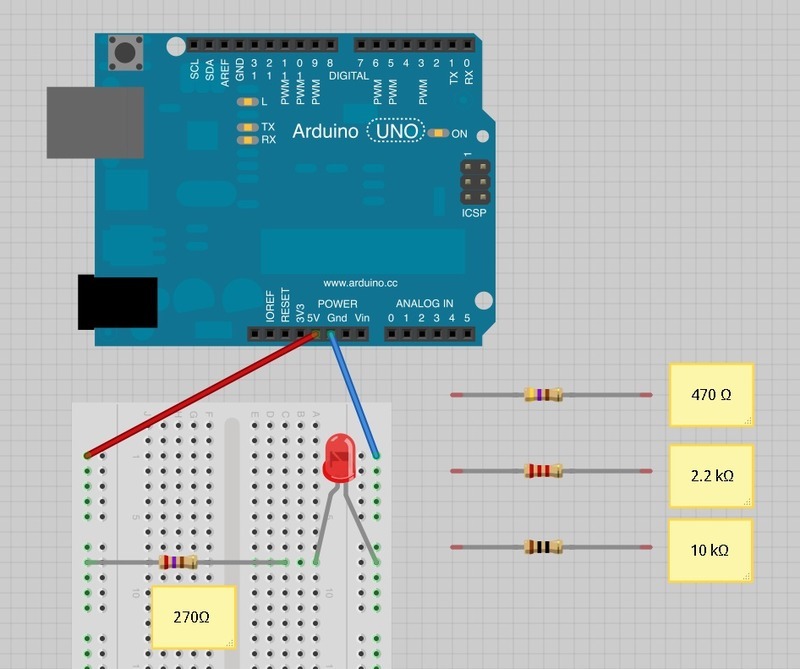
**Electronics & Circuit Diagrams Presentation**

**Getting Started with a Breadboard Exercise Sheet**

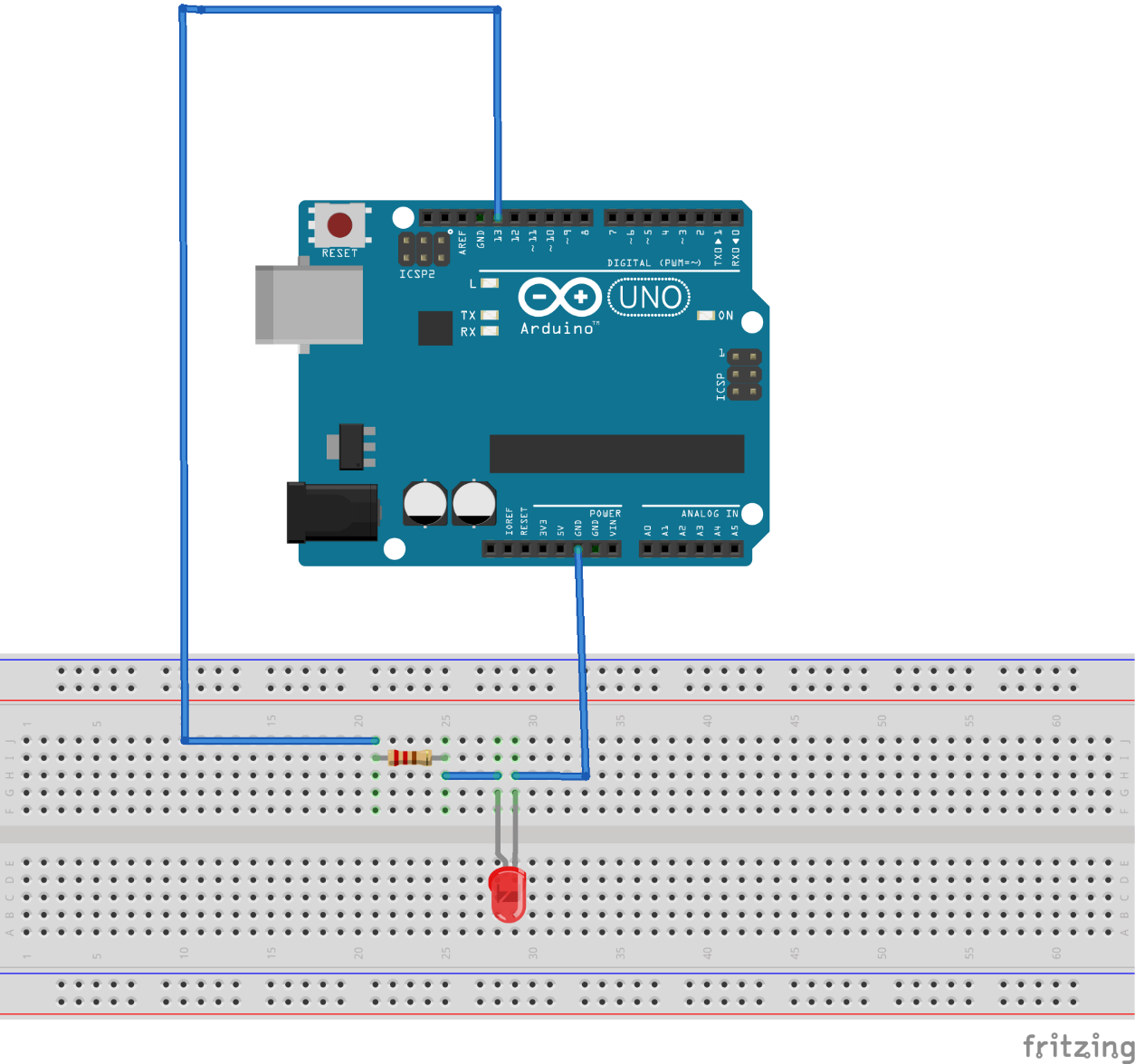
**Using the breadboard for the first time**

The Arduino is a convenient source of 5 Volts, that we will use to provide power to the LED and resistor. You do not need to do anything with your Arduino, except plug it into a USB cable. Set up your breadboard as shown below. This will provide power to light the LED. Remember the short connector on the LED is a cathode and should be connected to a common ground (Negative). No code is needed.



**Make a Simple LED Circuit**

Use a jumper cables and a resister of 220 ohm for this project. Will will use a pin and our code to send a voltage to the LED.



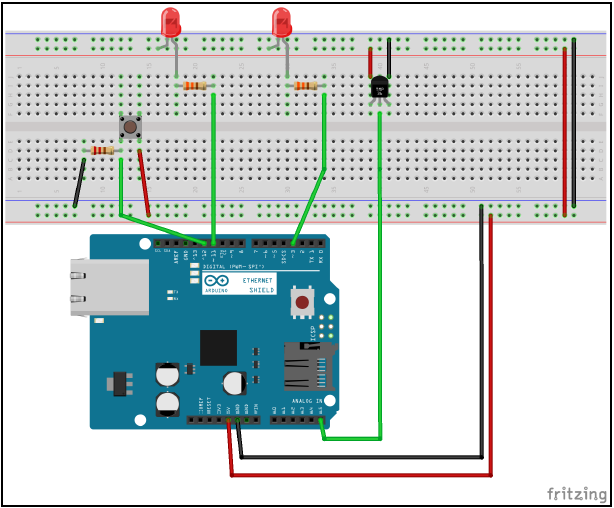
Use the code below to enable pin 13 and perform a digitalWrite to turn the light on providing power through pin 13.

|  |
| --- |
| int led = 13; // the pin the LED is connected to  void setup() {  pinMode(led, OUTPUT) // Declare the LED as an output  }  void loop() {  digitalWrite(led, HIGH) // Turn the LED on  } |

Now change the code to get the LED blinking:

|  |
| --- |
| int led = 13; // the pin the LED is connected to  void setup() {  pinMode(led, OUTPUT) // Declare the LED as an output  }  void loop() {  digitalWrite(led, HIGH) // Turn the LED on  delay(1000)// Wait for 1000 milliseconds (1 second)  digitalWrite(led, LOW) // Turn the LED off (By lowering voltage)  delay(1000); // wait for a second  } |

You will now need to create this prototype as explained in the presentation to complete the remaining exercises in this sheet. This example diagram shows a prototype that will have one button that will be used to turn a LED on or off, another LED that we can turn on or off and a TMP36 temperature sensor. 3 resistors of 220 ohm’s and jumper cables have also been used:



**Digital write**

To set the value of a digital pin in the Arduino programming language, we use the digitalWrite() function. This function takes the following syntax:

digitalWrite(pin, value);

The digitalWrite() function accepts two parameters, where the first one is the pin number and the second is the value to set. We should use either HIGH or LOW when setting the value of a digital pin.

The following code shows how to do this:

|  |
| --- |
| digitalWrite(LED\_ONE, HIGH); delay(500);  digitalWrite(LED\_ONE, LOW);  delay(500); |

In the preceding code, we set the pin defined by the LED\_ONE constant too HIGH and then pause for half a second. The delay() function in the Arduino programming language pauses the execution of the sketch for a certain amount of time. The time for this function is in milliseconds. After the delay() function we then set the pin defined by the LED\_ONE constant too LOW and wait another half a second before looping back to the beginning. The previous code can be used in the loop() function to blink an LED; however, before we do that we need to define the LED\_ONE constant and also set the pin mode. Let's look at the full sketch required to blink an LED.

|  |
| --- |
| #define LED\_ONE 11  void setup() {  pinMode(LED\_ONE, OUTPUT);  }  void loop() {  digitalWrite(LED\_ONE, HIGH);  delay(500);  digitalWrite(LED\_ONE, LOW);  delay(500); } |

This code starts off by defining the LED\_ONE constant and setting to 11. The pin mode for the LED\_ONE pin is then set in the setup() function. Finally, the code that will cause the LED to blink is added to the loop() function.

If you connect the prototype and ran this code, you should see one of the LEDs blinking. Now that we know how to write to a digital pin, let's see how we can read the value of one.

**Digital read**

To read the value of a digital pin in the Arduino programming language, we use the digitalRead() function. This function takes the following syntax:

digitalRead(pin);

The digitalRead() function takes one parameter, which is the number of the digital pin to read, and will return an integer value. The following code shows how we can use the digitalRead() function to read one of the digital pins on the Arduino:

|  |
| --- |
| int val = digitalRead(BUTTON\_ONE); |

With this code, the digitalRead() function will return the value of the pin defined by the BUTTON\_ONE constant and put that value into the variable named val. The val variable is defined to be an integer. However, the digitalRead() function will only return a 0 or a 1. We can use the same HIGH and LOW constants that we saw in the Digital write section to see if the pin is either high or low. Using these constants are preferred and makes your code more readable. Now let's see how we can use the digitalRead() function to read the status of a button.

The following code will read the status of the button from the prototype:

|  |
| --- |
| #define BUTTON\_ONE 12  void setup() {  Serial.begin(9600); //Sets serial data speed. Use 9600.  pinMode(BUTTON\_ONE, INPUT); }  void loop() {  int val = digitalRead(BUTTON\_ONE);  if (val == HIGH) {  Serial.println("Button HIGH"); }  else {  Serial.println("Button LOW"); } } |

This code starts off by defining the BUTTON\_ONE constant and setting it to 12. The serial monitor and the pin mode for the pin that the button is connected to are both configured in the setup() function. Within the loop button, the digitalRead() function is used to read the pin and the if statement is used to compare the value that was returned with the HIGH constant. If they are equal, then the message Button HIGH is sent to the serial monitor otherwise the message Button LOW is sent.

If this code is run on the prototype then you should see one of the two messages being printed to the serial monitor depending if the button is pressed or not. Now let's see how we can write to an analog pin on the Arduino.

**Analog write**

Analog values are written to the Arduino with the Pulse-Width Modulation (PWM) pins. Previously we have looked at what PWM is and how they work. On most Arduino boards the PWM pins are configured for pins 3, 5, 6, 9, 10, and 11; however, the Arduino Mega has significantly more pins available for PWM functionality. To perform an analog write, we use the analogWrite() function, which takes the following syntax:

analogWrite(pin, value);

The analogWrite() function accepts two parameters, where the first one is the pin number and the second is the value to set. The value for the analogWrite() function can range from 0 to 255.

Let's look at a sample sketch to see how we can use the analogWrite() function to fade a led in and out:

|  |
| --- |
| #define LED\_ONE 11  int val = 0; int change = 5;  void setup() {  pinMode(LED\_ONE, OUTPUT);  }  void loop() {  val += change;  if (val > 250 || val < 5)  { change \*= -1; }  analogWrite(LED\_ONE, val);  delay(100); } |

This code starts off by defining a LED\_ONE constant with a value of 11. This will be the pin that the LED is connected to. There are also two global variables defined, both of the integer type, named val and change. The val integer will store the current value of the analog pin, and the change integer will store how much the val integer should change each loop. The pin defined by the LED\_ONE constant is set to output mode within the setup() function. This will enable us to write to the pin and change the brightness of the LED connected to the pin.

The loop() function starts off by adding the change variable to the val variable, and the result is stored in the val variable. If the value of the val variable is greater than 250 or less than 5 we multiple the change variable by -1. This causes the change variable to rotate between 5 and -5, which causes the val variable to increase or decrease each loop. Finally, the value of the val variable is written to the pin defined by the LED\_ONE constant, and then there is a short delay before looping back.

**Analog Read**

Analog read We read the value from an analog pin using the analogRead() function. This function will return a value between 0 and 1023. This means that if the sensor is returning the full voltage of 5V, then the analogRead() function will return a value 1023, which results in a value of 0.0049V per unit (we will use this number in the sample code). The following code shows the syntax for the analogRead() function:

analogRead(pin);

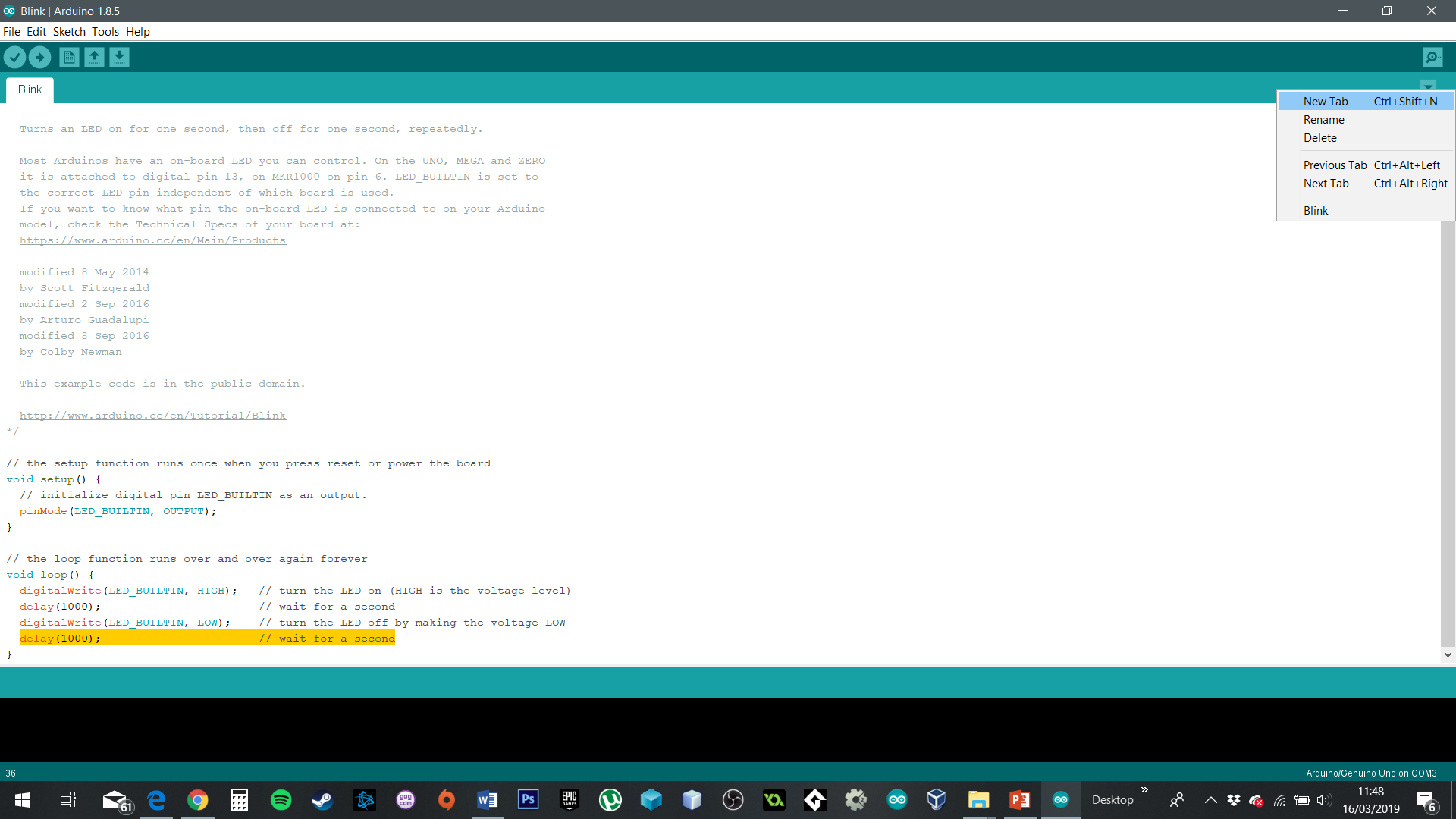
The analogRead() function takes one parameter which is the pin to read from. The following code uses the analogRead() function with a tmp36 temperature sensor to determine the current temperature:

|  |
| --- |
| #define TEMP\_PIN 5  void setup() {  Serial.begin(9600); }  void loop() {  int pinValue = analogRead(TEMP\_PIN);  double voltage = pinValue \* 0.0049;  double tempC = (voltage - .5) \* 100.0;  double tempF = (tempC \* 1.8) + 32;  Serial.print(tempC);  Serial.print(" - ");  Serial.println(tempF);  delay(2000); } |

The preceding code starts off by defining the pin that the temperature sensor is attached to which is the analog pin 5. The setup() function configures the serial monitor so the application can print the temperature to it. The loop() function begins by reading the analog pin and storing the value in the pinValue variable. To convert this value to the actual voltage, we multiply it by the 0.0049V value that we saw earlier in this section. If we look at the datasheet for the tmp36 temperature sensor, we will determine that the (voltage - .5) \*100.0 is the correct formula to calculate the temperature in Celsius. We can then use the standard formula (celsiusTemp \*1.8) + 32 to determine the temperature in Fahrenheit. Finally, we print these values to the serial monitor and delay for two seconds before beginning the loop again.

**Working with tabs and headers**

To create a Tab and add it to your sketch. See the top right arrow pointing down. Left click it as shown below.



When creating a new tab, the first thing we need to decide is what is going to the tab. For example in this section, we will create two new tabs. One will be named led.h and the other led. The led.h file will contain the constant definition, and the led file will contain code. When we create a tab with the .h extension we are creating, what is known in the C language, a header file. A header file is a file that contains declarations and macro definitions. These tabs can then be included in the normal code tabs. In the next section, we will see another type of tab which is the cpp tab. Once the new tabs are created, add the following code to the led.h tab:

|  |
| --- |
| #ifndef LED\_H  #define LED\_H  #define LED\_ONE 3  #define LED\_TWO 11  #endif |

This code will define two constants, which are the pin header numbers for the two LEDs on the prototype. The #ifndef and #endif ensure that the header file is imported only once within any tab. The #ifndef looks to see if the LED\_H constant is defined, and if not then it includes the code between the #ifndef and #endif. Now in the led tab add the following code:

|  |
| --- |
| void blink\_led(int led) {  digitalWrite(led, HIGH);  delay(500);  digitalWrite(led, LOW);  delay(500); } |

The blink\_led() function contains a single parameter, which will be the pin for the LED that we wish to blink. The function itself will turn the LED on for 1/2 a second and then turn it off. Now in the main tab, we will need to include an #include statement at the top of the tab to include the led.h header file. The following code shows how to do this:

|  |
| --- |
| #include "led.h" |

The #include statement will take a header file and includes it in the tab, allowing us to use the definitions within our code. If we attempted to use one of the constants within our code but forgot to include the header file, we would receive an error that the constant was not declared in this scope meaning the compiler was unable to find the declarations for the constant. If we are adding a header file from the sketch, we are working in, the name of the header file is surrounded by double quotes. If we include a header file from a separate library the name will be surrounded by the less than and greater than signs. You will see this as you use third-party libraries.

In the loop() function, we will want to call the blink\_led() function from the led tab. One thing to note here is we only need to include the #include statement for the header file and not for the tab that contains the code. The following shows the code for the main tab:

|  |
| --- |
| #include "led.h"  void setup() { // put your setup code here, to run once:  pinMode(LED\_ONE, OUTPUT);  pinMode(LED\_TWO, OUTPUT);  [ 137 ]  }  void loop() { // put your main code here, to run repeatedly:  blink\_led(LED\_ONE);  delay(1000);  blink\_led(LED\_TWO); } |

Now if you connect the prototype you should see the LEDs blink one after the other. Dividing your code between separate tabs is a great way to organize it when working with larger projects. This makes it a lot easier to maintain and organize your code.

Classes are usually used when creating libraries for the Arduino. While creating libraries is most likely beyond the scope of this unit, it is good to know what classes are and how to use them because you will need to make use of them for certain projects.