# THE SMART FACTORY

# Smart Factory Believers

**EMPOWERING THE
STEM WORKFORCE
OF TOMORROW**

**SMART ROVER
FREQUENTLY
ASKED QUESTIONS**

## GETTING STARTED

**How do I power the Smart Rover?**
To power the Smart Rover body you will need 6 AA batteries which are inserted into the bottom of the Smart Rover body.

**What resources are needed to use the Smart Rover?**
To use and program the Smart Module you will need a monitor, keyboard, and mouse. Each kit also includes a cable to connect your monitor and a wall power adapter to connect your Raspberry Pi in the Smart Module to power.

**Can I connect my Smart Module to a laptop instead of using a monitor, keyboard, and mouse?**
The Raspberry Pi in the Smart Module is a full microcomputer. Although there are methods of connecting a Raspberry Pi to a laptop, they are not advised for using the Smart Rover kit. The Smart Rover should only be used with a monitor, keyboard, and mouse, not a laptop.

**Can I power my Smart Module with another power source, or connect it to the batteries in the Smart Rover body?**
The Raspberry Pi in the Smart Module should only be used with the power cord provided.

# PROJECTS

**My Smart Rover body does not turn on. What should I do?**
Ensure you have placed 6 charged AAA batteries in the bottom of the Smart Rover body.

**My Smart Rover body turns on, but the wheels are not rotating. What should I do?**
Your Smart Rover wheels may have debris stuck to them. Clean the wheels and ensure they are free of any foreign objects or debris.

**How do I properly connect the jumper wires to the Smart Rover body?**
The terminals of motor control and Smart Rover rear are color-coded for with the jumper wires for ease of connection. Match the jumper wires snaps based on their color.

**Why is my buzzer not responsive?**
Ensure that the buzzer is connected properly, with the "+" end oriented as shown in the project manual. If needed, refer to the troubleshooting guide within the project manual to test the buzzer.

**Why is my LED light not responsive?**
Ensure that the LED light is connected properly and aligns with the direction of current flow as shown in the project manual. If needed, refer to the troubleshooting guide within the project manual to test the LED light.

**Where are the images the camera captures stored?**
The camera images are immediately discarded and are not stored anywhere in the Raspberry Pi within the Smart Module.

**Is it safe to connect the Smart Module to a power source overnight?**
Do not keep the Raspberry Pi within the Smart Module connected to power overnight. Only connect it to power when it is in-use.

**Can I work on a different project that is not on the manual?**
We encourage students to be creative and use the skills they've learned from the manual to design their own projects. Remember to never build short circuits, as detailed with in the project manual.

**Why is the Smart Rover not responding to my code?**
Ensure the circuit is properly built and all the components are fully connected by their snaps.

**Does the project manual come in other languages?**
The Smart Rover manual does not come in other languages, it is available only in English.

# REPLACEMENT PARTS
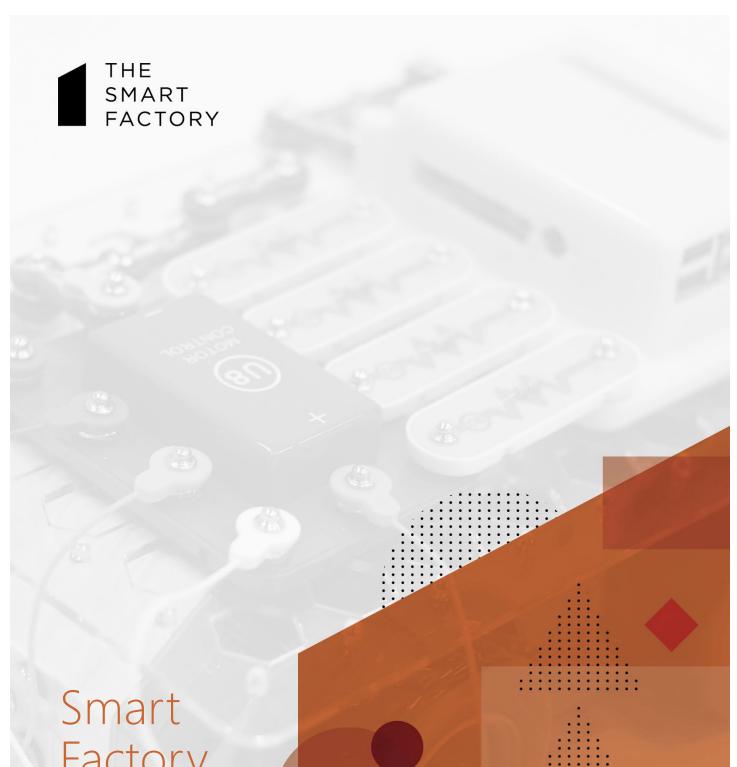
**Where can I get replacement parts?**
If you suspect your parts are broken, please refer to the troubleshooting guide in the project manual to test each component iteratively.

If you believe you have issues with your Snap Circuits® components, please contact Elenco® Electronics Customer Support via elenco.com or by emailing support@elenco.com.

If you believe you have issues with your Smart Module, including the Raspberry Pi or the camera within the component, please contact The Smart Factory @ Wichita via our contact email on our website, thesmartfactory.io.

**I need support with Snap Circuits® components.**
For questions or issues with any Snap Circuits® components, please contact Elenco Electronics® Customer Support.

# THE SMART FACTORY

# Smart Factory Believers

**EMPOWERING THE STEM WORKFORCE OF TOMORROW**

**SMART ROVER PROJECT CODE SOLUTIONS**

# SMART ROVER
# PROJECT CODE SOLUTIONS

**THE SMART FACTORY**

## PURPOSE

These code solutions serve as a troubleshooting guide for any questions faced while completing the Smart Rover projects, including the challenges within the projects.

```python
# Project 1

# Learning to program and use outputs

# Build the the Project 1 circuit and blink a LED

#Challenge 1
# Try changing the LED_On and LED_Off variables to change the blinking pattern

#Challenge 2
# Replace the color LED with buzzer or white LED to try other outputs

#Importing libraries
# Libraries are defined sets of code for specific uses
# Here we want the sleep function for timing and GPIO for the Pi's pin
from time import sleep
import RPi.GPIO as GPIO

#Let's define variables so we can use them later
# Variables are words that take on values within the code
# This way, we can edit the value at the beginning and the changes flow through
LED_Pin =  40 #the internal Pi pin number that goes to snap 7

# For challenge 1, we can try different values here to blink in new patterns
LED_On = 3 #duration of LED flash, seconds
LED_Off = 1 #duration in between flashes, seconds

#Setting up our pin
GPIO.setmode(GPIO.BOARD)
GPIO.setup(LED_Pin, GPIO.OUT, initial=GPIO.LOW)  #Output pin, start off

while True: #Looping over and over again
    sleep(LED_Off) #Keep LED off for defined duration
    GPIO.output(LED_Pin, GPIO.HIGH) #Turn LED on
    sleep(LED_On) #Keep LED on for defined duration
    GPIO.output(LED_Pin, GPIO.LOW) #Turn lED off

print(list(f(10)))
```

```python
# Project 2

# Learning to program and using inputs and outputs

# Build the the Project 2 circuit and control a LED with a button

#Challenge 1
# Try changing the LED_On and LED_Off variables to change the blinking pattern

#Challenge 2
# Replace the color LED with buzzer or white LED to try other outputs

#Challege 3
# Replace the push button with the phototransistor and cover it with your hand - what
 happens?

#Challenge 4
# Try changing the "If" statement from True to False - now what does the button do?

#Importing libraries
# Here we want the sleep function for timing and GPIO for the Pi's pins
from time import sleep
import RPi.GPIO as GPIO

#Let's define variables so we can use them later
Button_Pin =  38 #the internal Pi pin number that goes to snap 6
LED_Pin = 12 #the internal Pi pin number that goes to snap 3

# For challenge 1, we can try different values here to blink in new patterns
LED_On = 3 #duration of LED flash, seconds
LED_Off = 1 #duration in between flashes, seconds

#Setting up our pins
GPIO.setmode(GPIO.BOARD)
GPIO.setup(LED_Pin, GPIO.OUT, initial=GPIO.LOW)  #Output pin, start off
GPIO.setup(Button_Pin, GPIO.IN, pull_up_down=GPIO.PUD_DOWN) #Input pin, start open

while True: #Looping over and over again

    # Here we use the If statement which evaluates a logical expression
    # It is checking if the button is pressed by reading tha value of the pin
    # If the button pin reads True (on), then it executes the indented code

    if GPIO.input(Button_Pin) == False: #When the button is pressed, blink LED
        sleep(LED_Off) #Keep LED off for defined duration
        GPIO.output(LED_Pin, GPIO.HIGH) #Turn LED on
        sleep(LED_On) #Keep LED on for defined duration
        GPIO.output(LED_Pin, GPIO.LOW) #Turn lED off
```

```python
# If the button is not pressed, the code will go to the else statement
else:
    print('Button not pressed')
    sleep(1)
```

```python
# Project 3

# Learning to program, writing functions, and using inputs and outputs

# Build the the Project 3 circuit and control a LED and buzzer with a selector
# Press and hold buttons A, B, and C on the selector

#Challenge 1
# Try changing the Pin_On and Pin_Off variables to change the blinking pattern

#Challenge 2
# Replace the color LED with buzzer or white LED to try other outputs

#Challenge 3
# Try changing input pins A and C in the While loop to switch what A and C do when pr
essed

#Challenge 4
# Try changing output pins LED and Buzzer in the While loop to switch what A and C do
 when pressed

#Challenge 5
# Try switching the order of the LED and Buzzer functions for a cool lightshow when p
ressing B

#Importing libraries
# Here we want the sleep function for timing and GPIO for the Pi's pins
from time import sleep
import RPi.GPIO as GPIO

#Let's define variables so we can use them later
A_Pin =  40 #the internal Pi pin number that goes to snap 7
C_Pin =  38 #the internal Pi pin number that goes to snap 6
LED_Pin = 12 #the internal Pi pin number that goes to snap 3
Buzzer_Pin = 35 #the internal Pi pin number that goes to snap 4

# For challenge 1, we can try different values here to blink in new patterns
Pin_On = 3 #duration of LED flash, seconds
Pin_Off = 0.5 #duration in between flashes, seconds

#Setting up our pins
GPIO.setmode(GPIO.BOARD)

#Our output pins, start off
GPIO.setup(LED_Pin, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(Buzzer_Pin, GPIO.OUT, initial=GPIO.LOW)

#Our input pins from the selector
GPIO.setup(A_Pin, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
```

```python
GPIO.setup(C_Pin, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)

#Let's write some functions we can use to make the coding easier
# For a code snippet we will reuse, we can turn it into a function to call later
# The function name is in blue, and then the arguments it takes are in parentheses

#Here's a function for seeing if a selector button is pressed
# So, read_selector_button reads and returns the value of In_Pin
# This will be helpful for reading the A and C button pins
def read_selector_button(In_Pin):
    return GPIO.input(In_Pin)

#Here's a function for turning an output pin on
#So, output_pin_on takes in the pin number and turns it on after a defined delay
def output_pin_on(Out_Pin, Delay):
    sleep(Delay)
    GPIO.output(Out_Pin, GPIO.HIGH)

#Here's a function for turning an output pin off, can you fill in the missing pieces?
# Replace the ?? with the variables and then uncomment
def output_pin_off(Out_Pin, Delay):
    sleep(Delay) #wait the Delay
    GPIO.output(Out_Pin, GPIO.LOW) #turn the Out_Pin off

while True: #Looping over and over again

    # Here we can use the functions we defined to read buttons and control outputs
    # For the challenges, try changing the button and output pins in the below code

    # If A is pressed and C is not, let's blink the LED
    if read_selector_button(A_Pin) and not(read_selector_button(C_Pin)):
        output_pin_on(LED_Pin, Pin_Off)
        output_pin_off(LED_Pin, Pin_On)

    # If C is pressed and A is not, let's buzz the buzzer
    if read_selector_button(C_Pin) and not(read_selector_button(A_Pin)):
        output_pin_on(Buzzer_Pin, Pin_Off)
        output_pin_off(Buzzer_Pin, Pin_On)

    # If A and C are both pressed, by pressing B, maybe we can flash both LED and buzzer?
    # Replace the ?? with the LED_Pin and Buzzer_Pin variables and then uncomment
    if read_selector_button(A_Pin) and read_selector_button(C_Pin):
        output_pin_on(LED_Pin, Pin_Off)
        output_pin_off(Buzzer_Pin, Pin_On)
        output_pin_on(LED_Pin, Pin_Off)
        output_pin_off(Buzzer_Pin, Pin_On)

    # Wait 1 second to reset
```

```
sleep(1)
```

```python
# Project 4

# Learning to program, writing functions, and using motor control outputs

# Build the the Project 4 circuit and drive the rover along your designed path

#Challenge 1
# Try changing the drive time variable to create a new driving path

#Challenge 2
# Reorder the drive functions to create a new driving path

#Challege 3
# Create your own custom driving path using the different drive functions and time ar
guments

#Importing libraries
# Here we want the sleep function for timing and GPIO for the Pi's pins
from time import sleep
import RPi.GPIO as GPIO

#Let's define variables so we can use them later
Left_Forward_Pin =  36 #the internal Pi pin number that goes to snap 1
Left_Backward_Pin =  11 #the internal Pi pin number that goes to snap 2
Right_Forward_Pin =  12 #the internal Pi pin number that goes to snap 3
Right_Backward_Pin =  35 #the internal Pi pin number that goes to snap 4

#Here we can define the timing variables for the driving functions, in seconds
# For challenge 1, we can try different values here to drive in new patterns
Forward_Time = 2
Backward_Time = 1
Left_Turn_Time = 0.5
Right_Turn_Time = 0.5
Wait_Time = 1

#Setting up our pins
GPIO.setmode(GPIO.BOARD)
#Our output pins, start off
GPIO.setup(Left_Forward_Pin, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(Left_Backward_Pin, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(Right_Forward_Pin, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(Right_Backward_Pin, GPIO.OUT, initial=GPIO.LOW)


#Let's write some driving functions we can use later to program a driving path
def drive_forward(time):
    GPIO.output(Left_Forward_Pin, GPIO.HIGH) #Left motor forward
    GPIO.output(Right_Forward_Pin, GPIO.HIGH) #Right motor forward
    sleep(time)
```

```python
        GPIO.output(Left_Forward_Pin, GPIO.LOW) #Left motor off
        GPIO.output(Right_Forward_Pin, GPIO.LOW) #Right motor off
        print('forward')
        sleep(1)


def drive_left_turn(time):
        GPIO.output(Left_Backward_Pin, GPIO.HIGH) #Left motor backward
        GPIO.output(Right_Forward_Pin, GPIO.HIGH) #Right motor forward
        sleep(time)
        GPIO.output(Left_Backward_Pin, GPIO.LOW) #Left motor off
        GPIO.output(Right_Forward_Pin, GPIO.LOW) #Right motor off
        print('left turn')
        sleep(1)


def drive_right_turn(time):
        GPIO.output(Left_Forward_Pin, GPIO.HIGH) #Left motor forward
        GPIO.output(Right_Backward_Pin, GPIO.HIGH) #Right motor backward
        sleep(time)
        GPIO.output(Left_Forward_Pin, GPIO.LOW) #Left motor off
        GPIO.output(Right_Backward_Pin, GPIO.LOW) #Right motor off
        print('right turn')
        sleep(1)


# Can you finish the function by filling in the blanks for the pins and states?
# This is a backward driving function, so both backward pins should be High then Low
# Uncomment the code when complete
def drive_backward(time):
        GPIO.output(Left_Backward_Pin, GPIO.HIGH) #Left motor backward
        GPIO.output(Right_Backward_Pin, GPIO.HIGH) #Right motor backward
        sleep(time)
        GPIO.output(Left_Backward_Pin, GPIO.LOW) #Left motor off
        GPIO.output(Right_Backward_Pin, GPIO.LOW) #Right motor off
        print('backward')
        sleep(1)


#Here we can use a for loop to control the number of times the code is executed
# Changing the value of range() increases the number of loops performed
for n in range(1):

        # Let's use the driving functions defined above to create a driving path
        # For challenges 2 and 3, try changing the driving functions and order here
        sleep(Wait_Time)
        drive_forward(Forward_Time)
        drive_left_turn(Left_Turn_Time)
        drive_backward(Backward_Time)
        drive_right_turn(Right_Turn_Time)
```

```python
# Project 5

# Learning to program, writing functions, using motor control outputs, adding callbac
k function

# Build the the Project 5 circuit and drive the rover through button pushes, Simon Sa
ys style
# Press and hold the button to drive for that duration

#Challenge 1
# Try changing the drive functions to switch the driving directions

#Challenge 2
# Add new drive functions to activate in the loop to create a new path

#Challege 3
# Use the modulo opertor to change the driving directions based on even or odd number
ed presses

#Challege 4
# With the modulo, add new dirivng functions for even or odd numbered presses

#Importing libraries
# Here we want the sleep function for timing and GPIO for the Pi's pins
from time import sleep
import RPi.GPIO as GPIO
# We also now are using the general time library for the timer function
import time

#Let's define variables so we can use them later
Left_Forward_Pin  =  36 #the internal Pi pin number that goes to snap 1
Left_Backward_Pin =  11 #the internal Pi pin number that goes to snap 2
Right_Forward_Pin =  12 #the internal Pi pin number that goes to snap 3
Right_Backward_Pin =  35 #the internal Pi pin number that goes to snap 4
Button_Pin =  38 #the internal Pi pin number that goes to snap 6

#Setting up our pins
GPIO.setmode(GPIO.BOARD)
#Our output pins, start off
GPIO.setup(Left_Forward_Pin, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(Left_Backward_Pin, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(Right_Forward_Pin, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(Right_Backward_Pin, GPIO.OUT, initial=GPIO.LOW)
#Our input pin from the button
GPIO.setup(Button_Pin, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)


#Let's write some driving functions we can use later to program a pathdef drive_forwa
rd():
```

```python
def drive_forward(time):
    GPIO.output(Left_Forward_Pin, GPIO.HIGH) #Left motor forward
    GPIO.output(Right_Forward_Pin, GPIO.HIGH) #Right motor forward
    sleep(time)
    GPIO.output(Left_Forward_Pin, GPIO.LOW) #Left motor off
    GPIO.output(Right_Forward_Pin, GPIO.LOW) #Right motor off
    print('forward')
    sleep(1)

def drive_left_turn(time):
    GPIO.output(Left_Backward_Pin, GPIO.HIGH) #Left motor backward
    GPIO.output(Right_Forward_Pin, GPIO.HIGH) #Right motor forward
    sleep(time)
    GPIO.output(Left_Backward_Pin, GPIO.LOW) #Left motor off
    GPIO.output(Right_Forward_Pin, GPIO.LOW) #Right motor off
    print('left turn')
    sleep(1)

def drive_right_turn(time):
    GPIO.output(Left_Forward_Pin, GPIO.HIGH) #Left motor forward
    GPIO.output(Right_Backward_Pin, GPIO.HIGH) #Right motor backward
    sleep(time)
    GPIO.output(Left_Forward_Pin, GPIO.LOW) #Left motor off
    GPIO.output(Right_Backward_Pin, GPIO.LOW) #Right motor off
    print('right turn')
    sleep(1)

def drive_backward(time):
    GPIO.output(Left_Backward_Pin, GPIO.HIGH) #Left motor backward
    GPIO.output(Right_Backward_Pin, GPIO.HIGH) #Right motor backward
    sleep(time)
    GPIO.output(Left_Backward_Pin, GPIO.LOW) #Left motor off
    GPIO.output(Right_Backward_Pin, GPIO.LOW) #Right motor off
    print('backward')
    sleep(1)

# Here we are creating a timer function to record the duration of the button press
def button_press_timer():
    Start_Time = time.time() #start the timer
    while GPIO.input(Button_Pin): #while the button is pressed...
        print("Button Pressed")
    return round(time.time() - Start_Time,2) #stop the timer, return elapsed time

# For challenges 3 and 4, we will use a dummy variable to help with modulo operator
count = 0
# Replace the True with the modulo operator statement as %, which means remainder in
division
# So modulo 2 keeps track of odd and even presses since even divided by 2 has remaind
er of 0
```

```python
# To use this as a logical, let's try count % 2 == 0

while True: #Looping over and over again
    sleep(0.25)

    # If the button is pressed, let's use the timer function to see how long
    if GPIO.input(Button_Pin):
        Button_Time = button_press_timer()
        print('Button pressed ' + str(Button_Time) + ' seconds')

        if count % 2 == 0: # Try changing the True to the modulo for challenges 3 and
 4

            #For challenges 1 and 2, try adding new driving functions here
            drive_forward(Button_Time)

        else: # To be used in  challenges 3 and 4
            drive_backward(Button_Time)
            # Add other drive functions here for odd button presses

        count = count + 1 # We increment the counter for the next button press
```

```python
# Project 6

# Learning to program, writing functions, using motor control outputs, adding loop co
mplexity

# Build the the Project 6 circuit and have the rover be controlled by ambient light
# Turn down the ligth and point a flashlight at the rover to direct it

#Challenge 1
# Try changing the drive functions to switch the driving directions

#Challenge 2
# Add new drive functions to change its light seeking spin pattern

#Challege 3
# Add the 100 Ohm resistor in series with the photoresistor to increase light sensiti
vity

#Challege 4
# With the modulo operator, have the rover alternate left or right spins in light sea
rching

#Challenge 5
# After a certain amount of time, have the rover spin to look for light

#Importing libraries
# Here we want the time and sleep for timing and GPIO for the Pi's pins
import time
from time import sleep
import RPi.GPIO as GPIO

#Let's define variables so we can use them later
Left_Forward_Pin =  36 #the internal Pi pin number that goes to snap 1
Left_Backward_Pin =  11 #the internal Pi pin number that goes to snap 2
Right_Forward_Pin =  12 #the internal Pi pin number that goes to snap 3
Right_Backward_Pin =  35 #the internal Pi pin number that goes to snap 4
Photo_Pin =  38 #the internal Pi pin number that goes to snap 6

#Here we can define the timing variables for the driving functions, in seconds
Forward_Time = 2
Backward_Time = 1
Left_Turn_Time = 0.5
Right_Turn_Time = 0.5
Wait_Time = 1

#Setting up our pins
GPIO.setmode(GPIO.BOARD)
#Our output pins, start off
GPIO.setup(Left_Forward_Pin, GPIO.OUT, initial=GPIO.LOW)
```

```python
GPIO.setup(Left_Backward_Pin, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(Right_Forward_Pin, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(Right_Backward_Pin, GPIO.OUT, initial=GPIO.LOW)
#Our input pin from the button
GPIO.setup(Photo_Pin, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)


#Let's write some driving functions we can use later
def drive_forward(time):
    GPIO.output(Left_Forward_Pin, GPIO.HIGH) #Left motor forward
    GPIO.output(Right_Forward_Pin, GPIO.HIGH) #Right motor forward
    sleep(time)
    GPIO.output(Left_Forward_Pin, GPIO.LOW) #Left motor off
    GPIO.output(Right_Forward_Pin, GPIO.LOW) #Right motor off
    print('forward')
    sleep(1)

def drive_left_turn(time):
    GPIO.output(Left_Backward_Pin, GPIO.HIGH) #Left motor backward
    GPIO.output(Right_Forward_Pin, GPIO.HIGH) #Right motor forward
    sleep(time)
    GPIO.output(Left_Backward_Pin, GPIO.LOW) #Left motor off
    GPIO.output(Right_Forward_Pin, GPIO.LOW) #Right motor off
    print('left turn')
    sleep(1)

def drive_right_turn(time):
    GPIO.output(Left_Forward_Pin, GPIO.HIGH) #Left motor forward
    GPIO.output(Right_Backward_Pin, GPIO.HIGH) #Right motor backward
    sleep(time)
    GPIO.output(Left_Forward_Pin, GPIO.LOW) #Left motor off
    GPIO.output(Right_Backward_Pin, GPIO.LOW) #Right motor off
    print('right turn')
    sleep(1)

def drive_backward(time):
    GPIO.output(Left_Backward_Pin, GPIO.HIGH) #Left motor backward
    GPIO.output(Right_Backward_Pin, GPIO.HIGH) #Right motor backward
    sleep(time)
    GPIO.output(Left_Backward_Pin, GPIO.LOW) #Left motor off
    GPIO.output(Right_Backward_Pin, GPIO.LOW) #Right motor off
    print('backward')
    sleep(1)

# For challenge 4, we will use a dummy variable to help with modulo operator
count = 0
# Replace the True with the modulo operator statement as %, which means remainder in
division
```

```python
# So modulo 2 keeps track of odd and even presses since even divided by 2 has remaind
er of 0
# To use this as a logical, let's try count % 2 == 0

# For challenge 5, we will set a maximum light search time for the loop
Max_Search_Time = 4 #seconds
# If the rover has not found light by then, we can get out of the loop with a break s
tatement
# break exits the innermost loop and allows the rover to return to the first sleep co
mmand


while True: # Continuous outer while loop
    sleep(0.25)
    count = count + 1 # Increment the counter for the modulo

    # If the phototransistor detects enough light, drive towards it
    if GPIO.input(Photo_Pin):
        # For challenges 1 and 2, change driving instructions here
        drive_forward(Forward_Time)

    # If there's not enough light, let's look for it by spinning the rover
    else:
        # For challenge 5, we can use the timer function to control the light search
        Start_Time = time.time()
        while not(GPIO.input(Photo_Pin)):
            Elapsed_Time = round(time.time() - Start_Time,2)
            print('Not enough light, searching for more')

            if Elapsed_Time < Max_Search_Time: # Try changing the True to a comparati
ve (<) between
                        # Elapsed_Time and Max_Search_Time for challenge 5

                if count % 2 == 0: # Try changing the True to the modulo for challeng
e 4

                    drive_left_turn(Left_Turn_Time)
                    sleep(Wait_Time)

                else: # For challenge 4, modulo uses these drive commands on odd loop
s

                    drive_right_turn(Right_Turn_Time)
                    sleep(Wait_Time)
            else:
                break # Exits the loop after Max Search Time exceeded
```

```python
# Project 7

# Learning to program, writing functions, using motor control outputs, adding complex
  logic

# Build the the Project 7 circuit and drive the rover with button presses A, B, and C

# Set the controls for the rover for 3 unique commands, and possibly more?

#Challenge 1
# Try changing the drive functions to switch the driving directions for forward/backw
ards and turning

#Challenge 2
# Add new drive functions to change the driving patterns for each button press

#Challenge 3
# Incorporate the button press timer from project 5 to add Simon Says to driving func
tions

#Challege 4
# See how B uses a double If to see if its pressed and then released or held? Can you
  try
# something similar for A and C to create different commands there too?

#Challenge 5
# Replace the length-
3 snap connector with the phototransistor - now all three buttons
# are light dependant. Try controlling the rover to stay in the light.

#Importing libraries
# Here we want the time and sleep for timing and GPIO for the Pi's pins
import time
from time import sleep
import RPi.GPIO as GPIO

#Let's define variables so we can use them later
Left_Forward_Pin =  36 #the internal Pi pin number that goes to snap 1
Left_Backward_Pin =  11 #the internal Pi pin number that goes to snap 2
Right_Forward_Pin =  12 #the internal Pi pin number that goes to snap 3
Right_Backward_Pin =  35 #the internal Pi pin number that goes to snap 4
A_Pin =  40 #the internal Pi pin number that goes to snap 7
C_Pin =  38 #the internal Pi pin number that goes to snap 6

#Here we can define the timing variables for the driving functions, in seconds
Forward_Time = 2
Backward_Time = 1
Left_Turn_Time = 0.5
Right_Turn_Time = 0.5
```

```python
Wait_Time = 0.5

#Setting up our pins
GPIO.setmode(GPIO.BOARD)
#Our output pins, start off
GPIO.setup(Left_Forward_Pin, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(Left_Backward_Pin, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(Right_Forward_Pin, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(Right_Backward_Pin, GPIO.OUT, initial=GPIO.LOW)
#Our input pin from the button
GPIO.setup(A_Pin, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
GPIO.setup(C_Pin, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)


#Let's write some driving functions we can use later to program a pathdef drive_forwa
rd():
def drive_forward(time):
    GPIO.output(Left_Forward_Pin, GPIO.HIGH) #Left motor fwd
    GPIO.output(Right_Forward_Pin, GPIO.HIGH) #R motor fwd
    sleep(time)
    GPIO.output(Left_Forward_Pin, GPIO.LOW) #Left motor fwd
    GPIO.output(Right_Forward_Pin, GPIO.LOW) #R motor fwd
    print('fwd')
    sleep(1)


def drive_backward(time):
    GPIO.output(Left_Backward_Pin, GPIO.HIGH) #Left motor bkwd
    GPIO.output(Right_Backward_Pin, GPIO.HIGH) #R motor bkwd
    sleep(time)
    GPIO.output(Left_Backward_Pin, GPIO.LOW) #Left motor bkwd
    GPIO.output(Right_Backward_Pin, GPIO.LOW) #R motor bkwd
    print('bkwd')
    sleep(1)


def drive_left_turn(time):
    GPIO.output(Left_Backward_Pin, GPIO.HIGH) #Left motor bkwd
    GPIO.output(Right_Forward_Pin, GPIO.HIGH) #R motor fwd
    sleep(time)
    GPIO.output(Left_Backward_Pin, GPIO.LOW) #Left motor bkwd
    GPIO.output(Right_Forward_Pin, GPIO.LOW) #R motor fwd
    print('left turn')
    sleep(1)


def drive_right_turn(time):
    GPIO.output(Left_Forward_Pin, GPIO.HIGH) #Left motor bkwd
    GPIO.output(Right_Backward_Pin, GPIO.HIGH) #R motor fwd
    sleep(time)
    GPIO.output(Left_Forward_Pin, GPIO.LOW) #Left motor bkwd
    GPIO.output(Right_Backward_Pin, GPIO.LOW) #R motor fwd
```

```python
    print('right turn')
    sleep(1)



# Here we are creating a timer function to record the duration of the button press
def button_press_timer():
    Start_Time = time.time() #start the timer
    while GPIO.input(Button_Pin): #while the button is pressed...
        print("Button Pressed")
    return round(time.time() - Start_Time,2) #stop the timer, return elapsed time
# For challenge 3, try uncommenting the Press_Time statements, then use it for the
# the drive commands time arguments


while True: #Looping over and over again
    sleep(0.5)

    # Only pressing A
    if GPIO.input(A_Pin) and not GPIO.input(C_Pin): #only pressing A
        # For challenge 4, you can use a sleep delay and second if, else to see
        # whether A was pressed and released or held
        sleep(0.5)
        #Press B and hold, check if still pressed after delay
        if GPIO.input(A_Pin) and not GPIO.input(A_Pin):
            drive_forward(Forward_Time)
            drive_backward(Backward_Time)
        else:
            Press_Time = button_press_timer(A_Pin) # For challenge 3
            drive_forward(Forward_Time)

    # Only pressing C
    if GPIO.input(C_Pin) and not GPIO.input(A_Pin): #only pressing C
        # For challenge 4, you can use a sleep delay and second if, else to see
        # whether C was pressed and released or held

        Press_Time = button_press_timer(C_Pin) # For challenge 3
        drive_backward(Backward_Time)

    # Pressing B, we can use timing to determine if it's released or held
    if GPIO.input(C_Pin) and GPIO.input(A_Pin):
            sleep(0.5)
            #Press B and hold, check if still pressed after delay
            if GPIO.input(C_Pin) and GPIO.input(A_Pin):
                drive_left_turn(Left_Turn_Time)
            # Press B and released, not still pressed after delay
            else:
                drive_right_turn(Right_Turn_Time)
```

```python
# Project 08

# Trying out the Pi Camera and learning about the different image settings

# Build the the Project 8 circuit and experiment with the camera in cool ways

#Challenge 1
# Try changing the camera resolution to the minimum with 64, 64 and see how it looks

#Challenge 2
# Try changing the camera resolution the maximum with 2592, 1944 and
# the framerate to 15 and see how it looks

#Challege 3
# Try changing the camera rotation to flip it upside down (0) or left or right (90, 2
70)

#Challenge 4
# Try adding a text on top of the image and changing the colors and size

#Challenge 5
# Try looping through all the contrast and brightness options
# and annotate the image with their current levels

#Challenge 6
# Try looping through all the IMAGE_EFFECTS, EXPOSURE_MODES, and AWB_MODES options
# and annotate the image with their current levels


#Importing libraries
# Here we want sleep for timing and picamera for the Pi's camera
from picamera import PiCamera, Color
from time import sleep

# Setting up the camera
camera = PiCamera()

# Change the number of pixels and clarity of the camera
# For challenge 1 and 2, see what low and high resolution look like
camera.resolution = (2592, 1944)

# Change the rate at which the camera records images
camera.framerate = 15

# Rotate the image by x degrees
# Note that the camera assembly is upside down so 180 is right side up
# For challenge 3, try other rotation angles
camera.rotation = 270
```

```python
# For challenge 4, try annotating the image
# Add text on top of the image
camera.annotate_text = 'Hello World!'
# Change the text size on top of the image between 6 and 160
camera.annotate_text_size = 50
# Change the text color in front and back
camera.annotate_foreground = Color('red')
camera.annotate_background = Color('blue')

# Change the contrast between 0 and 100 (color/luminence difference between objects)
camera.contrast = 75

# Change the brightness of the image between 0 and 100
camera.brightness = 75

# Start the preview to view the camera image stream
camera.start_preview()
sleep(5)
camera.stop_preview()

# For challenge 5, try iterating through the brightness levels instead of contrast
camera.start_preview()
for i in range(100):
    camera.brightness = i
    camera.annotate_text = '%s' %i
    sleep(0.1)
camera.stop_preview()


# For challenge 6, try iterating through IMAGE_EFFECTS, EXPOSURE_MODES, and AWB_MODES

camera.start_preview()
for effect in camera.EXPOSURE_MODES:
    camera.annotate_text = '%s' %effect
    camera.exposure_mode = effect
    sleep(1)
camera.stop_preview()

camera.close()
```

```python
# Project 9

# Using the Pi camera to capture and analyze the surroudning light levels

# Build the the Project 9 circuit and flash the LED when certain light thresholds are
 exceeded
# Point a flashlight at the camera to activate the LED

#Challenge 1
# Try changing the Light Threshold value to keep the LED always on

#Challenge 2
# Try changing the Light Threshold value to keep the LED always off

#Challege 3
# Can you add another pin for the buzzer to sound when the ambient light is too low?

#Challege 4
# Can you swap out the Max and Min Light thresholds to activate the LED in darkness?

#Importing libraries
# Here we want sleep for timing, GPIO for the Pi's pins, & picamera for the Pi's came
ra
from time import sleep
import RPi.GPIO as GPIO
from picamera import PiCamera
# We will also need PiRGBArray and cv2 for computer vision/image processing
from picamera.array import PiRGBArray
import cv2
# Numpy is a great numerical tools package to help with the math required
import numpy as np

#Let's define variables so we can use them later
LED_Pin =  35 #the internal Pi pin number that goes to snap 4
Buzzer_Pin =  12 #the internal Pi pin number that goes to snap 3


#Setting up our pins
GPIO.setmode(GPIO.BOARD)
#Our output pins, start off
GPIO.setup(LED_Pin, GPIO.OUT, initial=GPIO.LOW)

#Setting up the camera for light detection
camera = PiCamera()
camera.resolution = (640, 480)
camera.framerate = 30
rawCapture = PiRGBArray(camera, size=(640, 480))

#Setting Min and Max values for HSV image analysis
# Like RGB, HSV is an image color scheme but it's not defined by a color ratio
```

```python
# Instead, it uses Hue (Color), Saturation (Grayness), and Value (Lightness)
# Hue runs 0 to 180 while Saturation and Value are 0 to 255

# For challenge 4, try setting the third value (Lightness) of Light_Min to 0
Light_Min = np.array([0,50,0], np.uint8)
Light_Max = np.array([180,255,255], np.uint8)

# Ambient light percentage threshold for turning the LED
# For challenges 1 and 2, try changing this value to affect the LED
Light_Threshold = 40

#Loop Counter, used to settle the camera with ambient light
i=0

# Using the video camera feature of the camera through an image capture for loop
for frame in camera.capture_continuous(rawCapture, format="bgr", use_video_port=True)
:
    #Capturing image from camera and converting to HSV format
    sleep(3)
    image = frame.array
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

    #Setting the lower bound as the average of the ambient light
    if i < 1:
        Ambient_Light = np.mean(np.mean(hsv[:,:,2]))
        #For challenge 4, try setting Light_Max to the ambient level instead
        Light_Max = np.array([180, 255, Ambient_Light], np.uint8)

    #Filtering out pixels with less lightness than the minimum/ambient average
    # This creates what's called a mask used in demarcating key regions of an image
    Light_Filter = cv2.inRange(hsv,Light_Min, Light_Max)

    #Percentage of pixels above the light threshold
    # This is calculated as the True regions of the mask / number of pixels in image
    Light_Percent = round(sum(sum(Light_Filter ==255))/(640*480),2)

    # If the light percentage threshold is exceeded, blink the LED
    if Light_Percent > Light_Threshold/100:
        print(str(Light_Percent) + ' of image above ambient light levels')
        GPIO.output(LED_Pin, GPIO.HIGH) #LED on
        sleep(2)
        GPIO.output(LED_Pin, GPIO.LOW) #LED off

    # For challenge 3, if there's not enough light, add a buzzer here
    else:
        print('Not enough light detected')
        GPIO.output(Buzzer_Pin, GPIO.HIGH) #Buzzer on
        sleep(2)
        GPIO.output(Buzzer_Pin, GPIO.LOW) #Buzzer off
```

```python
#Clearing image cache to avoid overwhelming the Pi memory
rawCapture.truncate(0)

# Iterate counter
i = i + 1
```

```python
# Project 10

# Using the Pi cameraera to capture and analyze the color profile of objects

# Build the Project 10 circuit and indicate the color of objects with LED and buzzer

#Challenge 1
# Try swapping the LED and buzzer outputs in the code and then also on the rover

#Challenge 2
# Try writing a function to handle the LED and buzzer so it can be called after each
color

#Challege 3
# Try adding a margin that the argmax for Color must exceed to be considered a certai
n color

#Challege 4
# Try adding a memory variable for the last color identified and activate flashes and
 buzzes
# a new LED or buzzer output based on the pattern, like Red then Green

#Importing libraries
# Here we want sleep for timing, GPIO for the Pi's pins, & picamera for the Pi's came
ra
from time import sleep
import RPi.GPIO as GPIO
from picamera import PiCamera
# Numpy is a great numerical tools package to help with the math required
import numpy as np

#Let's define variables so we can use them later
LED_Pin =  35 #the internal Pi pin number that goes to snap 4
Buzzer_Pin = 12 #the internal Pi pin number that goes to snap 3
Button_Pin =  38 #the internal Pi pin number that goes to snap 6


#Setting up our pins
GPIO.setmode(GPIO.BOARD)
#Our output pins, start off
GPIO.setup(LED_Pin, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(Buzzer_Pin, GPIO.OUT, initial=GPIO.LOW)
#Our input pins, start down
GPIO.setup(Button_Pin, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)


# Setting up camera for analysis and to emphasize colors
camera = PiCamera()
camera.resolution = (640, 480)camera.framerate = 30
sleep(2) #let the cameraera settle
```

```python
camera.iso = 100
camera.shutter_speed = camera.exposure_speed
camera.exposure_mode = 'off'
gain_set = camera.awb_gains
camera.awb_mode = 'off'
camera.awb_gains = gain_set

# Prepping for image analysis and eliminating background Noise
#Images are stored in a 3D array with each pixel having Red, Green, and Blue values
Image = np.empty((640,480,3),dtype=np.uint8)
Noise = np.empty((640,480,3),dtype=np.uint8)
RGB_Text = ['Red','Green','Blue'] #Array for naming color

# Let's remove the background 'Noise' colors to emphasis the object's color
camera.capture(Noise,'rgb')
Noise = Noise-np.mean(Noise)

# For challenge 2, let's create a function like we've done before for outputs
# It should have output_pin and delay time arguments to turn them High and then Low
def your_function(output_pin, delay):
    sleep(delay)
    GPIO.output(output_pin, GPIO.HIGH)
    sleep(delay)
    GPIO.output(output_pin, GPIO.LOW)

# For challenge 3, let's set a threshold the max color must exceed
# This will help the camera avoid mistakes in bad lighting or glare
Col_Margin = 0.8
# Let's check if the max * margin > mid
# with max as np.max(RGB_Array) and mid as np.median(RGB_Array)

#Looping with different images to determine object colors upon button press
print('Ready to take photo')
while True:

    # Press the push button to capture an image
    if GPIO.input(Button_Pin) == True:
        sleep(2)
        print('Photo taken')
        camera.capture(Image,'rgb')
        RGB_Array = []

        # For each of red, green, and blue, calculate the most prominent color throug
h means
        for col in range(0,3):
            RGB_Array.append(np.mean(Image[:,:,col]-np.mean(Image)-
np.mean(Noise[:,:,col])))
```

```python
    # For challenge 3, replace the True with the logical statement for the margin

    if np.max(RGB_Array) * Col_Margin > np.median(RGB_Array):
        Color = RGB_Text[np.argmax(RGB_Array)]
        print(Color)
    else:
        print('No prominent color found')

    # For challenge 4, let's look for a pattern like Red then Color
    # We can use an if statement to see if the Last_Color was Red
    # Replace this True with a logical to check, remember it's ==, not = here
    if Last_Color == 'Red':

        # Activate outputs based on the determined object color
        if Color == 'Red': #LED for Red object
            GPIO.output(LED_Pin, GPIO.HIGH) #LED on
            sleep(2)
            GPIO.output(LED_Pin, GPIO.LOW) #LED off

        if Color == 'Green': #Buzzer for Green object
            GPIO.output(Buzzer_Pin, GPIO.HIGH) #Buzzer on
            sleep(2)
            GPIO.output(Buzzer_Pin, GPIO.LOW) #Buzzer off

        if Color == 'Blue': #LED and Buzzer for Blue object
            GPIO.output(LED_Pin, GPIO.HIGH) #LED on
            GPIO.output(Buzzer_Pin, GPIO.HIGH) #Buzzer on
            sleep(2)
            GPIO.output(LED_Pin, GPIO.LOW) #LED off
            GPIO.output(Buzzer_Pin, GPIO.LOW) #Buzzer off

    # For challenge 4, update Last_Color after outputs
    Last_Color = Color
    print('Ready to take photo')
```

```python
# Project 11

# Using the Pi camera to capture and analyze the color profile of objects

# Build the the Project 11 circuit and drive the rover according to colored signs

#Challenge 1
# Try changing the colors associated with the driving commands, like flipping red and
 green

#Challenge 2
# Try adding a modulo operator to alternate between left and right turns on blue sign
s

#Challege 3
# Try setting the drive time variables based on the promince of the color from the ar
gmax

#Challenge 4
# Try adding a memory variable for the last color identified and dictate driving
# based on the pattern, like Red then Green

#Importing libraries
# Here we want sleep for timing, GPIO for the Pi's pins, & picamera for the Pi's came
ra
from time import sleep
import RPi.GPIO as GPIO
from picamera import PiCamera
# Numpy is a great numerical tools package to help with the math required
import numpy as np

#Let's define variables so we can use them later
Left_Forward_Pin =  36 #the internal Pi pin number that goes to snap 1
Left_Backward_Pin =  11 #the internal Pi pin number that goes to snap 2
Right_Forward_Pin =  12 #the internal Pi pin number that goes to snap 3
Right_Backward_Pin =  35 #the internal Pi pin number that goes to snap 4
Button_Pin =  38 #the internal Pi pin number that goes to snap

#Here we can define the timing variables for the driving functions, in seconds
Forward_Time = 2
Backward_Time = 1
Left_Turn_Time = 0.5
Right_Turn_Time = 0.5
Wait_Time = 1

#Setting up our pins
GPIO.setmode(GPIO.BOARD)
#Our output pins, start off
GPIO.setup(Left_Forward_Pin, GPIO.OUT, initial=GPIO.LOW)
```

```python
GPIO.setup(Left_Backward_Pin, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(Right_Forward_Pin, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(Right_Backward_Pin, GPIO.OUT, initial=GPIO.LOW)
#Our input pins, start down
GPIO.setup(Button_Pin, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)

#Let's write some driving functions we can use later
def drive_forward(time):
    GPIO.output(Left_Forward_Pin, GPIO.HIGH) #Left motor forward
    GPIO.output(Right_Forward_Pin, GPIO.HIGH) #Right motor forward
    sleep(time)
    GPIO.output(Left_Forward_Pin, GPIO.LOW) #Left motor off
    GPIO.output(Right_Forward_Pin, GPIO.LOW) #Right motor off
    print('forward')
    sleep(1)


def drive_left_turn(time):
    GPIO.output(Left_Backward_Pin, GPIO.HIGH) #Left motor backward
    GPIO.output(Right_Forward_Pin, GPIO.HIGH) #Right motor forward
    sleep(time)
    GPIO.output(Left_Backward_Pin, GPIO.LOW) #Left motor off
    GPIO.output(Right_Forward_Pin, GPIO.LOW) #Right motor off
    print('left turn')
    sleep(1)


def drive_right_turn(time):
    GPIO.output(Left_Forward_Pin, GPIO.HIGH) #Left motor forward
    GPIO.output(Right_Backward_Pin, GPIO.HIGH) #Right motor backward
    sleep(time)
    GPIO.output(Left_Forward_Pin, GPIO.LOW) #Left motor off
    GPIO.output(Right_Backward_Pin, GPIO.LOW) #Right motor off
    print('right turn')
    sleep(1)


def drive_backward(time):
    GPIO.output(Left_Backward_Pin, GPIO.HIGH) #Left motor backward
    GPIO.output(Right_Backward_Pin, GPIO.HIGH) #Right motor backward
    sleep(time)
    GPIO.output(Left_Backward_Pin, GPIO.LOW) #Left motor off
    GPIO.output(Right_Backward_Pin, GPIO.LOW) #Right motor off
    print('backward')
    sleep(1)


# For challenge 2, we will use a dummy variable to help with modulo operator
count = 0
# Replace the True with the modulo operator statement as %, which means remainder in
division
# So modulo 2 keeps track of odd and even presses since even divided by 2 has remaind
er of 0
```

```python
# To use this as a logical, let's try count % 2 == 0

# For challenge 3, we will set the variable color intensity to scale the drive times
# First, let's define it so we can use the code as is
Color_Intensity = 1

# Setting up camera for analysis and to emphasize colors
camera = PiCamera()
camera.resolution = (640, 480)
camera.framerate = 30
sleep(2) #let the cameraera settle
camera.iso = 100
camera.shutter_speed = camera.exposure_speed
camera.exposure_mode = 'off'
gain_set = camera.awb_gains
camera.awb_mode = 'off'
camera.awb_gains = gain_set

# Prepping for image analysis and eliminating background Noise
#Images are stored in a 3D array with each pixel having Red, Green, and Blue values
Image = np.empty((640,480,3),dtype=np.uint8)
Noise = np.empty((640,480,3),dtype=np.uint8)
RGB_Text = ['Red','Green','Blue'] #Array for naming color

# Let's remove the background 'Noise' colors to emphasis the object's color
camera.capture(Noise,'rgb')
Noise = Noise-np.mean(Noise)

#Looping with different images to determine object colors
print('Ready to take photo')
while True:

    #Press the push button to capture an image
    if GPIO.input(Button_Pin) == True:
        sleep(2)
        print('Photo taken')
        camera.capture(Image,'rgb')
        RGB_Array = []

        # For each of red, green, and blue, calculate the most prominent color throug
h means
        for col in range(0,3):
            RGB_Array.append(np.mean(Image[:,:,col]-np.mean(Image)-
np.mean(Noise[:,:,col])))
        Color = RGB_Text[np.argmax(RGB_Array)]
        print(Color)

        # For challenge 3, let's compare the most prominent color to the second most
        # We can use this ratio to set the Color_Intensity variable
```

```python
        # with max as np.max(RGB_Array) and mid as np.median(RGB_Array)
        # However, the color channels can be negative, so let's use a max to keep positive
        Color_Intensity = np.max([np.max(RGB_Array) / np.median(RGB_Array), 2])

        # For challenge 4, let's look for a pattern like Red then Color
        # We can use an if statement to see if the Last_Color was Red
        # Replace this True with a logical to check, remember it's ==, not = here
        if Last_Color == 'Red':

            # Activate motor controller outputs based on the determined object color
            if Color == 'Red': #Backward for Red object
                drive_backward(Backward_Time * Color_Intensity)
                sleep(Wait_Time)

            if Color == 'Green': #Forward for Green object
                drive_forward(Forward_Time * Color_Intensity)
                sleep(Wait_Time)

            if Color == 'Blue': #Turn for Blue object

                if count % 2 == 0: # Try changing the True to the modulo for challenge 2

                    drive_left_turn(Left_Turn_Time * Color_Intensity)

                else: # For challenge 2, modulo uses these drive commands on odd loops

                    drive_right_turn(Right_Turn_Time * Color_Intensity)

                sleep(Wait_Time)
                count = count + 1  # Increment the counter for the modulo

        # For challenge 4, update Last_Color after outputs
        Last_Color = Color
        print('Ready to take photo')




# Project 12

# Using the Pi camera to capture and analyze the surroudning light levels
```

```python
# Build the the Project 12 circuit and drive the rover to seek out light

#Challenge 1
# Try changing the Left and Right Thresholds to force different turning patterns

#Challenge 2
# Try using the modulo function and loop counter to go from forward to reverse every
few cycles

#Challege 3
# Can you add a timer to the loop to do a spin after a 30 seconds of searching?

#Challege 4
# Can you set the drive time duration based on the ratio of left-to-right light?

#Importing libraries
# Here we want sleep for timing, GPIO for the Pi's pins, & picamera for the Pi's came
ra
from time import sleep
import time
import RPi.GPIO as GPIO
from picamera import PiCamera
# We will also need PiRGBArray and cv2 for computer vision/image processing
from picamera.array import PiRGBArray
import cv2
# Numpy is a great numerical tools package to help with the math required
import numpy as np


#Let's define variables so we can use them later
Left_Forward_Pin =  36 #the internal Pi pin number that goes to snap 1
Left_Backward_Pin =  11 #the internal Pi pin number that goes to snap 2
Right_Forward_Pin =  12 #the internal Pi pin number that goes to snap 3
Right_Backward_Pin =  35 #the internal Pi pin number that goes to snap 4

#Here we can define the timing variables for the driving functions, in seconds
Forward_Time = 2
Backward_Time = 1
Left_Turn_Time = 0.5
Right_Turn_Time = 0.5
Wait_Time = 1

#Setting up our pins
GPIO.setmode(GPIO.BOARD)
#Our output pins, start off
GPIO.setup(Left_Forward_Pin, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(Left_Backward_Pin, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(Right_Forward_Pin, GPIO.OUT, initial=GPIO.LOW)
```

```python
GPIO.setup(Right_Backward_Pin, GPIO.OUT, initial=GPIO.LOW)


#Let's write some driving functions we can use later
def drive_forward(time):
    GPIO.output(Left_Forward_Pin, GPIO.HIGH) #Left motor forward
    GPIO.output(Right_Forward_Pin, GPIO.HIGH) #Right motor forward
    sleep(time)
    GPIO.output(Left_Forward_Pin, GPIO.LOW) #Left motor off
    GPIO.output(Right_Forward_Pin, GPIO.LOW) #Right motor off
    print('forward')
    sleep(1)

def drive_left_turn(time):
    GPIO.output(Left_Backward_Pin, GPIO.HIGH) #Left motor backward
    GPIO.output(Right_Forward_Pin, GPIO.HIGH) #Right motor forward
    sleep(time)
    GPIO.output(Left_Backward_Pin, GPIO.LOW) #Left motor off
    GPIO.output(Right_Forward_Pin, GPIO.LOW) #Right motor off
    print('left turn')
    sleep(1)

def drive_right_turn(time):
    GPIO.output(Left_Forward_Pin, GPIO.HIGH) #Left motor forward
    GPIO.output(Right_Backward_Pin, GPIO.HIGH) #Right motor backward
    sleep(time)
    GPIO.output(Left_Forward_Pin, GPIO.LOW) #Left motor off
    GPIO.output(Right_Backward_Pin, GPIO.LOW) #Right motor off
    print('right turn')
    sleep(1)

def drive_backward(time):
    GPIO.output(Left_Backward_Pin, GPIO.HIGH) #Left motor backward
    GPIO.output(Right_Backward_Pin, GPIO.HIGH) #Right motor backward
    sleep(time)
    GPIO.output(Left_Backward_Pin, GPIO.LOW) #Left motor off
    GPIO.output(Right_Backward_Pin, GPIO.LOW) #Right motor off
    print('backward')
    sleep(1)

#Setting up the camera
camera = PiCamera()
camera.rotation = 180
camera.resolution = (640, 480)
camera.framerate = 30
rawCapture = PiRGBArray(camera, size=(640, 480))

#Setting Min and Max values for Hue, Saturation (Grayness), and Value (Lightness)
Light_Min = np.array([0,50,155], np.uint8)
```

```python
Light_Max = np.array([255,255,255], np.uint8)

# Ambient light percentage of one side to the other, threshold for turning the rover
# For challenge 1, try adjusting these values to force more or fewer turns
Left_Threshold = 51
Right_Threshold = 51

# For challenge 2, we will use a dummy variable to help with modulo operator
count = 0
# Replace the True with the modulo operator statement as %, which means remainder in
division
# So modulo 2 keeps track of odd and even presses since even divided by 2 has remaind
er of 0
# To use this as a logical, let's try count % 2 == 0

# For challenge 2, we can use the timer function to control the light seach
Start_Time = time.time()
Max_Search_Time = 30 #seconds

# For challenge 4, we can initialize a variable for Light Intensity to scale the turn
 durations
Light_Intensity = 1

for frame in camera.capture_continuous(rawCapture, format="bgr", use_video_port=True)
:
    #Capturing image from camera and converting to HSV format
    sleep(3)
    Image = frame.array
    hsv = cv2.cvtColor(Image, cv2.COLOR_BGR2HSV)

    # Analyzing the value (lightness) layer of the image (3rd layer)
    Light = hsv[:,:,2]

    # Calculating the total light in the left and right halves of the image
    Left_Light = sum(sum(Light[:,0:320]))
    Right_Light = sum(sum(Light[:,320:]))

    # Determining the percentage of light of the left and right halves of the image
    Left_Light_Perc = Left_Light  / sum(sum(Light))
    Right_Light_Perc = Right_Light / sum(sum(Light))
    print('L = ' + str(Left_Light_Perc) + ' and R = ' + str(Right_Light_Perc))

    # For challenge 3, determining time passed since forward drive
    Elapsed_Time = round(time.time() - Start_Time,2)

    # For challenge 4, let's find the ratio of the max light to the min light
    # We can set this as the intensity with np.max([Left_Light_Perc, Right_Light_Perc
])
    # and np.min([Left_Light_Perc, Right_Light_Perc]), respectively
```

```python
    Light_Intensity = np.max([Left_Light_Perc, Right_Light_Perc]) / np.min([Left_Ligh
t_Perc, Right_Light_Perc])


    # If the left side is lighter than the threshold, turn left
    if Left_Light_Perc > Left_Threshold/100:
        drive_left_turn(Left_Turn_Time * Light_Intensity)

    # If the right side is lighter than the threshold, turn right
    else:
        if Right_Light_Perc > Right_Threshold/100:
            drive_right_turn(Right_Turn_Time * Light_Intensity)

        # If neither side exceeds the threshold, drive forward (or reverse?)
        else:

            if Elapsed_Time < Max_Search_Time: # Try changing the True to a compariti
ve (<) between
                        # Elapsed_Time and Max_Search_Time for challenge 3

                if count % 2 == 0: # Try changing the True to the modulo for challeng
e 2

                    drive_forward(Forward_Time)
                else: # For challenge 2, modulo uses these drive commands on odd loop
s

                    drive_backward(Backward_Time)

                count = count + 1  # Increment the counter for the modulo

            else: # If max search time exceeded, spin and look elsewhere for challeng
e 3
                drive_left_turn(Left_Turn_Time * 2)
                # Reset the timer for a new searching period
                Start_Time = time.time()
                print('here')

    sleep(Wait_Time)

    #Clearing image cache
    rawCapture.truncate(0)
# Project 13

# Using the Pi camera to capture and analyze the color profile of objects

# Build the the Project 13 circuit and drive the rover according to colored signs

#Challenge 1
# Try changing the colors associated with the driving commands, like flipping red and
 green
```

```python
#Challege 2
# Try adding a modulo operator to alternate between left and right turns on blue sign
s

#Challege 3
# Try setting the drive_time variable based on the promince of the color from the arg
max

#Challenge 4
# Try adding a memory array for the last two colors identified and dictate driving
# based on the pattern

#Importing libraries
from time import sleep
from picamera import PiCamera
from picamera.array import PiRGBArray
import numpy as np
import matplotlib.pyplot as plt
import RPi.GPIO as GPIO
import cv2

#Let's define variables so we can use them later
Left_Forward_Pin =  36 #the internal Pi pin number that goes to snap 1
Left_Backward_Pin =  11 #the internal Pi pin number that goes to snap 2
Right_Forward_Pin =  12 #the internal Pi pin number that goes to snap 3
Right_Backward_Pin =  35 #the internal Pi pin number that goes to snap 4
Button_Pin =  38 #the internal Pi pin number that goes to snap

drive_time = 1 # seconds
turns = 0 #turn counter

#Setting up our pins
GPIO.setmode(GPIO.BOARD)
#Our output pins, start off
GPIO.setup(Left_Forward_Pin, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(Left_Backward_Pin, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(Right_Forward_Pin, GPIO.OUT, initial=GPIO.LOW)
GPIO.setup(Right_Backward_Pin, GPIO.OUT, initial=GPIO.LOW)
#Our input pins, start down
GPIO.setup(Button_Pin, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)


#Let's write some driving functions we can use later to program a pathdef drive_forwa
rd():
def drive_forward(time):
    GPIO.output(Left_Forward_Pin, GPIO.HIGH) #Left motor fwd
    GPIO.output(Right_Forward_Pin, GPIO.HIGH) #R motor fwd
    sleep(time)
```

```python
        GPIO.output(Left_Forward_Pin, GPIO.LOW) #Left motor fwd
        GPIO.output(Right_Forward_Pin, GPIO.LOW) #R motor fwd
        print('fwd')
        sleep(1)


def drive_backward(time):
        GPIO.output(Left_Backward_Pin, GPIO.HIGH) #Left motor bkwd
        GPIO.output(Right_Backward_Pin, GPIO.HIGH) #R motor bkwd
        sleep(time)
        GPIO.output(Left_Backward_Pin, GPIO.LOW) #Left motor bkwd
        GPIO.output(Right_Backward_Pin, GPIO.LOW) #R motor bkwd
        print('bkwd')
        sleep(1)


def drive_left_turn(time):
        GPIO.output(Left_Backward_Pin, GPIO.HIGH) #Left motor bkwd
        GPIO.output(Right_Forward_Pin, GPIO.HIGH) #R motor fwd
        sleep(time)
        GPIO.output(Left_Backward_Pin, GPIO.LOW) #Left motor bkwd
        GPIO.output(Right_Forward_Pin, GPIO.LOW) #R motor fwd
        print('left turn')
        sleep(1)


def drive_right_turn(time):
        GPIO.output(Left_Forward_Pin, GPIO.HIGH) #Left motor bkwd
        GPIO.output(Right_Backward_Pin, GPIO.HIGH) #R motor fwd
        sleep(time)
        GPIO.output(Left_Forward_Pin, GPIO.LOW) #Left motor bkwd
        GPIO.output(Right_Backward_Pin, GPIO.LOW) #R motor fwd
        print('right turn')
        sleep(1)



# Setting up camera for analysis and to emphasize colors
camera = PiCamera()
camera.rotation = 180
camera.resolution = (640, 480)
camera.framerate = 30
sleep(2) #let the cameraera settle
camera.iso = 100
camera.shutter_speed = camera.exposure_speed
camera.exposure_mode = 'off'
gain_set = camera.awb_gains
camera.awb_mode = 'off'
camera.awb_gains = gain_set

# Prepping for image analysis and eliminating background noise
data = np.empty((640,480,3),dtype=np.uint8)
noise = np.empty((640,480,3),dtype=np.uint8)
```

```python
#Images are stored in a 3D array with each pixel having Red, Green, and Blue values
x,y = np.meshgrid(np.arange(np.shape(data)[1]),np.arange(0,np.shape(data)[0]))
rgb_text = ['Blue', 'Green','Red'] #Array for naming colors bgr
camera.capture(noise, 'bgr')
noise = noise-np.mean(noise) # Background 'noise

#Setting Min and Max values for Hue, Saturation (Grayness), and Value (Lightness)
BGR_HSV_Min = np.array([121, 61, 0], np.uint8)
BGR_HSV_Max = np.array([150, 90, 30], np.uint8)


#Looping with different images to determine object colors
print('Ready')
while True:
    #Press the push button to capture an image
     if GPIO.input(Button_Pin) == True:
        sleep(2)
        print('Photo taken')
        camera.capture(data,'bgr')
        hsv = cv2.cvtColor(data, cv2.COLOR_BGR2HSV)

        mean_array,std_array = [],[]
        for ii in range(0,3):
            #Calculating averages of each color to determine most prominent
            mean_array.append(np.mean(data[:,:,ii]-np.mean(data)-
np.mean(noise[:,:,ii])))
        color_picker = np.argmax(mean_array)
        Color = rgb_text[color_picker]
        print(Color)
        #print(np.mean(np.mean(hsv[ :, :, 0])))
        color_view = (hsv[ :, :, 0] >= BGR_HSV_Min[color_picker]) & \
                          (hsv[ :, :, 0] <= BGR_HSV_Max[color_picker])
        print(sum(sum(color_view)))
        target_color = np.mean(np.mean(hsv[:, :, 0][color_view]))
        print(target_color)
        sleep(3)
        camera.close()


        #Setting up the video camera
        vid_camera = PiCamera()
        vid_camera.resolution = (640, 480)
        vid_camera.framerate = 30
        vid_camera.rotation = 180

        rawCapture = PiRGBArray(vid_camera, size=(640, 480))

        # Ambient Color percentage threshold for turning the rover
        Left_Threshold = 65
```

```python
        Right_Threshold = 65
        Total_Threshold = 25 #percentage of target color visible

        #Driving duration
        drive_time = 0.1 #seconds

        #Color offset for hue target color
        color_offset = 7. 5


        for frame in vid_camera.capture_continuous(rawCapture, format="bgr", use_vide
o_port=True):
            #Capturing image from camera and converting to HSV format
            sleep(2)
            image = frame.array
            hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
            hue_view = hsv[:,:,0] #hue
            #color_view = hue_view.copy()
            target_color_view = (hue_view >= target_color - color_offset) & \
                            (hue_view <= target_color + color_offset)
            print(sum(sum(target_color_view))/ (640*480))
            #color_view[~color_mask] = np.nan
            Total_Color_Perc = sum(sum(target_color_view))/ (640*480)
            Left_Color = sum(sum(target_color_view [:,0:320]))
            Right_Color = sum(sum(target_color_view [:,320:]))
            Left_Color_Perc = Left_Color  / sum(sum(target_color_view ))
            Right_Color_Perc = Right_Color / sum(sum(target_color_view ))

            print('L = ' + str(Left_Color_Perc) + ' and R = ' + str(Right_Color_Perc)
)

            #cv2.imshow("result", image)

            if Total_Color_Perc > Total_Threshold/100:
                if Left_Color_Perc > Left_Threshold/100:
                    drive_left_turn(drive_time)
                else:
                    if Right_Color_Perc > Right_Threshold/100:
                        drive_right_turn(drive_time)
                    else:
                        drive_forward(2.5 * drive_time)
            else:
                drive_left_turn(5 * drive_time)
            #Clearing image cache
            rawCapture.truncate(0)
```