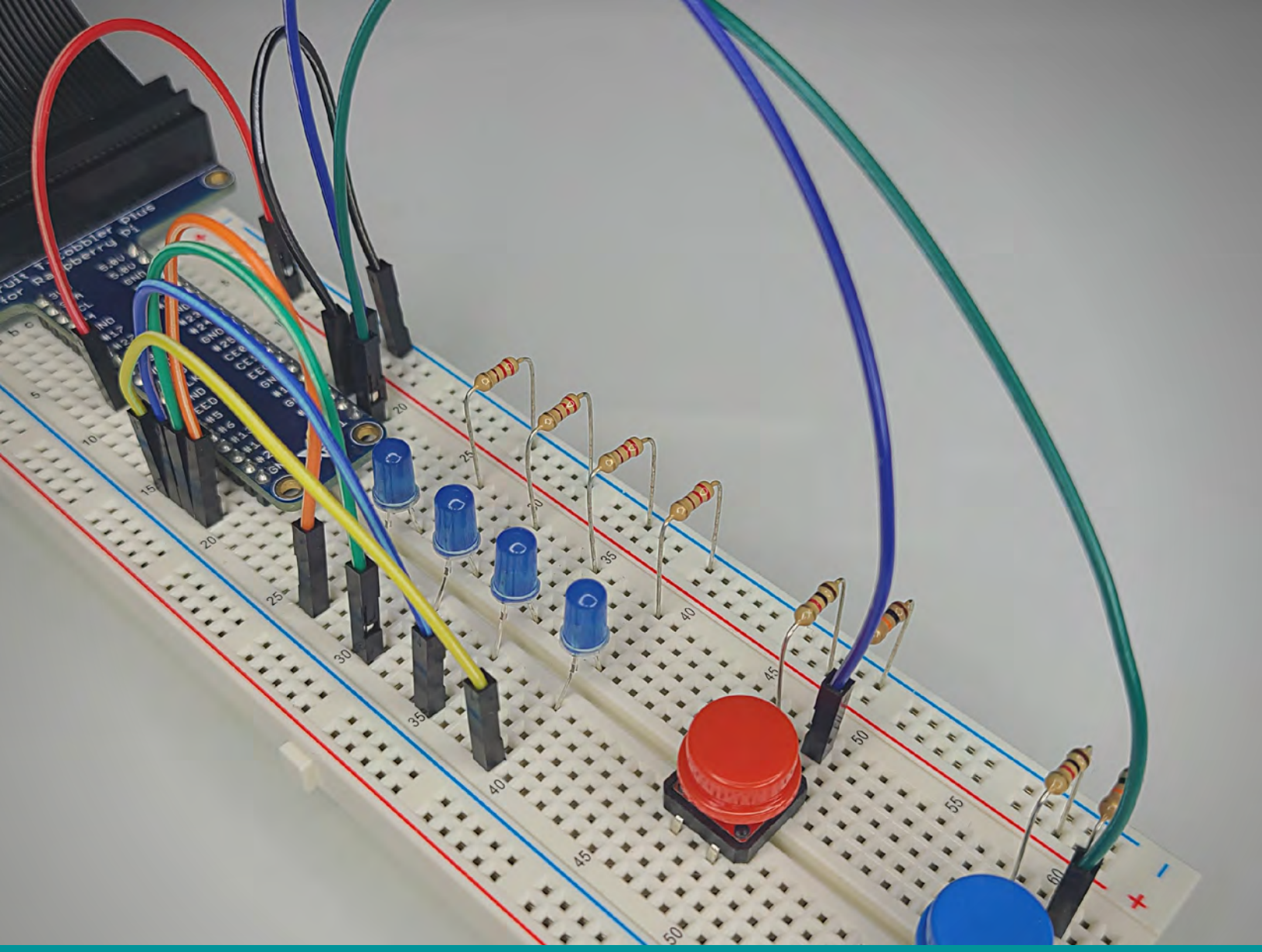


INTRO TO ROBOTICS

LEVEL A

Building Circuits & Beginning Programming

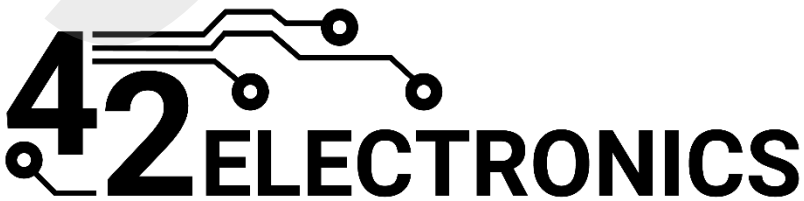
BY ERIC FEICKERT

42 ELECTRONICS

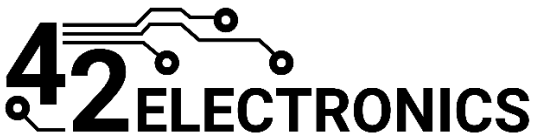
Intro to Robotics Level A

Building Circuits & Beginning Programming

By Eric Feickert



V21057



www.42electronics.com

LEVEL A TABLE OF CONTENTS

LESSON A-3

SERIES VS. PARALLEL CIRCUITS & OHM'S LAW

Lesson Overview

- ► Serial vs. Parallel Circuits
- ► Ohm's Law

Concepts to Review

- ► Breadboard Power, Rows and Columns (Lesson A-1)
- ► Connecting Battery to Breadboard (Lesson A-1)
- ► Building a Circuit with a Resistor and LED (Lesson A-2)
- ► LED Polarity, Forward Voltage (Vf) (Lesson A-2)

Materials Needed

- ❑ Breadboard Circuit from Lesson A-2, Activity #1
- ❑ 220-Ohm Resistor x1
- ❑ LED x1

LESSON A-3

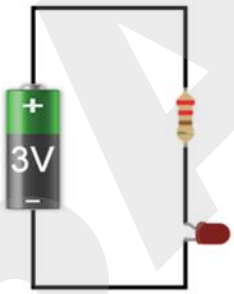
SERIES VS. PARALLEL CIRCUITS & OHM'S LAW

As you saw in the previous lesson, a circuit can be as simple as powering a single LED. However, in the real world, that's not a very useful application! You will generally see multiple components being powered by a single battery. For example, a remote control contains numerous switches, LEDs, and other components, all powered from a single battery. This lesson will dive into series vs. parallel circuits and teach you to power multiple component paths using a single battery.

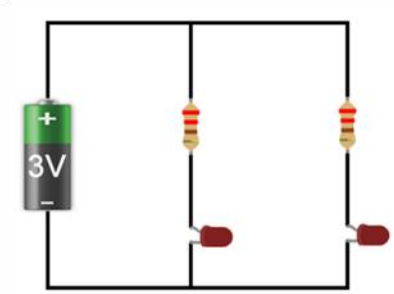
The circuit you built in the last lesson had a single path that ran from the anode (+) of the battery, through a resistor, through an LED, and then back to the cathode (-) of the battery. This is known as a series: one path of components running off the source battery.

But most projects or real-life applications would run in what's known as a parallel circuit. When running in parallel the current runs from the positive terminal of the battery, splits along multiple paths to power components, and then runs back to the negative terminal of the battery.

Circuit with components in series

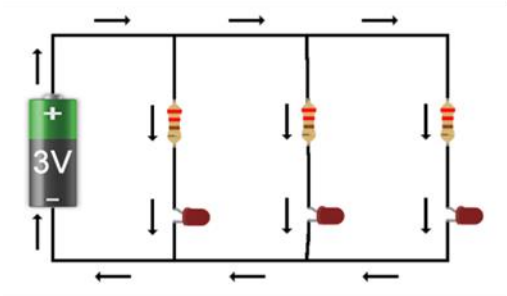


Circuit with two LED/resistor combinations running in parallel sharing the same power source

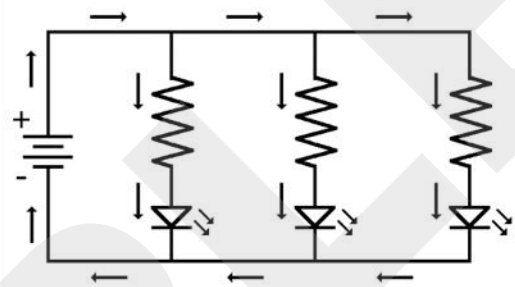


As you saw in Lesson A-2, Activity #2, there are some significant limitations to running serial circuits. Mainly that you would need a higher voltage battery to run anything more than a few simple components. Instead, circuits are normally run in parallel which allows the battery voltage to feed equally to all paths within the circuit. So, looking at Figure 3-1 (serial), with a 3-volt battery, 3-volts will feed resistor 1 and LED 1. In Figure 3-2 (parallel), a 3-volt battery will provide 3-volts of power to resistor 1 and LED 1 and 3-volts to resistor 2 and LED 2. Therefore, running circuits in parallel essentially makes the full battery voltage available across multiple paths.

Current flow in parallel components



Schematic equivalent



CAUTION!

Using parallel circuits to power multiple branches is convenient, however you must be careful to control current flowing through each branch using a resistor. Resistor 1 (R1) will only control current flowing through branch 1 including LED 1 (D1), while Resistor 2 (R2) is used to control current in branch 2. Without Resistor 2, branch 2 would have no current limit and the LED 2 (D2) could burn out due to excess current.

Factors in a Serial vs. Parallel Circuit

When working with circuits, it's helpful to understand the mathematical relationship between the battery voltage, current, and the amount of resistance your circuit will provide. For purposes of this course, you will only need to understand the mathematical equation known as Ohm's Law and how you would theoretically calculate this value. In later levels, you will learn the more complex math behind arriving at an exact value for any variable in the Ohm's Law equation which will allow you to design your own circuits.

First, here is a quick review of the definitions you learned in Lesson A-1:

- **VOLTAGE:** The difference in charge between two points, measured in volts; represented by V such as 1.5V or 3V.
- **CURRENT:** The rate at which charge is flowing, measured in Amperes or more commonly amps (A) or milliamps (mA) but this value will be represented by I in formulas. 1A equals 1000mA.
- **RESISTANCE:** A material's tendency to resist the flow of the current, measured in ohms; can be represented by Ω such as 100-ohms or 100Ω.
- Think of voltage as the potential ability to do work, current is how fast the work is being done, and resistance is anything keeping the work from being done faster.

We recommend you move to the Activities portion of the lesson and complete Activity #1 prior to reading through the remainder of this lesson.

Ohm's Law

The voltage, current, and resistance in a circuit are mathematically related. Ohm's Law can be used to determine the voltage, resistance, or current of any component in a circuit. If you have two out of the three values, you can calculate the third using:

$$V = I \times R \quad \underline{\text{or}} \quad \text{Voltage} = \text{Current} \times \text{Resistance}$$

To use the formula, ensure that you convert all values to the following units:

- V is in volts
- R is in ohms
- I is in amps – Note: This is the most problematic unit as current is often in milliamps and must be converted before using it for Ohm's Law, so 10mA would become .010 amps in the formula

What if you need to solve for current or resistance? The formula can be manipulated to solve for these:

$I = V / R$ <u>or</u> $R = V / I$

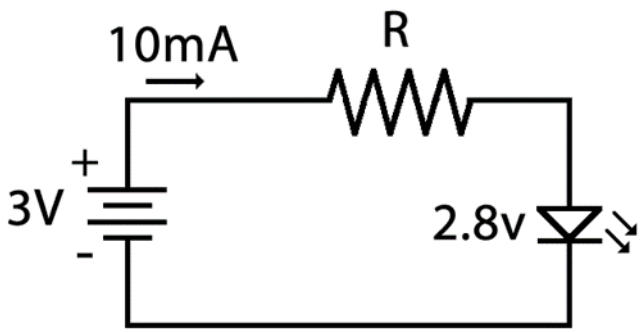
In future levels of this course, this formula will come in handy when selecting a resistor value to help limit current for an LED.

Note:

The following section is optional and skipping it will not keep you from completing the rest of this course. While calculating Ohm's Law is a useful exercise, this concept will be revisited in greater depth in future levels. What's important at this point is understanding that voltage, current, and resistance are mathematically related and calculating these values will be important once you move on to designing your own circuits.

Ohm's Law Calculation Example

Let's say you have an LED with a Vf of 2.8V, with a current limit of 10mA, and a supply voltage of 3-volts. The formula you would use is resistance equals the supply voltage minus the LED's forward voltage divided by the desired current in amps.



$$R = \frac{(Vsource - Vforward)}{Current}$$

$$R = \frac{(3 - 2.8)}{0.01}$$

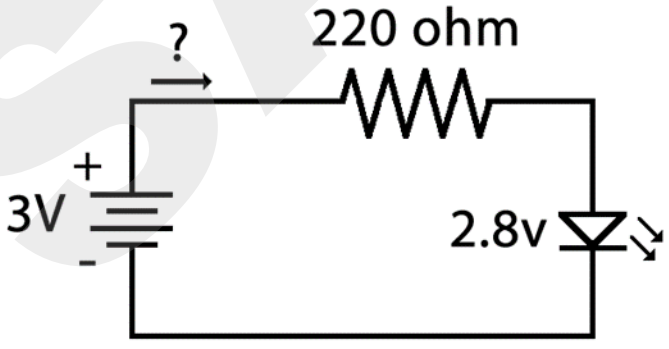
$$R = \frac{.2}{0.01}$$

$$R = 20 \text{ ohms}$$

Start by subtracting the LED Vf of 2.8 from the supply voltage of 3, and dividing that by 10mA or .010. This leaves you with .2 divided by .01 which equals 20 or 20-ohms. This means that with a supply voltage of 3-volts and an LED with a Vf of 2.8V, you would need a 20-ohm resistor to limit the current through the LED to 10mA.

10mA is well below the limit of 20mA, however LEDs need very little current to turn on so it's best to go as low as possible. Later in this course, you will power the breadboard using the Raspberry Pi. Since the Raspberry Pi is a small computer, it can only output a limited amount of current, so keeping the LED current to a minimum is a good idea.

Use Ohm's Law to determine how much current the LED is allowed when we use a 220 -ohm resistor and a 3-volt supply.



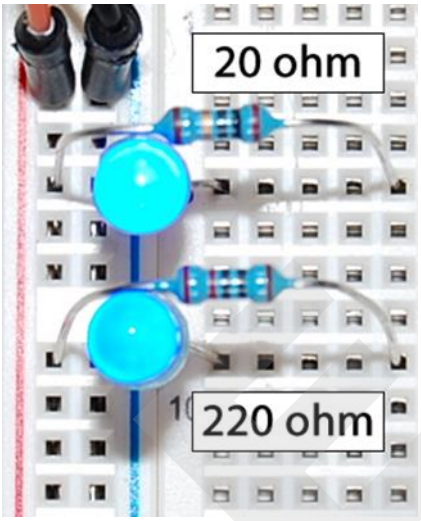
$$I = \frac{(Vsource - Vforward)}{Resistor}$$

$$I = \frac{(3 - 2.8)}{220}$$

$$I = \frac{.2}{220}$$

$$I = .0009$$

Using this formula, a 220-ohm limits the LED current to .0009 amps or .9mA. Comparatively, a 20-ohm resistor would limit the LED current to .010 amps or 10mA.



This means that if your voltage source was only able to supply 10mA of current, you could power a single LED with a 20-ohm resistor or you could power 11 LEDs using 220-ohm resistors.

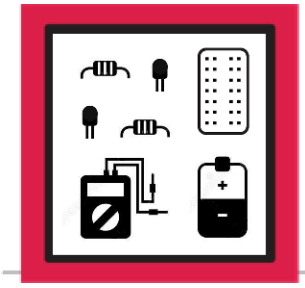
There is not much LED brightness difference between the two, so limiting the current as much as possible is the better option, especially when powering LEDs using the Raspberry Pi.

As you can see from this photo, the 20-ohm resistor powering the LED at 10mA is only marginally brighter than the 220-ohm resistor powering the LED at .9mA. Unless your application requires an extremely bright LED that needs a lot of current, you can normally get away with powering LEDs using very little current.

Don't worry if you don't quite understand the math. You will not need this equation for the remainder of this course and you will revisit this concept in future levels. The goal is for you to understand that voltage, current, and resistance are mathematically related, and that the equation can be used when designing circuits.

Note:

If you wish to dive deeper into Ohm's Law and the math involved, visit this webpage: www.42electronics.com/level-a-resources

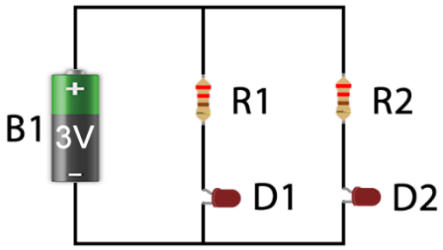


LESSON A-3

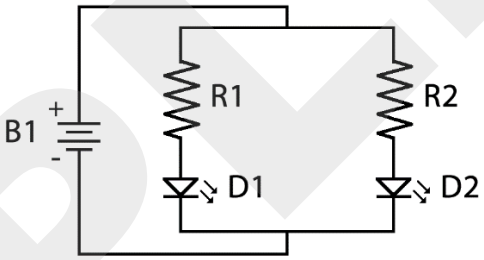
ACTIVITY #1:
BUILD A PARALLEL CIRCUIT

As in the previous lesson, below you will find the schematic for this project, for your review in future lessons. In this activity, you will add a second LED and resistor combination in parallel with the first.

Physical circuit



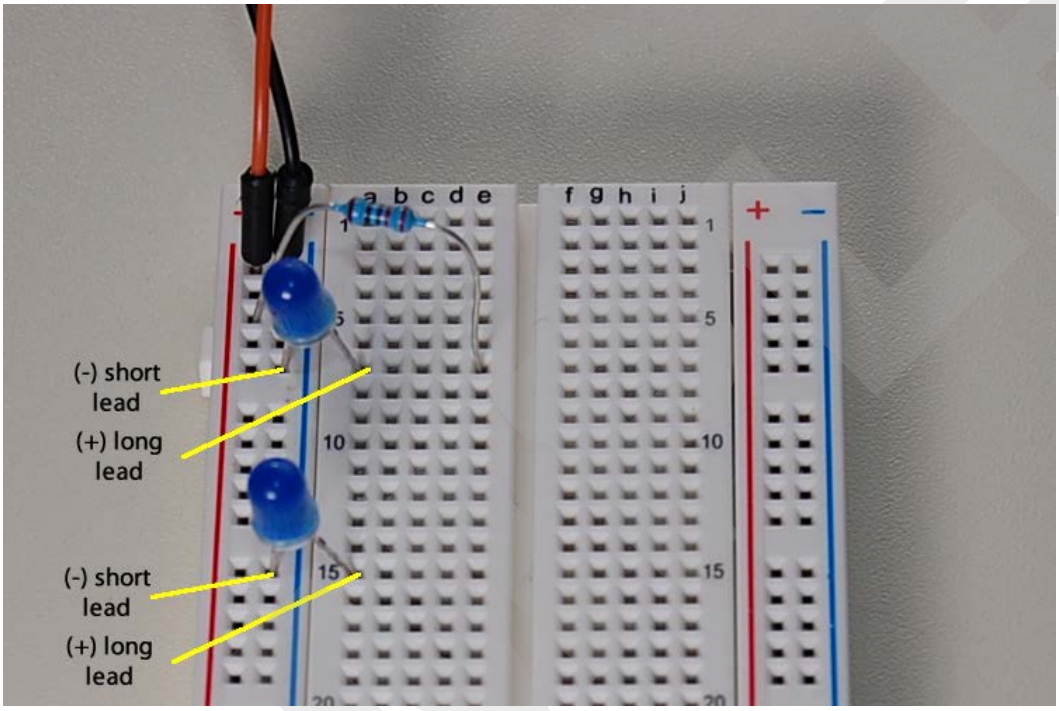
Schematic Diagram



Step #1

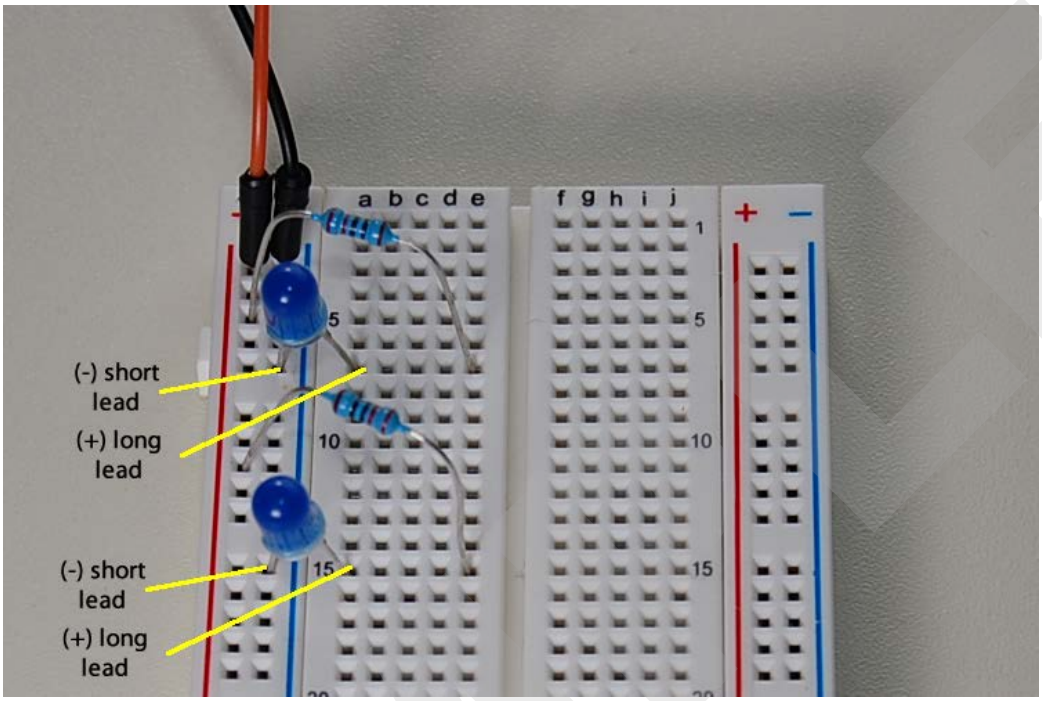
If it's not still built on your breadboard, construct the circuit from Lesson A-2, Activity #1 and then you may proceed with the activity for this lesson.

Using the circuit from in Lesson A-2, Activity #1 as a starting point, you will first add a second LED. Insert the anode of the LED into A15 and the cathode of a second LED into N1-15.



Step #2

Because the second LED you just added is functioning in parallel with the first, you will need to add a second resistor to control the flow of current. Connect a second 220-ohm resistor between P1-11 and E15.



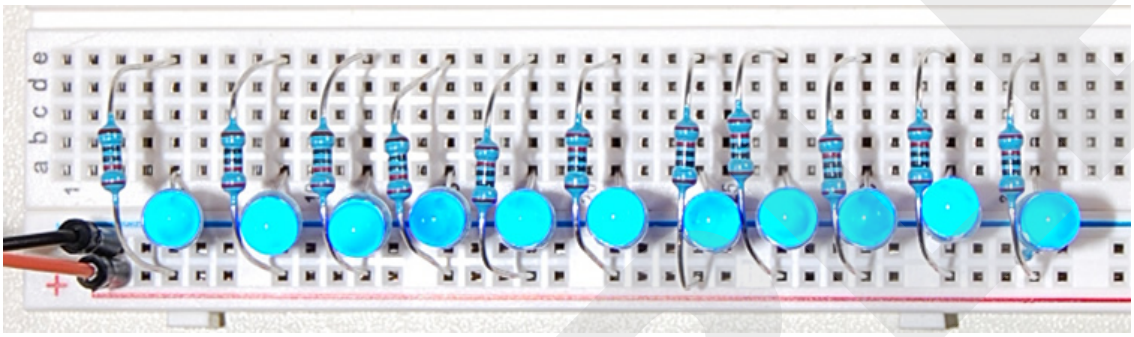
Step #3

Take a moment to track with your eye how the current flows in this circuit and ensure you understand the path from battery positive, through the components, and back to battery negative:

- Positive of the battery to P1-3 powers the positive power rail
- Current splits into two separate paths:
    - P1-5 through the first resistor to E7, across row 7 to A7, into the first LED, and out to N1-7
    - P1-11 through the second resistor to E15, across row 15 to A15, into the second LED, and out to N1-15
- Paths merge at the negative power rail and flow through N1-3 to negative of the battery

Step #4

If desired, you may continue to add the other resistors and LEDs in this kit to the circuit, in parallel, following the same pattern. The number of resistors and LEDs included in this kit are safe additions to this circuit. However, if you are working with additional components beyond what is included in this kit, keep in mind that current is not an infinite resource and you will eventually exceed the maximum current output rating of your power source. This is particularly true when you work with the Raspberry Pi in later lessons.



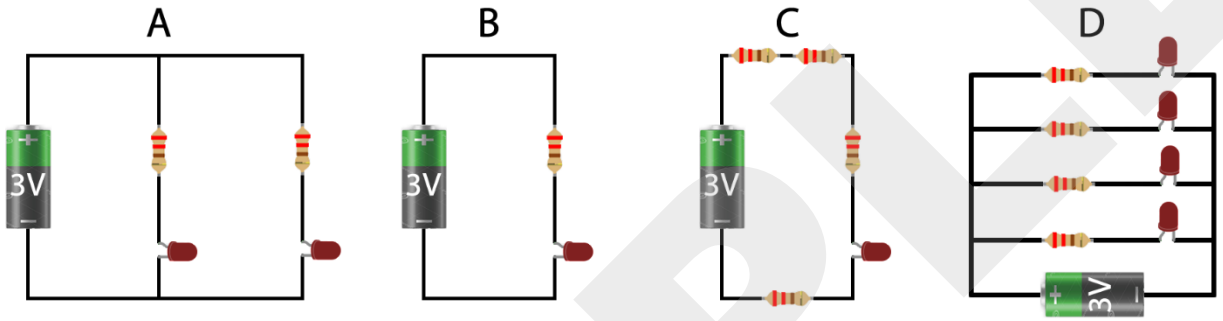
Step #5

Disassemble the circuit and return the components to the kit in preparation for the next lesson.

LESSON A-3

QUESTIONS FOR UNDERSTANDING

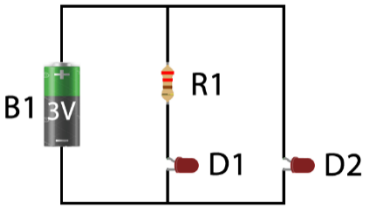
1. Here are some images of series and parallel circuits. Label each circuit as series or parallel.



2. Why would you use parallel circuits instead of keeping all components in series?

3. What is the formula for Ohms Law?

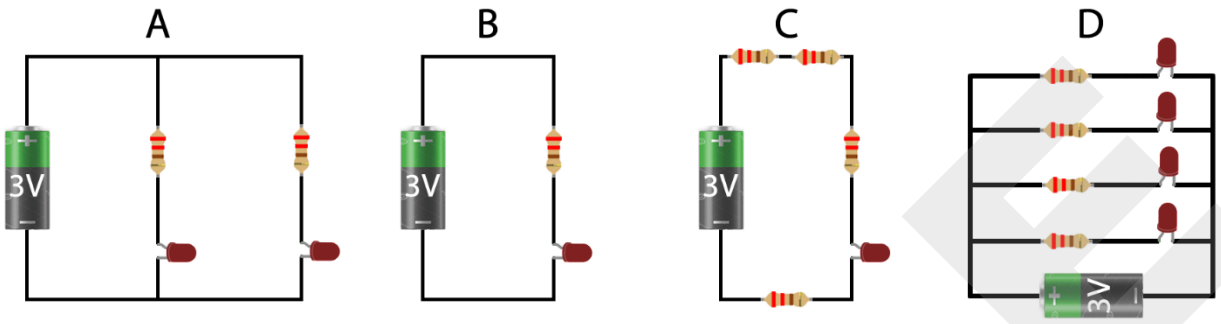
4. In this circuit, Resistor 1 is connected between the battery positive and LED 1. Will Resistor 1 in this configuration limit the current flowing through LED 2?



Answers Can be Found on the Next Page

Answers

1. **Here are some images of series and parallel circuits. Label each circuit as series or parallel.**



ANSWER: A: Parallel, B: Series, C: Series, D: Parallel

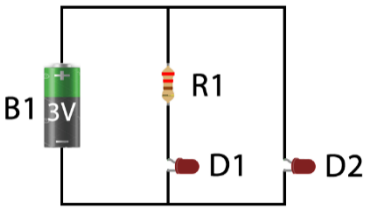
2. **Why would you use parallel circuits instead of keeping all components in series?**

ANSWER: A parallel circuit allows the voltage from the battery to be used equally across all paths in the circuit. So, a 3V battery will provide 3V to path 1, 3V to path 2, etc. This allows you to run multiple components on a single low-powered battery. Running multiple components on a series circuit will generally require a higher voltage battery to account for the Vf (forward voltage) required by certain components.

3. **What is the formula for Ohms Law?**

ANSWER: V = I x R or Voltage = Current x Resistance

4. **In this circuit, Resistor 1 is connected between the battery positive and LED 1. Will Resistor 1 in this configuration limit the current flowing through LED 2?**



ANSWER: No, Resistor 1 will only limit the current flowing along the path between battery positive, LED 1, and battery negative. The path containing LED 2 will require its own resistor (Resistor 2) to limit the current flowing through LED 2.

LESSON A-15

NESTED IF STATEMENTS &
STRING/INTEGER CONVERSION

Lesson Overview

- ► Nested If Statements
- ► Indentation in Formatting
- ► Strings vs. Integers
- ► Converting a Value to an Integer

Concepts to Review

- ► Program Flow, Strings, Variables, Print Commands, Order (Lesson A-11)
- ► Code Organization, User Input (Lesson A-12)
- ► Math Functions (Lesson A-13)
- ► Boolean Logic, If/Else Statements (Lesson A-14)

Materials Needed

❑ Assembled Raspberry Pi setup from Lesson A-9 including Raspberry Pi, monitor, keyboard, and mouse

LESSON A-15

NESTED IF STATEMENTS &
STRING/INTEGER CONVERSION

In this lesson you'll continue to learn how to work with if statements by adding increasingly complex criteria. You will then move on to learn how to specify strings versus integers in Python.

Nested If Statements

In Lesson A-14, you learned to structure simple if statements in Python. But what happens if you need to have multiple criteria? For example, if you only wanted to print a receipt if the sale value was $1 and the item was red? For this, you would need to use nested if statements.

A nested if statement, describes an if statement that's inside of, and dependent on, another if statement. This can be used to add extra layers of decisions that you might want to make happen in your program.

Formatting: Importance of Indentation

When formatting nested if statements, indentation is very important. This tells Python when statements are dependent upon each other.

For example, this program simulates a code that might open a lock. The lock can only open with the code 345.

```
if x == 3:
    print('First digit accepted')
    if y == 4:
        print('Second digit accepted')
        if z == 5:
            print('Third digit Accepted')
            print('Lock open')
```

What will happen when the digits 3, 4, 5 are entered?

The code digits are initially programmed in using the variables x, y, and z. The first if condition x == 3 will be evaluated, and if True, **First digit accepted** will be printed, and the next condition will be evaluated. If y == 4 evaluates as True then **Second digit accepted** will be printed, and the third condition z == 5 will be evaluated. If this evaluates as True, then **Third Digit accepted** and **Lock open** will both be printed.

The only way **Lock Open** can ever be printed is if all three conditions preceding it evaluate as True. If any of the digits do not equal the expected digit, then the rest of the code will be skipped, and the lock will not open.

But what happens if the statements aren't properly indented? Python no longer sees them as dependent upon one another and you can't obtain the conditional buildup seen in the last example.

For example, consider the following code:

```
if x == 3:
    print('First digit accepted')
if y == 4:
    print('Second digit accepted')
if z == 5:
    print('Third digit Accepted')
    print('Lock open')
```

What happens if the user enters 555?

Since the first digit is wrong, the first condition will evaluate as False, so no message would be printed. Due to the improper indentation, the second condition will also be evaluated, even though the first digit was incorrect. Without indentation, Python doesn't realize it's not supposed to proceed to the next line of code! In this case, the second digit is also incorrect, so no message will be printed.

The problem really shows up when the third condition is tested. This digit is correct, so the condition will evaluate as True, and both **Third digit accepted** and **Lock open** will be printed. The lock was allowed to open, even though two of the three digits were incorrect. Without indentation, Python treated all three conditions as independent of one another.

If your intent is to have multiple if statements trigger an action, then make sure your indentation is allowing the conditions to be evaluated as expected. Also, test to make sure your program is operating as designed, and that incorrect entries don't generate unexpected behavior.

ALWAYS USE FOUR SPACES TO INDENT CODE. It may be tempting to use the tab key out of convenience, but tabs can be interpreted differently by different programs. Spaces are treated the same way in every program, so they are the safest way to learn to indent your code. If you need more levels of indentation just add four more spaces for each level:

```
first_line              # this line has no spaces
    second_line         # this line has 4 spaces
        third_line      # this line has 8 spaces
    fourth_line         # this line has 4 spaces
        fifth_line      # this line has 8 spaces
```

Strings vs. Integers

As you learned in the previous lesson, strings are one or more characters made up of letters, numbers, or symbols. You can ask Python if string '6' equals string '6' and it will evaluate as True. This happens because those two characters are the same, much like Python would say that the strings 'a' and 'a' are equal. You can also add string 'a' to string 'a' and you will get an output string of 'aa'. This worked great in previous lessons to add “My name is “ and “name” together to get a full sentence.

The problem occurs when you try to do mathematical calculations with strings of numbers. Consider the following program:

```
a = '4'

b = '2'

c = a + b

print(c)
```

What number do you think will be printed to the console as c? Here's a hint: it's not 6.

42 will be printed because the character '4' added to the character '2' is '42'. Python doesn't understand that these characters represent numeric values. For that you will need to use integers.

Converting a Value to an Integer

Integers are only allowed to be numbers, so trying to store the value 'apple' as an integer will result in an error. Here are some examples:

Statement	Action	Use for Numeric Functions?
x = '6'	This will store the value 6 as a string in a variable called x	No
x = 6	This will store the value 6 as an integer in a variable called x	Yes
x = input('Type a number: ')	Typing 6 at this prompt will cause the program to store the value of 6 as a string in a variable called x	No

The last one is a bit of a problem. How do you use an input of 6 for a mathematical equation when Python is storing it as a string? Fortunately, Python has a great way to convert back and forth between strings and integers. The command for converting a numeric string to an integer is:

```
int()
```

Just insert a variable value between the parentheses, and Python will convert the value to an integer, so you can use that value for math purposes. Remember, this will only work on numeric values. Attempting to convert the string value 'apple' into an integer will result in a Python error.

Here is an example of converting a string to an integer:

`x = '42'`	This will initially store the value of 42 as a string in variable x
`y = int(x)`	This will take the current value of x, convert it to an integer, and store that new value in the variable y

This integer command will be very useful in Activity #3, where you will convert the numeric user input from a string into an integer, so you can compare it to other numbers later in the program.

Converting a Value to a String

But what if you have the opposite problem? You have an integer you need to convert to a string so you can combine it with another string? Python cannot merge integers and strings to be displayed together so you must convert any integers to strings.

The command to convert a value to a string is:

`str()`

It can be used just like the **`int()`** command above. Here is an example of converting an integer to a string:

`x = 42`	This will initially store the value of 42 as an integer in variable x
`y = str(x)`	This will take the current value of x, convert it to a string, and store that new value in the variable y

This can be helpful when trying to print integers and strings together. As noted above, Python cannot concatenate (merge) strings and integers together in the same print command.

Bad Code	Good Code

`x = 42`
`print('Your number is ' + x)`
 	
This print statement is trying to combine the string 'Your number is ' and the integer value of x. This will result in an error.
	`x = 42`
	`y = str(x)`
	
	`print('Your number is ' + y)`
	
	Since x is being converted into a string value called y, this print statement can combine 'Your number is ' and the string value of y, with no errors. 'Your number is 42' will be printed to the shell.

This conversion can also happen within a single line. In Lesson A-13 you used this statement to bring in user input as an integer:

```
var = int(input('Enter a number: '))
```

Normally the input command will bring in any value entered as a string, even numeric values. The problem with this is that you can't perform math operations on a string, even when the string is something like '6'. That '6' might as well be 'apple' or 'tree'. Python has no idea how to subtract 2 from 'apple' or from '6'.

The input string must be converted to an integer using the **int()** command. There are a couple of ways to accomplish the same thing:

Option #1:

```
var = input('Enter a number: ')

var = int(var)
```

The input from the user is initially saved to the variable **var**. Then the int(var) will take the integer value **var** and overwrite the string value of **var** with the integer value.

Or you can do this by wrapping the **int()** around the entire input command as seen below.

Option #2:

```
var = int(input('Enter a number: '))
```

In this example, from Lesson 13, the user is asked for input, that input is converted to an integer, and that integer value is saved to the variable called **var**. This way is slightly more complicated looking, but it results in one less line of code.

LESSON A-15

ACTIVITY #1:
ADD FIVE YEARS TO YOUR AGE

In this activity you will write a program that will ask the user for their current age, add five years, and print a message letting them know what their age will be in five years.

Step #1

Ask the user how old they are and store it in a variable called **age**:

```
age = input('How old are you: ')
```

Step #2

The users age is now stored as a string in a variable called age, however Python can't add five to a string. The string value must be converted to an integer, so Python can add to the value. Store that new integer value as a variable called **age_int**:

```
age_int = int(age)
```

Step #3

Now Python can add five years to the current value of **age_int**:

```
age_int = age_int + 5
```

This will take the current value of **age_int**, add 5 to it, and save it as the updated value of **age_int**.

Step #4

You want to have a nice message that combines the string **Your age in five years will be** and the **age_int** value. Since the **age_int** value is currently an integer it cannot be combined with the other string. You must convert the **age_int** value from an integer back to a string. Save this new string as **age_str**:

```
age_str = str(age_int)
```

Step #5

You now have two strings that can be combined into a single **print** command. Combine the string **Your age in five years will be** and the **age_str** value into a **print** command:

```
print('Your age in five years will be ' + age_str)
```

Step #6

Verify the fully assembled program is free of errors:

```
age = input('How old are you: ')

age_int = int(age)


age_int = age_int + 5


age_str = str(age_int)

print('Your age in five years will be ' + age_str)
```

Step #7

Run the program a few times using different age values and make sure that you get the expected output each time. Save the program if desired (optional).

LESSON A-15

ACTIVITY #2:
AGE CALCULATOR

In this activity you will write a program that will ask a user for the year they were born and if their birthday has occurred yet this year. The program will then calculate the user's age at the end of this year and will subtract a year if their birthday has not yet occurred. The program will then display their current age in a single print statement.

Step #1

Ask the user for the year they were born in and store it as a variable called **year**:

```
year = input('What year were you born: ')
```

Step #2

Ask the user if their birthday has happened yet this year and store it as a variable called **bday**:

```
bday = input('Has your birthday happened this year? (y/n): ')
```

Step #3

Now, convert the year from a string to an integer using the **int()** command and store it as a variable called **year_int**:

```
year_int = int(year)
```

Step #4

The **year_int** is now storing the integer value of the year the user was born. You can use this value for math purposes. Subtract that year from the current year and store the result as a variable called **age**:

```
age = 2018 – year_int
```

Step #5

The **age** value is currently the user's age at the end of this year. If their birthday has not yet occurred, then they will be a year younger than the age value. Use an if condition to see if the value of **bday** is equal to **'n'**, and if so, subtract 1 from the value of **age**. Remember to end the if statement with a colon and to indent the code to be executed by the if statement:

```
if bday == 'n':

    age = age - 1
```

If the user answered **'n'** then 1 year will be subtracted from the value of **age**. You don't need an else statement because if the user answered **'y'** then **age** does not need to be modified.

Step #6

The **age** value is currently an integer, so it cannot be easily printed with a string. Convert **age** to its string equivalent using the **str()** command, and use a variable called **age_str** to store the string value:

```
age_str = str(age)
```

Step #7

Build a **print** statement that will bring everything together:

```
print('Your current age is ' + age_str)
```

Step #8

Verify the fully assembled program is free of errors:

```
year = input('What year were you born: ')

bday = input('Has your birthday happened this year (y/n): ')


year_int = int(year)

age = 2018 - year_int


if bday == 'n':

    age = age - 1


age_str = str(age)


print('Your current age is ' + age_str)
```

Step #9

Run your program a few times to make sure the age it's outputting is correct, and that changing whether or not your birthday happened this year gives you the correct age. Save the program if desired (optional).

LESSON A-15

ACTIVITY #3:
GUESS A NUMBER

You will now write a program that will ask you to guess a number (hint: the number is 6). If you guess too low, too high, or just right, the program will let you know.

Step #1

Ask the user to input a number between 1 and 10 and assign it to a variable called **guess**:

```
guess = input('Pick a number between 1 and 10: ')
```

Step #2

Now that you have the user input stored as a string named **guess**, convert it to an integer, so you can compare it to other numeric values, to determine if the guess is too low, too high, or the correct number:

```
guess_int = int(guess)
```

This will take the string value of **guess**, convert it to an integer value, and store it back in a variable called **guess_int**.

Step #3

Compare the numeric value of **guess_int** to some different criteria. Use the first **if** condition to check if the guess is too low:

```
if guess_int < 6:

  print('Too low, better luck next time!')
```

This will check to see if the number is less than 6. If so, it will print the message in the print statement.

Step #4

Use an **elif** condition to check if the guess was greater than 6:

```
elif guess_int > 6:

  print('Too high, better luck next time!')
```

Step #5

Add the **elif** condition that will trigger if the user correctly guesses the number 6:

```
elif guess_int == 6:

  print('Great guess,  you are correct!')
```

Step #6

Add the final **`else`** condition that will display a message if the user's guess does not trigger any of the previous statements:

```
else:

  print('Invalid input, please try again')
```

Step #7

There's a problem with the code above. Can you spot it?

What if the user guesses 42? 42 is greater than 6 so the "Too high" print statement will be triggered. The rules only allow for guesses between 1 and 10, so you need to limit the conditions to be more specific. Fix the less than 6 condition first. You need to allow for guesses between 1 and 5, so anything greater than 0 or anything less than 6. That would look like this:

```
if guess_int > 0 and guess_int < 6:
```

This will work but you can combine the two conditions to make one shorter statement:

```
if 0 < guess_int < 6:
```

This condition will only evaluate as True for guesses between 0 and 5. We can apply this same logic to the greater than 6 condition to limit it guesses that are greater than 6 but less than or equal to 10:

```
elif 6 < guess_int < = 10:
```

Now if a user guesses 42 it will no longer trigger the **`'Too high'`** condition. That guess will trigger the else statement and print **`"Invalid input, please try again"`**.

Step #8

Verify the fully assembled program is free of errors:

```
guess = input('Pick a number between 1 and 10: ')

guess_int = int(guess)


if 0 < guess_int < 6:

  print('Too low, better luck next time!')

elif 6 < guess_int <= 10:

  print('Too high, better luck next time!')

elif guess_int == 6:

  print('Great guess,  you are correct!')

else:

  print('Invalid input, please try again')
```

Step #9

Try running the program and make a bunch of different guesses to ensure you get the expected output from each guess. Save the program if desired (optional).

LESSON A-15

QUESTIONS FOR UNDERSTANDING

1. Can Python print strings and integers together in the same print statement?

2. Is the indentation optional for nested if statements?

3. What modifications would the program from Activity #1 need to give the user their age in 20 years?

4. What modifications would the program from Activity #3 need to allow guesses from 1 to 100 and change the right answer to 42?

Answers Can be Found on the Next Page

Answers

1. **Can Python print strings and integers together in the same print statement?**

   ANSWER: No, strings and integers cannot be used together in the same print statement. You can only have all strings or all integers in a print statement. Since most print statements have letters or words included, it is generally necessary to convert integers to strings so they can be included in the print statement.

2. **Is the indentation optional for nested if statements?**

   ANSWER: No. Lack of indentation removes the conditional (nested) nature of the statements. Without indentation, the statements will each evaluate individually without relying on the conditions of the previous statements.

3. **What modifications would the program from activity #1 need to give the user their age in 20 years?**

   ANSWER: Yes, you can modify the code as follows (the change to the code is highlighted for your convenience):

```
age = input('How old are you: ')

age_int = int(age)


age_int = age_int + 20


age_str = str(age_int)

print('Your age in twenty years will be ' + age_str)
```

4. **What modifications would the program from Activity #3 need to allow guesses from 1 to 100 and change the right answer to 42?**

   ANSWER: Yes, you can modify the code as follows (the change to the code is highlighted for your convenience):

```
guess = input('Pick a number between 1 and 100: ')
guess_int = int(guess)


if 0 < guess_int < 42:
  print('Too low, better luck next time!')
elif 42 < guess_int <= 100:
  print('Too high, better luck next time!')
elif guess_int == 42:
  print('Great guess,  you are correct!')
else:
  print('Invalid input, please try again')
```

LESSON A-18

TWO PLAYER REACTION GAME

Lesson Overview

- ▶ Physical Inputs
- ▶ Electrical Configurations
- ▶ Random Module Command
- ▶ Using the Time Module
- ▶ Trimming a Long Number
- ▶ Using While Loops

Concepts to Review

- ▶ Building a Circuit with a Resistor and LED (Lesson A-2)
- ▶ Jumper Wires (Lesson A-4)
- ▶ Using Switches (Lesson A-5)
- ▶ Program Flow, Strings, Variables, Print Commands, Order (Lesson A-11)
- ▶ Code Organization (Lesson A-12)

Materials Needed

- ❑ Assembled Raspberry Pi setup from Lesson A-9 including Raspberry Pi, monitor, keyboard, and mouse
- ❑ 1K-Ohm Resistors x2
- ❑ 10K-Ohm Resistors x2
- ❑ Jumper Wires x3
- ❑ Pushbutton Switches x2

LESSON A-18

TWO PLAYER REACTION GAME

In this lesson, you will continue to add to your new coding skills by working with physical inputs, learning more ways to use the time and random modules, trimming long numbers, and a new way to use the loop command. You will then use the skills you've gained over the last 17 lessons to create a two-player game.

Inputs

In previous lessons you learned about programs that can take input from a user in the form of answering questions and save those answers as variables in your program. In this lesson, you will learn about a physical way programs can interact with users by using pushbutton switches attached to inputs on the Raspberry Pi.

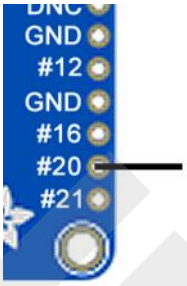
Physical inputs have three states they can operate in:

- Low – connected to ground
- High – connected to positive voltage supply; in the case of the Raspberry Pi this is 3.3-volts
- Floating – not connected to anything

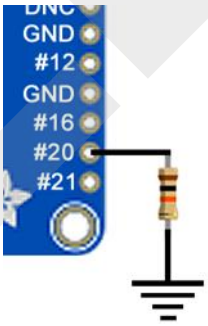
High and low are the best options for an input, since those states can be very easily identified by a program. A floating input should not typically be used because, just like the name, the voltage level on the input can float all over the place, causing the input to be read as high or low, unpredictably.

Electrical Differences in Configurations

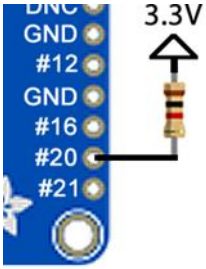
FLOATING: This condition is created when the input does not have a pull-up or pull-down resistor to set its initial electrical state. This will result in the input level “floating” up and down and can cause unreliable input results so it shouldn’t generally be used. Best practice is to use pull-up or pull-down resistors on inputs.



PULL DOWN RESISTOR: A large resistance value (like 10K-ohm) is used to pull the input low (ground). The resistor when used in this configuration is called a pull-down resistor. The value of this input is expected to be low unless its pulled high by other components.

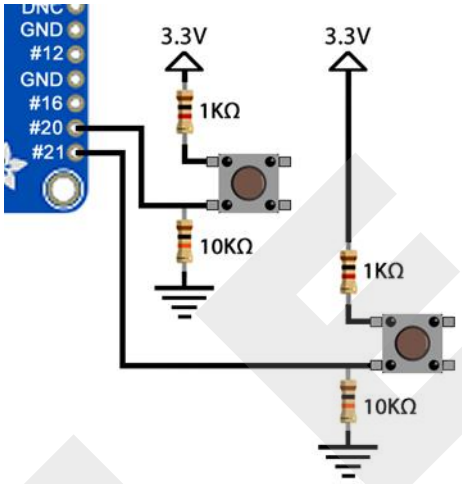


PULL UP RESISTOR: A large resistance value (like 1K-ohm) is used to pull the input high (3.3-volts). The resistor when used in this configuration is called a pull-up resistor. The value of this input is expected to be high unless its pulled low by other components.



Example Circuit

In the circuit for the game (Activity #1), you will use pull down resistors on two inputs and switches to pull those inputs high when you push the button. To the right is the diagram for that circuit.



GPIO 20 and 21 are always connected to ground using 10K-ohm resistors. Once a button is pressed, current will flow from the 3.3-volt power source, through the 1K-ohm resistor, through the switch, and into the input pin. The 1K-ohm resistor is in place to limit the amount of current allowed to flow into the input, protecting your Raspberry Pi.

Please note, when it comes to GPIO inputs, high is True and low is False. So, if you wanted to check if GPIO 20 is high you would use something like:

```
if GPIO.input(20) == True:

    print('GPIO 20 is high')
```

Or to check if GPIO 21 is low:

```
if GPIO.input(21) == False:

    print('GPIO 21 is low')
```

These types of input condition checks will come in handy when you start programming the game.

Random Module Command

You've already learned how to choose a random value from a list, but what if you don't want to use a separate list? You can do this by using the **`random.randint()`** command. The **`randint`** portion of this command stands for random integer. You can load the parentheses after this command with a range of values. For example:

`random.randint(2,25)` will choose a random number between 2 and 25

`random.randint(50,100)` will choose a random number between 50 and 100

You can use this in a program by setting a variable equal to this random value:

```
delay = random.randint(1,5)
```

This will cause **`random.randint`** to choose a random integer between 1 and 5, and save that value in a variable called **`delay`**.

Other Uses For the Time Module

You've already seen the time module used for making a program sleep for some preselected amount of time. Another part of this module can tell you the exact number of seconds that have happened since January 1, 1970. This point in time for computer systems is called the epoch, which means a notable point in time. Currently, just over 1.5 billion seconds have elapsed since that point in time.

The amount of time since then isn't really important, but it's what you can do with this precise time that is very useful. Running the command **`time.time()`** will give you the exact time from 1/1/70 to this moment, down to the 10 millionth of a second:

```
time.time()
```

This command will output something like 1518720190.4135857. A value like this can be used to very accurately time how long something takes in your program. Let's see how quickly python can print two separate print statements:

```
import time

print(time.time())

print(time.time())
```

Running this program might print these values:

1518728621.4256926

1518728622.4260027

Subtracting the first value from the second will give you the amount of time that elapsed between the two print statements. In this case, that value is 0.0003101, which you can round to 0.0003 seconds. That's three ten thousandths of a second between the first and second print statements. That should be fast enough for anything you hope to do with Python running on the Raspberry Pi!

If you wanted to time how long something took then you could use some code like this:

```
import time

time_start = time.time()

time.sleep(1)

time_total = time.time() - time_start

print(time_total)
```

This program will import the time module and create a variable called time_start that will equal the current time. The program will then pause for one second, then create a variable called time_total that will equal the new current time minus the original start time. The last line will print the value of time_total.

Trimming a Long Number

When working with large numbers like time.time() can output, it might be useful to round the digits to something shorter that can more easily be read. Python has a built-in formatting command for just this purpose:

```
rounded_number = '%.3f' % variable_to_round_off
```

In this example the numeric value **3** determines how many values will be left after the decimal. In this case **variable_to_round_off** will be rounded to the third decimal place and saved as the variable **rounded_number**.

Running **'%.3f'** on 1.4256926 will give you 1.426

Running **'%.1f'** on 1.4256926 will give you 1.4

Running **'%.0f'** on 1.4256926 will give you 1

This will make working with those huge time values much easier when you create the game program.

While Loops

A while loop can be used to make your program keep trying to do something, over and over, until some condition is met. Look at the following code:

```
loop = 'yes'



while loop == 'yes':

    print('Still looping')
```

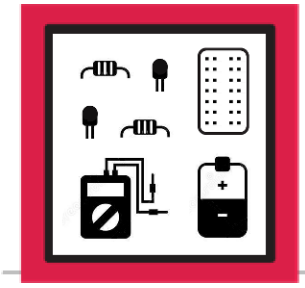
> **WARNING!**
>
> Don't try running the exact code above as it will result in an endless loop, which means there is no way to exit the loop. You can stop the program manually in Thonny, but the program as coded here will try to run forever.

The value of loop is initially set to **yes**. The while loop will keep running until **loop** no longer equals **yes**. Since there is no code to change the value of **loop** to anything but **yes**, the while loop will execute forever.

This is where something like a GPIO input pin can come in handy. You can tell the loop to run over and over until a GPIO pin is high:

```
loop = 'yes'



while loop == 'yes':

     print('Still looping')

     if GPIO.input(20) == True:

          loop = 'no'
```

Every time the loop runs through, **`Still looping`** will be printed, and the if statement will be evaluated. If GPIO 20 is low (False) during the check then nothing will change, and the loop will continue running, over and over. If GPIO 20 is high (True) during one of these condition checks, then the value of loop will be changed to 'no' and the while loop will no longer run, moving on to any code in the program below the while loop.

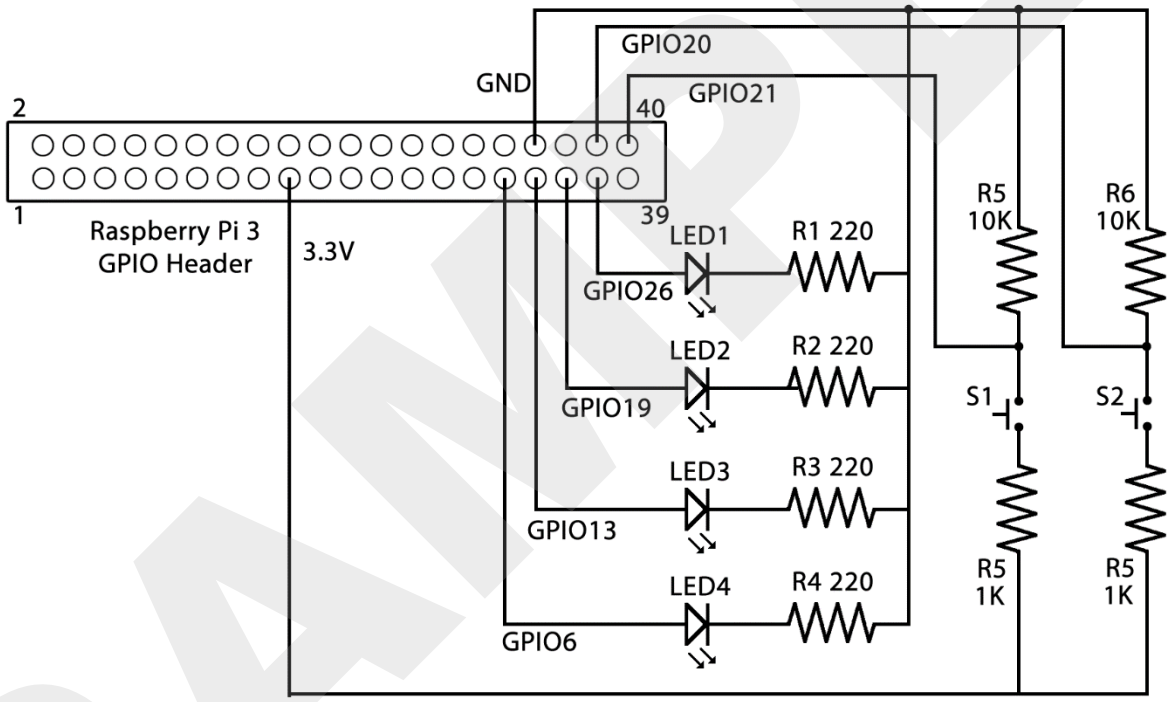


LESSON A-18

ACTIVITY #1:
MODIFY THE CIRCUIT TO ADD THE SWITCHES

The activities for this final lesson for Level A are designed to combine many of the skills you have acquired over the last 17 lessons to build a Python program and a circuit to allow you and a friend to play a two-player game testing your reaction time.

In this activity you will modify the four LED circuit from Lesson A-17, Activity #3, to add two switches so it can be used in a two-player game.



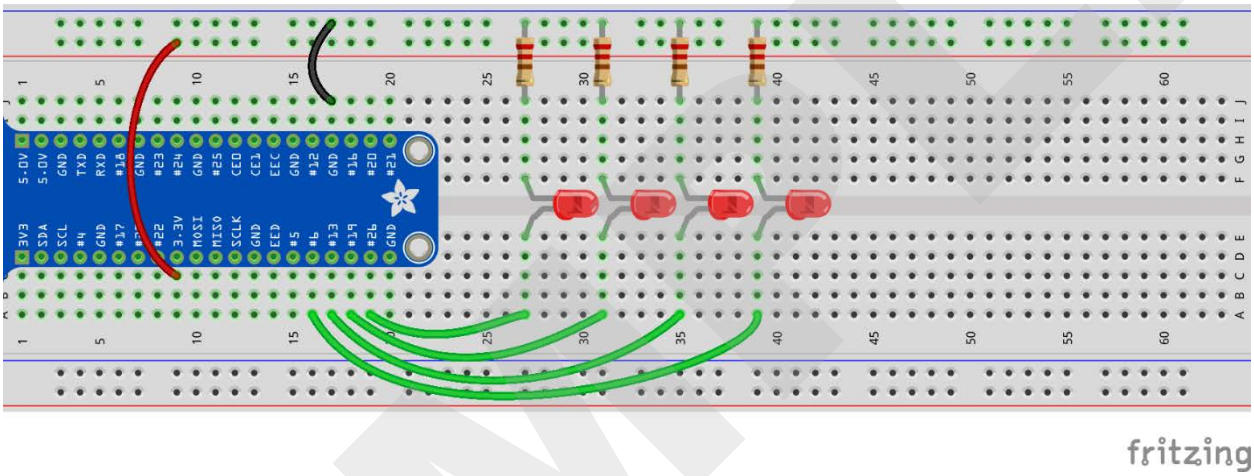
Step #1

Power down your Raspberry Pi so you can modify the breadboard components.

Use the four LED circuit from Lesson 17, Activity #3 as a starting point.

Step #2

Connect a jumper wire from C9 to P2-9 to carry 3.3-volts from the Raspberry Pi over to the P2 bus:

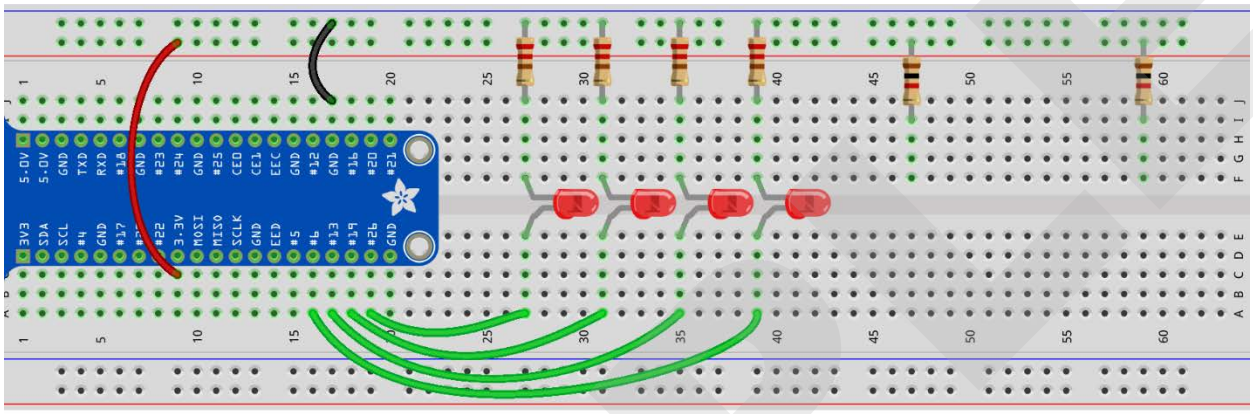


Step #3

Now the positive rail P2 has 3.3-volts available. You now need to make that available to the switches using some 1K-ohm resistors. Add two 1K-ohm resistors in the following locations:

R1 between P2-47 and I47

R2 between P2-59 and I59.

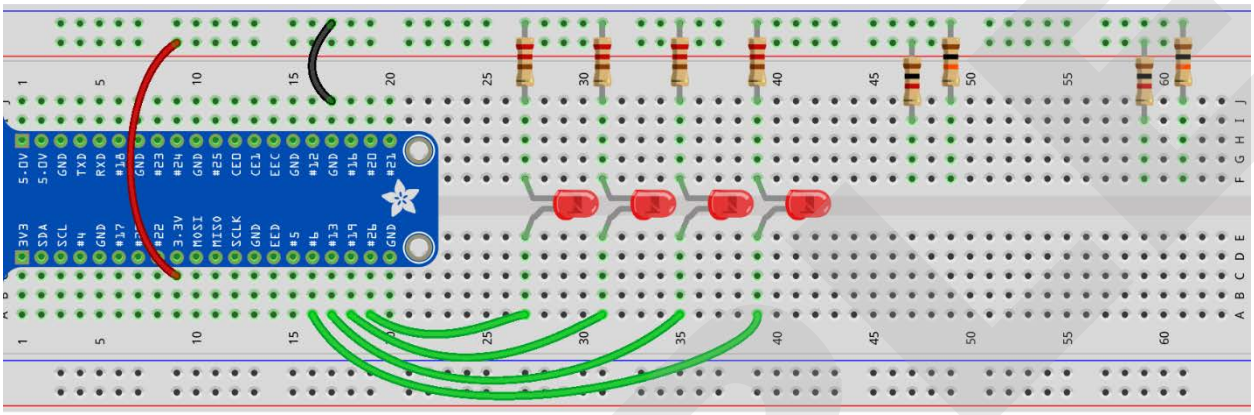


Step #4

The switches will need a path to ground using the 10K-ohm resistors. Add two 10K-ohm resistors between the following points:

R3 between N2-49 and J49

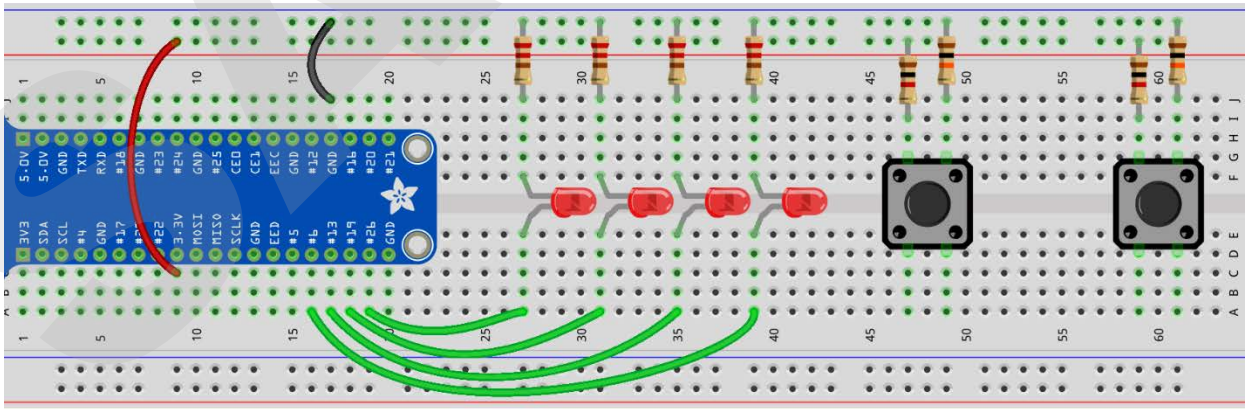
R4 between N2-61 and J61



Step #5

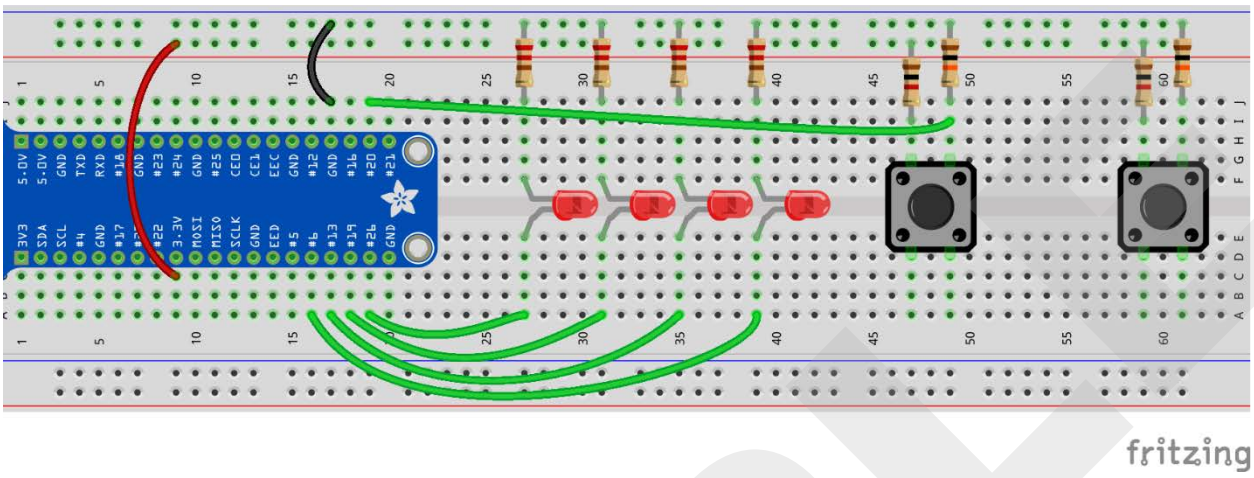
Insert the switches. Only two of the four pins on each switch will be used. Connect two pins on each switch across these locations:

Switch 1 - G47 and G49, Switch 2 – G59 and G61.



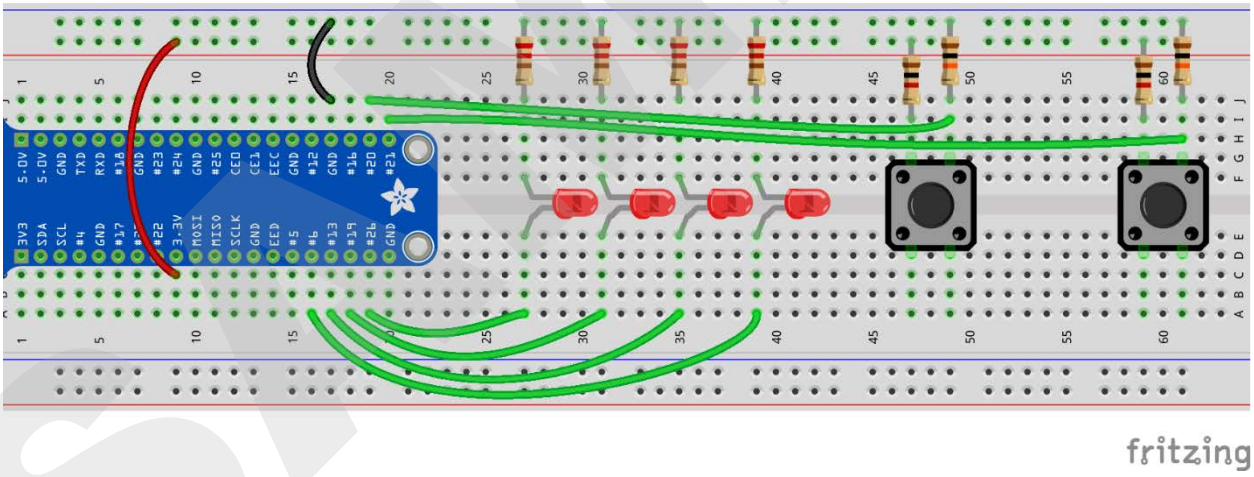
Step #6

Each switch must be wired to an input. Connect switch 1 to GPIO 20 by adding a jumper wire between H19 and I49:



Step #7

Connect switch 2 to GPIO 21 by adding a jumper wire between I20 and H61:



Step #8

Double check all connections against the drawing above, and then power up the Raspberry Pi.

Your circuit is now complete. In the next activity you will write a program that will use the new switches.

LESSON A-18

ACTIVITY #2:
CODING THE TWO PLAYER GAME

In this activity you will write a program that will illuminate an LED with a 1 to 5 second delay, start a timer when the LED turns on, stop the timer when a player presses a button, and display who pressed first and that player's winning reaction time.

Step #1

In this program you will be talking to GPIO pins, using the **`time.sleep()`** command, and the **`random.randint()`** command. Open Thonny and start the program by loading the appropriate modules:

```
import RPi.GPIO as GPIO

import time

import random
```

Step #2

Setup your board numbering type and GPIO pin assignments.

```
GPIO.setmode(GPIO.BCM)
```

Step #3

Configure the GPIO pin assignments. GPIO 26 will be the output for the LED and GPIO 20 and 21 will be used as switch inputs:

```
GPIO.setup(26, GPIO.OUT)

GPIO.setup(20, GPIO.IN)

GPIO.setup(21, GPIO.IN)
```

Step #4

You could turn on the start LED as soon as the program starts, but instead add a random delay to randomize the number of seconds to delay before the LED turns on.

Assign a random integer from 1 to 5 to the variable called **wait**. Use **time.sleep** to sleep for an amount of seconds equal to the (wait) value:

```
wait = random.randint(1,5)

time.sleep(wait)
```

The first statement will create a variable named wait and set it equal to the value of a random integer between the starting point of 1 and stopping point of 5. Now that a 1, 2, 3, 4, or 5 has been randomly selected for the value of **wait**, you can call the **time.sleep** command and specify a delay in seconds equal to the value of **wait**.

Step #5

Now that the random wait has ended, it's time to turn on the start LED and start a timer. You can turn on the LED by pushing GPIO 26 high. You can start the timer by making a variable called **timestart** equal the current value of **time.time()**:

```
GPIO.output(26,GPIO.HIGH)

timestart = time.time()
```

Step #6

Now that the start LED is on, and you have stored the current time, make a while loop to check for a winner. Create a variable called winner and set it equal to 'No'. This will allow you to continue running the program until there is a winner:

```
winner = 'No'
```

Step #7

Create a while loop that will keep the while loop running until a button is pressed:

```
while winner == 'No':
```

The contents of this while statement will continue to execute as long as winner equals 'No'. If winner equals anything other than 'No', the loop will be skipped, and the remainder of the program will run.

Step #8

Add some if statements to the while loop so you can modify the value of winner based on which player pushed their button first. Player 1 will be assigned to GPIO 20, which is connected to the switch closest to the wedge. Player 2 will be assigned to GPIO 21, which is connected to the switch furthest from the wedge.

```
while winner == 'No':

  if GPIO.input(20) == True:

    winner = 'Player 1'

  if GPIO.input(21) == True:

    winner = 'Player 2'
```

Each time the while loop is executed the if statements will be checked for truth. If input GPIO 20 is high (True) during the check then winner will be set to ‘Player 1’. If input GPIO 21 is high (True) during the check then winner will be set to ‘Player 2’. If neither input was high then the value of winner will not be modified, and the while loop will run again.

Note:

Make sure the indentation is correct on the while loop or it will not loop properly.

Step #9

Build the code that will run once a winner is determined. First, create a variable called **time_total** that will be equal to the current time using the **time.time()** command minus the starting time that was stored in **timestart**:

```
time_total = time.time() – timestart
```

This will take the current time in seconds and subtract the previously stored **timestart** value in seconds. The result is the number of seconds between turning on the LED and Player1 or Player2 pressing a button.

Step #10

The value of the number calculated in Step #9 is a very long number like 1.28389596939. You can get this into a more display-friendly format by removing all digits past thousandths of a second:

```
time_total = “%.3f” % time_total
```

This will take the value of **time_total**, remove everything past 3 decimal places, and save the result back to the **time_total** variable. So, 1.28389596939 will become 1.284. Now you have a reaction time that can be displayed nicely in a print statement.

Step #11

Now that there is a winner and a reaction time, you can build some print statements that let you know who won and what their reaction time was:

```
print(winner + ' wins!')

print('Your reaction time was ' + time_total + ' seconds')
```

The first statement will print the value of winner with " wins!" added to the end of the string. A single space has been added before the word "wins" in order to ensure correct spacing between Player 1 or Player 2 and wins. Without the additional space you will end up with Player 1wins! or Player 2wins! in your console output. The same behavior can happen in the second statement, with "was " and " seconds". There are extra spaces to properly space the words.

Step #12

Now that you've printed the winning player and their reaction time, turn off the LED and clean up all the GPIO pins:

```
GPIO.output(26, GPIO.LOW)

GPIO.cleanup()
```

This allows the program to exit gracefully.

Step #13

Verify the fully assembled program is free of errors:

```
import RPi.GPIO as GPIO

import time

import random


GPIO.setmode(GPIO.BCM)

GPIO.setup(26, GPIO.OUT)

GPIO.setup(20, GPIO.IN)

GPIO.setup(21, GPIO.IN)


wait = random.randint(1,5)

time.sleep(wait)


GPIO.output(26,GPIO.HIGH)

timestart = time.time()


winner = 'No'


while winner == 'No':
  if GPIO.input(20) == True:
    winner = 'Player 1'
  if GPIO.input(21) == True:
    winner = 'Player 2'


time_total = time.time() - timestart
```

```
time_total = “%.3f” % time_total


print(winner + ' wins!')

print('Your reaction time was ' + time_total + ' seconds')


GPIO.output(26, GPIO.LOW)

GPIO.cleanup()
```

Step #14

Find a friend or family member who will be most impressed with the skills you have gained during Level A of this course and play the game with them. Remember to let them win occasionally!

Save this program for future game playing or to modify in the future as your skills develop further.

LESSON A-18

QUESTIONS FOR UNDERSTANDING

1. Can you change Player 1 and Player 2 to other names?

2. How would you increase the random timer range from 1-5 seconds to 1-10 seconds?

3. Can you change which of the four LEDs is used to start the game?

Answer Can be Found on the Next Page

Answers

1. **Can you change Player 1 and Player 2 to other names?**

   ANSWER: Yes, you can modify the code as follows (the changes to the code have been highlighted for your convenience):

```
while winner == 'No':

  if GPIO.input(20) == True:

    winner = 'Arthur 1'

  if GPIO.input(21) == True:

    winner = 'Marvin 2'
```

2. **How would you increase the random timer range from 1-5 seconds to 1-10 seconds?**

   ANSWER: You can modify the code as follows (the changes to the code have been highlighted for your convenience):

```
wait = random.randint(1,10)

time.sleep(wait)
```

3. **Can you change which of the four LEDs is used to start the game?**

   ANSWER: Yes. These statements throughout the program set up GPIO 26 as an output, push GPIO 26 high, and then pull GPIO 26 back low:

   GPIO.setup(26, GPIO.OUT)

   GPIO.output(26, GPIO.HIGH)

   GPIO.output(26, GPIO.LOW)

   Your circuit also has LEDs attached to GPIO 6, GPIO 13, and GPIO 19. Changing the pin number in all three of these lines to either 6,13, or 19 will cause it to use that LED instead.

42 ELECTRONICS

INTRO TO ROBOTICS LEVEL A

Scope and Sequence

Lesson 1

Introduction to Components: Batteries and Breadboards

- What is Electricity?
  - Static Electricity vs. Current Electricity
  - Voltage, Current, and Resistance
- What is a Circuit?
  - Short Circuits
  - Open Circuits vs. Complete Circuits
- Circuit Components
  - Batteries
    - Anode vs. Cathode
  - Breadboards
    - Soldered Circuits vs. Breadboard Circuits
    - Breadboard Connections and Power
- Activity: Powering Breadboard Connections

Lesson 2

Introduction to Components: Resistors and LED

- Resistors
  - How Resistors Work
  - Using Resistors to Build Circuits
  - Calculating Resistance Value
- Light Emitting Diode
  - How LEDs Work
  - LED Polarity
  - Pairing Resistors and LEDs
- Calculating Forward Voltage
- Activities
  - Activity #1: Build a Circuit to Illuminate an LED
  - Activity #2: Build a Series Circuit

Lesson 3

Series vs. Parallel Circuits and Ohm's Law

- Series vs. Parallel Circuits
  - Limitations to Series Circuits
  - Resistor Use in Parallel Circuits
  - Understanding Voltage, Resistance, and Current in Serial vs. Parallel Circuits
- Ohm's Law
  - Introduction to Ohm's Law
  - Mathematical Formula
  - Ohm's Law Calculation Example
- Activities: Build a Parallel Circuit

Lesson 4

Introduction to Components: Jumper Wires

- Jumper Wire
  - Uses for Jumper Wire
  - Size and Type of Jumper Wire
  - Spacing Components to Avoid Short Circuits
- Activity: Build a Circuit Using Jumper Wires

Lesson 5

Introduction to Components: Switches

- Switches
  - Common Uses for Switches
  - Types of Switches
    - Maintained vs. Momentary
    - Normally Open vs. Normally Closed
  - Poles and Throws
- Labeling Components
- Activities
  - Activity #1: Controlling Two LEDs with One Switch
  - Activity #2: Using Two Switches to Independently Control LEDs

Lesson 6

Introduction to Components: Red-Blue-Green LED (RGB LED)

- RGB LEDs
  - Common Anode vs. Common Cathode
  - Proper Placement in Breadboard
- Activities
  - Activity #1: Illuminate the Red Element of the RGB LED
  - Activity #2: Add the Blue Element on a Switch
  - Activity #3: Controlling Colors on a Switch

Lesson 7

Troubleshooting Circuits

- Introduction to Troubleshooting
- Troubleshooting Steps:
  - Verify There is a Failure
  - Check the Simplest or Most Likely Solution First and Retest
  - Half-Splitting
  - Repair the Problem and Retest
- Practical Applications
  - Intermittent Problems
  - Equipment Failure
- Activities
  - Activity #1: Building and Troubleshooting a Circuit
  - Activity #2: Additional Troubleshooting Practice

Lesson 8

Introduction to Reading Schematics

- Schematics
  - Reading Schematics
  - Common Schematic Symbols
    - Wires
    - Power
    - Switches
    - Resistors
    - Diodes
    - Capacitors
    - Transistors
    - Integrated Circuits
    - Header
- Activities
  - Activity #1: Building a Series Circuit Using a Schematic
  - Activity #2: Building a Parallel Circuit Using a Schematic
  - Activity #3: Working with a Schematic

Lesson 9

Setting Up the Raspberry Pi

- Raspberry Pi Hardware
- Raspberry Pi Software
    - Types of Software
        - Raspian OS
        - Python
        - Nano
        - Thonny
    - Types of Interface
        - GUI
        - Terminal
- Optional Lesson: Understanding Sudo and Update Commands
    - APT-GET Update
    - APT-GET DIST-UPGRADE
- Activities
    - Activity #1: Installing the Raspberry Pi in a Protective Case
    - Activity #2: Connecting Peripherals to the Raspberry Pi
    - Activity #3: Safely Powering the Raspberry Pi On and Off
    - Activity #4: Connecting the Raspberry Pi to the Internet
    - Activity #5: Updating the Raspberry Pi's Software

Lesson 10

Introduction to Software: Terminal and Thonny

- Nano Overview
- Thonny Overview
- Error Checking Options
- Activities
    - Activity #1: Creating a Python Program in Nano
    - Activity #2: Creating a Python Program in Thonny
    - Activity #3: Exploring Thonny's Error Checking Features

Lesson 11

Creating Python Programs

- Program Flow
- Strings
- Variables
    - Spaces and Capitalization
    - Integers
    - Equations
- Print Command
    - Printing a String
    - Printing a Variable
- Order
- Activities
    - Activity #1: Reading and Writing Basic Python Code
    - Activity #2: Writing Basic Python Code

Lesson 12

Code Organization, User Input, and Merging Strings

- Keeping Code Organized
    - Carriage Returns
    - Comments
        - Formatting Comments
        - Commenting Out Code
- User Input
- Merging Strings (Concatenation)
- Activities
    - Activity #1: Reading and Writing Python Code
    - Activity #2: Writing a Simple Program in Python

Lesson 13

Math Functions, Lists, and Importing Modules

- Math Functions
- Lists
  - Formatting Lists
  - Index Values
- Importing and Using Modules
  - Time Module
  - Random Module
- Activities
  - Activity #1: Exponential Math Calculations
  - Activity #2: Importing the Random Module
  - Activity #3: Random Dice Program
  - Activity #4: Importing the Time Module
  - Activity #5: Times Up! Game

Lesson 14

Introducing If/Else Statements

- Boolean Logic
  - Coding Comparison Operators
  - Connecting Multiple Logic Expressions
- Programming for Decisions
  - If Statements
  - Else Statements
  - Using Multiple Statements Inside an If Statement
  - Elif Statements
  - Formatting Concerns
- Activities
  - Activity #1: Using Boolean Logic
  - Activity #2: Deciphering Code
  - Activity #3: Writing Logical Code

Lesson 15

Nested If Statements and String/Integer Conversion

- Nested If Statements
  - Indentation
- Strings vs. Integers
  - Converting a Value to an Integer
  - Converting a Value to a String
- Activities
  - Activity #1: Add Five Years to Your Age
  - Activity #2: Age Calculator
  - Activity #3: Guess A Number

Lesson 16

Controlling a Breadboard Circuit with the Raspberry Pi

- General Purpose Input Output (GPIO)
  - Pin States: Low vs. High
  - Outputs
- GPIO Header
  - GPIO Pin Numbering
  - GPIO Header Pin Assignments
- Python Commands and Process for Working with GPIO Pins
  - Importing Module
  - Specifying Pins
  - Cleanup Operations
- Activities
  - Activity #1: Preparing the Equipment for Connection
  - Activity #2: Powering an LED Using the Raspberry Pi

Lesson 17

Loops

- Introduction to Loops
  - Coding Loops
- Activities
  - Activity #1: Build a 4 LED Circuit
  - Activity #2: Create a Program to Test Circuit Functionality
  - Activity #3: Using Loops to Control LEDs

Lesson 18

Final Project: Two Player Reaction Game

- Inputs
  - Electrical Differences in Configurations
- Another Random Module Command
- Other Uses for the Time Module
- Trimming a Long Number
- While Loops
- Activities
  - Activity #1: Add Switches to the Circuit
  - Activity #2: Coding the Two Player Game