LESSON 18

TWO PLAYER REACTION GAME

OBJECTIVE

For your final project for this level for the course, create a game in Python that will test your reaction time versus another player.

MATERIALS

This lesson will use the equipment set-up you assembled in Lesson 9 including the Raspberry Pi, monitor, keyboard, and mouse. No internet connection is needed.

You will also need the completed circuit built in Lesson 17, Activity #3 connected to the Raspberry Pi via the ribbon cable and wedge.

In addition, you will need:

- 2 x 1k-ohm Resistor
- 2 x 10k-ohm Resistor
- 3 x Jumper Wire
- 2 x Switches

REVIEW CONCEPTS

If you do not feel comfortable with the following concepts, please review them before proceeding.

- How to connect a LED and resistor to form a functional circuit (Lesson 2)
- How to use jumper wire (Lesson 4)
- How to use switches (Lesson 5)
- Program Flow, Strings, Variables, Print Commands, Order (Lesson 11)
- Code Organization (Lesson 12)

- Math, Lists, Importing Modules, Module Commands (Lesson 13)
- Using a ribbon cable and wedge to attach the Raspberry Pi to the breadboard (Lesson 16)
- Configuring GPIO Pins (Lesson 16)
- Loops (Lesson 17)

LESSON

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 high or low unpredictably.

ELECTRICAL DIFFERENCES IN CONFIGURATIONS

Here are the electrical differences between these 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.

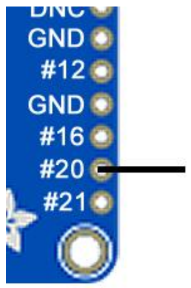


Figure 18-1 Input floating

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.

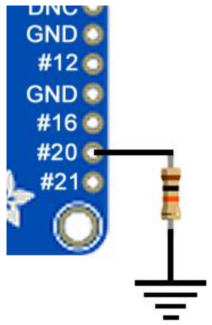


Figure 18-2 Input with pull-down resistor

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.

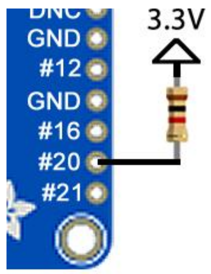


Figure 18-3 Input with pull-up resistor

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. Here is the diagram for that circuit:

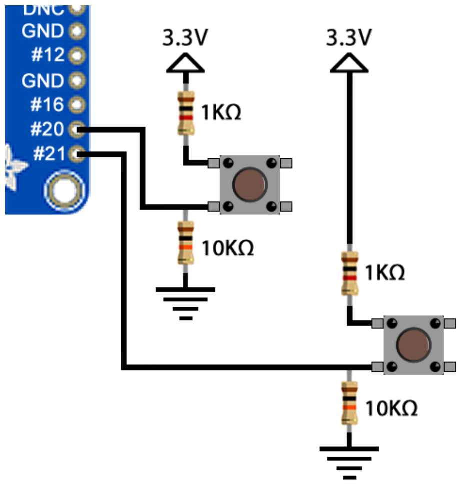


Figure 18-4 Circuit for connecting switches to two GPIO inputs

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 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 this exact code 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.

ACTIVITIES

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.

ACTIVITY #1 – MODIFY THE CIRCUIT TO ADD THE SWITCHES

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

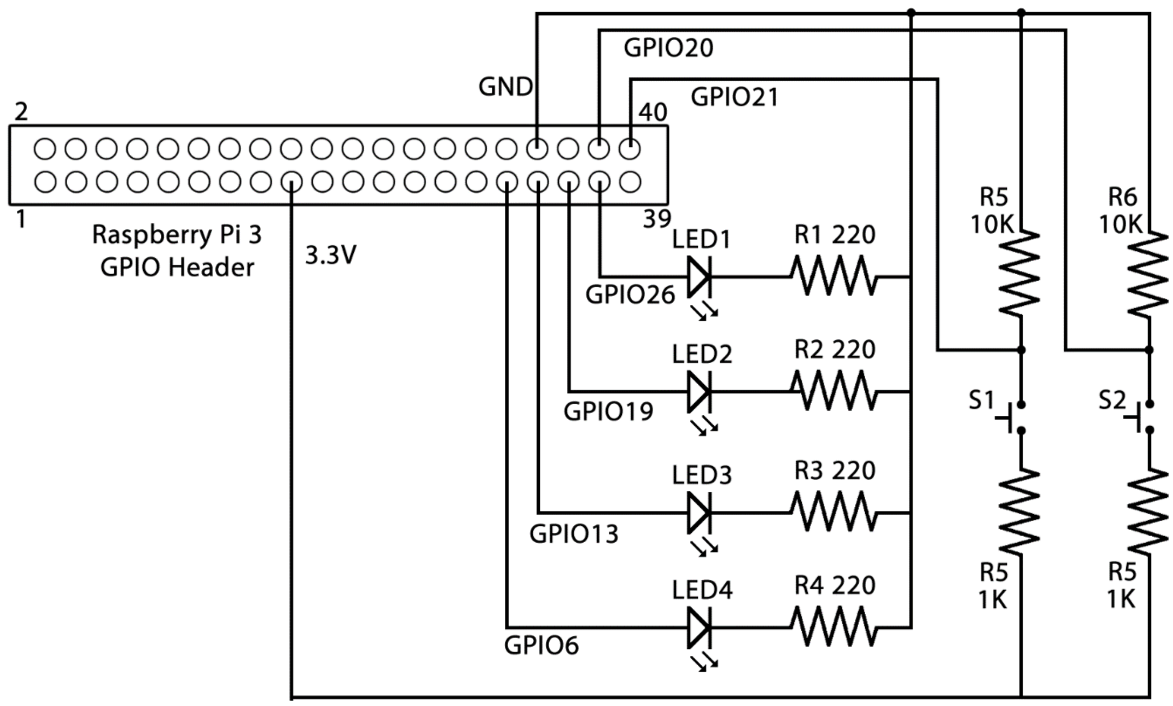


Figure 18-5 Schematic diagram of four LEDs and two pushbutton switches connected to GPIO pins

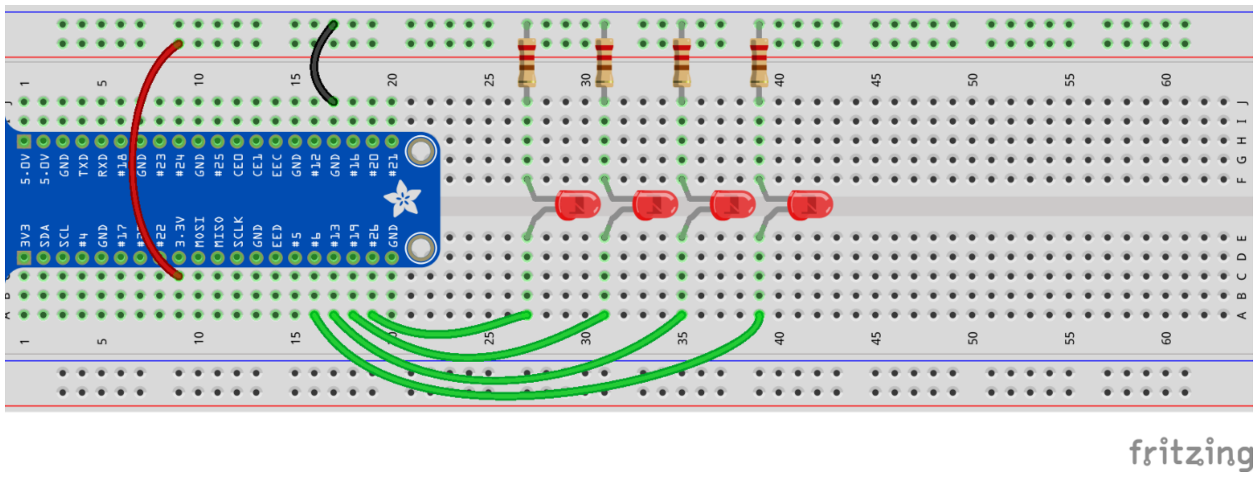
STEP #1

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:

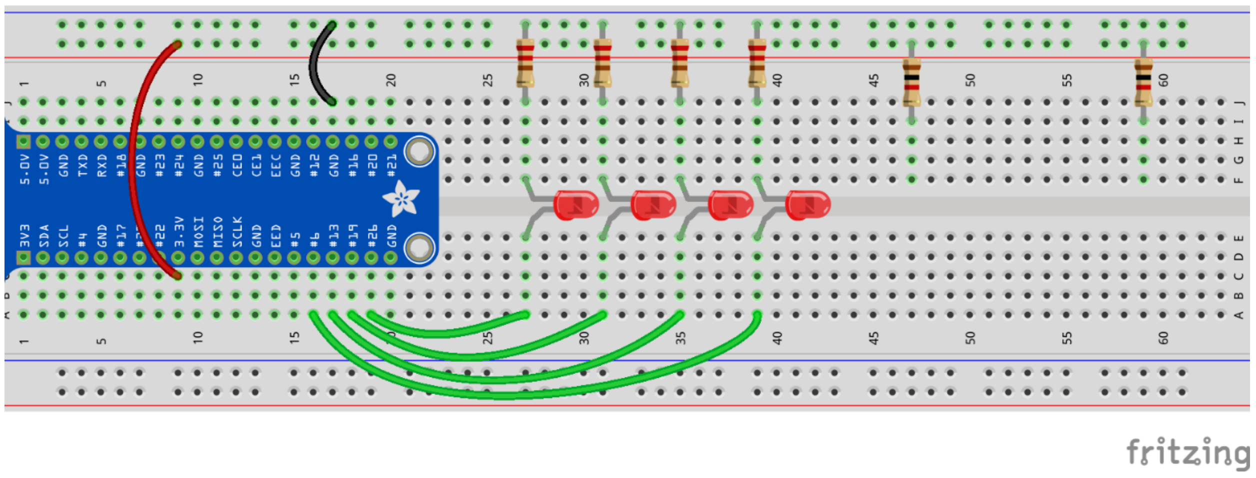


STEP #3

Now that positive rail P2 has 3.3-volts. You 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.

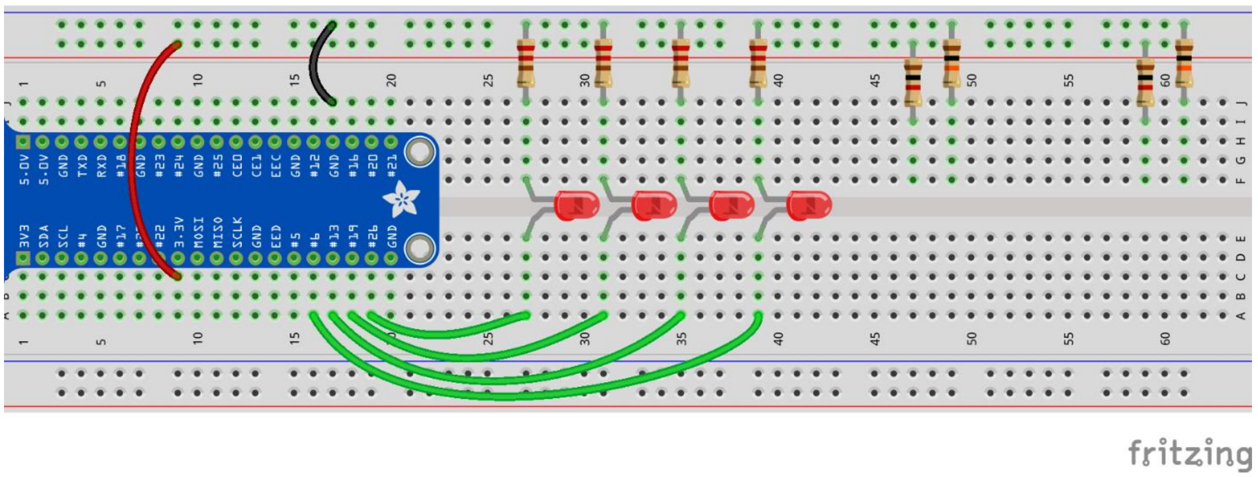


STEP #4

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

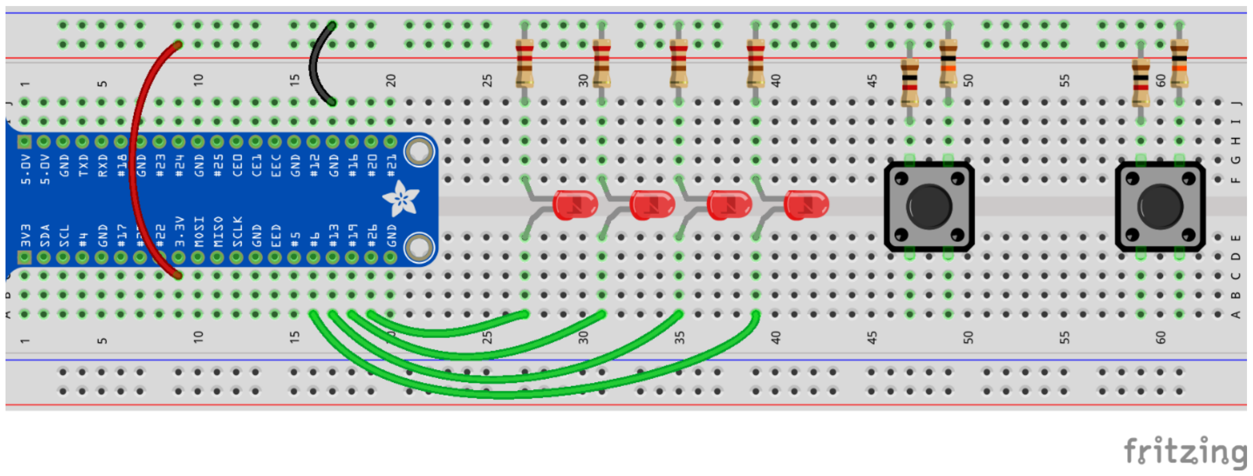
R4 between N2-61 and J61



STEP #5

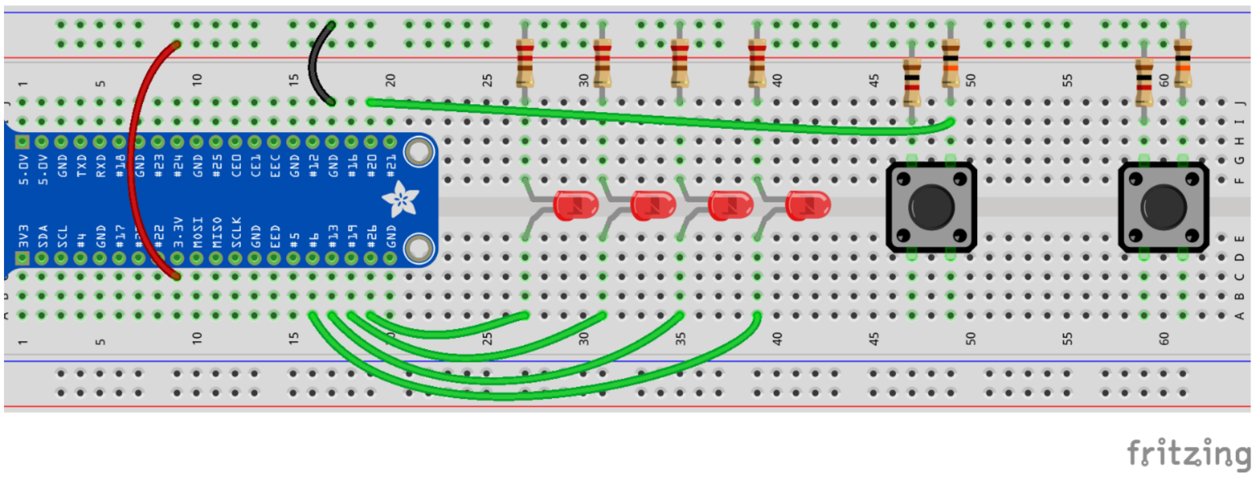
Insert the switches. Only two of the four pins on each switch will be used. *Connect two pins on each switch across these locations*:

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



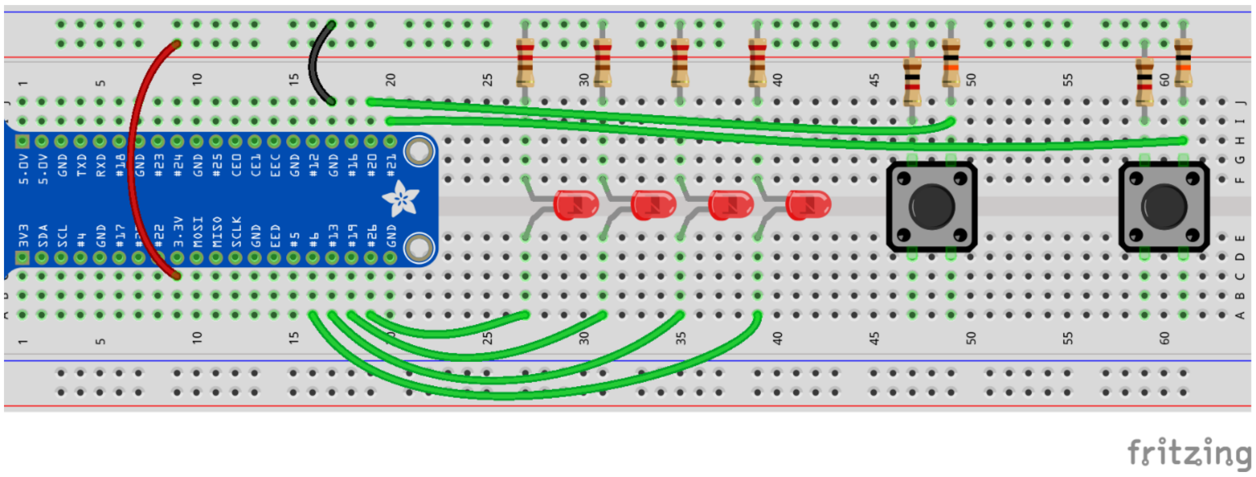
STEP #6

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.

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. Player1 will be assigned to GPIO 20, which is connected to the switch closest to the wedge. Player2 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`** that will be equal to the current time using the **`time.time()`** command minus the starting time that was stored in **`timestart`**:

```
time = 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 stripping off all digits past thousandths of a second:

```
time = “%.3f” % time
```

This will take the value of time, strip off everything past 3 decimal places, and save the result back to the time 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 + ' 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 = time.time() - timestart

time = “%.3f” % time



print(winner + ' wins!')

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



GPIO.output(26, GPIO.LOW)

GPIO.cleanup()
```

STEP #14

Find a friend of 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!

Optional: Save the program if desired.

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?

Answers can be found on the next page.

ANSWERS TO THE QUESTIONS FOR UNDERSTANDING

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.

CONCLUSION

Congratulations! You have completed Level A of this course. You have learned a great deal of new skills and you should be proud of yourself!

What's next?

- Order a copy of Level B of this course to continue to learn to work with more electronic components and coding commands: www.42electronics.com/products/level-b-curriculum-kit

- Check out www.42electronics.com/level-a-extra-projects for a list of projects you can practice with while you wait for Level B to arrive.

Intro to Robotics Course of Study:

Level A: Building Circuits and Beginning Programming

Level B: Working with Sensors and Intermediate Programming **NEXT STEP**

Level C: Incorporating Video and Advanced Programming

Level D: Working with Motors and Taking It Mobile