

Aeroponics and Vertical Gardening: Programming Activity: Digital Line Outputs

Hardware:

- ✓ [Sparkfun RedBoard Microcontroller*](#)
- ✓ [A to MiniB USB cable](#)
- ✓ PC or Laptop (Windows, Mac OS, Linux)
- ✓ [Solid State Relay](#)
- ✓ [Jumper wires](#)
- ✓ [Small breadboard](#)

*Any Arduino Uno device will work the same.

Objectives:

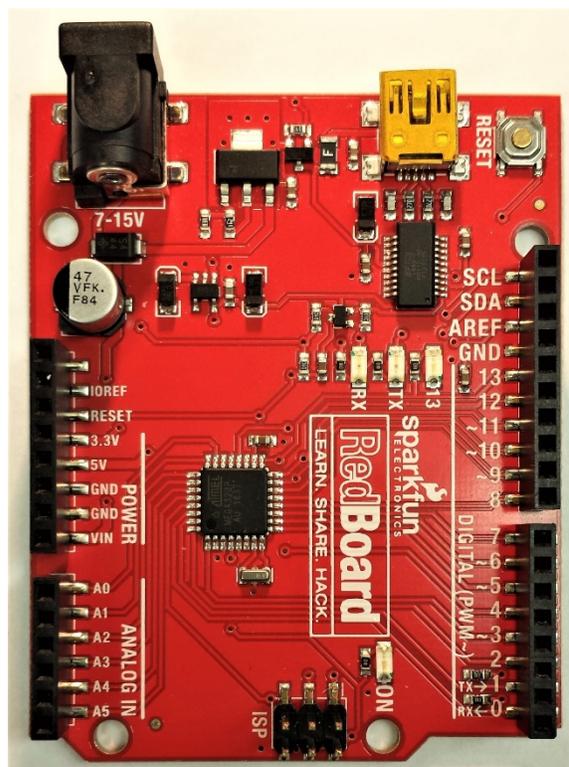
- ✓ Work effectively in a cooperative learning environment
- ✓ Explain the characteristics and function of the digital I/O channels on the Arduino
- ✓ Download and install Arduino software on a Windows PC
- ✓ Connect the Arduino device to the USB port on the computer
- ✓ Initialize the Arduino device on the Windows PC
- ✓ Connect the Arduino device to external devices via a solid state relay.
- ✓ Write a simple program to control outputs using the following commands:
 - `const int`
 - `setup()`
 - `pinMode()`
 - `loop()`
 - `digitalWrite()`
 - `delay()`

Data Acquisition and Control Hardware

Sparkfun RedBoard Microcontroller

The Sparkfun RedBoard is an Arduino-type microcontroller provides both analog and digital connections through a series of pins arranged along the left and right edges of the device. The right side of the device provides connection to 14 digital input/output (DIO) channels. The right side of the device also provides pins for use with R3 shields. The left side of the device provides connection to 6 analog inputs, in addition to assorted power inputs and outputs.

In this lesson, we will focus on the 14 DIO channels located on the righthand side of the microcontroller. These channels are structured into two port registers, Port D and Port B. Port D contains eight channels, identified as digital pins 0 through 7, and Port B contains 6 channels, identified as digital pins 8 through 13. Each channel can be configured individually as an input or output, or each port can be configured as a set of inputs or a set of outputs. When a program is run, the DIO channels configure as programmed.



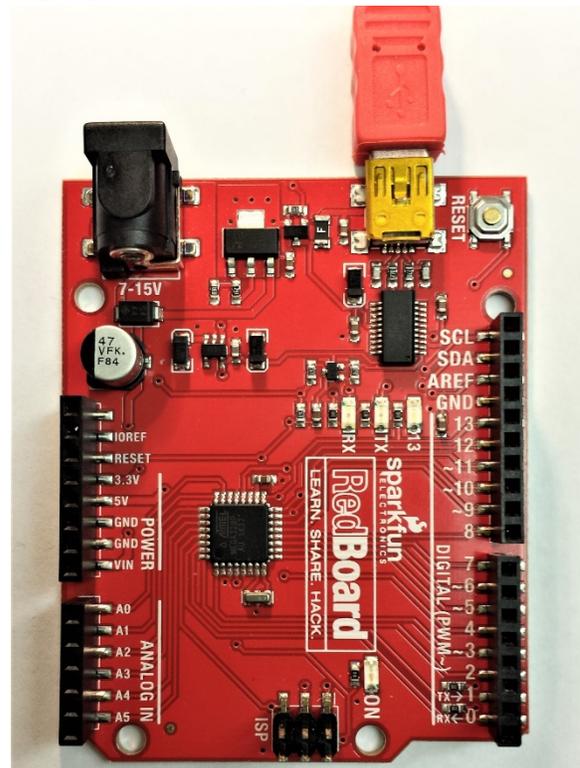
Solid State Relay

The DIO channels are limited to 5 volts (V) direct current (DC) with a maximum of 40 milliamps of current. This isn't very much power, so additional circuitry is often necessary to increase the voltage and amperage (A) to control higher-powered devices. The Solid-State Relay provides this function in this activity.

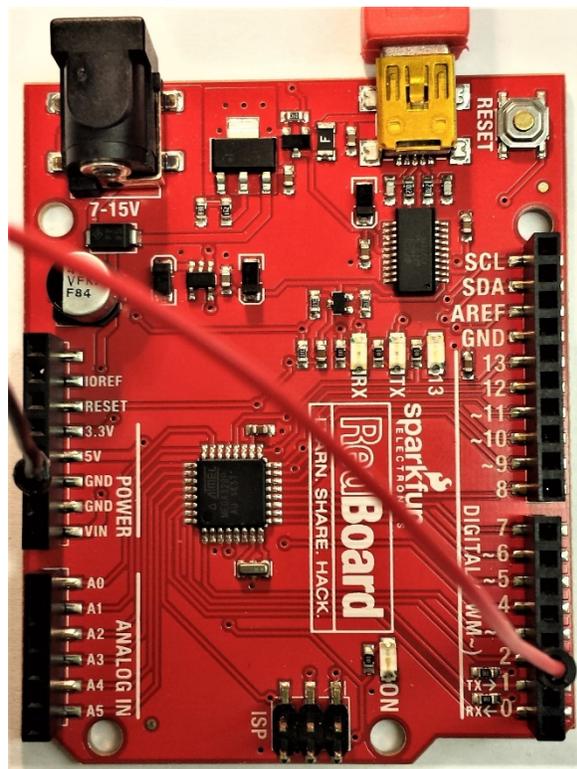
The output of the solid-state relay can control any alternating current (AC) between 24 V and 240 V, and up to 10 A. The relay is controlled by a 5 V to 24 V DC input, which is convenient since our DIO channels have an output voltage of 5 VDC. We will use a DIO channel to control the solid-state relay. Watch [this video](#) on how a basic relay works. A solid-state relay produces the same effect, but uses special materials called semiconductors instead of moving mechanical parts.

Connecting the RedBoard Device to a PC or Laptop

To program the RedBoard device to control digital outputs, the device needs to be connected to the PC with the USB cable. To connect the PC, plug the smaller “MiniB” side of the USB cable into the MiniB connector located near the top right corner of the RedBoard microcontroller. Then plug the larger “A” side of the USB cable into a USB port on the PC. The USB cable allows a program written on the PC to be transferred to the microcontroller. The USB cable can also provide power to the board, although this is not necessary if the device is connected to an external power supply that provides 7 to 15 VDC, using the large power connector near the top left-hand corner of the device. If an external power supply is connected, the USB cable is only needed to transfer a program to the device, and can then be disconnected once the transfer is complete.



Connecting the RedBoard to the Solid-State Relay



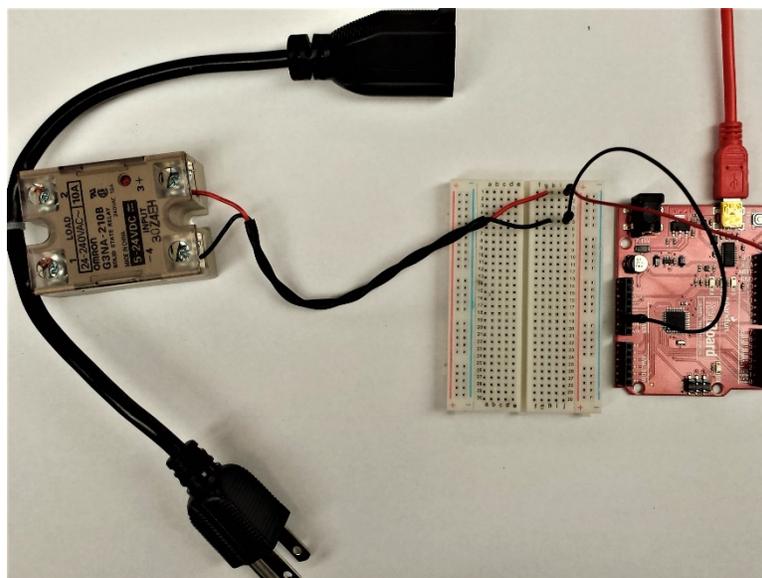
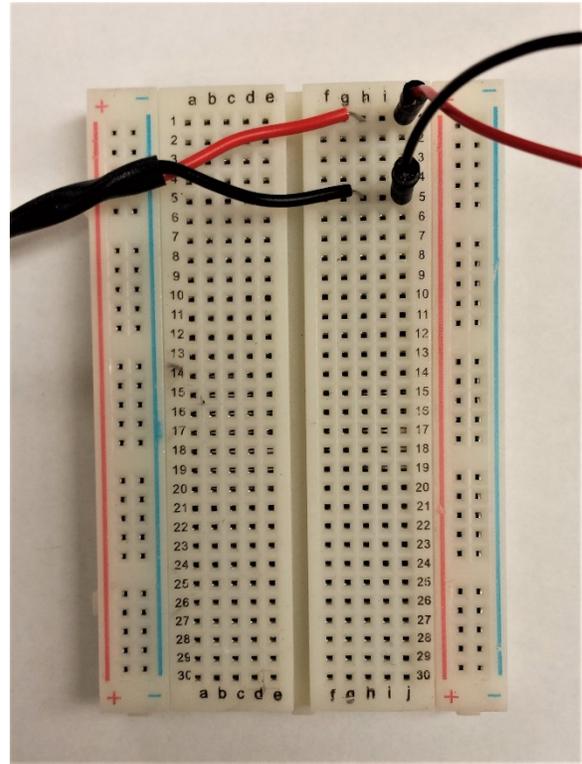
Several connections are required to connect the device to the PC and to the solid-state relay. Before continuing, watch [this video](#) on how to use a breadboard. After watching the video, find any pin on the RedBoard that is labeled “GND” (ground). Insert one end of a black jumper wire into the GND pin.

Next, find the DIO pin on the right-hand side of the device labeled “2.” Connect one end of a red jumper wire from Pin 2. It is important that you do not use Pin 0 or Pin 1, since these pins are connected to the USB serial communication which may cause problems if they are used for this type of application.

Now connect the other ends of the black and red jumper wires to connection points on the central section of the breadboard on separate rows. Next, connect the solid-state relay to

the breadboard by connecting the red (positive) wire to a point in the same horizontal row as the red jumper wire connecting Pin 2 on the RedBoard device. Connect the black (negative) wire from the solid-state relay to a point in the same horizontal row as the black jumper wire connecting the ground pin from the RedBoard device.

In this configuration, when voltage is applied to Pin 2 from the microcontroller, current will travel through the red jumper wire, then through the red wire to the relay via the breadboard, activate the relay, and then through the black wire back to the GND pin on the RedBoard device via the breadboard, thus making a complete circuit and activating the solid-state relay in the process. You will now be able to turn the relay on or off by programming Pin 2 to turn on or off. When the relay turns on or off, any device connected to the relay's output will also turn on or off. Thus, when Pin 2 is on, any device connected to the relay output will also turn on. Your setup should look very similar to the image below:



Starting and Setting Up the Arduino Integrated Development Environment

Introduction

The Arduino integrated development environment (IDE) is a simple, yet powerful application which allows the user to create programs for Arduino which utilize the C and C++ programming libraries. The text-based language used in Arduino programming is extremely useful for data acquisition and control. While Arduino has not yet seen a particularly wide adoption in industry, the C and C++ programming languages are heavily used in industry, so learning the Arduino programming language is useful because it mimics these languages.

Installing Arduino Software and Initializing the RedBoard Device

To get the RedBoard device up and running, the latest software for Arduino Uno must be downloaded and installed on a Windows PC. To download the most recent software version, open a web browser and navigate to <http://www.arduino.cc/en/main/software>. Locate the link to the Windows Installer and click the link. If you are using a Windows 10 PC, you can also find the “Arduino IDE” app in the Microsoft app store and install it that way. Windows will open a dialog box asking what you want to do with the file. Click “Save File.” Once the .exe file has been downloaded and saved, navigate to your Downloads folder and open the file. Windows will ask you whether you want to allow the program to make changes to your computer. Allow this action. The Arduino setup utility will then open a dialog box requiring you to accept the license agreement. Click “I Agree.” Follow the prompts to finish the installation of the Arduino software.

On some computer operating systems, you may also need to install Arduino drivers separately. If your computer has trouble locating drivers, follow these instructions to install them manually. Use the USB cable to connect the RedBoard device to the PC. Open a web browser and navigate to <http://www.sparkfun.com/FTDI>. Follow the instructions appropriate to your operating system.

Starting the Arduino IDE

From the Windows desktop, select *Start>All Programs>Arduino IDE*. An initial screen identifying the software will, appear, followed automatically by a new “sketch” (program). At this point, we are ready to begin writing our first program.

Programming Arduino—Line Outputs

Introduction

The first program we will write will perform the following functions in the sequence outlined below:

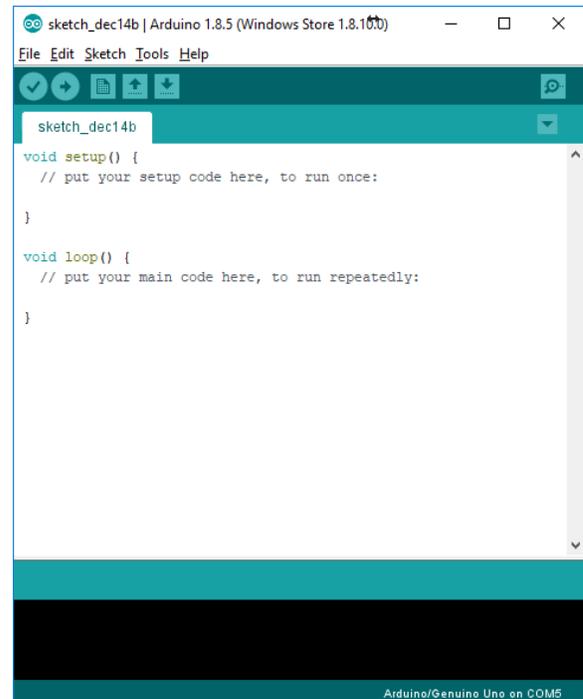
- Turn an LED on for 1 second
- Turn an LED off for 1 second
- repeat sequence

Arduino IDE Syntax

The Arduino utilizes a text based programming technique. It is therefore necessary to learn some basic grammar rules, known in programming as “syntax,” in order to successfully write a program. If our syntax in a program is incorrect, the device will not know how to interpret our commands.

In the open sketch, you’ll notice that the IDE has started us off with two commands, called “functions”: `void setup()` and `void loop()`. Each of these is followed by a set of “curly braces” that look like this: `{ }`. Curly braces act similarly to a new paragraph in English writing, telling the program that a new set of instructions resides inside the curly braces. How that set of instructions is run depends on the function which corresponds to the curly braces. For example, the “`setup()`” function tells the RedBoard that the set of instructions inside the curly braces is to be run once, and once only. The “`loop()`” function tells the device that the set of instructions inside the curly braces is to be run repeatedly in a never-ending loop. Inside the curly braces, each line of code that you write must end with a semicolon (;). This acts much like a period in English, separating one line of instruction from the next.

You’ll also see what’s called a “comment” that looks like this: “`// put your setup code here, to run once`”. A comment is anything in a line that follows `//`, and the RedBoard device will simply skip over that line. Comments are used by the programmer to remind himself or herself what different parts of the program are doing. We can write anything in a comment for our own benefit, and the device will simply ignore it.



For now, this is all the syntax we need to know. We'll learn additional rules as they come up.

Defining Variables: Integer Variables (int)

The first step in writing our program is to define names for things we will be using in our program. To do this we use the text "int", which is short for "integer." The int command assigns a "variable" (a name we define) to an integer "value" (a whole number, positive, negative, or zero). In this case, we'll use the name "LED_pin" as the name for the digital output we are programming. We could use any name such as "x" or "y," but since we will be turning an LED on and off, LED_pin makes sense, so we'll use that. Establishing an integer variable also means that we assign the name we've chosen to a specific integer value.

In this case, the integer value will correspond to the digital channel we are programming. Digital pin 13 on the RedBoard device is internally connected to an LED located near the right-hand side of the board. When Pin 13 is turned on, the LED will also turn on. So, since we want to control the LED, we will assign our variable name to digital pin 13. Defining variables comes before the program actually starts, so we'll write our code before the void setup() command.

Syntax:

```
int var = val;
```

Parameters:

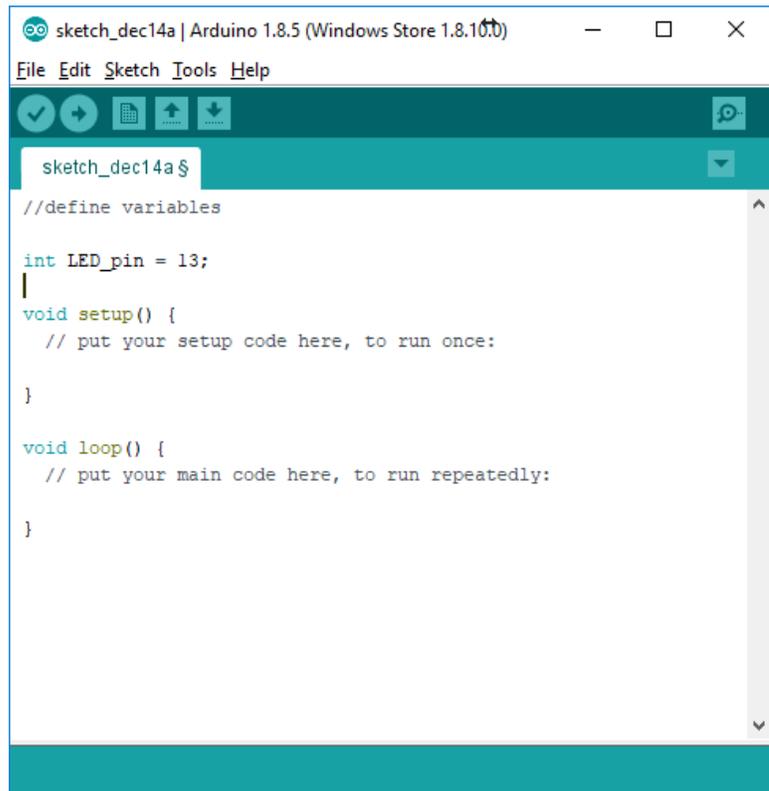
Var: variable, the name you are defining for the integer value

val: value, the integer value (...-1,0,1,2...) you are naming

Code to write:

```
int LED_pin = 13;
```

This code you have just written created a new integer variable named "LED_pin" and assigned it the value of 13, which will correspond to digital pin 13 which is connected to an LED on the RedBoard device.

A screenshot of the Arduino IDE interface. The window title is "sketch_dec14a | Arduino 1.8.5 (Windows Store 1.8.10.0)". The menu bar includes "File", "Edit", "Sketch", "Tools", and "Help". Below the menu bar is a toolbar with icons for checkmark, back, forward, upload, and download. The main editor area shows the following code:

```
sketch_dec14a $
//define variables

int LED_pin = 13;
|
void setup() {
  // put your setup code here, to run once:
}

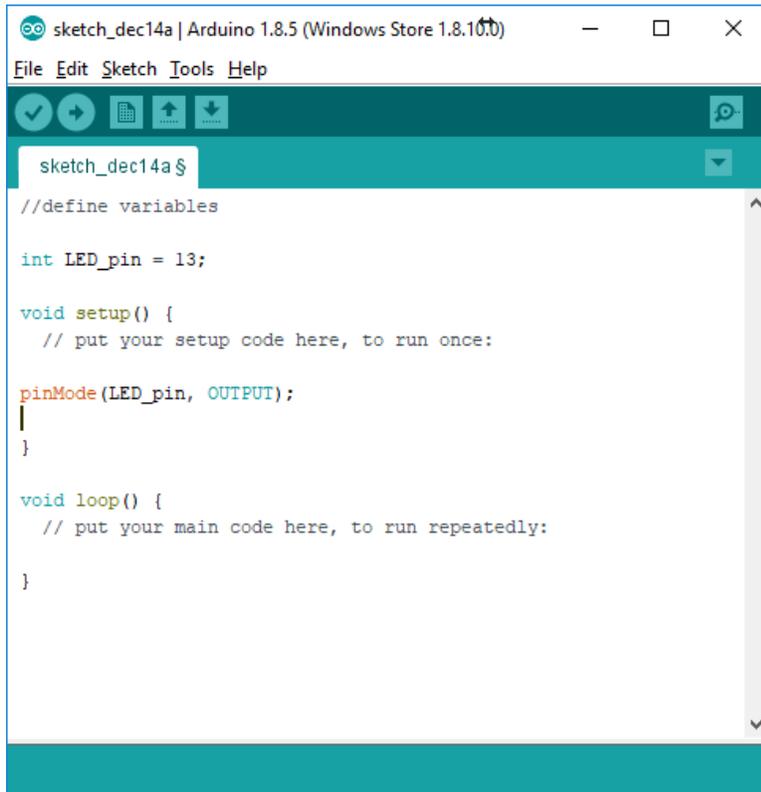
void loop() {
  // put your main code here, to run repeatedly:
}
```

Initializing DIO Pins: setup() and pinMode()

The first function in our sketch will be the `setup()` function. A function is simply a type of procedure or routine which the device will follow. The `setup()` function is initiated when the sketch starts, and runs only once at the beginning of the program. In our new sketch, the `setup()` function is already in place, and looks something like this:

```
void setup() {  
  
}
```

We will use the `setup()` function to assign the variable “LED_pin” to pin 13 as an output. To do this, we will use the `pinMode()` function. `pinMode()` simply calls out a variable that we have already defined and then assigns that variable as an input or an output. This function is what attaches our new variable to pin 13, rather than just assigning it a numerical value of 13. We will place this function inside the curly brackets, since it is part of the `setup` function and only needs to run once.

A screenshot of the Arduino IDE interface. The window title is "sketch_dec14a | Arduino 1.8.5 (Windows Store 1.8.10.0)". The menu bar includes "File", "Edit", "Sketch", "Tools", and "Help". Below the menu bar is a toolbar with icons for running, saving, and other functions. The main text area shows the following code:

```
sketch_dec14a$  
//define variables  
  
int LED_pin = 13;  
  
void setup() {  
  // put your setup code here, to run once:  
  
  pinMode(LED_pin, OUTPUT);  
}  
  
void loop() {  
  // put your main code here, to run repeatedly:  
  
}
```

Syntax:

```
pinMode(var, mode);
```

Parameters:

`var`: variable name assigned to the numeric value of the pin whose mode you wish to set

`mode`: INPUT, OUTPUT, or INPUT_PULLUP, sets the mode of the pin

Code to Write:

```
pinMode(LED_pin,  
OUTPUT);
```

At this point, we have created a variable called “LED_pin,” and assigned it to digital pin 13 as a

digital output.

Programming a Loop: loop()

The next step in creating our sketch is to program a loop. A loop function simply tells the device to run the set of functions inside the curly brackets in order until it reaches the end of the loop, and then to start over again at the beginning of the functions inside the curly brackets. In this way, we can create a program which runs continuously as long as

the device has power. In our sketch, the loop function should already be in place by default, so we just need to add functions inside the curly brackets.

Turning on an output: `digitalWrite()`

In order to get some action out of our digital output pin, we need to tell it to do something. This is done by using the `digitalWrite()` function. This function “writes” a value of HIGH (on) or LOW (off) to a specific digital output pin. Writing a value of “HIGH” to LED_pin will cause Pin 13 to turn on, and also cause the connected LED to turn on. We want this to happen as part of our loop, so we’ll write this part of our code inside the curly brackets following the `loop()` function.

Syntax:

```
digitalWrite(var,  
value)
```

Parameters:

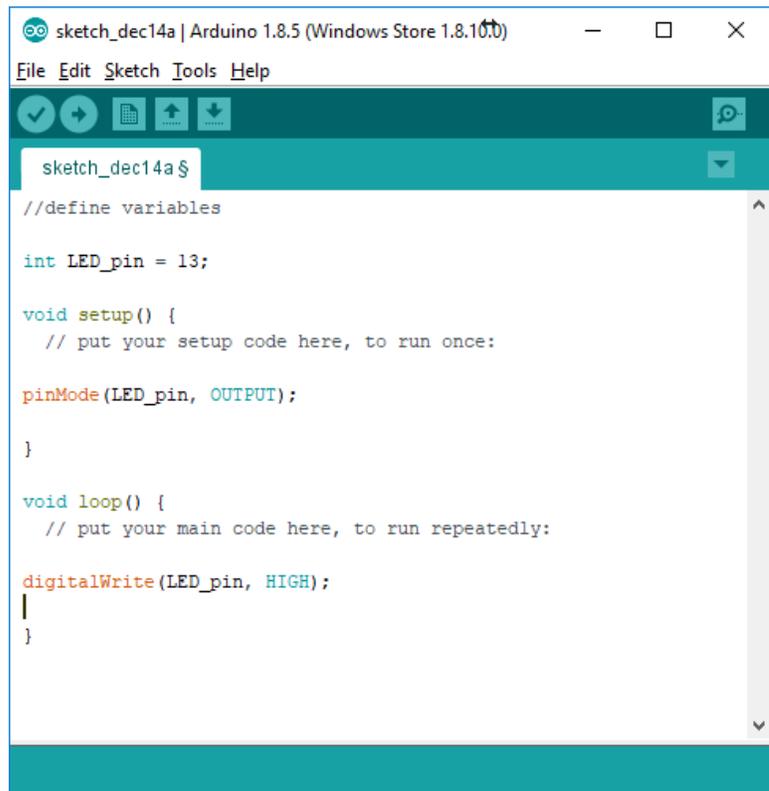
var: variable name of the pin you wish to turn on or off.

value: HIGH or LOW
(HIGH = 5V DC, LOW = 0V DC)

Code to Write:

```
digitalWrite(LED_pin,  
HIGH);
```

This will turn the LED_pin on, applying 5 volts to digital pin 13, lighting the internally connected LED.



```
sketch_dec14a | Arduino 1.8.5 (Windows Store 1.8.10.0)
File Edit Sketch Tools Help

sketch_dec14a $
//define variables

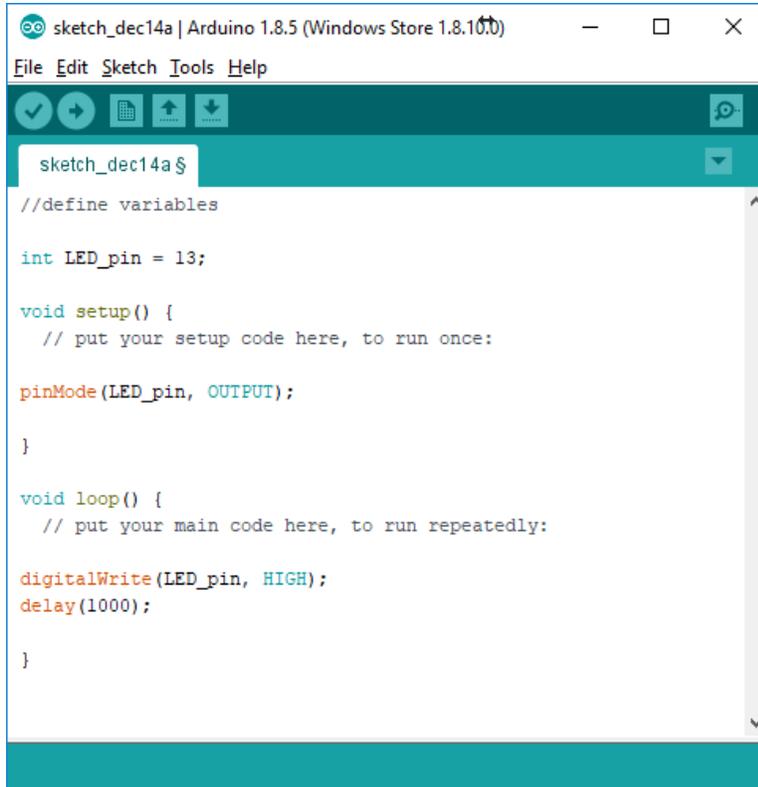
int LED_pin = 13;

void setup() {
  // put your setup code here, to run once:
  pinMode(LED_pin, OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(LED_pin, HIGH);
}
```

Programming a Delay: `delay()`

Next, we want to tell the device how long to keep our LED lit. We can do this using a `delay()` function. A delay simply tells the device to wait for a specified amount of time before moving on to the next line of code. The time is measured in milliseconds (ms), so for our one second delay we will use a value of 1000 ms.



```
//define variables

int LED_pin = 13;

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

  pinMode(LED_pin, OUTPUT);
}

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

  digitalWrite(LED_pin, HIGH);
  delay(1000);
}
```

Syntax:

`delay (milliseconds)`

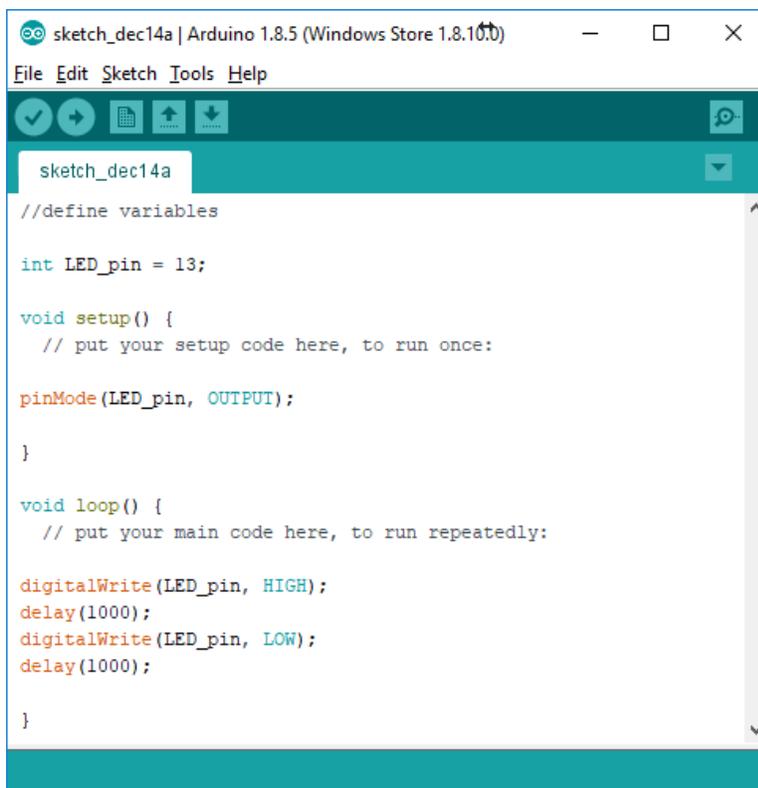
Parameters:

`milliseconds`: the number of milliseconds to pause, e.g. 500 ms for ½ second.

Code to Write:

`delay (1000) ;`

This will cause the program to pause for 1 second while the LED is lit.



```
//define variables

int LED_pin = 13;

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

  pinMode(LED_pin, OUTPUT);
}

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

  digitalWrite(LED_pin, HIGH);
  delay(1000);
  digitalWrite(LED_pin, LOW);
  delay(1000);
}
```

Next, use `digitalWrite` function to turn `LED_pin` off. Following the 1 second delay, program a `digitalWrite` function to turn the `LED_pin` LOW. Then program a delay function to wait an additional 1 second while the LED is off.

At this point, the program will run each step in the loop in sequence until it reaches the end of the `loop()` function, at which point it will start over at the beginning of the `loop()` function. This will result in the LED blinking on and off at 1 second intervals for as long as the RedBoard device has power connected to it.

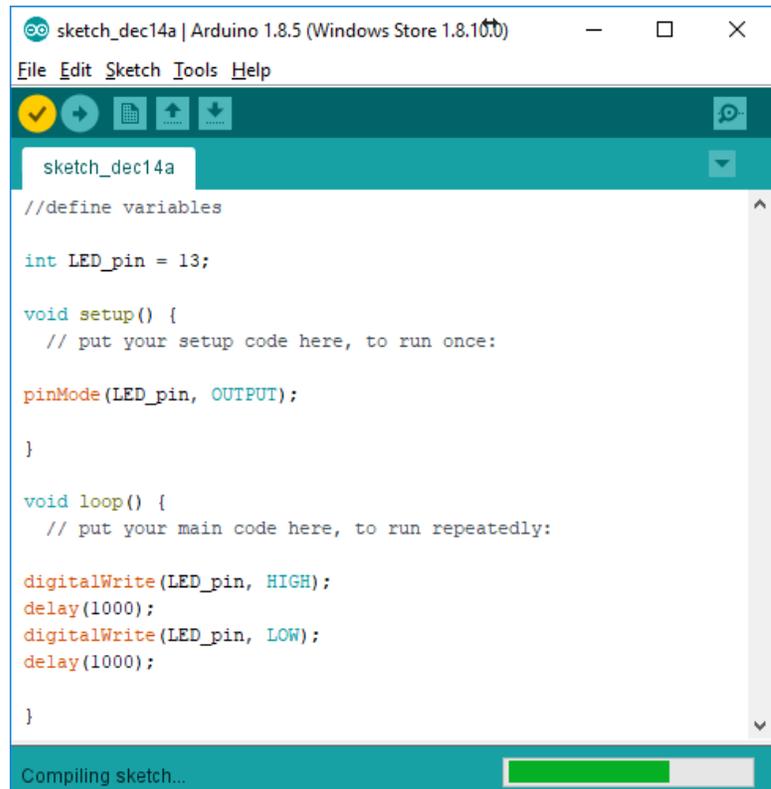
Verify and Upload

Now we need to compile our code and check for errors. We do this by clicking the “Verify” button, which looks like a green checkmark in the top left corner of the Arduino IDE window.

The IDE will then check your sketch for errors, and compile the program. Compiling means that the IDE is converting your program into a lower-level form in which the RedBoard can execute the program. If your program has errors, you will get an error message and the errors will be displayed in red at the bottom of your window.

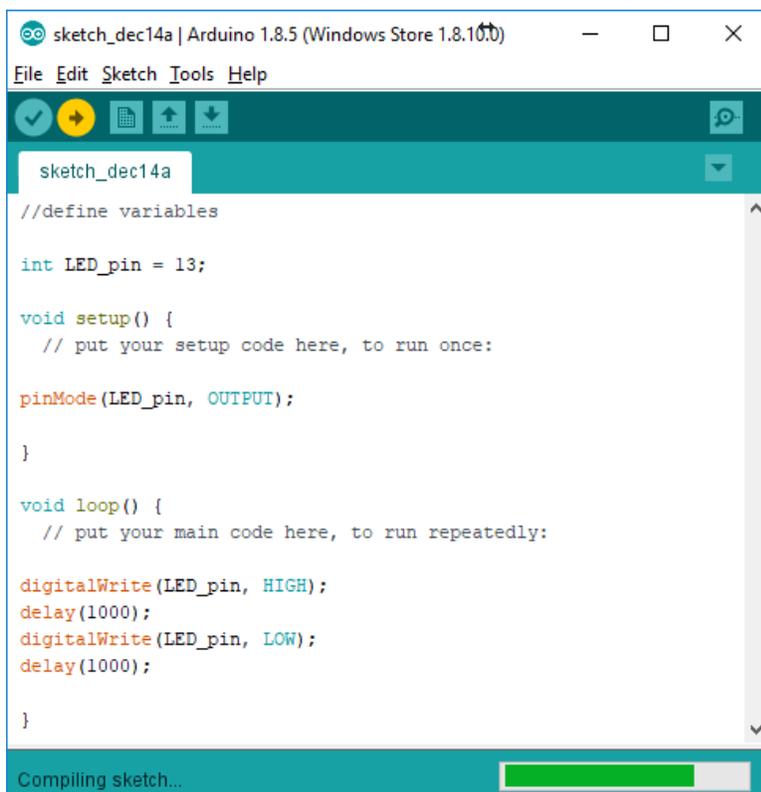
Common errors include missing semicolons, misplaced curly brackets/parentheses, and incorrectly formatted functions.

If you have errors, check your sketch thoroughly to ensure it looks like that shown in the image. The IDE will also



provide helpful hints in red text at the bottom of your screen, and highlight the line of code which has the error.

Once you have verified the sketch, make sure the RedBoard is connected to your computer via USB cable, and then click the “Upload” button, which looks like an arrow next to the “Verify” button in the top left corner of your window. You should see blinking lights on the RedBoard which show that the program is being uploaded from the PC to the device. If there is an error on the upload, go to the “Tools” menu at the top of the



window, select “Port: COM__”, and then choose a different COM port. You may have to try a couple of different ports to get the right one. Once the upload is complete, RedBoard will immediately begin running the program, so you should see a blue LED on the device blinking on for 1 second and off for 1 second repeatedly. If not, check your program to make sure that you followed the instructions exactly.

Finishing the Program

Now that we have tested our program and verified that it is working correctly, modify the sketch to complete the sequence shown near the beginning of this activity. To do this, all you should need to do is change the number of milliseconds in your delay functions.

- Turn the LED on for 2 seconds
- Turn the LED off for 3 seconds
- repeat sequence

When you are finished, have your teacher check your program to make sure it works correctly.

Team Assignment

As a team, write a program to control the solid state relay (connected to Pin 2) to control a sprayer pump to turn on for 2 seconds every 5 minutes. When you have constructed your aeroponic vertical garden, you will use this program to mist your plants' roots. At that point, you may need to adjust the misting schedule to promote the best plant growth.