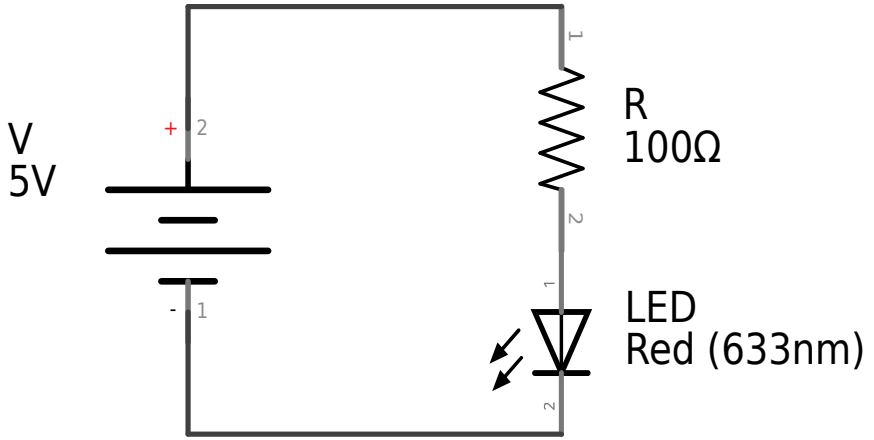
# *12 Ways to Blink an LED*

Charlotte Hackerspace
Neil Roeth

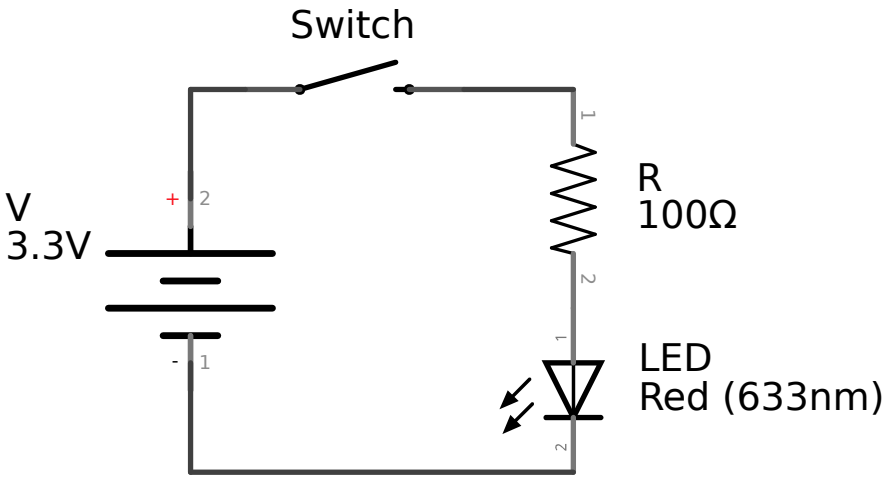
# LED basic circuit

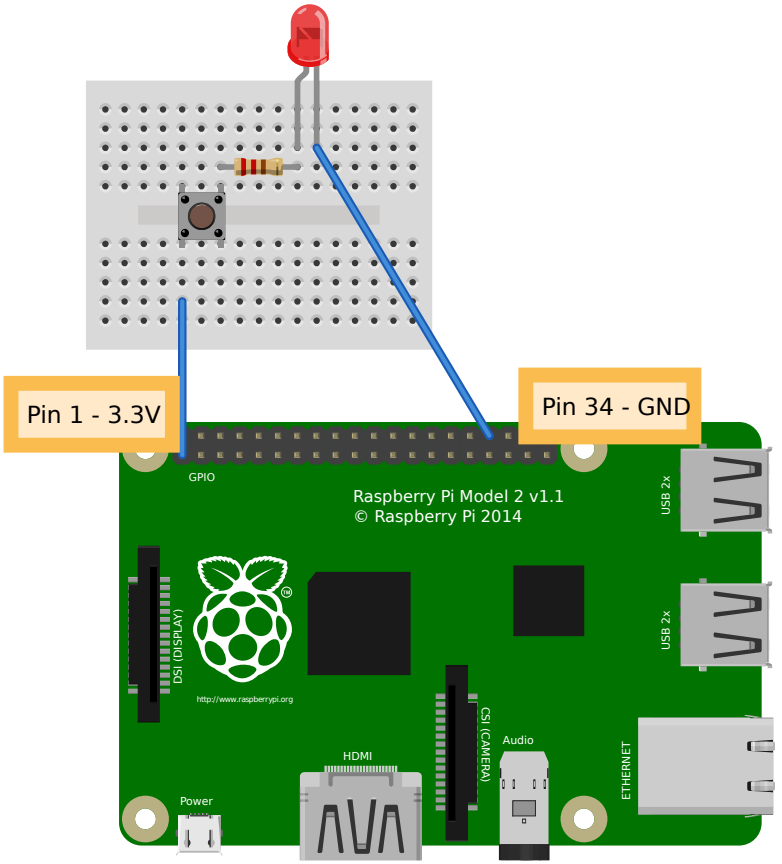



- Need to limit current (no magic smoke)
- $V = V_{LED} + V_R$
- $V_R = IR$ (Ohm's Law)
- $V = V_{LED} + IR \Rightarrow R = (V - V_{LED})/I$
- LED: $V_{LED} \sim 2.5V$

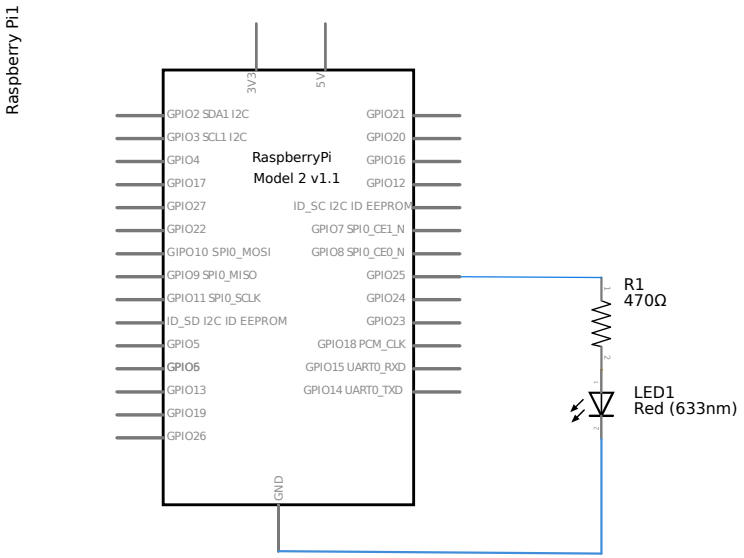
- Raspberry Pi: 3.3V, 16mA
- $R = (3.3 - 2.5)/0.016 = 50$ ohms
- Arduino: 5V, 40mA
- $R = (5 - 2.5)/0.040 = 62.5$ ohms
- Bigger is safer

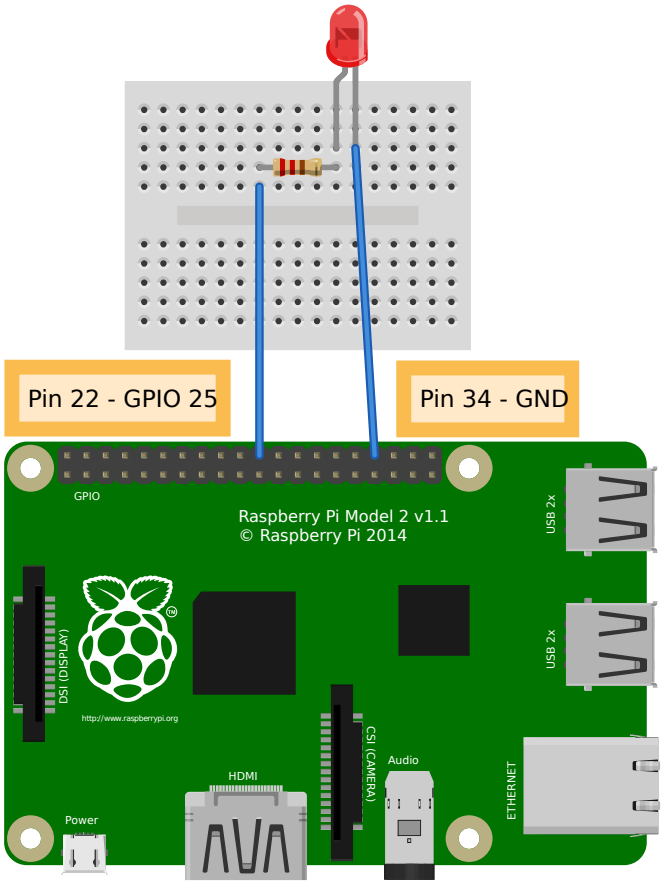
# Blink LED with pushbutton



# Blink LED with Pi



- Create a file named BlinkRaspberryPi.py with Python code:

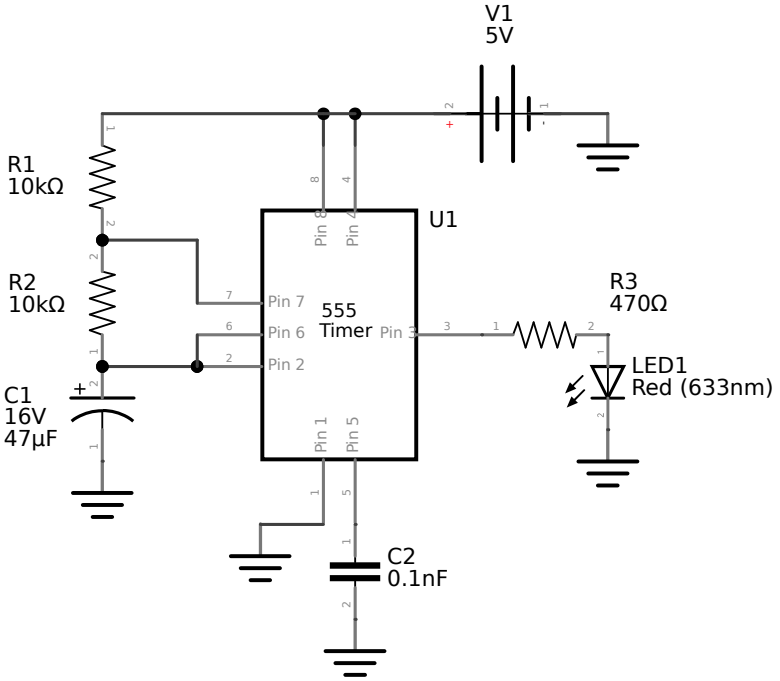
```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
GPIO.setup(25, GPIO.OUT)

while True:
    GPIO.output(25, GPIO.HIGH)
    time.sleep(1)
    GPIO.output(25, GPIO.LOW)
    time.sleep(1)
```

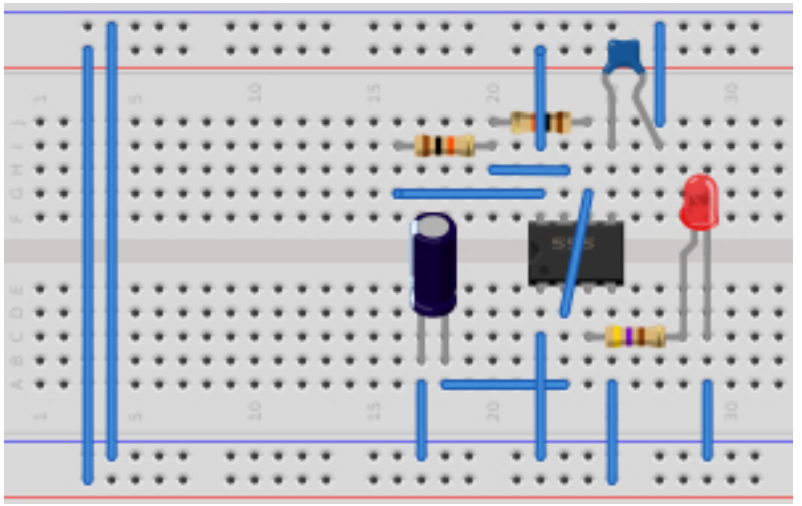
- Ensure you have the Python libraries for accessing GPIO pins installed:

```
$ sudo apt-get install python-rpi.gpio python3-rpi.gpio
```

- Run it: `python BlinkRaspberryPi.py`
- What does script do?
- Raspberry Pi has digital outputs only (HIGH, LOW)

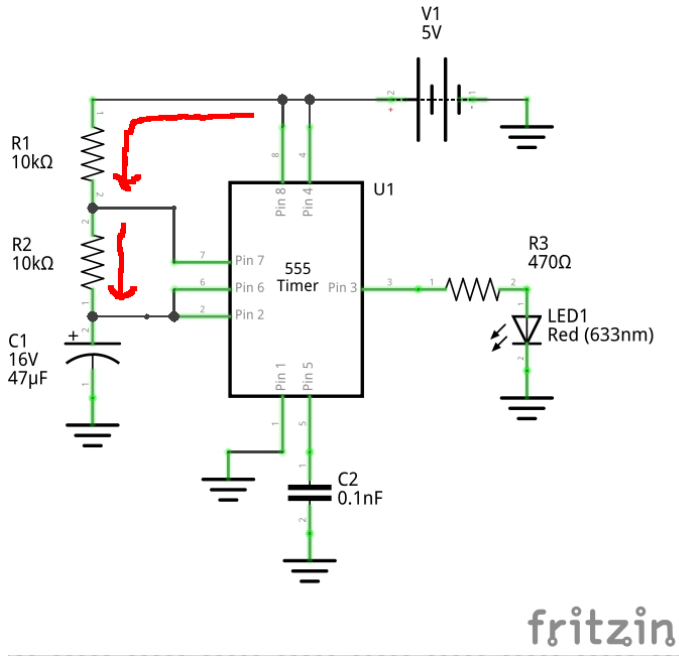
# 555 Astable Multivibrator

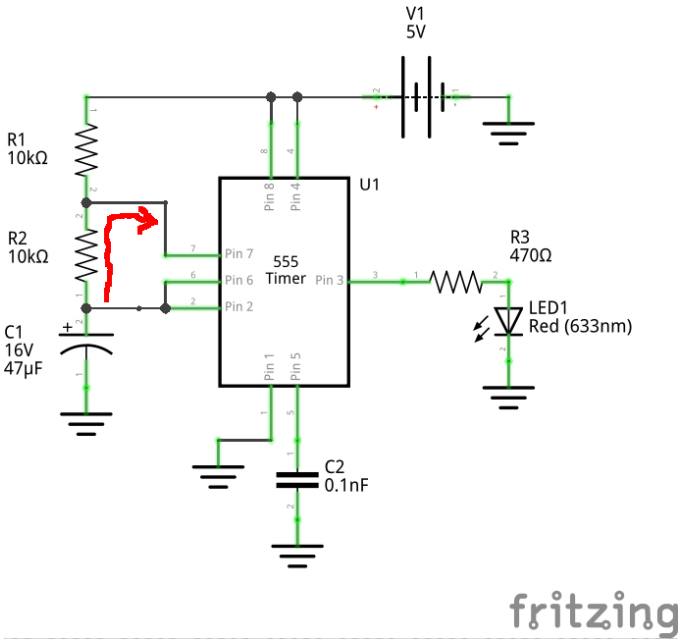


- Connect 5V from Pi to red rail, ground to blue rail
- Dot on IC is pin 1, C1 has +/-
- Timing is determined by how quickly capacitor C1 charges/discharges

# How the 555 circuit works




- Charging (left): Capacitor C1 charges through R1 and R2
- When capacitor voltage reaches 2/3 V1, pin 7 connects to ground
- Discharging (right): Capacitor C1 discharges through R2
- When capacitor drops to 1/3 V1, pin 7 disconnects from ground
- Water analogy: voltage like pressure, current like flow rate
- Capacitor like bucket, resistor like hose restriction
- Bigger capacitor or bigger resistor means longer time
- Time constant for charging = (R1+R2)*C1
- Time constant for discharging= R2*C1

# Arduino

- https://www.arduino.cc/ , Download, “Linux ARM (experimental)”
- `$ cd ~/Downloads`
- `$ tar Jxf arduino-1.6.*-linuxarm.tar.xz`
- `$ cd arduino-1.6.*-linuxarm`
- `$ ./install.sh`
- `$ ./install.sh`
- Start the Arduino IDE, set Board and Port
- Open the Blink example, compile and upload
- If using Nano, use built in LED on pin 13
- If not Nano, wire LED and resistor to a pin, change program to suit.

# ATtiny

- Attiny 45 is a tiny Arduino (6 I/O pins, 4kb memory)
- Use Arduino IDE to set up:
  - Open `File→Preferences`
  - Go to “Additional Boards Manager URL”
  - `https://raw.githubusercontent.com/damellis/attiny/ide-1.6.x-boards-manager/package_damellis_attiny_index.json`
  - Click OK
  - Open `Tools→Board→Boards Manager`
  - Scroll to the bottom to “attiny”
  - Click once, Install button will appear, click it
  - Should see “INSTALLED” next to “attiny” when complete
  - Should now have two ATtiny options in `Tools→Board` menu:
    - ATtiny 25/45/85
    - ATtiny 24/44/84

# Set up Arduino as ATtiny programmer

- Load sketch `File→Examples→11.ArduinoISP→ArduinoISP`
- Upload to Arduino.
- Go to `Tools→Board` and select "ATtiny 25/45/85".
- Go to `Tools→Processor` and select "ATtiny 45".
- Go to `Tools→Clock` and select "Internal 1MHz".
- Go to `Tools→Programmer` and select "Arduino as ISP".

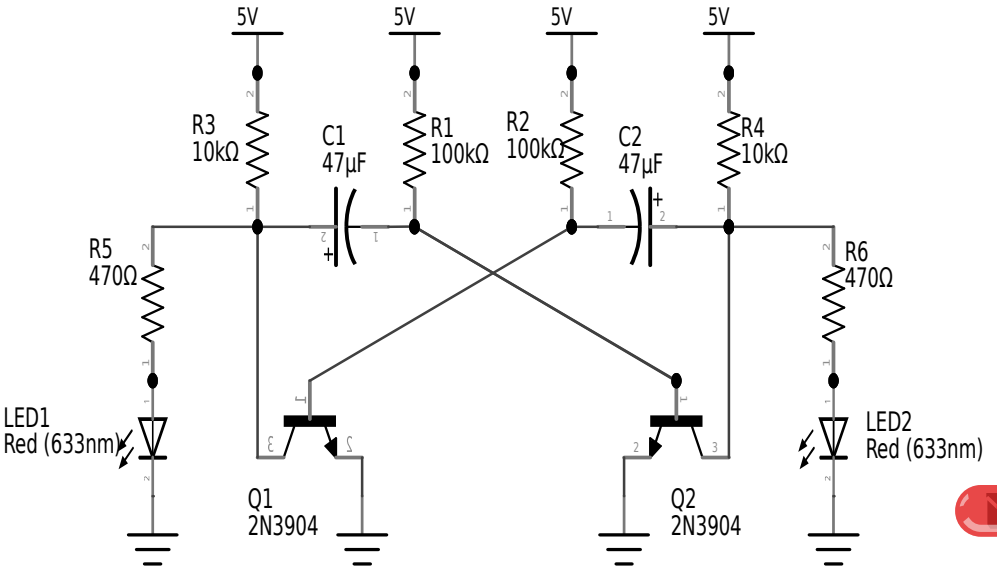
# Connect Attiny to Arduino, upload sketch

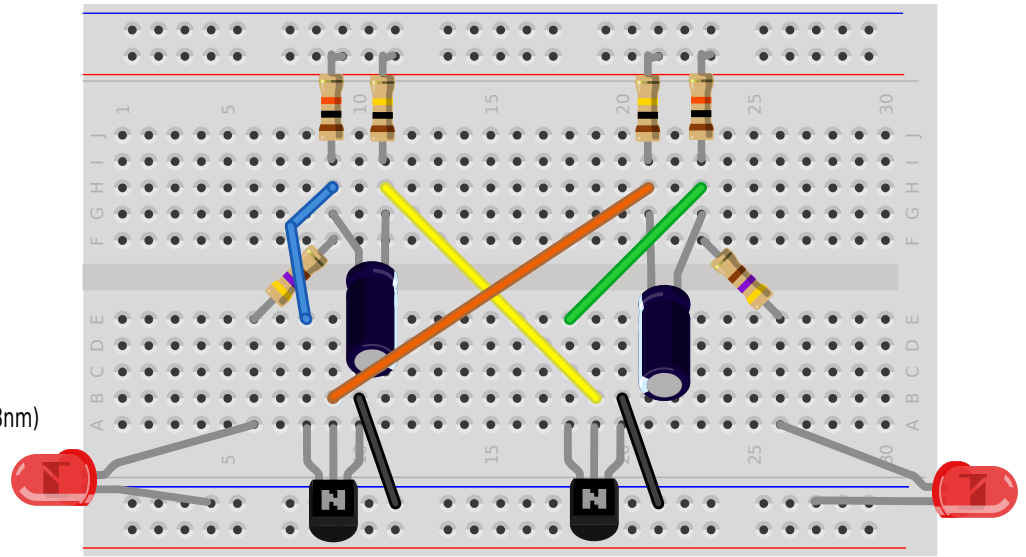
- Wire up ATtiny to Arduino with jumpers (see below).
- If Nano, put 47uF capacitor between RST (+) and GND (-).
- Wire an LED and resistor to pin 5 (logical pin 0).
- Load the Blink sketch from Examples (or from GitHub).
- Modify to use pin 0 instead of 13.
- Upload, LED blinks.

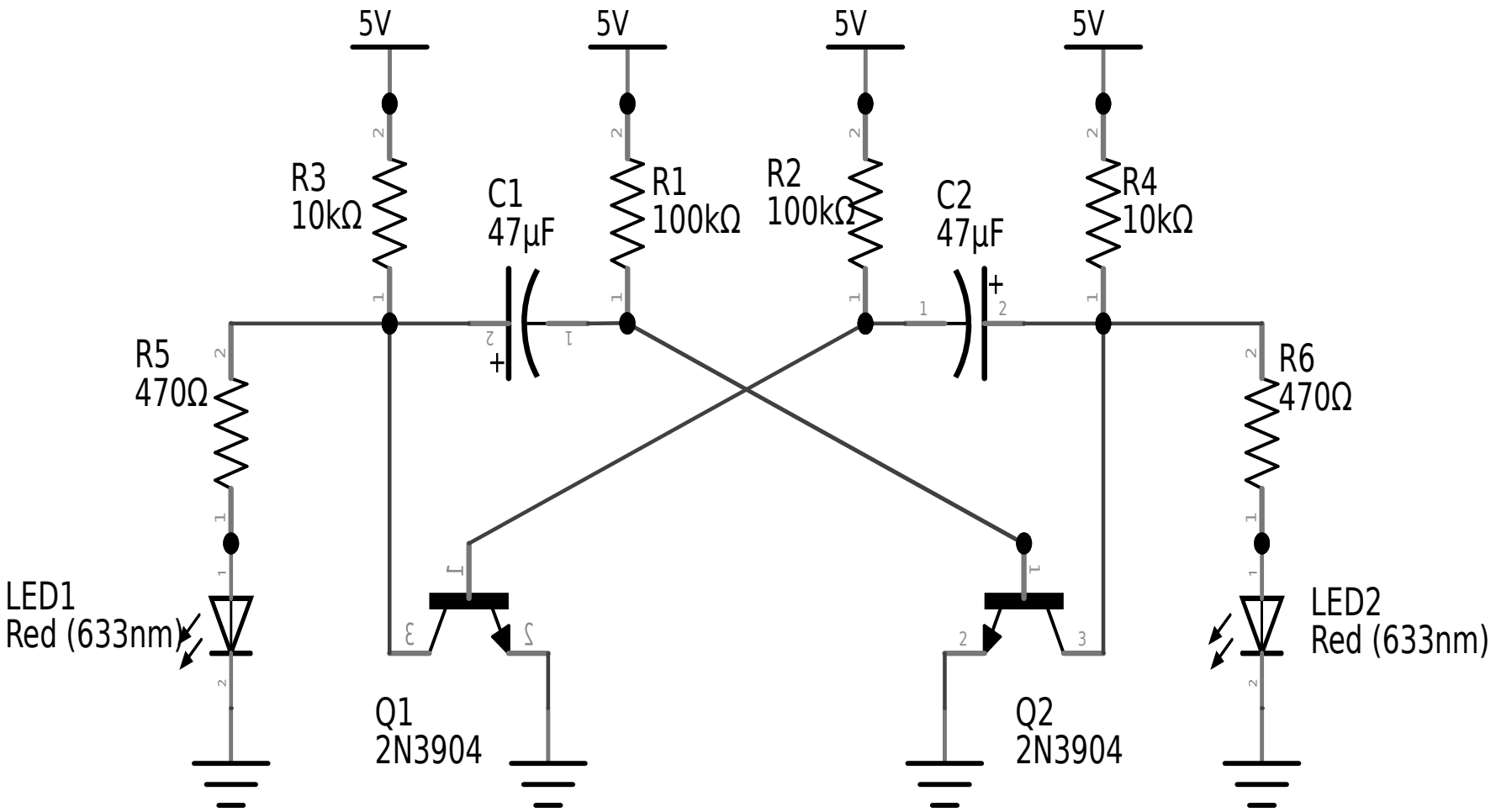
| Pin name | Attiny pin | Nano | Uno |
|---|---|---|---|
| RST | 1 | D10 | 10 |
| GND | 4 | GND | GND |
| MOSI | 5 | D11 | 11 |
| MISO | 6 | D12 | 12 |
| SCK | 7 | D13 | 13 |
| VCC | 8 | 5V | 5V |

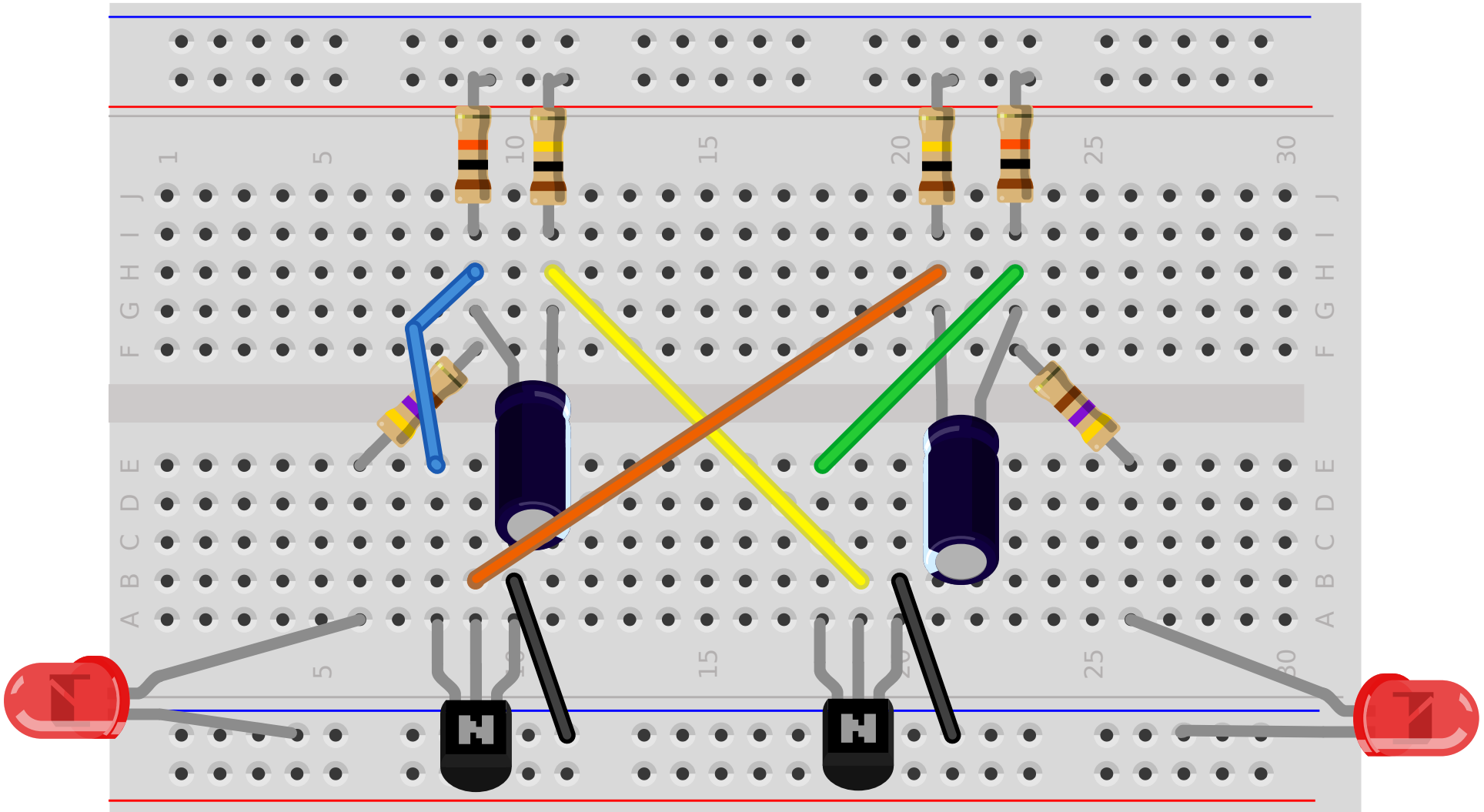
# Transistor Astable Multivibrator

- Similar to the 555 astable multivibrator but with super simple transistor
- Timing similar to 555: R1*C1 and R2*C2
- Note: transistors as switches – useful for Raspberry Pi pins, too



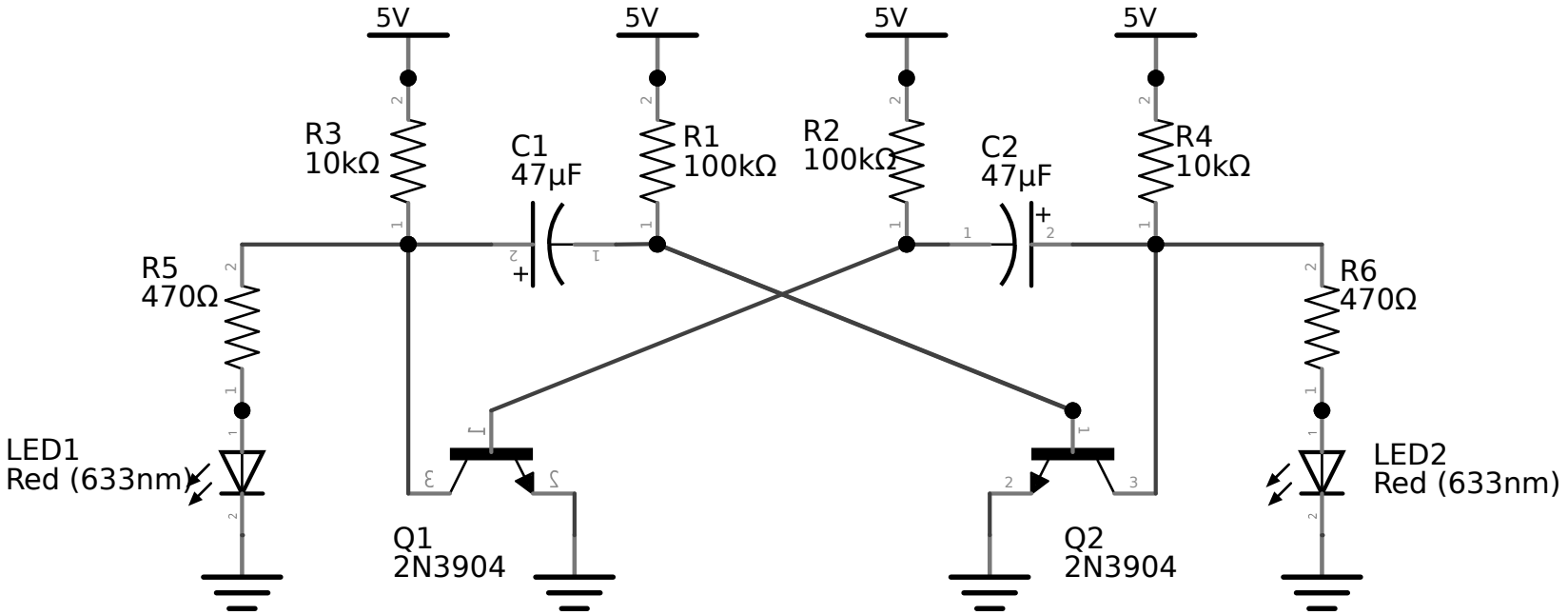




5V
5V
5V
5V
R3
10kΩ
C1
47µF
R1
100kΩ
R2
100kΩ
C2
47µF
R4
10kΩ
R5
470Ω
R6
470Ω
LED1
Red (633nm)
LED2
Red (633nm)
Q1
2N3904
Q2
2N3904


1
5
10
15
20
25
30
A B C D E
F G H I J

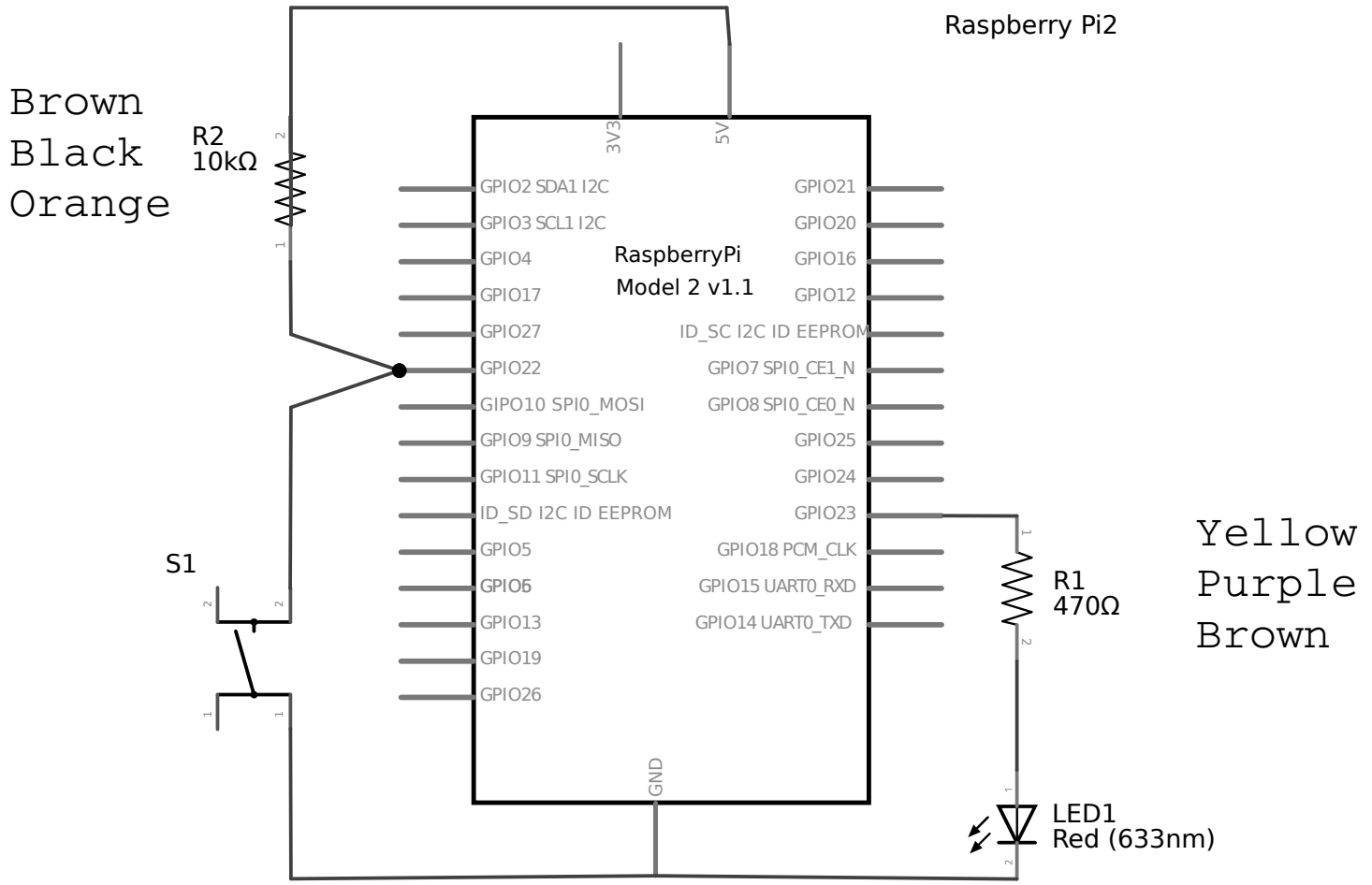
# How It Works (briefly)

- Transistors Q1 and Q2 are switches (rather than amplifiers)
- Timing determined by R1*C1 and R2*C2
- If C1 discharged, C2 charged, then Q1 on, Q2 off
- C1 charges until it turns on Q2
- Voltage on C2 drops, which turns off Q1



# Pi as Intermediary

- Pushbutton press turns LED on, release turns LED off



# RpiReadWrite.py (on GitHub)

```
import RPi.GPIO as GPIO
import time

BUTTON = 22
LED = 23

GPIO.setmode(GPIO.BCM)
GPIO.setup(BUTTON, GPIO.IN)
GPIO.setup(LED, GPIO.OUT)

try:
    while True:
        inputValue = GPIO.input(BUTTON)
        if (GPIO.LOW == inputValue):
            GPIO.output(LED, GPIO.HIGH)
        else:
            GPIO.output(LED, GPIO.LOW)
        time.sleep(2)
except KeyboardInterrupt:
    GPIO.cleanup()
```

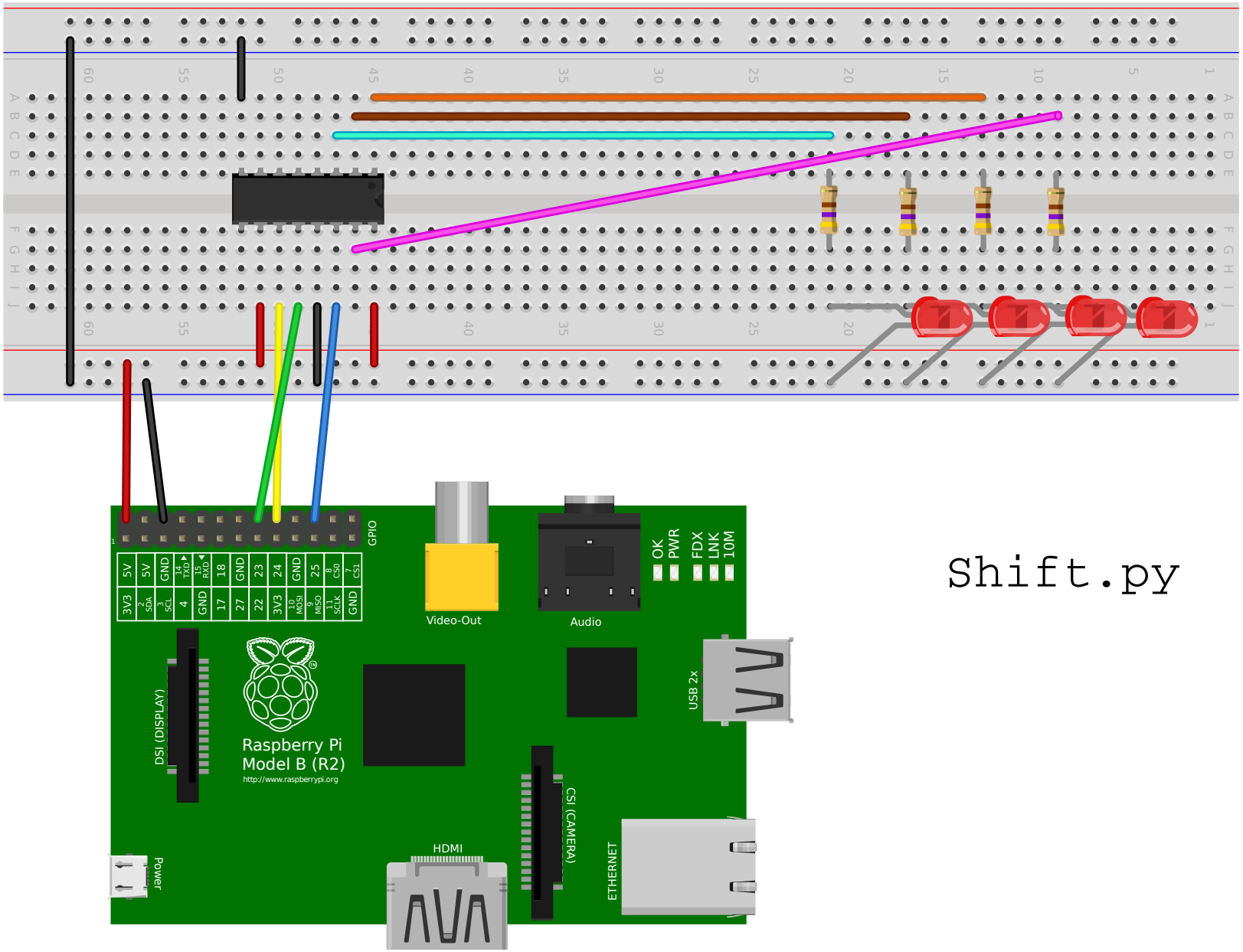
# Shift Registers

- A way to control N outputs using fewer than N pins on a Raspberry Pi
- Push bits one at a time from Pi to shift register (n = 1, 2, …N)
- On each step, bit n gets shifted to bit n+1 and new bit goes into bit 0
- Final step is to set all N outputs at once
- Can chain together multiple shift registers to control more outputs

# Push `0101` to Shift Register

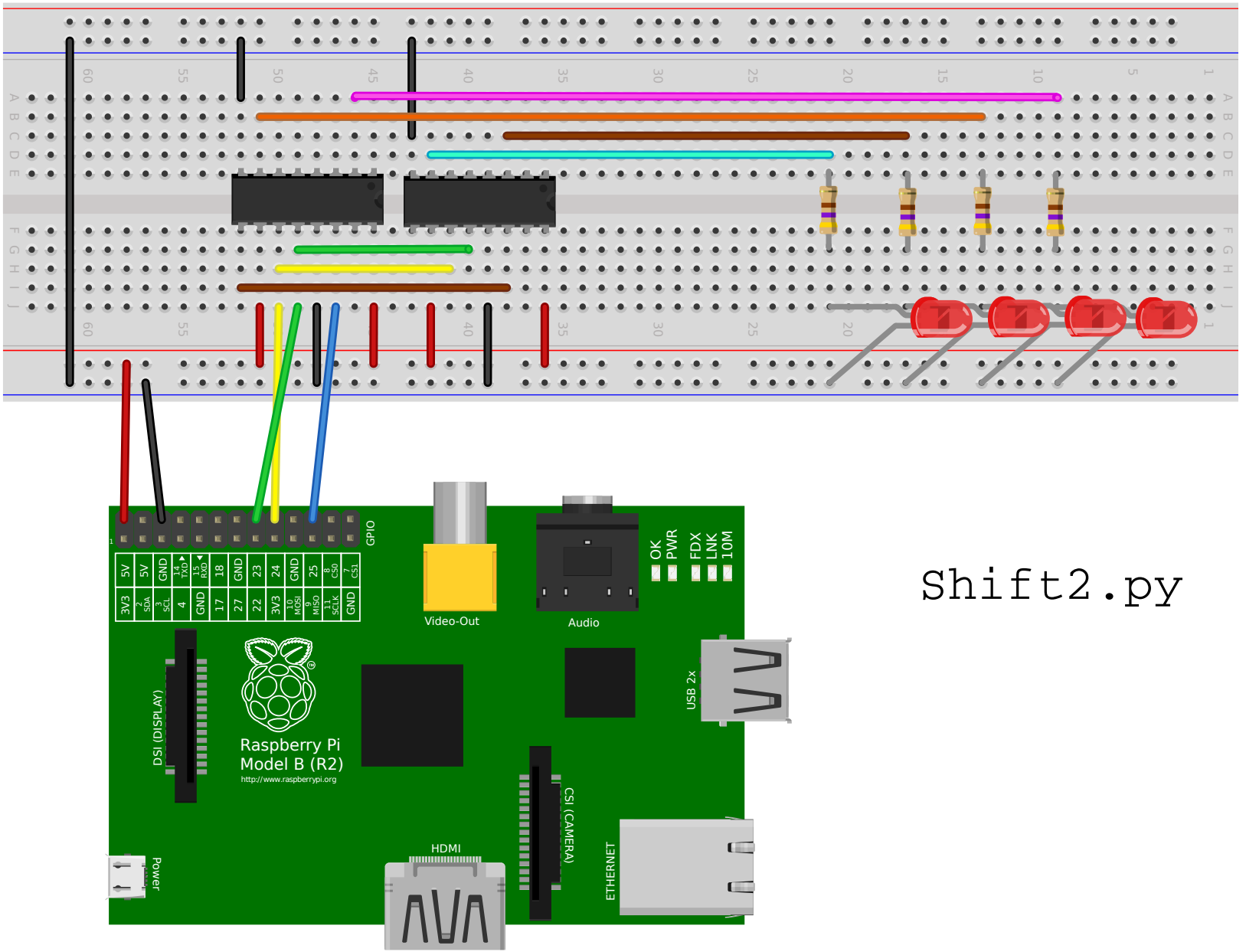
| Output | Shift Register | Pi | Input |
|---|---|---|---|
| xxxx | xxxx | x | 0101 |
| xxxx | xxxx | 0 | 101 |
| xxxx | xxx0 | 1 | 01 |
| xxxx | xx01 | 0 | 1 |
| xxxx | x010 | 1 | |
| xxxx | 0101 | | |
| 0101 | 0101 | | |

# Shift Register Circuit




`Shift.py`

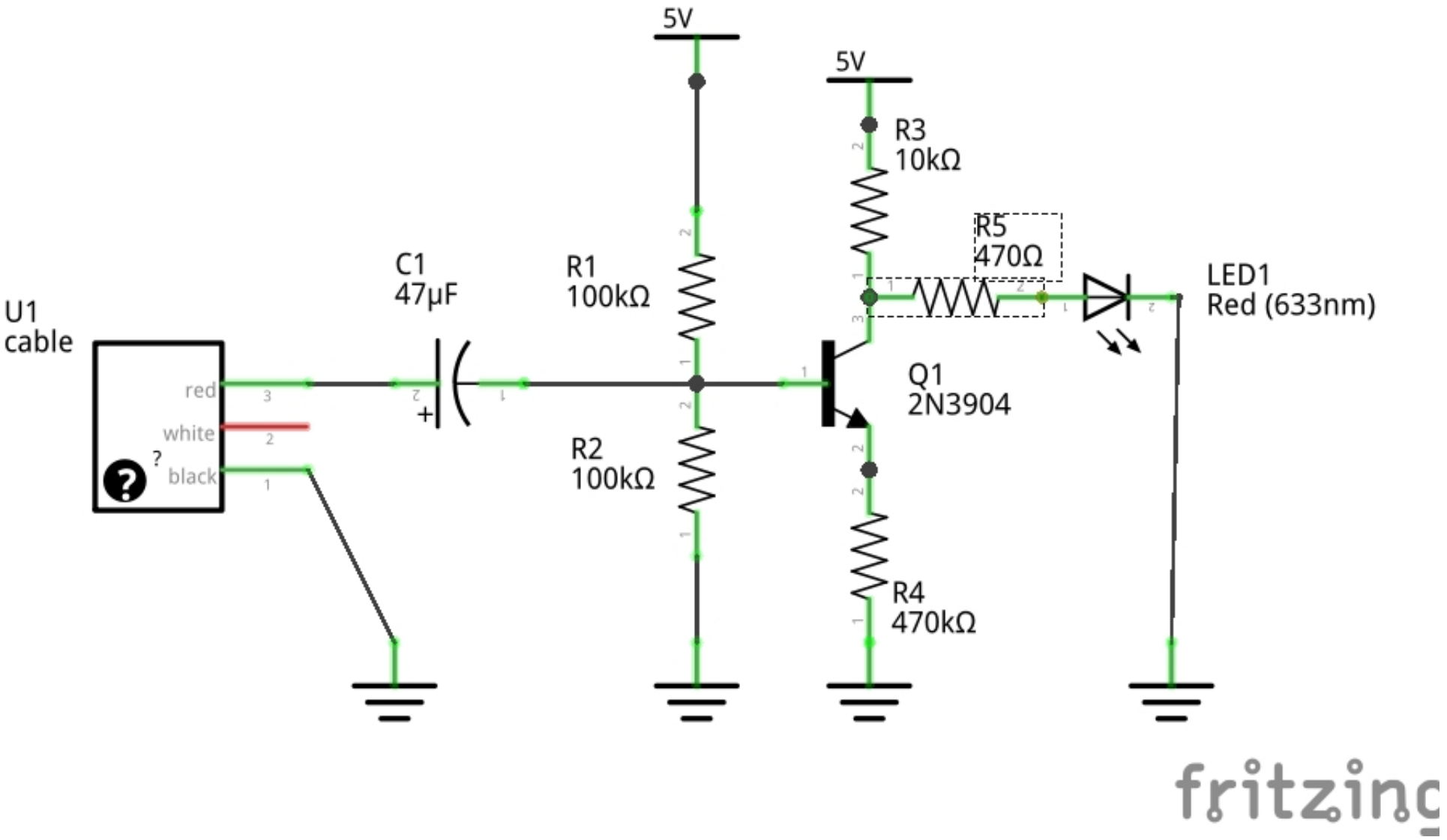
# Dual Shift Register Circuit



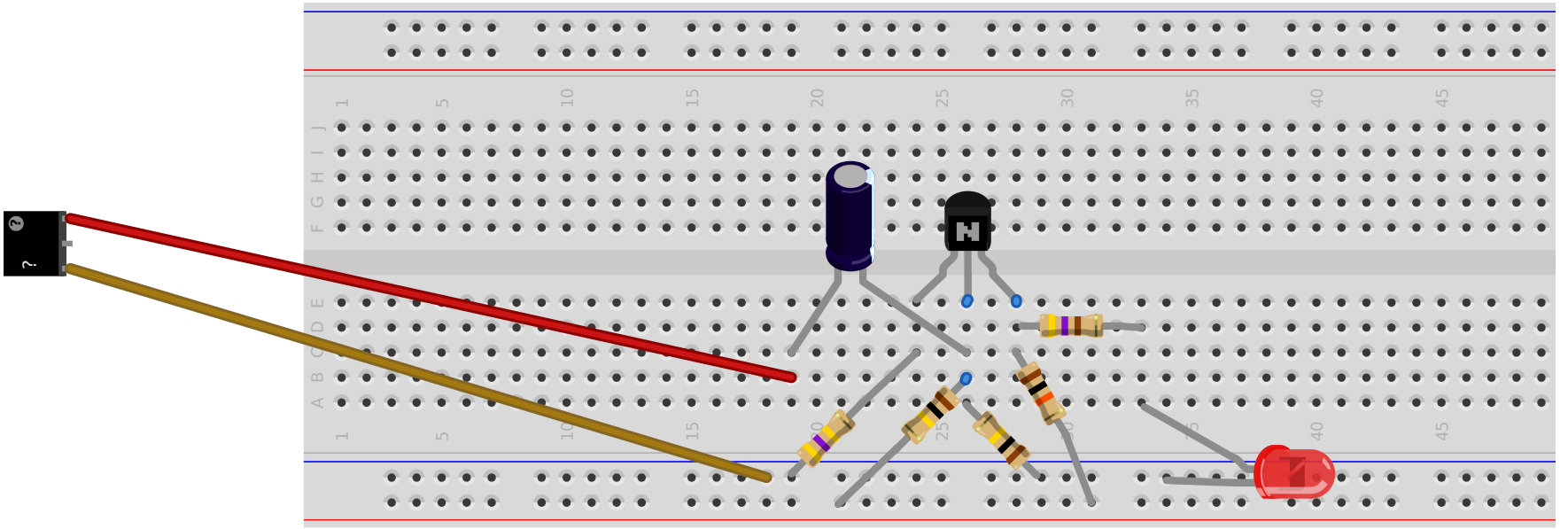

`Shift2.py`

# RPi Audio as Function Generator

- An audio signal (headphones, speaker output) is just an AC electrical output.
- Playing a “sound” file that is just a single frequency will generate a sine wave on the output.
- Line level output  is a few hundred millivolts.
- We will amplify using the 2N3904 transistor.




R1, R2: voltage divider for bias
R3/R4 = gain

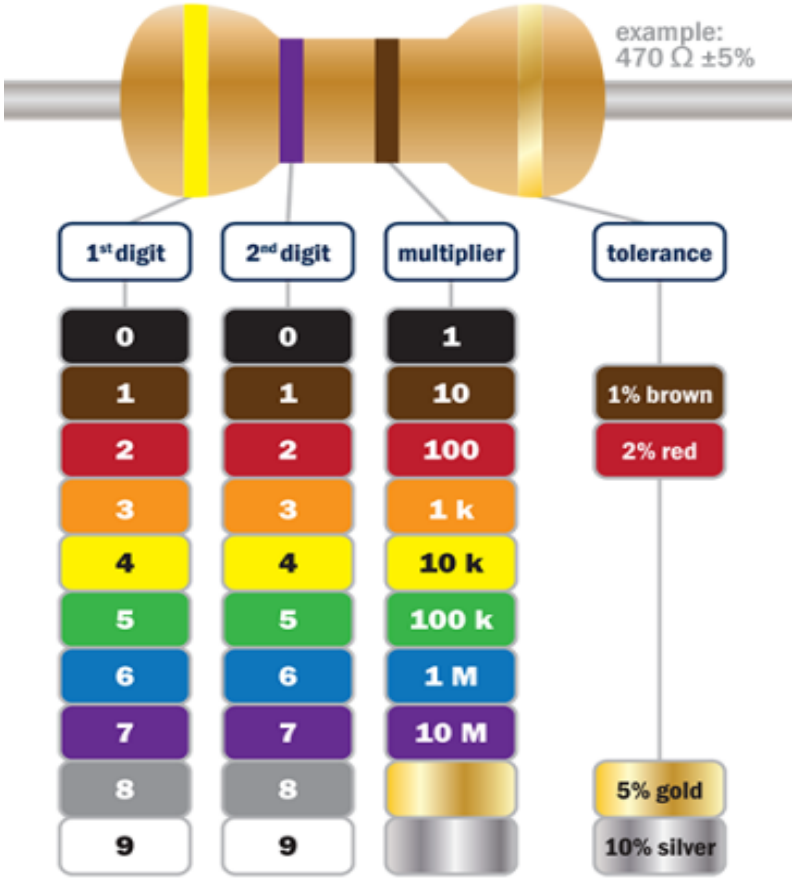


Play “sound” files:

```
$ sudo apt-get install omxplayer
$ omxplayer -o local 1Hz.mp3
$ omxplayer -o local 1000Hz.mp3
$ omxplayer -o local 1001Hz.mp3
$ omxplayer -o local mix.mp3
```

Mix.mp3 is the 1000Hz plus the 1001Hz. Why the same as 1Hz?

# Reference



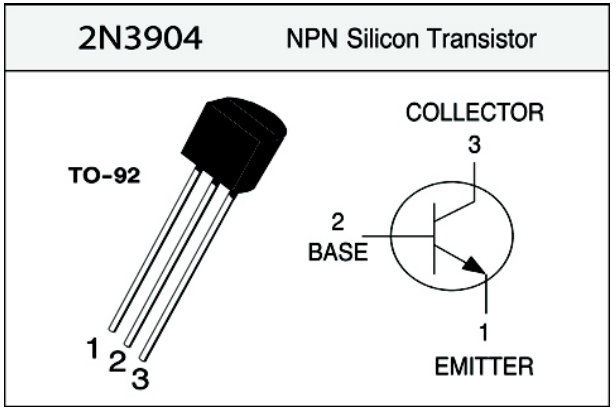

## Raspberry Pi 3 GPIO Header

| Pin# | NAME | NAME | Pin# |
|---|---|---|---|
| 01 | 3.3v DC Power | DC Power 5v | 02 |
| 03 | GPIO02 (SDA1 , I²C) | DC Power 5v | 04 |
| 05 | GPIO03 (SCL1 , I²C) | Ground | 06 |
| 07 | GPIO04 (GPIO_GCLK) | (TXD0) GPIO14 | 08 |
| 09 | Ground | (RXD0) GPIO15 | 10 |
| 11 | GPIO17 (GPIO_GEN0) | (GPIO_GEN1) GPIO18 | 12 |
| 13 | GPIO27 (GPIO_GEN2) | Ground | 14 |
| 15 | GPIO22 (GPIO_GEN3) | (GPIO_GEN4) GPIO23 | 16 |
| 17 | 3.3v DC Power | (GPIO_GEN5) GPIO24 | 18 |
| 19 | GPIO10 (SPI_MOSI) | Ground | 20 |
| 21 | GPIO09 (SPI_MISO) | (GPIO_GEN6) GPIO25 | 22 |
| 23 | GPIO11 (SPI_CLK) | (SPI_CE0_N) GPIO08 | 24 |
| 25 | Ground | (SPI_CE1_N) GPIO07 | 26 |
| 27 | ID_SD (I²C ID EEPROM) | (I²C ID EEPROM) ID_SC | 28 |
| 29 | GPIO05 | Ground | 30 |
| 31 | GPIO06 | GPIO12 | 32 |
| 33 | GPIO13 | Ground | 34 |
| 35 | GPIO19 | GPIO16 | 36 |
| 37 | GPIO26 | GPIO20 | 38 |
| 39 | Ground | GPIO21 | 40 |

Rev. 2
29/02/2016

www.element14.com/RaspberryPi

# Components


2N3904
NPN Silicon Transistor
TO-92
1 2 3
COLLECTOR
3
2
BASE
1
EMITTER