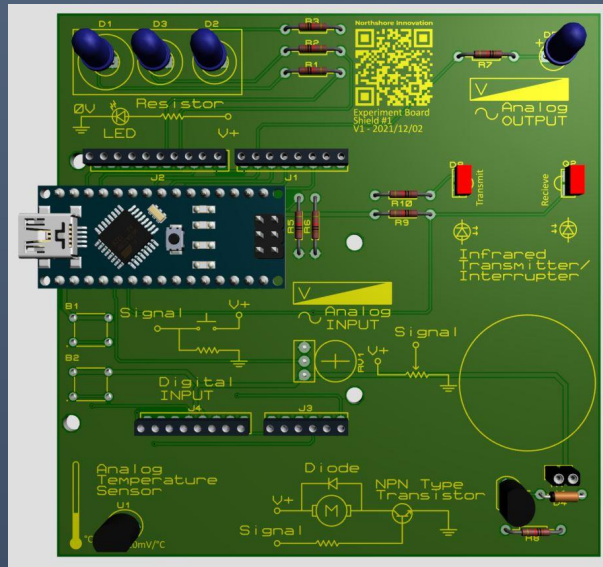
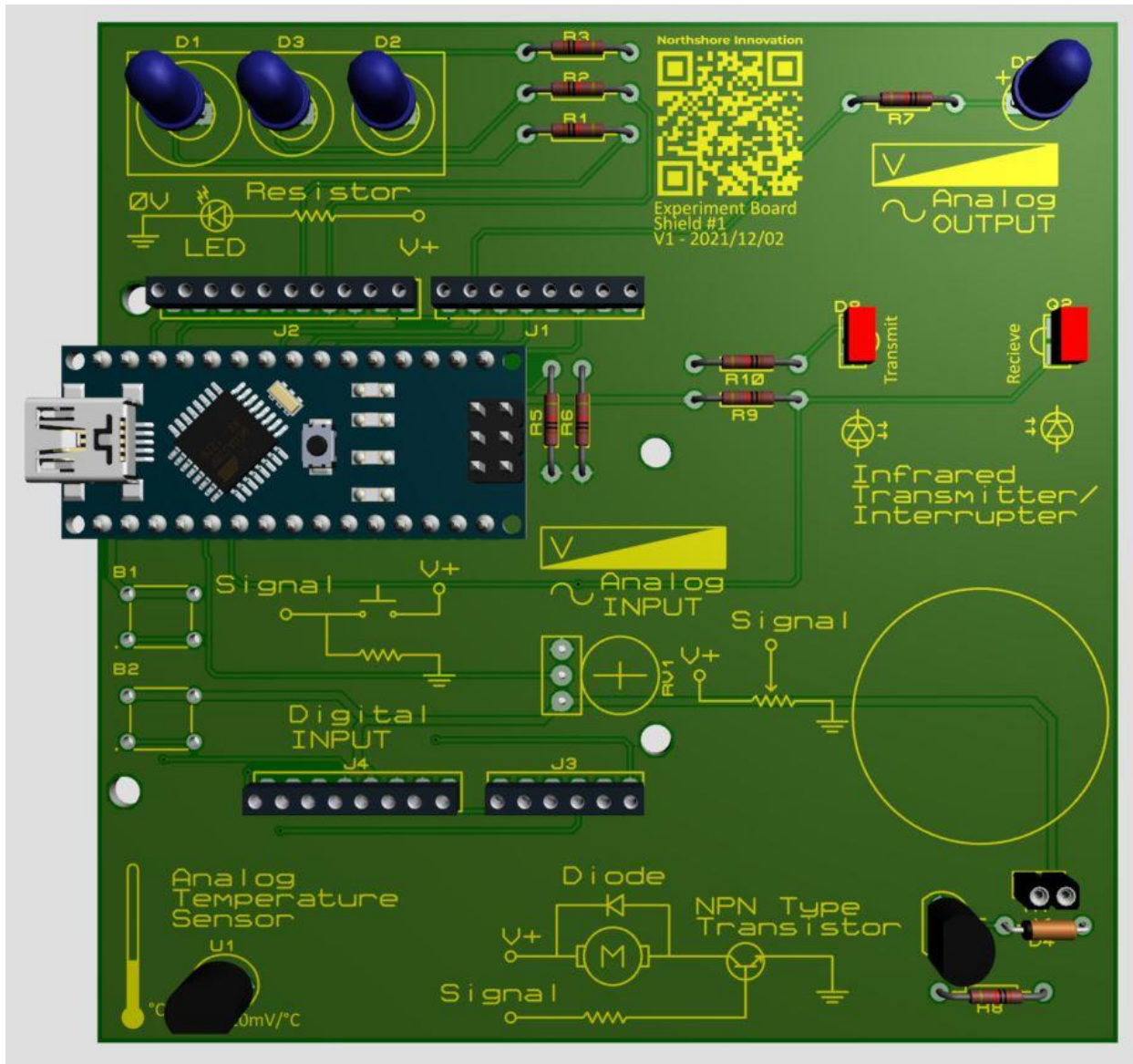


# Northshore Innovation



## Experiment Board V1.0

SHIELD #1  
SCOTT CARD



Digital				Analog			
I/O	Scratch	Arduino	Shield	I/O	Scratch	Arduino	Shield
Output	Digital 10	DIG12	D2 (GREEN)	Output	Analog 6	DIG6 (PWM)	D5 (BLUE)
Output	Digital 11	DIG11	D3 (YELLOW)	Output	Analog 9	DIG9 (PWM)	D8 (Infrared diode)
Output	Digital 12	DIG10	D1 (RED)	Input	Analog IN 0	AN_IN0	RV1 (Potentiometer)
Input	Digital 2	DIG2	B2 (TOP)	Input	Analog IN 1	AN_IN1	Q2 (Infrared receiver)
Input	Digital 3	DIG3	B1 (BOTTOM)	Input	Analog IN 2	AN_IN2	U1 (Temperature Sensor)

## Contents

Getting Started.....	3
Foreword- What is Scratch for Arduino(S4A)?.....	3
Before plugging the Arduino.....	3
In the Kit.....	4
What is included .....	4
What you will need .....	4
How the experiments work. ....	4
Introduction to Scratch .....	5
1 <sup>st</sup> Project -Hello world for Hardware.....	7
2 <sup>nd</sup> Project -The switch.....	8
3 <sup>rd</sup> Project -The Conditional Loop .....	9
4 <sup>th</sup> Project -Bright Light!.....	10
5 <sup>th</sup> Project -What temperature is it?.....	11
6 <sup>th</sup> Project -The better timer .....	13
7 <sup>th</sup> Project -Traffic Light.....	15
Continue With or Without Scratch .....	17
Glossary.....	18
Notes:.....	19

## Getting Started

### Foreword- What is Scratch for Arduino(S4A)?

Scratch is a programming language that uses visual blocks to create code. Developed by MIT to educate school-aged children, it has become a valuable tool in schools throughout the world.

The Arduino is a family of development boards that allows users to produce projects and learn the hardware aspect of computer science quickly and reliably. With a limited number of inputs, outputs, communication protocols, this low-cost option is a great entry point to embedded processors. The Arduino programming language is a modified version of C++ with many built in libraries.

S4A was developed to allow students to interact with the Arduino using the same Scratch programming format, thereby avoiding the need to learn higher level programming languages.

Meant to complement Scratch and the Arduino, this experiment board will allow the user to interact with the processor and the physical world in a controlled manner. Experiments are designed to show the different aspects of the embedded processor as they discover digital, analog, inputs, outputs, and programming methods.

Once finished the experiments, the Arduino and Scratch can be used to interface with other devices or used with the experiment board to create new projects. The Experiment board may also be used directly with the Arduino and programmed using the Arduino Integrated Design Environment (IDE).

### Before plugging the Arduino

The Arduino (or clone) requires a driver to work on the computer. The first step is to install the driver, then the various programs required, including the Arduino IDE.



***NOTE: Skipping this step may permanently destroy your hardware!***

1. Driver
  - a. [Elegoo NANO driver](#)
  - b. [Uno FTDI driver](#) (usually does not need a manual driver install)
2. Arduino IDE
  - a. [Windows](#)
  - b. [Mac OS X](#)
3. S4A
  - a. [Windows](#)
  - b. [Mac OS X](#)
4. Supplying your own Arduino or refreshing an existing.
  - a. <http://s4a.cat/>

## In the Kit

### What is included

Development board with one of the following:

- Header (Provide your own Arduino)
- Arduino Nano (ELEGOO)
- Arduino UNO
- Kit for assembly (Arduino not included)

### What you will need

- USB cable
- Windows PC or MAC

### How the experiments work.

Each experiment is intended to learn a new concept. Each concept may require 2 or more new tools to be learned. Mistakes when learning to program are inevitable but, learning to troubleshoot them is a valuable tool and can be more beneficial than a flawless execution of the instructions. That said, when first discovering new concepts, it is important to have early and frequent success. If an experiment does not work, step back and work through the steps to look for any mistakes.

As the experiments progress, they become more difficult and rely on tools learned in the previous examples. For this reason, it is not suggested to do the experiments out of order.

To perform the experiment, create the code using the blocks exactly as shown in each step. The result of each block will be described and tested as we go. Once the experiment is done, feel free to experiment with the code.

For younger student, guidance or separate learning plan may be required, depending on the level being taught to. Caution must be taken not to frustrate the learner and it is critical that the first projects are successful to prevent them from feeling overwhelmed.

Math concepts, such as the conversions required by the Analog to Digital Converter (ADC) will most likely be too advanced for most students. However, it is a necessary procedure to allow the user to get access to the sensors.

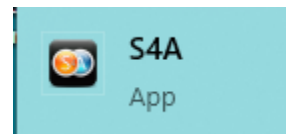
Enjoy the adventure!

## Introduction to Scratch

Scratch uses *Code Blocks* to build a program as opposed to a text-based language. This makes the introduction of coding easy for novices as there is no need to worry about context and structure. Simply pull in the block you want, snap them together in the correct order and run the program to see the results. Made a mistake? No problem! Trouble shooting is the best learning tool and will teach valuable lessons itself. Feel free to use this section as a reference while coding.

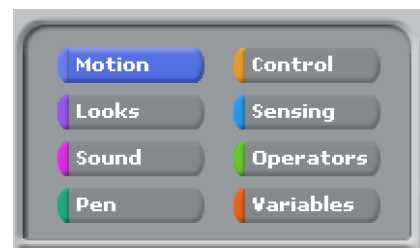
### Launching Scratch for Arduino

Once Scratch for Arduino has been installed and configured (See: *Before plugging the Arduino*) you can access the program by searching programs in your START menu (Windows Key). Type S4A in the start menu for a shortcut!



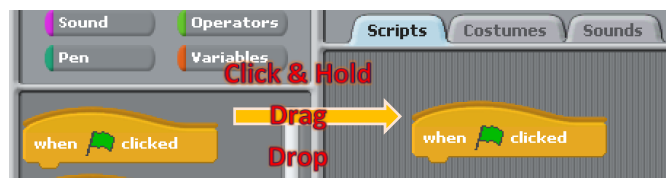
### Modules

There are 8 modules within S4A, each with its own set of commands. They are grouped into relevant command types. Feel free to experiment with each of them by selecting the tabs shown here. Note that the colours of the tabs match the colour of the commands. This will help you navigate the instructions for the experiments.



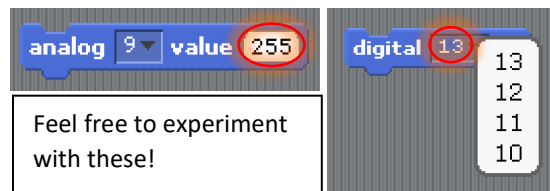
### Commands

Selecting a command or instruction is as simple as click and hold the item you want. Next drag it to the program space (canvas) and release. Notice that the blocks have key shapes allowing them to snap together. Some blocks must be used at the beginning, the end or within other blocks.



### Pull Downs and Variable Fields

Many of the command that S4A use to interact with the outside world will have editable fields. These may select the pin, like the pull down shown here for *Digital 10~13*. It can also be used for setting a value as shown for *Analog 9* which has a value of 255.



### Duplicating

Once a code block or section of code is created, it is possible to copy it for reuse. Simply right click on the block and select *duplicate*. Selection may also include variables, values, operators, or code within a loop.

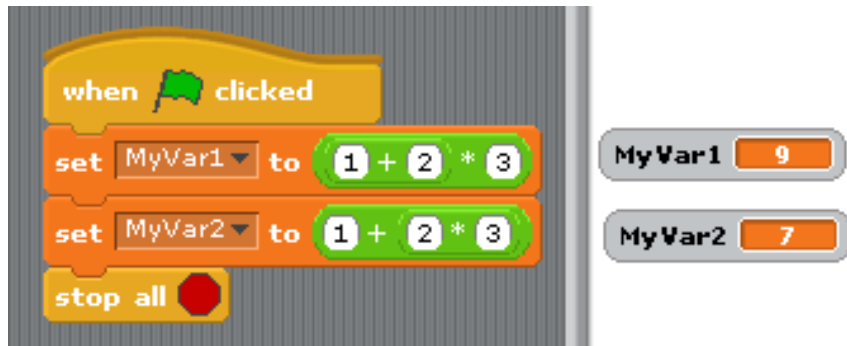


### Deleting Code

Made a mistake or have unneeded code? Deleting code blocks or sections of code is as easy as dragging them back off the canvas. Any code that is not attached to the set of commands with a beginning will be ignored.

### Nesting instructions

Many instructions, especially the arithmetic functions, will require code blocks to inside other code blocks. For example, setting a variable to a mathematical equation or multiple math functions being performed on the same variable. Each function acts as a bracketed operation.



\*Note: that the order of operations is dependent on the structure, not the rules of algebra.

$$MyVar1 = (x + y) \times z$$

$$MyVar2 = x + (y \times z)$$

### Running the Code

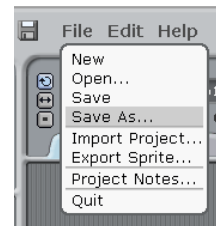
Once you are ready to test or run your code, simply follow the instructions of the start block. For example: click the start or press the start key.

### Stopping the Code

If there is no return (like in a *forever loop*) the code will terminate on its own. If there is a loop, clicking on the start block will stop the program and the hardware will remain in its current state.

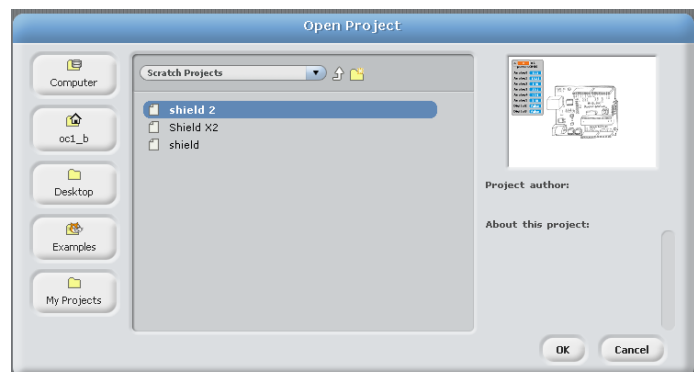
### Saving Code

At any time, feel free to save your code. This is a great way to ensure that you can recover your file should the program crash. It is also good to keep your files as a reference for the future use.



### Opening Saved Code

Once a file has been saved or you wish to open an example file, use the File>Open... and choose the desired file. If files are stored in a different directory, you may navigate from the default as well as creating your own.



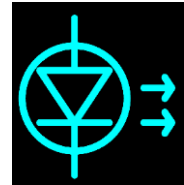
# 1<sup>st</sup> Project -Hello world for Hardware

## Step #1. Turn on an LED

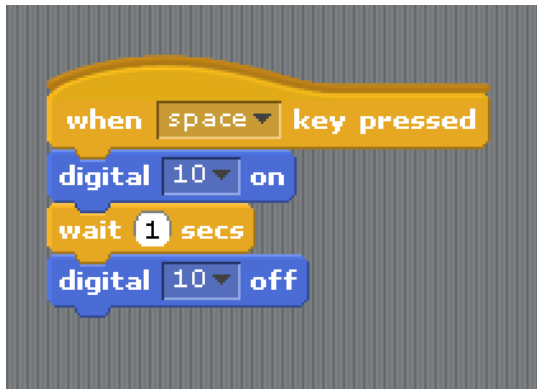


What is an LED?

An LED is a Light Emitting Diode. When a voltage is applied (in the right direction) it will light up. LED are used everywhere today for many things. This is the symbol for an LED. The GREEN LED should now be lit.

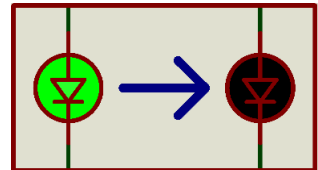


## Step #2. Turn an LED on then off

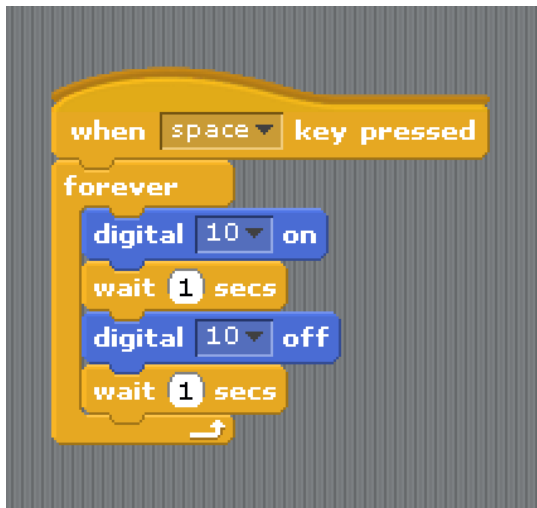


What are states?

In electronics we say that digital has only two states, on or off (1 or 0). This means that we if we set a signal to 1 it is referred to a high or on. When we set it to 0 is if low or off. It is important in this experiment to add a *wait* command to allow us to see the LED is on. Without the delay, the led turn off too fast for us to see it!



## Step #3. Make a Loop



What is the Loop?

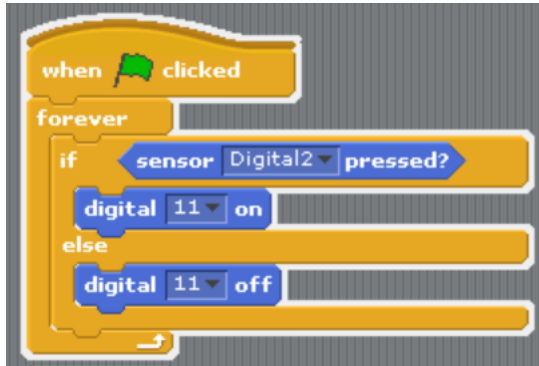
A loop is a statement that tells the program to repeat a part of the code. The *forever* loop will continue to do the instructions inside the command until the program is stopped. This code will make the GREEN LED tun on and off.

Try to change the values of the wait command to see what happens.



## 2<sup>nd</sup> Project -The switch

### Step #1 Digital input using if/else

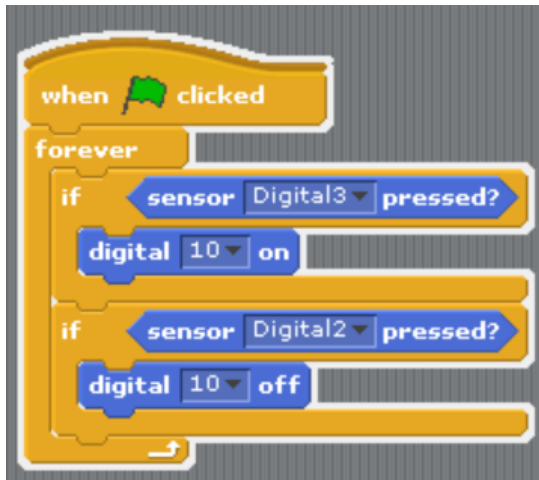


#### Input and Output?

Input refers to information going onto the computer. In this case the information is provided by button and is either on or off (1 or 0). The output for our project is the LED and can be either on or off (1 or 0).

In this code, the state of the output can also be reversed, so that the LED is on when the button is not pushed and off when it is.

### Step #2 Latch using IF



#### What is a latch?

A latch condition is when a state is set by a signal until some other signal is given. In this case when button B1 is pushed the LED will turn on, and remain on, until button B2 is pushed.

In this case, we have created an ON/OFF switch. You can see that the light will turn on, and stay on, until the command to turn it off is made.

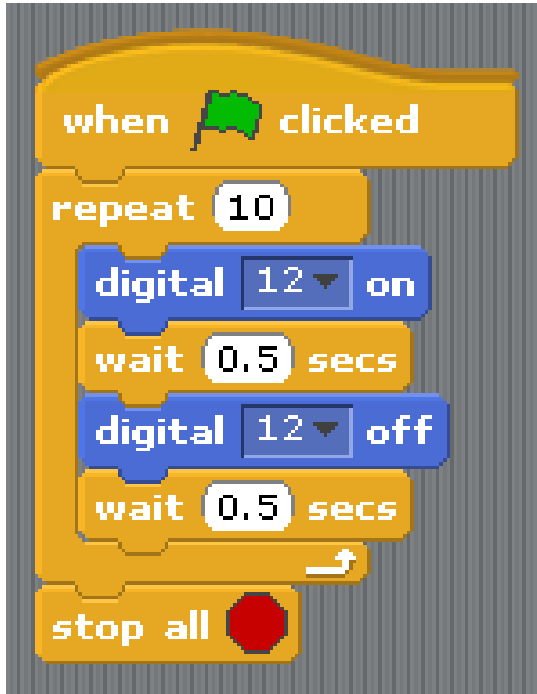


1

<sup>1</sup> [https://www.automationdirect.com/adc/overview/catalog/pushbuttons\\_-z-\\_switches\\_-z-\\_indicators/start\\_-z-\\_stop\\_combo\\_control\\_stations](https://www.automationdirect.com/adc/overview/catalog/pushbuttons_-z-_switches_-z-_indicators/start_-z-_stop_combo_control_stations)

### 3<sup>rd</sup> Project -The Conditional Loop

#### Step #1 Conditional loop



```
when clicked
repeat 10
  digital 12 on
  wait 0.5 secs
  digital 12 off
  wait 0.5 secs
stop all
```

There are several types of conditional loops in coding. Two that are available in Scratch are Repeat [*n*] and Repeat until <*n*>. In this example we will use the repeat loop. The value given will determine how many times the instructions will be repeated. In this example the green LED will turn on and off 10 times

Not all conditions need to be physical. The signal could be a value, like a counter, or even the passing of time.

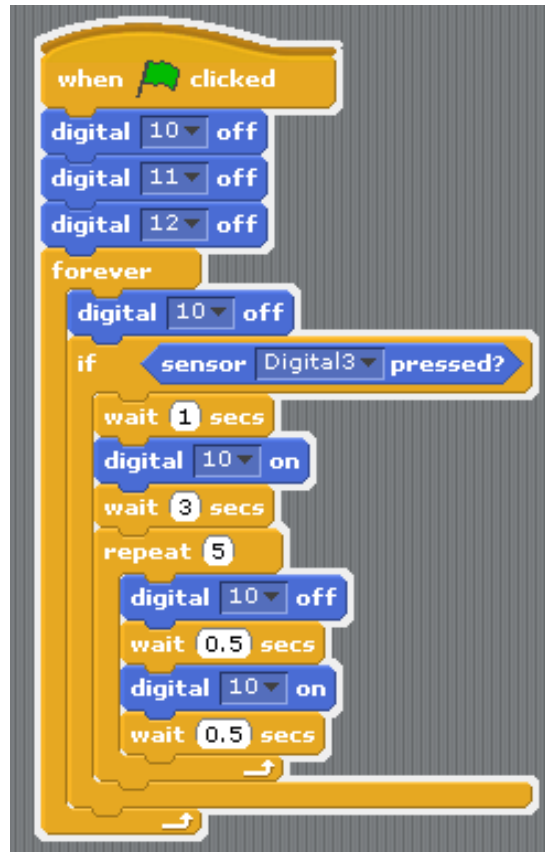
Try adding the *wait* command to unlatch the LED from Project #2. This will give you practice saving, opening, and closing files. You can also *Save As* with a modified file name.



#### Step #2

For an added challenge, combine project #2 and #3 and add a flashing LED before turning it off. This example is like what you may see at a cross walk!

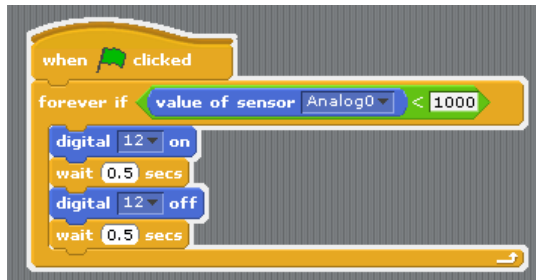
\*Note: The preconditioning of the output states before beginning the main loop. This is good programming practice as it sets all the variable to a known state.



```
when clicked
digital 10 off
digital 11 off
digital 12 off
forever
  digital 10 off
  if sensor Digital3 pressed?
    wait 1 secs
    digital 10 on
    wait 3 secs
    repeat 5
      digital 10 off
      wait 0.5 secs
      digital 10 on
      wait 0.5 secs
```

## 4<sup>th</sup> Project -Bright Light!

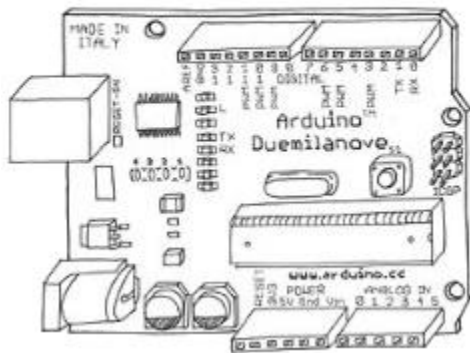
### Step #1 Operators



Operators are arithmetic functions within the code that may include addition (+), subtraction (-), multiplication (\*), and division (/) as well as round, and more complicated operations. They also include comparisons such as greater than (>), less than (<), and equal to. Boolean operations (such as: and, or, and not) are also available but not covered in these experiments.

### Step #2 Analog input (potentiometer)

Arduino 1 value of sensor Analog0 1023



What is analog?

As we saw with the switch experiment, digital is either on or off (1 or 0). But analog signal is between on and off. They are often used to measure values that have a range (minimum and maximum) and are often in a unit call a volt. The computer converts this voltage to a digital signal. In the case of the project, the voltage ranges from 0 to 5 volts and that is represented as 0 to 1023.

Adjust the potentiometer (Analog Input) to see the value change.

### Step #3 Analog output (LED)



In this experiment, we read the value of the potentiometer and change the value so it can be sent to the LED. The value must be an integer between 0 and 255 but the analog read can be 0-1023. Therefore, we round the result of the analog read divided by five.

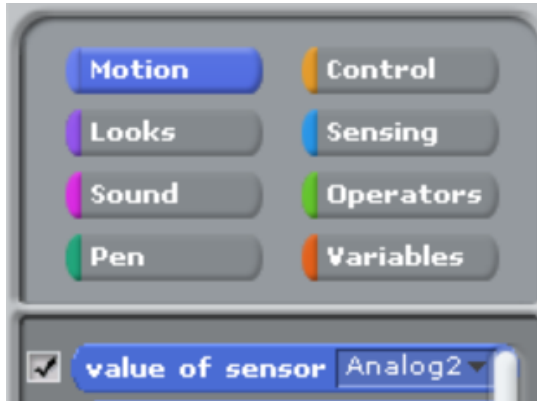


2

<sup>2</sup> [https://media.digikey.com/Photos/Adafruit%20Industries%20LLC/MFG\\_3990.jpg](https://media.digikey.com/Photos/Adafruit%20Industries%20LLC/MFG_3990.jpg)

# 5<sup>th</sup> Project -What temperature is it?

## Step #1 Sensors

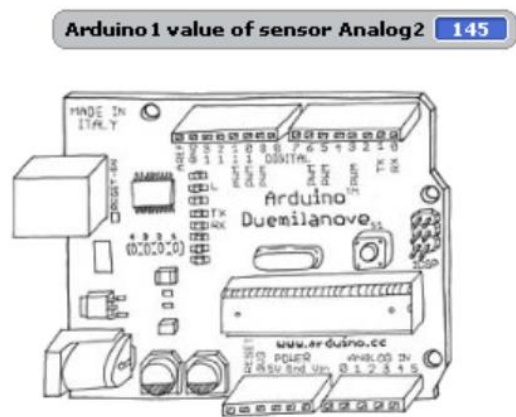


Sensors are devices that allow the computer to sense the environment around them. Some are as simple as a button but could also include: light, temperature, movement, direction, sound, conductivity, or almost anything imaginable.

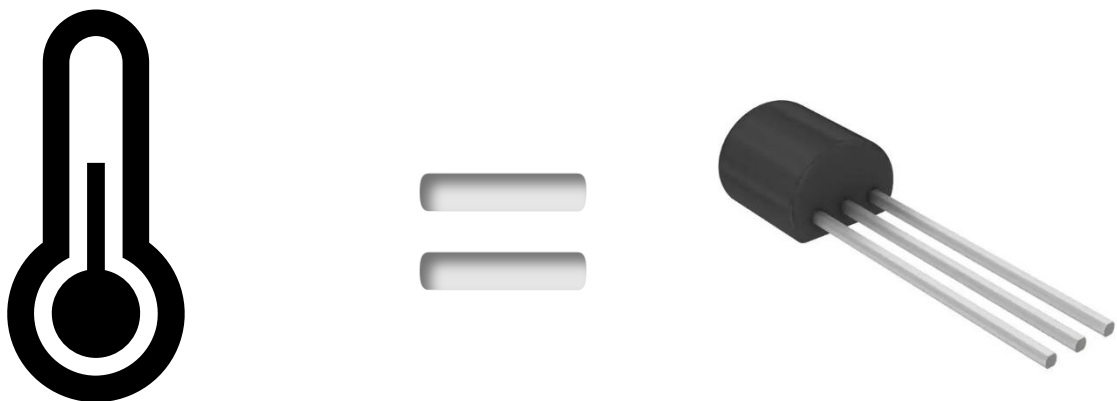
Many Sensors used are analog devices with output an analog signal that is related to the level being sensed. This means for a temperature sensor, as the temperature increases, so does the output voltage. Provided on the development board is a temperature sensor. The output of this sensor is connected to Analog 2.

\*NOTE: the formula to convert temperature [in °C] to the output voltage is:

$$V_{OUT} = Temperature[°C] * 0.01 + 0.5$$



3



<sup>3</sup> [https://media.digikey.com/Renders/~/~/Pkg.Case%20or%20Series/TO-92-3\(StandardBody\),TO-226\\_straightlead.jpg](https://media.digikey.com/Renders/~/~/Pkg.Case%20or%20Series/TO-92-3(StandardBody),TO-226_straightlead.jpg)

## Step #2 Variable



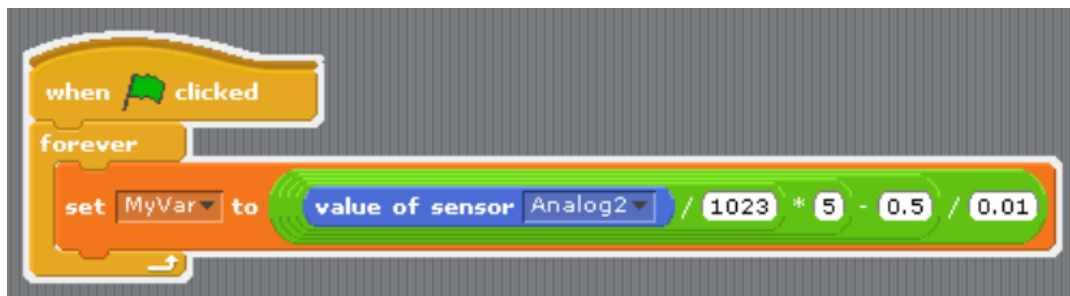
Variables can be thought of as a place to store a value. This variable can be passed from one function to another as well as be stored or displayed. To create a variable, select *Make a variable* then give it a logical name. One technique programmers use is call Cammel case. This uses capital letters to separate words.

For example: myFirstVariable

### Math

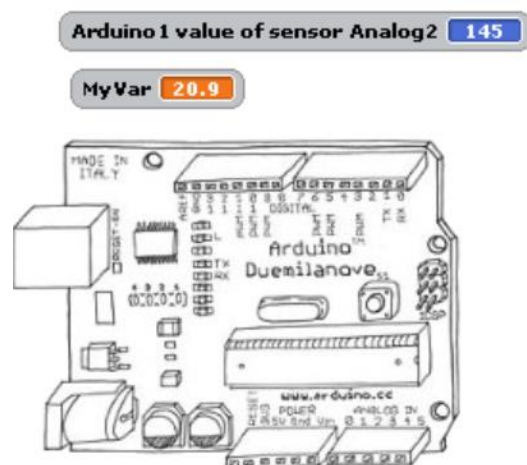
Computers are the ideal tool for doing math, as a matter of fact, that is why they were invented! We can do simple math or more complicated calculations as needed. In the case below, we convert the voltage (represented as an ADC value of *Analog 2*) into a meaningful unit – temperature, in °C.

\*Note: Be care full to nest the operations correctly! There are a lot of them.



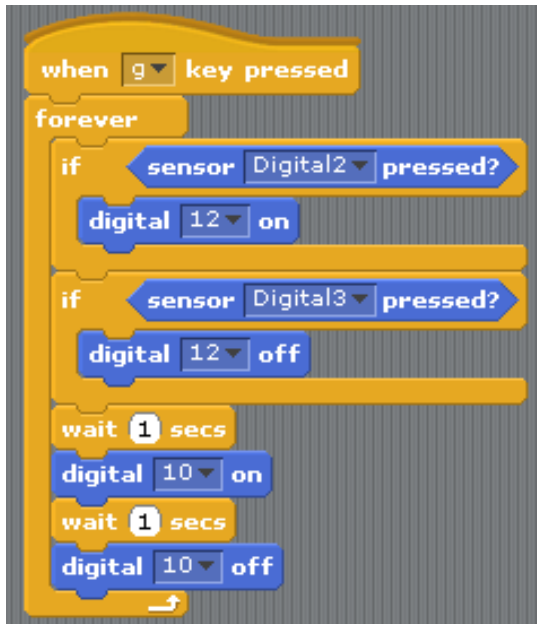
If the checkbox is enabled next to the variable you created, you will now be able to see your value. Typical room temperature will be about 20°C. If this value is not close, check the formula above.

Once you are sure the sensor is operating correctly, feel free to gently touch the sensor to even pinch it with your fingers. You will notice that the temperature increases and will go back down once you let go.



## 6<sup>th</sup> Project -The better timer

### Step #1 The Problem



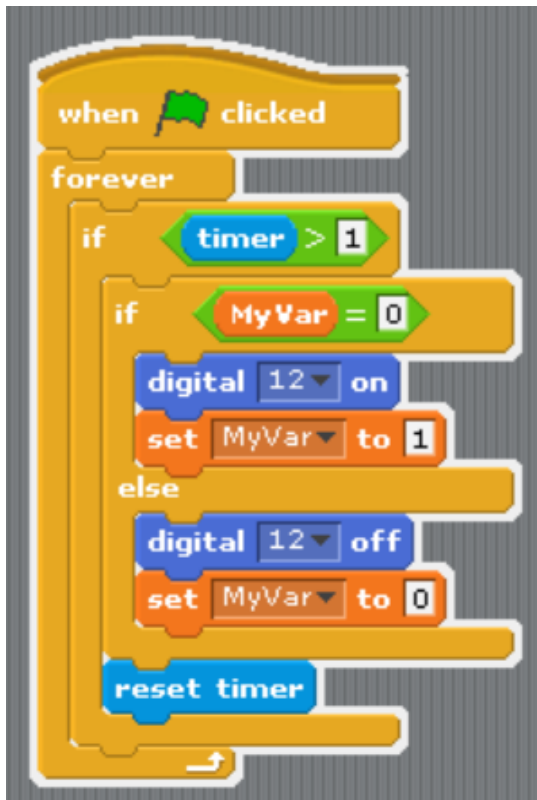
Modern computers are capable of doing many things at the same time, but simple machines only have a single *core* to perform its instructions. This means that if you instruct the processor to wait, it will stop and can do no other functions.

In this example, we can see that the LED will turn on and off when the buttons are pushed. However, the second LED also turns on and off every two seconds.

During the flashing of the LED, notice that the button press will have no effect.

\*HINT: For this project, try loading project #2 as a starter file. Be sure to do a *→File → Save as* so you do not overwrite your work.

### Step #2 The Timer

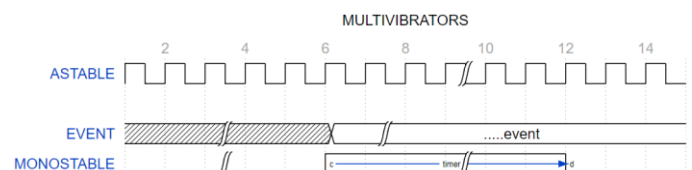


Most processors have something called a timer. This is a clock that will continue to run in the background, regardless of what else is happening. Scratch is supplied with one timer that we can use.

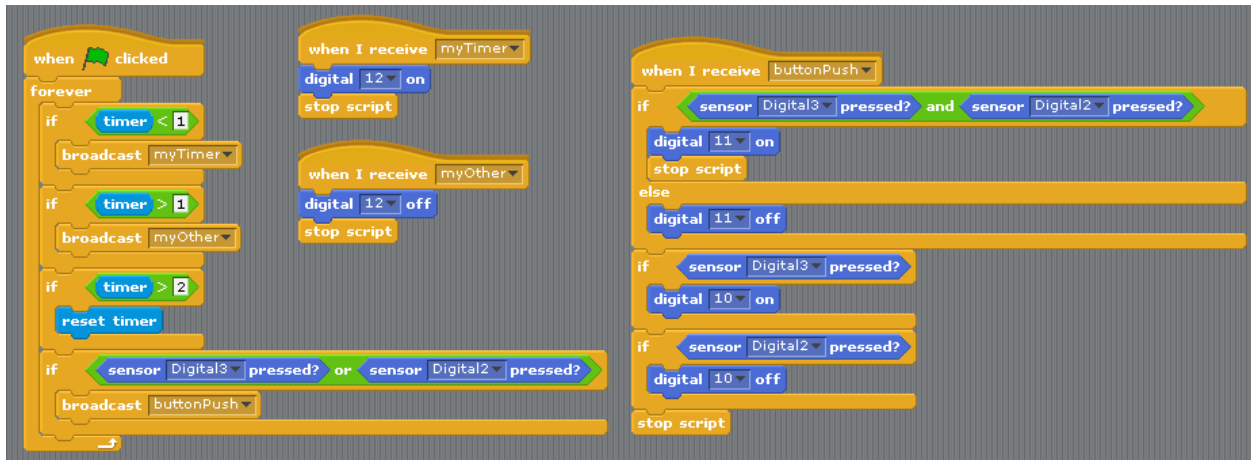
The timer will count the number of seconds and can be measured in fractions of a second. By using an if statement and a comparison operators ( $\leq$ ) to evaluate the timer value.

Once the timer has *expired*, it is important to reset it to zero if it is to be cyclic or *astable multivibrator*. The timer can also be used as a one-time event or *monostable*.

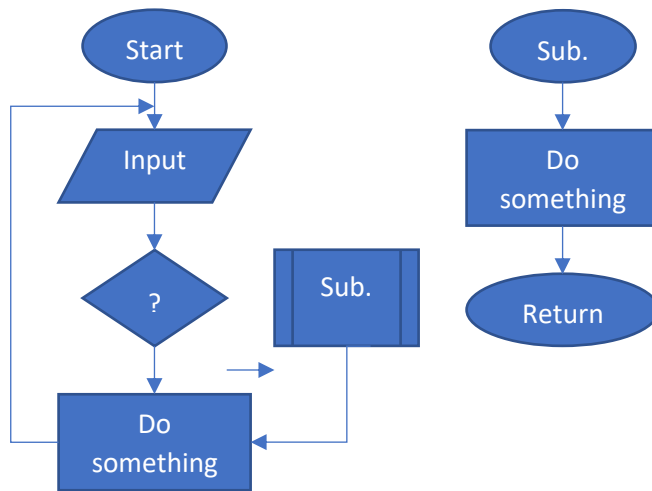
Use this code to replace the wait section of your project. Note that the button functions now work almost immediately.



### Step #3 The Subroutine



When programs become more complicated or you have several things that can happen at the same time, programmers use subroutines. A subroutine is a dependent piece of code that can be called from many different sources (saving space) and makes your code easier to read.



In this project, we have created several sub routines that are called from the main forever loop dependent on the time. The first is before the time reaches 1 second, which sets the green LED on. Once the timer is greater than 1 second, the second sub routine is called which turns off the LED. Once the timer has exceeded 2 seconds, the timer is reset in the main loop. This could also happen in a sub routine.

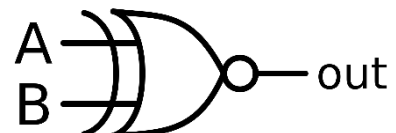
These are simple subroutines with a single function however, they could have more complicated sets of instructions, like the next example.

### Step #4 Boolean operations

The final if statement uses a logic called Boolean. This is a digital concept where we have a yes or no, on or off, 1 or 0 state. At its most time we can check the status of a *bit*. For example, is Digital2 on? We also can check the opposite by using the *not*. In our project we are using an if statement to check if either button is pushed. Once in the subroutine, we can determine if both are pushed or either one. Depending on the condition of the buttons, different actions may be taken.

\*Note: that the pressing of the button does not interfere with the LED flashing or vice versa. It is also noteworthy that the stop script is added to the one subroutine. This will stop the script from performing the rest of the operation.

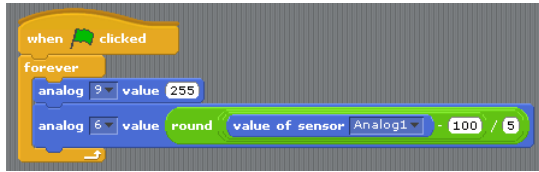
\*Bonus: This combination of *OR* and *AND* is called an *XOR* or *exclusive OR*. The inverse is called an *XNOR*.





# 7<sup>th</sup> Project -Traffic Light

## Step #1 Photo interrupter



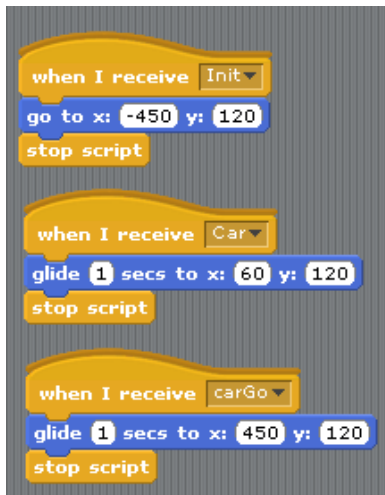
Not all light is visible by people. Many devices, like TV remotes, use a beam of invisible light to communicate. In this example we are turning on the Infrared led and measuring the result on the receiver.

## Step #2 Add car sprite?



Scratch is capable of graphics and user interfaces. One such tool is the use of sprites. New sprites can be loaded from other sources, created, or loaded from the library. For this experiment, load an existing sprite from the *Transportation* directory. Chose the vehicle that best suits you!

## Step #3 Program Car Sprite Routines

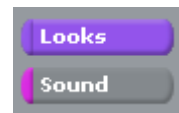


We will now create three sub routines: one for when the program is initiated, one for the car at the stop light, and one for the car leaving.

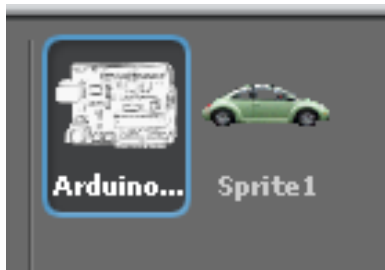
The values in the *glide* command represent the sprites' location on the screen and the secs determines how long it will take to travel there.

These may be modified to personalize the project.

\*Bonus: Text and sounds can be added using the looks and sounds icons. Feel free to experiment with these tools!



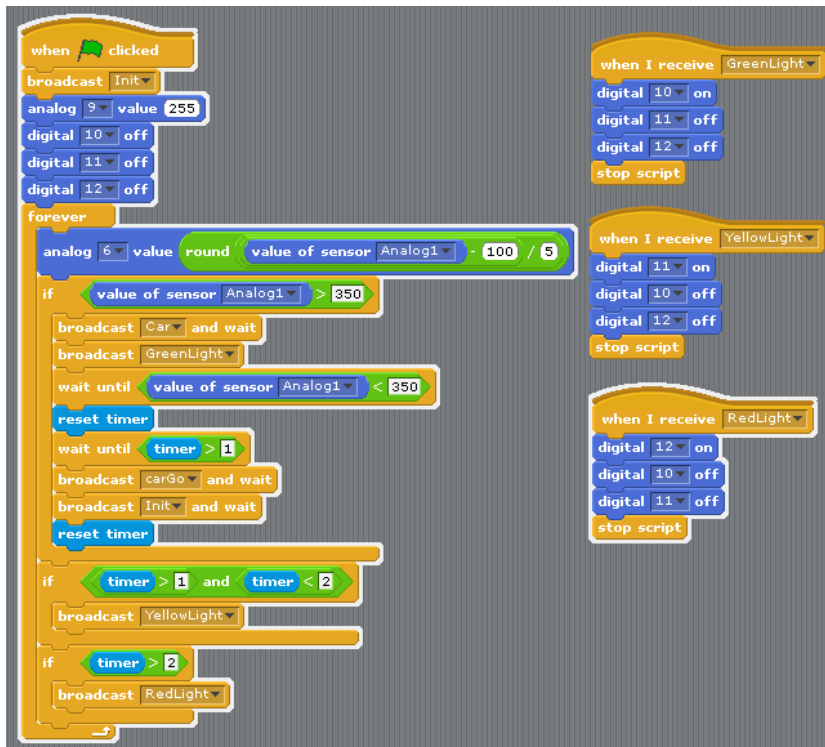
## Step #4 Return to Main Strite



Once finished moving the car and adding any other features you wish, return to the Arduino sprite by clicking on the icon.



## Step #5 Main Sprite Program with Subroutines



In the main program, we start by initializing the system. Broadcasting “Init” will place our sprite off the screen, we turn on the IR LED, and turn off all of the visible LEDs.

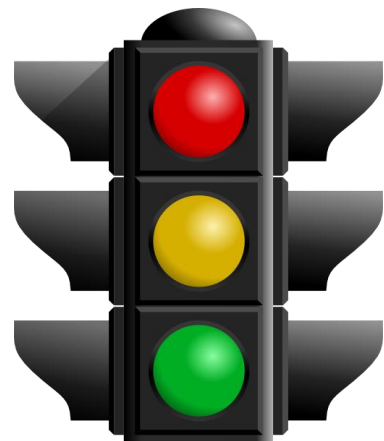
We then enter our forever loop. This is called the main loop in conventional programming and is where the program should stay. We set the blue led to mirror the IR receiver to help visualize what is happening. Place a finger between the transmitter and receiver and you will see that the LED becomes brighter.

The first if statement determines if the IR beam is broken. This would indicate that a car is in the intersection. We now call the sub routine to move the car into location. Once it has moved, we call the subroutine to turn the green light on as well as turn off any other traffic lights.

We now come to the wait command. The program will halt here until the car moves. Remove your finger and the car will pull away. Note that this is called “blocking code” and is usually frowned upon by programmers as it prevents other functions from occurring.

Once the intersection is cleared, car drives off and then is reset to its home location. The timer is reset, and the program now leaves the first if command.

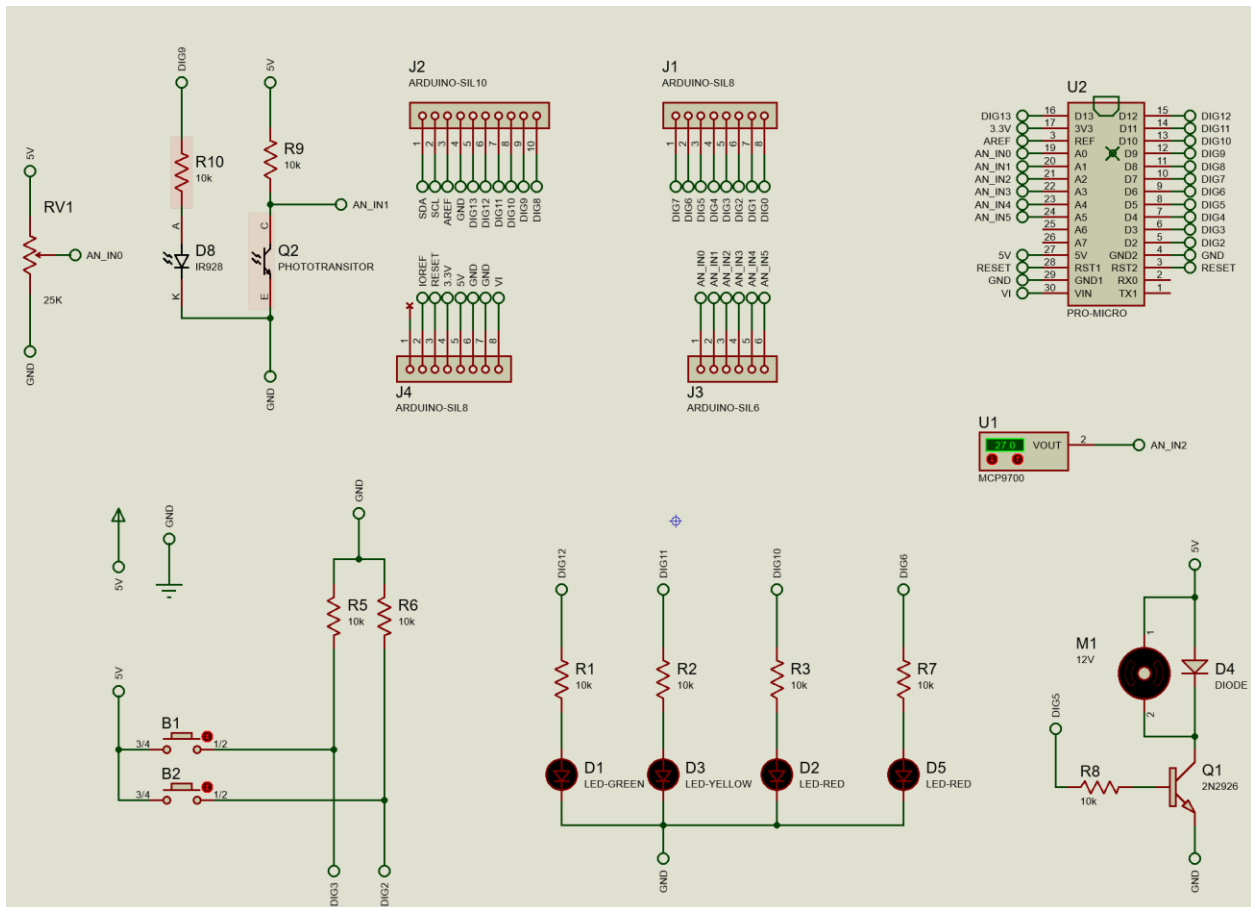
The next if statements evaluates if the timer is between 1 and two seconds. If true, the sub routine for the yellow light is called. If the timer is greater than two seconds, the red-light subroutine is called by the next if statement. These two if statements are not blocking commands. This means that if at any time a vehicle comes into the intersection, the program will abort the process in favor of the green light routine.<sup>4</sup>



<sup>4</sup> <http://4.bp.blogspot.com/-4R6Xkz9OPOg/UsUKBxN1zpl/AAAAAAAAAFGk/kbHfCcXdXS8/s1600/658-traffic-light-design.png>

## Continue With or Without Scratch

Now that we have experience with Scratch, we can continue to develop our own project using our imagination. The Arduino however is powerful and capable of many more functions. The Development board may be used with the Arduino in its native ecosystem called the Arduino IDE. This is a free software that will allow programming in C++ without restrictions that Scratch may impose. Many spare pins are available using the various headers to which the user may interface. Have fun exploring the full potential of the development board and Arduino!



## Glossary

Command	Description	Variable(s) type
<b>Control</b>		
When clicked.....	Run the program when clicked.....	None
When [n] key pressed.....	Run the program when key pressed.....	[A-Z, 0-9 or arrow]
When Arduino 1 clicked.....	Run the program when the sprite is clicked.....	None
Waid [n] seconds.....	Stop the program for n seconds.....	[Positive real number]
Forever.....	Will repeat code inside continuously.....	None
Repeat [n].....	Will repeat the code n time.....	[Positive Integer]
Broadcast [n].....	Used to trigger external code.....	*see: When I receive
Broadcast [n] and wait.....	Trigger external code but will hold.....	[subroutine name]
When I receive [n].....	Used to identify external code.....	[subroutine name]
Forever if <n> <sup>5</sup> .....	Combines the Forever and If statement.....	< Comparison or Sensor>
If<n>.....	Checks the condition of the statement.....	< Comparison or Sensor>
If<n>else.....	If the statement is true A, otherwise B.....	< Comparison or Sensor>
Wait until <n>.....	Will stop the program until a condition.....	< Comparison or Sensor>
Repeat until<n>.....	Will repeat until a condition is met.....	< Comparison or Sensor>
Stop script.....	Will end a program or sub-program.....	None
Stop all.....	Well halt all routines.....	None
<b>Motion</b>		
Value of sensor [Analog n].....	Reads the value of the analog input n.....	[Analog 0~5, Digital 2~3]
Sensor [Digital n] pressed.....	Reads the input of digital n.....	[Digital 2~3]
Digital [n] on.....	Set the digital output n on.....	[10~13]
Digital [n] off.....	Set the Digital output n off.....	[10~13]
Move[n].....	Instantly move sprite forward.....	[Integer]
Turn[n].....	Rotate sprite.....	[Integer]
Point in Direction[n].....	Point sprite in cardinal direction.....	[0, 90, 180, -90]
Point towards[n].....	Point sprite towards another sprite.....	[User defined]
Go to x: [x]y: [y].....	Instantly move sprite to a location.....	[Integer, integer]
Go to[n].....	Instantly move a sprite to another sprite's location.....	[user defined]
Glide[n]secs to x: [x]y: [y].....	Will move a sprite to a location over time.....	[Rational, int., int.]

<sup>5</sup> Forever If is no longer supported in Scratch and therefore should be avoided.

## Operators

$(x) + (y)$	Will use the result of $x + y$	(Rational number)
$(x) - (y)$	Will use the result of $x - y$	(Rational number)
$(x) * (y)$	Will use the result of $x \times y$	(Rational number)
$(x) / (y)$	Will use the result of $x \div y$	(Rational number)
$(x) < (y)$	Comparison operator $x$ less than $y$	(Rational number)
$(x) = (y)$	Comparison operator $x$ equal to $y$	(Rational number)
$(x) > (y)$	Comparison operator $x$ greater than $y$	(Rational number)
$(x)$ and $(y)$	Boolean operation $\&\&$	< Digital 2~3 >
$(x)$ or $(y)$	Boolean operation $\ \ $	< Digital 2~3 >
not( $x$ )	Boolean operation $!x$	< Digital 2~3 >

## Variables

Make a variable	Dialog to make a variable name	[variable name]
Delete a variable	Dialog to remove variables	[variable name]
<input checked="" type="checkbox"/> ( $n$ )	When checked, the variable $n$ appears	*see: Make Variable
Set[ $n$ ] to [ $x$ ]	Used to set a variable $n$ to $x$ value	[name, real number]
Change[ $n$ ] by [ $x$ ]	Increment or decrement $n$ by $x$	[name, real number]
Show variable [ $n$ ]	Will display the current value of $n$	[variable name]
Hide variable [ $n$ ]	Will hide the current value of $n$	[variable name]
Make a List	Create a list for look-up	Not covered in this module

Sound, Sensing, Looks, and Pen all follow standard Scratch programming and are not covered in this version.

## Notes:

- Nested if statements may not function in this version.