# YOUTH WORKBOOK


# STEM

SCIENCE TECHNOLOGY ENGINEERING MATH

5 May 2018

@BSidesAugusta

# TABLE OF CONTENTS

# STEM References

## BSides Augusta STEM

| | |
|---|---|
| **BSides Augusta STEM Website** | https://bsidesaugusta.org/stem/ |
| **BSides Augusta STEM Files** | https://github.com/BSidesAugusta/STEM |

## Labs Material List

| *Qty* | *Description* | *Price* |
|---|---|---|
| **1** | **Raspberry Pi 3 Model B Desktop Starter Kit (16 Gb, White)**<br>**https://goo.gl/HTtGbD** | **$65.00** |
| **1** | **SunFounder Raspberry Pi Holder**<br>**https://goo.gl/BWERpv** | **$8.99** |
| **1** | **SunFounder Raspberry Pi 3 Zero Starter Kit**<br>**https://goo.gl/KDBVyy** | **$35.99** |
| **1** | **E-Projects EPC-105 16 Value Resistor Kit, 10 Ohm - 1M Ohm, 1/2 Watt**<br>**https://goo.gl/5Fb7BP** | **$12.01** |
| **1** | **HDMI Cable**<br>**https://goo.gl/EshKE8** | **$2.50** |
| **1** | **HDMI to DVI Converter**<br>**https://goo.gl/ycbwqW** | **$6.99** |
| | ***Total*** | ***$131.48*** |

## Raspberry Pi 3 References

| | |
|---|---|
| **Raspberry Pi** | https://www.raspberrypi.org/ |
| **Raspberry Pi Education** | https://www.raspberrypi.org/education/ |
| **Raspberry Pi Products** | https://www.raspberrypi.org/products/ |

## Python 3 References

| | |
|---|---|
| **Python 3 Downloads** | https://www.python.org/downloads/ |
| **Python 3 Documentation** | https://docs.python.org/3/ |
| **Python 3 Standard Library** | https://docs.python.org/3/library/index.html |
| **tutorialspoint Python Training** | https://www.tutorialspoint.com/python3/index.html |
| **codecademy Python Training** | https://www.codecademy.com/catalog/language/python |
| **Udemy Python Training** | https://www.udemy.com/ |

## Electronics References

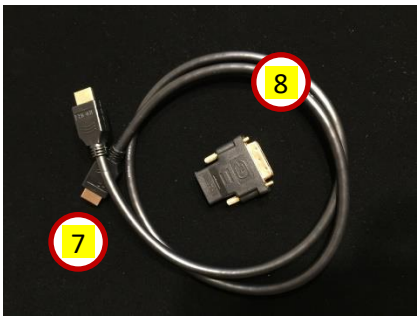
| | |
|---|---|
| **Sunfounder Raspberry Pi Tutorials** | https://www.sunfounder.com/learn |
| **Raspberry Pi Projects Book PDF** | https://www.raspberrypi.org/magpi-issues/Projects_Book_v1.pdf |

# Raspberry Pi 3 Setup

## Inventory Equipment

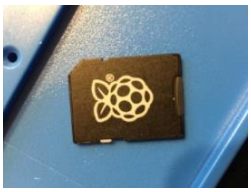
Take a moment to make sure you have everything you need.

1. Raspberry Pi 3 Model B Desktop Starter Kit
2. Raspberry Pi NOOBs Micro SD Card
3. Raspberry Pi Power Supply
4. SunFounder Raspberry Pi Holder
5. SunFounder Raspberry Pi 3 Zero Starter Kit
6. Raspberry Pi 3 Case
7. HDMI Cable
8. HDMI to DVI Converter
9. E-Projects EPC-105 16 Value Resistor Kit, 10 Ohm - 1M Ohm, 1/2 Watt (Pack of 400)

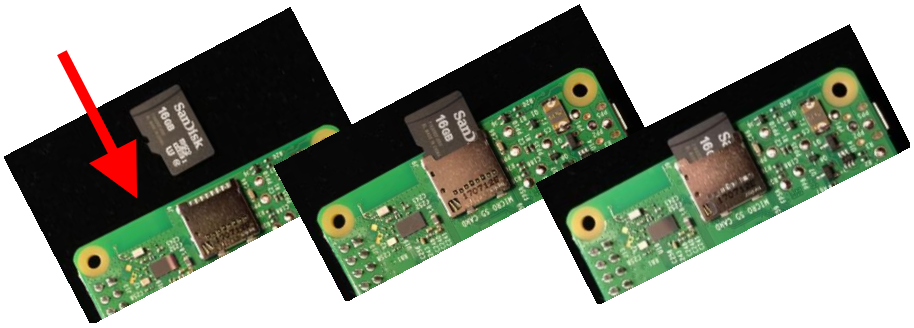

## Raspberry Pi 3 Electronic Kit Setup

Setting up the BSides STEM Raspberry Pi 3 (RPi) electronic kit is very easy. We will assemble the Raspberry Pi electronics kit and then connect it to the classroom's mouse, keyboard and monitor.

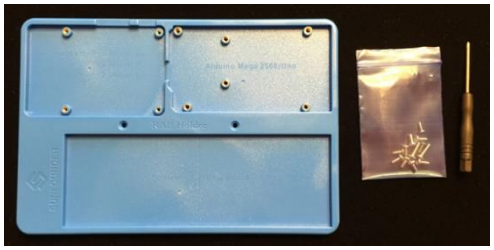


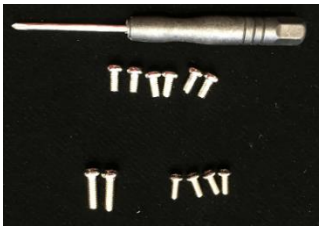
1. Remove the Raspberry Pi 3 NOOBs micro SD card from the package and slide out the SanDisk NOOBs Miro SD card out of the SD card adapter. Then remove the Raspberry Pi 3 from the packaging and turn the board upside down and insert the NOOBs SD card into the micro SD card holder.

2. Remove the Raspberry Pi 3 Model B from the box. This micro SD card is preloaded with the Raspbian OS and NOOBS installer. Turn the board over to locate the micro SD card holder. Gently slide the SD card into the holder with the SanDisk label facing upward. The card should spring latch into place.

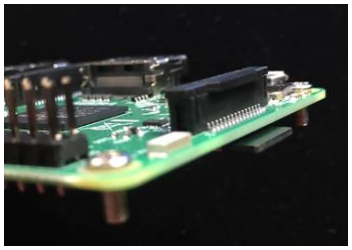


3. Remove the SunFounder Raspberry Pi 3 Holder from the packaging. You will need to use the four small screws and the screw driver to mount the Raspberry Pi onto the holder.

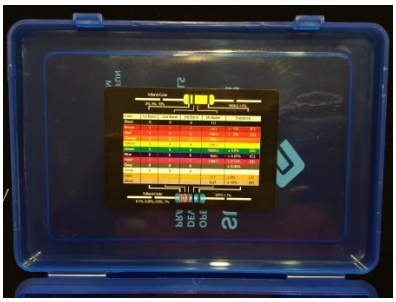




4. Turn the Raspberry Pi board so it is facing up. Drop the four short screws into the four mounting holes. Then carefully place the Raspberry Pi 3 onto the holder as shown in the picture. Use the Philips screwdriver to tighten the screws down.

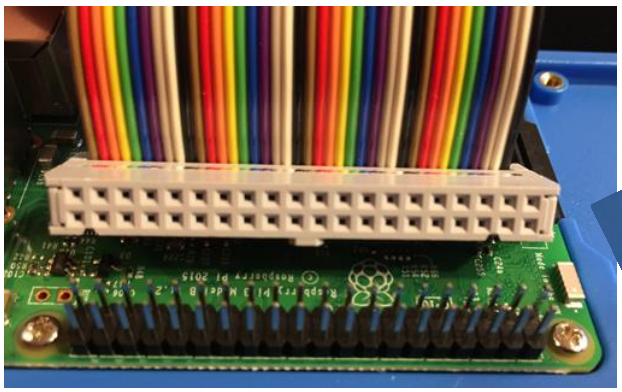


5. Unpack the SunFounder Raspberry Pi 3 Zero Starter Kit contents. Remove the resistor value color code chart sticker and place it on the inside cover of the blue electrical box. You will be referring to this chart for determining the resistor ohms.

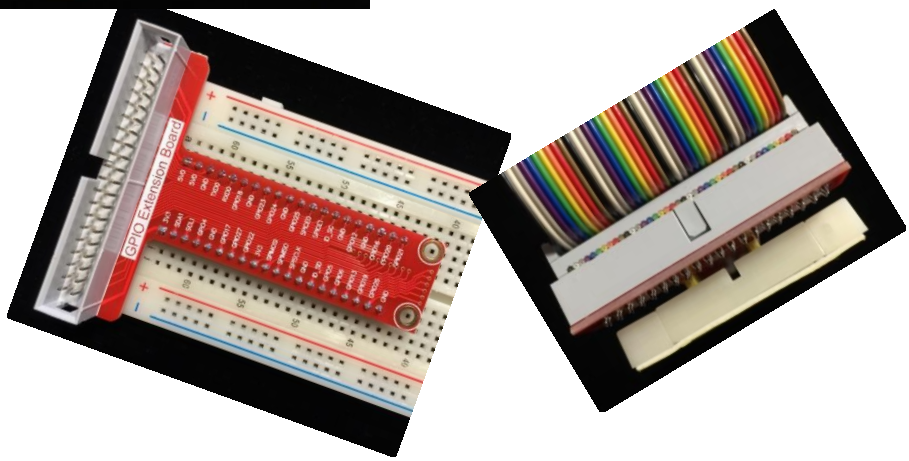



6. Unpack and store all of the smaller electronic components in the blue storage container.

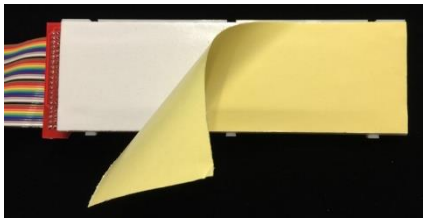
7. Now you will connect the Raspberry Pi General Purpose Input / Output (GPIO) pins to the bread board. This will allow you to use electronics with the Raspberry Pi. Layout the colorful 40 conductor ribbon cable with the black wire end on the right-hand side. *Carefully line up the board's GPIO pins to the ribbon cable connector. Carefully press the ribbon cable connector onto the pins.*

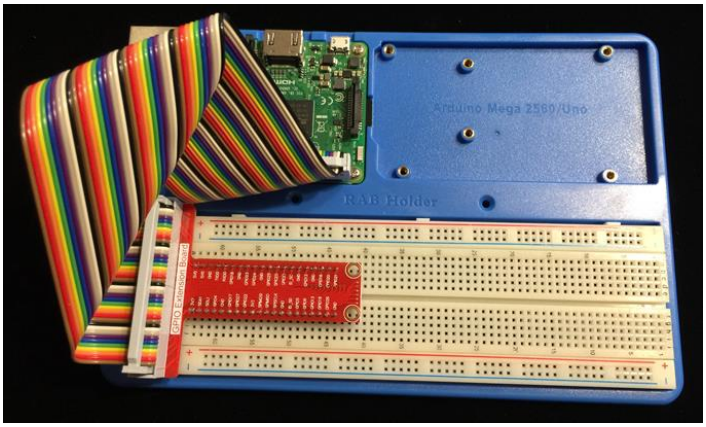


8. The other end of the 40-conductor ribbon cable will be connected to the breadboard by using the "T" shape GPIO Extension Connector. *Carefully line up the bottom pins of the GPIO connector to the edge of the large breadboard. Gently press the connector into the breadboard.* Then connect the keyed ribbon cable connector to the GPIO connector.

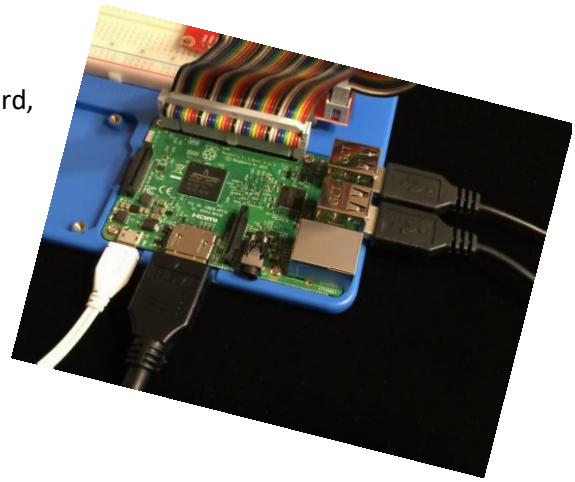



9. Peel off the back of the sticker on the bottom side of the breadboard. Then firmly place the board onto the large section of the Raspberry Pi holder as shown in the picture.

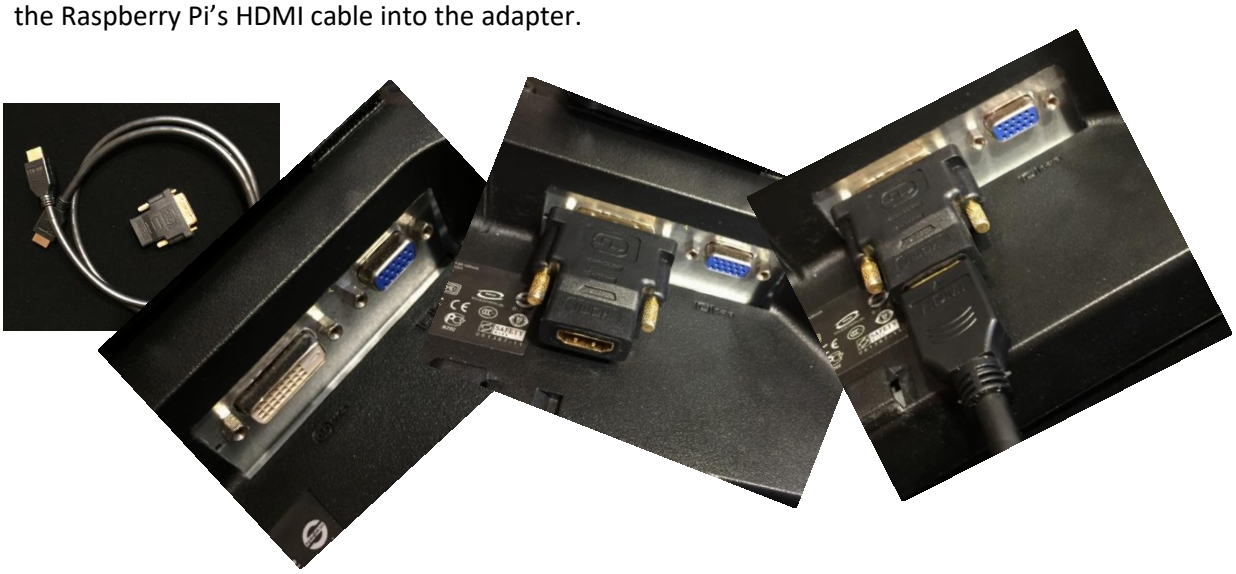




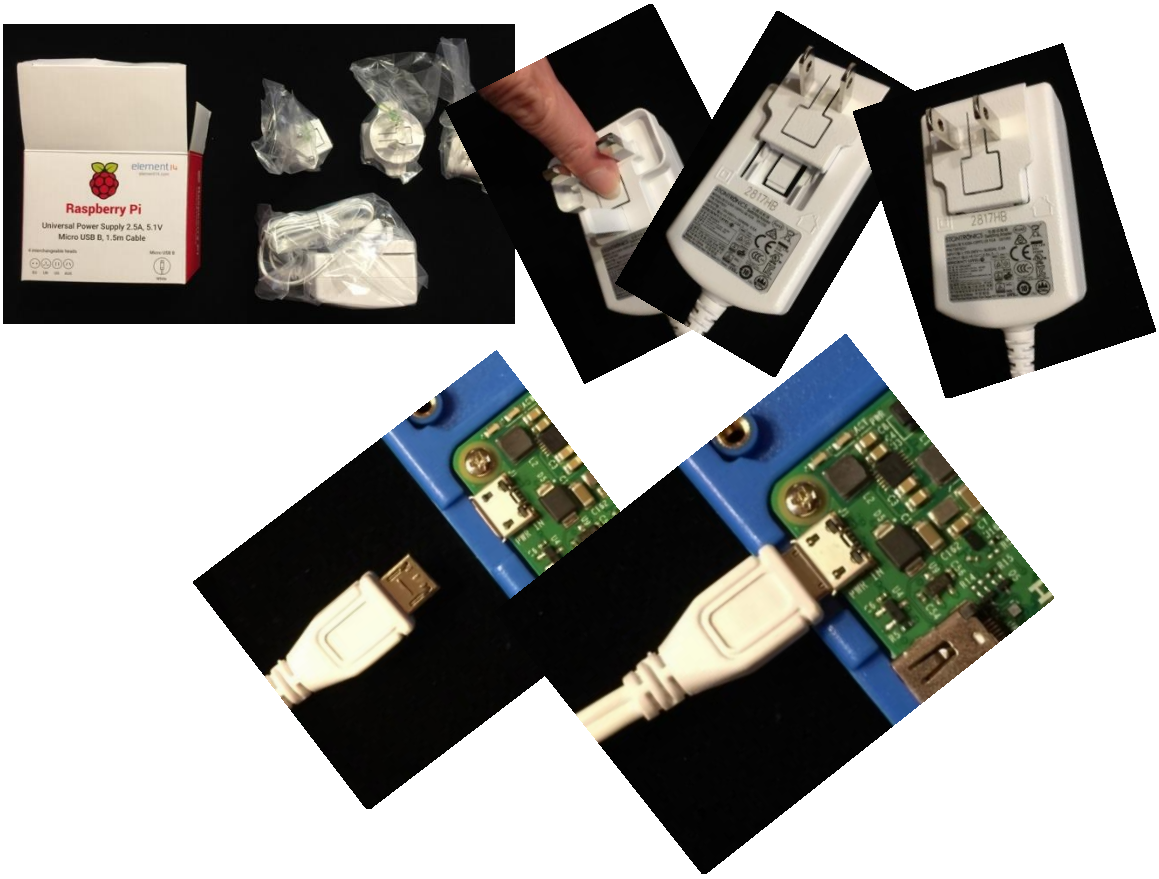
10. We will be borrowing the computer room's keyboard, mouse and monitor. Plug in the USB connectors from the keyboard and mouse into the Raspberry Pi's USB ports. Then plug in the HDMI cable into the HDMI port.



11. Now we will plug the HDMI to DVI adapter into the classroom's monitor DVI port. Then plug in the Raspberry Pi's HDMI cable into the adapter.



12. Unpack the Raspberry Pi power adapter. This adapter comes with the incorrect power plug prongs installed. Remove the prongs by pressing on the small square with your finger and slide off the prong assembly. Now locate the United States two prong assembly and slide it into the adapter until it snaps into place. Now plug the adapter into the AC outlet. It is now time to plug the Raspberry Pi power cord into the Raspberry Pi's power slot.

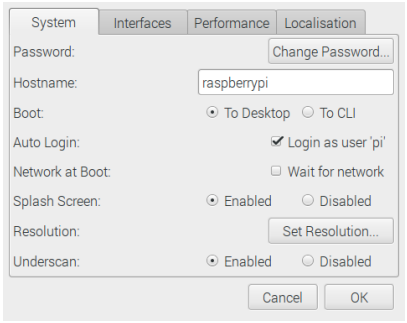


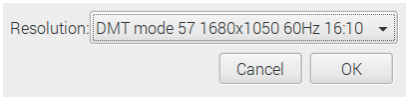
## RASPBIAN OS CONFIGURATION

There are several different operating systems (OS) available for the Raspberry Pi 3. You will be using the preinstalled Raspbian OS from the New Out Of the Box (NOOB) Micro SD card. The Raspberry Pi 3 will automatically boot up to the Raspbian OS once power is applied to the board. There are a few first-time configuration settings that need to be set.

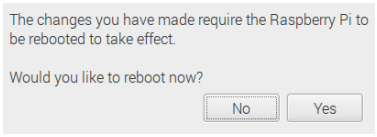
### Video Resolution

1. Press the Raspberry Start button in the upper left-hand corner.
2. Select “Preferences” => “Raspberry Pi Configuration” to pop up the configuration settings window.
3. Press “Set Resolution…” to popup the video resolution options.
4. Select Resolution: “DMT mode 57 1680x1050 60Hz 16:10” and Press “OK”.



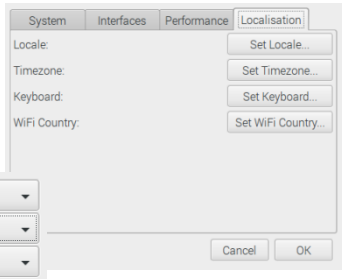



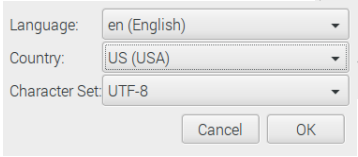

5. We do not wish to reboot right away, so Press “No” to cancel the reboot.

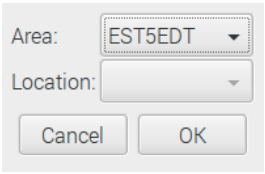



### Localization Configuration

6. Select the Localisation tab.
7. Press “Set Locale…” and verify Language is “en (English)” and Character Set: “UTF-8”. Select Country: “US (USA)” and Press “OK”.



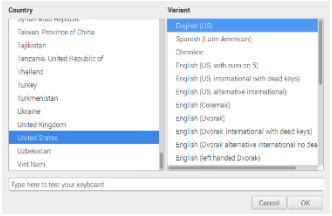




### Timezone Configuration

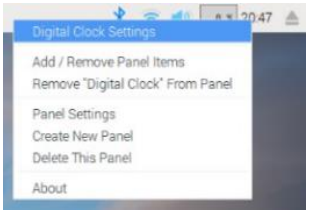
8. Press “Set Timezone…” and Select Area: “ETS5EDT” and Press “OK”.

### Keyboard Configuration

9. Press “Set Keyboard…” to pop up the keyboard layout window. Select Country: “United States” and Variant: “English (US)”. Press “OK” to close the window.

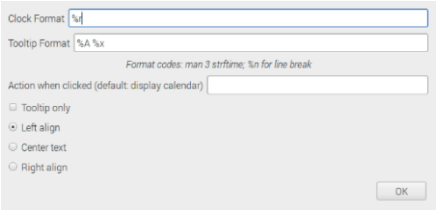


## Raspbian OS Clock




10. Press the clock readout in the upper left-hand corner and Select "Digital Clock Settings".

11. Change the Clock Format: "%r" and Press "OK".




## Wi-Fi Connection

The Wi-Fi connection needs to be configured to allow Internet access to update the Raspbian OS.

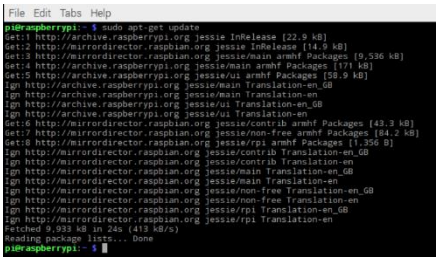
1. Press the Wi-Fi icon in the upper right corner to view the available Wi-Fi connections. Access to the classroom's Wi-Fi connection will be provided during the STEM session.

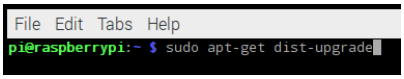
## Raspbian OS Update

*(ONLY DO THIS ON YOUR HOME RPi!)* There will not be enough time to update the Raspbian OS during the BSides STEM session, so please update your Raspberry Pi at home. Updating the OS can take up to 45 minutes. All newly installed OS should be updated for added cyber security protection. Enter the following commands to update the Raspbian OS to the latest software packages.

1. Open a Terminal window by clicking on the icon in the upper left corner.

2. Enter the following command line to make the Raspbian OS capture the latest package version lists from the software vendor's update servers:
"sudo apt-get update" and Press the Enter key.




3. The next step is to command the Raspbian OS to upgrade all of the installed software packages: "sudo apt-get dist-upgrade" and Press the Enter key. Note that this may take up 15 minutes or longer depending upon the network speeds to download all of the OS update packages. When prompted: Type "q" to quit viewing the readme file. The update process then unpacks all of the downloaded packages and installs each of the software packages. Note that this may take over 30 minutes to complete with a good Internet connection.

4. Close the Terminal window by typing: "exit" and Press the Enter Key.

***Congratulations! You are ready to do some programming & electronics.***

# INTRODUCTION TO PROGRAMMING WITH PYTHON

## PYTHON 3 INTEGRATED DEVELOPMENT ENVIRONMENT (IDLE)

We will be using Python's Integrated Development Environment (IDLE) to write and run our programs. The Raspbian OS has two versions of Python versions to choose from: 2 or 3. We will be using the Python 3 for the BSides STEM labs. Now, let's start programming by starting up the Python 3 IDLE application.

1. Click the Raspberry "Quick Launch" icon in the upper left corner.
2. Select "Programming" => "Python 3 (IDLE)" to open the Python 3.4.2 Shell window. You will notice that the command prompt looks like ">>>". This is where we can enter commands or run Python statements after this prompt.

## PROGRAMMING LAB 1 - "HELLO BSIDES AUGUSTA STEM STUDENT!" AT THE PYTHON PROMPT

Now it's time to write your first Python program using your RPi! The Python IDLE application will automatically color the programming text according to its purpose. Text showing up in Red indicate that they are reserved words for command or functions. Green text indicates basic text characters.

1. Enter in the following Python code after the command prompt ">>>". Use the apostrophe [‘] which is on the same key as the quote mark ["].

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on Linux
Type "copyright", "credits" or "license()" for more information.
>>> print ('Hello BSides Augusta STEM Student!')
Hello BSides Augusta STEM Student!
>>>
```

   Did you notice how the text color was automatically changed? The print function keyword changed to Purple text, character text changed to Green and the parentheses stayed Black. The Python IDLE also popped up the print's function layout for your reference after you typed in "print(".

2. Now Press the Enter key. The Python line printed out the output in Blue text:
   Hello BSides Augusta STEM Student!

***Congratulations*** on writing and running your very first Python program! Run the program again by typing the code and Press the Enter key to see the same screen output. This was not so bad since there was only one line of code; however, most programs have several lines of code.

But, nobody wants to spend the time to retype the same code over and over again. From now on we will be writing our Python code to a text file with the .py extension. For example, filename Hello.py. This will allow us to save our program and quickly run it any time in the future.

## PROGRAMMING LAB 2 - "HELLO BSIDES AUGUSTA STEM STUDENT!" IN A PYTHON FILE

Most Python programming programs are several lines long and are stored in text files with the ".py" file extension. From now on you will be saving your Python programming lines in files with unique identifying filenames. Storing programs in files allows you to quickly run the code at any time and it also provides a document for future quick referencing. We will now create the "Hello BSides Augusta STEM Student" Python program file.

1. Go to the File menu and click on "New File" which will bring up a new Python IDE window. Save this file as "Lab 2 - Hello.py".

2. Most programming languages have a method which allows developers to make comments within their code. Good programmers document their code with comments which basically explain the purpose of the program and the more complicated lines of code. Python 3 allows insertion of comments by starting out the line with the pound sign "#". This comment character tells the Python 3 IDE to ignore all characters after this symbol on that line.

   Professional programmers add a documentation header to the top of every file. This comment header informs future programmers about the details of the program. Now enter in your header for the "Lab 2 - Hello.py" program as shown below. Just enter in your name for the Developer section since you are writing the program. Notice how the comment text in the Python IDE turns Red. The Python 3 IDLE color codes the program to make it easier for the programmer to read.

```
##############################################
# Filename: Lab 2 - Hello.py
# Developer: BSides Augusta STEM
# Language: Python 3.4
# Description: This program will output to the screen
# "Hello BSides Augusta STEM Student!"
```

   Save the file after entering in these comments. Try to save your code frequently to ensure that you do not lose your work in the event of a power failure or that you accidently closed the IDE window.

3. We are ready to add the print statement as shown below. First add the line code comment which explains what the following line does. In our case it prints out our statement to the screen. Notice how the Python IDLE colors the reserved keywords like the "print" function statement to Purple and the text characters to print out to Green. Programming functions usually perform a single operation like the Python 3 print function. All functions have a name and take input parameters that are listed between the brackets "(" and ")". In this case, the print function takes in a string of text characters to print to the monitor. Make sure to save the file again.

```
# Print to Screen
print('Hello BSides Augusta STEM Student!')
```

4. To run the program, go to the Run menu and click on Run Module. A shortcut method for running the program is to press “F5”. You should see the program output in the other Python Shell window.

```
>>> ================================ RESTART ================================
>>>
Hello BSides Augusta STEM Student!
>>>
```

***Congratulations, you’ve just run your first Python program file!***

## ***Lab Challenge Questions***

Did you accidently mistyped in the program which caused a runtime error? If so, that is actually a great way to learn how to troubleshoot your code. Understanding how to interpret the Python runtime errors is a very important skill for a software developer. The suggested errors below will have you purposely insert programming errors so that you can learn from the generated runtime error outputs. Take the time and write down the error received. Then fix the error and validate that your program runs before moving onto the next error.

1. Remove the comment character “#” before the Filename: BSides Augusta STEM and run the code.

2. Rename the reserved function name “print” to “rint”. Did you notice the color of text changed?

3. Remove the “(“ in the print() function.

4. Remove the single quote mark “ ‘ “ and notice that the green highlighted text color changed. That is a good clue to recognize that there is a coding issue.

***Pay attention to the Python 3 IDE text coloring coding since it will help you to discover coding errors before you attempt to run the program.***

## PROGRAMMING LAB 3 – ASK & PRINT NAME

Python can do so much more than printing "Hello Sides Augusta STEM Student!" to the monitor screen. Python can take input from the user and much more. In lab 2 you will prompt for the user's name and output "Yourname, - The Raspberry Pi is a Friend and Not Food!".

Programmers frequently like to reuse programs to save time from typing and re-inventing the wheel. Save the previous program "Lab 2 – Hello.py" as "Lab 3 – AskPrintName.py". You can reuse the header comment as a template and the print statement. Modify the code to match the program below and save.

```
#################################################
# Filename: Lab 3 - AskPrintName.py
# Developer: BSides Augusta STEM
# Language: Python 3.4
# Description: This program prompts for your name and outputs
# "Yourname! 'The Raspberry Pi 3 is a Friend and Not Food!"

# Prompt for the User's Name
print('What is your name?')

# Capture User's Keyboard Input
name = input()

# Output Message to the Screen
print(name, ' - The Raspberry Pi 3 is a Friend\nand Not Food!')
```

The second line of code uses a variable, "name", to store the user's keyboard response after the return key is pressed. Then it prints out the variable value "Your Name" along with the string of characters " – The Raspberry Pi 3 is a Friend and Not Food!". Now press F5 to run the program which should provide similar results as shown below.

```
>>> ================================ RESTART ================================
>>>
What is your name?
Bruce
Bruce - The Rapsberry Pi 3 is a Friend
and Not Food!
```

### ***Lab Challenge Questions***

1. What happens if you type in your first and last name?
2. What does the screen output look like if you remove the special escape newline character "\n"?
3. What happens when you replace the special escape character "\n" with the tab "\t"?
4. Change the variable name in "name = input()" to "banna = input()". What happened when you ran the program?

## PROGRAMMING LAB 4 – COMPARE NUMBERS

You learned how to print and receive text input characters, but now you will learn how to convert the text into an integer number. Using numbers will allow us to make calculations and logical comparisons between numbers. The compare numbers program will demonstrate how to convert (sometimes called cast or casting) text into an integer. The newly converted integer number will be compared to numbers ranging from 0 to 10. The output will be False if the input number does not equal the comparison number. Otherwise, it will output True if the two numbers are equal. Save the Lab 3 program as "Lab 4 – CompareNumber.py" and reuse the header comment and any other code that you can use again.

```
################################################
# Filename: Lab 4 - CompareNumber.py
# Developer: BSides Augusta STEM
# Language: Python 3.4
# Description: The user is prompted to enter a number from 0 & 10.
# That number is then compared to numbers from 0 to 10 and outputs
# the comparison of the two numbers as True or False.

# Prompt the User for a Number
print('Enter a number from 0 to 10')

# Convert the entered character value into an Integer number.
num = int(input())

# Loop x from 0 to 10
for x in range(11):

    # Output the Entered Number to x Loop and Output Comparison
    print('Is',num,'equal to ',x,'?', num == x)
```

Remember that the input() function waits for the user to enter in a keyboard response. In this case, the response should be a number from 0 to 10. The number comes back as a string text character, so it must be casted into an integer with the int() function. The returned number is then assigned to the variable "num" which can be used for mathematical comparisons.

The Orange "for x in range(11)" statement runs a series of numbers for the variable "x" from 0 to 10. The "x" variable will then compared to the user's input number which is stored in variable "num".

Notice how the print function's parameters include strings, integer variables and number comparisons. The integer numbers are printed out when they are included with the characters with a comma separating them. This includes the Boolean (0 or 1; False or True) equation results from "num == ".

```
>>> ================================ RESTART ================================
>>>
Enter a number from 0 to 10
8
Is 8 equal to  0 ? False
Is 8 equal to  1 ? False
Is 8 equal to  2 ? False
Is 8 equal to  3 ? False
Is 8 equal to  4 ? False
Is 8 equal to  5 ? False
Is 8 equal to  6 ? False
Is 8 equal to  7 ? False
Is 8 equal to  8 ? True
Is 8 equal to  9 ? False
Is 8 equal to  10 ? False
>>>
```

Press F5 to see your program in action!

## ***Lab Challenge Questions***

1. What is the first comparison number?  Note that several programming languages have "Zero Based" functions.  That means that they start counting from 0 and not from 1!.

2. How many number values are generated with the "range(11)" statement?  Don't forget to count the 0 value.

3. Change the value in "range(11)" to "range(20)".  What number range is produced from the change?

The compare numbers program offers more opportunities for practicing our coding runtime troubleshooting skills.  Create the following errors and run the program to see the runtime errors.

1. Change the int() casting function name to float().  What happened with the screen output of the user's number?

2. Replace the variable comparison "num==x" with the variable assignment "num=x".  The print function sure does not like you to assign a variable (using one mathematical comparison operator) as a passed in parameter.

## PROGRAMMING LAB 5 – CONVERT TEMPERATURE

This program converts the user's entered Fahrenheit temperature into Celsius. Then based on the temperature it returns one of three different responses: "It is too cold!", "It is very hot!" or "The temperature is perfect!". You will learn how to implement a basic mathematical equation along with mathematical comparison operators in an "if-elif-else" statement. You should now be efficient in making the new Python file "Lab 5 - ConvertTemp.py" and copying the code below.

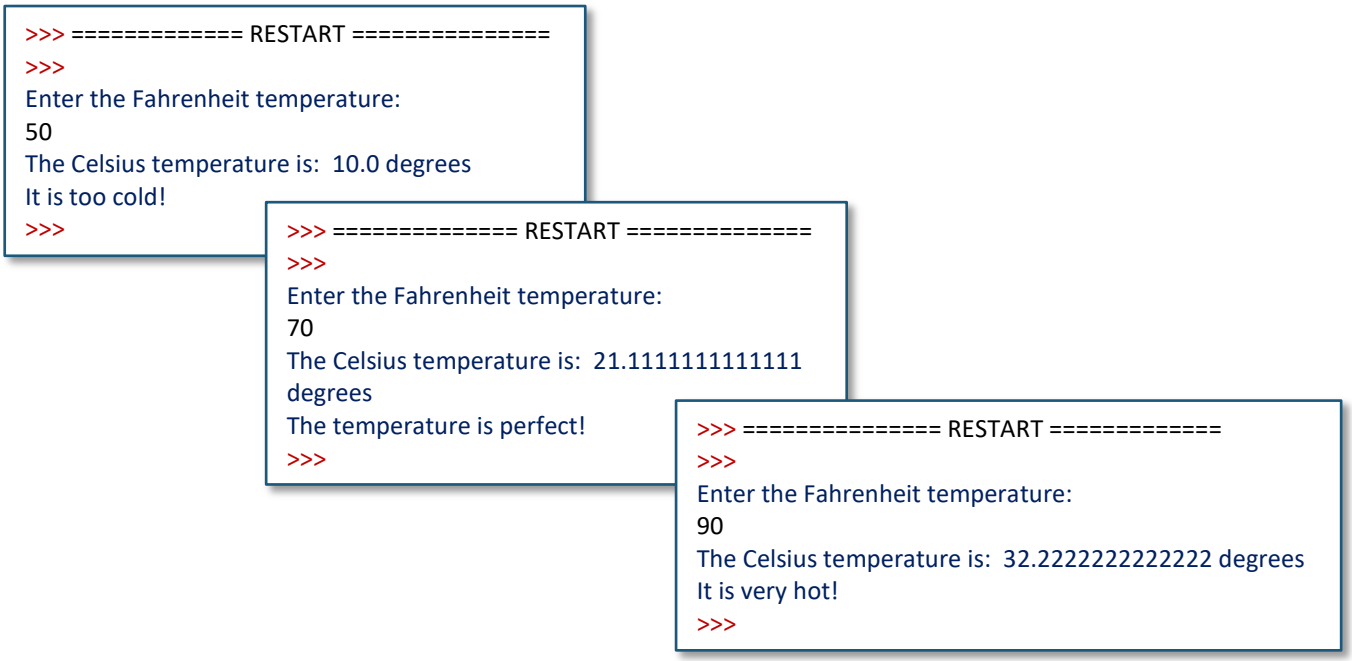
```
################################################
# Filename: Lab 5 - ConvertTemp.py
# Developer: BSides Augusta STEM
# Language: Python 3.4
# Description: This program prompts the user for the temperature
# in Fahrenheit and converts it to Celsius. The Celsius
# temperature value is displayed to the screen and whether
# the temperature is perfect, too cold or very hot.

print('Enter the Fahrenheit temperature: ')
tempF = float(input() )

tempC = (tempF - 32.0)*(5.0/9.0)
print('The Celsius temperature is: ', tempC, ' degrees')

if (tempF > 85):
      print('It is very hot!')
elif (tempF < 60):
       print('It is too cold!')
else:
      print('The temperature is perfect!')
```

The Fahrenheit to Celsius temperature conversion equation uses the user's input value which was casted into a floating-point number. The equation result is then placed into the tempC variable which is printed out to the screen.

The next section uses the "if-elif-else" conditional statement to determine how comfortable the temperature really is. The Fahrenheit temperature variable is compared to three different ranges to determine if it is hot, perfect or cold. Each Boolean comparison operation within the if statement "(" ")" should result into either a "True" or "False" condition. The indented print statement after the conditional test is only ran if the comparison statement results in "True". Otherwise, a "False" result will not run the indented print line below the comparison.

Say the first "if" comparison results in "False", then the next "elif" comparison will be tested. Please note that "elif" is short for "else if" as in other programming languages. The "else:" section runs only if all of the previous comparisons resulted in "False". The if-elif-else conditional statement produces the following screen output for the temperature range.

<-----It is too cold!-----59|60-----The temperature is perfect!-----84|85-----It is very hot!----->




## ***Lab Challenge Questions***

1. Try entering a letter like "G" instead of a number. Why will this not work?

2. Enter in a temperature value of 44 degrees. Then change the number casting function "float(input())" to "int(input())" and run the program with 44 degrees as an input. Why did you get the same temperature conversion result? (Note: some language might not allow you to mix number types in equations, but Python does not care in this case.)

3. Remove the indentions before the print functions within the if-elif-else statement. What error to you get? Indentions are very important in Python. In this case, every line of code that is to be ran after the if-elif-else comparisons must be indented. Now insert two spaces before each print statement in the if-elif-else conditional statement and run the program.

4. The if-elif-else condition statement tests for "greater than" or "less than" temperature values. How can you change the comparison operators to test for "equal to and greater" and "equal to and less than"?

5. Remove the ":" after the "if (tempF > 85):" statement. Notice the invalid syntax error produced.

6. Change the "if (tempF > 85):" to "if (tempF < 85):" and run the code with 10, 70 and 90-degree values. Why are the temperature comments wrong? Troubleshoot by using each temperature value above and running it though the three different if-elif-else conditions.

## PROGRAMMING LAB 6 – GUESS A NUMBER

Python has several prewritten and tested specialty code functions that can be used for coding development. Note that each function does one unique task, for example, the *randint* function returns a random integer number. The random integer function is stored in the *random* programming class which includes other functions related to the random topic. Use Google to search for Python 3 random class to learn more about the other functions.

```
################################################
# Filename: Lab 6 - GuessANumber.py
# Developer: BSides Augusta STEM
# Language: Python 3.4
# Description: This game will ask the user to guess a number between 1 and 10. The program will inform the user
# if they are correct.  Enter 11 or More to quit.

# Import Special Classes
from random import randint

# Initialization
guess = 0
RandomNumber = 0
# Inform the User How to Quit the Game.
print('Enter 11 or More to Quit the Game.\n\n')

# Loop until they want to quit by entering more than 10.
while guess < 11:
    # Prompt for the Number Guess
    print('\n\nHow lucky are you?')
    print('Guess an integer number from 1 to 10.')
    guess = int(input())

    if (1 <= guess <= 10):
        # Capture the Input
        RandomNumber = randint(1,10)
        # Output the Number
        print('You guessed ',guess)
        print('The Random Number was ',RandomNumber)

        if (guess == RandomNumber):
        # Sprinkles for the Winner
        print('\nSPRINKLES FOR THE WINNER!')

# Thank you for playing the game
print('Thank you for playing!\n\n')
```

The Guess the Number game program will teach you about importing the Python 3 random integer function to provide a random number. The line "from random import randint" informs the Python IDLE to import the function "randint()" function from the *random* class.

The while loop statement shown below in Orange runs a conditional programming loop as long as the "guess" variable is less than 11. All indented statements below the while loop will run as long as "guess" is Less Than 11. The user can quit the game by entering in a number 11 or greater which will result in a "False" condition which will break the while loop condition. Now enter the lab program below.

Running the Guess the Number game should result in an output similar to below. The program tells you if you guessed wrong, gives you sprinkles for guessing right and quitting the game after entering a number greater than 10.

```
>>> ================================ RESTART ================================
>>>
Enter 11 or More to Quit the Game.

How lucky are you?
Guess an integer number from 1 to 10.
3
You guessed  3
The Random Number was  7

How lucky are you?
Guess an integer number from 1 to 10.
5
You guessed  5
The Random Number was  3

How lucky are you?
Guess an integer number from 1 to 10.
3
You guessed  3
The Random Number was  3

SPRINKLES FOR THE WINNER!

How lucky are you?
Guess an integer number from 1 to 10.
22
Thank you for playing!
>>>
```

## ***Lab Challenge Questions***

1. Comment out the import class function line "from random import randint" and run the code. What function could not be found?

2. Change the initialization of "guess = 0" to "guess = 30" and run the program. Why did the game start and end so quickly?

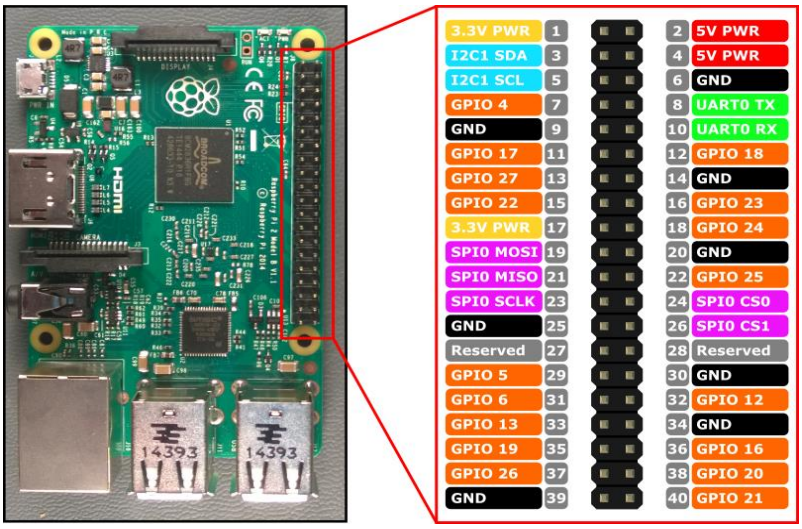
3. Change the conditional statement from "while guess < 11" to "while guess < 5". Run the program with 3, 4, 5 and 12. Why does the program end so early?

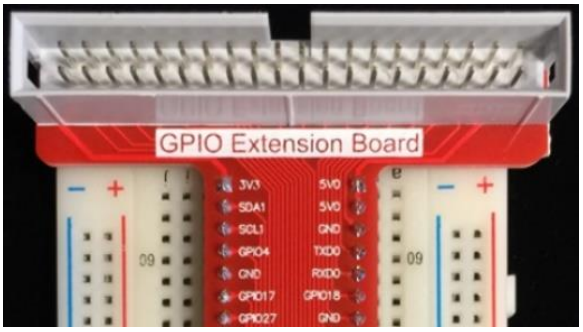
# RASPBERRY PI & ELECTRONICS

You now have some basic experience with Python 3 programming on the Raspberry Pi 3. One of the great reasons to program on the Raspberry Pi is interfacing with its input and outputs ports for controlling electronics and robotic devices. Later on, we will learn how to program the RPi to interface basic electronic devices like LEDs, buzzers and push buttons.

## GPIO ON THE RPI

One of the exciting things about the RPi is its 40 General Purpose Input & Output (GPIO) pins. The actual physical RPi pin numbers are shown in the numbered gray squares. Each pin has a colored description next to the gray pin numbers. Notice the pins labeled with names with GPIO 4, GPIO 21 and so on. You will be frequently referencing these GPIO pins by these names in the Python code. Some people plug a jumper wire onto a GPIO pin and then plug the other end into the electronic breadboard. This method is a little cumbersome since you have to count the RPi pins on the board in order to locate the proper one. An easier method is to attach a 40-pin ribbon cable and then attach the other end to the breadboard with the GPIO Extension Connector.




The "T" shaped GPIO Extension Connector makes wiring electronics a lot easier since the GPIO pin names are printed by the pins.

We use Python to control when to send output signals and/or read input signals through the GPIO pins. These GPIO output pins provides either a high signal output voltage (+3.3 VDC) or a low signal voltage (0 VDC).

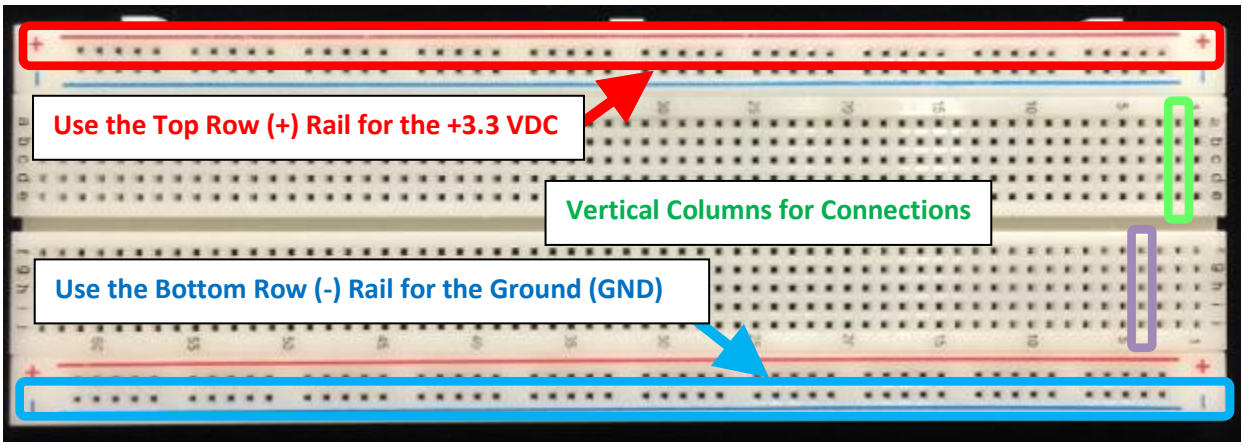
Output electronic devices that may be controlled by the RPi may include LEDs, servos, 7-segmented display, buzzers and other types of electronics. Several of these devices can be found in the electronic kit.

Input devices that could send a signal to the RPi includes pushbuttons, switches, resolvers and sensors. All of the electronics can be easily wired together by using the breadboard. We will get into the basic details of these sensors in a little bit.

## ELECTRONIC BREADBOARD

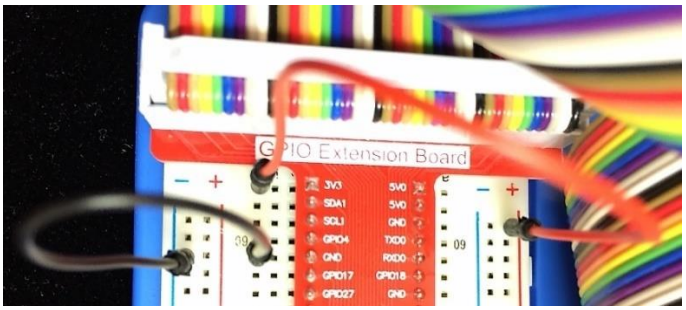
The electronic breadboard provides electronic connections between direct current voltage (VDC) supplies and the electronic devices. The GPIO Extension Connector also provides pins for the 3.3 VDC power and the GPIO pins. Refer to the red box shown in the picture below. The two outer rows of connector holes provide the same electrical connection. Note that the red line (+) is typically reserved for positive power and the blue line is used for the ground. In our case, we will install a jumper from the 3.3 VDC power pin to a pin in the top row by the red line. This will provide +3.3 VDC power to all pin holes across that row. The bottom (-) blue line of connectors holes will be used for the power ground.

You might have noticed that the GPIO connector also provides a 5 VDC power pin. The higher voltage might be needed for some electronic experiments; however, we just need the 3.3 VDC power for the BSides STEM labs.




One method of wiring power to electronic circuits involves a positive 3.3 VDC on the top and the power ground (GND) at the bottom. This method sometimes helps condense the electronics into a smaller area. Plugging in a jumper from the GPIO GND pin to the bottom blue box row provides the common ground for the electronic wiring.

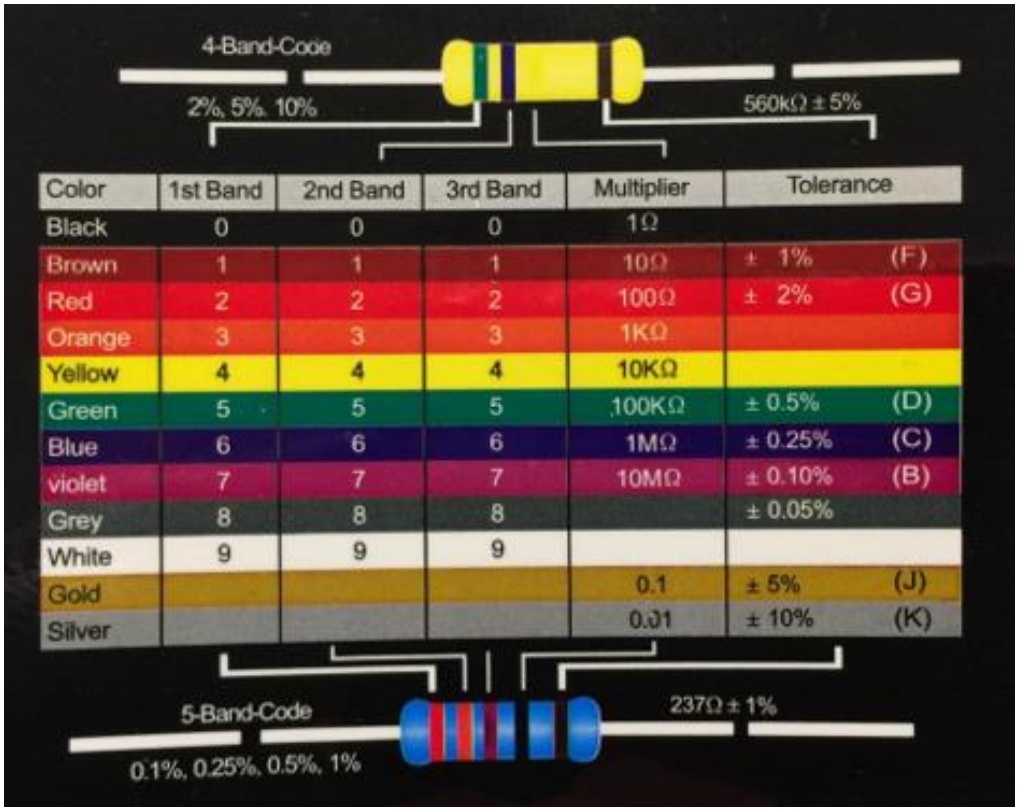
The electronic devices are connected together by using the horizontal rows as in the green and purple columns. The electrical connection is only made between the 5 pin holes in the columns. Note that the electronic devices can be connected directly to the +3.3 VDC or Ground as needed.



It is a good time to wire up the GPIO +3.3 VDC power to the top row (+) with a Red jumper wire as shown in the picture. Then use a Black jumper wire to connect the GPIO GND to the bottom blue line (-) row. Now that the DC power wiring is complete, we can start with the electronics labs.

## RESISTORS

Resistors are frequently used in electronic circuits to limit current, provide a voltage drop, or pull up or down a signal. Most common resistors are barrel shaped with two wire leads coming out of the ends. You will also note several colored bands around the barrel. These bands are a color code to reveal the size, in Ohms, and the percentage error for the ohms value. Each color represents a number value from 0 to 9. The resistor color chart can be found in your electronic box cover for quick reference. The first three bands provide the first three-digit Ohm values and the fourth band indicates the multiplier. The last band is the manufacture's ohm size error in percent. A gold band has up to a 5% resistor size error, so a 100 Ohm resistor could actually be any value between 95 to 105 ohms.



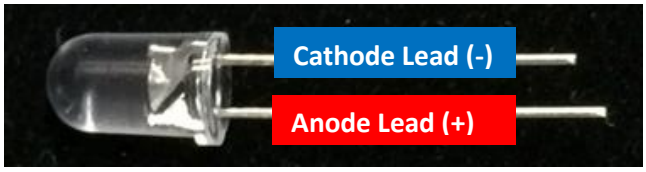

The chart below shows the electronic kit's resistor color bands and ohm values.

| *1st Band* | *2nd Band* | *3rd Band* | *Multiplier* | *Tolerance* | *Ohms* |
|---|---|---|---|---|---|
| Red (2) | Red (2) | Black (0) | Black (x1) | Gold (+/- 5%) | 220 |
| Yellow (4) | Violet (7) | Brown (1) | | Gold (+/- 5%) | 470 |
| Brown (1) | Black (0) | Black (0) | Brown (x10) | Gold (+/- 5%) | 1K |
| Yellow (4) | Violet (7) | Red (2) | | Gold (+/- 5%) | 4.7K |
| Brown (1) | Black (0) | Black (0) | Red (x100) | Gold (+/- 5%) | 10K |
| Brown (1) | Black (0) | Black (0) | Orange (x1K) | Gold (+/- 5%) | 100K |
| Brown (1) | Black (0) | | Green (x100K) | Gold (+/- 5%) | 1M |
| Green (5) | Brown (1) | | Green (x100K) | Gold (+/- 5%) | 5.1M |

## LIGHT EMITTING DIODES (LED)

Light emitting diodes are small silicone-based bulbs which can last well over 10,000 hours. These electronic devices are found in several devices such as computers, TVs, remote controls, digital clocks, and several other devices with light indicators.

LEDs emit light when a DC current is passed through the device from the anode lead (+) side to the cathode lead (-). Visualize the 3.3 VDC power providing a current flow of electrons to the anode (+) LED lead and coming out of the cathode (+) lead. Note that they can burn out if too much current passes through them so a current limiting resistor must be installed in series with the LED. There are a couple of methods to determine which of the two LED leads are the anode (+) or cathode (-).




Most LEDs have different length leads. The long lead is anode (+) and the short lead is the cathode (-) lead.

The second method is to look for the larger of the two metal sections inside of the LED. The largest section is the cathode (-) lead and the smallest section is the anode (+) lead.

Resistors are measured in units of Ohms. The higher the ohms value means that it will restrict more of the DC current. All LEDs require a certain amount of current, measured in milliamps (mA), to initially turn ON. Milliamps are the first three digits to the right of the decimal point. If we have 0.025 A (amps) then we move the decimal point to the right three places and apply the unit label mA to get 25 mA.

There is a maximum current value in which the LED will burn out if the level is exceeded. Another concern to consider is the Raspberry Pi GPIO pins can only provide so much current before the board itself burns up. ***The maximum safe RPi current output value is 20 mA.*** So, you are probably wondering how to calculate the DC current. The main secret is calculating the current flowing through the known resistor value by using Ohms Law:

$$I_{(Current)} = \frac{V_{(Voltage)}}{R_{(Resistance)}}$$

We also need to know the constant voltage drop across the LED when power is applied. ***The voltage drop across the LED is always a constant value.*** That value is usually stated in the LED's electronics' specifications or we can measure it with a digital voltage meter. The constant LED voltage drop for our electronic kit's Red LED is 1.9 VDC. Now we use that known $V_{LED}$ to calculate the voltage across the resistor ($V_R$). So far, we know several required values in order to calculate the current going through the LED:

+3.3 VDC Power ($V_{Supply}$)

LED Voltage Drop ($V_{LED}$) = 1.9 VDC

Current Limiting Resistor Size = 470 ohms (R)

It is time to start filling the Ohm's Law equation to calculate the LED current. The voltage drop across the resistor ($V_R$) is easily calculated from subtracting the voltage drop ($V_{LED}$) from the supply voltage ($V_{Supply}$).

$$V_R = V_{Supply} - V_{LED}$$

$$V_R = 3.3 \text{ VDC} – 1.9 \text{ VDC} = 1.4 \text{ VDC}$$

Now we can calculate the LED current from the formula below.

$$I_{(LED)} = \frac{V_{Supply} - V_{LED}}{R_{(Resistance)}}$$

$$I_{(LED)} = \frac{3.3 \text{ VDC} - 1.9 \text{ VDC}}{470 \text{ Ohms}}$$

$$I_{(LED)} = \frac{1.4 \text{ VDC}}{470 \text{ Ohms}}$$

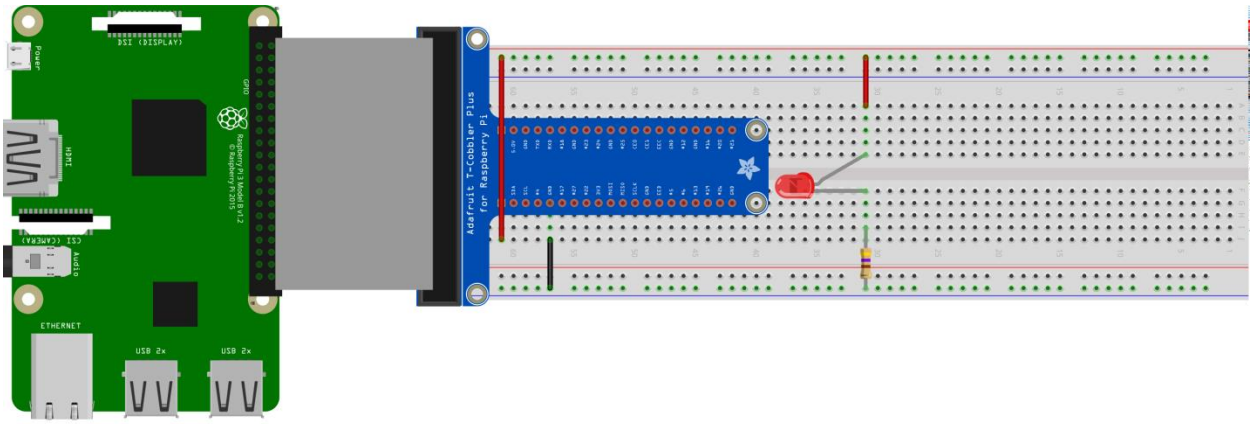
$$I_{(LED)} = 0.00298 \; Amps = 3.0 \; mA$$

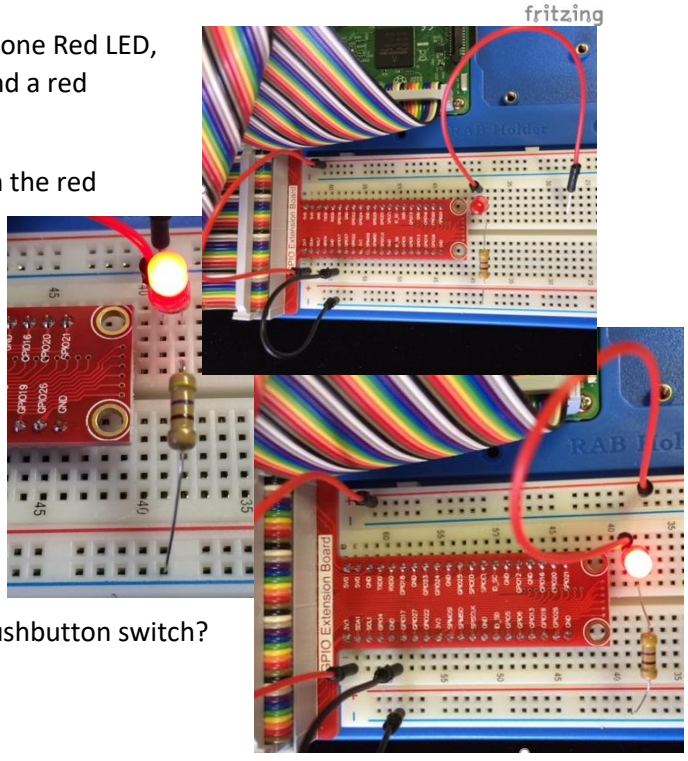
Yeah! The 3.0 mA is lower than the maximum GPIO pin current of 20 mA. ***Another thing to consider is that the Raspberry Pi 3 Model B total maximum current for all GPIO pins is only 50 mA!*** That means that we can only wire up 16 of the Red LEDs to the RPi (16 x 3.0 mA = 48.0 mA). The current going through the LEDs might be able to be lowered by increasing the current limiting resistor size. But there is a possibility that the current is not enough to turn on the LED. To find out, try a larger resistor size and see if the LED lights up. Refer to the chart below for the voltage drop across each LED in the electronic kit. Please note that the 470 Ohm resistors were purchased separately from the electronic kit. The 4.7K Ohm (4,700 Ohm) resistor is being used to dim down the super bright clear LED.

| *LED Color* | $V_{LED}$ | $I_{LED}$ | $R_{Current\ Limiting}$ |
|---|---|---|---|
| Red | 1.9 VDC | 3.0 mA | 470 Ohms |
| Yellow | 2.1 VDC | 4.5 mA | 470 Ohms |
| Green | 2.2 VDC | 2.6 mA | 470 Ohms |
| Clear | 2.7 VDC | 1.3 mA | 4.7K Ohms |

## ELECTRONIC LAB 1 - HARD WIRED LED CIRCUIT

Enough talking about LEDs and resistors, it is time to wire a basic Red LED circuit on the breadboard. The LED will be ON all of the time until the 3.3 VDC power lead is disconnected. Basically, breaking and connecting the jumper cable acts like a switch. Start out assembling the circuit on the far-left side of the breadboard to save room for adding onto the circuit. Note that one of the leads on the current limiting resistor is connected directly to ground. This will save us from using a ground jumper wire which will physically get in the way later.




Assemble the LED circuit as shown in the picture with one Red LED, one 470 Ohm Resistor (Red-Red-Black-Black Bands) and a red jumper wire for the +3.3 VDC power.

Note that the LED anode (+) lead is facing the left with the red jumper connected.

Connect the red jumper wire to the 3.3 VDC power on top the board. The Red LED should be ON, if not, then you might have the LED polarity backwards. Just switch the LED leads to see if that works. If not, then verify the rest of your connections.

The LED can be turned OFF by pulling out the 3.3 VDC jumper wire and turning it back ON by attaching the jumper wire back to 3.3 VDC power.

Now wouldn't it be easier to control the LED with a pushbutton switch?

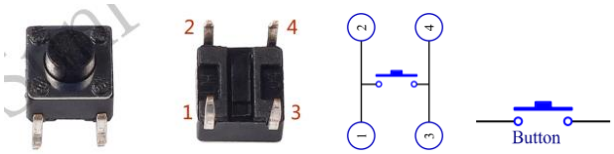
### ***Lab Challenge Questions***

1. Why does the LED turn OFF when you pull out one of the resistor leads?

## ELECTRONIC LAB 2 – HARD WIRED PUSHBUTTON CONTROLLED LED

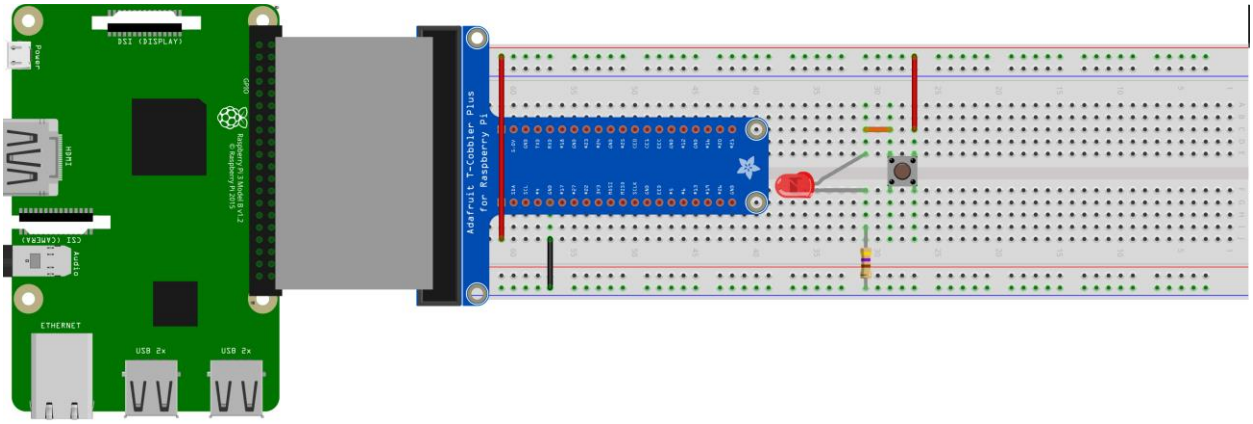
It is time to insert the pushbutton switch into the circuit to connect and disconnect the power to the LED. In electrical circuits we evaluate if the circuit connection is “closed” or “open”. A “closed” connection means that current can flow through the device. An “open” connection means that the current flow is mechanically broken.

The electronic kit includes one normally “open” pushbutton switch. Normally “open” means that the electrical connection between the two terminals is “open” until the button is pressed and then it becomes a “closed” connection. Refer to the push button electrical diagram in far-right picture. This is similar to you connecting and disconnecting the 3.3 VDC jumper lead to the LED circuit.



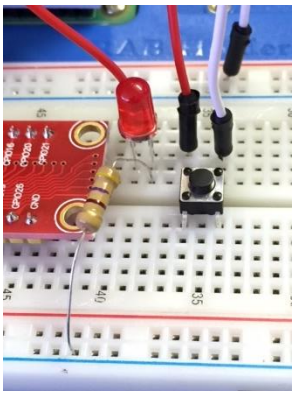

For your information, a normally “closed” pushbutton allows the current to flow through the device until the button is pressed. At that time the pushbutton circuit would be considered “open”.

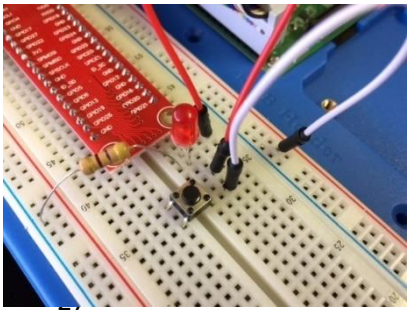
It is easier to remember to use one set of the protruding leads. That is to use pin set 1 & 3 or 2 & 4. Just make sure that those pins are not inserted in the same breadboard electrical columns. Refer to the bottom right picture to see how to insert the pushbutton so that all four pins are in their isolated breadboard connection columns.




fritzing

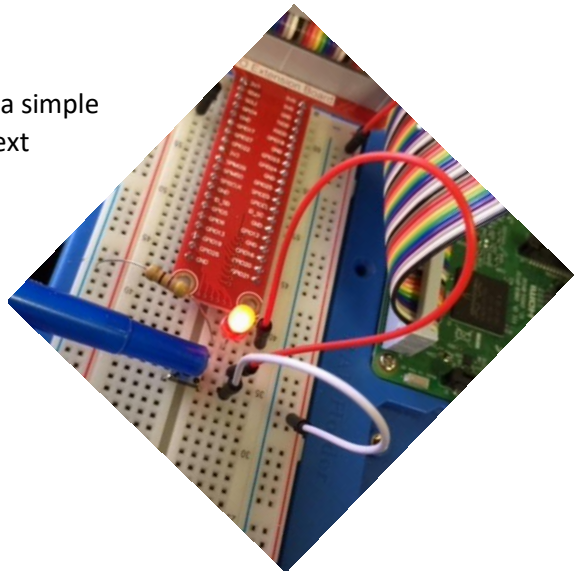
Look at the sides of the pushbutton switch. Notice that two leads come out of two opposite sides. You will be using one of the two sets of these lead for the LED circuit. Carefully press the pushbutton switch over the board’s middle section and to the right side of the breadboard. Also note that one of the two sets of leads should be facing up.





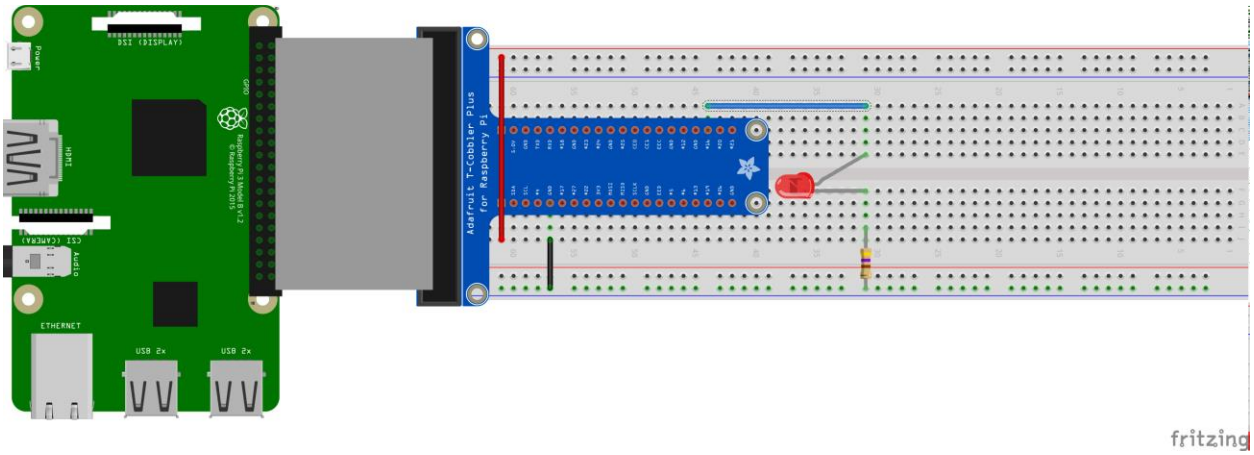
Now move the LED circuit jumper lead end from 3.3 VDC power row to one of the pushbutton leads. Then install a new jumper for the second button lead to the 3.3 VDC power row.

The Red LED should be OFF until you press the pushbutton. This is a simple mechanical switching method to control the current to the LED. Next you will learn how to use the Raspberry Pi and Python programming to control the LED.



## ELECTRONIC LAB 3 - FLASHING LED

It is time now to combine your new electronics circuit knowledge with the Raspberry Pi and Python programming! In this lab we will replace the pushbutton switch with the RPi GPIO pin 16. You will be writing a Python program to command the RPi to supply either 3.3 VDC or 0 VDC to the GPIO pin 16.




Please stop any Python program that might be running by pressing "CTL"+"C". Save the current program as "ELab 3 – RedLEDFlash.py". Now start modifying the old header comments to match the one below and Save!

```
################################################
# Filename: ELab 3 - RedLEDFlash.py
# Developer: BSides Augusta STEM
# Language: Python 3.4
# Description: This program will make the Raspberry Pi flash
# the Red LED ON for 5 seconds and then OFF for 3 seconds.
# This program runs in an infinite loop until CTL-C is pressed.
```

After the header comment, you will write the Python Import Classes section which imports additional Python code needed for our project. The "import time" line retrieves the Python time programming class which includes several different time functions. This will allow us to use the "sleep()" function in the "time class". The sleep function creates a delay in seconds.

The "import RPi.GPIO as GPIO" line of code imports the GPIO function from the Python's Raspberry Pi Class. This will allow you to control the RPi's inputs and outputs pins. Save your Code!

```
################################################
# Import Classes
################################################
# Import Time Class
import time

# Import Raspberry Pi General Purpose Input & Output Class
import RPi.GPIO as GPIO
```

The initialization section is a very important part of the programming code. All variables are declared and set to their starting values. For example, the LEDredOnTime variable is set to 5.0 seconds and the LEDred variable is set to 16 (used for specifying GPIO 16 later on). The GPIO.setmode(GPIO.BCM) function allows Python to refer to the GPIO by pin number. The last initialization line "GPIO.setup(LEDred,GPIO.OUT)" contains our variable LEDred which equals to 16. Python sees this line as "GPIO.setup(16,GPIO.OUT)". The command sets the RPi's GPIO 16 into an output pin for controlling the Red LED.

```
###############################################
# Initialization
###############################################
# Configure LED ON/OFF Times in Seconds
LEDredOnTime = 5.0
LEDredOffTime = 3.0

# Configure LED Variable to GPIO Pins
LEDred = 16

# Config to Refer to GPIO by Pin Number
GPIO.setmode(GPIO.BCM)

# Configure GPIO Pins to Outputs
GPIO.setup(LEDred,GPIO.OUT)
```

Notice how the print() function in the main program informs the user how to shut down the program. It is always a good idea to inform the user how to use and shut down the command line program.

```
###############################################
# Main Program
###############################################
# Run the code until the CTL+C buttons end the program.
# The try...except handles the keyboard interruption
# by pressing the CTL+C buttons. It allows the user
# to shut down the program in a more graceful & safer way.

print ('Press CTL+C to Shut down the Red Flashing LED Program')
try:
    # Run the Loop Continuously
    while(1):
        # Turn ON the Red LED
        GPIO.output(LEDred, True)
        print ('RED LED ON for',LEDredOnTime,'seconds')
        time.sleep(LEDredOnTime)

        # Turn OFF the Red LED
        GPIO.output(LEDred, False)
        print ('RED LED OFF for',LEDredOffTime,'seconds')
        time.sleep(LEDredOffTime)
```

The “try:” statement line is the first part of the “try:” “except” error handling for Python. The main code is placed after the “try:” statement and special keyboard and error code is placed after the “exception”. We are using the try-exception error handling code to allow the program to be stopped by pressing the keyboard keys “CTL”+”C” and safely shut down the program.

The flashing LED program uses the “while(1)” loop which runs the program infinitely. Did you notice the “1” parameter being passed into the while() function? This function evaluates conditional statements like if variable X < 17, true, or in our case just 1. The code within the while() is ran if the conditional statement evaluates to true or false, or 1 or 0. In this case, “while(1) is always true so the code within the loop will always run.

The first thing our code does is to turn ON the Red LED by outputting 3.3 VDC out of GPIO 16 to the Red LED circuit. Then the user is formed that the Red LED is ON through a print statement. The LED is kept ON for 5.0 seconds with the “time.sleep(LEDredOnTime)” function. Remember that Python sees this line statement as “time.sleep(5.0)”.

After the 5.0 second ON time, the Red LED is then turned OFF for 3.0 seconds with similar code “time.sleep(LEDredOffTime)” or seen by Python as “time.sleep(3.0)”.

The Red LED will continue to flash ON and OFF until the program is stopped with “CTL”+”C”.

```
# The exception section is called whenever a keyboard interrupt happens.
# This allows you to control how gracefully the program shuts down.
except KeyboardInterrupt:

    # Safety Shut Down Configuration for the Raspberry Pi.
    # The cleanup() method sets all initialized GPIO outputs back to
    # input pins. This will prevent you from accidentally shorting out
    # an output pin to ground during the electronic breadboarding
    # process. All GPIO pins should be configured to inputs when the
    # program is not running.
    GPIO.cleanup()

    # Output the Shut Down Message
    print ('Program has been properly shut down. Goodbye!')
```

The code after the “except KeyboardInterrupt:” is executed only after a keyboard interrupt like pressing the “CTL”+”C” keys. We use this section to gracefully shut down the program and reconfigure the RPi. The print statement lets the user know that the program was properly shut down. In this section the Raspberry Pi’s GPIO pins are set to inputs for safety purposes with the “GPIO.cleanup()” function. ***This will prevent you from accidently shorting out an output pin to ground during the circuit build.***

Save the program and run it. The screen output should be similar to the one below. Verify that the program screen output states that the LED is ON or OFF for the right amount of time. Then shut down the program with "CTL"+"C". This will set all of the GPIO pins to outputs so that we can safely connect the RPi GPIO pin 16 to out Red LED circuit.

```
>>> ================================ RESTART ================================
>>>
Press CTL+C to Shut Down the Red Flashing LED Program
RED LED ON for 5.0 seconds
RED LED OFF for 3.0 seconds
RED LED ON for 5.0 seconds
RED LED OFF for 3.0 seconds
RED LED ON for 5.0 seconds
RED LED OFF for 3.0 seconds
RED LED ON for 5.0 seconds
RED LED OFF for 3.0 seconds
Program has been properly shut down. Goodbye!
>>>
```

## ***Lab Challenge Questions***

1. Which variables would you change to make the LED stay ON for 8 seconds and OFF for 2 seconds?

2. What does the input parameters x and y for the function GPIO.output(x,y) represent?

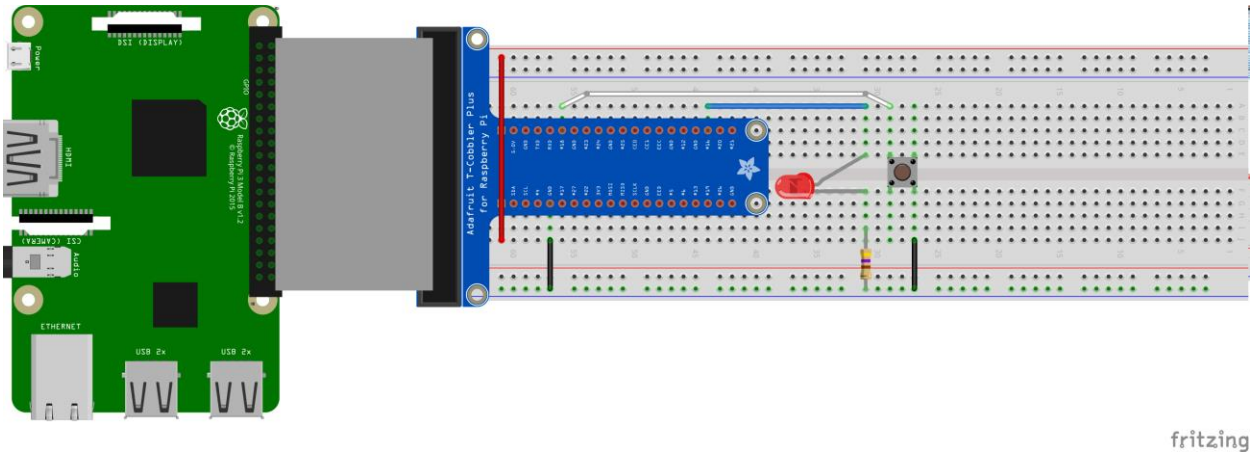
3. Why is it important to end a RPi Python program for electronics with GPIO.cleanup()?

4. What does the "as" in "import RPi.GPIO as GPIO" do for this program?
   Need a hint? Replace all "GPIO" with "Oranges" except for the GPIO in "import RPi.GPIO". Why does the code still run?

   Use the IDLE editor's top menu Edit->Replace to change back all "Oranges" back to "GPIO". Save and run the code to verify that it still works.

## ELECTRONIC LAB 4 –PUSHBUTTON INPUT CONTROLLED LED

In the Electronic Lab 2 – Hardwired Pushbutton Controlled LED you learned how to use a normally "open" pushbutton to control a Red LED. A hardwired LED control circuit requires a person to actually press the pushbutton to turn ON the LED. Another way to control the LED is to run the pushbutton into a GPIO input pin and have the RPi control the LED through a GPIO output pin.




Save "ELab 3 – RedLEDFlash.py" as "ELab 4 - PushbuttonLED.py" and modify the code to match the details below. The Import Classes section will remain the same for this project.

```
################################################
# Filename: ELab 4 - PushbuttonLED.py
# Developer: BSides Augusta STEM
# Language: Python 3.4
# Description: The Raspberry Pi detects when the pushbutton
# is pressed and then turns ON the LED.
# This program runs in an infinite loop until CTL-C is pressed.

################################################
# Import Classes
################################################
# Import Time Class
import time

# Import Raspberry Pi General Purpose Input & Output Class
import RPi.GPIO as GPIO
```

Note the new lines of Python code in the Initialization Section which handles the pushbutton input into the RPi. The variable "Pushbutton = 18" will be used to define pin 18 as a GPIO Input in the "GPIO.setup(Pushbutton, GPIO.IN, pull_up_down = PUD_UP) statement. The first and second parameters in the GPIO.setup function sets GPIO pin 18 to an Input port. The third parameter configures the RPi to use internal pull-up resistor on pin 18. So, what does this really mean?

Pull-up resistors provide very clean and stable High and Low signals to the input pins. These pull-up resistors can be installed on the breadboard or it can be done internally with Raspberry Pi's internal resistor. The "pull_up_down = PUD_UP" parameter enables the internal pull-up resistor to be used. This is done with an internal RPi pull-up resistor wired to 3.3 VDC to produce a solid High (3.3 VDC) signal to the RPi pin 18 when the button is not pressed. Pressing the pushbutton shorts out GPIO 18 to ground which creates a Low signal.

The pushbutton is wired with one lead to ground and the other lead going to GPIO 18. The signal at GPIO will be High (+3.3 VDC) when the button is not pressed. In other words, the internal pull-up resistor applies the 3.3 VDC to the input pin.

```
##############################################
# Initialization
##############################################
# Configure Variable for GPIO Pins
Pushbutton = 18
LEDred = 16

# Configure Control Variables
LEDonDelay = 3.0

# Config to Refer to GPIO by Pin Number
GPIO.setmode(GPIO.BCM)

# Configure GPIO Pins to Inputs/Outputs
GPIO.setup(Pushbutton,GPIO.IN,pull_up_down=GPIO.PUD_UP)
GPIO.setup(LEDred,GPIO.OUT)
```

The signal to GPIO 18 will be pulled down (shorted to ground) to 0 VDC when the pushbutton is pressed.

```
###############################################
# Main Program
###############################################
# Run the code until the CTL+C buttons end the program.
# The try...except handles the keyboard interruption
# by pressing the CTL+C buttons. It allows the user
# to shut down the program in a more graceful & safer way.

print ('Press CTL+C to Shut down the Pushbutton LED Program')
try:
    # Run the Loop Continuously
    while(True):

        # Check Input State
        input_state = GPIO.input(Pushbutton)

        # Check for Button Pushed
        if input_state == False:

            # Turn ON the Red LED
            GPIO.output(LEDred, True)
            print('Button Pressed - Turn LED ON for',LEDonDelay,'seconds')
            time.sleep(LEDonDelay)

            # Turn OFF the Red LED
            GPIO.output(LEDred, False)
            print('Reset LED to OFF')
```

The main pushbutton program runs in an infinite loop. The Python statement “GPIO.input(18)” function checks the signal value on GPIO 18 and sets a True or False value to variable “input_state”. Then that variable is evaluated for “False” which means the pushbutton was pressed which created a Low signal

```
# The exception section is called whenever a keyboard interrupt happens.
# This allows you to control how gracefully the program shuts down.
except KeyboardInterrupt:

    # Safety Shut Down Configuration for the Raspberry Pi.
    # The cleanup() method sets all initialized GPIO outputs back to
    # input pins. This will prevent you from accidentally shorting out
    # an output pin to ground during the electronic breadboarding
    # process. All GPIO pins should be configured to inputs when the
    # program is not running.
    GPIO.cleanup()

    # Output the Shut Down Message
    print ('Program has been properly shut down. Goodbye!')
```

on pin 18. The LED is turned ON for the defined “LEDonDelay” time of 3.0 seconds after the button is pressed. Then the LED is turned OFF and the program continues in an infinite loop.

## ***Lab Challenge Questions***

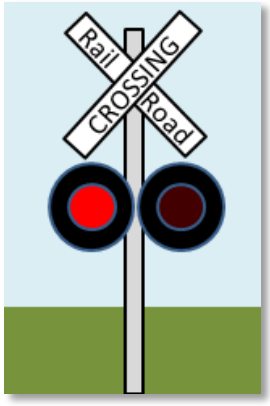
1. How do you change the program to move the pushbutton input pin to GPIO 23?

2. The user wants the LED ON time to be 0.5 seconds. How can you make that change?

3. Change the “if input_state == False:” to “if input_state == True:” and run the program and press the button. Explain how the program and circuit is working now.

4. What happens if you switch the logic from “True” to “False” in the “while(False):” conditional statement. Try it if you don’t know the answer.

## ELECTRONIC LAB 5 – FLASHING RAILROAD CROSSING LIGHTS

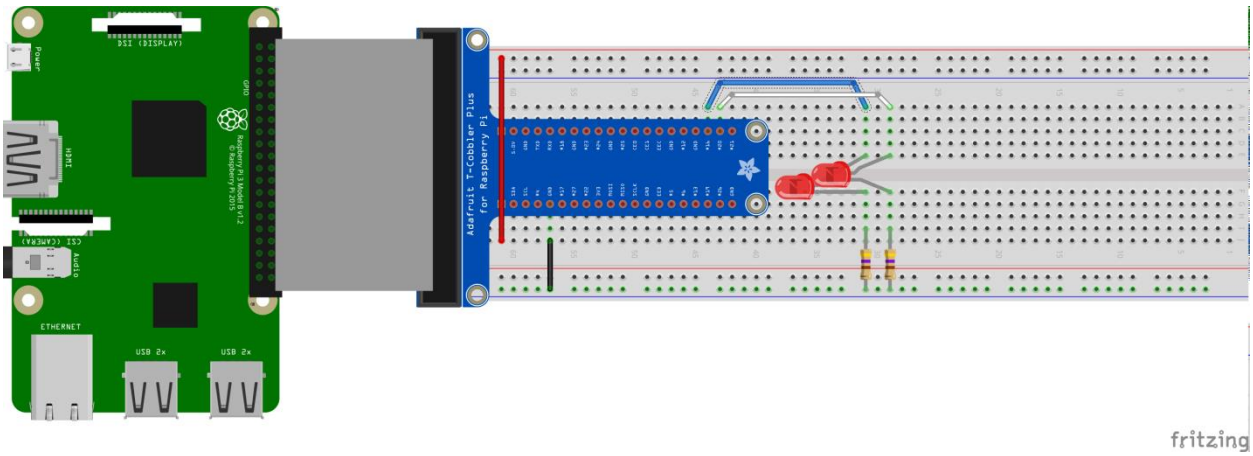



You just learned how to make one flashing Red LED. The challenge now is to add a second Red LED to make a railroad crossing sign. Most programmers and electrical engineers follow a list of requirements to develop the program and electronic circuit. Open the previously created program "ELab 3 – RedLEDFlash.py" and save it as "ELab 5 – RailroadCrossing.py". Follow the requirement list below to make code and electronic modifications as needed.

The Railroad Crossing Sign Program Requirements are:

- Use Two Red LEDs
- Assign GPIO 16 to LED 1
- Assign GPIO 20 to LED 2
- Flash Back and Forth for a Total 18 seconds. Only One LED ON at a Time.
- Flashing LEDs should stay ON for 3 seconds and OFF for 3 seconds.
- Both LEDs should stay ON after 18 seconds of flashing for 10 seconds.
- The program should run in an infinite loop.
- Exit the program by "CTL"+"C" to generate the keyboard interrupt.

Developers usually break down complex projects into small milestones. Taking on all of the railroad crossing sign controller would be too much to take on, so we will break this project into smaller software and electronics milestones. The first milestone is to create a program to continuously flash both Red LED circuits. We will start making the code flashing LED code and then wire up the two Red LED circuits.




The first code change modification is to the Header Comments as shown below. Both the time and RPi.GPIO classes are still needed for this program.

```
##############################################
# Filename: ELab 5 - RailroadCrossing.py
# Developer: BSides Augusta STEM
# Language: Python 3.4
# Description: This program will make the Raspberry Pi 3
# a railroad controller. The two Red LEDs should flash in
# alternate. Each staying on for 3 seconds at a time. After
# 18 seconds, both LEDs should stay on for 10 seconds.
# This program runs in an infinite loop until CTL-C is pressed.

##############################################
# Import Classes
##############################################
# Import Time Class
import time

# Import Raspberry Pi General Purpose Input & Output Class
import RPi.GPIO as GPIO
```

The initialization section will need some changes to support the second Red LED. We need to change the variable LEDredOnTime from 5.0 to 3.0 seconds. Variable LEDBothOnTime was created and set to 10 seconds. Then we change variable name LEDred to LEDred1 for program control clarity.

*Note how the variable LEDredOffTime was commented out by inserting the pound sign (#) in front of the variable statement. This is a common way to remove executable code without deleting the entire line. It is a great way to quickly troubleshoot your code by just commenting out the suspected lines.*

Next the GPIO 20 needs to be configured as an output pin. Then variable LEDred2 was created and set to 20 and used in the GPIO.setup(LEDred2,GPIO.OUT).

```
##############################################
# Initialization
##############################################
# Configure LED ON/OFF Times in Seconds
LEDredOnTime = 3.0
# LEDredOffTime = 3.0
LEDBothOnTime = 10.0

# Configure LED Variable to GPIO Pins
LEDred1 = 16
LEDred2 = 20

# Config to Refer to GPIO by Pin Number
GPIO.setmode(GPIO.BCM)

# Configure GPIO Pins to Outputs
GPIO.setup(LEDred1,GPIO.OUT)
GPIO.setup(LEDred2,GPIO.OUT)
```

The main program will very much resemble the Red Flashing LED program. This time when Red LED1 is turned ON, Red LED2 will be turned OFF for 3 seconds. This will alternate turning the LED ON and OFF indefinitely.

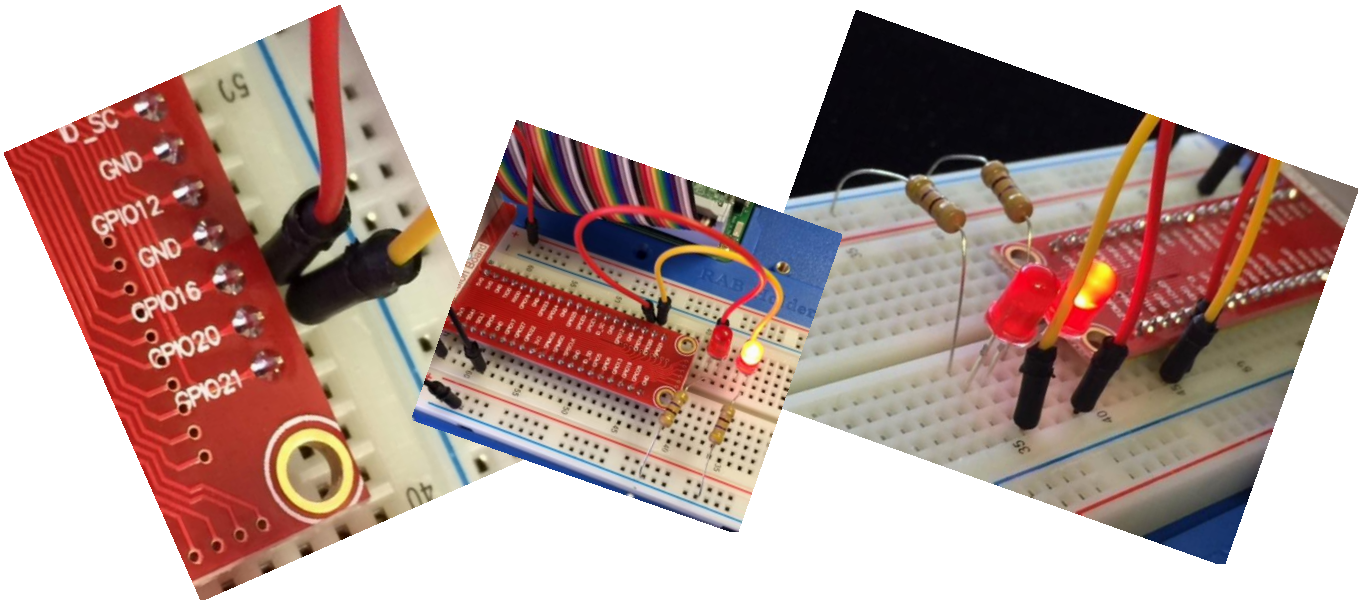
```
##############################################
# Main Program
##############################################
# Run the code until the CTL+C buttons end the program.
# The try...except handles the keyboard interruption
# by pressing the CTL+C buttons. It allows the user
# to shut down the program in a more graceful & safer way.

print ('Press CTL+C to Shut down the Red Flashing LED Program')
try:
    # Run the Loop Continuously
    while(1):
        # Red LED1 ON & Red LED2 OFF
        GPIO.output(LEDred1, True)
        GPIO.output(LEDred2, False)
        print ('Red LED1 ON & Red LED2 OFF for',LEDredOnTime,'seconds')
        time.sleep(LEDredOnTime)

        # Red LED1 OFF & Red LED2 ON
        GPIO.output(LEDred1, False)
        GPIO.output(LEDred2, True)
        print ('Red LED1 OFF & Red LED2 ON for',LEDredOnTime,'seconds')
        time.sleep(LEDredOnTime)
```

The try-exception for the keyboard interrupt will be reused entirely to allow shutting down the program with “CTL”+”C”.

```
# The exception section is called whenever a keyboard interrupt happens.
# This allows you to control how gracefully the program shuts down.
except KeyboardInterrupt:

    # Safety Shut Down Configuration for the Raspberry Pi.
    # The cleanup() method sets all initialized GPIO outputs back to
    # input pins. This will prevent you from accidentally shorting out
    # an output pin to ground during the electronic breadboarding
    # process. All GPIO pins should be configured to inputs when the
    # program is not running.
    GPIO.cleanup()

    # Output the Shut Down Message
    print ('Program has been properly shut down. Goodbye!')
```

The first attempt of programming the dual flashing LEDs is complete. Now it is time to wire the two separate Red LED circuits. You already have the Red LED1 circuit working off of RPi’s GPIO16. Build another Red LED circuit and put it next to the Red LED1 circuit and connect it to GPIO 20.

Run the dual alternating Red LED program to make sure the lights bounce back and forth. The screen output should look as shown below.



```
>>> ================================ RESTART ================================
>>>
Press CTL+C to Shut Down the Red Flashing LED Program
Red LED1 ON & Red LED2 OFF for 3.0 seconds
Red LED1 OFF & Red LED2 ON for 3.0 seconds
Red LED1 ON & Red LED2 OFF for 3.0 seconds
Red LED1 OFF & Red LED2 ON for 3.0 seconds
Red LED1 ON & Red LED2 OFF for 3.0 seconds
Red LED1 OFF & Red LED2 ON for 3.0 seconds
Program has been properly shut down. Goodbye!
>>>
```

We should be very happy that the alternating Red LEDs are flashing alternatively ON for 3 seconds at a time. Now we can modify the code to meet the alternating flashing for 18 seconds and then turn on both LEDs for 10 seconds. Then the entire sequence starts over indefinitely.

Let's analyze our program a little bit more. Each ON and OFF sequence takes 3.0 seconds each for a total of 6.0 seconds for one ON/OFF cycle. We need them to flash for a total of 18 seconds. So how many flashing ON/OFF cycles does it take to fill up 18 seconds? We just take 18 seconds divided by 6 seconds gives us 3 cycles of programming loops.

Go into the Main Program section and add the "for x in range(3):" above the Red LED ON/OFF lines of code with the line comment. Note the indentation of the Python code. This is very important for defining which lines belong within the "try:", "while", "for-in" sections.

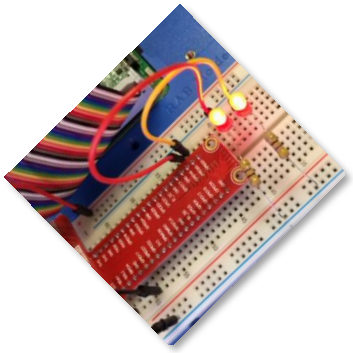
Now insert a print line after the "for x in range(3):" statement. This will print to the screen the loop iteration from 0 to 2.

Lastly, the both Red LEDs ON for 10 seconds need to be programmed after the third loop iteration is completed. Make sure that the Red LED ON statements are lined up with the while() loop. Again, we inserted a line comment for the section and printed to the screen that both LEDs are ON.

```
##############################################
# Main Program
##############################################
# Run the code until the CTL+C buttons end the program.
# The try...except handles the keyboard interruption
# by pressing the CTL+C buttons. It allows the user
# to shut down the program in a more graceful & safer way.

print ('Press CTL+C to Shut down the Red Flashing LED Program')
try:
    # Run the Loop Continuously
    while(1):

        # Run 3 Loops of LEDs ON/OFF
        for x in range(3):
            print('Loop x =',x)

            # Red LED1 ON & Red LED2 OFF
            GPIO.output(LEDred1, True)
            GPIO.output(LEDred2, False)
            print ('Red LED1 ON & Red LED2 OFF for',LEDredOnTime,'seconds')
            time.sleep(LEDredOnTime)

            # Red LED1 OFF & Red LED2 ON
            GPIO.output(LEDred1, False)
            GPIO.output(LEDred2, True)
            print ('Red LED1 OFF & Red LED2 ON for',LEDredOnTime,'seconds')
            time.sleep(LEDredOnTime)

        # Red LED1 OFF & Red LED2 ON
        GPIO.output(LEDred1, True)
        GPIO.output(LEDred2, True)
        print ('Both LEDs ON for', LEDBothOnTime,'seconds\n')
        time.sleep(LEDBothOnTime)

# The exception section is called whenever a keyboard interrupt happens.
# This allows you to control how gracefully the program shuts down.
except KeyboardInterrupt:

    # Safety Shut Down Configuration for the Raspberry Pi.
    # The cleanup() method sets all initialized GPIO outputs back to
    # input pins. This will prevent you from accidentally shorting out
    # an output pin to ground during the electronic breadboarding
    # process. All GPIO pins should be configured to inputs when the
    # program is not running.
    GPIO.cleanup()

    # Output the Shut Down Message
    print ('Program has been properly shut down. Goodbye!')
```

Save the program and do a test run. Did your LEDs flash back and forth three times and then both turn ON for 10 seconds? Your program output and circuit should look like something below.

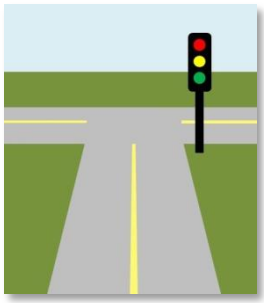
```
>>> ================================ RESTART ================================
>>>
Press CTL+C to Shut Down the Red Flashing LED Program
Loop x = 0
Red LED1 ON & Red LED2 OFF for 3.0 seconds
Red LED1 OFF & Red LED2 ON for 3.0 seconds
Loop x = 1
Red LED1 ON & Red LED2 OFF for 3.0 seconds
Red LED1 OFF & Red LED2 ON for 3.0 seconds
Loop x = 2
Red LED1 ON & Red LED2 OFF for 3.0 seconds
Red LED1 OFF & Red LED2 ON for 3.0 seconds

Both Red LEDs ON for 10.0 seconds

Loop x = 0
Red LED1 ON & Red LED2 OFF for 3.0 seconds
Red LED1 OFF & Red LED2 ON for 3.0 seconds
Loop x = 1
Red LED1 ON & Red LED2 OFF for 3.0 seconds
Program has been properly shut down. Goodbye!
>>>
```



## ***Lab Challenge Questions***

1. Your employer wants both Red LEDs to be ON for 3 minutes. Which variable would you change? What is the math equation that you would be set equal to the chosen variable to 3 minutes of time?
2. How would you change the program to run 5 periods of ON/OFF with each LED ON time of 5 seconds?

# ELECTRONIC LAB CHALLENGES



## ELECTRONIC LAB CHALLENGE 1 – SINGLE TRAFFIC STOP LIGHT

It is time for you to spread your wings and soar out of the nest! Don't worry, you will not crash and burn. Make the project more manageable by tackling small parts of the requirements. Start with the Red light, then the Yellow and finally the Green light. Below are the programming requirements:

**Filename: ELC 1 - TrafficStopLight.py**
**Choose one of your previous programs to reuse the code.**

**LEDS: Red, Yellow and Green**

**3 - 470 Ohm LED Current Limiting Resistors**

**GPIO Output Pin Assignments**

- **GPIO 16 – Red LED Output**
- **GPIO 20 – Yellow LED Output**
- **GPIO 21 – Green LED Output**

**Place the LEDs in Order on the Breadboard: Red, Yellow & Green**

- **Traffic Stop Light Sequence Logic**

| *Sequence* | *Traffic Light 1* |
|---|---|
| 1 | Red<br>ON 5 Seconds |
| 2 | Green<br>ON 3 Seconds |
| 3 | Yellow<br>ON 2 Seconds |

**Only One LED ON at a Time**

**Print to Screen**

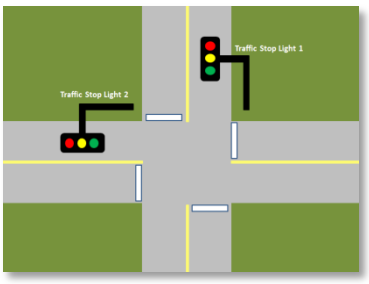
- **"Red Light ON for XX seconds"**
- **"Green Light ON for XX seconds"**
- **"Yellow Light ON for XX seconds"**

**Exit Program with Keyboard "CTL"+"C". Inform the user by printing the program shut down instructions when your program starts.**

**Program Runs in an Infinite Loop**

***Show off your RPi traffic light control system to your BSides STEM team members when you are done.***

## ELECTRONIC LAB CHALLENGE 2 – TWO TRAFFIC LIGHT STOP LIGHTS




Let's build upon the previous single traffic light control system. Your next challenge is to add a second traffic light for the four-way intersection. That means that both lights have to run in opposite mode to avoid automobile collisions.

**Filename: ELC 2 - 2TrafficStopLights.py**
**Reuse the Code & Electronic Circuit Electronic Lab Challenge 1 – Single Traffic Stop Light Code and Electronic Circuit**

**Use Same Requirements as Electronic Circuit Electronic Lab Challenge 1 – Single Traffic Stop Light**

**GPIO Output Pin Assignments for 2nd Traffic Light**

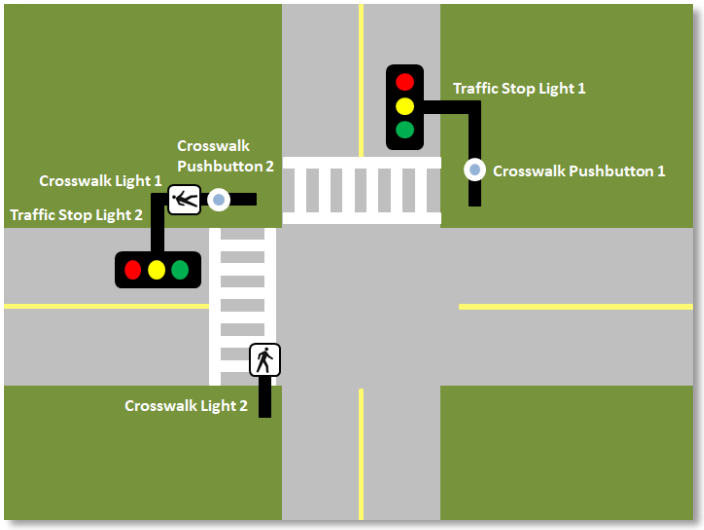
- **GPIO 13 – Red LED Output**
- **GPIO 19 – Yellow LED Output**
- **GPIO 26 – Green LED Output**

**Traffic Stop Light's Sequence Logic**

| *Sequence* | *Traffic Light 1* | *Traffic Light 2* |
|---|---|---|
| 1 | Red<br>5 Seconds | Green<br>3 Seconds |
| 2 | | Yellow<br>2 Seconds |
| 3 | Green<br>3 Seconds | Red<br>5 Seconds |
| 4 | Yellow<br>2 Seconds | |

***Remember to show off your working traffic light control System!***

## ELECTRONIC LAB CHALLENGE 3 – TWO TRAFFIC STOP LIGHTS WITH CROSSWALK PBS & LIGHTS




You should now have a working two Traffic Lights control system. Now the intersection needs a pedestrian crosswalk light with a pushbutton request. Add on the following list of requirements shown below to the new traffic light control system.

**Filename: ELC 3 - 2TrafficStopLightsCW.py**
**Use the Previous Electronic Lab Challenge 2 Two Traffic Stop Lights lab Code & Electronic Circuit**

**Use the Previous Traffic Light Lab Challenge Requirements**

**GPIO Output Pin Assignments for 1st Traffic Light Crosswalk LED Which Indicates It is OK to Cross when ON.**

**Traffic Stop Light 1 Crosswalk GPIO Pin Assignments**

- **GPIO 12 – Clear LED Output**
- **GPIO 18 – Crosswalk Pushbutton Input**

**Traffic Stop Light 2 Crosswalk GPIO Pin Assignments**

- **GPIO 5 – Clear LED Output**
- **GPIO 22 – Crosswalk Pushbutton Input**

**Crosswalk Push Button 1 Logic**

- **When Pressed, Add 10 Seconds to Red Light 1**

**Crosswalk Push Button 2 Logic**

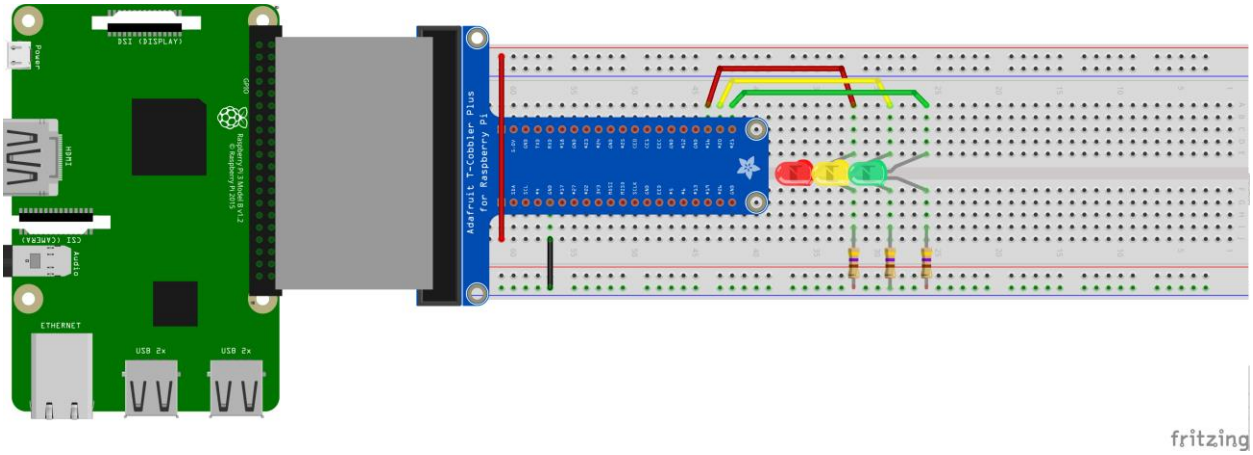
- **When Pressed, Add 10 Seconds to Red Light 2**

**Use the 4.7K Ohm Resistors for the Clear LEDs.**

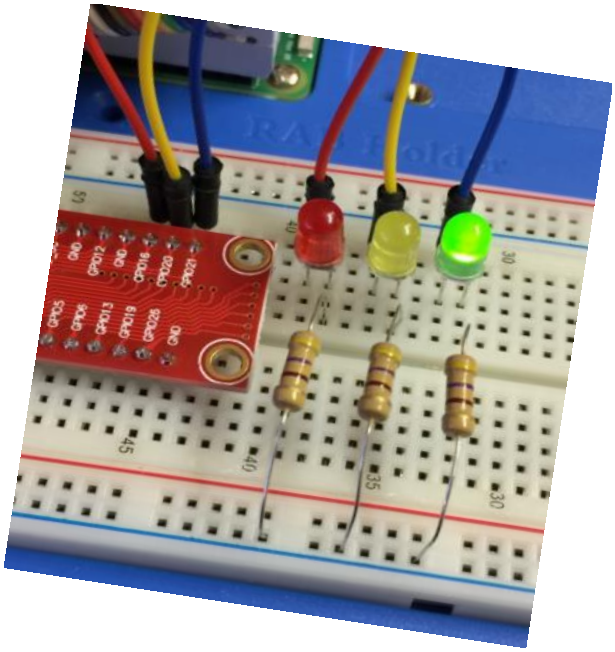
***Demonstrate your traffic light control system!***

# ELECTRONIC LAB CHALLENGE SOLUTIONS

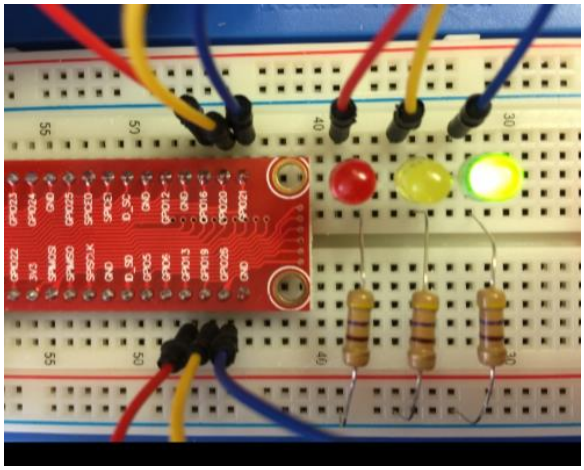
## ELECTRONIC LAB CHALLENGE 1: TRAFFIC STOP LIGHT




```
################################################
# Filename: ELC 1 - TrafficStopLight.py
# Developer: BSides Augusta STEM
# Language: Python 3.4
# Description: This program will make the Raspberry Pi three GPIO
# output ports control three LEDs.  The Red, Yellow and Green LEDs
# turn ON/OFF like a street traffic light.
# This traffic light control program runs in an infinite loop.

################################################
# Import Classes
################################################
# Import Time Class
import time

# Import Raspberry Pi General Purpose Input & Output Class
import RPi.GPIO as GPIO

################################################
# Initialization
################################################
# Configure LED ON Time in Seconds
LEDredOnTime = 5.0
LEDyellowOnTime = 2.0
LEDgreenOnTime = LEDredOnTime - LEDyellowOnTime

# Configure LED Variables to GPIO Pins
LEDred = 16
LEDyellow = 20
LEDgreen = 21
```



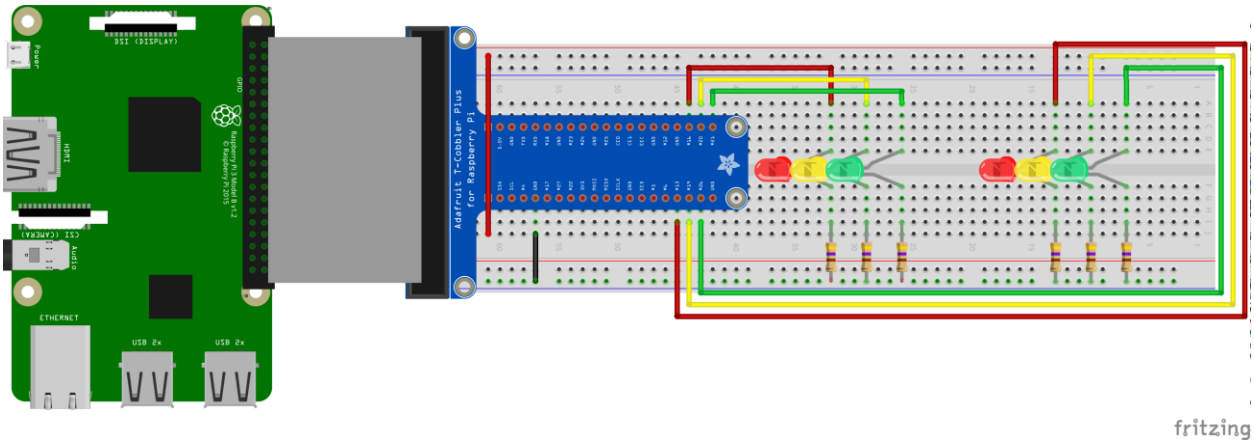
```
# Config to Refer to GPIO by Pin Number
GPIO.setmode(GPIO.BCM)

# Configure GPIO Pins to Outputs
GPIO.setup(LEDgreen,GPIO.OUT)
GPIO.setup(LEDred,GPIO.OUT)
GPIO.setup(LEDyellow,GPIO.OUT)

################################################
# Main Program
################################################
# Run the code until the CTL+C buttons end the program.
# The try...except handles the keyboard interruption
# by pressing the CTL+C buttons. It allows the programmer
# to shut down the program in a more graceful & safer way.

print ('Press CTL+C to Shut Down the Traffic Light Program')
try:
    # Run the Loop Continuously
    while(1):
        # Turn ON Only the Red Light
        GPIO.output(LEDred,True)
        GPIO.output(LEDyellow,False)
        GPIO.output(LEDgreen,False)
        print ('RD1 ON for',LEDredOnTime,'seconds')
        time.sleep(LEDredOnTime)

        # Turn ON Only the Green Light
        GPIO.output(LEDred,False)
        GPIO.output(LEDyellow,False)
        GPIO.output(LEDgreen,True)
        print ('GN1 ON for',LEDgreenOnTime,'seconds')
        time.sleep(LEDgreenOnTime)

        # Turn ON Only the Yellow Light
        GPIO.output(LEDred,False)
        GPIO.output(LEDyellow,True)
        GPIO.output(LEDgreen,False)
        print ('YL ON for',LEDyellowOnTime,'seconds\n')
        time.sleep(LEDyellowOnTime)
```



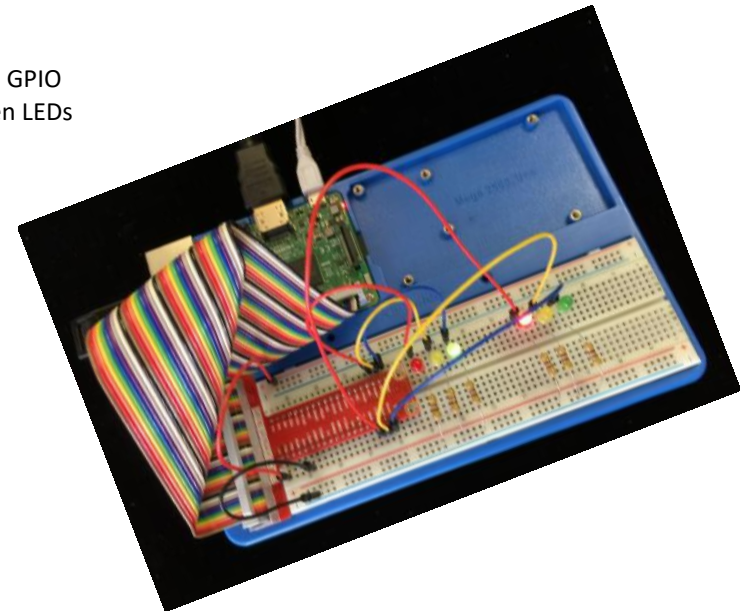
```
# The exception section is called whenever a keyboard interrupt happens.
# This allows you to control how gracefully the program shuts down.
except KeyboardInterrupt:

    # Safety Shut Down Configuration for the Raspberry Pi.
    # The cleanup() method sets all initialized GPIO outputs back to
    # input pins. This will prevent you from accidentally shorting out
    # an output pin to ground during the electronic breadboarding
    # process. All GPIO pins should be configured to inputs when the
    # program is not running.
    GPIO.cleanup()
```

```
# Output the Shut Down Message
print ('Program has been properly shut down. Goodbye!')
```

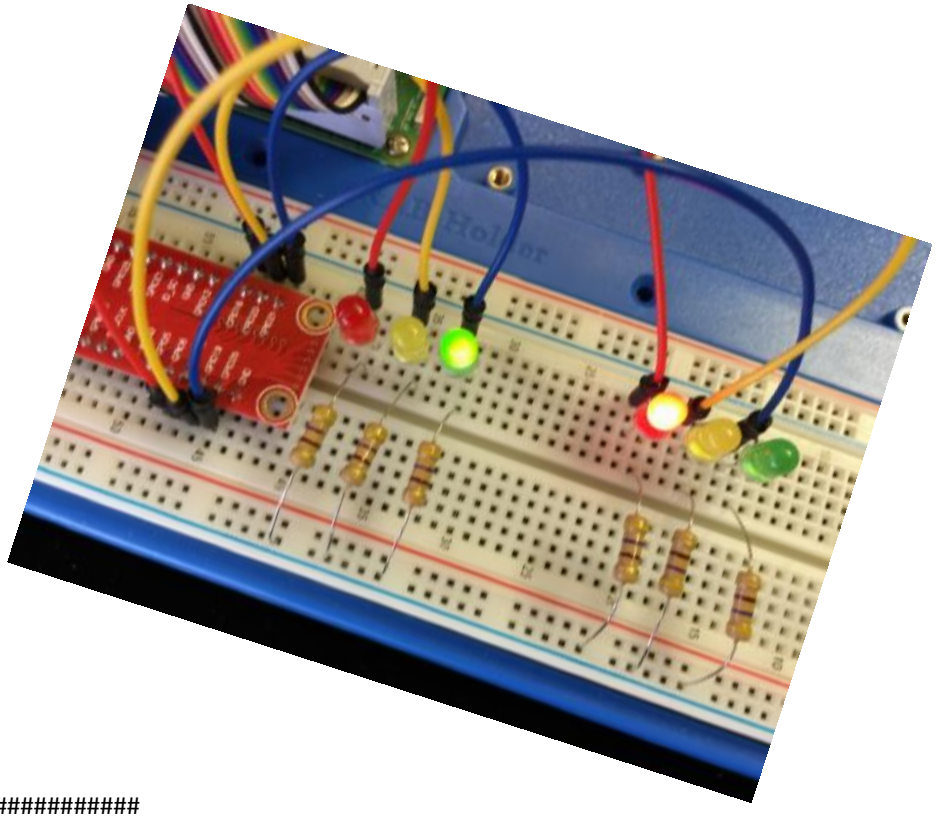
## ELECTRONIC LAB CHALLENGE 2: TWO TRAFFIC STOP LIGHTS




```
################################################
# Filename: ELC 2 - 2TrafficStopLights.py
# Developer: BSides Augusta STEM
# Language: Python 3.4
# Description: This program will make the Raspberry Pi three GPIO
# output ports control three LEDs.  The Red, Yellow and Green LEDs
# turn ON/OFF like a street traffic light.
# This traffic light control program runs in an infinite loop.
```



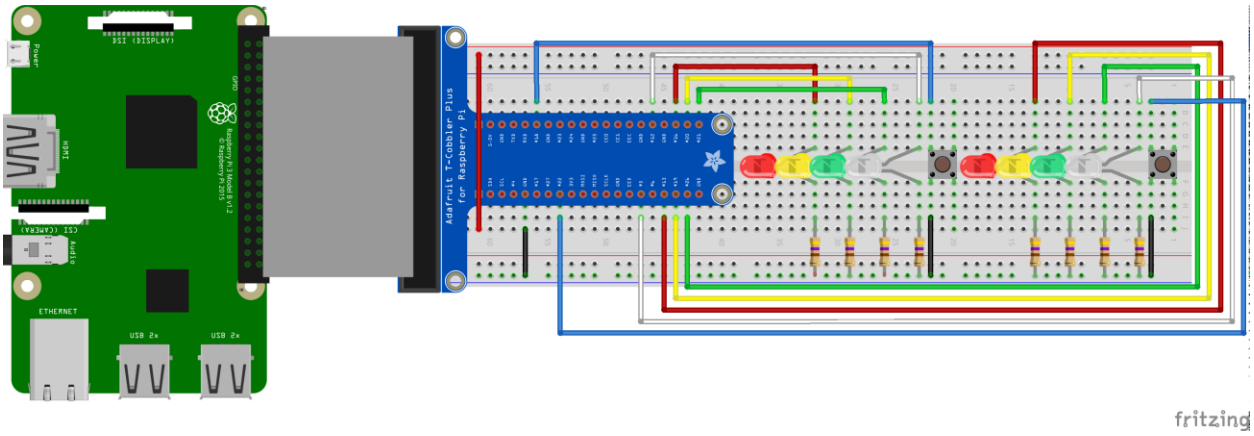
```
################################################
# Import Classes
################################################
# Import Time Class
import time

# Import Raspberry Pi General Purpose Input & Output Class
import RPi.GPIO as GPIO

################################################
# Functions
################################################
# Traffic Lights Control
def TrafficLightsCtl(RedPin,RedTF,YellowPin,YellowTF,GreenPin,GreenTF):
    GPIO.output(RedPin,RedTF)
    GPIO.output(YellowPin,YellowTF)
    GPIO.output(GreenPin,GreenTF)

################################################
# Initialization
################################################
# Configure LED ON Time in Seconds
LEDredOnTime = 5.0
LEDyellowOnTime = 2.0
LEDgreenOnTime = LEDredOnTime - LEDyellowOnTime

# Configure LED Variables to GPIO Pins
```



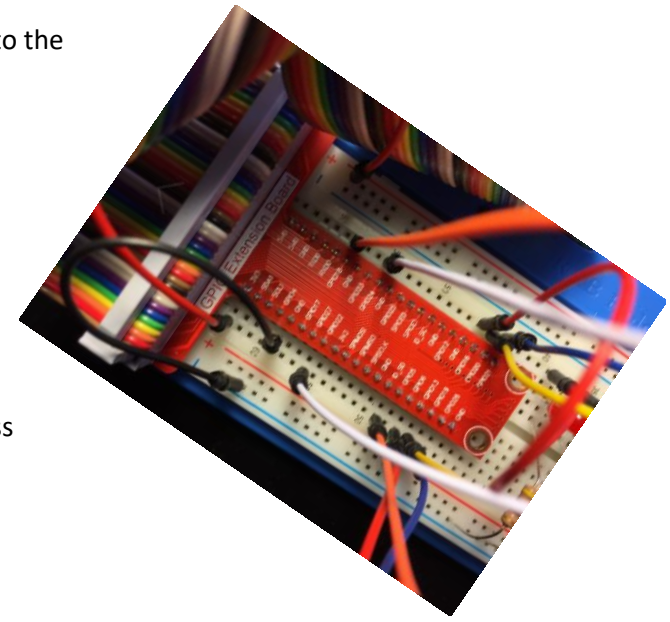
```
# Traffic Light 1
LEDred = 16
LEDyellow = 20
LEDgreen = 21

# Traffic Light 2
LEDred2 = 13
LEDyellow2 = 19
LEDgreen2 = 26

# Config to Refer to GPIO by Pin Number
GPIO.setmode(GPIO.BCM)

# Configure GPIO Pins to Outputs
# Traffic Light 1
GPIO.setup(LEDgreen,GPIO.OUT)
GPIO.setup(LEDred,GPIO.OUT)
GPIO.setup(LEDyellow,GPIO.OUT)

# Traffic Light 2
GPIO.setup(LEDgreen2,GPIO.OUT)
GPIO.setup(LEDred2,GPIO.OUT)
GPIO.setup(LEDyellow2,GPIO.OUT)

##############################################
# Main Program
##############################################
# Run the code until the CTL+C buttons end the program.
# The try...except handles the keyboard interruption
# by pressing the CTL+C buttons. It allows the programmer
# to shut down the program in a more graceful & safer way.

print ('Press CTL+C to Shut Down the Traffic Light Program')
try:
    # Run the Loop Continuously
    while(1):
        ################################
        # 1 Red Light ON
        TrafficLightsCtl(LEDred,True,LEDyellow,False,LEDgreen,False)
        print ('RD1 ON for',LEDredOnTime,'seconds')

        # 2 Green Light ON
        TrafficLightsCtl(LEDred2,False,LEDyellow2,False,LEDgreen2,True)
        print ('GN2 ON for',LEDgreenOnTime,'seconds\n')
        time.sleep(LEDgreenOnTime)

        # 2 Yellow ON
        TrafficLightsCtl(LEDred2,False,LEDyellow2,True,LEDgreen2,False)
        print ('YL2 ON for',LEDyellowOnTime,'seconds\n')
        time.sleep(LEDyellowOnTime)

        ################################
        # 1 Green Light ON
```

```
        TrafficLightsCtl(LEDred,False,LEDyellow,False,LEDgreen,True)
        print ('GN1 ON for',LEDgreenOnTime,'seconds')

        # 2 Red Light ON
        TrafficLightsCtl(LEDred2,True,LEDyellow2,False,LEDgreen2,False)
        print ('RD2 ON for',LEDredOnTime,'seconds\n')
        time.sleep(LEDgreenOnTime)

        ##################################
        # 2 Yellow ON
        TrafficLightsCtl(LEDred,False,LEDyellow,True,LEDgreen,False)
        print ('YL1 ON for',LEDyellowOnTime,'seconds\n')
        time.sleep(LEDyellowOnTime)


# The exception section is called whenever a keyboard interrupt happens.
# This allows you to control how gracefully the program shuts down.
except KeyboardInterrupt:

    # Safety Shut Down Configuration for the Raspberry Pi.
    # The cleanup() method sets all initialized GPIO outputs back to
    # input pins. This will prevent you from accidentally shorting out
    # an output pin to ground during the electronic breadboarding
    # process. All GPIO pins should be configured to inputs when the
    # program is not running.
    GPIO.cleanup()

    # Output the Shut Down Message
    print ('Program has been properly shut down. Goodbye!')
```

## ELECTRONIC LAB CHALLENGE3: TWO TRAFFIC STOP LIGHTS WITH CROSS WALK PBS & LIGHTS

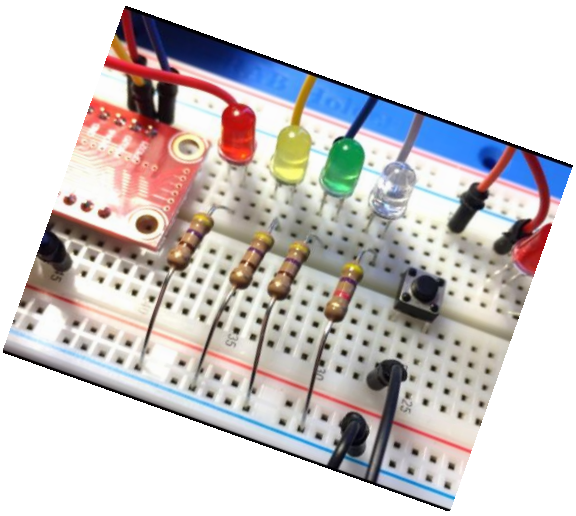



```
################################################
# Filename: ELC 3 - 2TrafficStopLightsCW.py
# Developer: BSides Augusta STEM
# Language: Python 3.4
# Description: This program will make the Raspberry Pi control
# four GPIO output ports which will turn ON/OFF four LEDs: Red,
# Yellow, Green & White. The LEDs will simulate two traffic
# lights with crosswalk sign and pushbuttons.
# Pressing the Cross Walk Pushbutton will add 10 Seconds to the
# Traffic Stop Light time.
# This traffic light control program runs in an infinite loop.
```



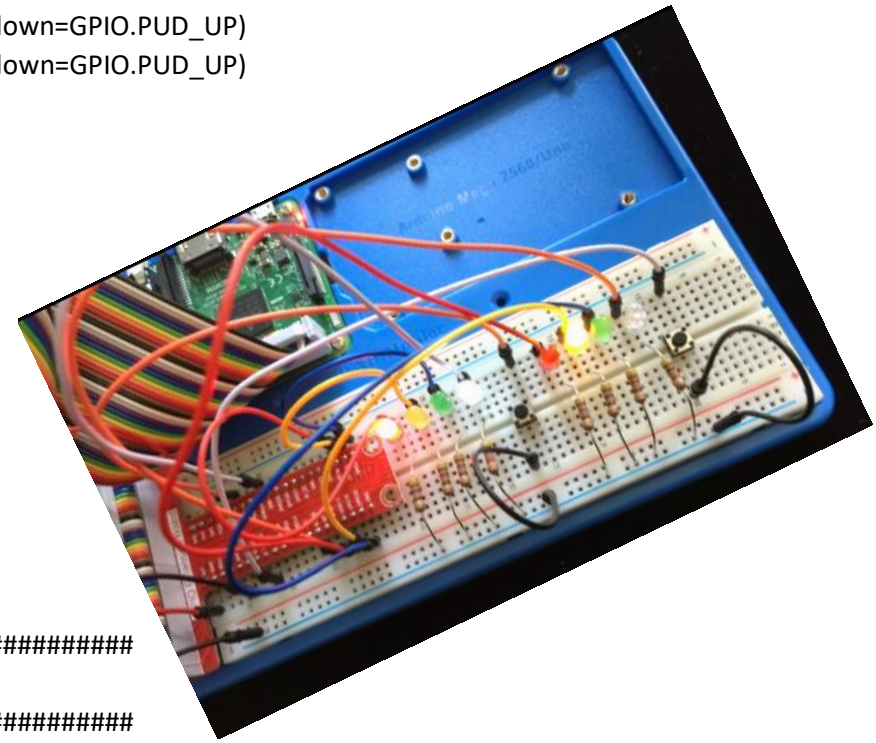

```
################################################
# Import Classes
################################################
# Import Time Class
import time

# Import Raspberry Pi General Purpose Input & Output Class
import RPi.GPIO as GPIO

################################################
# Functions
################################################
# Traffic Lights Control
def TrafficLightsCtl(RedPin,RedTF,YellowPin,YellowTF,GreenPin,GreenTF,WhitePin,WhiteTF):
    # Input Parameters
    # RedPin    => GPIO Pin Number
    # RedTF     => True Turns ON LED; False Turns OFF LED
    # YellowPin => GPIO Pin Number
```

```
    # YellowTF  => True Turns ON LED; False Turns OFF LED
    # GreenPin  => GPIO Pin Number
    # GreenTF   => True Turns ON LED; False Turns OFF LED
    # WhitePin  => GPIO Pin Number
    # WhiteTF   => True Turns ON LED; False Turns OFF LED

    GPIO.output(RedPin,RedTF)
    GPIO.output(YellowPin,YellowTF)
    GPIO.output(GreenPin,GreenTF)
    GPIO.output(WhitePin,WhiteTF)
```



```
################################################
# Initialization
################################################
# Configure Pushbutton Variables
Pushbutton1 = 18
Pushbutton2 = 22

# Configure LED ON Time in Seconds
LEDredOnTime = 5.0
LEDyellowOnTime = 2.0
LEDgreenOnTime = LEDredOnTime - LEDyellowOnTime

# Configure LED Variables to GPIO Pins
# Traffic Light 1
LEDred1 = 16
LEDyellow1 = 20
LEDgreen1 = 21
LEDwhite1 = 12

# Traffic Light 2
LEDred2 = 13
LEDyellow2 = 19
LEDgreen2 = 26
LEDwhite2 = 5

# Congigure Red LED ON Time Delay for Crosswalk
LEDredDelay = 10
LEDgreenDelay = LEDredDelay - LEDyellowOnTime

# Config to Refer to GPIO by Pin Number
GPIO.setmode(GPIO.BCM)
```

```
# Configure GPIO Pins to Inputs/Outputs
GPIO.setup(Pushbutton1,GPIO.IN,pull_up_down=GPIO.PUD_UP)
GPIO.setup(Pushbutton2,GPIO.IN,pull_up_down=GPIO.PUD_UP)

# Configure GPIO Pins to Outputs
# Traffic Light 1
GPIO.setup(LEDred1,GPIO.OUT)
GPIO.setup(LEDyellow1,GPIO.OUT)
GPIO.setup(LEDgreen1,GPIO.OUT)
GPIO.setup(LEDwhite1,GPIO.OUT)

# Traffic Light 2
GPIO.setup(LEDred2,GPIO.OUT)
GPIO.setup(LEDgreen2,GPIO.OUT)
GPIO.setup(LEDyellow2,GPIO.OUT)
GPIO.setup(LEDwhite2,GPIO.OUT)
```



```
###############################################
# Main Program
###############################################
# Run the code until the CTL+C buttons end the program.
# The the try...except handles the keyboard interruption
# with the CTL+C button press, It allows the programmer
# to shut down the program in a more graceful & safer way.

print ('Press CTL+C to Shut Down the Traffic Light Program')
try:
    # Run the Loop Continuously
    while(1):
        ###################################################################
        # Check to See if the Crosswalk 1 Pushbutton Was Pressed
        # Add a Delay to the Next Traffic Light 1 Red Light when Pressed
        input_state = GPIO.input(Pushbutton1)

        # Add Red LED Delay if PB is Pressed
        if input_state==False:
            print('Crosswalk 1 Button Pressed')
            CalcLEDredOnTime = LEDredOnTime + LEDredDelay
            CalcLEDgreenTime = CalcLEDredOnTime - LEDyellowOnTime
        else:
            CalcLEDredOnTime = LEDredOnTime
            CalcLEDgreenTime = LEDredOnTime - LEDyellowOnTime

        ###################################################################
```

```
# Red Traffic Light 1 Control Section
# Traffic Light 2 Yellow Light ON Time is a Constant Amount of Time
# Traffic Light 2 Green Light ON Time is Calculated
#       Green Light 2 = Red Light 1 - Yellow Light Constant

################################
# 1 Red Traffic Light ON
TrafficLightsCtl(LEDred1,True,LEDyellow1,False,LEDgreen1,False,LEDwhite1,True)
print ('RD1 ON for',CalcLEDredOnTime,'seconds')

################################
# 2 Green Traffic Light ON
TrafficLightsCtl(LEDred2,False,LEDyellow2,False,LEDgreen2,True,LEDwhite2,False)
print ('GN2 ON for',CalcLEDgreenTime,'seconds\n')
time.sleep(CalcLEDgreenTime)

################################
# 2 Yellow Traffic Light ON
TrafficLightsCtl(LEDred2,False,LEDyellow2,True,LEDgreen2,False,LEDwhite2,False)
print ('YL2 ON for',LEDyellowOnTime,'seconds\n')
time.sleep(LEDyellowOnTime)



#####################################################################
# Check to See if the Crosswalk 2 Pushbutton Was Pressed
# Add a Delay to the Next Traffic Light 1 Red Light when Pressed
input_state = GPIO.input(Pushbutton2)

# Add Red LED Delay if PB is Pressed
if input_state==False:
      print('Crosswalk 2 Button Pressed')
      CalcLEDredOnTime = LEDredOnTime + LEDredDelay
      CalcLEDgreenTime = CalcLEDredOnTime - LEDyellowOnTime
else:
      CalcLEDredOnTime = LEDredOnTime
      CalcLEDgreenTime = LEDredOnTime - LEDyellowOnTime

#####################################################################
# Red Traffic Light 2 Control Section
# Traffic Light 1 Yellow Light ON Time is a Constant Amount of Time
# Traffic Light 1 Green Light ON Time is Calculated
#       Green Light 1 = Red Light 2 - Yellow Light Constant
```

```
        ##################################
        # 1 Green Light ON
        TrafficLightsCtl(LEDred1,False,LEDyellow1,False,LEDgreen1,True,LEDwhite1,False)
        print ('GN1 ON for',CalcLEDgreenTime,'seconds')

        ##################################
        # 2 Red Light ON
        TrafficLightsCtl(LEDred2,True,LEDyellow2,False,LEDgreen2,False,LEDwhite2,True)
        print ('RD2 ON for',CalcLEDredOnTime,'seconds\n')
        time.sleep(CalcLEDgreenTime)

        ##################################
        # 2 Yellow ON
        TrafficLightsCtl(LEDred1,False,LEDyellow1,True,LEDgreen1,False,LEDwhite1,False)
        print ('YL1 ON for',LEDyellowOnTime,'seconds\n')
        time.sleep(LEDyellowOnTime)


# The exception section is called whenever a keyboard interrupt happens.
# This allows you to control how gracefully the program shuts down.
except KeyboardInterrupt:

    # Safety Shut Down Configuration for the Raspberry Pi.
    # The cleanup() method sets all initialized GPIO outputs back to
    # input pins. This will prevent you from accidentally shorting out
    # an output pin to ground during the electronic breadboarding
    # process. All GPIO pins should be configured to inputs when the
    # program is not running.
    GPIO.cleanup()

    # Output the Shut Down Message
    print ('Program has been properly shut down. Goodbye!')
```

# THANK YOU TO OUR SPONSORS

## PLATINUM SPONSOR

TaxSlayer®

## GOLD SPONSOR

Security Onion Solutions

## SILVER SPONSORS

AND
APPLIED NETWORK DEFENSE

ISSA
Information Systems Security Association
Greater Augusta Chapter

## BRONZE SPONSORS

NWI NUCLEAR WORKFORCE INITIATIVE®
*Growing Our Own Through Collaboration*
SRS COMMUNITY REUSE ORGANIZATION

BOYS & GIRLS CLUBS
OF THE CSRA