Python

7/8/2025

MAYFLOWER
A.S.T.E.M.
ACADEMY

1

Introduction

What is Python?

- Object oriented language
- Interpreted language
- Supports dynamic data type
- Independent from platforms
- Focused on development time
- Simple and easy grammar
- High-level internal object data types
- Automatic memory management
- It’s free (open source)!

Language properties

- Everything is an object
- Modules, classes, functions
- Exception handling
- Dynamic typing, polymorphism
- Static scoping
- Operator overloading
- Indentation for block structure

High-level data types

- Numbers: int, long, float, complex
- Strings: immutable
- Lists and dictionaries: containers
- Other types for e.g. binary data, regular expressions, introspection
- Extension modules can define new “built-in” data types

Variables Types

Variables have a type, which defines the way it is stored.

Variable Types	Basic Examples	Real World Examples
Integer	Z = 1	Release_Year= 2003
Float	Z = 1.23	Rating = 8.2
String	Z = “Python”	Movie = “Finding Nemo”
Boolean	Z = True	Is_Released = True
List	Z = [“Alice”, “Lily”]	Actors = [“Elen DeGeneres”, “Alexander Gould”]
Dictionary	Z = {“Name”: “Alan”, “ID”: 1}	Box_Office_Collection= {“US”: “339.7M”, “UK”: “67.1M”}
Tuple	Z= (“Monday”, 26, “June”)	Release_Date = (“30th”, “May”, 2003)
Set	Z= {2,3,5}	Genre = {“Animation”, “Family”, “Adventure”}

EDUCBA

https://www.educba.com/python-variables/

Casting

```
x = 10    # This is an integer
y = "20"  # This is a string
x + y
```

```
x = 10    # This is an integer
y = "20"  # This is a string
x + int(y)
```

```
x = "10"
y = "20"
x + y
```

Arithmetic operations

Operators	Meaning	Example	Result
+	Addition	4 + 2	6
–	Subtraction	4 – 2	2
*	Multiplication	4 * 2	8
/	Division	4 / 2	2
%	Modulus operator to get remainder in integer division	5 % 2	1
**	Exponent	5**2 = 5^2	25
//	Integer Division/ Floor Division	5//2 -5//2	2 -3

https://www.devopsschool.com/blog/arithmetic-operators-in-python/

Comparison operators

Operators	Meaning	Example	Result
<	Less than	5<2	False
>	Greater than	5>2	True
<=	Less than or equal to	5<=2	False
>=	Greater than or equal to	5>=2	True
==	Equal to	5==2	False
!=	Not equal to	5!=2	True

https://www.devopsschool.com/blog/relational-comparison-operators-in-python/

Logical operators

logical operator	Description	Syntax
and	True if both statements are true	a and b
or	True if one of the statements is true	a or b
not	Reverse the result. If the result is true it will return false and viceversa	not a not b

Example

```
x = 14
y = 42

xDivisible = ( x % 2 ) == 0 # check if x is a multiple of 2
yDivisible = ( y % 3 ) == 0 # check if y is a multiple of 3

not (xDivisible and yDivisible)
```

Printing

```
str1 = "which means it has even more than"
str2 = 76
str3 = "quirks"
print(str1 + str2 + str3)
```

```
str1 = "It has"
str2 = 76
str3 = "methods!"
print(str1, str2, str3)
```

List

```
fruits = ["apple", "orange", "tomato", "banana"]  # a list of strings
print(type(fruits))
print(fruits)
```


```
fruits[4]
```

```
len(fruits)
```

List

```
fruits[2] = "apricot"
print(fruits)
```

```
fruits.append("lime")     # add new item to list
print(fruits)
fruits.remove("orange")   # remove orange from list
print(fruits)
```

Lists with integers

```
nums = list(range(10))
print(nums)
```

```
nums = list(range(0, 100, 5))
print(nums)
```

List

```
print(nums)
print(nums[1:5:2]) # Get from item 1(starting point) through item 5
print(nums[0:3]) # Get items 0 through 3(not included)
print(nums[4:]) # Get items 4 onwards
print(nums[-1]) # Get the last item
print(nums[::-1]) # Get the whole list backwards
```

```
print(len(nums))   # number of items within the list
print(max(nums))   # the maximum value within the list
print(min(nums))   # the minimum value within the list
```

Tuples

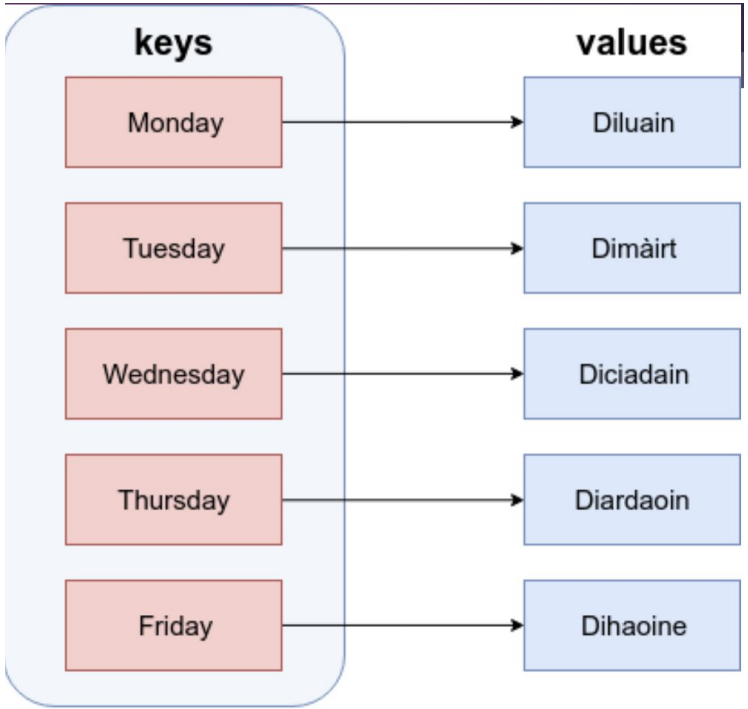
Effectively lists that are immutable (can't be changed)

```
fruits = ("apple", "orange", "tomato", "banana")
print(type(fruits))
print(fruits)
```

Dictionaries

They are effectively 2 lists combined – keys and values

We use the keys to access the values instead of indexing them like a list • Each value is mapped to a unique key



Dictionaries

```
days = {"Monday": "Diluain", "Tuesday": "Dimàirt",
        "Wednesday": "Diciadain", "Thursday": "Diardaoin",
        "Friday": "Dihaoine"}
print(type(days))
print(days)
```

- Values are mapped to a key
- Values are accessed by their key
- Key are unique and are immutable
- Values cannot exist without a key

```
days["Friday"]
```

```
'Dihaoine'
```

Dictionaries

```
days.update({"Saturday": "Disathairne"})
print(days)
days.pop("Monday")  # Remove Monday because nobody likes it
print(days)
```

```
print(days.keys())   # get only the keys of the dictionary
print(days.values()) # get only the values of the dictionary
```

If Else

```
x = True
if x:
    print("Executing if")
else:
    print("Executing else")
print("Prints regardless of the outcome of the if-else block")
```

- Code is grouped by its indentation
- Indentation is the number of whitespace or tab characters before the code.
- If you put code in the wrong block then you will get unexpected behavior

If-elif-else

```
if condition1:
    condition 1 was True
elif condition2:
    condition 2 was True
else:
    neither condition 1 or condition 2 were True
```

```
purchasePrice = float(input("Price at which you have purchased bitcoins: "))
currentPrice = float(input("Current price of the bitcoins: "))

if currentPrice < purchasePrice*0.9:
    print("Not a good idea to sell your bitcoins now.")
    print("You will lose", purchasePrice - currentPrice, "£ per bitcoin.")
elif currentPrice > purchasePrice*1.2:
    print("You will make", currentPrice - purchasePrice, "£ per bitcoin.")
else:
    print("Not worth selling right now.")
```

For loop

```
for item in itemList:
    do something to item
```

Allows us to iterate over a set amount of variables within a data structure. During that we can manipulate each item however we want.

```
fruitList = ["apple", "orange", "tomato", "banana"]
for fruit in fruitList:
    print("The fruit", fruit, "has index", fruitList.index(fruit))
```

```
numbers = list(range(10))
for num in numbers:
    squared = num ** 2
    print(num, "squared is", squared)
```

While loop

```
n = 0
while n < 5:
    print("Executing while loop")
    n = n + 1

print("Finished while loop")
```

```
n = 0
while True:  # execute indefinitely
    print("Executing while loop")

    if n == 5:  # stop loop if n is 5
        break

    n = n + 1

print("Finished while loop")
```

Exception Handling

Using **try / except** to handle runtime errors.

```
try:
    x = int(input("Enter a number: "))
except ValueError:
    print("Invalid input!")
```

Function

```
def printNum(num):
    print("My favourite number is", num)

printNum(7)
printNum(14)
printNum(2)
```

$$(val - src[0]) \times \frac{dst[1] - dst[0]}{src[1] - src[0]} - dst[0]$$

```
def roundNum(num):
    remainder = num % 1
    if remainder < 0.5:
        return num - remainder
    else:
        return num + (1 - remainder)

# Will it work?
x = roundNum(3.4)
print (x)

y = roundNum(7.7)
print(y)

z = roundNum(9.2)
print(z)
```

```
# Generic scale function
# Scales from src range to dst range
def scale(val, src, dst=(-1,1)):
    return (int(val - src[0]) / (src[1] - src[0])) * (dst[1] - dst[0]) + dst[0]

print(scale(49, (-100,100), (-50,50)))
print(scale(49, (-100,100)))
```

Python built-in function

Built-in Functions

A
abs()
aiter()
all()
anext()
any()
ascii()

B
bin()
bool()
breakpoint()
bytearray()
bytes()

C
callable()
chr()
classmethod()
compile()
complex()

D
delattr()
dict()
dir()
divmod()

E
enumerate()
eval()
exec()

F
filter()
float()
format()
frozenset()

G
getattr()
globals()

H
hasattr()
hash()
help()
hex()

I
id()
input()
int()
isinstance()
issubclass()
iter()

L
len()
list()
locals()

M
map()
max()
memoryview()
min()

N
next()

O
object()
oct()
open()
ord()

P
pow()
print()
property()

R
range()
repr()
reversed()
round()

S
set()
setattr()
slice()
sorted()
staticmethod()
str()
sum()
super()

T
tuple()
type()

V
vars()

Z
zip()

_
__import__()

https://docs.python.org/3/library/functions.html

2

Algorithms

Efficiency of algorithms

There are many algorithms to perform the same task. Which one to choose?

- The one that runs faster?
- The one that requires less space?

Speed (time complexity)

- Also called running or execution time.
- Denoted by the letter O (big oh).

Space consumption (space complexity)

- Memory usage.
- Also represented using big O.
- Always less than or equal to the time requirement.

Running Time

To measure running time, count the number of operations that are most often executed.

Input size depends on the problem:

- Searching in an list of size n.
- In a graph with v vertices and e edges.

Sequential Search

Searching an list (ordered or unordered)

Example list: [3, 5, 12, 2, 56, 32, 8, 14] (list index: 0 to 7)

```
def sequential_search(list_in, target):
    for i in range(len(list_in)):
        if list_in[i] == target:
            print(f"{target} found")
            return True
    print(f"{target} not found")
    return False
```

Best case? Worst case?

Basic Operation: **Comparison (list_in[i] == target)** → Done **n** times
Running time: **f(n) = n**

Asymptotic Complexity

Asymptotic analysis is a method of describing a limiting behavior.

The O notation asymptotically bounds a function:

- Estimate the complexity in the asymptotic sense, i.e., for arbitrarily large inputs.

O(f(n)) is a group of functions that:

- Behave like f(n) as n gets large.
- Ignore constant multiples.
- O(n) includes: n, n*20, n+17, 5*n+log(n)
- O(n^2) includes: n^2, n^2+20*n−99, n^2−n^½

Order of growth

Description	Order of Growth	Description	Example
constant	1	statement	add two numbers
logarithmic	log N	divide in half	binary search
linear	N	loop	find the maximum
log-linear	N log N	divide and conquer	mergesort
quadratic	N^2	double loop	check all pairs
cubic	N^3	triple loop	check all triples
exponential	2^N	exhaustive search	check all subsets

Binary Search: Searching for 12

Searching an list (ordered) [3 5 12 22 56 62 85 94 95 99] (list index: 0 to 9)

Step 1:

- Left = 0, Right = 9, Middle = 4 (56)
- Is 12 == 56?
- Is 12 < 56? ⇒ Search left half.

Step 2:

- Left = 0, Right = 3, Middle = 1 (5)
- Is 12 == 5?
- Is 12 < 5? ⇒ Search right half.

Step 3:

- Left = 2, Right = 3, Middle = 2 (12)
- Is 12 == 12?

Binary Search

```
def binary_search(array, target):
    left, right = 0, len(array) - 1
    while left <= right:
        middle = (left + right) // 2
        if array[middle] == target:
            print(f"{target} found at index {middle}")
            return middle
        elif target < array[middle]:
            right = middle - 1
        else:
            left = middle + 1
    print(f"{target} not found")
    return -1
```

Executed y times where n = 2^y
log_a(b) = c is equivalent to b = a^c
the loop is executed log n times

Best case: O(1) (target found in the middle)
Worst case: O(log n)

3
Simple Exercise

Practices

1. Calculate the area and perimeter of a rectangle. Only accept positive integers. If the input is invalid, print an error.
2. Write a program to check if a number is even or odd and positive or negative.
3. Given a list, use a for loop to find the maximum value without using max().
    a. my_list = [ ]
    b. find_max(my_list)
4. Write a function is_prime(n) that returns True if n is a prime number, else False.

4

Class

What are objects

An object is a datatype that stores data, but ALSO has operations defined to act on the data. It knows stuff and can do stuff.

- Generally represent:
  - tangible entities (e.g., student, airline ticket, etc.)
  - intangible entities (e.g., data stream)
- Interactions between objects define the system operation (through message passing)

E.g.: A Circle:

- Has **attributes** (knows stuff): – radius, center
- Has **method** (can do stuff): – area, circumference

Design of Circle object

- All objects are said to be an **instance** of some **class**. The class of an object determines which attributes the object will have.
- A class is a description of what its instances will know and do.

```
class Circle:
    def __init__(self, center, radius):
        self.center = center  # [x, y]
        self.radius = radius

    def area(self):
        return math.pi * self.radius ** 2

    def circumference(self):
        return 2 * math.pi * self.radius
```

Constructors

- The object's constructor method is named __init__
- The primary duty of the constructor is to set the state of the object's attributes (instance variables)
- Constructors may have default parameters

```
def __init__(self, center=None, radius=1):
    if center is None:
        center = [0, 0]
    self.center = center
    self.radius = radius
```

```
# Using defaults:
c1 = Circle()
print(c1.center)  # [0, 0]
print(c1.radius)  # 1

# Providing custom values:
c2 = Circle([5, 5], 3)
print(c2.center)  # [5, 5]
print(c2.radius)  # 3
```

Special methods

Method	Purpose
`__init__`	Constructor: initializes object state
`__str__`	String representation for `print()` and `str()`
`__eq__`	Defines `==` equality comparison
`__lt__`	Defines `<` less-than comparison
`__le__`	Defines `<=` less-than-or-equal
`__gt__`	Defines `>` greater-than comparison
`__ge__`	Defines `>=` greater-than-or-equal
`__getitem__`	Defines `obj[key]` indexing
`__setitem__`	Defines `obj[key] = value`

Circle

```
def __str__(self):
    return f"Circle(center={self.center}, radius={self.radius})"
def __eq__(self, other):
    return self.radius == other.radius
def __lt__(self, other):
    return self.radius < other.radius
```

```
c = Circle([5, 5], 3)
print(c) #Circle(center=[5, 5], radius=3)

c1 = Circle([0, 0], 5)
c2 = Circle([1, 1], 7)

print(c1 == c2)    # False
print(c1 < c2)     # True
print(c1 > c2)     # False
```

Using Instance Variables

In Python, assigning a variable that doesn't exist creates it automatically.

What About Instance Variables?

```
c1 = Circle([0, 0], 5)
c1.name = "circle"
```

- Python creates a new instance variable bob on the myCircle object automatically.

But... It Is Bad Practice

✅ You should not create instance variables this way outside of the constructor unless necessary.

Attributes / Instance Variables

- Attributes represent the characteristics of a class. When an object is instantiated and the values are assigned to attributes, they are then referred to as instance variables.
- The values of the instance variables define the state of the individual object
- They are referred to as instance variables because the values assigned to an individual object (instance of a class) are unique to that particular class
- Attributes may be public or private (although due to their specific implementation, they are not truly private in Python)
- If the attributes are private, they serve to enforce the concept of information hiding

Why class

- Classes and objects are more like the real world. They minimize the semantic gap by modeling the real world more closely
- The semantic gap is the difference between the real world and the representation in a computer.
- Classes and objects allow you to define an interface to some object (it's operations) and then use them without know the internals.
- Defining classes helps modularize your program into multiple objects that work together, that each have a defined purpose

Encapsulation

Encapsulation is the concept of bundling data (attributes) and methods (functions) together within a class and restricting direct access to some parts of the object.

- Protects the internal state of an object from unintended or harmful changes
- Provides controlled access through methods (getters/setters).
- Makes code more organized and modular.

Abstraction

Abstraction is the concept of hiding complex implementation details while exposing only the necessary parts to the user.

- Helps manage complexity in programs.
- Allows you to use objects without needing to understand their internal workings.
- When you use: print(myCircle.area()), you do not need to know how area is calculated internally, you just know it will give you the area.

5

Libraries

Library

A library in Python is a collection of pre-written code that you can use to perform common tasks easily without writing everything from scratch.

You import libraries using the import statement

- Perform mathematical calculations.
- Work with data efficiently.
- Control hardware like sensors and LEDs (for Raspberry Pi).
- Visualize data for analysis.

time

Used for time-related functions like delays and timestamps.

```
import time
time.sleep(1)  # Wait for 1 second
print(time.time())  # Current time in seconds
```

Example: Adding delays in LED blink, debounce buttons.

math

Provides mathematical functions like square root, trigonometric functions, power, etc.

```
import math
print(math.sqrt(16))  # 4.0
print(math.sin(math.pi / 2))  # 1.0
```

Useful for sensor data calculations and processing.

numpy

Powerful library for numerical computing and matrix operations.

Useful for handling large data arrays, performing mathematical operations efficiently.

```
import numpy as np
arr = np.array([1, 2, 3, 4])
print(arr * 2)  # [2, 4, 6, 8]
```

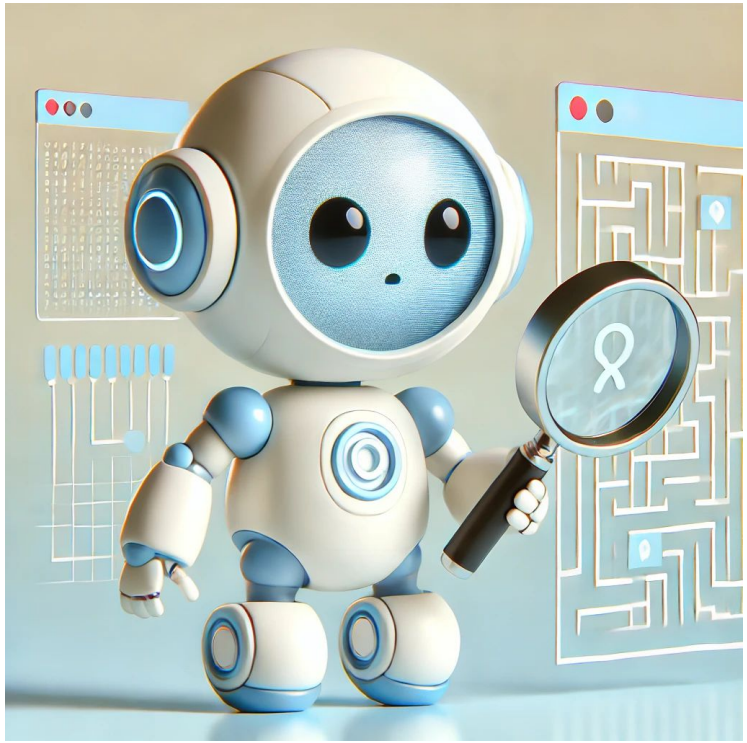
matplotlib

Library for data visualization and plotting graphs.

Helps visualize sensor data, example: ADC data.

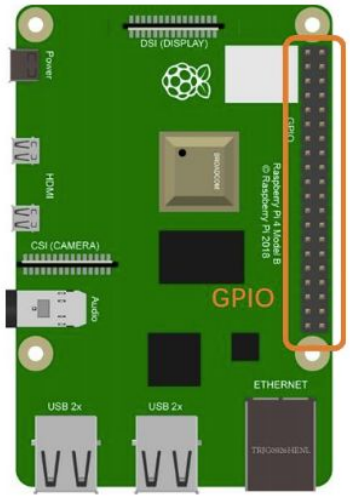
```
import matplotlib.pyplot as plt
x = [1, 2, 3, 4]
y = [10, 20, 25, 30]
plt.plot(x, y)
plt.xlabel('Time')
plt.ylabel('Sensor Value')
plt.show()
```

Hands on Lab



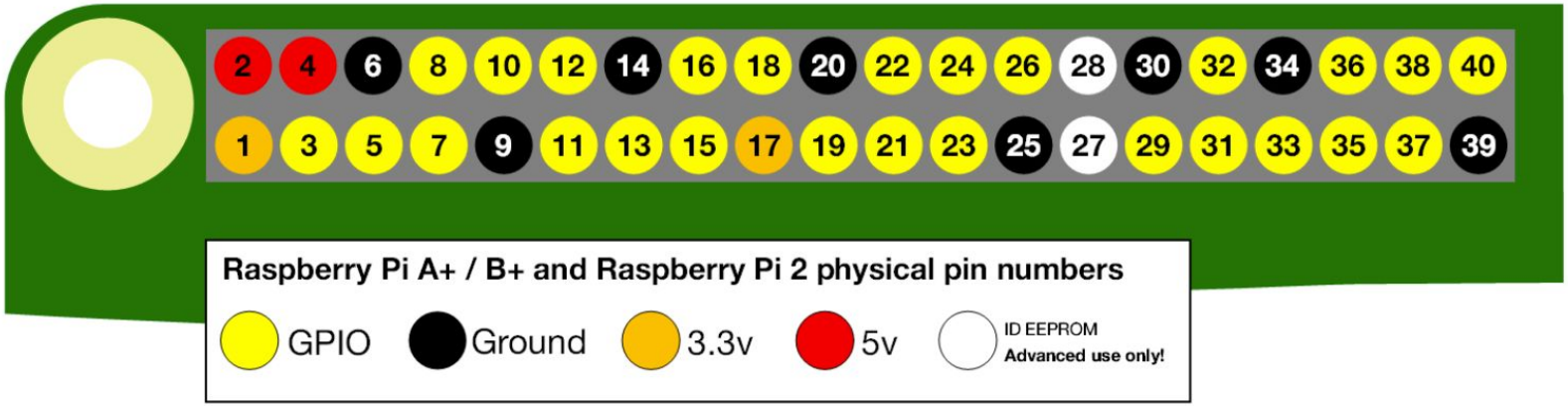
MAYFLOWER
A.S.T.E.M.
ACADEMY

Board Pinout



Pin 1	Pin 2	
+3V3		+5V
GPIO2 / SDA1		+5V
GPIO3 / SCL1		GND
GPIO4		TXD0 / GPIO 14
GND		RXD0 / GPIO 15
GPIO17		GPIO 18
GPIO27		GND
GPIO22		GPIO 23
+3V3		GPIO 24
GPIO10 / MOSI		GND
GPIO9 / MISO		GPIO 25
GPIO11 / SCLK		CE0# / GPIO8
GND		CE1# / GPIO7
GPIO0 / ID_SD		ID_SC / GPIO1
GPIO5		GND
GPIO6		GPIO12
GPIO13		GND
GPIO19 / MISO		CE2# / GPIO16
GPIO26		MOSI / GPIO20
GND		SCLK / GPIO21
Pin 39	Pin 40	

Physical Numbering



General Purpose Input/Output (GPIO)

- Generic Pin: Found on a microprocessor or microcontroller.
- Configuration: Can be set as input or output.
- Operation: Can be enabled or disabled.
- Operating Levels:
  - HIGH
  - LOW
- Output Pins: Can be written to and read from.
- Input Pins: Readable; writing over input pins possible.
- Interrupts: Pins configured as input can operate as interrupts.
- Purpose: No predefined operation or purpose, highly versatile.

Input Mode

- **Input Mode:** Configures GPIO pins to receive signals.
- **Floating State:** When not connected, the pin is in an undefined state, leading to erratic behavior.

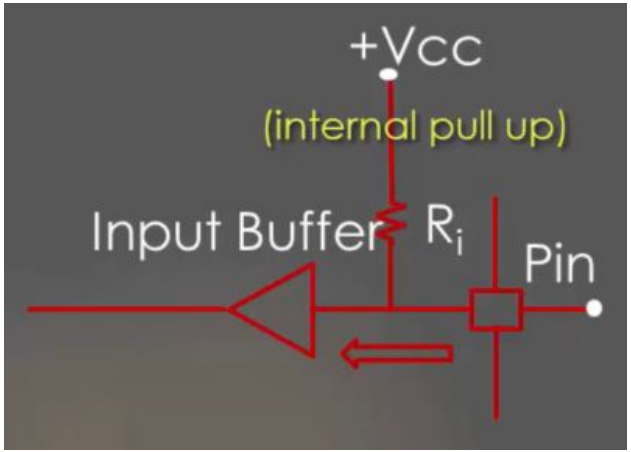


Figure 1. GPIO input mode with **Pull-up Resistor**

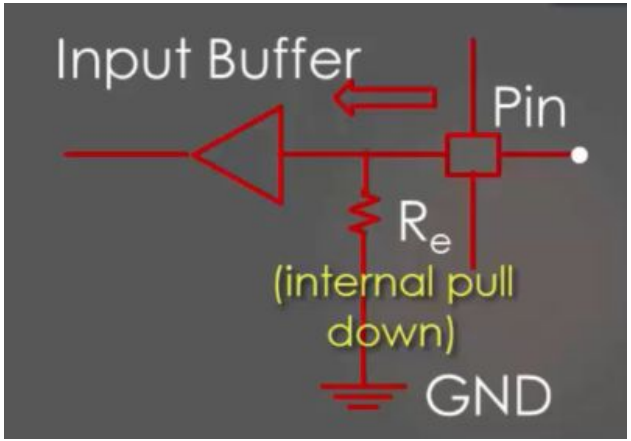
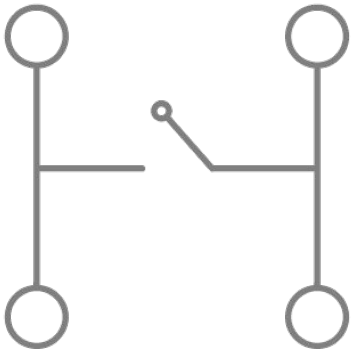


Figure 2. GPIO input mode with **Pull-down Resistor**

https://fastbitlab.com/gpio-input-mode-pull-pull-state/

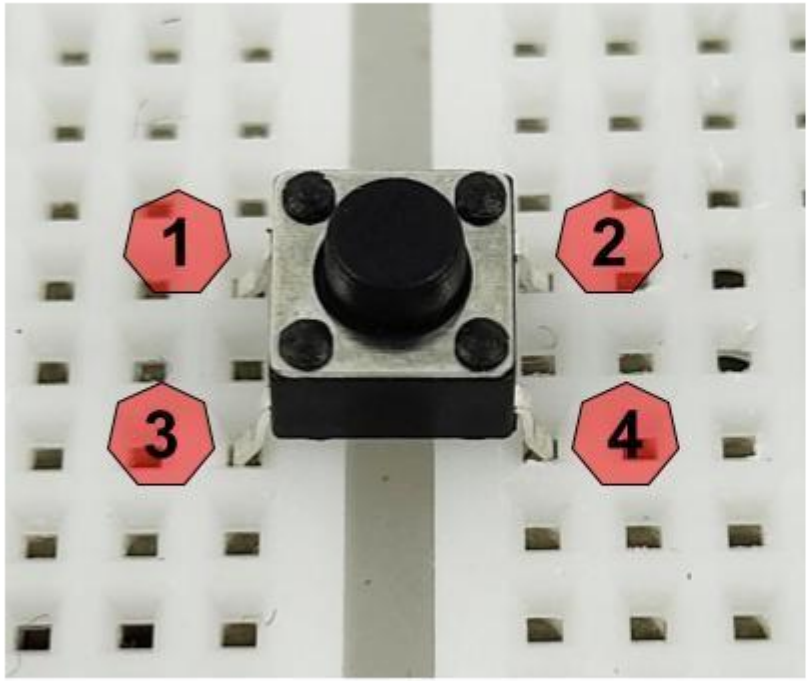
Switch / Button

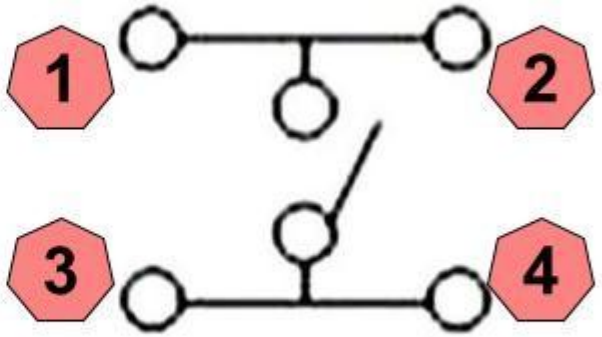
- A switch is a simple device used to interrupt the flow of electricity in a circuit.
- Can either connect (closed state) or disconnect (open state) the circuit, sending signals to the microcontroller.



https://www.freecodecamp.org/news/a-simple-explanation-of-pull-down-and-pull-up-resistors-660b308f116a/

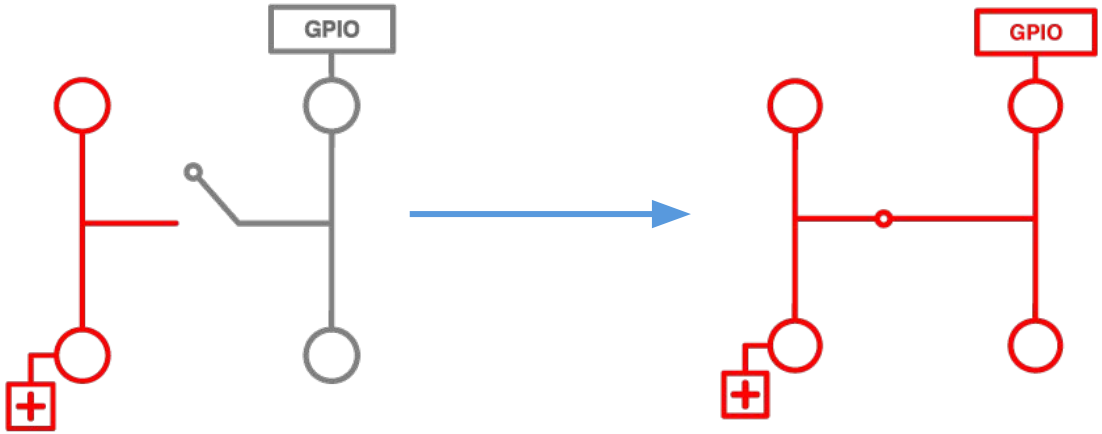
Switch / Button





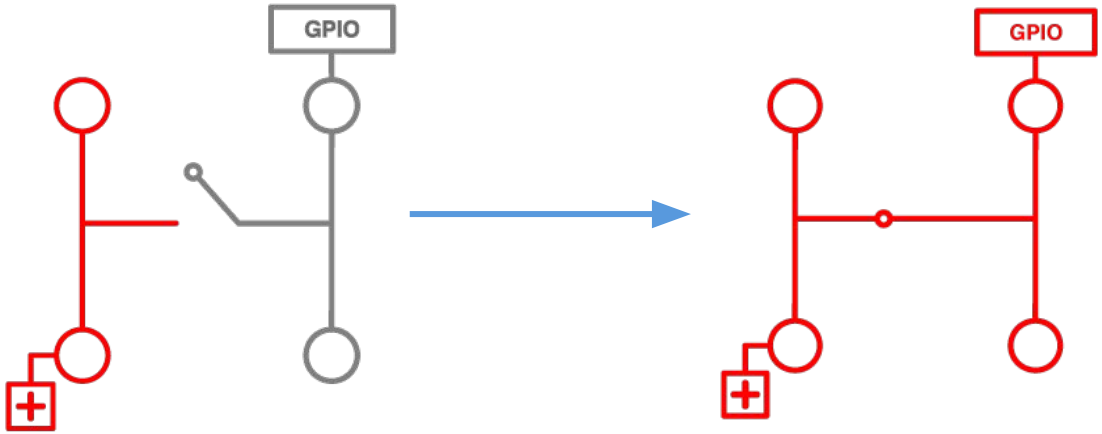
GPIO and Switch

- Connect positive (5v, 3.3V, or VCC) to the left side of the circuit.
- Now, when the button is pressed, the GPIO will read a 1, and all is good.



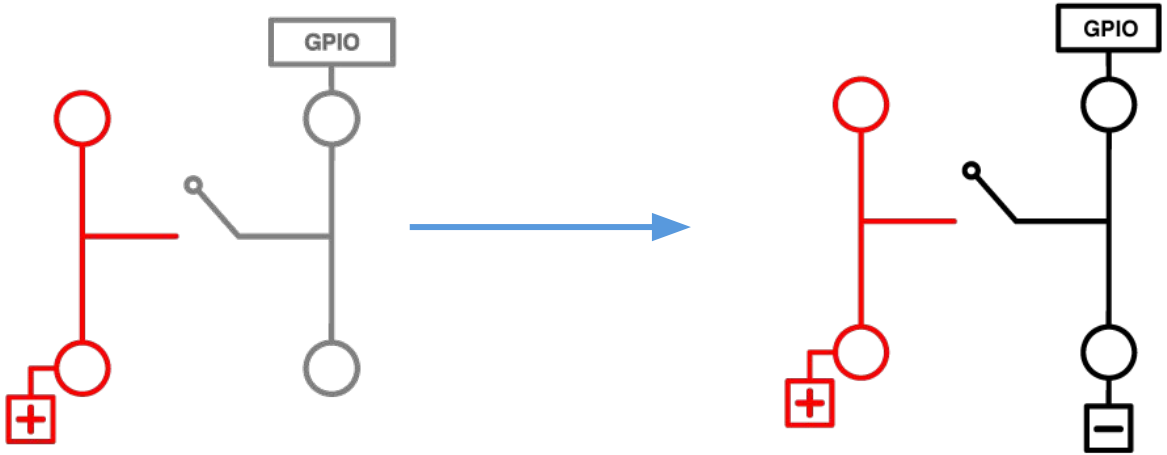
Ensuring a Logical 0

- Without a connection, electromagnetic interference can cause the GPIO to fluctuate, leading to unreliable readings.



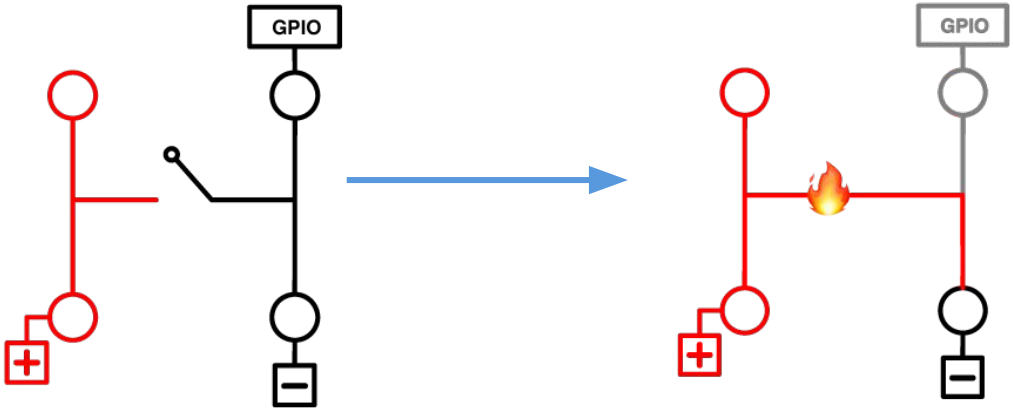
Ensuring a Logical 0

Connect the GPIO to GND to ensure the GPIO reads a logical 0 when the switch is not pressed.



Short Circuits

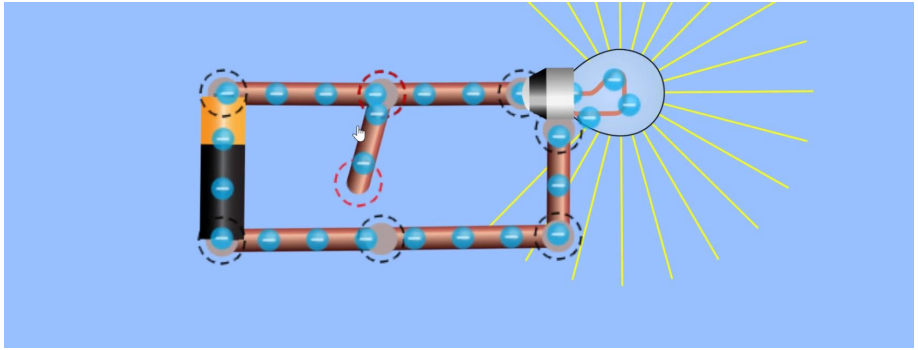
Electricity will take the path of least resistance.



What's a short circuit?

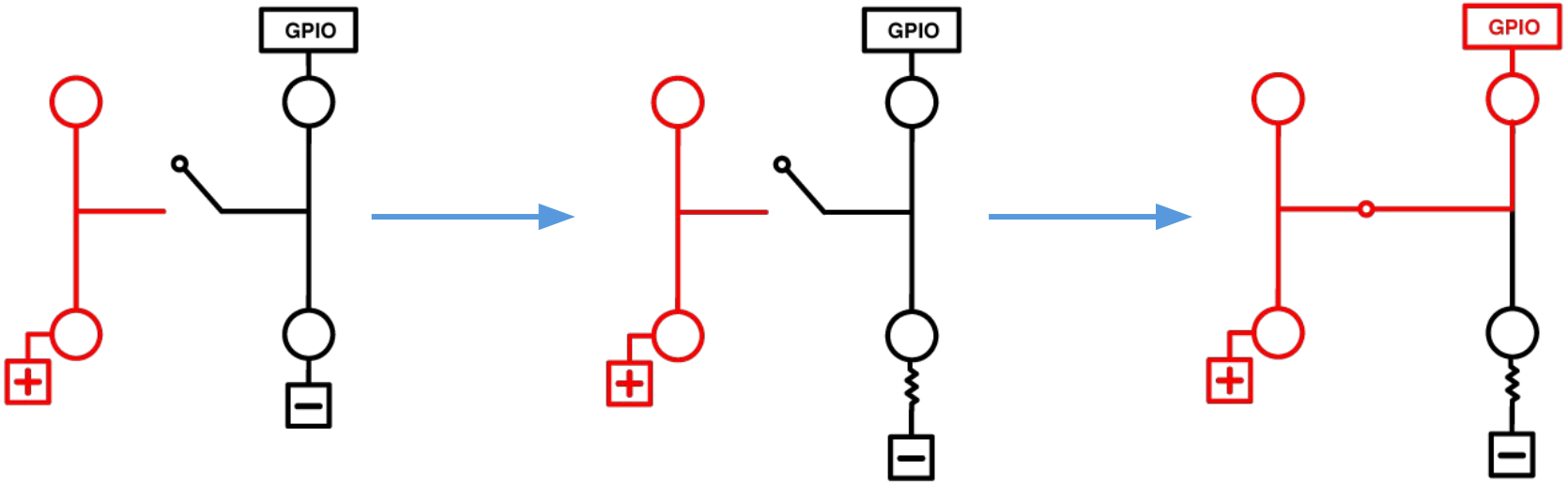
- A zero-resistance shortcut between two sides of a power source, like touching the positive and negative terminals of a battery.
- Can cause excessive current, burn out components, start fires, or even cause explosions.
- Ensure circuits are unpowered while building to avoid accidental shorts.

Circuit Construction Kit: DC - Virtual Lab



Pull Down Resistor

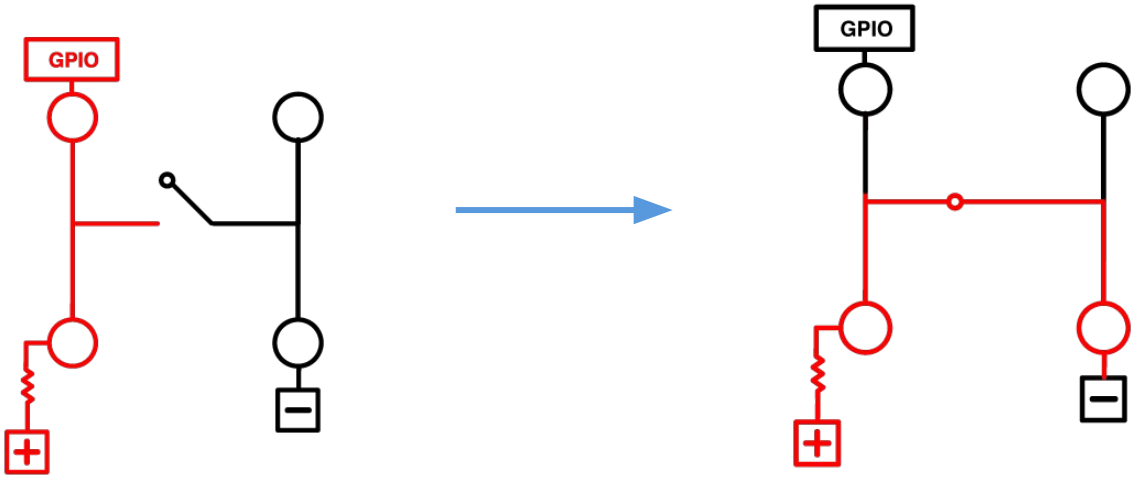
Add a resistor to prevent short circuits, allowing the GPIO to read a logical 1 when the button is pressed.



Pull Up Resistor

Connect the GPIO to VCC.

GPIO reads 1 when the button is not pressed and 0 when pressed, as current flows to ground when the button is pressed.



Pull-Up vs Pull-Down

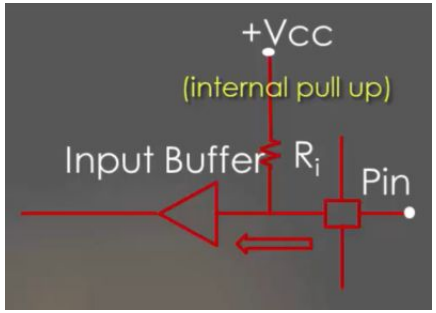


Figure 1. GPIO input mode with **Pull-up Resistor**

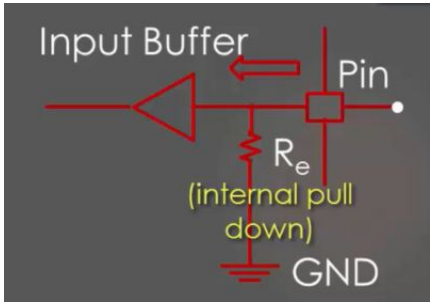


Figure 2. GPIO input mode with **Pull-down Resistor**

Connects GPIO pin to a high voltage (Vcc) through a resistor.

Ensures the pin reads a high state when not driven by an external signal.

Connects GPIO pin to ground (GND) through a resistor.

Ensures the pin reads a low state when not driven by an external signal.

Lab

2

Communication

I²C (Inter-Integrated Circuit)

I²C (Inter-Integrated Circuit) is a

- Synchronous
- multi-master
- multi-slave
- packet switched
- single-ended

Serial computer bus invented in 1982 by Philips Semiconductor

It is widely used for attaching lower-speed peripheral ICs to processors and microcontrollers in short-distance, intra-board communication.

I²C (Inter-Integrated Circuit)

A particular strength of I²C is the capability of a microcontroller to control a network of device chips with just two general-purpose I/O pins and software.

I²C uses only two bidirectional open-drain lines, Serial Data Line (SDA) and Serial Clock Line (SCL), pulled up with resistors.

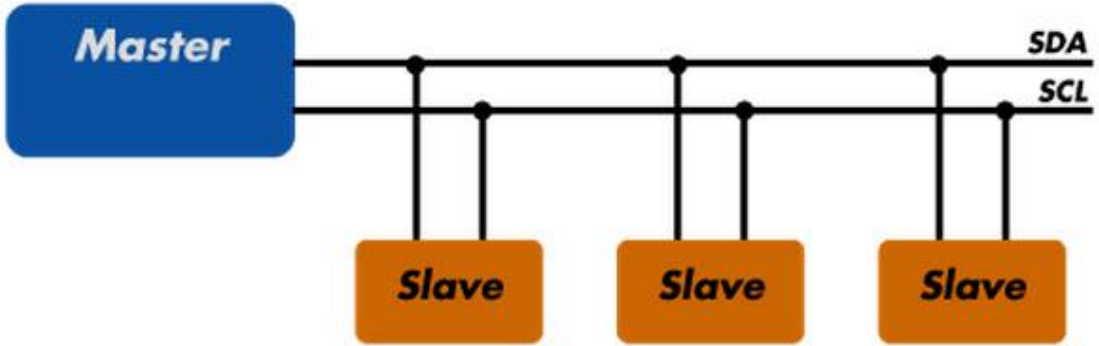
Many other bus technologies used in similar applications, such as Serial Peripheral Interface Bus, require more pins and signals to connect multiple devices.

Typical voltages used are +5 V or +3.3 V

I²C (Inter-Integrated Circuit)

The I2C bus drivers are “open drain”, meaning that they can pull the corresponding signal line low, but cannot drive it high. Thus, there can be no bus contention where one device is trying to drive the line high while another tries to pull it low, eliminating the potential for damage to the drivers or excessive power dissipation in the system.

Each signal line has a pull-up resistor on it, to restore the signal to high when no device is asserting it low.

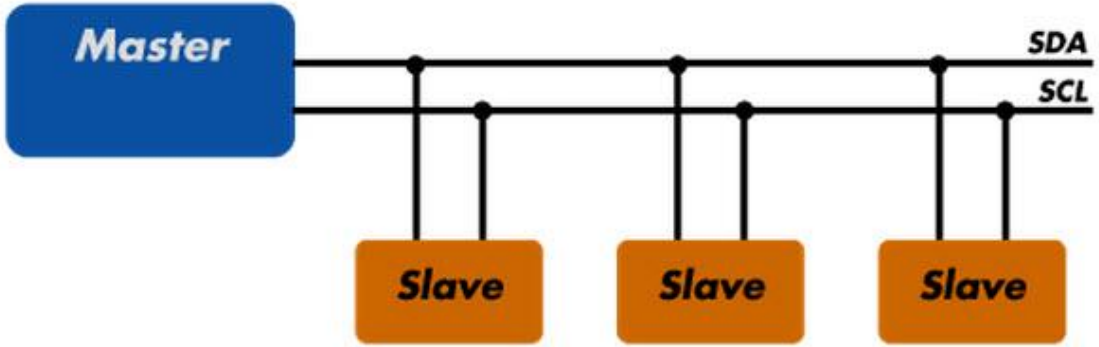


Architecture

The bus has two roles for nodes: master and slave:

- Master node – node that generates the clock and initiates communication with slaves.
- Slave node – node that receives the clock and responds when addressed by the master.

The bus is a multi-master bus, which means that any number of master nodes can be present.



Advantages & Disadvantages

Advantages

- Only uses two wires.
- Supports multiple masters and multiple slaves.
- ACK/NACK bit gives confirmation that each frame is transferred successfully.
- Hardware is less complicated than with UARTs.
- Well known and widely used protocol.

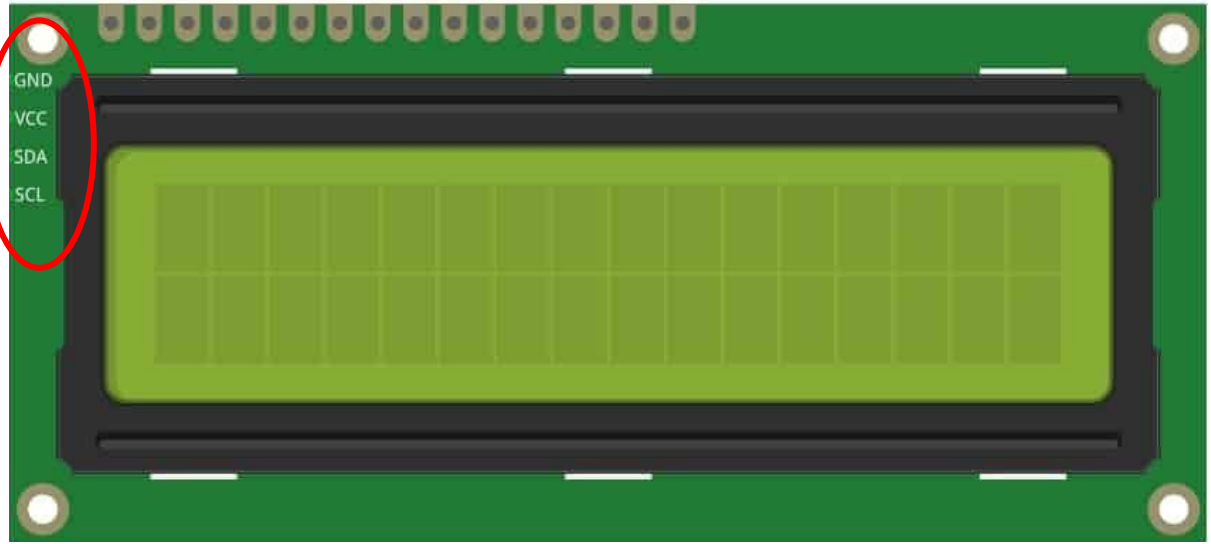
Disadvantages

- Slower data transfer rate than SPI.
- The size of the data frame is limited to 8 bits.
- More complicated hardware needed to implement than SPI.

Applications

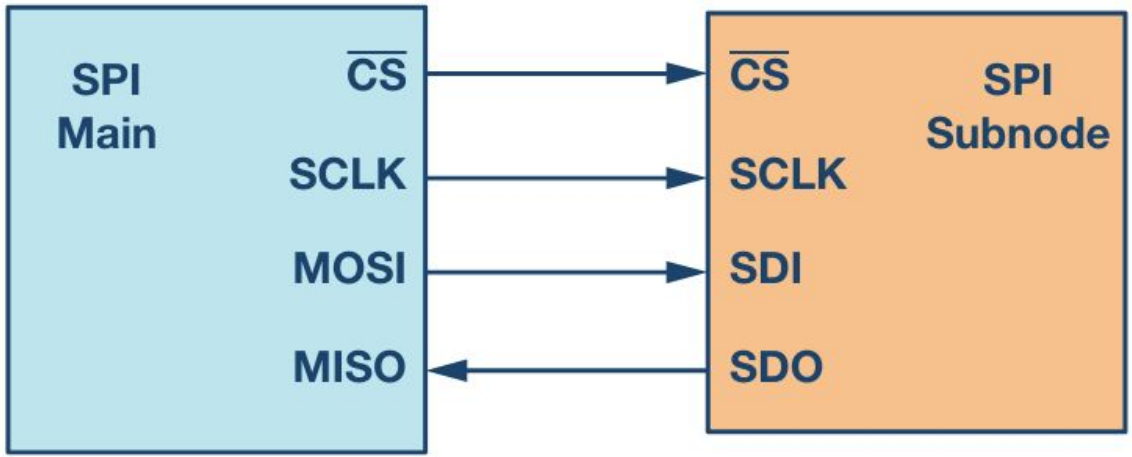
We'll probably find using I2C if we ever build projects that use in these modules.

- OLED displays
- Gyroscope
- Accelerometer



Components of SPI

- MOSI (Master Out Slave In): Carries data from master to slave.
- MISO (Master In Slave Out): Carries data from slave to master.
- SCLK (Serial Clock): Clock signal generated by the master.
- SS (Slave Select) / CS (Chip Select): Selects the slave device for communication.

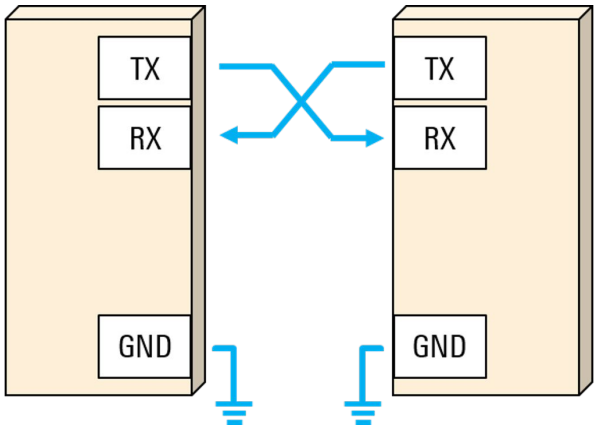


UART

UART stands for **universal asynchronous receiver / transmitter** and defines a protocol, or set of rules, for exchanging serial data between two devices.

It is very simple and only uses two wires between transmitter and receiver to transmit and receive in both directions. Both ends also have a ground connection.

- **implex:** Data is sent in one direction only
- **half-duplex:** Each side speaks but only one at a time
- **full-duplex:** Both sides can transmit simultaneously



UART Asynchronous

One of the big advantages of UART is that it is asynchronous – the transmitter and receiver do not share a common clock signal.

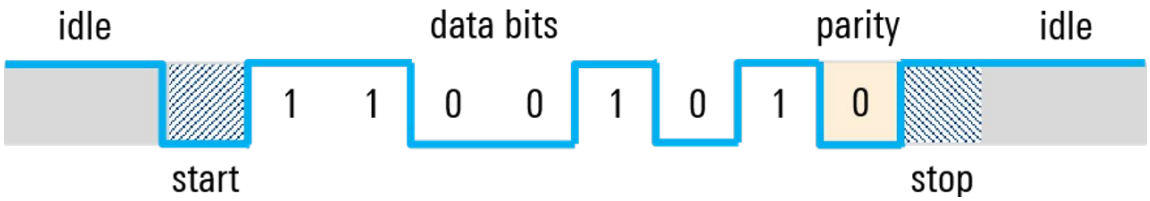
Although this greatly simplifies the protocol, it does place certain requirements on the transmitter and receiver. Since they do not share a clock, both ends must transmit at the same, pre-arranged speed in order to have the same bit timing. The most common UART baud rates in use today are 4800, 9600, 19.2K, 57.6K, and 115.2K.

In addition to having the same baud rate, both sides of a UART connection also must use the same frame structure and parameters.

UART Data Frame

As with most digital systems, a “high” voltage level is used to indicate a logical “1” and a “low” voltage level is used to indicate a logical “0”.

Since the UART protocol doesn’t define specific voltages or voltage ranges for these levels, sometimes high is also called “mark” while low is called “space”. In the idle state where no data is being transmitted the line is held high. This allows an easy detection of a damaged line or transmitter.

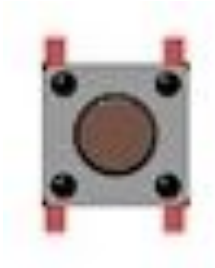


3

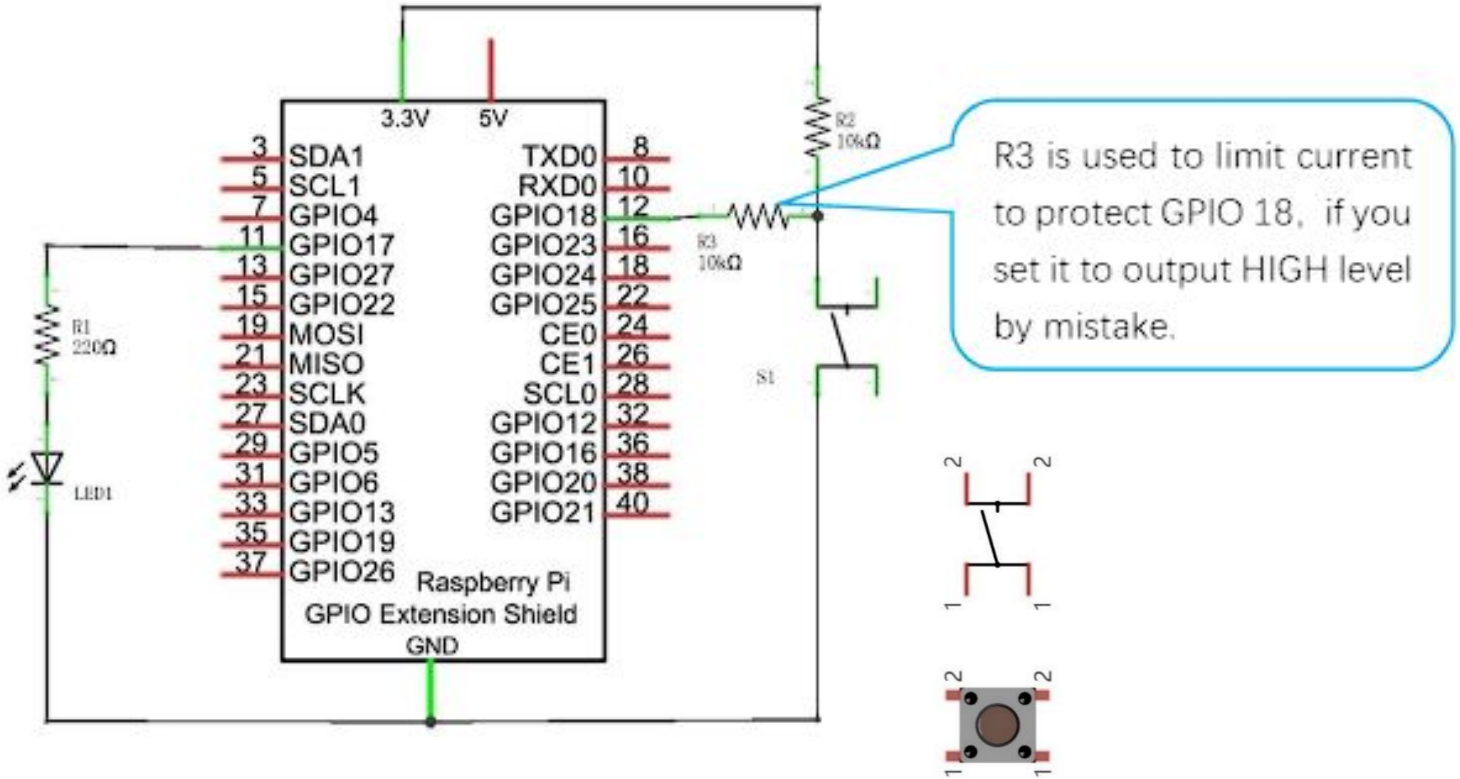
Lab

Push Button LED – Components

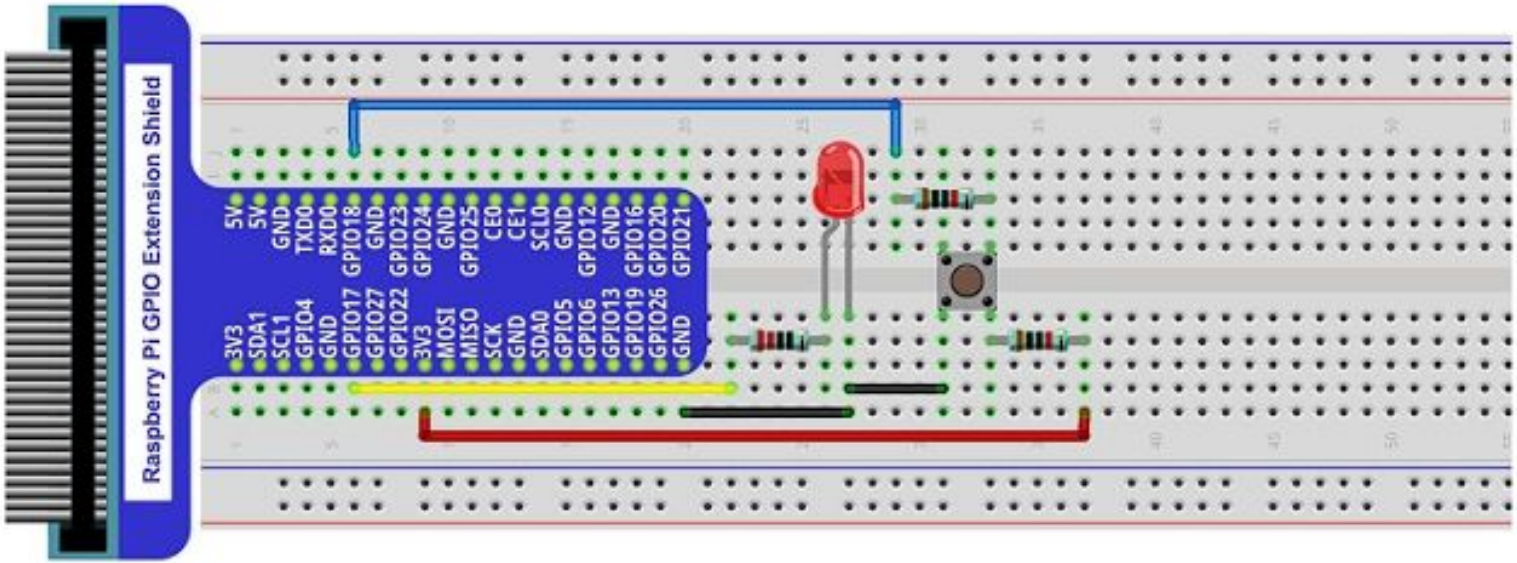
- LED * 1
- Resistor 220Ω * 1
- Resistor 10kΩ * 2
- Push Button Switch * 1
- Jumper Wires



Push Button LED – Schematic diagram



Push Button LED – Circuit



Lamp

we still detect the state of Push Button Switch to control an LED.

Here we need to define a variable to define the state of LED.

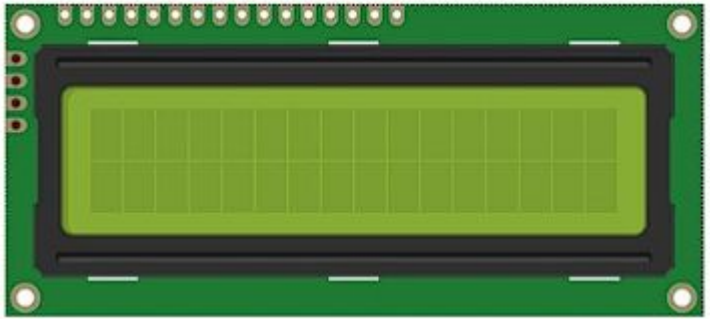
When the button switch is pressed once, the state of LED will be changed once.

This will allow the circuit to act as a table lamp.

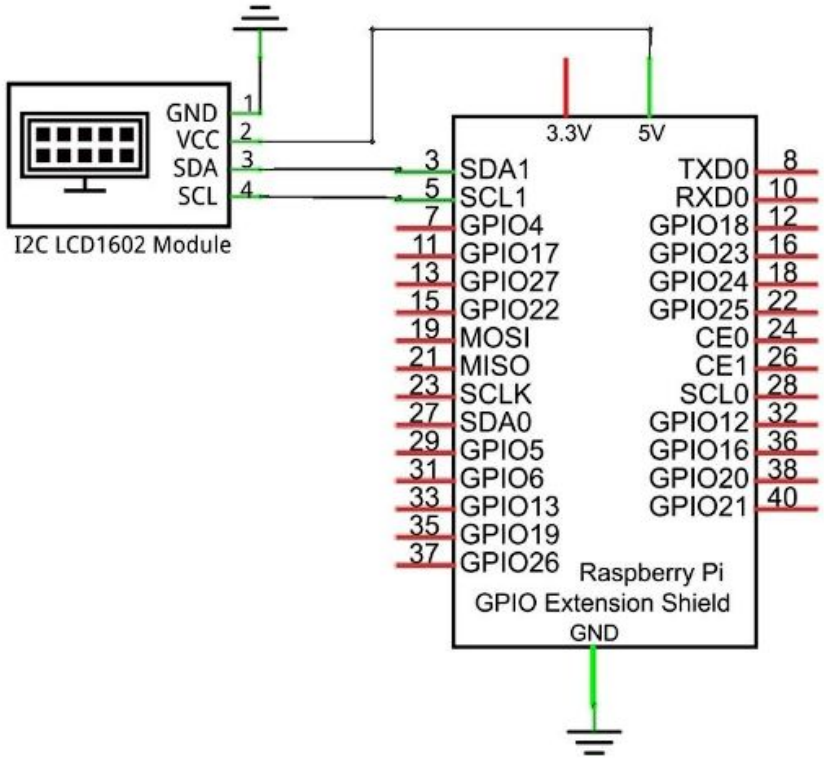
https://github.com/jjia131/ASTEAM-2025-Lab-Resources

I2C LCD - Components

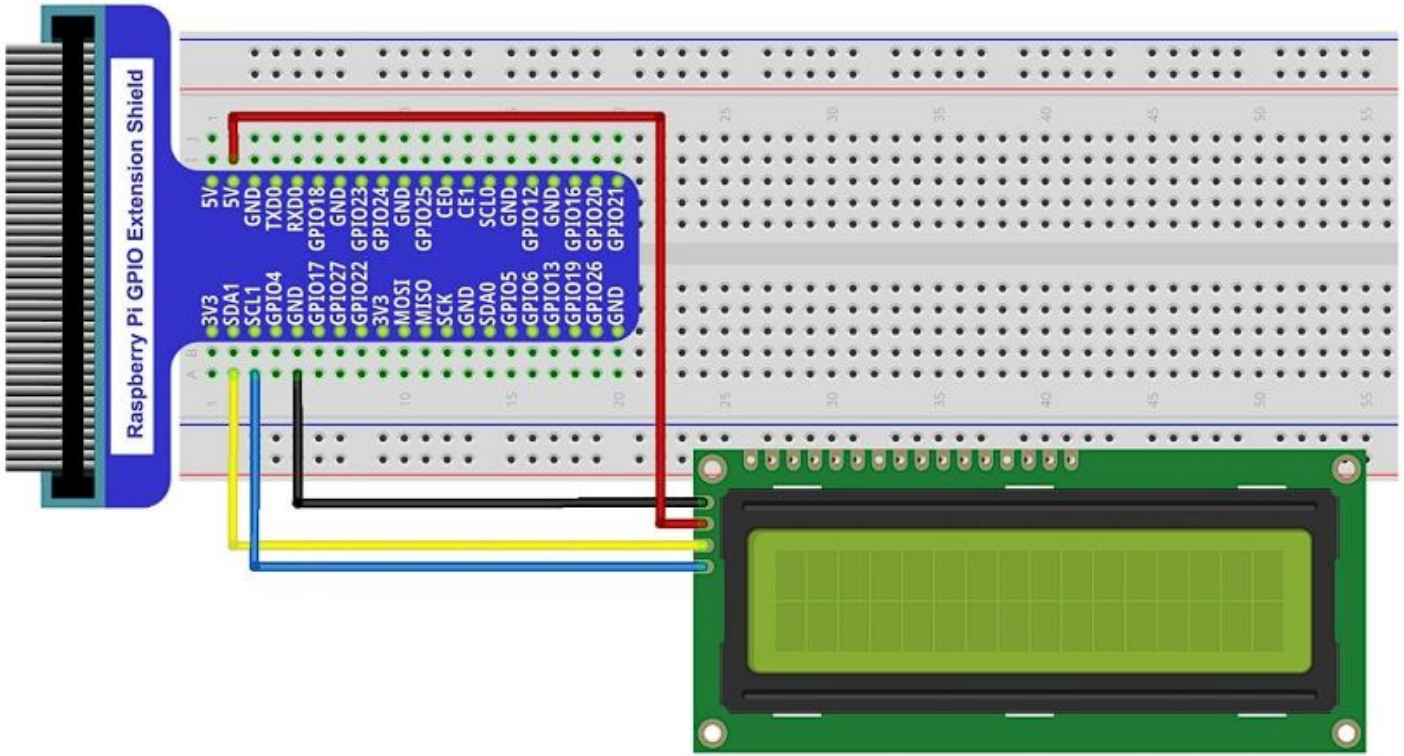
- I2C LCD1602 * 1
- Jumper Wires



I2C LCD - Schematic diagram

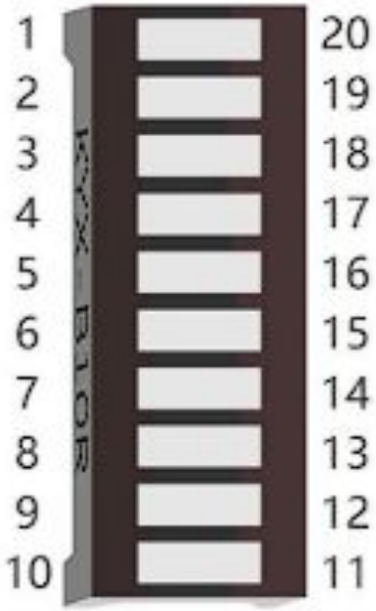


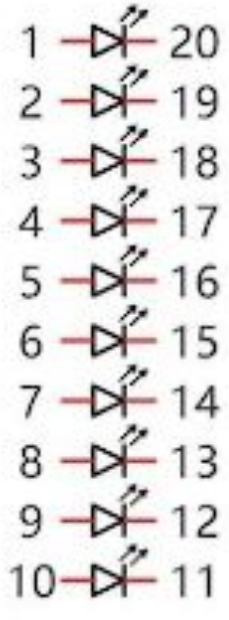
I2C LCD – Circuit



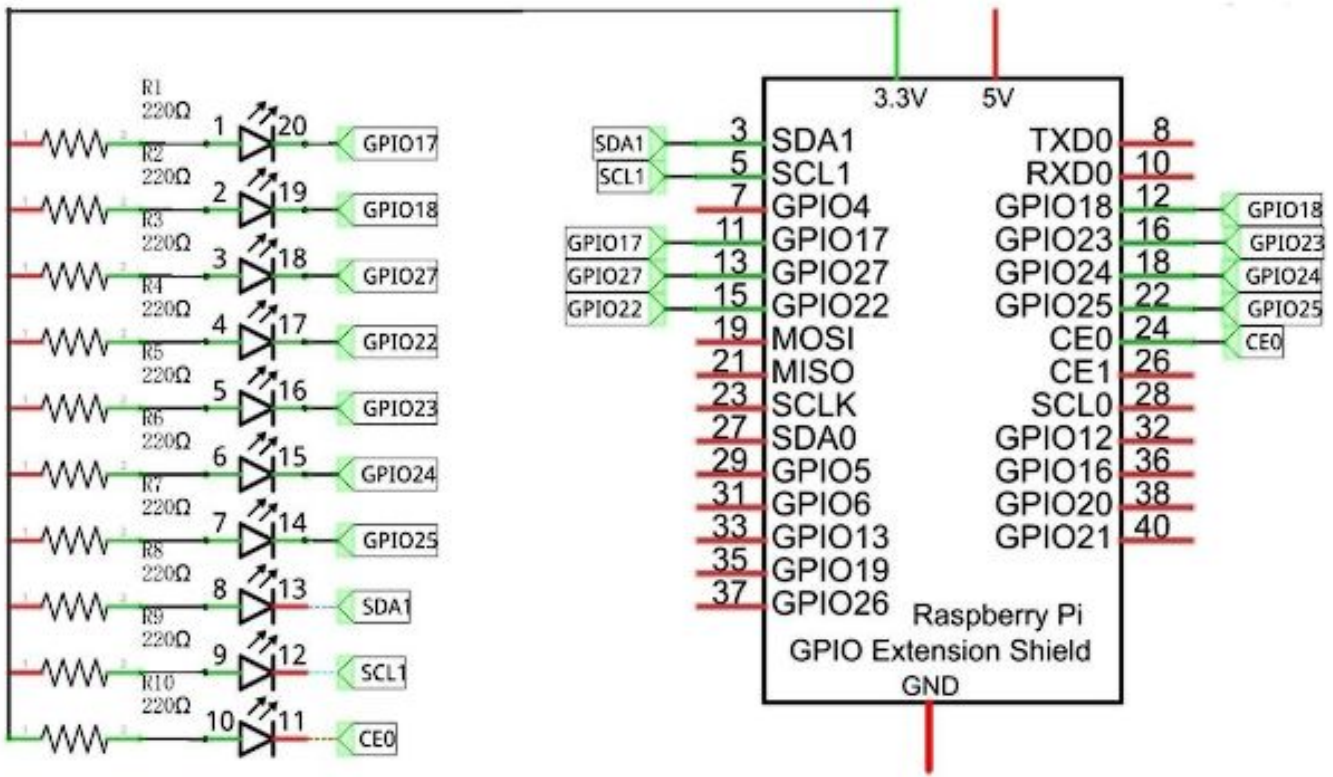
Flowing Water Light – Components

- Bar Graph LED * 1
- Resistor 220Ω * 10
- Jumper Wires

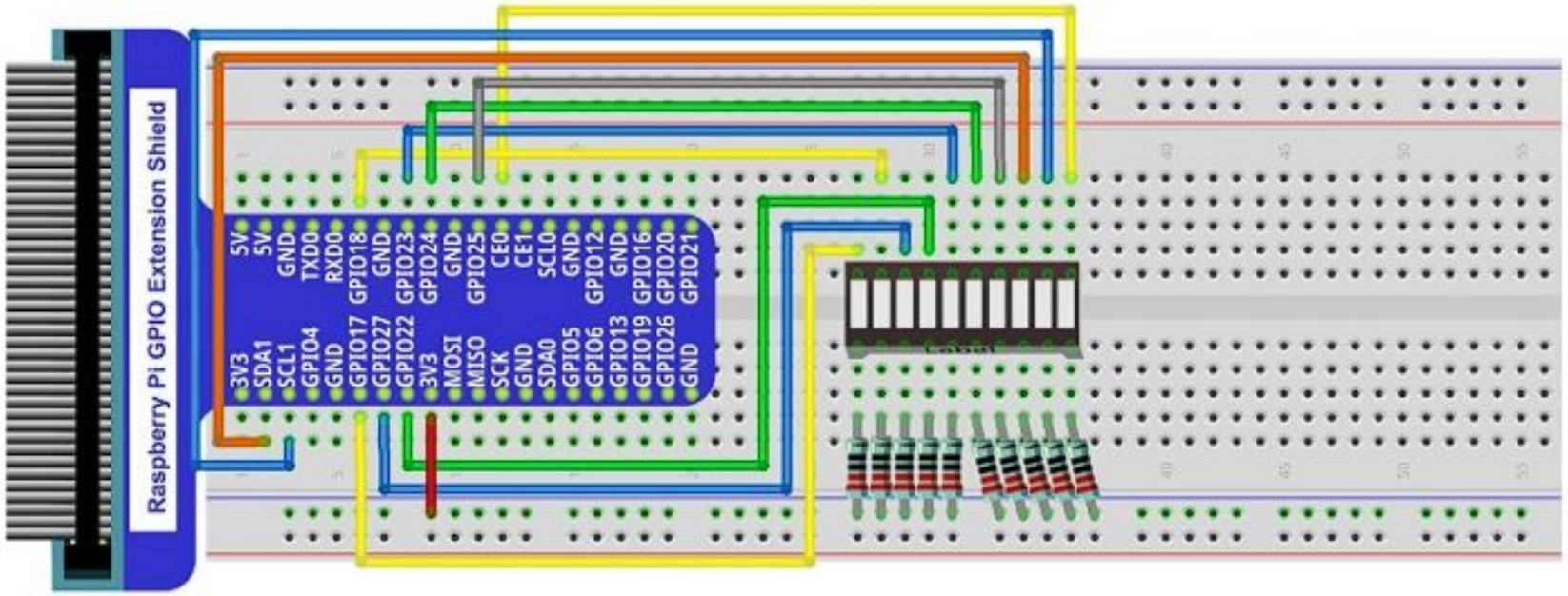




Flowing Water Light – Schematic diagram



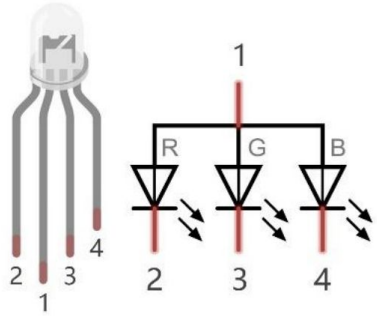
Flowing Water Light – Circuit

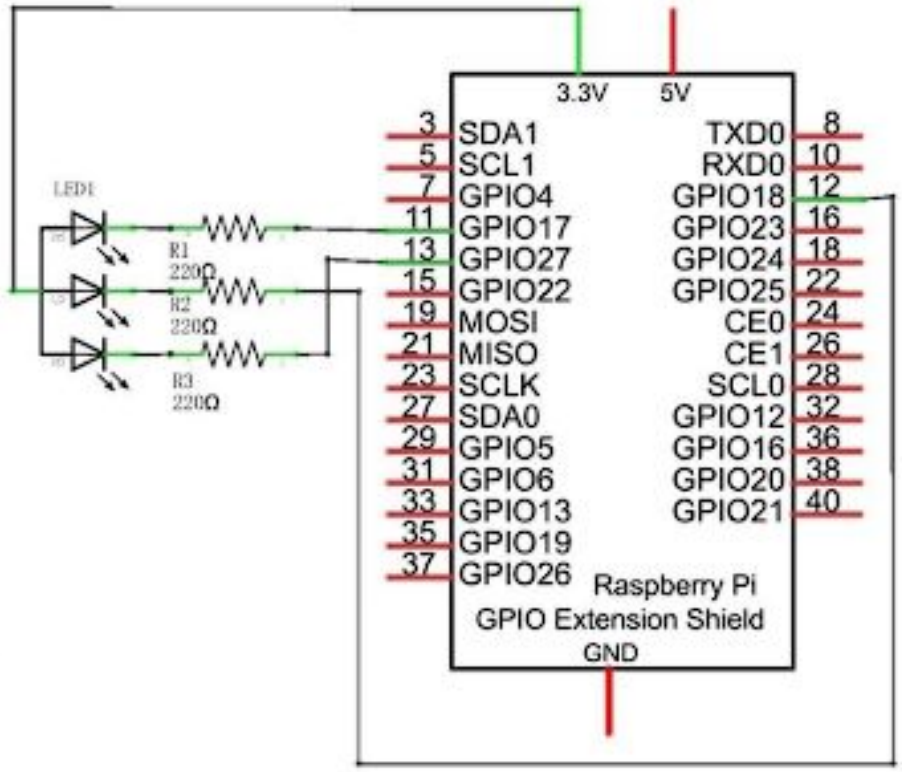


Multicolored LED - Components

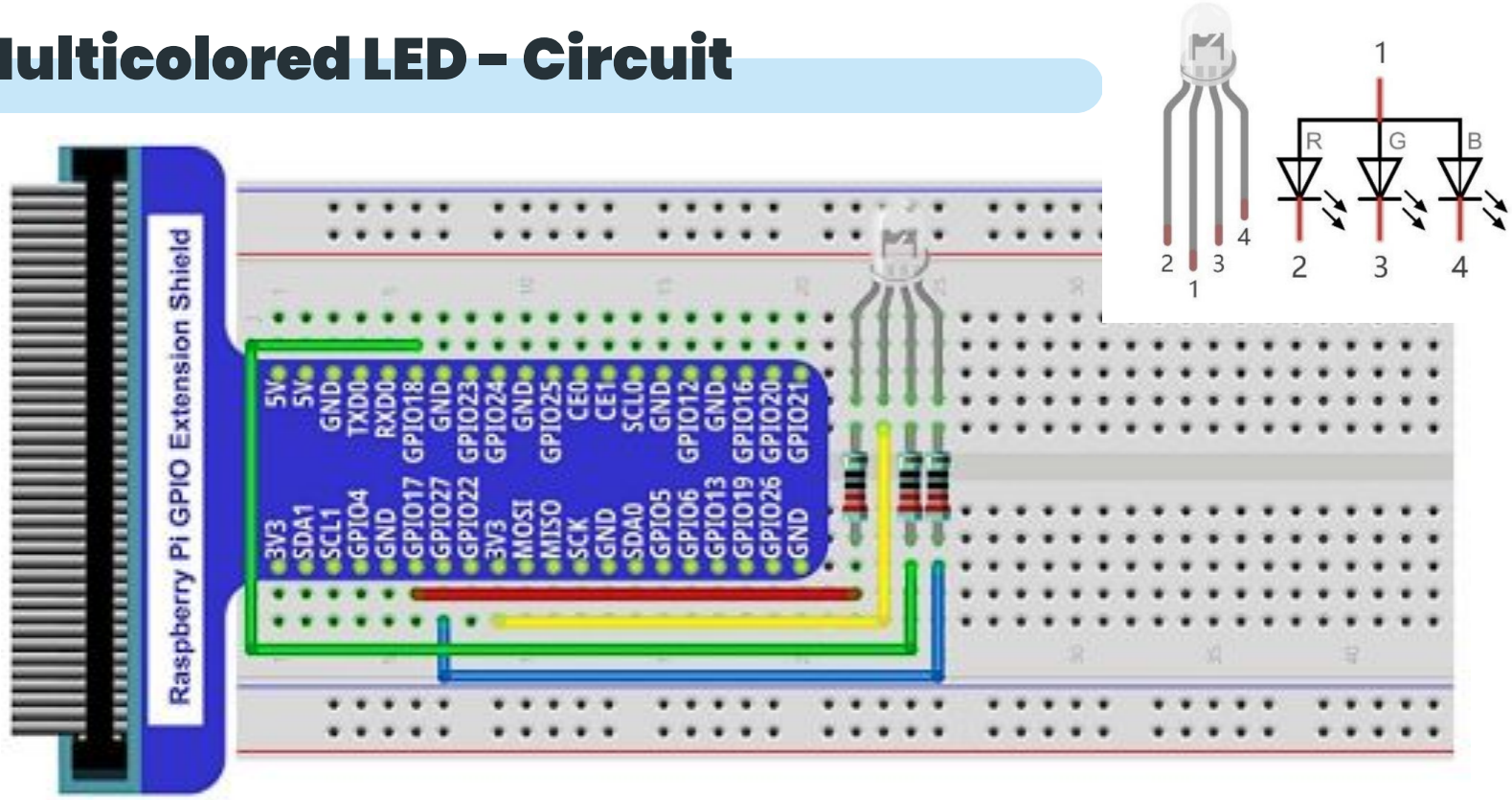
- RGB LED * 1
- Resistor 220Ω * 3
- Jumper Wires

Multicolored LED - Schematic diagram





Multicolored LED – Circuit



4

ADC

Digital Systems and Binary Numbers

Digital age and information age

Digital computers

- General purposes
- Many scientific, industrial and commercial applications

Digital systems

- Telephone switching exchanges
- Digital camera
- Digital TV

Discrete information-processing systems

- Manipulate discrete elements of information
- For example, {1, 2, 3, ...} and {A, B, C, ...}...

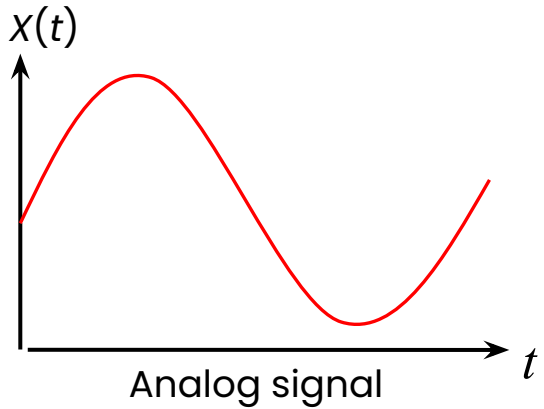
Analog and Digital Signal

Analog system

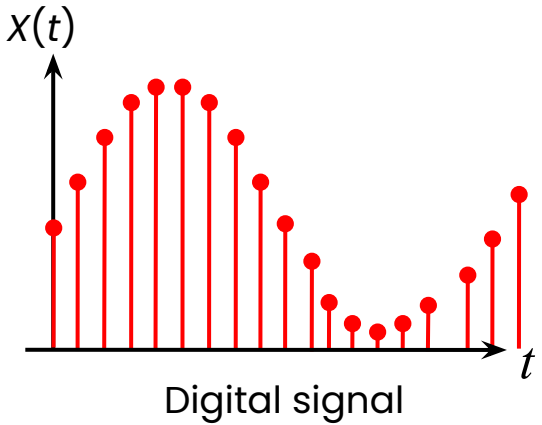
- The physical quantities or signals may vary continuously over a specified range.

Digital system

- The physical quantities or signals can assume only discrete values.
- Greater accuracy



Analog signal



Digital signal

Binary Digital Signal

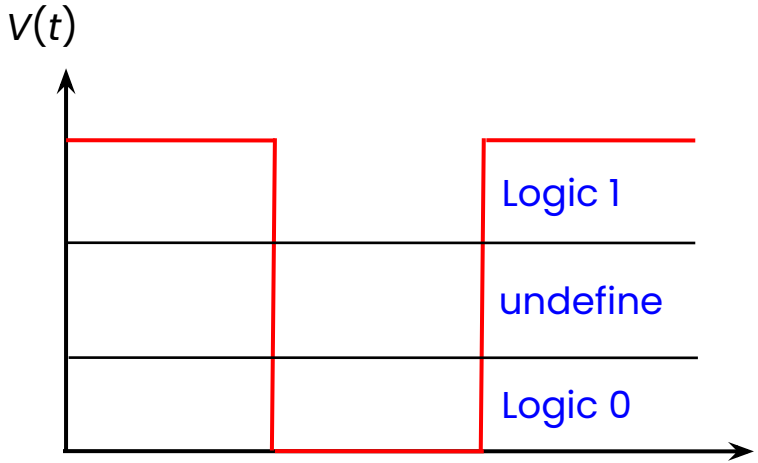
An information variable represented by physical quantity. For digital systems, the variable takes on discrete values.

- Two level, or binary values are the most prevalent values.

Binary values are represented abstractly by:

- Digits 0 and 1
- Words (symbols) False (F) and True (T)
- Words (symbols) Low (L) and High (H)
- Words (symbols) OFF and ON

Binary values are represented by values or ranges of values of physical quantities



Binary digital signal

Decimal Number System

Base (also called radix) = 10

- 10 digits { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }

Digit Position

- Integer & fraction

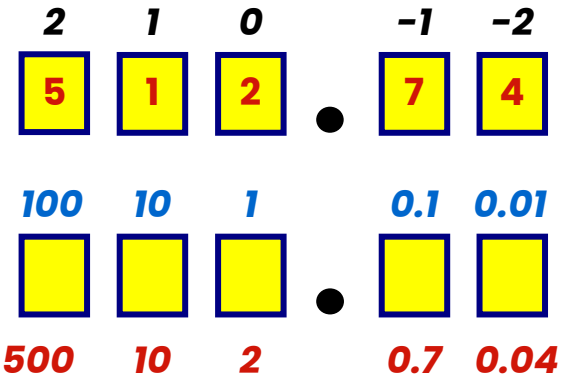
Digit Weight

- Weight = (*Base*) *Position*

Magnitude

- Sum of "*Digit* x *Weight*"

Formal Notation



$$d_2*B^2+d_1*B^1+d_0*B^0+d_{-1}*B^{-1}+d_{-2}*B^{-2}$$

$$(512.74)_{10}$$

Binary Number System

Base = 2

- 2 digits { 0, 1 }, called *b*inary dig*its* or "*bits*"

Weights

- Weight = (*Base*) *Position*

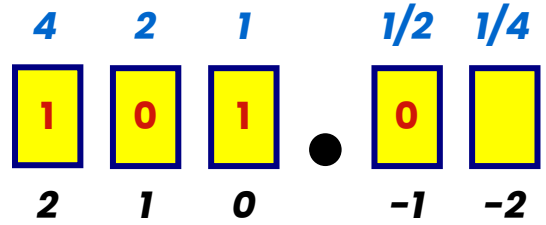
Magnitude

- Sum of "*Bit* x *Weight*"

Formal Notation

Groups of bits 4 bits = *Nibble*

8 bits = *Byte*



$1*2^{2}+0*2^{1}+1*2^{0}+0*2^{-1}+1*2^{-2}$

$=(5.25)_{10}$

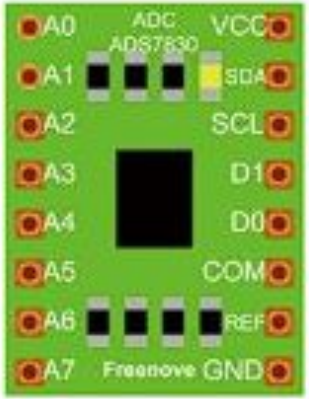
$(101.01)_{2}$

1011

11000101

Analog Digital Converter - Components

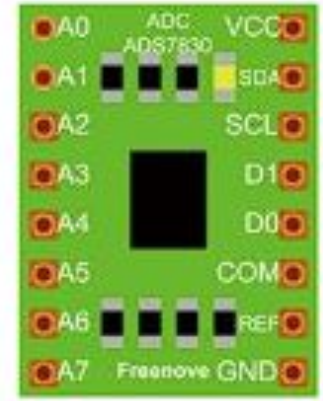
- Rotary Potentiometer * 1
- ADC Module * 1
- Jumper Wires



Analog Digital Converter

An ADC is an electronic integrated circuit used to convert analog signals such as voltages to digital or binary form consisting of 1s and 0s. The range of our ADC module is 8 bits, that means the resolution is 2^8=256, so that its range (at 3.3V) will be divided equally to 256 parts.

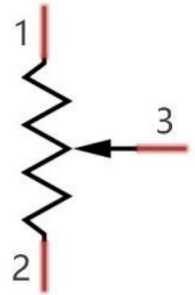
Any analog value can be mapped to one digital value using the resolution of the converter. So the more bits the ADC has, the denser the partition of analog will be and the greater the precision of the resulting conversion.



Rotary Potentiometer

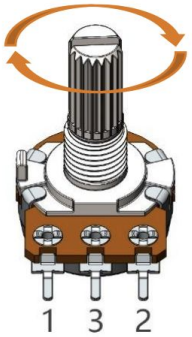
Potentiometer is a resistive element with three Terminal parts. Unlike the resistors that have a fixed resistance value, the resistance value of a potentiometer can be adjusted. A potentiometer is often made up by a resistive substance (a wire or carbon element) and movable contact brush. When the brush moves along the resistor element, there will be a change in the resistance of the potentiometer output side (3) (or change in the voltage of the circuit that is a part).

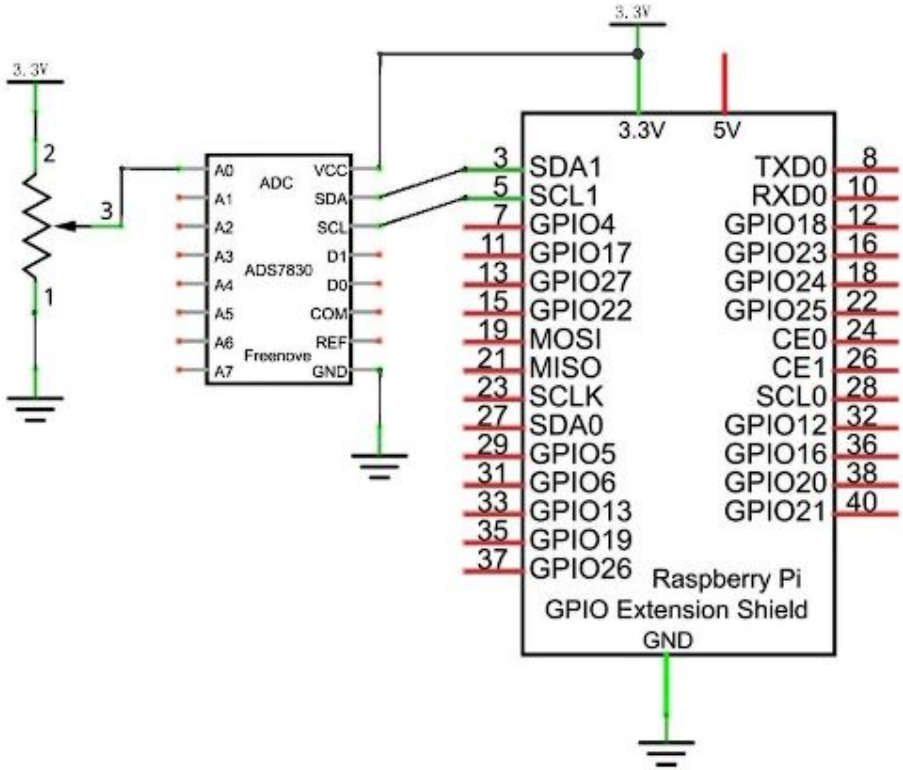
Between potentiometer pin 1 and pin 2 is the resistive element (a resistance wire or carbon) and pin 3 is connected to the brush that makes contact with the resistive element.



https://github.com/jjia131/ASTEAM-2025-Lab-Resources

ADC - Schematic diagram





https://github.com/jjia131/ASTEAM-2025-Lab-Resources

sudo apt-get install python-smbus

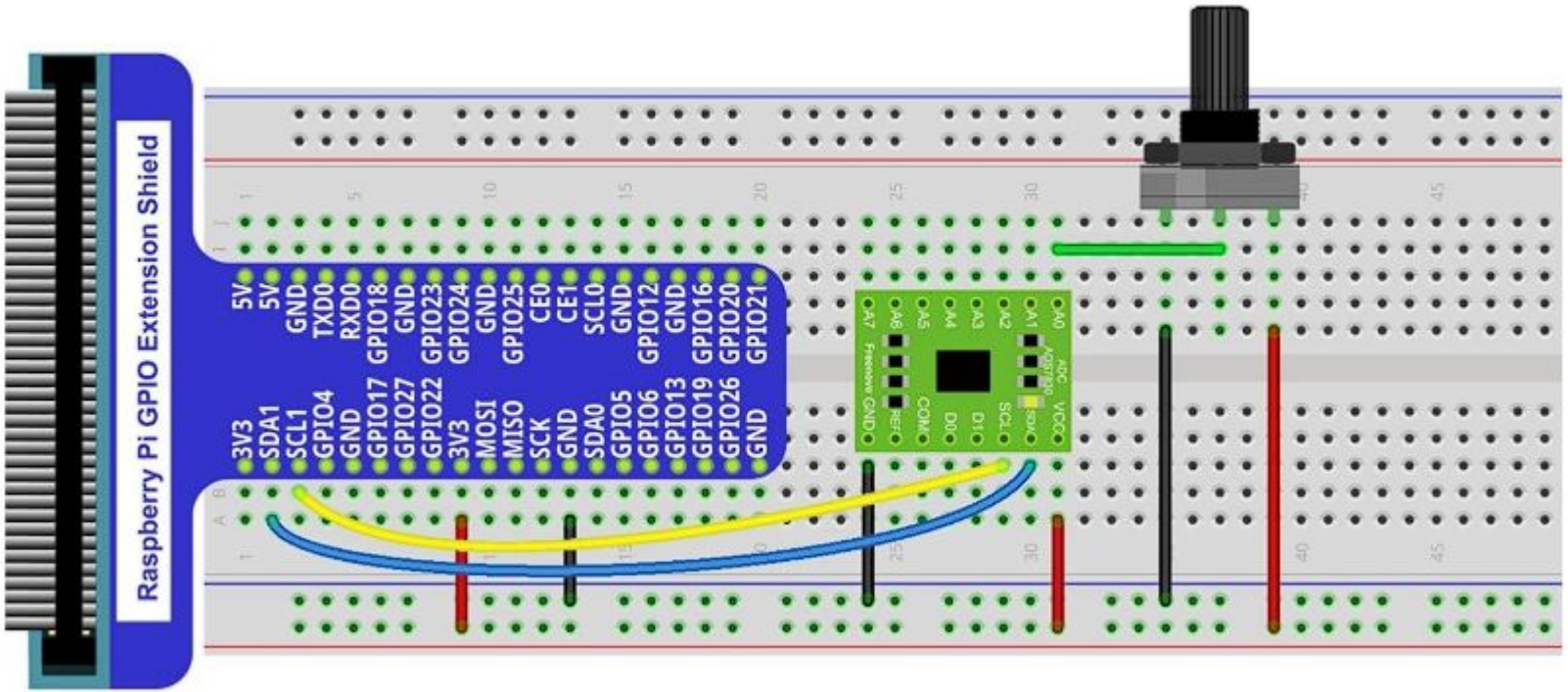
sudo apt-get install python3-smbus

tar -zxvf ADCDevice-1.0.3.tar.gz

cd ADCDevice-1.0.3

sudo python3 setup.py install

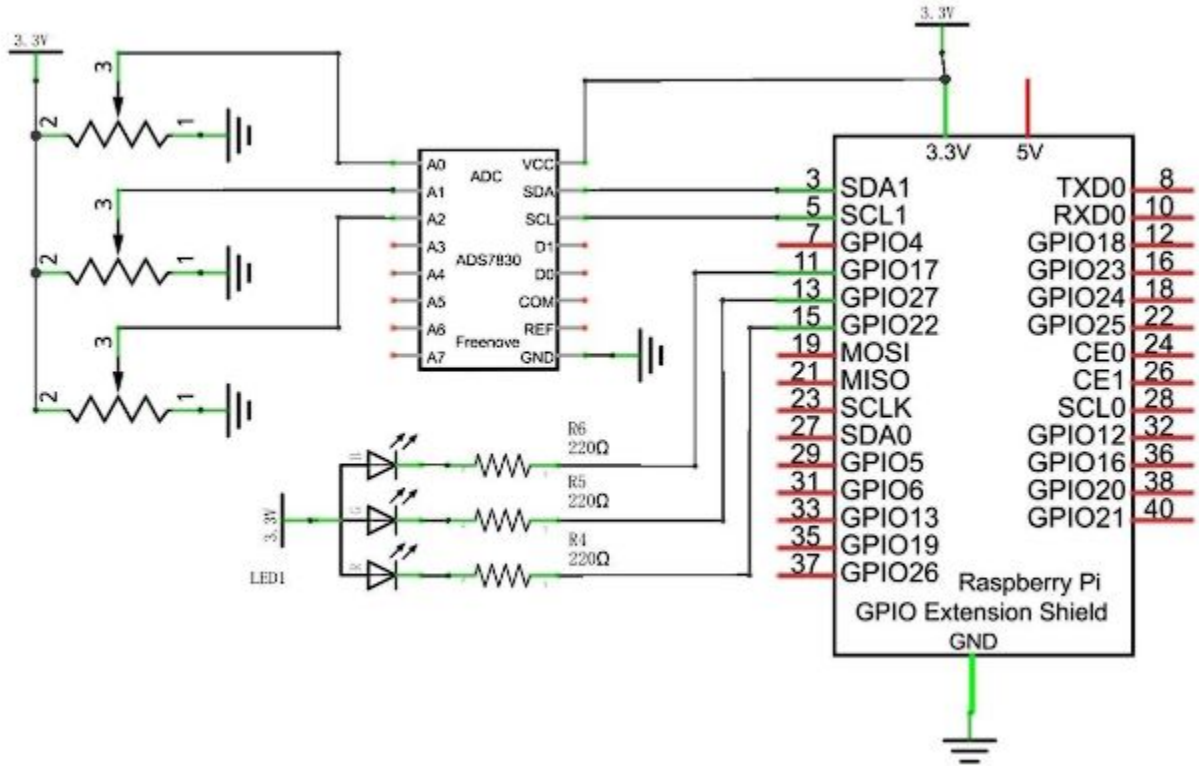
ADC - Circuit


Raspberry Pi GPIO Extension Shield
5V
5V
GND
TXD0
RXD0
GPIO18
GND
GPIO23
GPIO24
GND
GPIO25
CE0
CE1
SCL0
GND
GPIO12
GND
GPIO16
GPIO20
GPIO21
3V3
SDA1
SCL1
GPIO4
GND
GPIO17
GPIO27
GPIO22
3V3
MOSI
MISO
SCK
GND
SDA0
GPIO5
GPIO6
GPIO13
GPIO19
GPIO26
GND
A7
A6
A5
A4
A3
A2
A1
A0
Freenove GND
REF
COM
D0
D1
SCL
SDA
VCC
ADC
ADS7830

Colorful Light - Components

- Rotary Potentiometer * 3
- ADC Module * 1
- RGB LED * 1
- Resistor 220Ω * 3
- Jumper Wires

Colorful Light - Schematic diagram



Colorful Light – Circuit