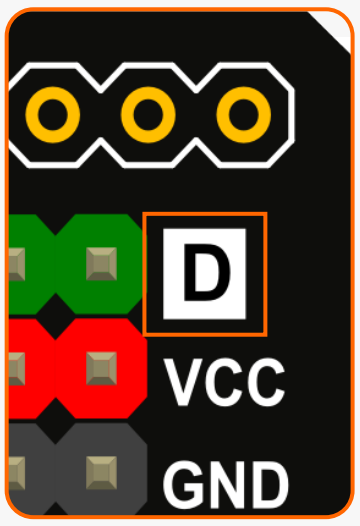
It stacks neatly on top of your microcontroller, “shielding” it.

The diagram below gives you an overview.



Analog and digital pins are marked on the expansion shield.

The area with “A” is for analog sensors, and the area with “D” is for digital ones.



The advantage of the I/O expansion shield is that there are more power and ground pins than those on the bare microcontroller board. This gives you enough power pins to connect various sensors at the same time.

On the expansion shield, there are a row of power pins in red and a row of GND pins in black below the digital pins.

Different colors shown in our DF Kit have different meanings:

Green = Digital signal

Blue = Analog signal

Red = Power

Black = Ground

If you would like to know more about the sensor expansion shield, please refer to the DFRobot wiki for information:

http://www.dfrobot.com/wiki/index.php/IO_Expansion_Shield_for_Arduino_V7_SKU:DFR0265

This session gives you a general idea of what makes our devices work.
Are you excited to get your devices going? You’ll get a chance to in the next lesson!

Lesson 3

Analog and Digital Signals

03

DFROBOT
DRIVE THE FUTURE

Let's begin!

Let’s draw an analogy between microcontrollers and people

The microcontroller is the “brain”.

Code is the microcontroller’s “mind”.

Hardware (sensors and actuators) are the microcontroller’s “body”.

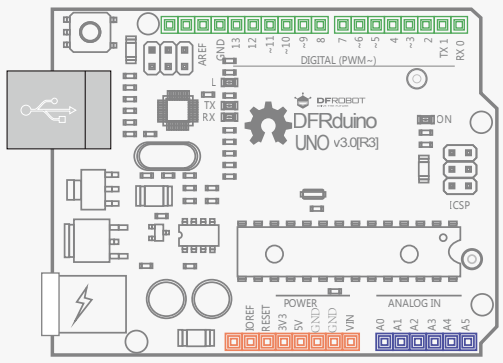
Signals, such as analog and digital signals, are the microcontroller’s “nerve impulses”.

In this session, we will explore the differences between analog and digital signals in more detail.

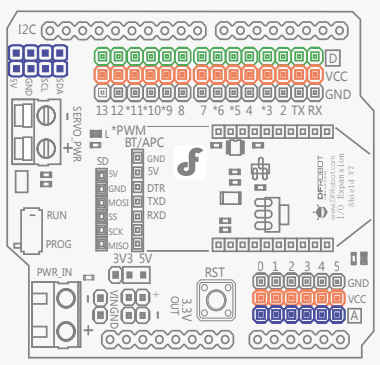
Digital Signals

We will use a digital button module to demonstrate a digital signal. This is included in your kit.

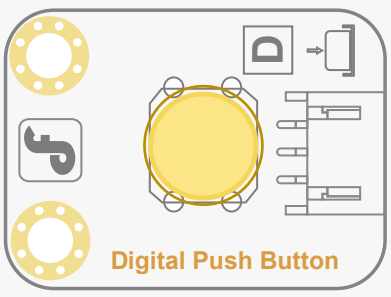
Parts Needed:



DFRduino Uno
(and USB cable)
x1



I/O Sensor Expansion
Shield V7.1
x1



Digital Push
Button Module
x1

Connections

Take the Sensor Expansion Shield and stack it on top of your microcontroller – make sure that the pins are correctly aligned to avoid damage to the shield.

Take the Digital Push Button module and connect it to Digital Pin 2 (be sure that the power, ground and signal connections are correct or you risk damaging your components!)

See the diagram below for further details:

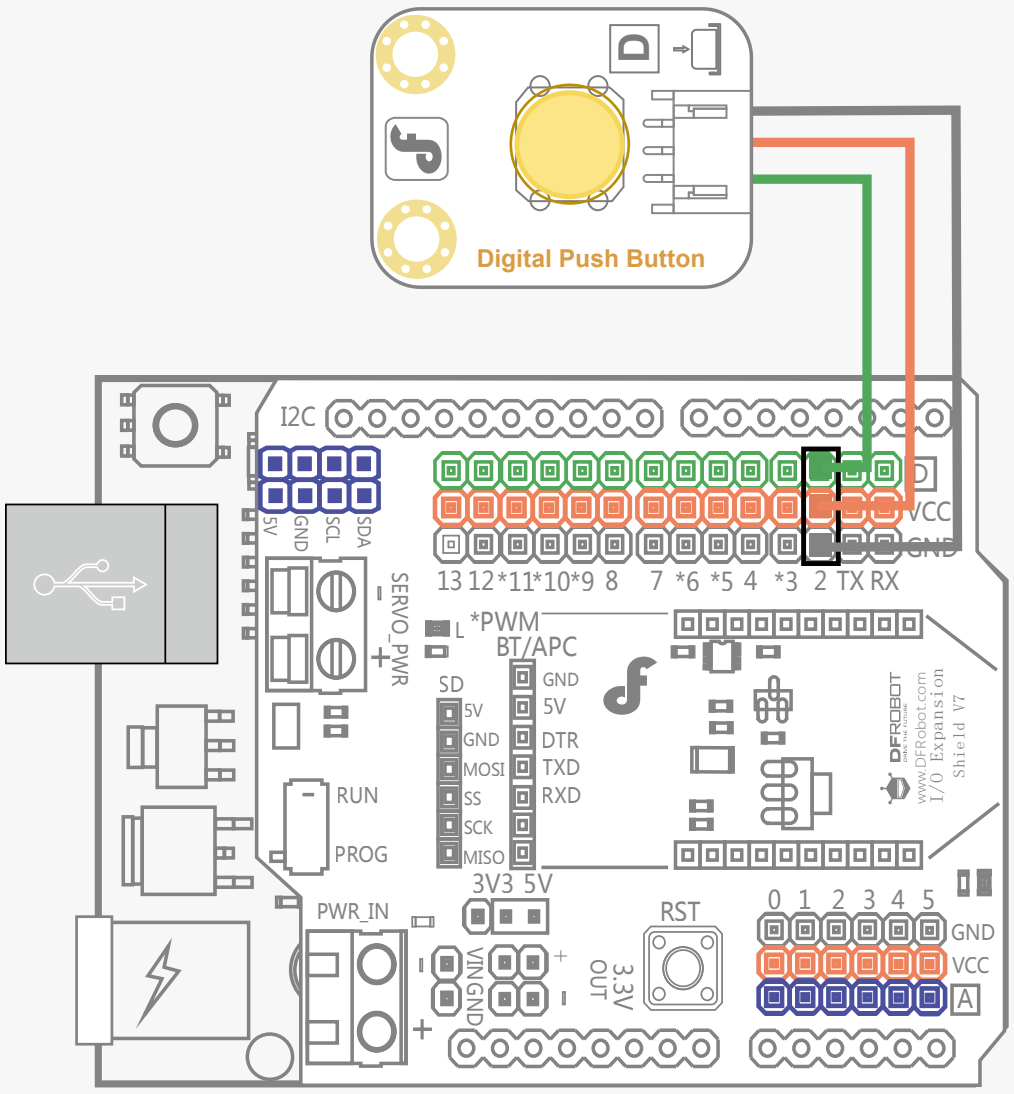


Fig. 2-1 Digital Pin Connection

When the connections are made, connect the USB cable. This will power on the microcontroller. We can now prepare our program to upload.

Serial Port Monitoring

Open Arduino IDE and select: `File > Examples > 01. Basics > DigitalReadSerial`

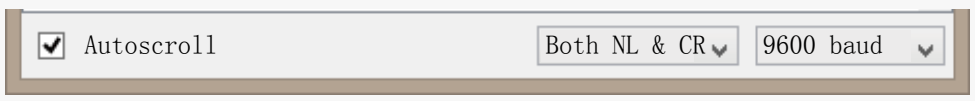
The following code should appear:

```
int pushButton = 2;                //connect to digital pin 2
void setup() {                      // initial function
  Serial.begin(9600);               // set up baud rate of the serial port
  pinMode(pushButton, INPUT);       // set the button to be in the output mode
}
void loop() {                                  //main function
  int buttonState = digitalRead(pushButton);     // record statistics about the status of digital pin 2
  Serial.println(buttonState);    // Serial port print statistics about the status of digital pin 2
  delay(1);                                    // delay 1ms
}
```

Click “Upload”. Arduino IDE will verify the code and then upload it to your microcontroller.

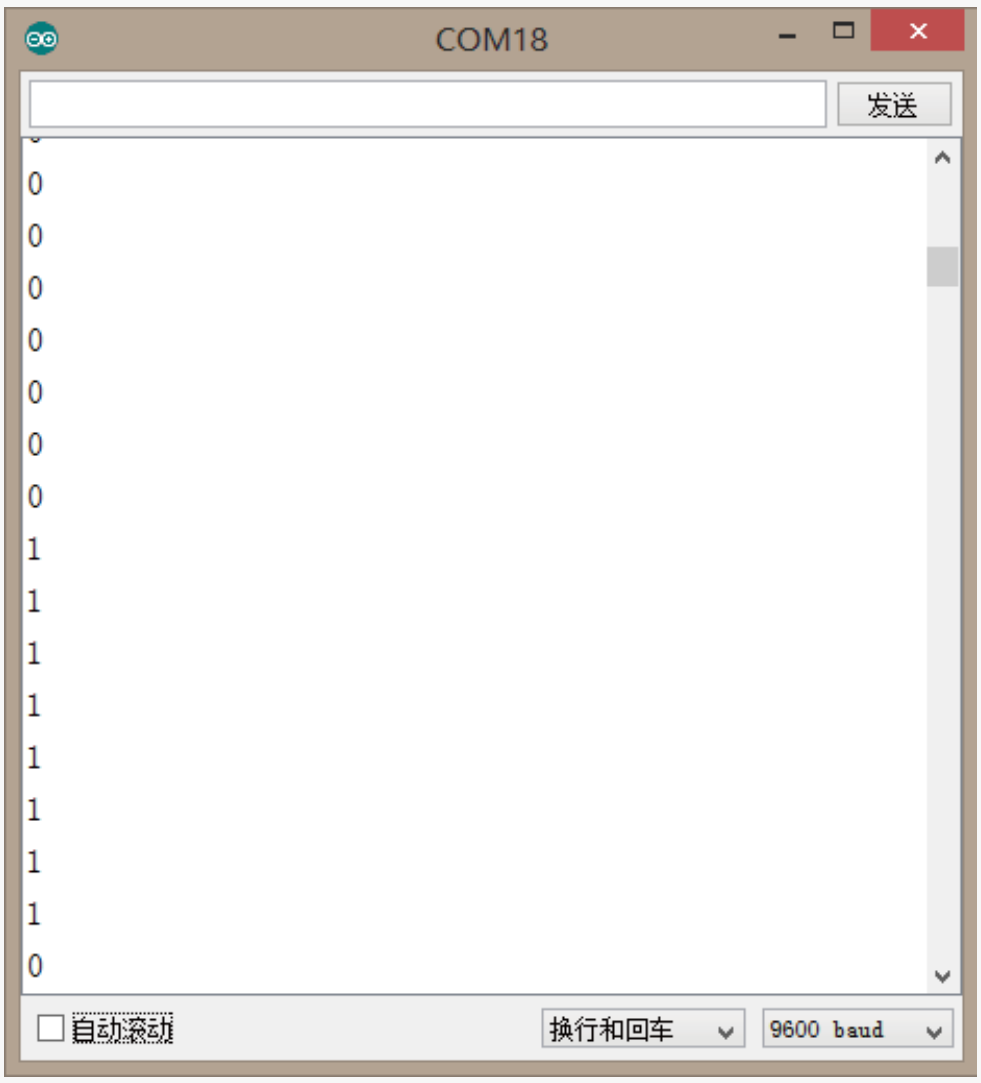
Once the upload is complete, open the serial monitor. To do this, navigate to `Tools > Serial Monitor`, or click the magnifying glass icon on the top right hand corner of the toolbar.

The baud rate is the data speed that the serial port communicates. It needs to be set to `9600 baud`.



The serial monitor shows information from the microcontroller. We are going to use it to view the status of the button. You should see “0” appearing continuously. This indicates that the button is off (not being pressed).

If you press the button, you will see “1” appear for as long as the button is pressed. This indicates that the button is on (being pressed).

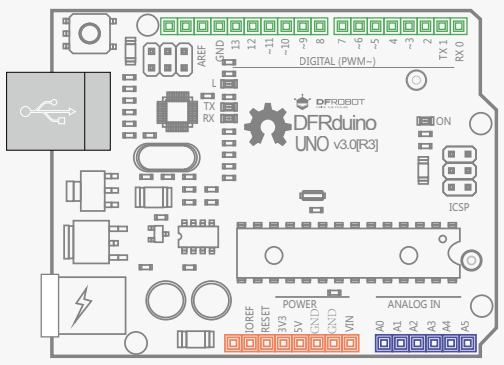


“0” and “1” or off and on are the two values of digital signal. We can use this in various ways with our devices.

Analog Signal

In this section we are going to explore analog signal in more detail. We are going to use a sensor which uses analog values: a rotation sensor.

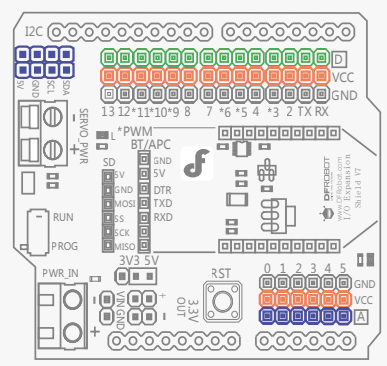
Parts Needed:



x1

DFRduino Uno

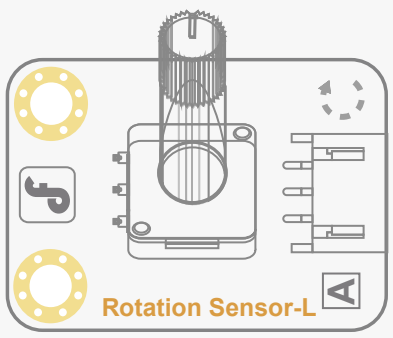
(and USB cable)



x1

I/O Sensor Expansion

Shield V7.1



x1

Analog Rotation Sensor

Connections

Connect the rotation sensor to Analog Pin 0. Make sure that signal, ground and power are connected to the correct corresponding pins, just like in the previous session. Connect the USB cable to the microcontroller to power it on. We can now upload a program.

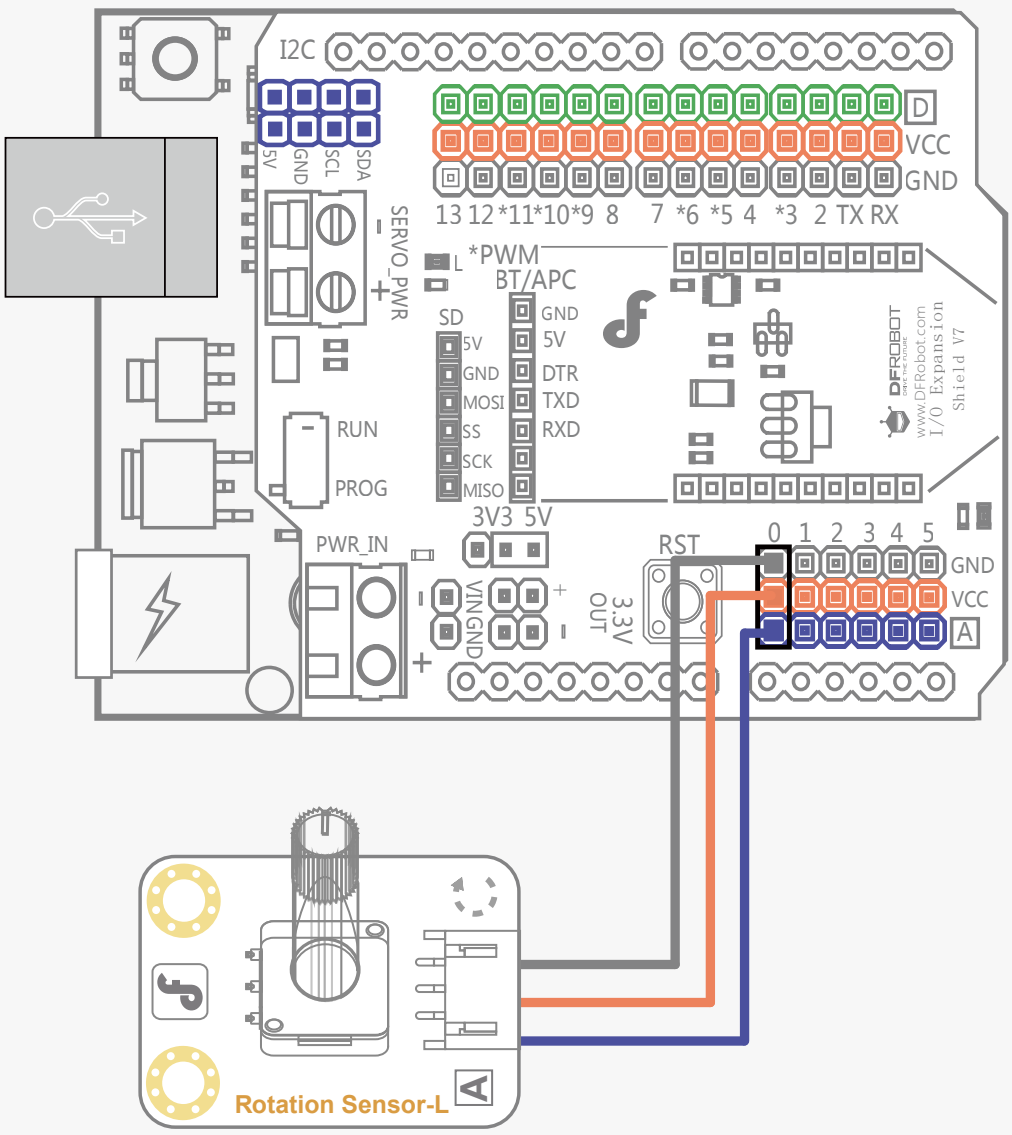


Fig. 2-2 Analog Pin Connection

Code Input

In Arduino IDE, navigate to `File > Examples > 01. Basics > AnalogReadSerial`

A new window with a program will appear. It should read as follows:

```
void setup() {  //Initial setup

  Serial.begin(9600); //Set baud rate

}

void loop() { //Main function

  int sensorValue = analogRead(A0);     // Read status of analog pin 0.

  Serial.println(sensorValue);          // Print status of pin 0 in Serial Monitor
  delay(1);//Delay 1ms

               }
```

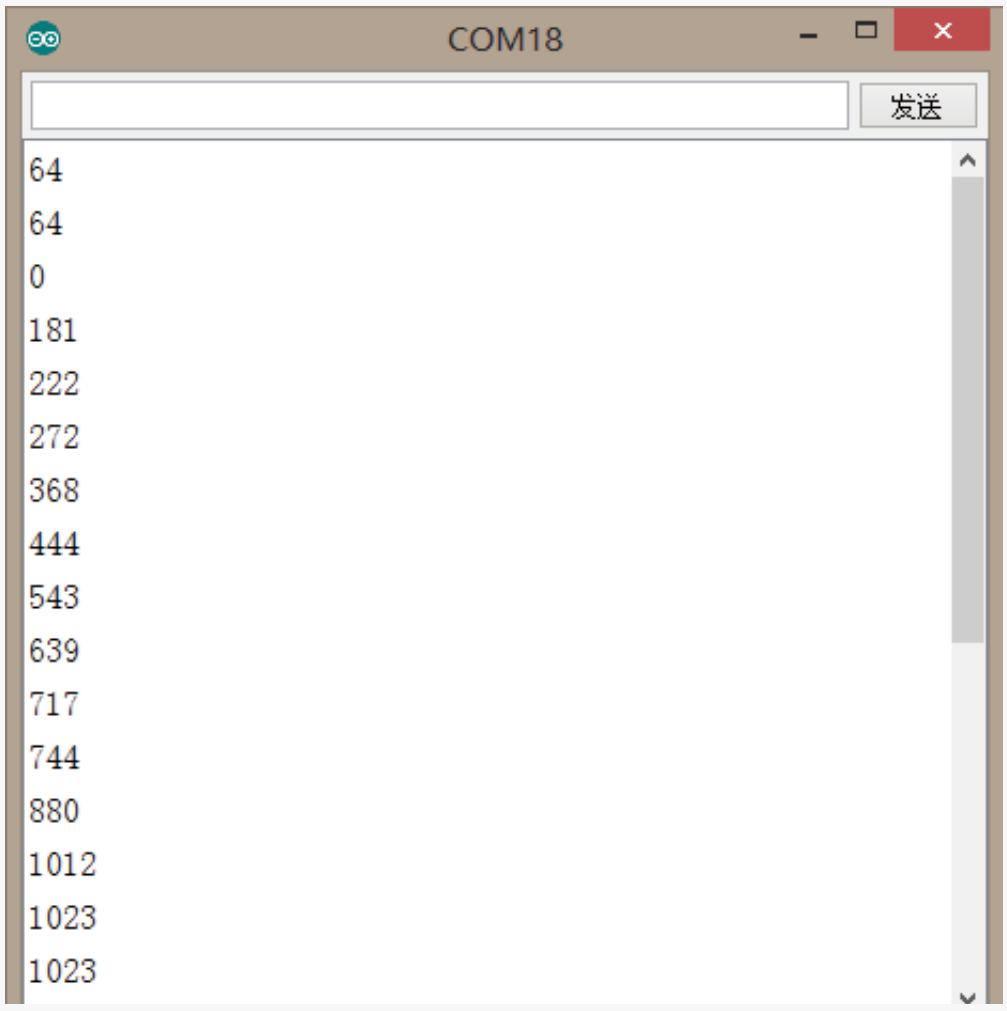
Click “Upload”. Arduino IDE will verify the code and then upload it to the microcontroller.

Once the upload is complete, open Arduino IDE’s serial port monitor. To do this, navigate to `Tools > Serial Monitor`

Set baud rate of the serial port to be `9600 baud`.

A series of values will flash in a column in the serial monitor.

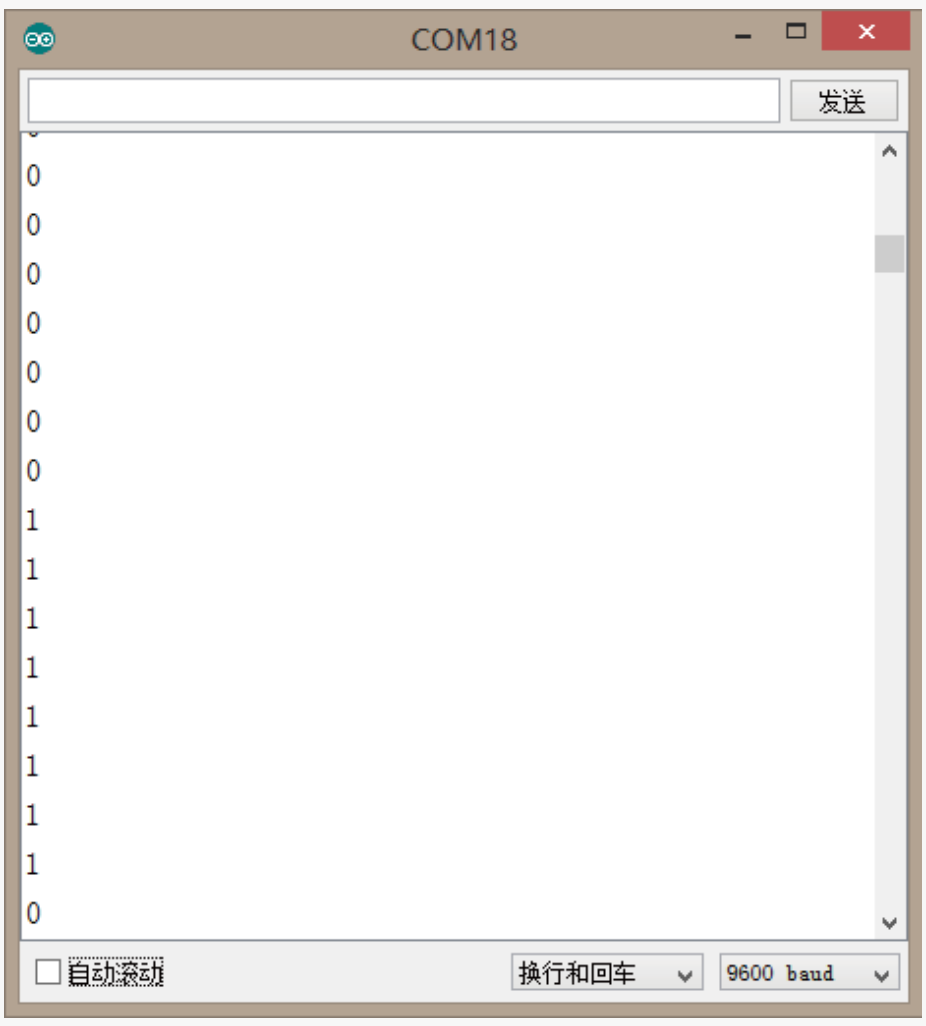
Try to turn the rotational sensor. A figure between 0 and 1023 will show up, depending on how far the rotational sensor’s shaft is rotated. 0 means no rotation and 1023 means a full rotation.

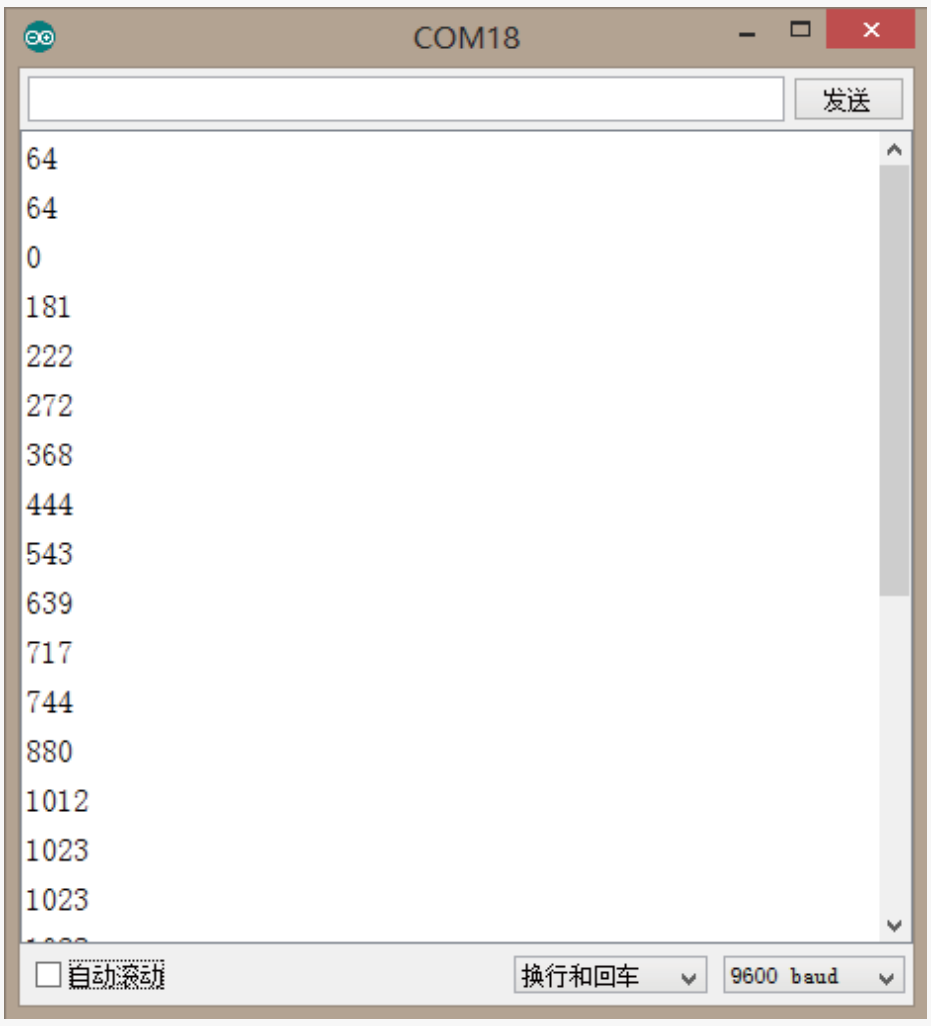


The Difference Between Digital Pins And Analog Pins

Serial Port Monitor

We use the Serial Monitor to view information from the microcontroller. We can also use it to send information back to the microcontroller, and interact in with it in various ways while it runs a program.





Code Differences

If you examine the code in detail, you can see that digital pins and analog pins are read using different commands.

To read the value of a digital pin, we use the command: `digitalRead()`

To read the value of an analog pin, we use the command: `analogRead()`

Digital Pins:

```
int buttonState = digitalRead(pushButton); // read values from digital pin2
```

Analog Pins:

```
int sensorValue = analogRead(A0); //read values from analog pin0
```

This will be explained in more detail in the next few sessions.

Try It Yourself

The sensors in your DFRobot kit connect in much the same way as the ones demonstrated. Just make sure your power, ground and signal connections are correct. Once connected, you can check the serial monitor for output and see what values they output. Try to experiment and have fun!

Lesson 4

Make an LED Blink

DFROBOT
DRIVE THE FUTURE

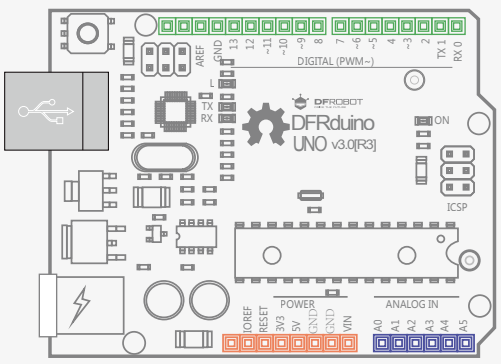
04

Make an LED Blink

We've given you a brief idea of what Arduino is and how it works, as well as some theory behind digital and analog signals. Now let's start making some real stuff. We will start with the basics: how to make an LED blink.

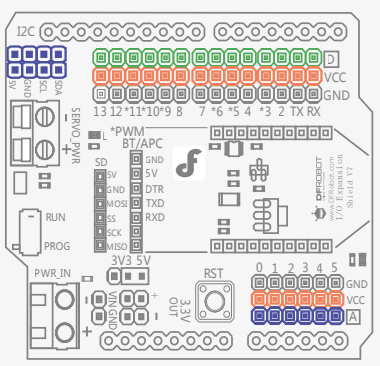
In the previous session, we've used the blink program to test the microcontroller. This time we'll attach our own LED to digital pin 13 instead of using the microcontroller's integrated LED.

Parts Needed:



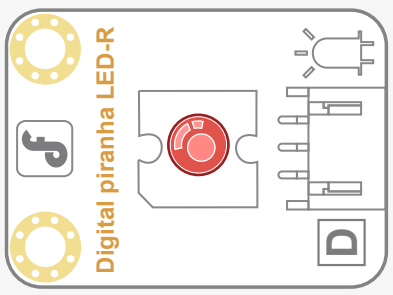
x1

DFRduino UNO
(with USB cable)



x1

I/O Expansion Shield V7.1



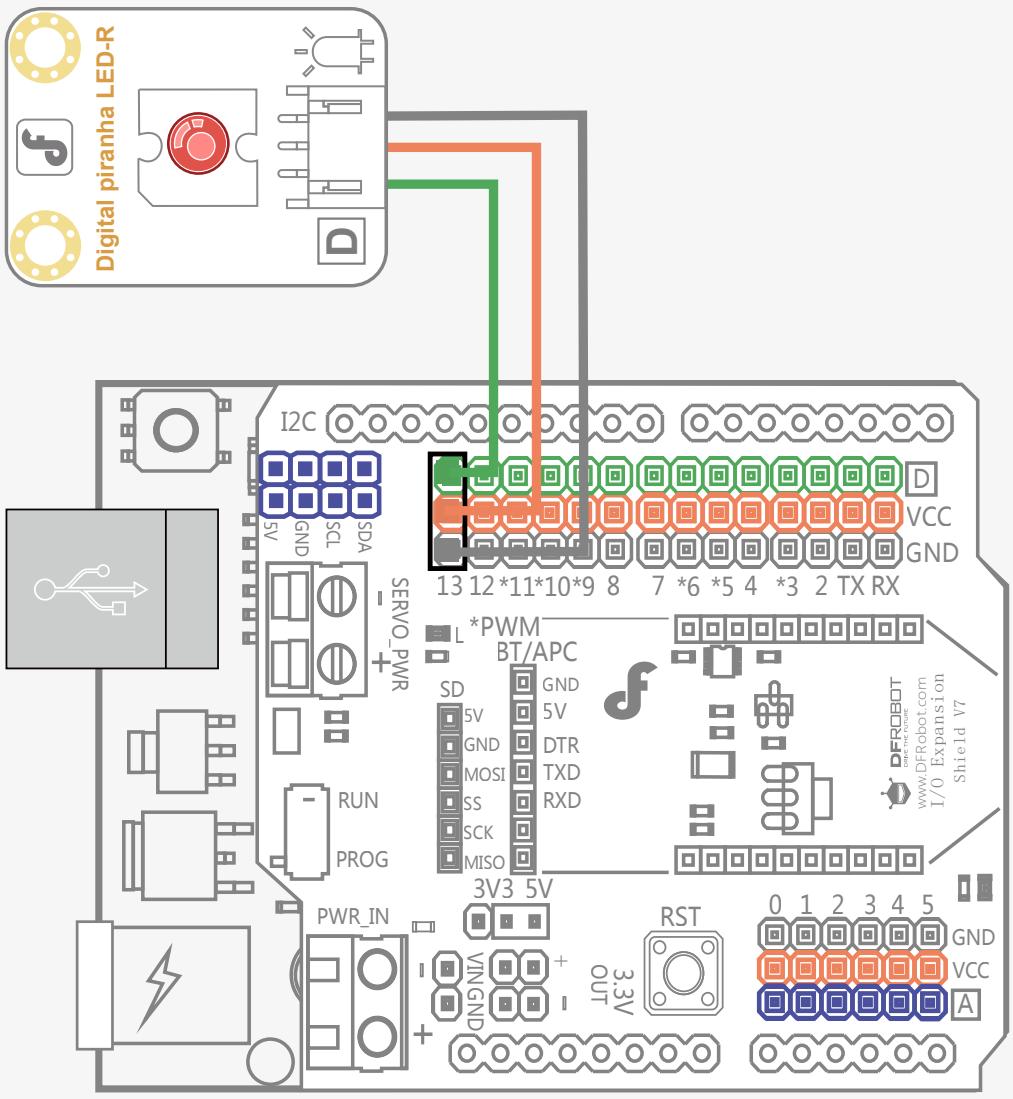
x1

Digital Piranha LED-R

Connections

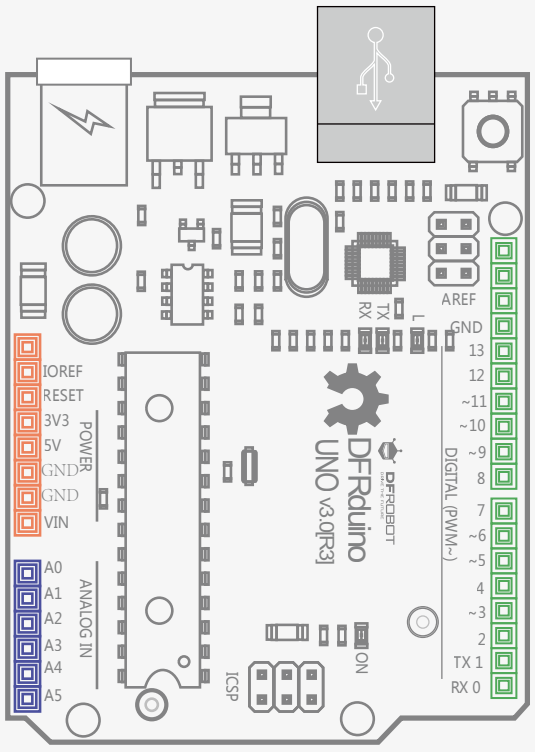
Connect the Digital Piranha LED-R module to digital pin 13 on the microcontroller. Make sure your power, ground and signal connections are correct or you risk damaging your components.

When all your connections are made, connect the microcontroller to your computer via the USB cable.

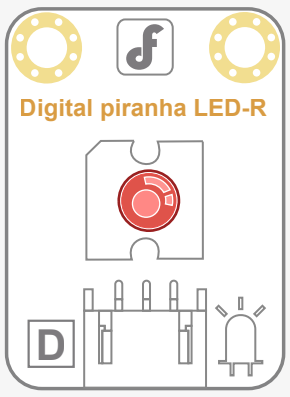


Hardware Analysis (Digital Output)

In the previous sessions, we've mentioned inputs and outputs. The device we will make is composed of two parts: a control and an output. The microcontroller is the control unit and the digital Piranha LED-R module is the output unit. No input unit is included in this particular device.



Control Device



Output Device

Code Input

Open Arduino IDE.

You can open the program by navigating to `File > Examples > 01. Basics > Blink`

(Although we encourage you to type the code out by hand to develop your skills).

The code is displayed below:

Sample Code 1-1:

```
// Item One —— LED Blink
//The LED will turn on for one second and then turn off for one second

int ledPin = 13;
void setup() {
  pinMode(ledPin, OUTPUT);
}
void loop() {
  digitalWrite(ledPin, HIGH);
  delay(1000);
  digitalWrite(ledPin, LOW);
  delay(1000);
}
```

Once the code is input and you are happy with it, click “Verify” in Arduino IDE. The IDE will check the code for errors. If there are no errors, you are ready to upload. Click “upload” and Arduino IDE will upload the code to your microcontroller. When the upload is complete, your external LED will blink!

Now let’s review the program and see how it works:

Code Analysis

A program is composed of two main parts:

```
void setup() {
```

This part of the code is executed when the device is initialized. It runs only once. It is used to declare outputs and inputs of the device.

```
void loop() {
```

This part of the code is executed after `void setup(){`.
Once initialized, it runs in a loop forever (or until the device is powered off or a component fails). This is where all the key arguments of the code are contained.

Consider this code:

```
pinMode( pin, mode)
```

The code above shows that `ledPin` is set up as the output mode. The comma in the brackets is the code syntax and needs to be there to separate the parameters.

```
pinMode( ledPin, OUTPUT);
```

The code above shows that `ledPin` is set up as the output mode. The comma in the brackets is the code syntax and needs to be there to separate the parameters.

What is `ledPin`?
Examine the first line of the code below:

```
int ledPin = 13;
```

Here we have assigned pin13 the variable name "`ledPin`". In other words `ledPin` represents pin 13 on the microcontroller
"`int`" stands for integer and means that `ledPin` is an integer data type.

What if you want to attach the LED to pin10 instead of pin13?

```
pinMode( 10, OUTPUT);
```

You just have to change the pin value in the code from 13 to 10.

Let's take a look at the `loop()` function, which contains only one function: `digitalWrite()`.

The syntax of the function `digitalWrite()` is as follows:

```
digitalWrite(pin,value)
```

When we set pinMode() as OUTPUT, the voltage will be either be `HIGH` (5V or 3.3V for the control board) or `LOW` (0V).

```
digitalWrite(ledPin,HIGH);
//LED on
digitalWrite(ledPin,LOW);
//LED off
```

When `HIGH` is included in `digitalWrite()` as shown above, pin 13 will be `HIGH` and the LED will turn on. When `LOW` is included in `digitalWrite()` as shown above, pin 13 will be LOW and the LED will turn off.

Don't forget the following line:

```
delay(1000);
```

The number in brackets means the amount of milliseconds of delay. There are 1000 milliseconds in 1 second, so this means the duration of delay is 1 second. The LED will turn on for 1 second and then turn off for 1 second continuously.

You might notice "`//`" and "`/*...*/`" located ahead of the code:

```
// Item One—LED Blink
    / *
Description: LED on and off
alternately every other sec-
ond
 */
```

Anything written after "`//`" or inside "`/*...*/`" will not be compiled. These are code annotations. We use these to write comments about the code in plain English. "//" will comment for one line of code. "/*...*/" will comment for multiple lines. Arduino IDE will help us recognise annotations by turning the text grey, and the compiler will ignore the text when code is uploaded. There are no limits for how many annotations you can make on a program.

Exercise

Try to make the LED light blink faster or slower. For example, keep the LED light off for 5 seconds and then quickly blink for 250 milliseconds

Lesson 5

Sensor Light

DFROBOT
DRIVE THE FUTURE

05

Sensor Light

In this session, we will create a sensor light

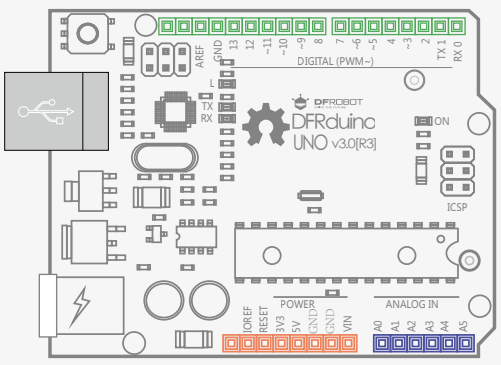
The behavior we want to implement is as follows:

When the sensor detects motion, the LED light will turn on

When no motion is detected, the LED light will turn off.

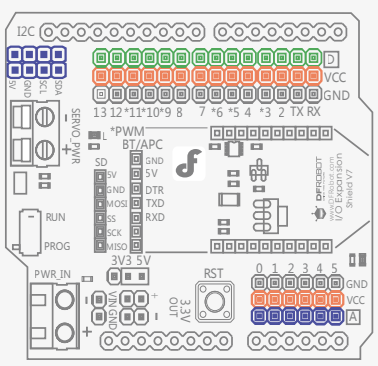
To detect motion, we can use an IR sensor that detects infra-red light.

Parts Needed:



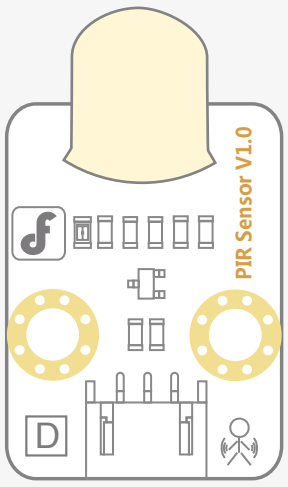
x1

DFRduino UNO
(with USB cable)



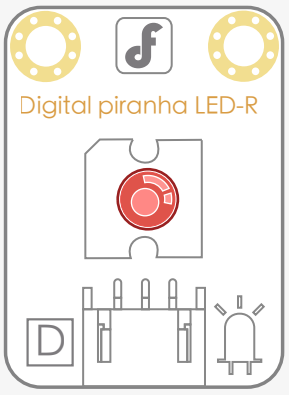
x1

I/O Expansion Shield V7.1



x1

PIR Sensor



x1

Digital Piranha LED-R

Connections

We need to make the following connections:

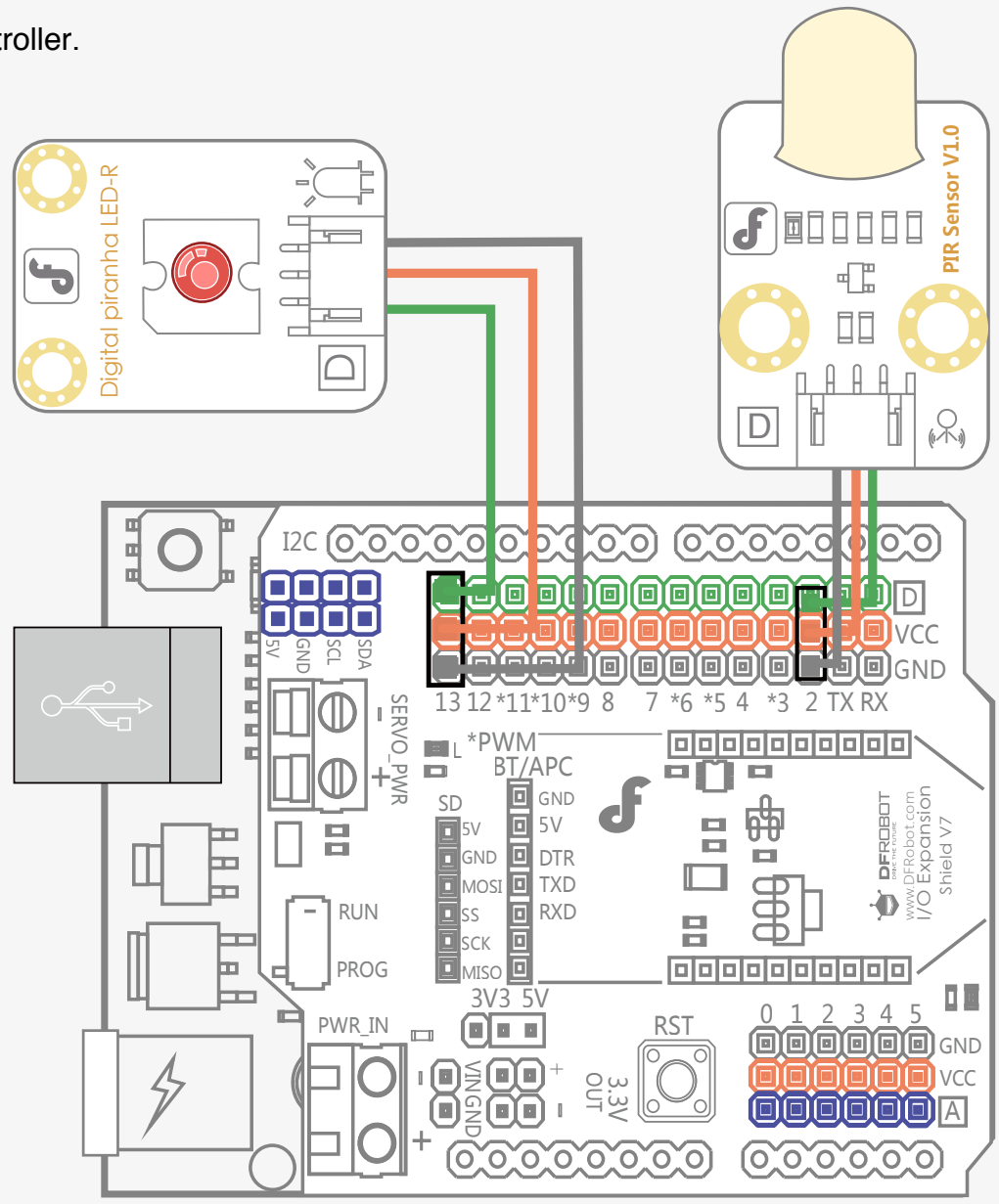
Connect the IR sensor to digital pin 2

Connect the Digital Piranha LED-R to digital pin 13

Refer to the diagram below for further details.

Be sure your power, ground and signal connections are correct or you risk damaging your components.

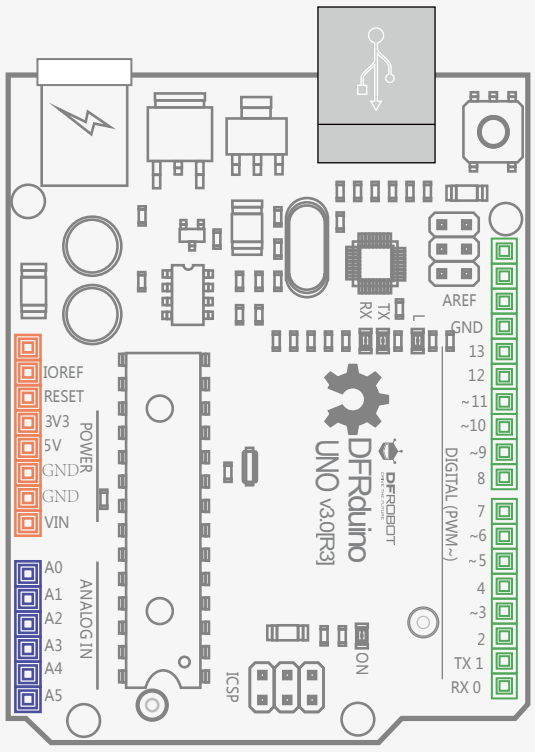
When your connections are complete, you can move to Arduino IDE to upload code to the microcontroller.



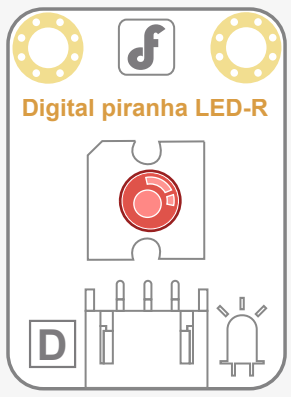
Hardware Analysis

This device is composed of three parts: an input unit, a control unit and an output unit. The IR sensor is the input unit; the microcontroller is the control unit and the Digital Piranha LED-R is the output unit.

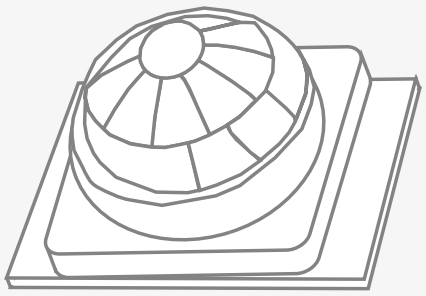
The IR sensor and LED use digital values, so both should be attached to a digital pin.



Control Device



Output Device



Input Device

Code Input

```
//Sample Code 2-1:Item Two—Sensor Light

int sensorPin = 2;              // IR sensor to be attached to digital pin NO.2

int ledPin =  13;                // Digital Piranha LED-R to be attached to digital pin NO.13

int sensorState = 0;            // variable sensorState for storing statistics about the status of the sensor

void setup() {
  pinMode(ledPin, OUTPUT);          // LED is the output unit
  pinMode(sensorPin, INPUT);       // Sensor is the input unit
}

void loop() {
  sensorState = digitalRead(sensorPin);    // Read statistics collected from the sensor

  if (sensorState == HIGH) {        // If it is HIGH, LED light will be turned on
    digitalWrite(ledPin, HIGH);
  }
  else {                                 // If it is LOW, LED light will be turned off
    digitalWrite(ledPin, LOW);
  }
}
```

Use this sample code to implement the behavior we want.

You can copy and paste it in to Arduino IDE, but if you want to develop your skills we recommend typing it out.

When you have finished, click “Verify” to check the code for syntax errors. If the code verifies successfully, you can upload it to your microcontroller.

When the sensor detects motion, the LED should turn on.

When no motion is detected, the LED should turn off.

Code Analysis

The IR sensor is the input unit; the LED is the output unit. They are initialized in the `setup()` function.

```
pinMode(ledPin, OUTPUT);            // Define LED as the output unit
pinMode(sensorPin, INPUT);          // Define sensor as the input unit
```

We can read values from digital pins using the function `digitalRead()`. This can be written in the `loop()` function.

```
sensorState = digitalRead(sensorPin);
```

The function is as follows:

```
digitalRead(pin)
```

This function is used to read the states of digital pins, either `HIGH` or `LOW`. When the IR sensor detects motion, the state is `HIGH`.

The next part of the code will determine actions based on the state of the sensor. (Remember: `HIGH` means 1; `LOW` means 0).

The function is as follows:

`digitalRead(pin)`

This function is used to read the states of digital pins, either `HIGH` or `LOW`. When the IR sensor detects motion, the state is `HIGH`.

The next part of the code will determine actions based on the state of the sensor.

The digital sensor can only read two values (`HIGH` or `LOW`).

We'll use the `if` statement here:

The `if` statement is usually used with a comparison operator, such as == (equal to), ! = (not equal to), < (less than), or > (greater than). Using one of these comparison operators allows the program to evaluate an expression, and then make a logical decision. Usually we use it to test if a particular condition has been reached.

Let's examine the application of the `if` statement in our code:

```
  if (sensorState == HIGH) {
    digitalWrite(ledPin, HIGH);
  }
  else{
    digitalWrite(ledPin, LOW);

}
}
```

In this case, the if statement evaluates whether the sensor state is `HIGH` or `LOW`. If it is `HIGH`, it will turn the LED `HIGH` (on) as well.

`else` is used to give an alternative output if the output condition in the statement is not met.

If the sensor is not `HIGH`, the LED is `LOW` (off).

We briefly touched upon comparison operators above. Let's examine them in more detail:

- `x==y` (x is equal to y)
- `x! =y` (x is not equal to y)
- `x<y` (x is less than y)
- `x>y` (x is greater than y)
- `x<=y` (x is less than or equal to y)
- `x>=y` (x is greater than or equal to y)

When we use these in code, the microcontroller will make an evaluation between values and then make a decision.

EXERCISE

Make a model of a ghostly face with flashing LEDs inside it. You can change the interval of the LED flashing to make them quicker or slower for different effects. It would be great for a haunted house!

Lesson 6

Mini Lamp

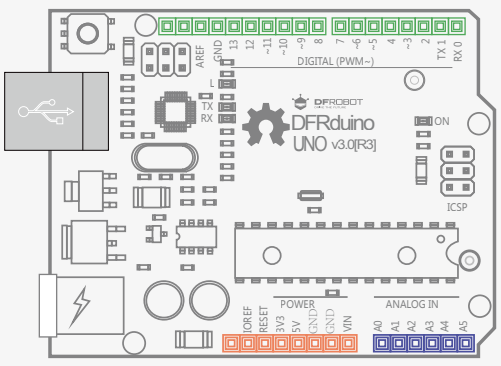
06

DFROBOT
DRIVE THE FUTURE

Mini Lamp

A lamp is something we use on a daily basis, usually made up of a light source and a simple switch. Let’s make our own.

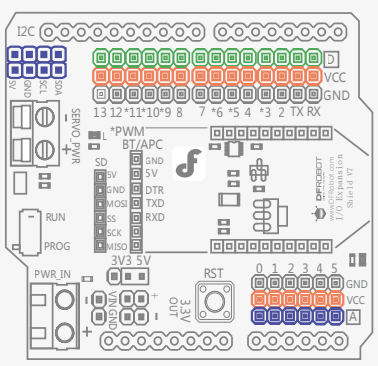
Parts Needed:



x1

DFRduino Uno

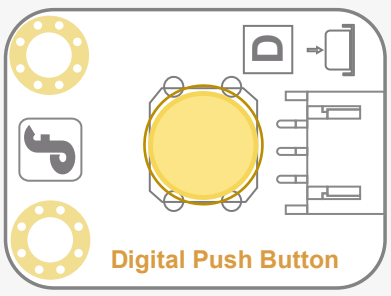
(with USB cable)



x1

I/O Sensor Expansion

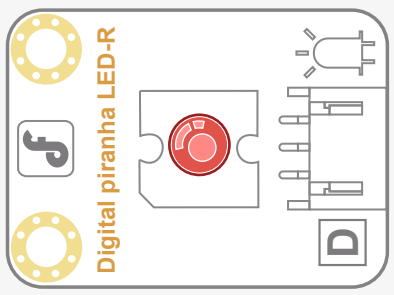
Shield V7.1



x1

Digital Push

Button Module



x1

Digital Piranha

LED-R

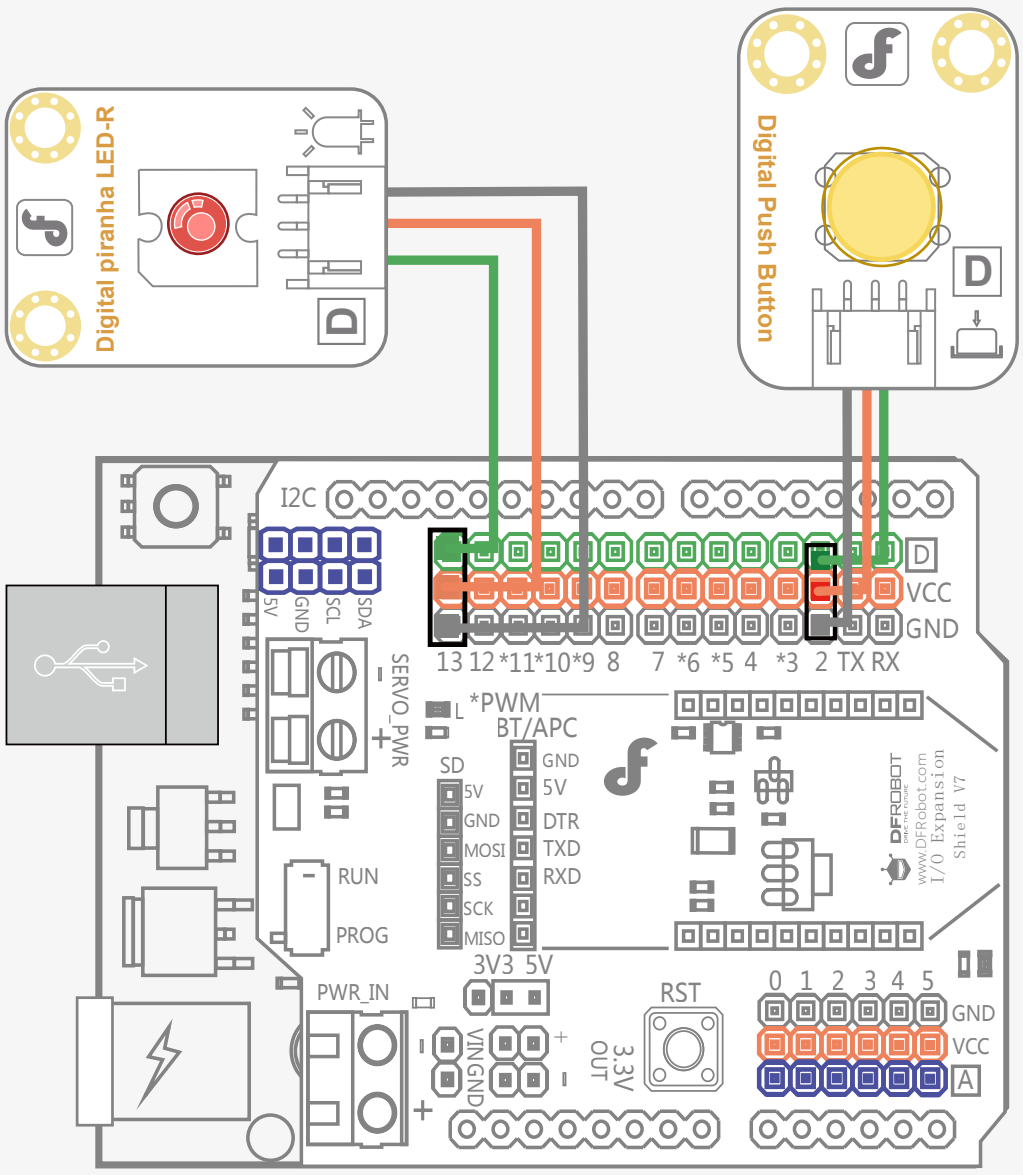
Connections

Connect the Digital Push Button to digital pin 2

Connect the Digital Piranha LED-R to digital pin 13

Be sure that your power, ground and signal connections are correct or you risk damaging your components.

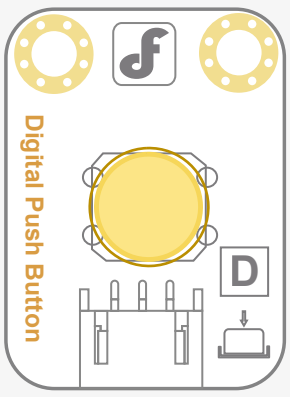
When you have connected the components and checked your connections, connect the microcontroller to your PC with the USB cable so you can upload a program.



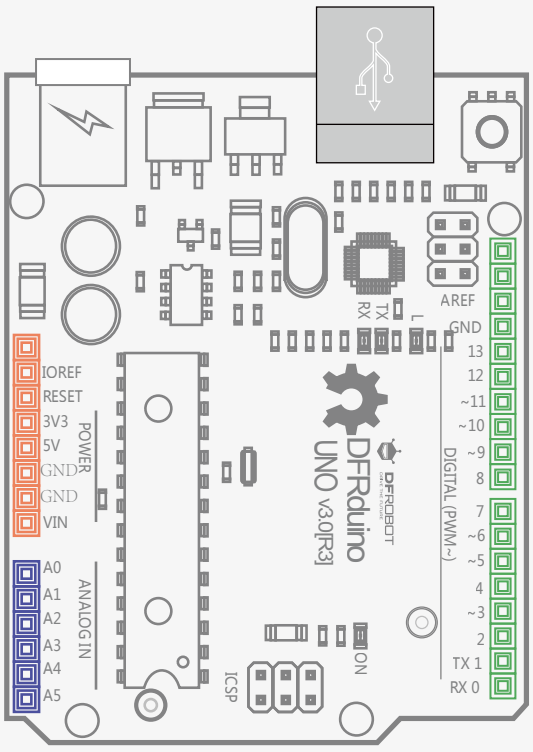
Hardware Analysis
(Digital Input – Digital Output)

In the control device we have made:

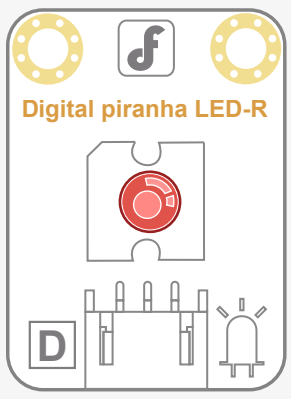
The digital push button is the input unit and the LED is the output unit. Like our last experiment, we have one digital input and one digital output.



Input Device



Control Device



Output Device

Code Input

```
//Sample Code 3-1: Mini Lamp


int buttonPin = 2;
int ledPin = 13;
int ledState = HIGH;
int buttonState;
int lastButtonState = LOW;
long lastDebounceTime = 0;
long debounceDelay = 50;

void setup() {
  pinMode(buttonPin, INPUT);
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, ledState);
}
void loop() {

  int reading = digitalRead(buttonPin);

  if (reading != lastButtonState) {
    lastDebounceTime = millis();
  }
  if ((millis() - lastDebounceTime) > debounceDelay) {
    if (reading != buttonState) {
      buttonState = reading;
      if (buttonState == HIGH) {
        ledState = !ledState;
      }
    }
  }
  digitalWrite(ledPin, ledState);
  lastButtonState = reading;
}
```

Use this sample code to implement the behavior we want.

You can copy and paste it in to the Arduino IDE, but if you want to develop you skills we recommend typing it out.

When you have finished, click “Verify” to check the code for syntax errors. If the code verifies successfully, you can upload it to your microcontroller. Once the code has uploaded, press the button. The LED should turn on.

When the button is pressed again, the LED should turn off.

Code Analysis

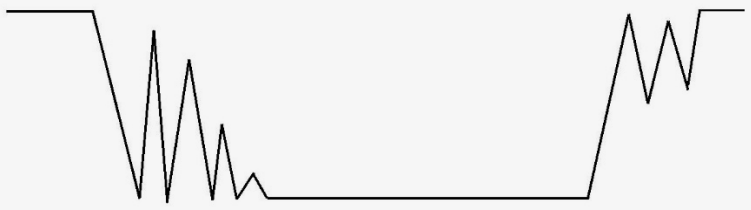
The button is the input device and the LED is the output device.

```
pinMode( buttonPin, INPUT);
pinMode( ledPin, OUTPUT);
```

`digitalWrite()` reads the state of the button

```
int reading =
digitalRead( buttonPin);
```

When the button switches between `HIGH` and `LOW` states, there will be a brief period of oscillation. This is visually represented below:



To avoid signal errors such as this, we will wait when the signal changes and read it some time after.

The function `millis()` will record the time when data collected has changed

```
if ( reading != lastButtonState) {
   lastDebounceTime =
            millis();
            }
```

`millis()` is a function which measures duration. It is measured in milliseconds (1 second = 1000 milliseconds). This particular function will start counting when the microcontroller is powered on.

Wait for another 50ms and check if the button's signal has the same value with its current state. If not, change the state of the button. If the button is in the state of `HIGH` (meaning the button has been pressed), change the state of `LED`.

```
  if (( millis() - lastDebounceTime) >
debounceDelay) {
    if ( reading != buttonState) {
      buttonState = reading;

      if ( buttonState == HIGH) {
          ledState = ! ledState;
      }
    }
```

Exercise

People are always listening to music on their head-phones. This means that if they are at home and the doorbell rings, they cannot hear it.

A doorbell with a mini lamp would solve this problem. If the doorbell is pressed, the lamp will light up. Why not try and add this feature to your own doorbell at home?

Lesson 7

Sound Activated LED

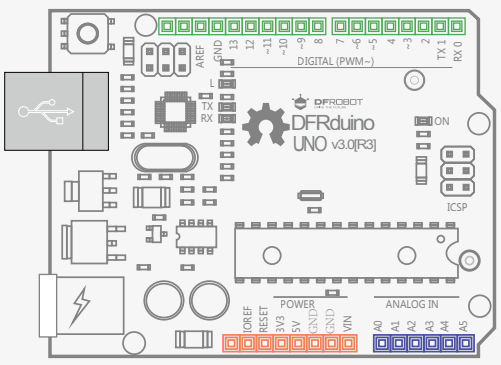
07

DFROBOT
DRIVE THE FUTURE

Sound Activated LED

In this session we will make a sound activated LED. You will be able to make an LED flash by clapping your hands. To make it work, we need a sound sensor. We can use a sound sensor to make various interactive and sound control devices besides this experiment.

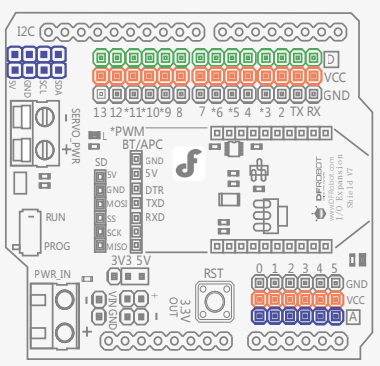
Parts Needed:



x1

DFRduino Uno

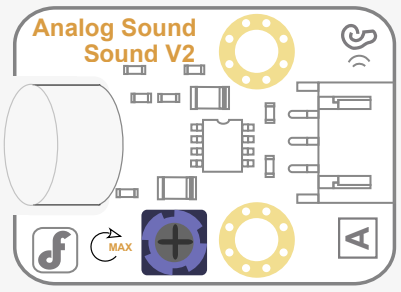
(with USB cable)



x1

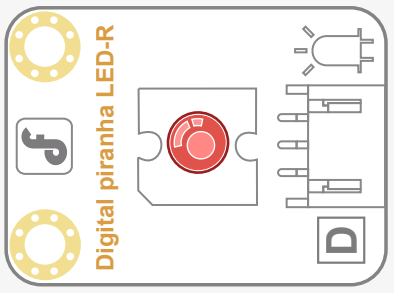
I/O Sensor Expansion

Shield V7.1



x1

Analog Sound Sensor



x1

Digital Piranha LED-R

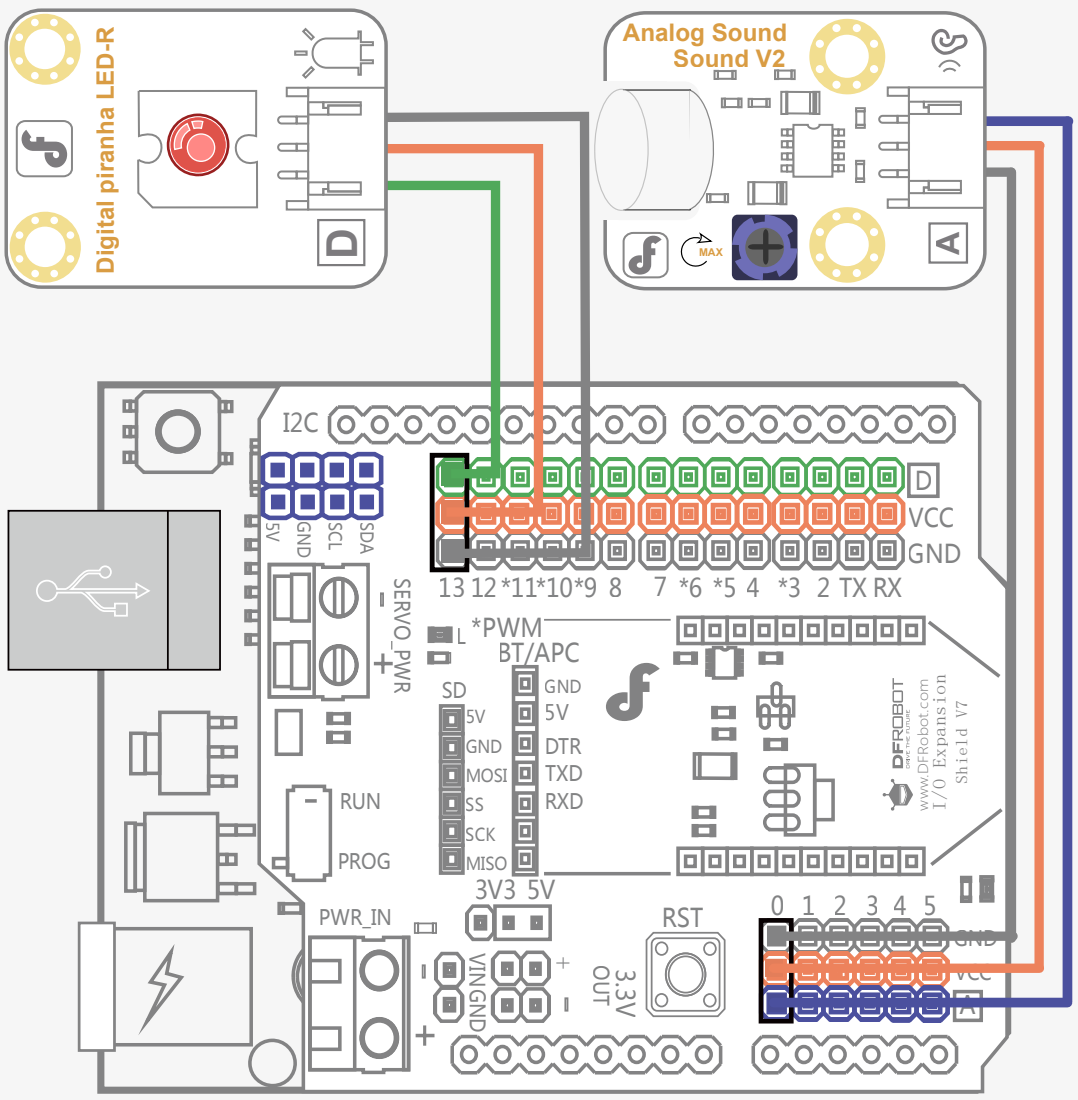
Connections

Connect the analog sound sensor to analog pin 0

Connect the Digital Piranha LED-R to Digital pin

13

Use the diagram below for reference.



Be sure that your power, ground and signal connections are correct or you risk damaging your components.

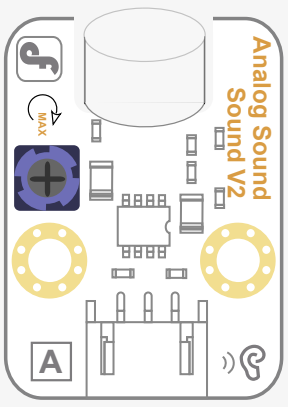
When you have connected the components and checked your connections, connect the microcontroller to your PC with the USB cable so that you can upload a program.

Hardware Analysis
(Analog Input – Digital Output)

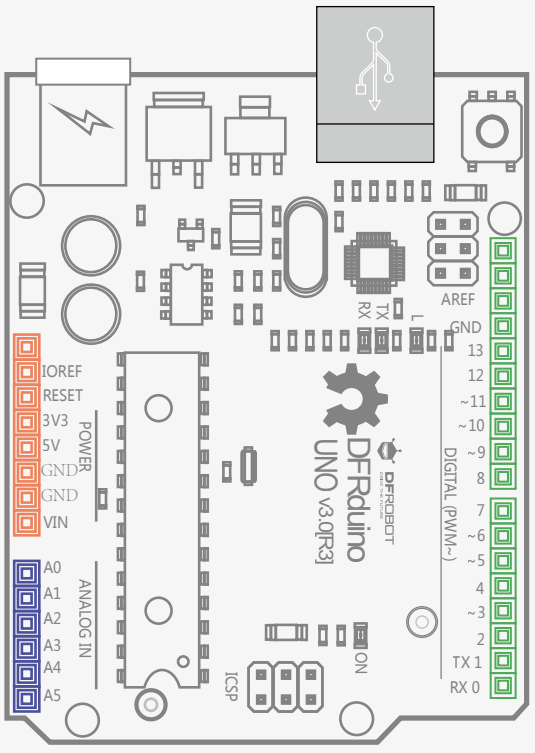
In the previous sessions we have talked about analog and digital signals. The analog sound sensor is, as the name suggests, an analog sensor.

Digital signals have only two values: 0 and 1
Analog signals have many values, ranging between 0 - 1023

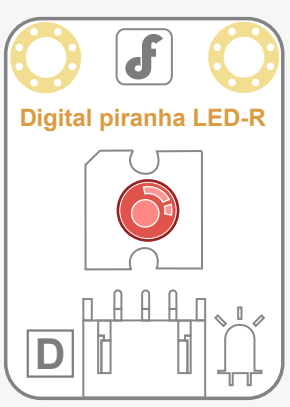
More detailed information can be found in Session 2.



Input Device



Control Device



Output Device

Code Input

```
// Item Four—Sound Control Light
int soundPin = 0; // Analog sound sensor is to be attached to analog
int ledPin = 13; // Digital Piranha LED-R is to be attached to digital

void setup() {
pinMode(ledPin, OUTPUT);
// Serial.begin(9600); // You can uncomment this for monitoring
}

void loop(){
int soundState = analogRead(soundPin); // Read sound sensor's value

// Serial.println(soundState); // serial port print the sound sensor's value
// if the sound sensor's value is greater than 10, the light will be on for 10 seconds.
//Otherwise, the light will be turned off

if (soundState > 10) {
digitalWrite(ledPin, HIGH);
delay(10000);
}else{
digitalWrite(ledPin, LOW);
}
}
```

Use this sample code to implement the behavior we want.
You can copy and paste it in to Arduino IDE, but if you want to develop you skills we recommend typing it out.

When you have finished, click “Verify” to check the code for syntax errors. If the code verifies successfully, you can upload it to your microcontroller.

Once the code has uploaded, try clapping your hands. The LED should flash on.
When there is no sound, the LED should remain off.

Code Review

In the `setup()` function, the LED is declared as an `OUTPUT`.

Why isn't the sound sensor declared as an `INPUT`?

The reason is because all analog pins are in `INPUT` mode by default, so **we don't need to set them!**

The sound sensor is the input unit, and it should read from analog pin 0. The function for this is:

```
analogRead(pin)
```

This function is used to read the analog pin. Your microcontroller's analog pins are attached to an A/D converter (analog to digital converter - a special chip on the microcontroller board), so voltage between 0 and 5V will be mapped to a number between 0 and 1023. Each number collected represents a voltage. For example, 512 = 2.5V

Finally, an `if` statement checks if the result exceeds a certain value.

```
if (soundState > 10) {

...
}else{

... }
```

You can open the serial monitor and test out a threshold value that best represents the sound volume you need. Then you can adjust the value in the code accordingly.

Lesson 8

Fading LED

08

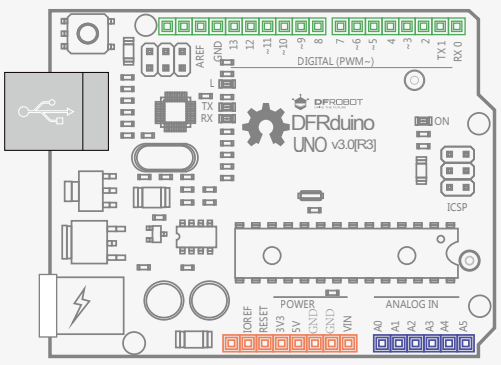
DFROBOT
DRIVE THE FUTURE

Fading LED

In previous sessions we've learned how to turn an LED on and off. In this project we're going to make a fading LED. The LED will go from dim to bright and bright to dim continuously.

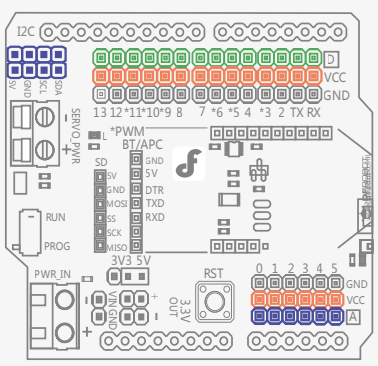
To achieve this we are going to use **pulse width modulation (PWM).**

Parts Needed:



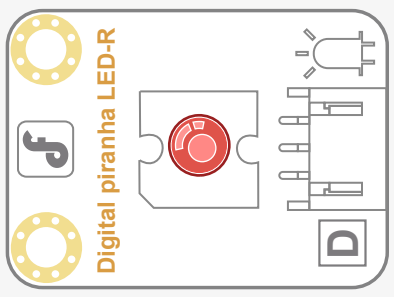
x1

DFRduino Uno
(with USB cable)



x1

I/O Sensor Expansion
Shield V7.1



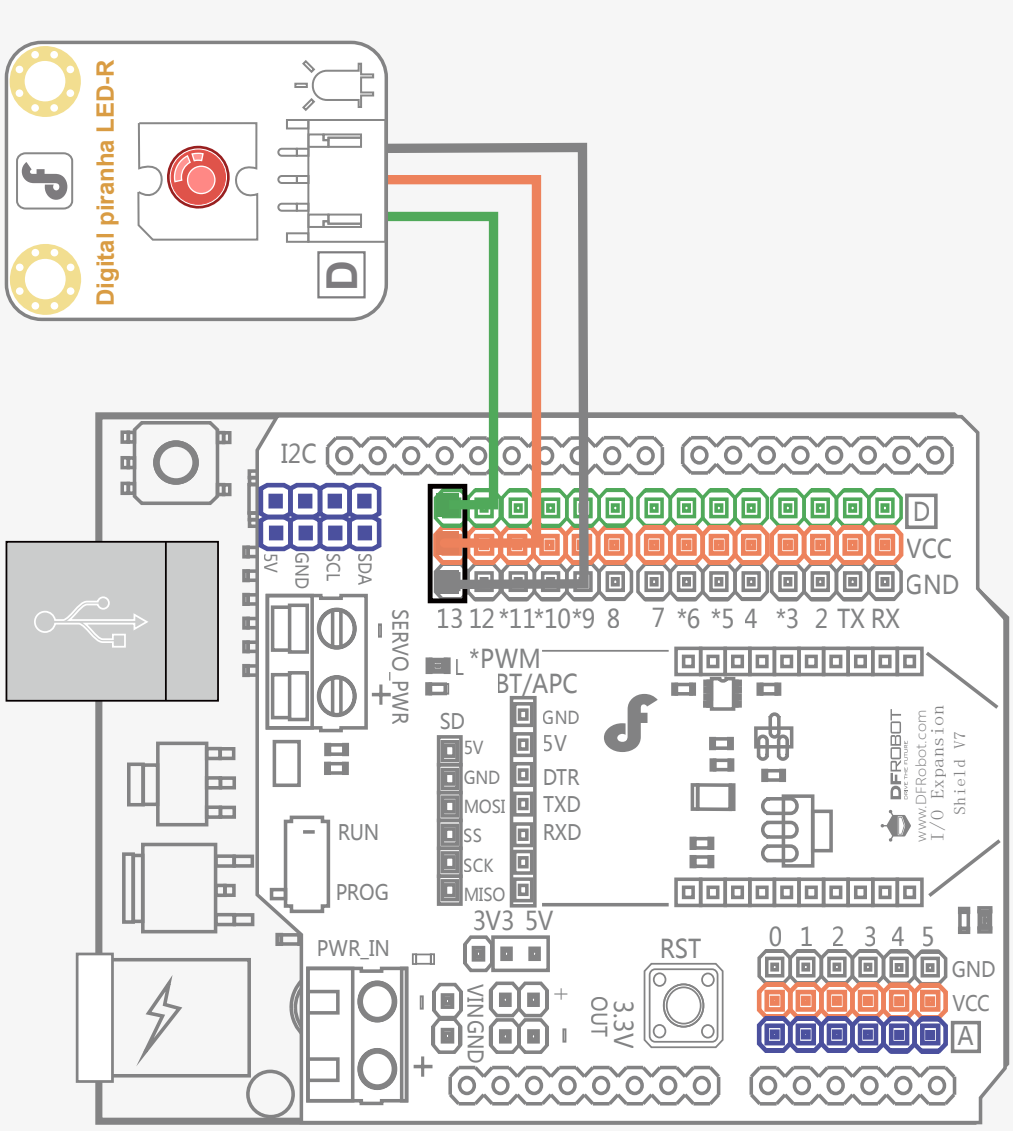
x1

Digital Piranha LED

Connections

Attach the Digital Piranha LED-R to digital pin 10.

Use the diagram below for reference:

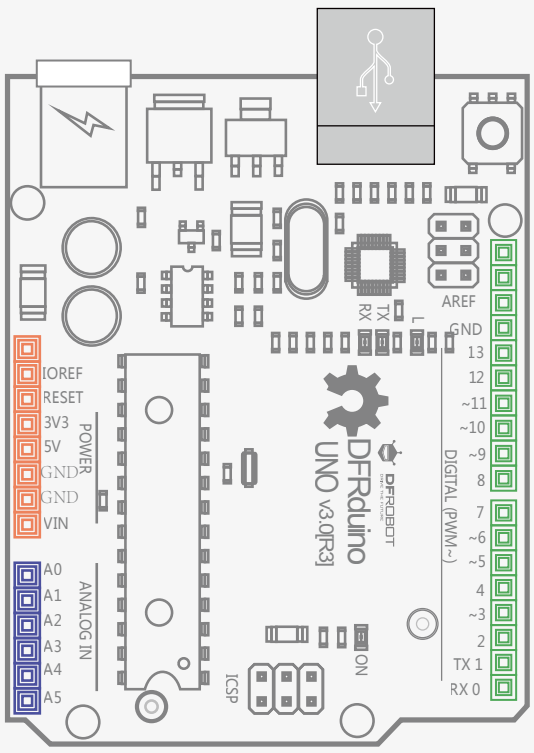


Be sure that your power, ground and signal connections are correct or you risk damaging your components.

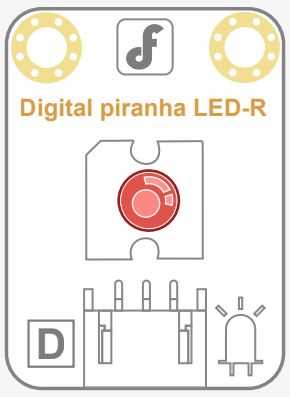
When you have connected the components and checked your connections, connect the microcontroller to your PC with the USB cable so you can upload a program.

Hardware Analysis (Analog Output)

This device is similar to experiment 1 where we made an LED blink. We also only have one output unit. The difference in this project is that we are using the LED for analog output, rather than digital output. You can refer to the code section for more information about this.



Control Device



Output Device

Code Input

```
// Item Five- Fading LED
int ledPin = 10;


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

void loop(){
  for (int value = 0 ; value < 255; value=value-1) {
     analogWrite(ledPin, value);
     delay(5);
  }
  for (int value = 255; value >0; value=value-1){
     analogWrite(ledPin, value);
     delay(5);
}
}
```

Use this sample code to implement the behavior we want.

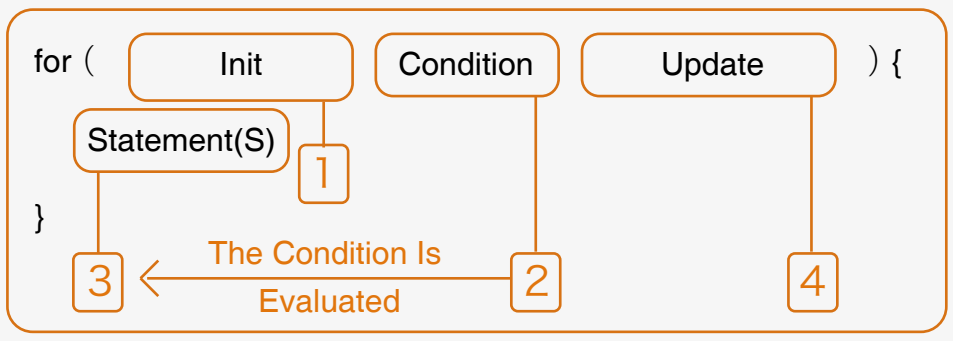
You can copy and paste it in to the Arduino IDE, but if you want to develop you skills we recommend typing it out.

When you have finished, click “Verify” to check the code for syntax errors. If the code verifies successfully, you can upload it to your microcontroller.

Once the code has uploaded, you should see the LED gradually growing brighter and then dimming.

Code Analysis

When we need to repeat a line of code a number of times, we can use a `for` loop.



The following is the process of a for loop:

1. The “init” statement is executed first and only once.
2. The “condition” is evaluated. If it is false, the loop is stopped.
3. The “statement(s)” is executed.
4. “Update” is executed and the code returns to step 2.

Let’s return to our code:

```
for (int value = 0; value < 255; val-
ue=value+1){
        ...
}
for (int value = 255; value >0;
value=value-1){

   ...
    }
```

These two `for` loops make the value increase from 0 to 255 and then decrease from 255 to 1. These values are used to control the brightness of our LED.

Now let’s take a look at the `analogWrite()` function.

Digital pins only have two values: 0 or 1. How can we send an analog value to a digital pin?

- By using `analogWrite()`.

There are six digital pins on your microcontroller that are marked with the symbol “~”, which means they can send out a PWM signal.

The function is written in the following form:

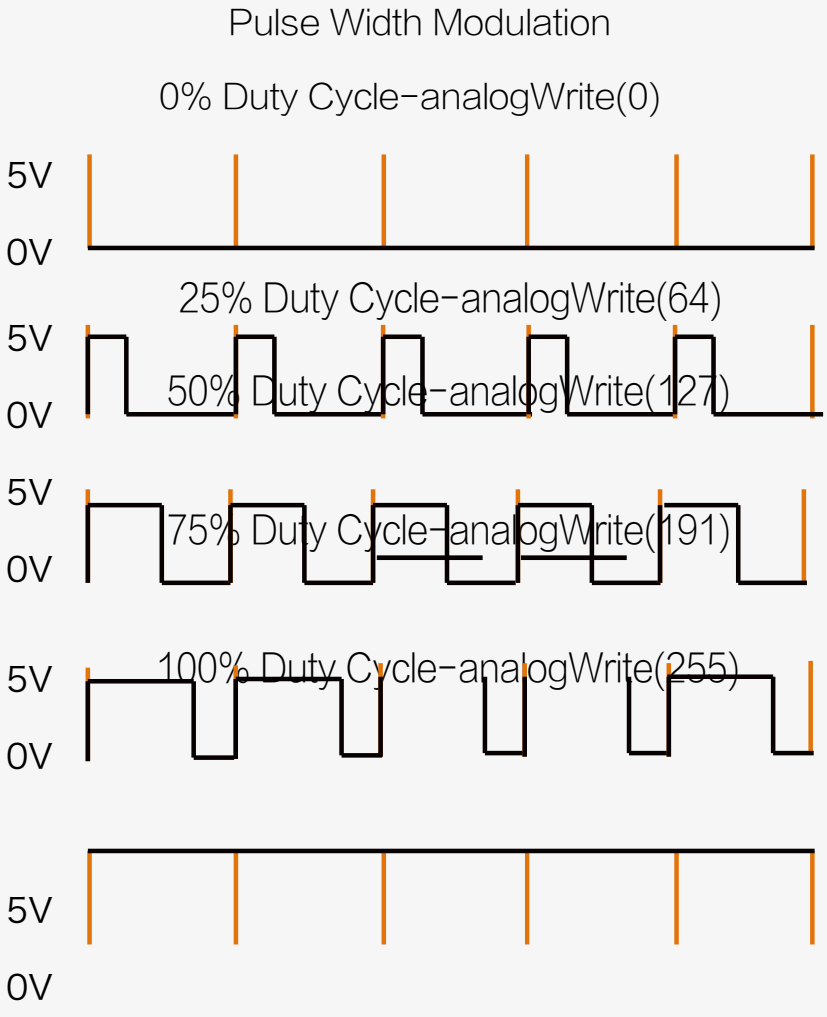
```
analogWrite( pin, value)
```

The `analogWrite()` function is used to write an analog value between 0 - 255 a PWM pin. `analogWrite()` can only be used for PWM pins, which are pins 3, 5, 6, 9, 10 and 11 on your microcontroller.

PWM stands for **pulse width modulation**. Very simply, PWM allows us to achieve analog results with digital signals by switching signals between `HIGH` and `LOW` thousands of times a second. Using this, we can simulate voltages between 0v and 5v.

If you were to look at the PWM signal on an oscilloscope (a device that you can use to view electrical signals), you would see a square wave. The top of the square is the HIGH signal, and the bottom is `LOW`. The greater the width of the square, the longer the `HIGH` pulse – hence the name pulse width modulation.

Let's explore PWM in more detail by examining these five square waves:



Orange vertical lines on this diagram represent an interval.
Black lines represent the electric pulse.
Each value included in the `analogWrite( value)` function can be mapped to a certain percentage. 0 = 0% and 255 = 100%.

The duty cycle refers the length of time the pulse goes from `LOW` to `HIGH` and `HIGH` to `LOW`.
The duty cycle of the first square wave is 0% and the corresponding value is 0.
If the duty cycle is 100%, the analog value will be 255.
The longer the duty cycle, the brighter the light will be.

PWM's electronic applications include adjusting the brightness of an LED, controlling the speed of a DC motor, and many others. Can you think of any other applications for it?

With the same hardware connections we've used in this session, the LED can achieve completely different effects by using a different code. Have a play with the code and see if you can make different lighting effects. Use your imagination and have fun with it!

Lesson 9

Light Regulator

09

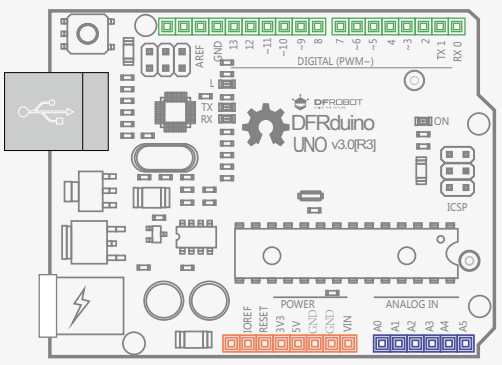
DFROBOT
DRIVE THE FUTURE

Light Regulator

A light regulator is a device with which you can control brightness of a light. In this session, we'll show you how to control the brightness of an LED with an analog rotation sensor (also known as a potentiometer). Using this, the light will change from dim to bright and bright to dim depending on the rotation of the sensor. The greater the rotation angle, the brighter the LED, and vice versa.

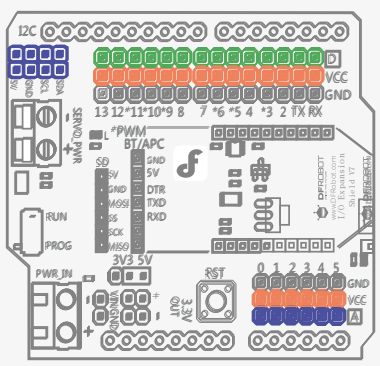
The rotation sensor can also be used to control the rotation angle of a servo or the speed of a DC motor. You can find many uses for the rotation sensor.

Parts Needed:



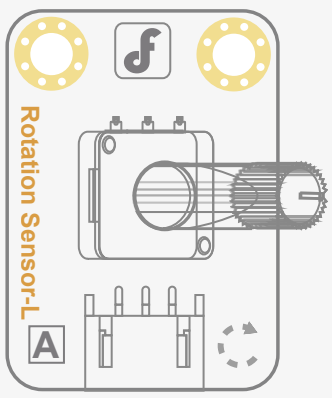
x1

DFRduino Uno
(with USB cable)



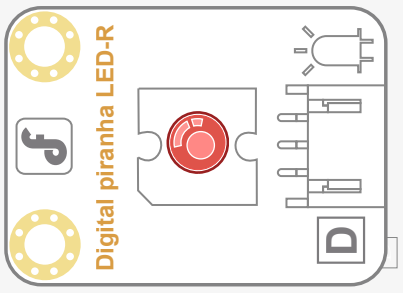
x1

I/O Sensor Expansion Shield
V7.1



x1

Analog Rotation Sensor



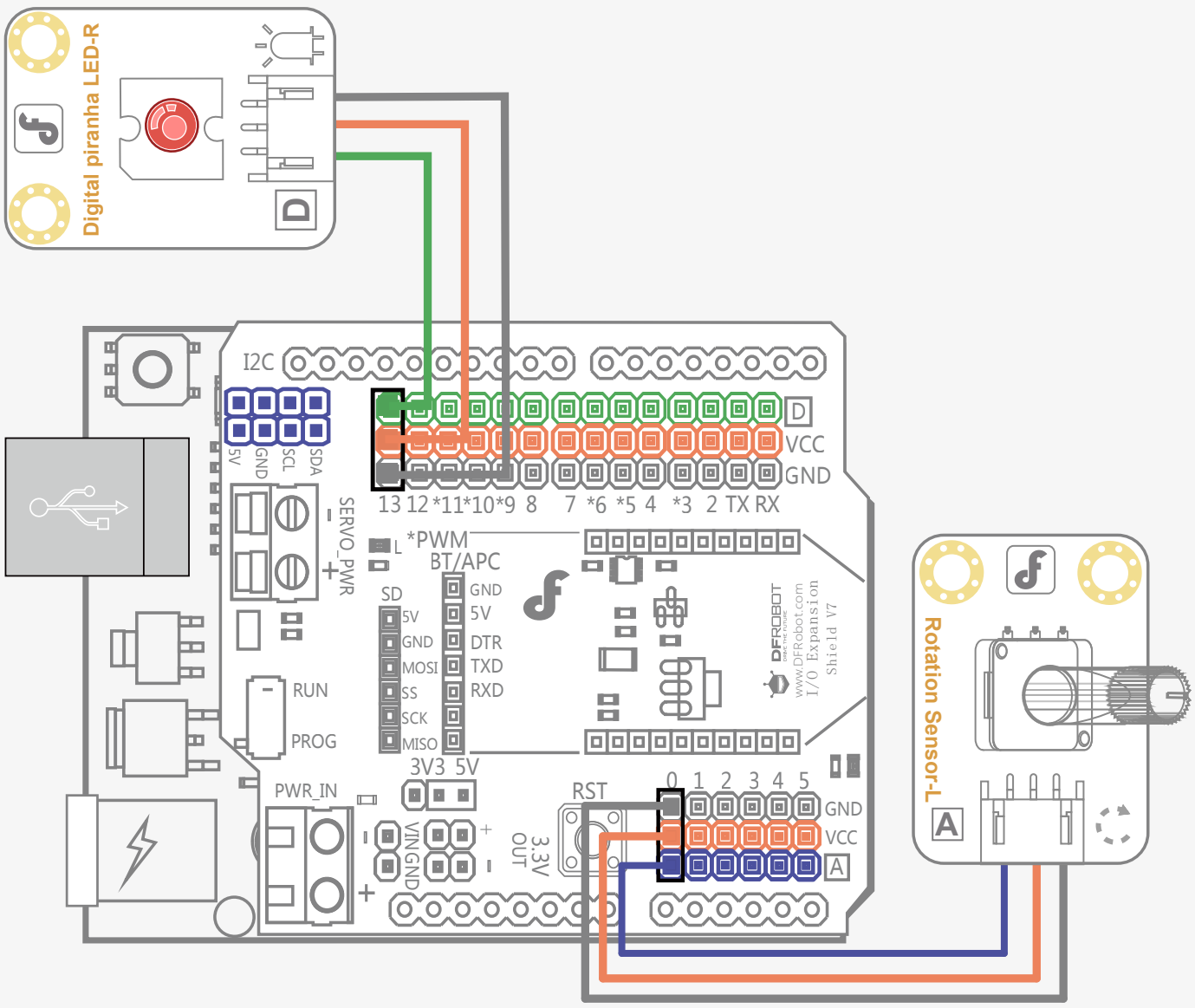
x1

Digital Piranha LED-R

Connections

Be sure that your power, ground and signal connections are correct or you risk damaging your components.

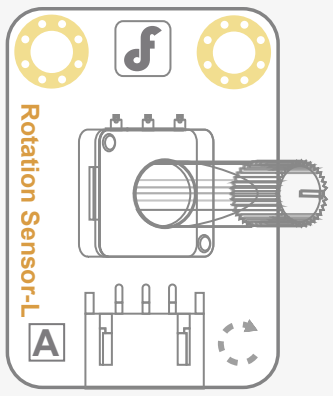
When you have connected the components and checked your connections, connect your microcontroller to your PC with the USB cable so you can upload a program.



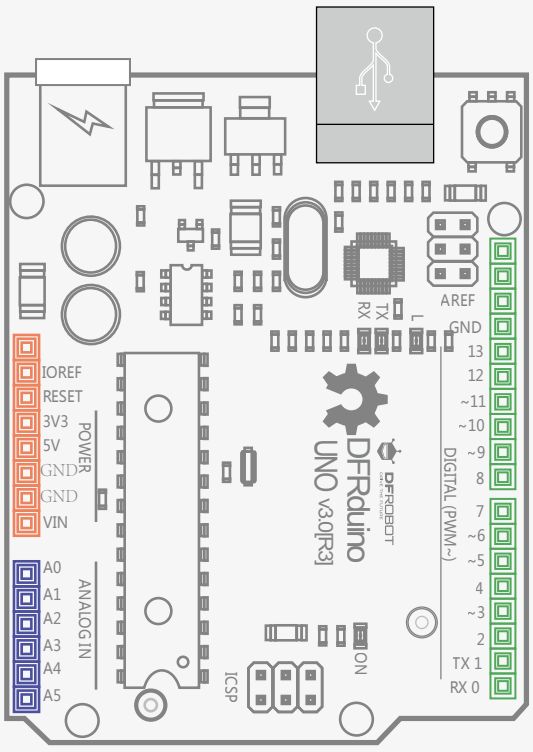
Hardware Analysis
(Analog Input-Analog Output)

In the last session we learned how to achieve analog output with a digital pin using PWM.
In this session, we are going to control analog output using analog input.

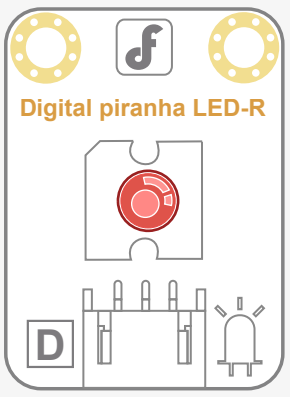
The rotation sensor is the input device. The LED is the output device. Both input and output devices use analog values.



Input Device



Control Device



Output Device

Code Input

```
// Item Six—Light -regulator
int potPin = 0;  // Analog Rotation Sensor shall be attached to analog pin 0
int ledPin = 10; // LED shall be attached to digital pin No.10

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

void loop() {
  int sensorValue = analogRead(potPin);    // collect value at analog pin No.0
// A value between 0 and 1023 to mapped to a value between 0 and 255
int outputValue = map(sensorValue, 0, 1023, 0, 255);
analogWrite(ledPin, outputValue);              // write corresponding value for LED
  delay(2);
}
```

Use this sample code to implement the behavior we want.

You can copy and paste it in to the Arduino IDE, but if you want to develop you skills we recommend typing it out.

When you have finished, click “Verify” to check the code for syntax errors. If the code verifies successfully, you can upload it to your microcontroller.

Turn the shaft of the rotation sensor slowly. The LED’s brightness should change as you turn it.

Code Analysis

Here we have used the `map` function. The `map` function is used to map a value within a range to one within another range. Let's examine it a little closer:

The function's syntax is as follows:

```
map( value,  fromLow,
fromHigh, toLow, toHigh)
```

The function maps a value between `fromLow` and `fromHigh` to one between `toLow` and `toHigh`.

The parameters included in the map function are:

`value` the value that needs to be mapped

`fromLow` lower limit of the current range

`fromHigh` upper limit of the current range

`toLow` lower limit of the target range

`toHigh` upper limit of the target range

One advantage of map function is that the lower limit of both ranges can be a number greater than their upper limits:

```
y = map ( x, 1, 50, 50, 1);
```

The value included in the map function can also be a negative number:

```
y = map ( x, 1, 50, 50, -100);
```

Let's re-examine the code:

```
int outputValue = map( sensorValue, 0, 1023, 0, 255);
```

The code above is used to map a value (0-1023) read from the analog pin to one used on the PWM pin (0-255).

Lesson 10

Open Sesame

10

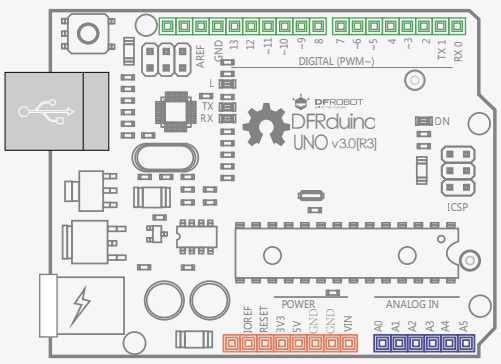
DFROBOT
DRIVE THE FUTURE

Open Sesame!

In this session we are going to use servos to make a door move. The door can't be opened easily – you will have to complete a challenge before it does!

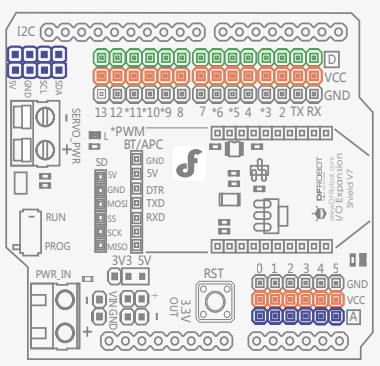
The challenge will be explained below…

Parts Needed:



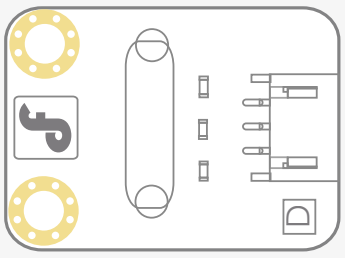
x1

DFRduino Uno
(with USB cable)



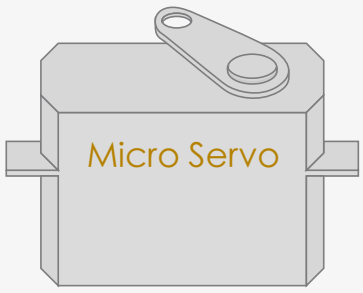
x1

I/O Sensor Expansion Shield
V7.1



x1

Digital Tilt Sensor



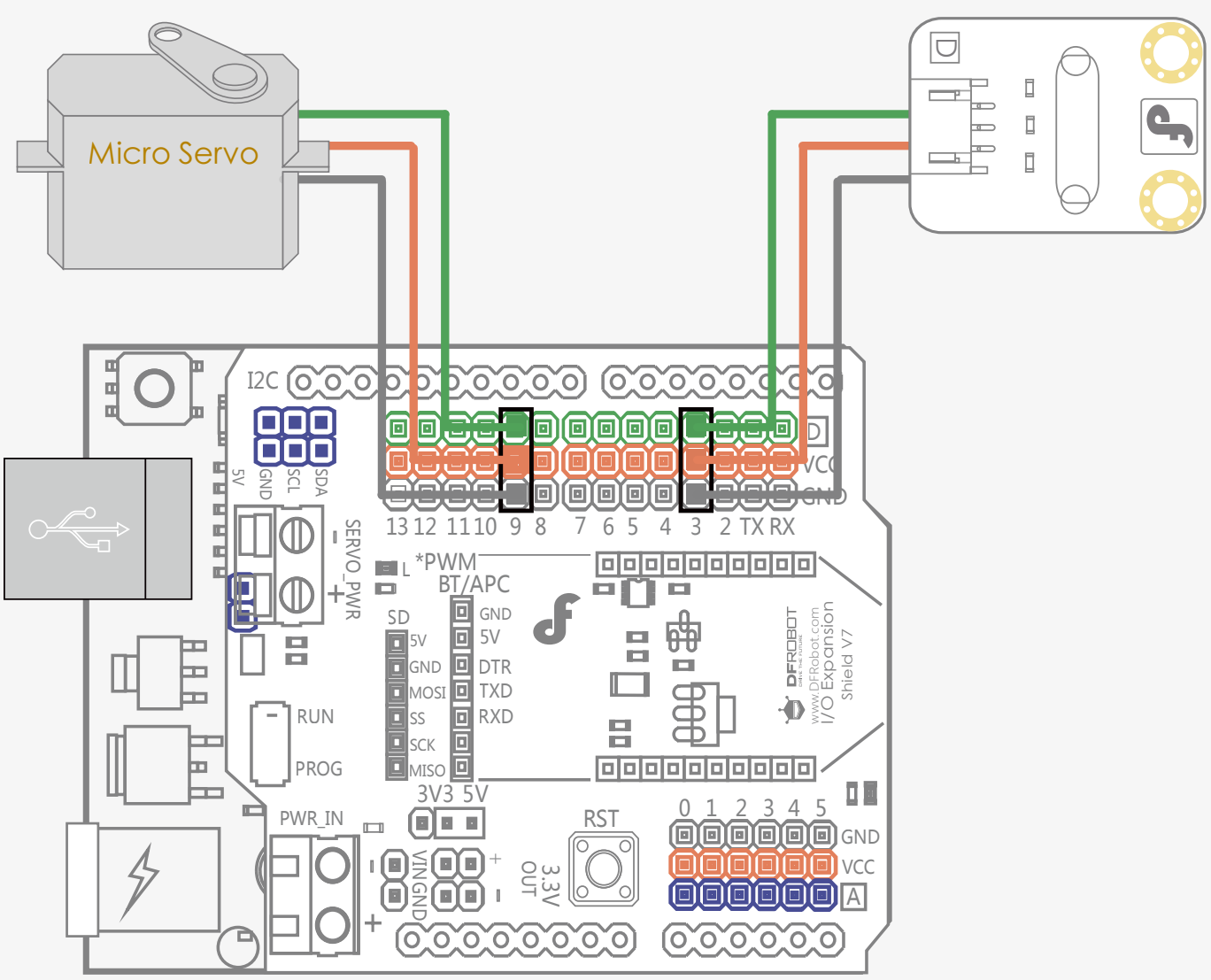
x1

TowerPro SG50 Servo

Connections

Connect the TowerPro SG50 to digital pin 9 Connect the Digital Tilt Sensor to digital pin 3

Review the diagram below for reference:



Be sure that your power, ground and signal connections are correct or you risk damaging your components.

When you have connected the components and checked your connections, connect the microcontroller to your PC with the USB cable so you can upload a program.

Code Input

```
#include <Servo.h>
int sensorPin = 3; // Digital Vibration Sensor-Digital 3
Servo myservo;
int pos = 0;
void setup() {
  Serial.begin(9600);
  pinMode(sensorPin, INPUT);
  myservo.attach(9); // Servo– Digital 9
}
void loop() {
  int sensorState = digitalRead(sensorPin); //Read state of the digital vibration sensor
  Serial.println(sensorState);
  if (!sensorState) { //If the state is 0, The servo moves 2°  more but                    does  not exceed 180°
    pos = pos + 2;
    if (pos >= 180) {
      pos = 180;
    }
    myservo.write(pos); //Write in rotation angle of the Servo
    Serial.println(pos); // Print rotation angle in serial port
    delay(100);
  } else { // Else, the servo moves 2°  less but not less than 0° .
    pos = pos - 2;
    if (pos <= 0) {
      pos = 0;
    }
    myservo.write(pos);
    Serial.println(pos);
    delay(100);
  }
  delay(1);
}
```

Use this sample code to implement the behavior we want.

You can copy and paste it in to the Arduino IDE, but if you want to develop you skills we recommend typing it out.

When you have finished, click “Verify” to check the code for syntax errors. If the code verifies successfully, you can upload it to your microcontroller.

Shake the tilt sensor and you‘ll see the servo’s rotation angle gradually increase. When you stop shaking, the servo’s rotation angle will decrease like a door slowly closing.

Code Analysis

Here we have used the <Servo.h> library

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

This library is included in Arduino IDE. The default location of <Servo.h> should be: Arduino-1.0.5/ libraries/ Servo/ Servo.h

You will be able to find it in the Arduino IDE menu bar: Sketch > Import Library > Servo

We cannot use the functions in the library directly. We need to create a servo object in code. It would serve as a medium to connect our code with the library.

```
Servo myservo;
```

Once the servo object is declared, you can use functions from the library with the following method:

```
myservo.functionName();
```

Don't forget the . between myservo andf unctionName

We use the function attach() to define which pin the servo should be controlled by.

```
attach(pin);
```

The attach () function has one parameter – pin , which refers to one of the digital pins (we do not recommend using digital 0 or 1 due to TX and RX interference). Here we have chosen digital pin 9.

```
myservo.attach(9);
```

How do we write the corresponding angle in the code?

```
write(pos);
```

pos represents the angle we want the servo to turn to.

Lesson 11

Pandora's Box

11

DFROBOT
DRIVE THE FUTURE

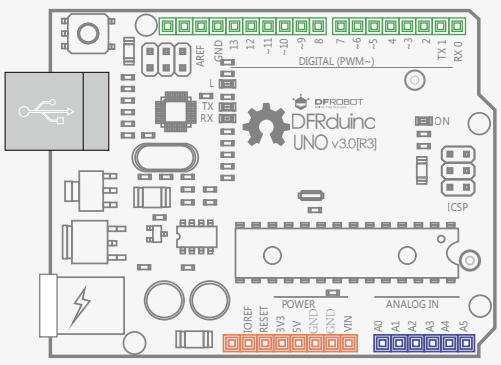
Pandora's Box

Do you dare to open Pandora’s box?

The box stays closed during the daytime, but when night comes it will gradually open and the light inside will glow brighter and brighter.

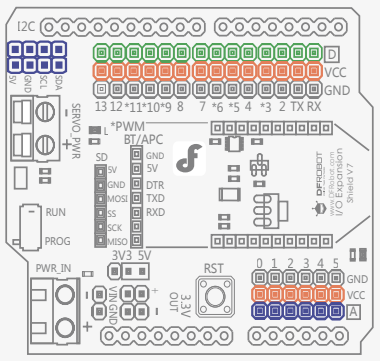
An analog ambient light sensor is used to detect the intensity of light outside the box. When the value reaches the limit for the night, the servo will open the box and an inner-led will glow brighter.

Parts Needed:



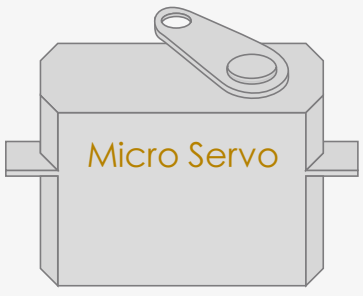
DFRduino Uno
(with USB cable)

x1



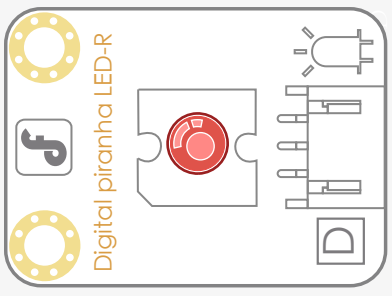
I/O Sensor Expansion
Shield V7.1

x1



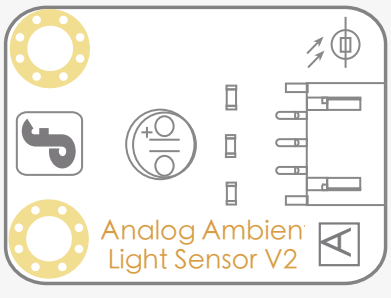
TowerPro SG50 Servo

x1



Digital Piranha LED-R

x1



Analog Ambient Light Sensor V2

x1

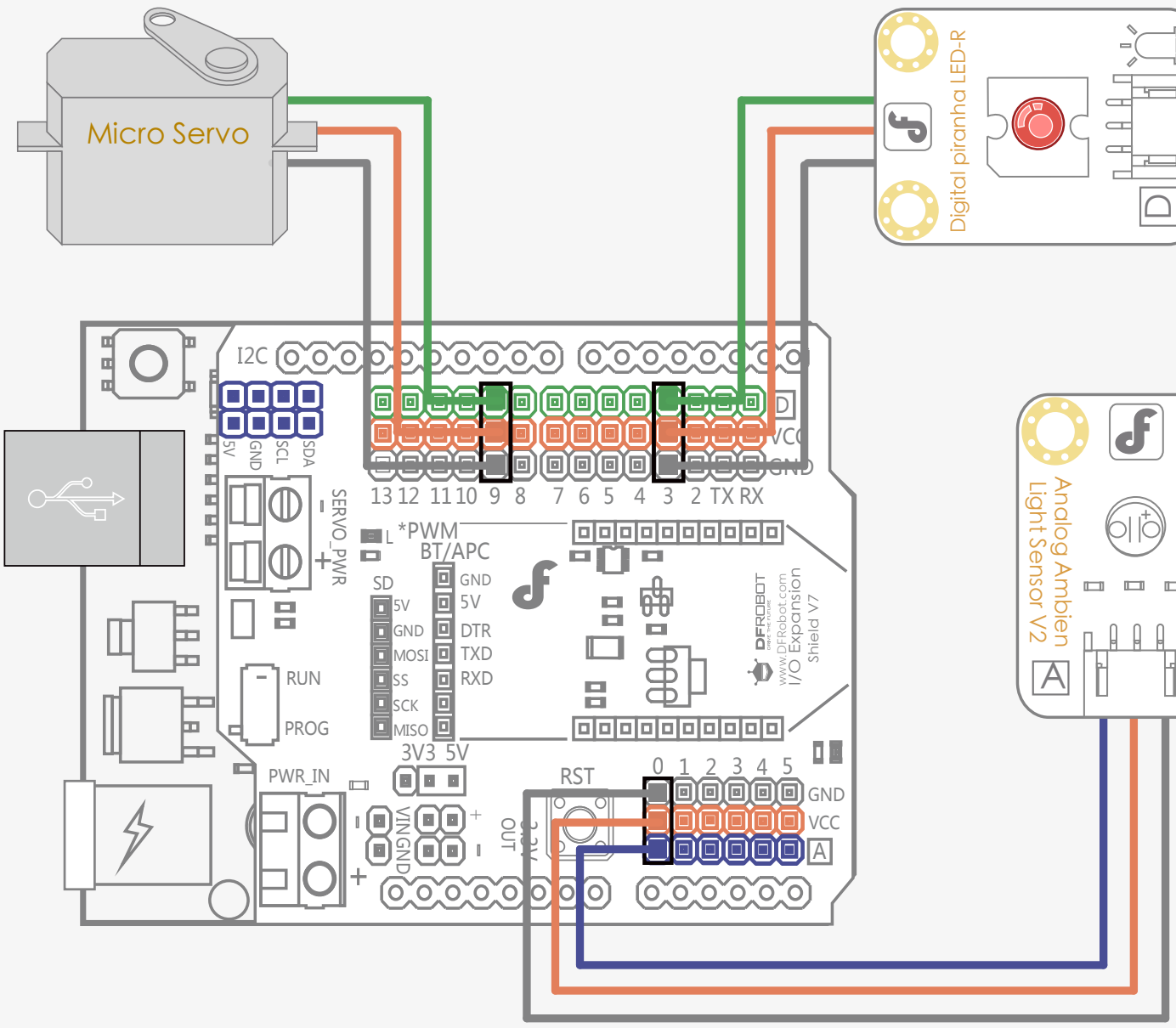
Connections:

Connect the TowerPro SG50 Servo to digital pin 9

Connect the Analog Ambient Light Sensor V2 to analog pin 0

Connect the Digital Piranha LED-R to digital pin 3

You can refer to the diagram below for reference:



Be sure that your power, ground and signal corrections are correct or you risk damaging your components.

When you have connected the modules and checked your connections, connect the microcontroller to your PC with the USB cable so you can upload a program.

Code Input

```
#include <Servo.h>
Servo myservo;
int LED = 3; // Set the LED light to be digital pin 3
int val = 0; // val stores analog ambient light sensor’ s value
int pos = 0;
int light = 0;

void setup() {
  pinMode(LED, OUTPUT); // LED is set to be in the output mode
  Serial.begin(9600); // Baud rate of the serial port is set to be 9600
  myservo.attach(9); // attach the servo to digital pin 9
  myservo.write(0); // initial angle is 0
}

void loop() {
  val = analogRead(0); // read the analog ambient light sensor’ s value
  Serial.println(val); // Check sensor value in serial port
  if (val < 40) { // when val is less than the pre-set value, increase the angle
    pos = pos + 2;
    if (pos >= 90) { // The angle should not be more than 90
      pos = 90;
    }
    myservo.write(pos); // write angle of the servo
    delay(100);
    light = map(pos, 0, 90, 0, 255); // As the angle expands, the LED becomes brighter
    analogWrite(LED, light); // write the brightness value
  } else {
    pos = pos - 2; // decrease 2°
    if (pos <= 0) {
      pos = 0; // until 0°
    }
    myservo.write(pos); // write angle of the servo
    delay(100);
    light = map(pos, 0, 90, 0, 255); // As the angle shrinks, the LED light becomes darker
    analogWrite(LED, light); // write the brightness value
  }
}
```

Use the sample code to implement the behavior we want.
You can copy and paste it in to the Arduino IDE, but if you want to develop you skills we recommend typing it out.

When you have finished, click “Verify” to check the code for syntax errors. If the code verifies successfully, you can upload it to your microcontroller.

Attach the servo on to the hinge of the box so that the arm is able to open it. The ambient light sensor needs to be placed outside the box to detect the ambient lighting. The LED is placed inside the box.

Place the box somewhere dark and check if it opens automatically.

If you have questions about the functions in the sample code, please refer to the previous sessions for details.

We hope these sessions are just the beginning of your robotics journey.

Be creative and have fun playing with your DFRduino and modules. If you need more new and exciting modules, you can get them from the DFRobot Store:

http://www.dfrobot.com/

If you would like to share your creations with us or have any questions or comments about these tutorials, log in to the DFRobot Maker community forum and tell us all about it!

http://www.dfrobot.com/forum/

Good Luck!