

Intro to the Raspberry Pi (RPi) & The Internet of Things (IoT)

Class 3 – 2018 / 03 / 25

Linux Recap Cheat Sheet

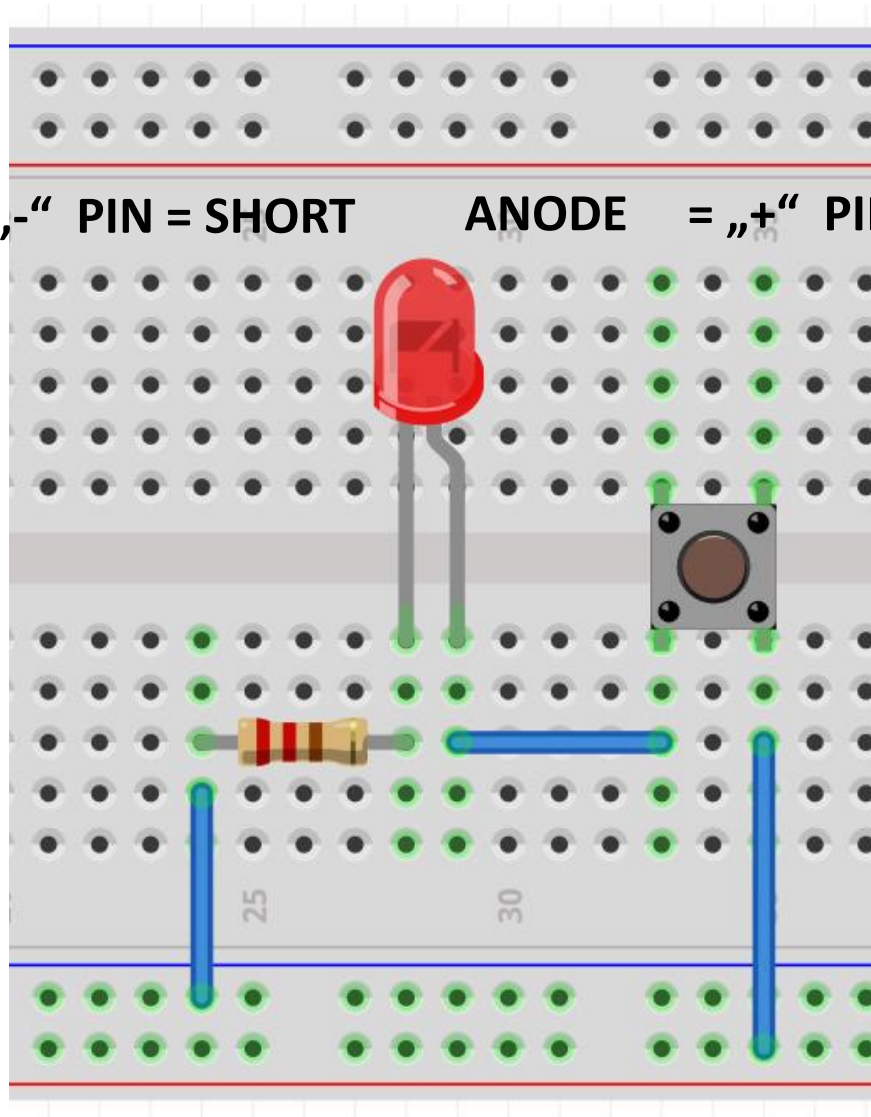
<https://www.michael-wessel.info/rpi/fwunixref.pdf>

There:

- **„File Commands“**
- **and some „System Info“**
- **There is more!**

Last Week – LED Circuit

CATHODE = „-“ PIN = SHORT ANODE = „+“ PIN = LONG



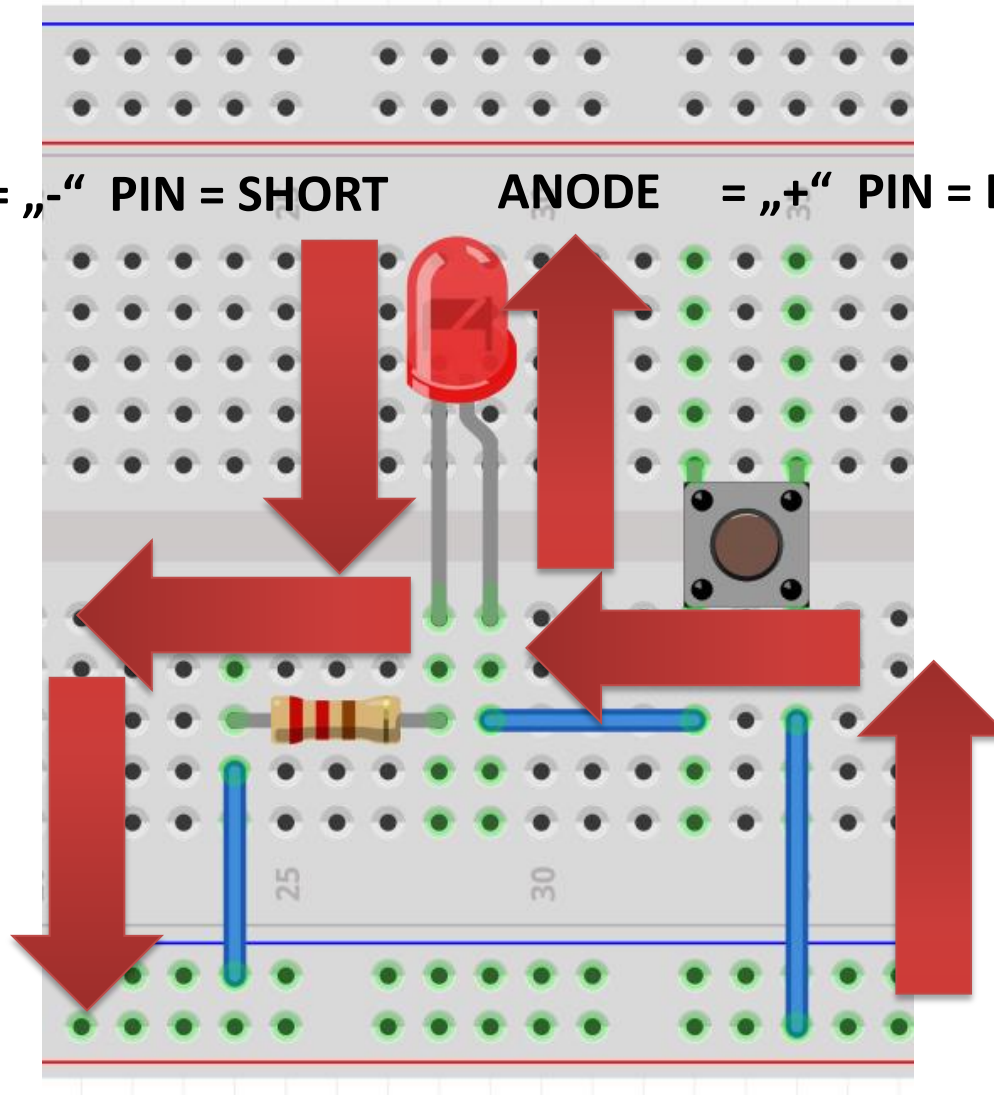
MINUS = BLUE = GND
PLUS = RED = 3.3 Volt



Flow of Electric Current from „+“ -> „-“

CATHODE = „-“ PIN = SHORT

ANODE = „+“ PIN = LONG



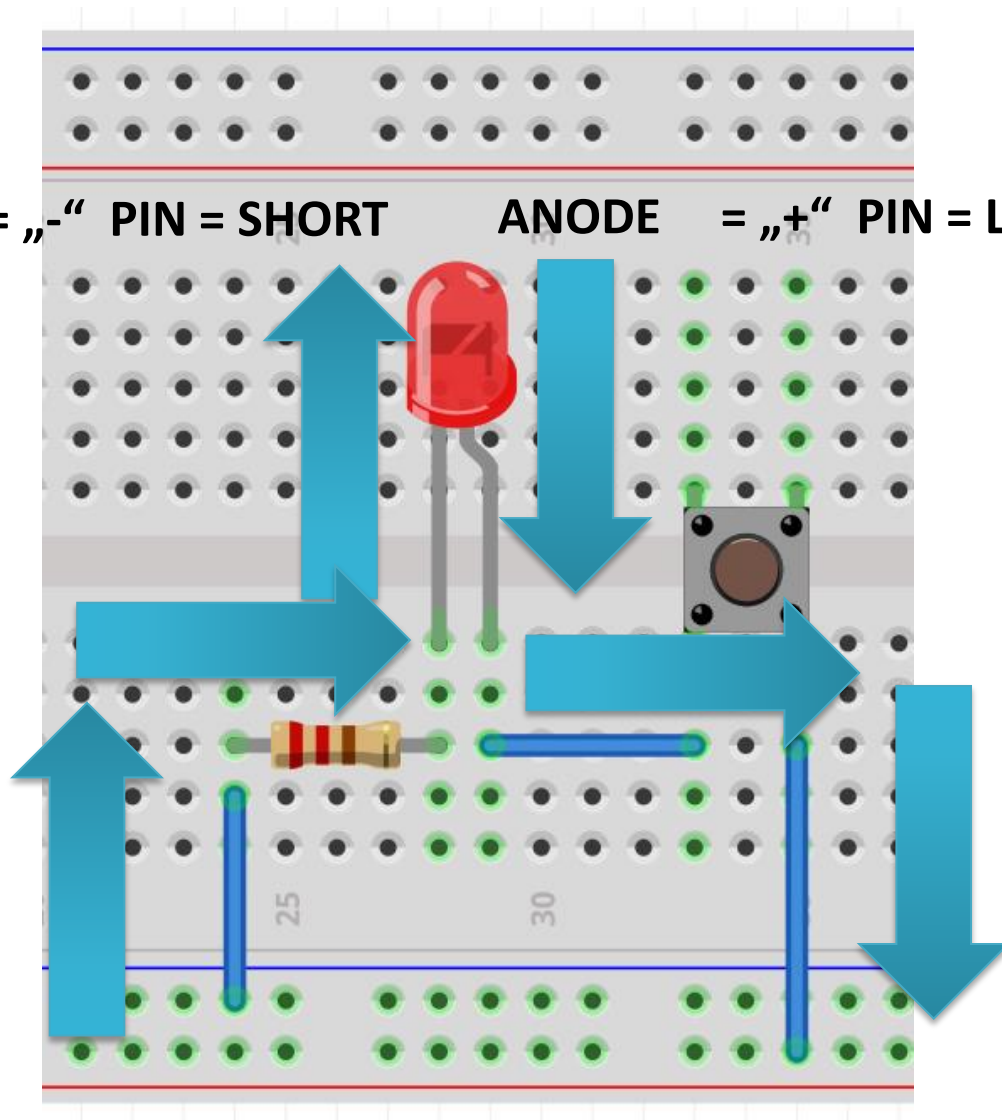
MINUS = BLUE = GND

PLUS = RED = 3.3 Volt



Flow of Electrons is Reversed!

CATHODE = „-“ PIN = SHORT ANODE = „+“ PIN = LONG

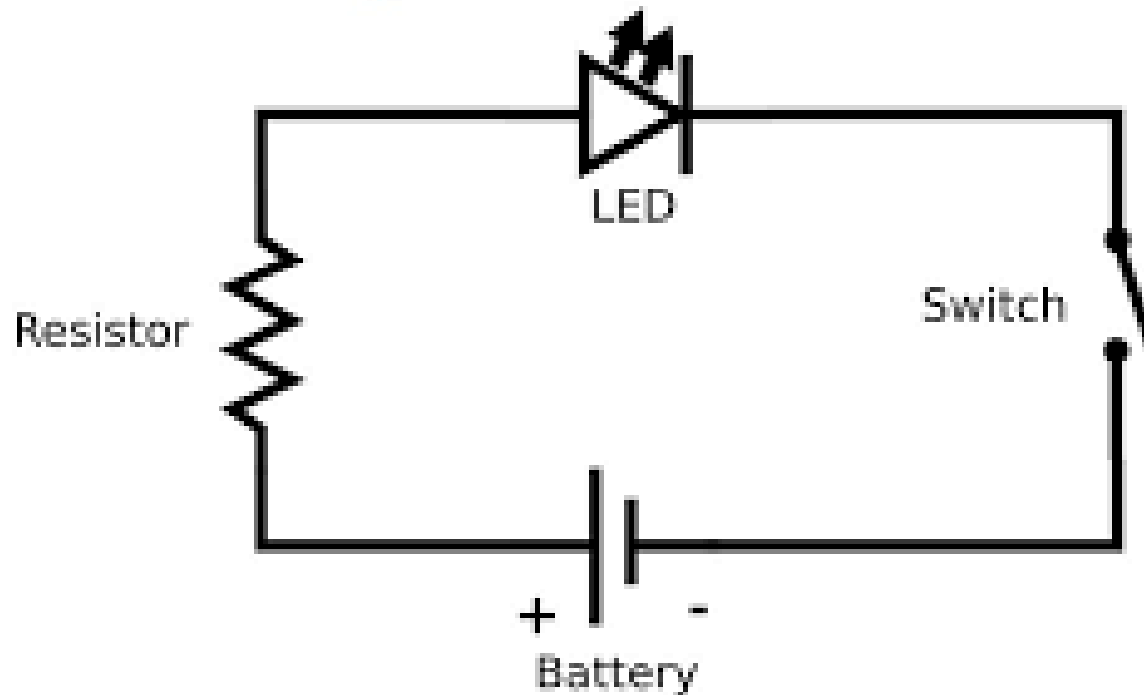


MINUS = BLUE = GND
PLUS = RED = 3.3 Volt

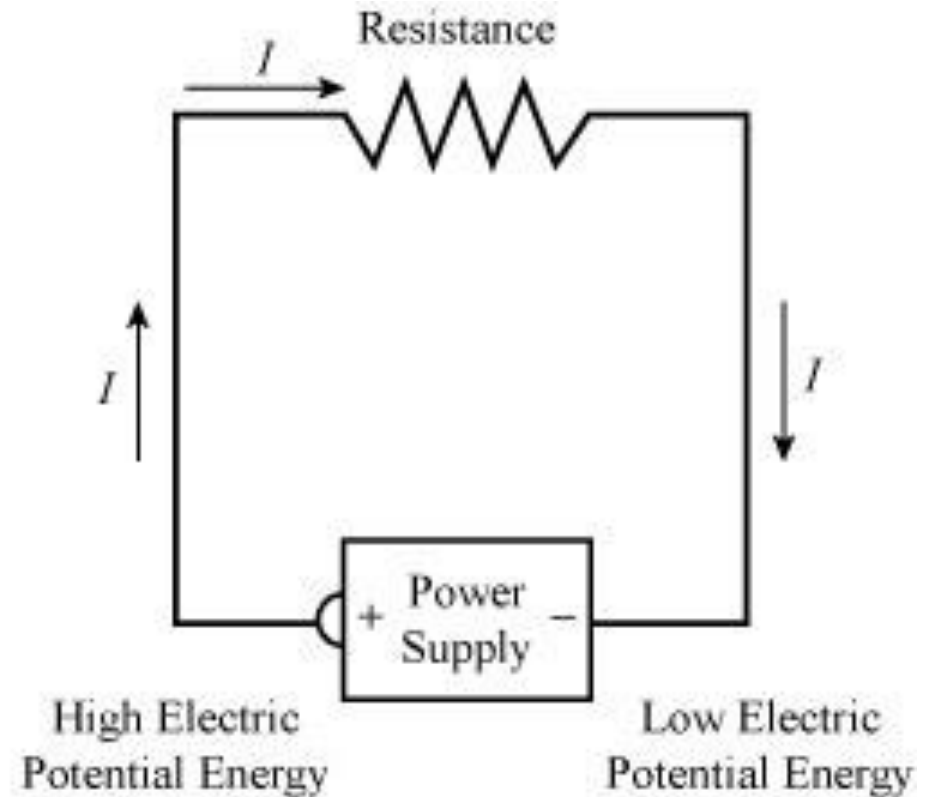
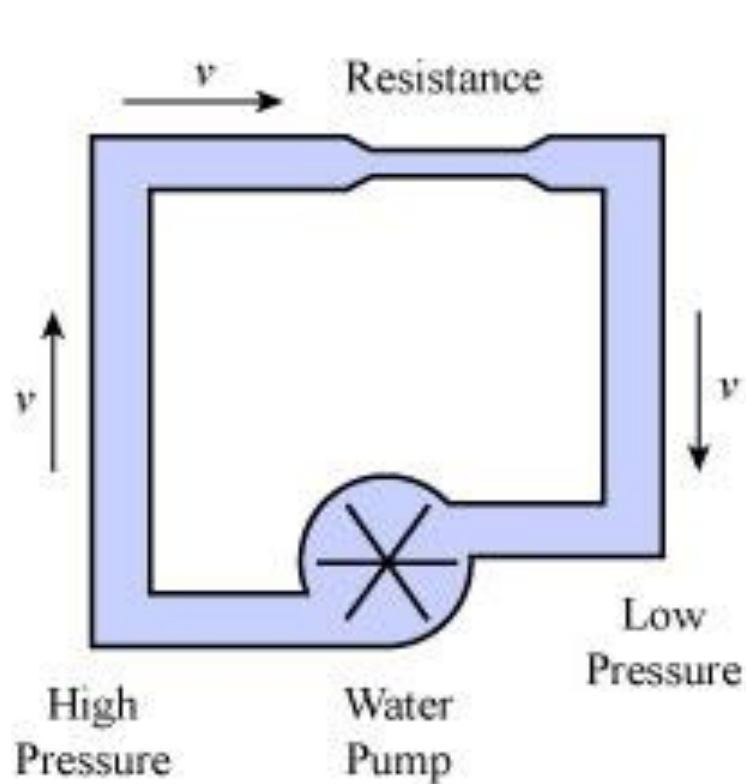


What's different in ours?
Does it matter?

Circuit Diagram



Water Analogy for Resistor – Current Reducer



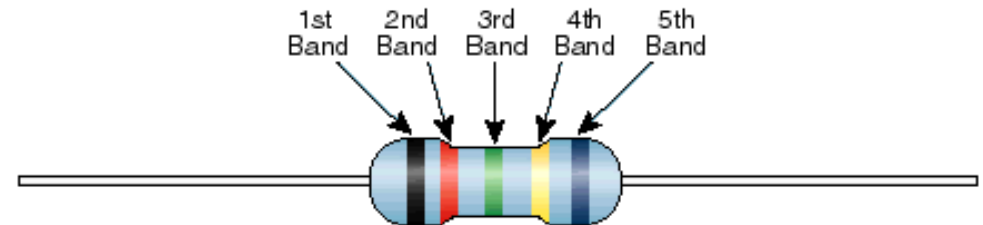
The Resistor is Important!

- Without resistor, the LED will be destroyed immediately



- Resistors are measured in „Ohms“ = Ω Greek Letter
- Have a „color code“ to indicate how many Ohms
- The more Ohms, the higher the resistance, the less current gets through
- LEDs require 220– 440 Ω

Standard EIA Color Code Table 5 Band: $\pm 1\%$, $\pm 25\%$, $\pm 5\%$, $\pm 1\%$

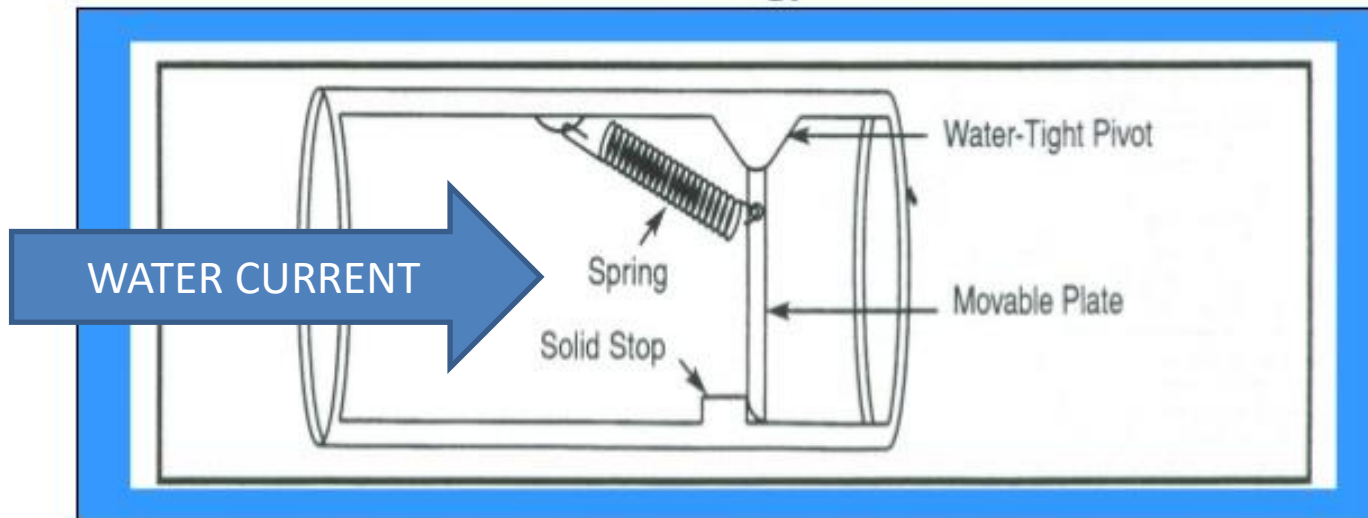


Color	1st Band (1st figure)	2nd Band (2nd figure)	3rd Band (3rd figure)	4th Band (multiplier)	5th Band (tolerance)
Black	0	0	0	10^0	
Brown	1	1	1	10^1	$\pm 1\%$
Red	2	2	2	10^2	
Orange	3	3	3	10^3	
Yellow	4	4	4	10^4	
Green	5	5	5	10^5	$\pm 5\%$
Blue	6	6	6	10^6	$\pm 25\%$
Violet	7	7	7	10^7	$\pm 1\%$
Gray	8	8	8	10^8	
White	9	9	9	10^9	
Gold				10^{-1}	

Chart Provided By 

Water Analogy for Diode – One Way Street / Valve for Electricity

Water Analogy of Diodes



- When water pressure on left overcomes the restoring force of spring, the gate is opened and water is allowed to flow →.
- When water pressure is from right to left, the gate is pressed against the solid stop and no water is allowed to flow.
- Spring restoring force is analogous to 0.6V needed to forward bias a Si diode.

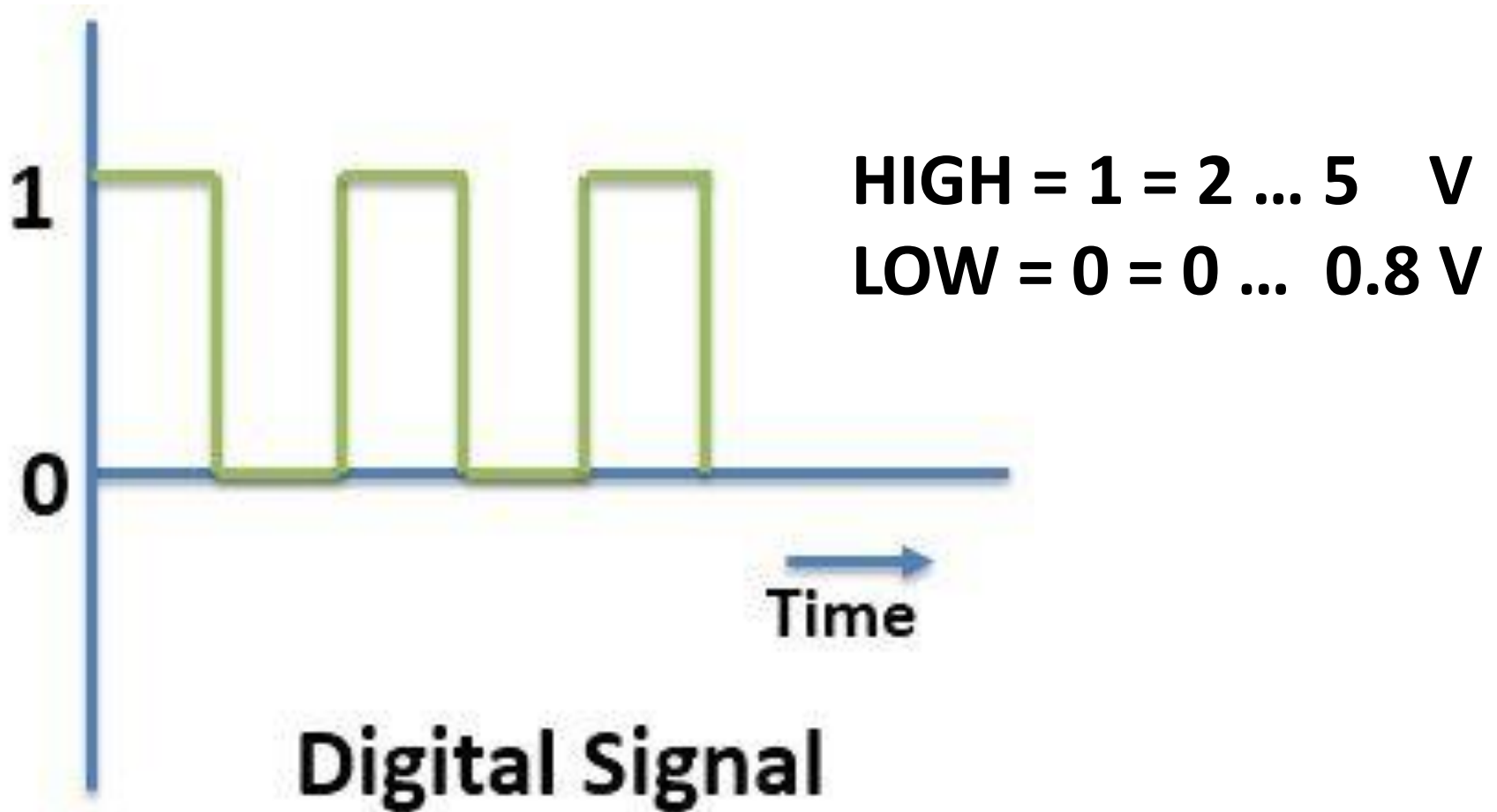
Experiment... What happens when we use 5 V instead of 3.3 V ?



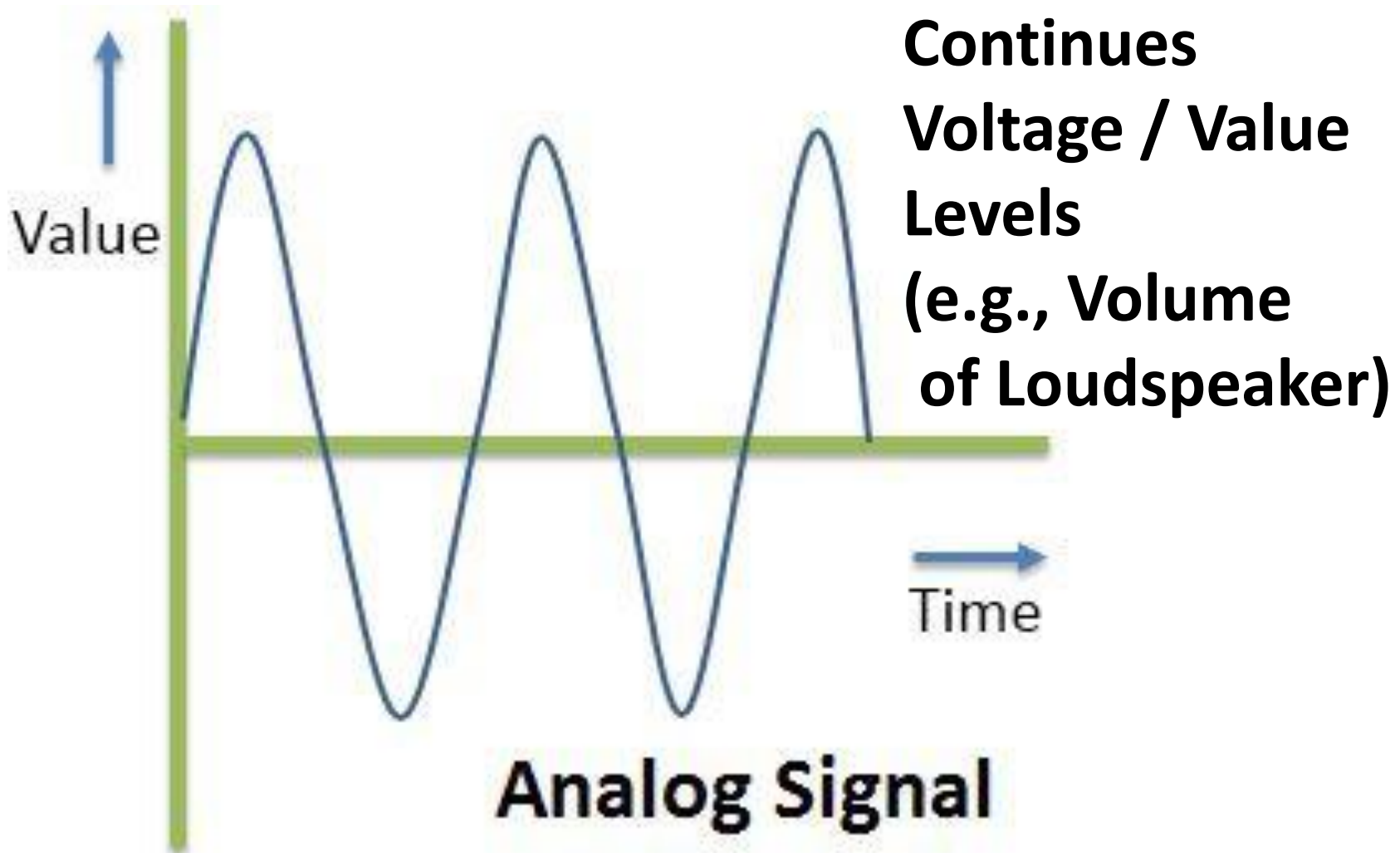
Answer

- Nothing!
 - There is no change in brightness of the LED
 - Unlike a light bulb
 - LEDs cannot be „dimmed“ liked that
 - But there is a way...
We will learn later how to dimm LEDs...
- LEDs are very much „binary“ = on / off = 1 / 0 = high / low devices!
 - A traditional lightbulb is more „analog“
Here, Brightness can be changed by changing Voltage / Current
 - Not with LEDs (or only very slightly)

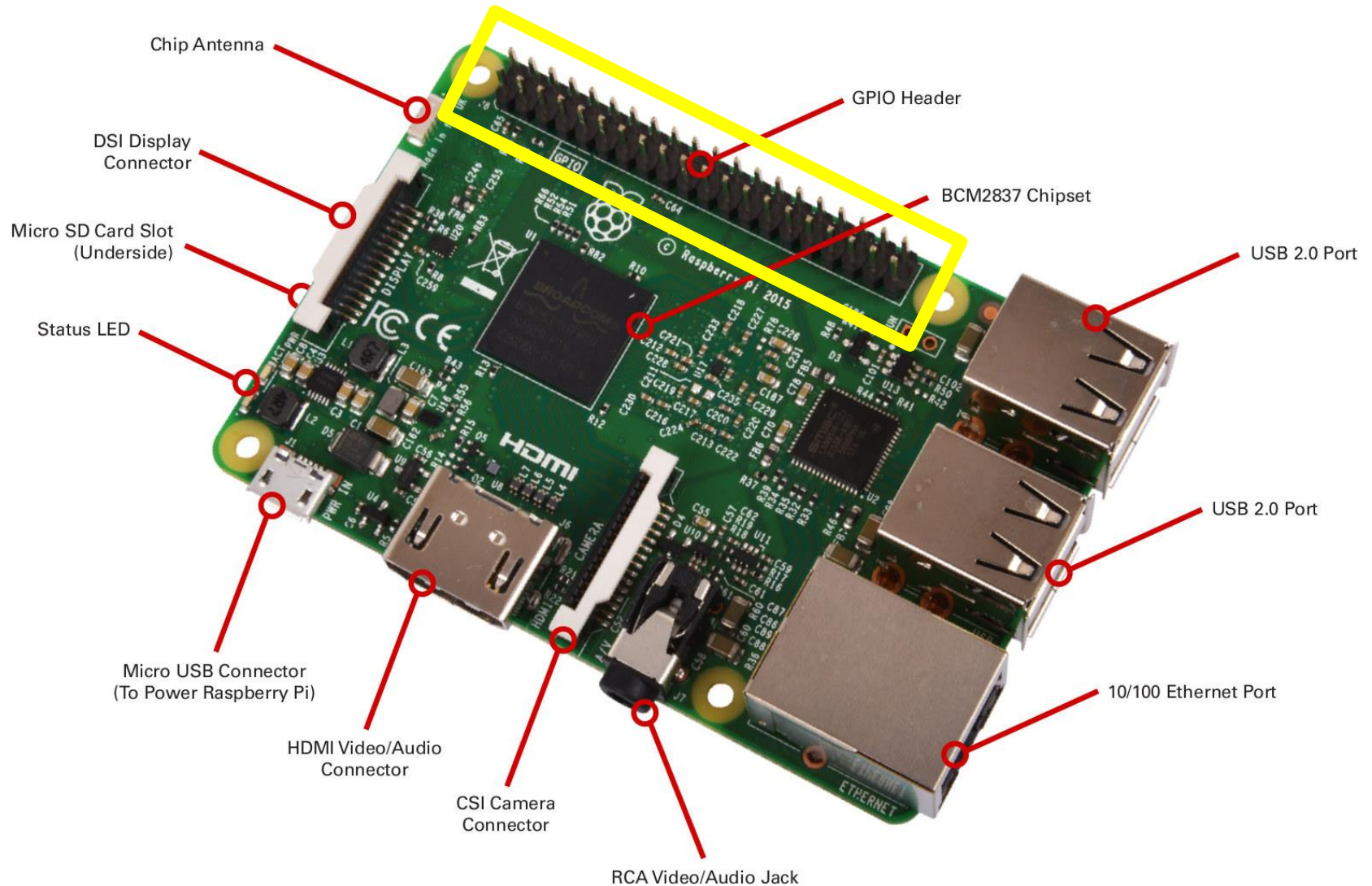
Digital Signals - „Binary“ 1, 0
On / Off, Yes / No, HI(GH) / LO(W), ...



Analog Signal



General Purpose Input Output Ports (GPIO)

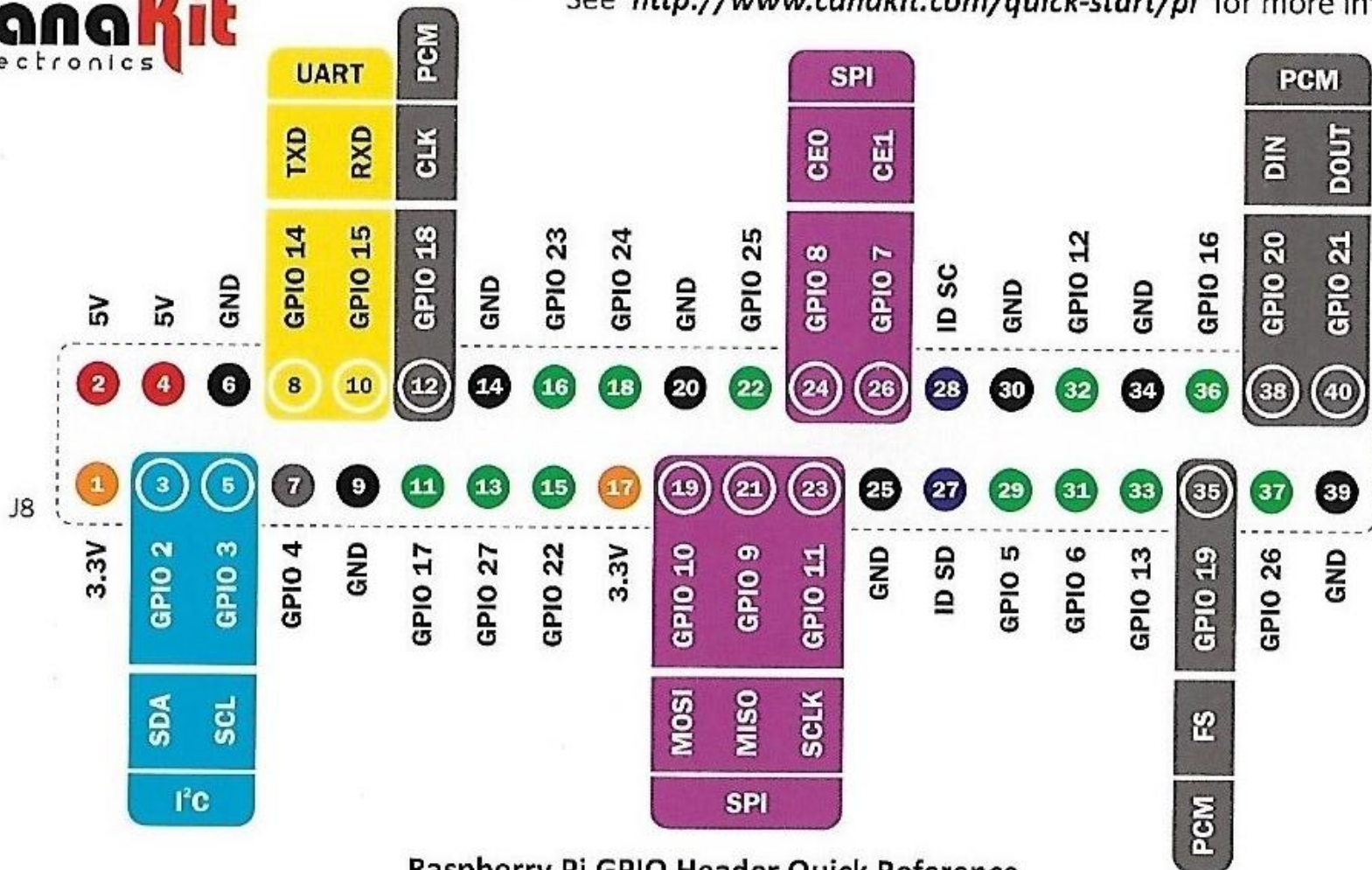


Functions of GPIO Pins

Also see <https://pinout.xyz/#>



See <http://www.canakit.com/quick-start/pi> for more information



Raspberry Pi GPIO Header Quick Reference

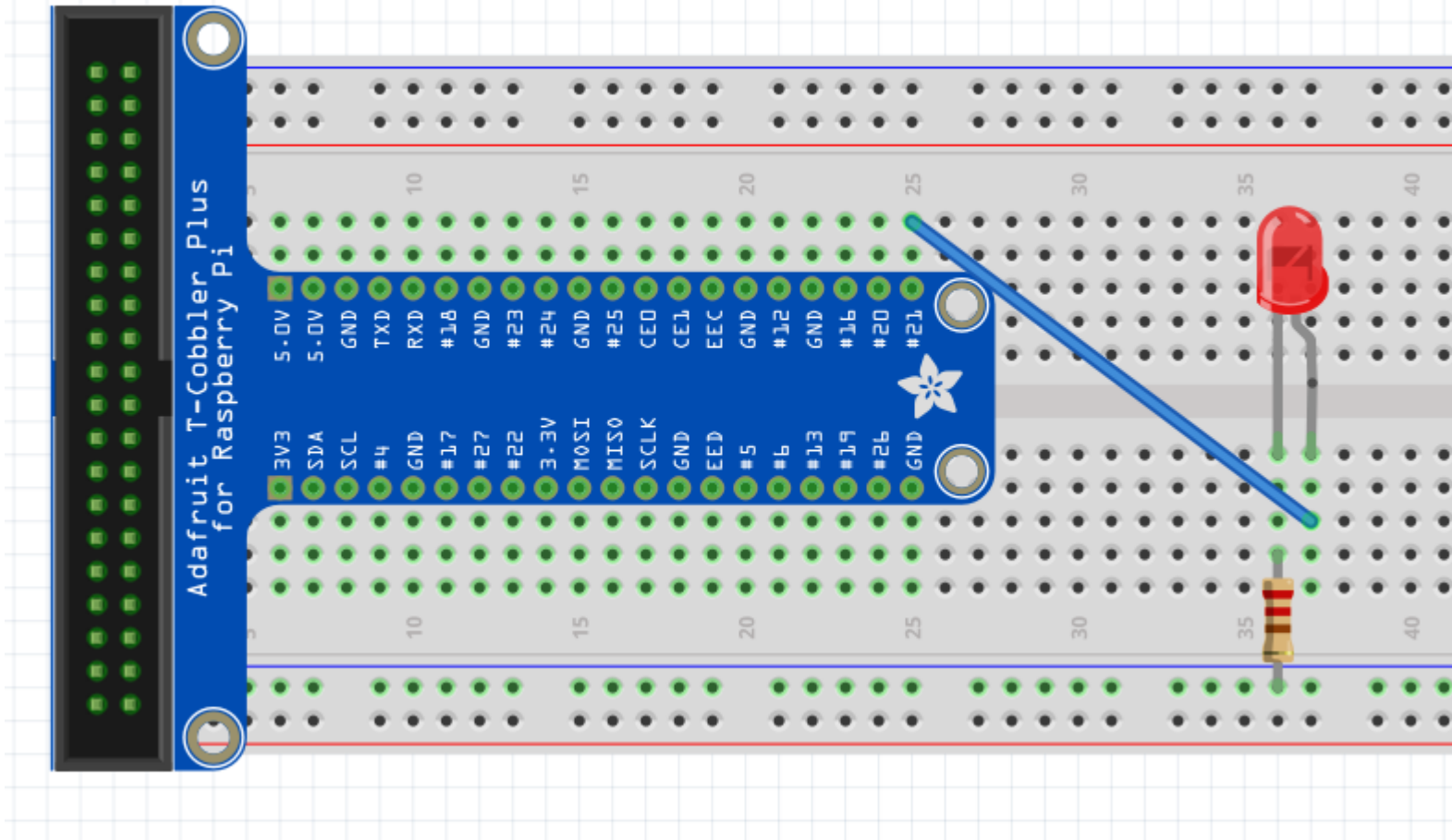
Key Features

- **GPIO Port = „General Purpose Input Output“ Port**
 - 40 pin „header“
 - Digital inputs and outputs (direction per pin configurable)
 - PWM = „Pulse Width Modulation“
 - SPI, I2C: Industry Standards Serial Protocols
 - Serial Peripheral Interface (SPI) – for Sensors, ...
 - 3 or 4 wires
 - Multiple devices with „select device“ signal
 - One master only
 - Full duplex
 - Longer ranges
 - I2C = I Squared C = I Two C = IIC = I²C (Inter-Integrated Circuit)
 - 2 wires only
 - Multiple devices on same bus, with chip addressing
 - Multiple masters possible
 - Cheaper
 - Half duplex
 - I2C more flexible, SPI faster
 - GPIOs need to be treated with care though; can be damaged by static electricity!
- **However, no ANALOG inputs / outputs (unlike Arduino)**

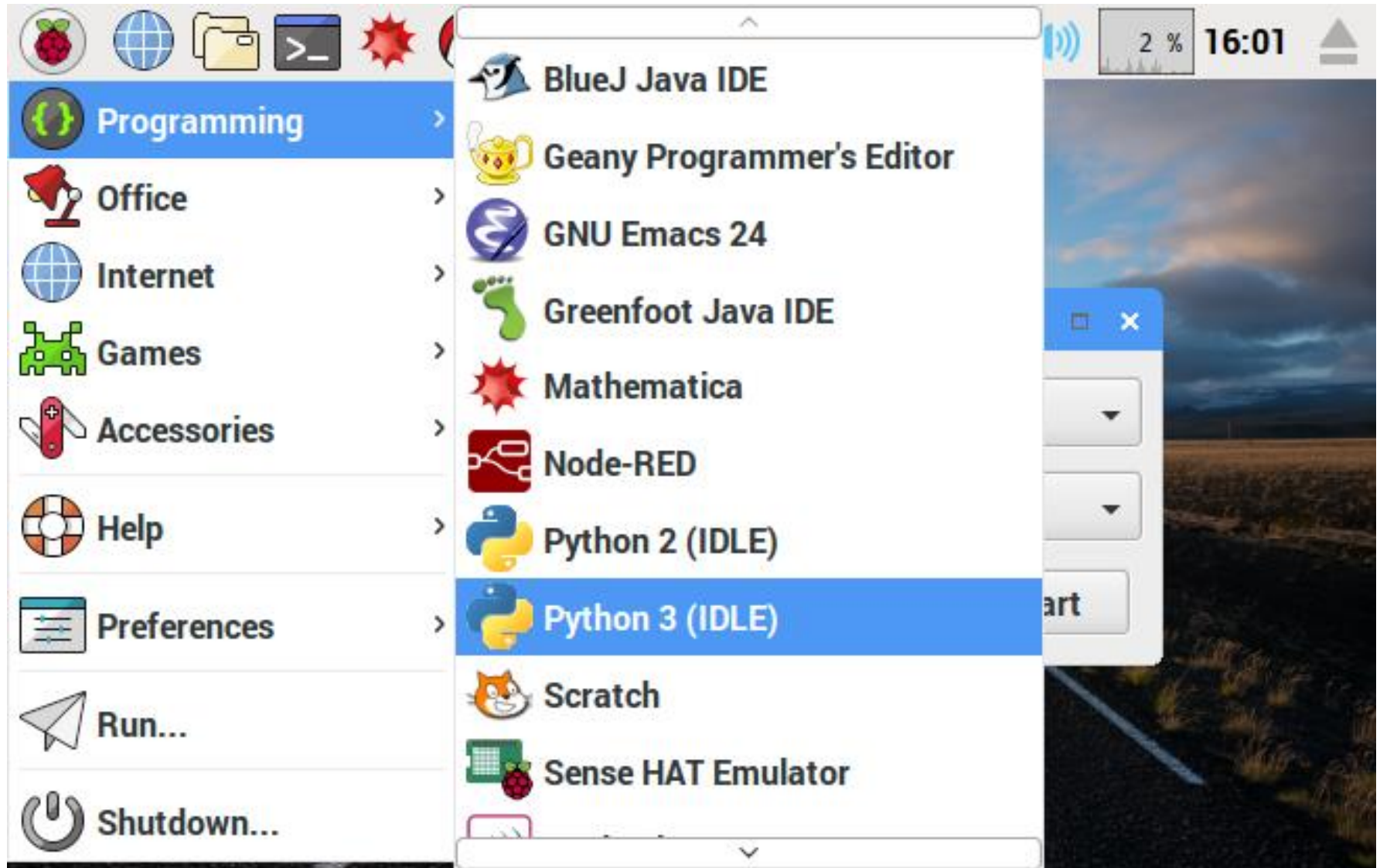
Careful with GPIO Ports !

- **Never:**
 - **Connect 5 V or 3.3 V directly to MINUS (GND)**
Connect PLUS directly to MINUS (GND)
-> SHORT CIRCUIT
 - **Connect GPIO PIN in OUTPUT MODE directly to MINUS (GND) – if the PIN = HIGH = 3.3 V, then this is a SHORT CIRCUIT, too**
 - It is OK if pin is set to INPUT mode, though
 - For OUTPUT mode, one always needs a „consumer“ – i.e., and LED and a resistor, otherwise it is a short circuit

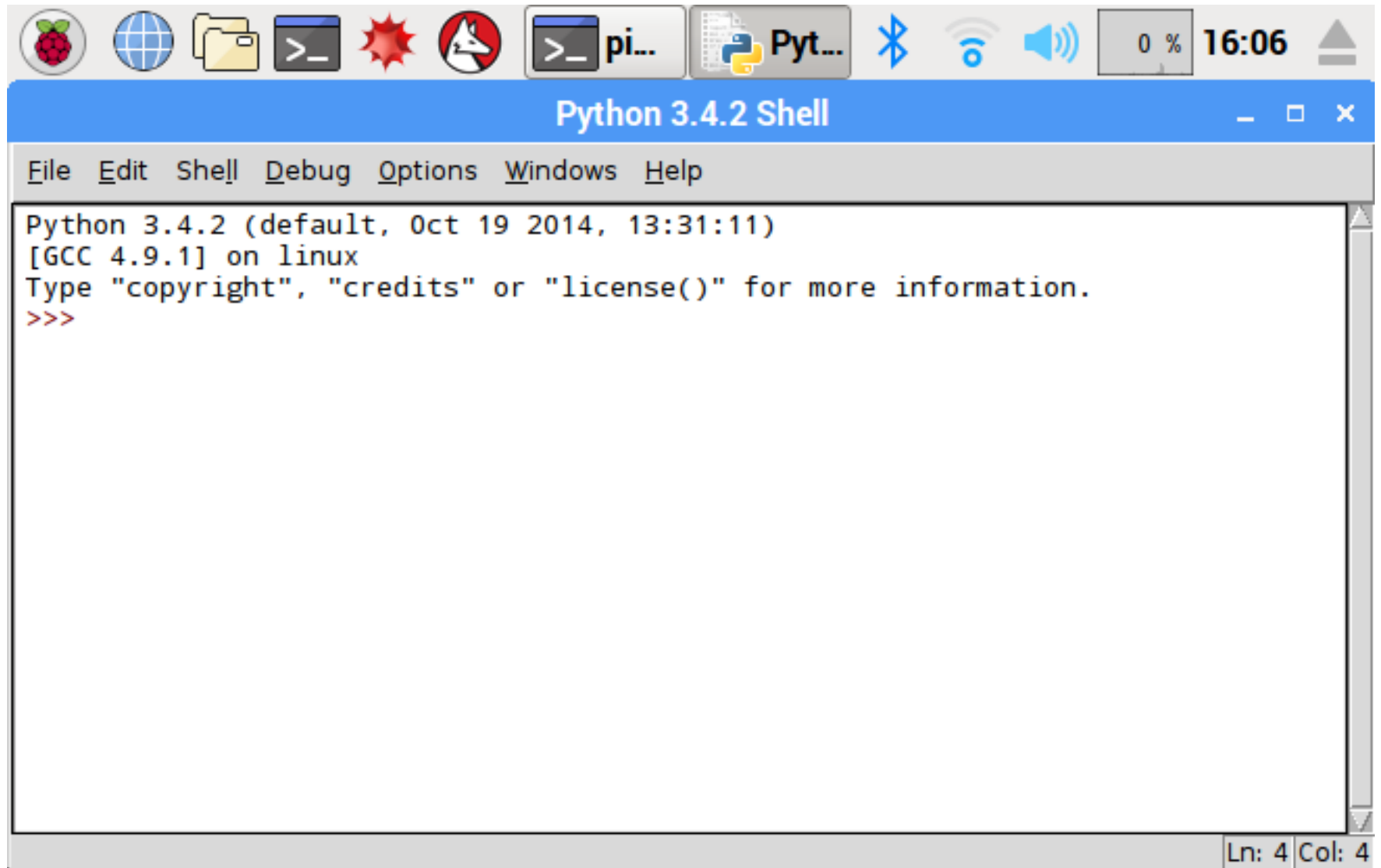
Blink LED Circuit – „Hello World“ of Embedded Programming



... on to Python GPIO Programming



Python Shell / Console / ...

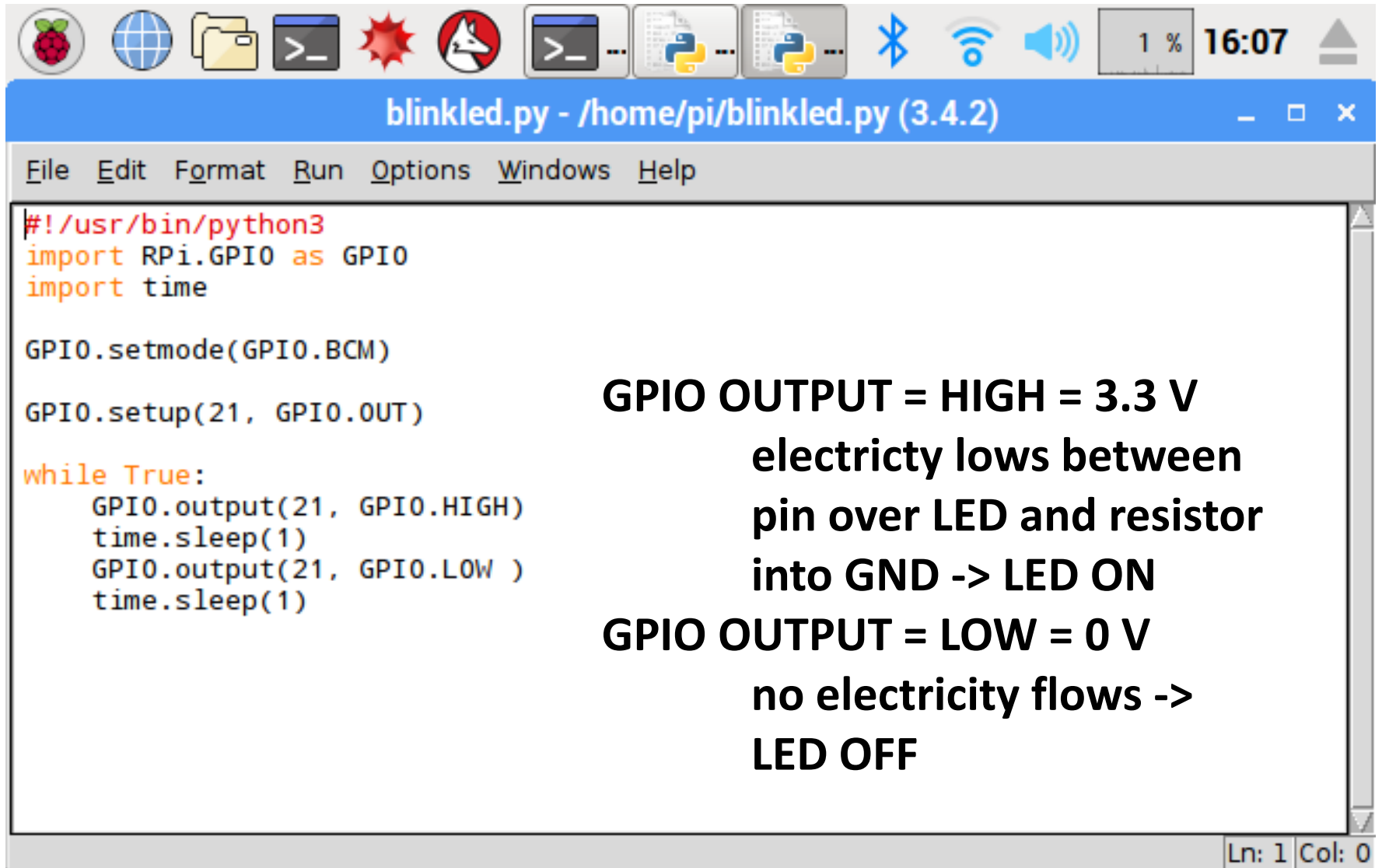


The image shows a desktop environment on a Raspberry Pi. The top panel contains various icons including the Raspberry Pi logo, a globe, a folder, a terminal icon, a red star, a cat icon, a terminal icon labeled 'pi...', a Python icon labeled 'Pyt...', Bluetooth, Wi-Fi, and a speaker icon. On the right, it shows '0 %' battery and the time '16:06'. A window titled 'Python 3.4.2 Shell' is open, displaying the Python 3.4.2 startup message and the prompt '>>>'. The window has a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Windows', and 'Help'. The status bar at the bottom right shows 'Ln: 4 Col: 4'.

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>>
```

Ln: 4 Col: 4

Editor – Use Function-F5 / Ctrl-c



The screenshot shows a Raspberry Pi desktop environment with a terminal window open. The terminal window has a blue title bar that reads "blinkled.py - /home/pi/blinkled.py (3.4.2)". The menu bar includes File, Edit, Format, Run, Options, Windows, and Help. The code in the terminal is a Python script that blinks an LED by toggling the GPIO output between HIGH and LOW states with a 1-second delay. To the right of the code, there are two text blocks explaining the logic: HIGH output (3.3V) turns the LED on, and LOW output (0V) turns it off. The status bar at the bottom right shows "Ln: 1 Col: 0".

```
#!/usr/bin/python3
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

GPIO.setup(21, GPIO.OUT)

while True:
    GPIO.output(21, GPIO.HIGH)
    time.sleep(1)
    GPIO.output(21, GPIO.LOW )
    time.sleep(1)
```

GPIO OUTPUT = HIGH = 3.3 V
electricity flows between
pin over LED and resistor
into GND -> LED ON

GPIO OUTPUT = LOW = 0 V
no electricity flows ->
LED OFF

Ln: 1 Col: 0

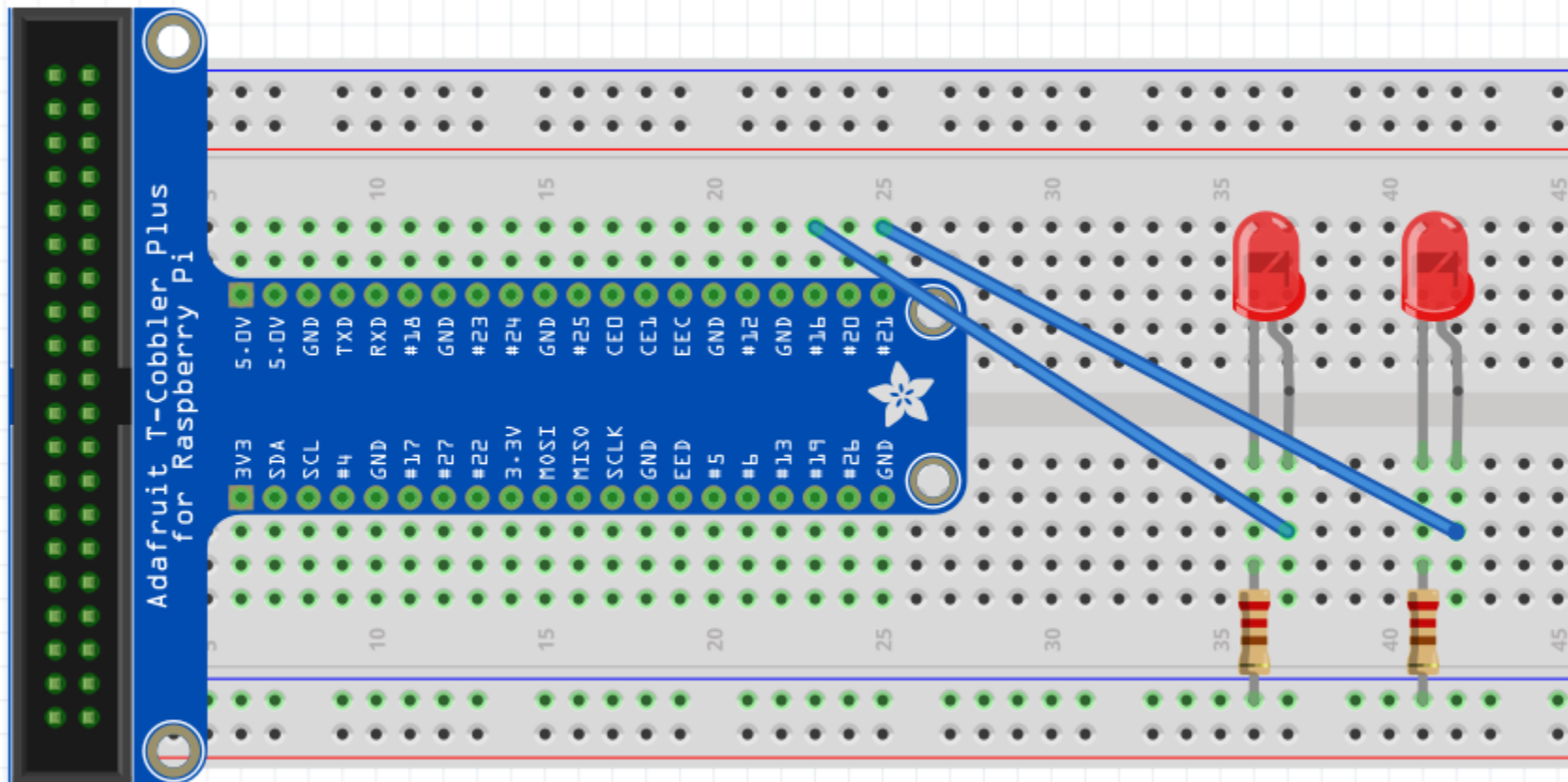
Download Example Program

<https://www.michael-wessel.info/rpi/>

Improvements

- This is a poorly structured program!
 - Let's learn some Python
- Clean up source code
 - Write a couple of functions, „setup“, „led_on“, „led_off“, „main“
 - Avoid „magical numbers“, e.g. for PIN numbers in code
Use global variables for led1, ...
 - DRY Principle in Programming – DO NO REPEAT YOURSELF
 - Write a „main“ function
 - read in blinking speed from keyboard using:
speed = int(input("Speed?"))
 - Also, use GPIO.cleanup() at the end
 - try:**
<main function code>
 - finally:**
GPIO.cleanup()

Blink 2 LEDs



```

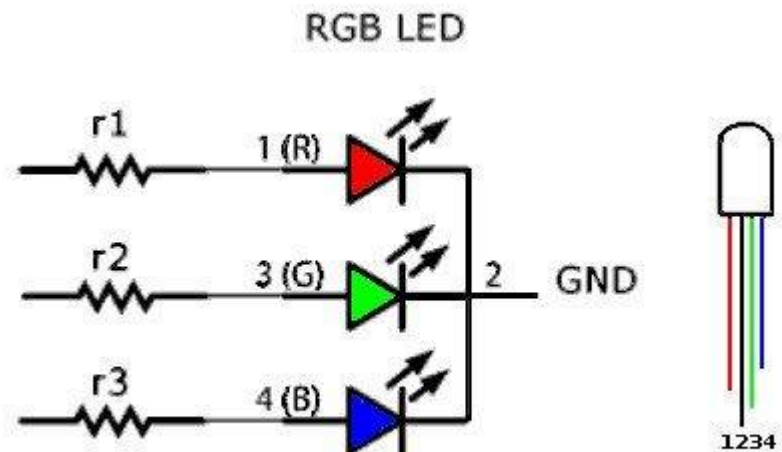
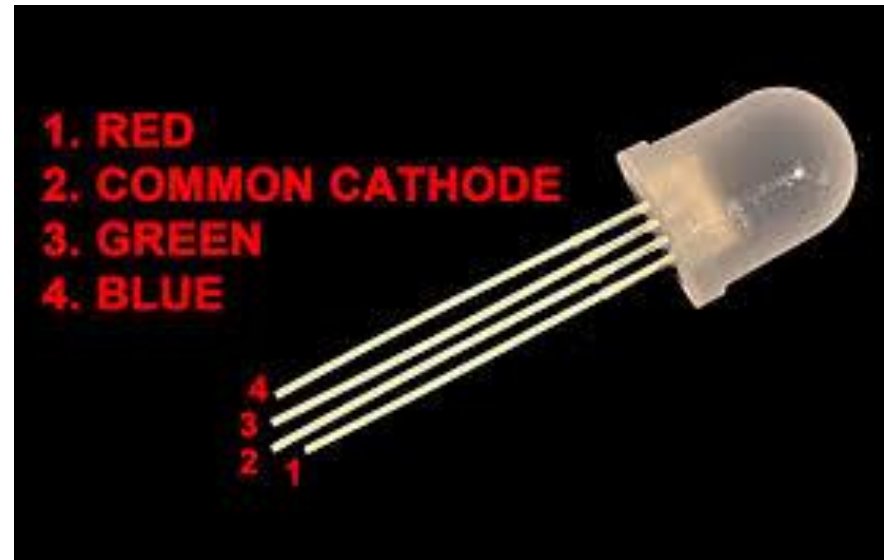
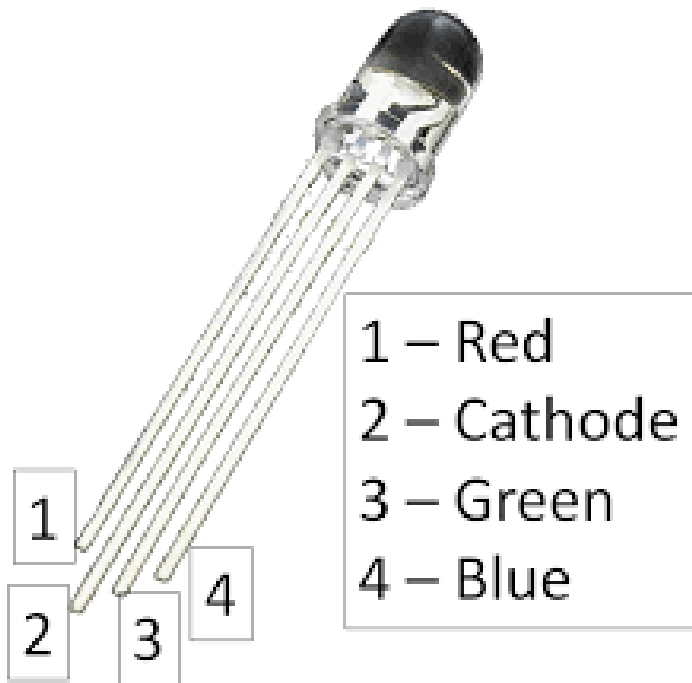
#!/usr/bin/python3
import RPi.GPIO as GPIO
import time
|
led1 = 21
led2 = 16
def setup():
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(led1, GPIO.OUT)
    GPIO.setup(led2, GPIO.OUT)

def led_on(led_pin):
    GPIO.output(led_pin, GPIO.HIGH)
def led_off(led_pin):
    GPIO.output(led_pin, GPIO.LOW)
def get_led_pin(led):
    if led == 1:
        return led1
    elif led == 2:
        return led2
    else:
        raise ValueError("Bad LED: ", led)
def main():
    speed = int(input("Speed in milliseconds? ")) / 1000.0
    led = int(input("Led? "))
    led_pin1 = get_led_pin(led)
    led_pin2 = get_led_pin(3-led)
    try:
        setup()
        while True:
            led_on(led_pin1)
            led_off(led_pin2)
            time.sleep(speed)
            led_off(led_pin1)
            led_on(led_pin2)
            time.sleep(speed)
    finally:
        led_off(led1)
        led_off(led2)
        GPIO.cleanup()

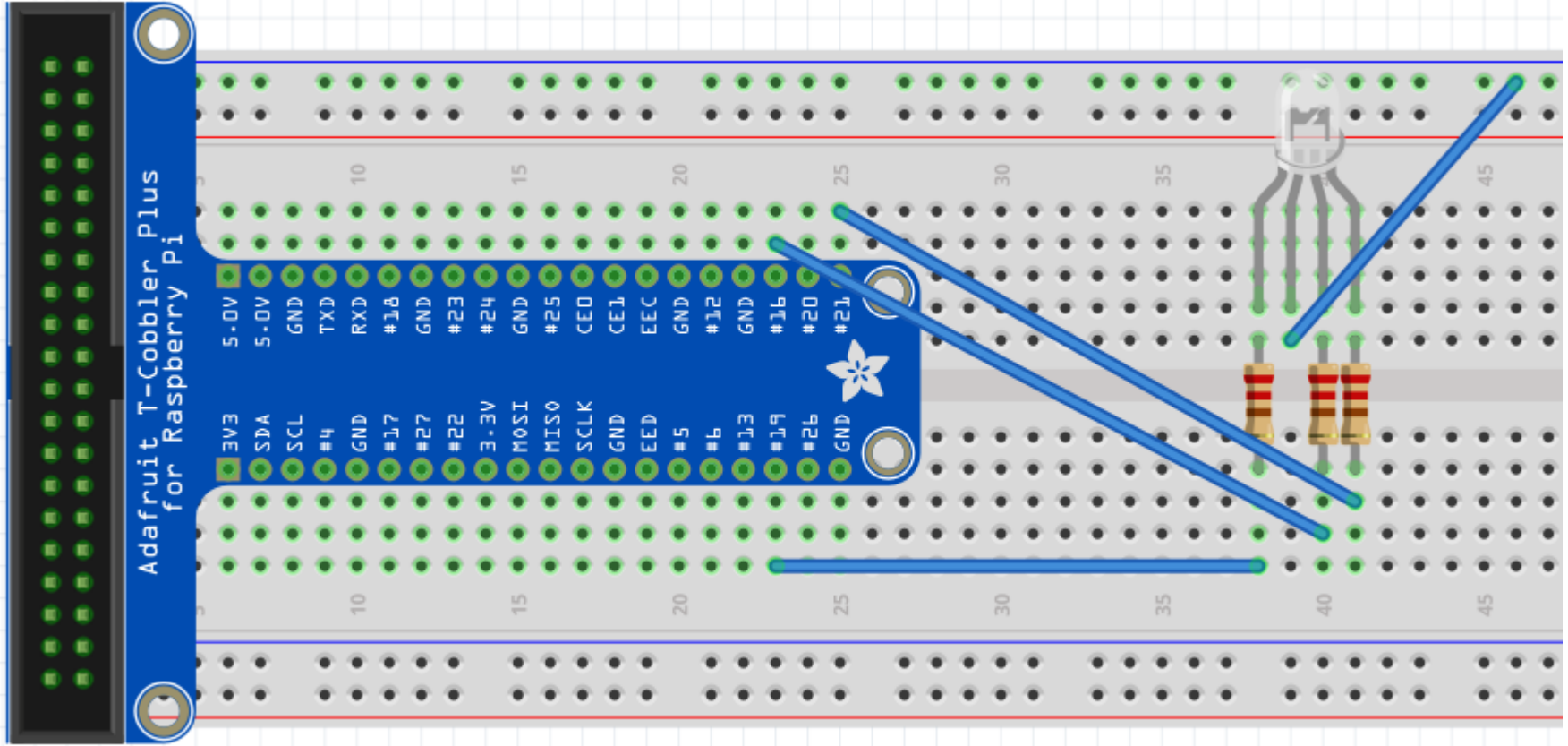
main()

```

RGB LED – Common CATHODE



RGB LED



```
#!/usr/bin/python3
import RPi.GPIO as GPIO
import time

leds = [ 21, 16, 19 ]

def setup():
    GPIO.setmode(GPIO.BCM)
    for led_pin in leds:
        GPIO.setup(led_pin, GPIO.OUT)

def led_on(led_pin):
    GPIO.output(led_pin, GPIO.HIGH)

def led_off(led_pin):
    GPIO.output(led_pin, GPIO.LOW)

def main():
    speed = int(input("Speed in milliseconds? ")) / 1000.0

    try:
        setup()
        while True:
            for led in leds:
                led_on(led)
                time.sleep(speed)
                led_off(led)
    finally:
        for led in leds:
            led_off(led)
        GPIO.cleanup()

main()
```

```
#!/usr/bin/python3
import RPi.GPIO as GPIO
import time

leds = [ 21, 16, 19 ]
button = 26

def setup():
    GPIO.setmode(GPIO.BCM)
    for led_pin in leds:
        GPIO.setup(led_pin, GPIO.OUT)
    GPIO.setup(button, GPIO.IN)

def led_on(led_pin):
    GPIO.output(led_pin, GPIO.HIGH)

def led_off(led_pin):
    GPIO.output(led_pin, GPIO.LOW)

def main():
    speed = int(input("Speed in milliseconds? ")) / 1000.0

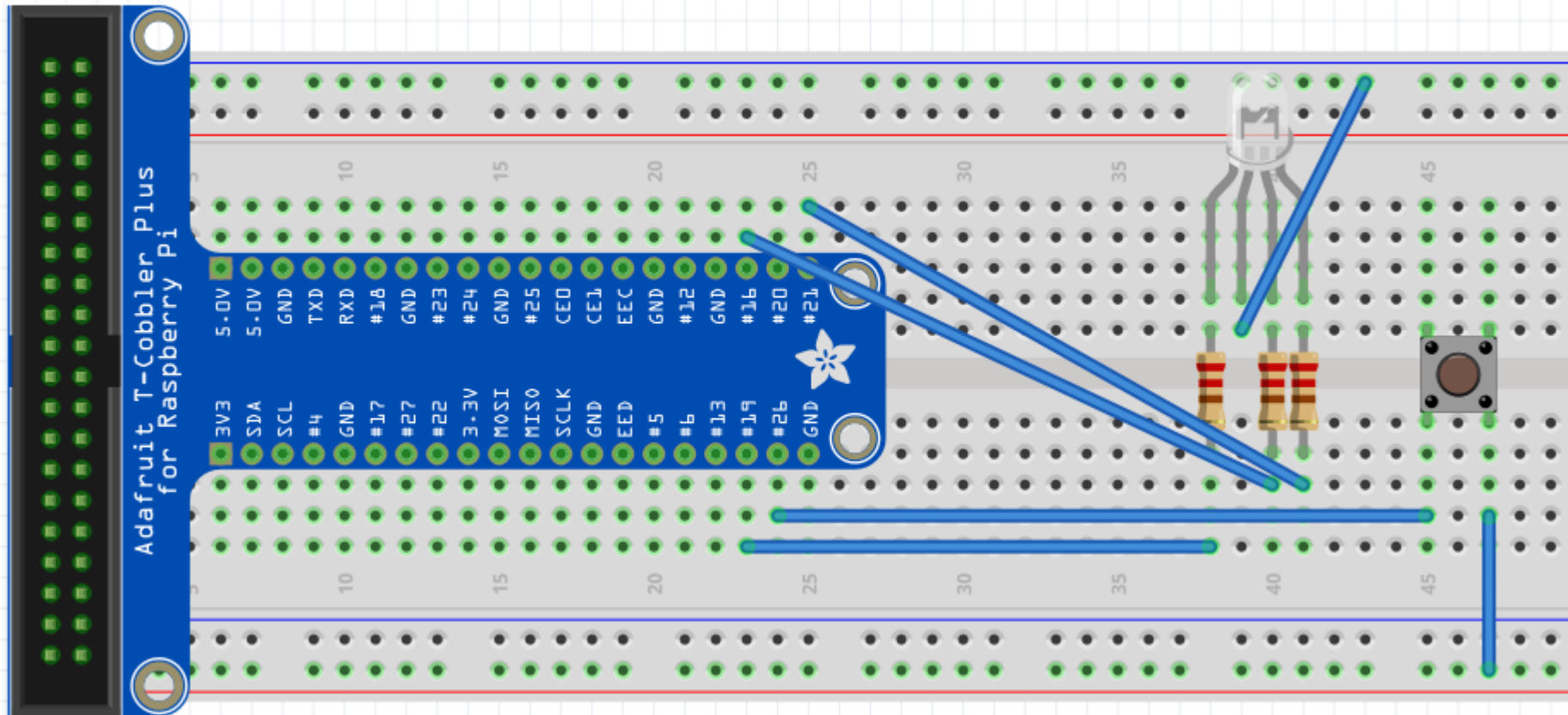
    try:
        setup()
        while True:
            for led in leds:
                led_on(led)
                time.sleep(speed)
                led_off(led)
            for led in leds[::-1]:
                led_on(led)
                time.sleep(speed)
                led_off(led)
    finally:
        for led in leds:
            led_off(led)
        GPIO.cleanup()

main()
```

Digital Input - Button



How NOT To Connect the Button – But let's try, you will be amazed!!



```
#!/usr/bin/python3
import RPi.GPIO as GPIO
import time

leds = [ 21, 16, 19 ]
button_pin = 26

def setup():
    GPIO.setmode(GPIO.BCM)
    for led_pin in leds:
        GPIO.setup(led_pin, GPIO.OUT)
    GPIO.setup(button_pin, GPIO.IN)

def led_on(led_pin):
    GPIO.output(led_pin, GPIO.HIGH)

def led_off(led_pin):
    GPIO.output(led_pin, GPIO.LOW)

def read_button(button):
    return GPIO.input(button_pin)

def main():

    try:
        setup()
        led = 0

        while True:
            if read_button(button_pin) == GPIO.HIGH:
                led_off(leds[led])
                led += 1
                led %= len(leds)
                led_on(leds[led])

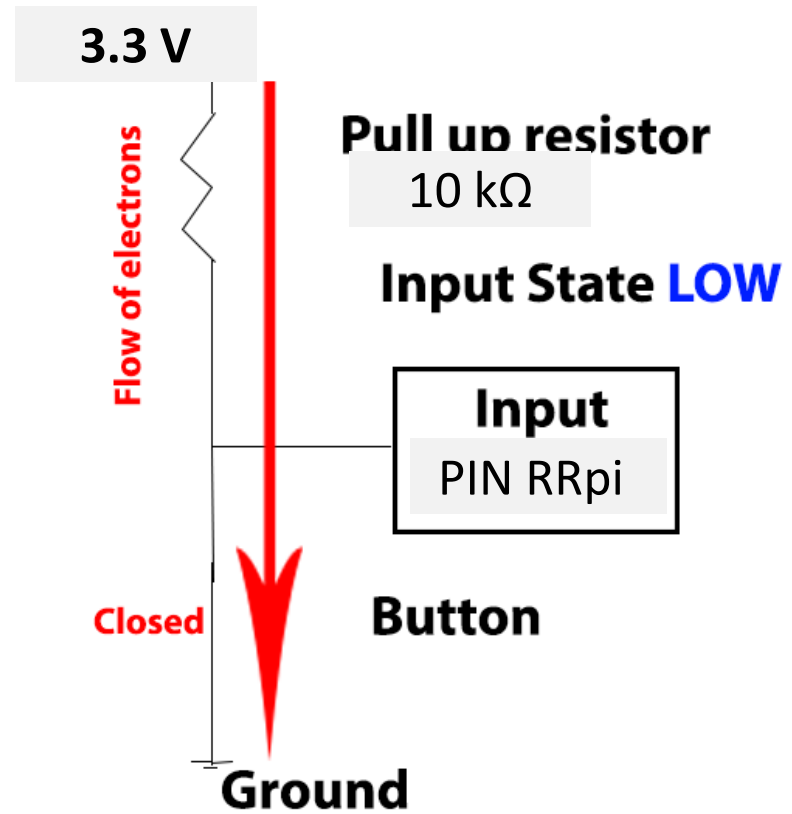
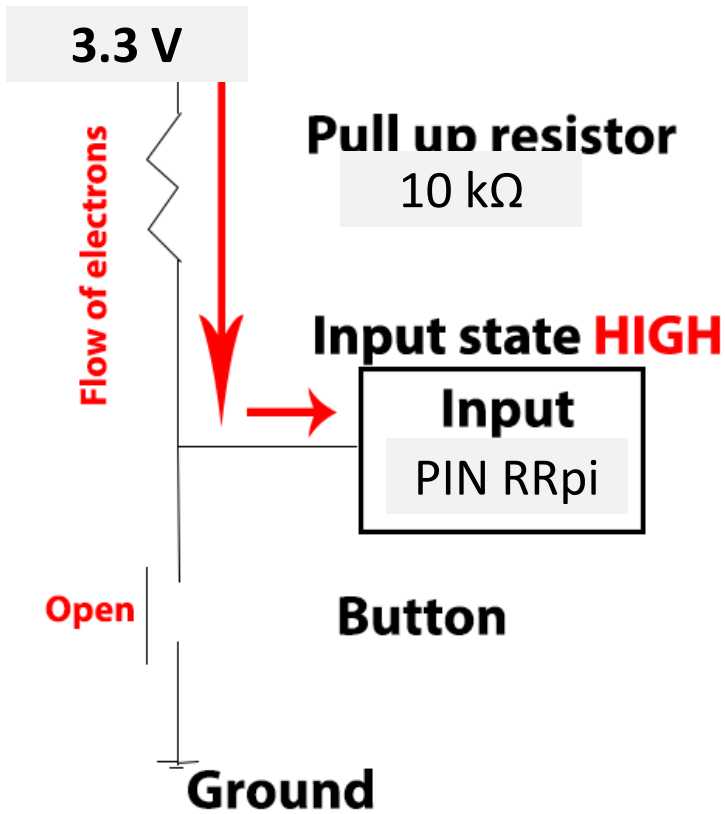
    finally:
        for led in leds:
            led_off(led)
        GPIO.cleanup()

main()
```

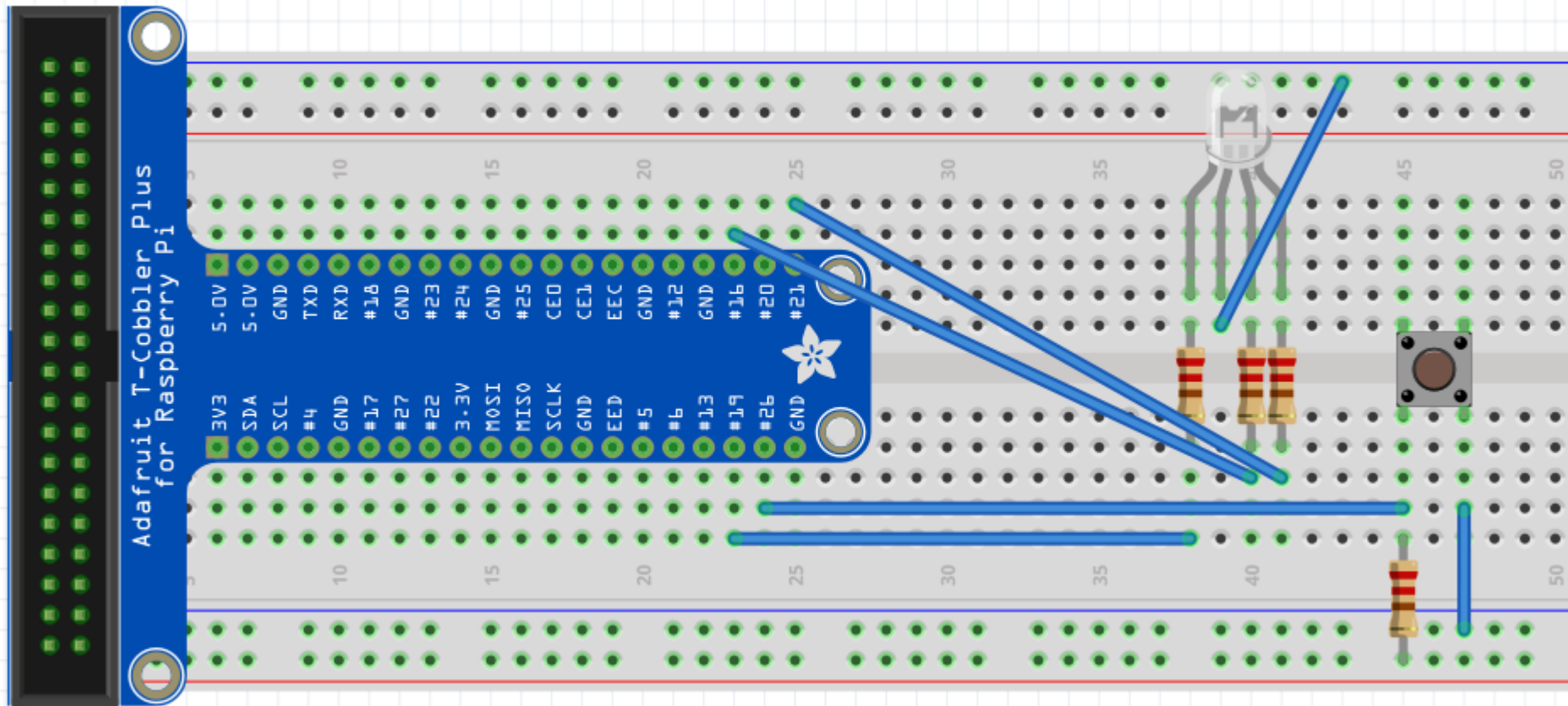
Explanation – Floating Inputs

- Unconnected Inputs do not have a well-defined logical state!
 - They are neither 0 / LOW nor 1 / HIGH
 - We need to use something more elaborate
 - **Pull Up Resistor circuit**
 - When not pressed, logic level = HIGH
 - When pressed, logic level = LOW
 - There are also **Pull Down resistors**
 - Inverse function (different circuit!)

Pull Up Resistor



Important - Use the 10 k Ω resistor !
From the smaller pack of 5 resistors



And one more problem...

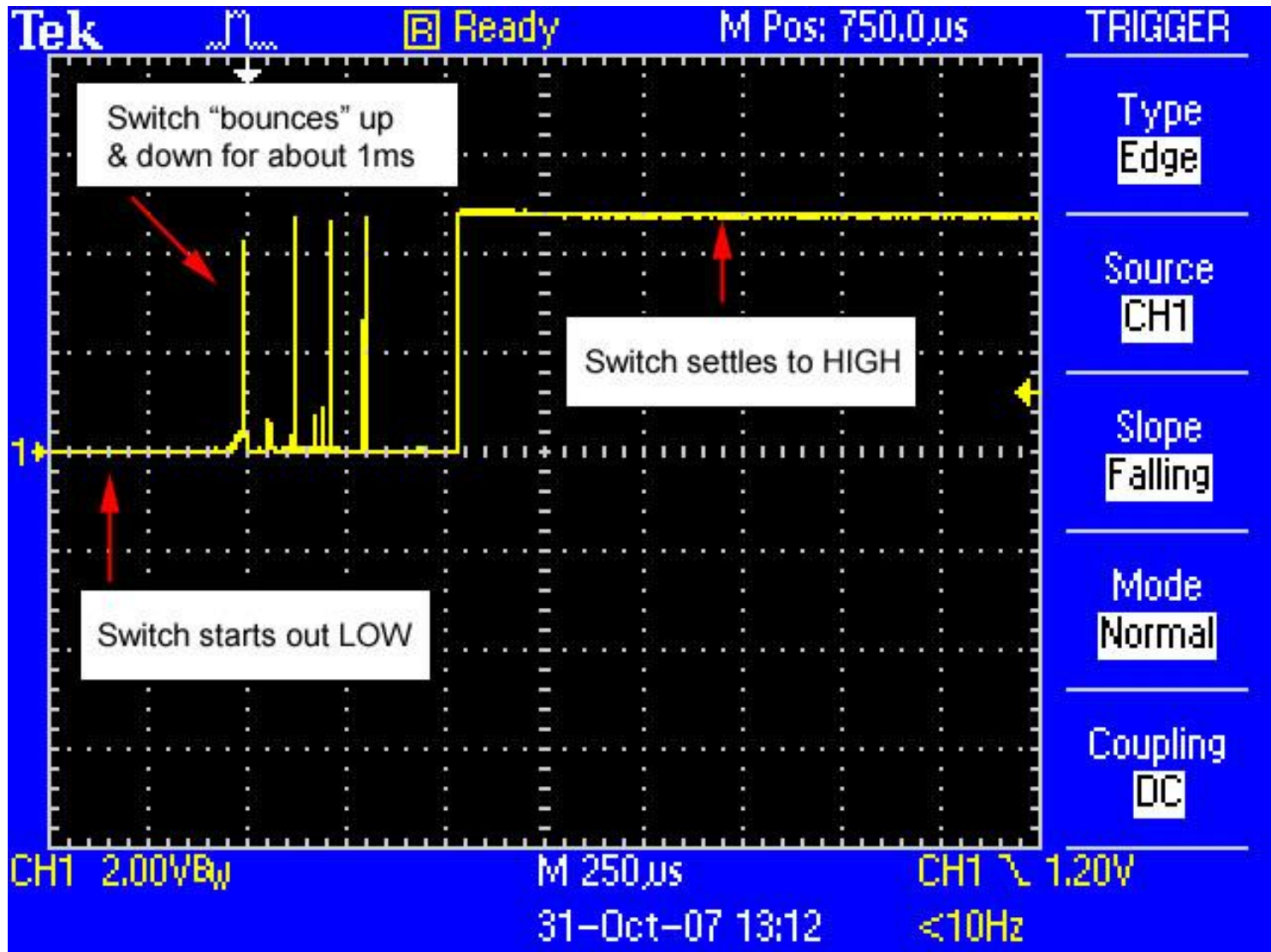
- Even if we wait until the button goes LOW again before we continue in our main loop:

```
while read_button(button_pin) == GPIO.LOW:  
    pass
```



**... then there is STILL something strange going on!
Sometimes, the LED jumps over colors, or there
is no change at all! WHY?**

Button Debouncing



```
#!/usr/bin/python3
import RPi.GPIO as GPIO
import time

leds = [ 21, 16, 19 ]

# configure as pull up - pin over 10 kOhms to 3.3 and button
# other button pin to GND
# normally, HIGH, when closed, LOW

button_pin = 26

def setup():
    GPIO.setmode(GPIO.BCM)
    for led_pin in leds:
        GPIO.setup(led_pin, GPIO.OUT)
    GPIO.setup(button_pin, GPIO.IN)

def led_on(led_pin):
    GPIO.output(led_pin, GPIO.HIGH)

def led_off(led_pin):
    GPIO.output(led_pin, GPIO.LOW)

def read_button(button):
    return GPIO.input(button_pin)

def main():
    try:
        setup()
        led = 0
        while True:
            if read_button(button_pin) == GPIO.LOW:
                led_off(leds[led])
                #led += 1
                #led %= len(leds)
                led = ( led + 1 ) % len(leds)
                led_on(leds[led])
                time.sleep(0.1)
                while read_button(button_pin) == GPIO.LOW:
                    pass
    finally:
        for led in leds:
            led_off(led)
        GPIO.cleanup()

main()
```