**Week2**

**2)a) Printing your biodata on the screen**

**Program:**

# Define a function 'personal\_details'.

def personal\_details():

# Define variables 'name' and 'age' and assign values to them.

name, age = "Simon", 19

# Define a variable 'address' and assign a value to it.

address = "Bangalore, Karnataka, India"

# Print the personal details using string formatting.

print("Name: {}\nAge: {}\nAddress: {}".format(name, age, address))

# Call the 'personal\_details' function to display the details.

personal\_details()

**Output:**

Name: Simon

Age: 19

Address: Bangalore, Karnataka, India

**2)b) Printing all the primes less than a given number**

**Program:**

def is\_prime(MyNum):

if MyNum < 2:

return False

for i in range(2, MyNum // 2 + 1):

if MyNum % i == 0:

return False

return True

def print\_primes\_less\_than(x):

print(f"Prime numbers less than {x} are:")

for i in range(2, x):

if is\_prime(i):

print(i, end=" ")

# Call the function to print primes less than 50

print\_primes\_less\_than(90)

**Output:**

Prime numbers less than 90 are:

2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89

**2)c) Finding all the factors of a number and show whether it is a perfect number, i.e., the sum of all its factors (excluding the number itself) is equal to the number itself.**

**Program:**

n = int(input("Enter any number: "))

sum1 = 0

for i in range(1, n):

if(n % i == 0):

sum1 = sum1 + i

if (sum1 == n):

print("The number is a Perfect number!")

else:

print("The number is not a Perfect number!")

**Output:**

Enter any number: 6

The number is a Perfect number!

**Week3**

**1)a) Write a function to read data from a file and display it on the screen**

**Program:**

import os

filename = input("Enter the name of the file with .txt extension: ")

if os.path.exists(filename):

with open(filename, 'r') as file:

for line in file:

print(line, end='')

else:

print("The file was not found. Please check the name and try again.")

**Output:**

Enter the name of the file with .txt extension: word.txt

Here's a simple and clean Python function that reads data from a file and displays it on the screen:

**1)b) Define a boolean function is palindrome(<input>)**

**Program:**

# Define a function

def isPalindrome(string):

if string == string[::-1]:

return "The string is a palindrome."

else:

return "The string is not a palindrome."

# Enter input string

string = input("Enter string: ")

print(isPalindrome(string))

**Output:**

Enter string: noon

The string is a palindrome.

**1) c) Write a function collatz(x) which does the following: if x is odd, x = 3x + 1; if x is even, then x = x/2. Return the number of steps it takes for x = 1**

**Program:**

def collatz(x):

steps = 0

while x != 1:

if x % 2 == 0:

x = x // 2 # Use integer division to avoid float

else:

x = 3 \* x + 1

steps += 1

return steps

n = int(input("Enter a number: "))

print(f"It took {collatz(n)} steps to reach 1.")

**Output:**

Enter a number: 12

It took 9 steps to reach 1.

**1)d) Write a function N(m, s) = exp(-(x-m)2/(2s2))/sqrt(2π)s that computes the Normal distribution.**

**Program:**

import numpy as np

def normal\_dist(x, mean, sd):

prob\_density = (1 / (np.sqrt(2 \* np.pi) \* sd)) \* np.exp(-0.5 \* ((x - mean) / sd) \*\* 2)

return prob\_density

# Example values

mean = 0

sd = 1

x = 1

result = normal\_dist(x, mean, sd)

print(result)

**Output:**

0.24197072451914337

**Week4**

**1. The package numpy**

**a) Creating a matrix of given order m x n containing random numbers in the range 1 to 99999**

**Program:**

import numpy as np

def create\_random\_matrix(m, n):

matrix = np.random.randint(1, 100000, size=(m, n))

return matrix

# Example usage:

m = int(input("Enter number of rows (m): "))

n = int(input("Enter number of columns (n): "))

random\_matrix = create\_random\_matrix(m, n)

print("Generated Random Matrix:")

print(random\_matrix)

**Output:**

Enter number of rows (m): 2

Enter number of columns (n): 2

Generated Random Matrix:

[[13024 78225]

[97000 46569]]

**1)b) Write a program that adds, subtracts and multiplies two matrices. Provide an interface such that, based on the prompt, the function (addition, subtraction, multiplication) should be performed**

**Program:**

**#Addition**

# importing numpy as np

import numpy as np

# creating first matrix

A = np.array([[1, 2], [3, 4]])

# creating second matrix

B = np.array([[4, 5], [6, 7]])

print("Printing elements of first matrix")

print(A)

print("Printing elements of second matrix")

print(B)

# adding two matrix

print("Addition of two matrix")

print(np.add(A, B))

#Substraction

# importing numpy as np

import numpy as np

# creating first matrix

A = np.array([[1, 2], [3, 4]])

# creating second matrix

B = np.array([[4, 5], [6, 7]])

print("Printing elements of first matrix")

print(A)

print("Printing elements of second matrix")

print(B)

# subtracting two matrix

print("Subtraction of two matrix")

print(np.subtract(A, B))

**# Program to multiply two matrices using nested loops**

# 3x3 matrix

X = [[12, 7, 3],

[4, 5, 6],

[7, 8, 9]]

# 3x3 matrix

Y = [[5, 8, 1, 2],

[6, 7, 3, 0],

[4, 5, 9, 1]]

# 3x4 matrix

# Result matrix (3x4) initialized to zero

result = [[0, 0, 0, 0],

[0, 0, 0, 0],

[0, 0, 0, 0]]

# Iterate through rows of X

for i in range(len(X)):

# Iterate through columns of Y

for j in range(len(Y[0])):

# Iterate through rows of Y

for k in range(len(Y)):

result[i][j] += X[i][k] \* Y[k][j]

# Print the result matrix

for r in result:

print(r)

**Output:**

**Printing elements of first matrix**

[[1 2]

**[3 4]]**

**Printing elements of second matrix**

[[4 5]

**[6 7]]**

**Addition of two matrix**

[[ 5 7]

**[ 9 11]]**

**Printing elements of first matrix**

[[1 2]

**[3 4]]**

**Printing elements of second matrix**

[[4 5]

**[6 7]]**

**Subtraction of two matrix**

[[-3 -3]

**[-3 -3]]**

[114, 160, 60, 27]

[74, 97, 73, 14]

[119, 157, 112, 23]

**1)c) Write a program to solve a system of n linear equations in n**

**Program:**

import numpy as np

# Initializing variables

x1 = 0

x2 = 0

x3 = 0

epsilon = 0.01 # Convergence tolerance

converged = False

x\_old = np.array([x1, x2, x3]) # Storing the old values of x1, x2, x3

print('Iteration results')

print(' k, x1, x2, x3 ')

# Iterating up to 50 times

for k in range(1, 50):

# Gauss-Seidel Iteration

x1 = (14 - 3\*x2 + 3\*x3) / 8

x2 = (5 + 2\*x1 - 5\*x3) / (-8)

x3 = (-8 - 3\*x1 - 5\*x2) / (-5)

# Create an array of current values for x1, x2, x3

x = np.array([x1, x2, x3])

# Calculate the change between the old and new solutions (Euclidean distance)

dx = np.sqrt(np.dot(x - x\_old, x - x\_old))

# Print the current iteration and values of x1, x2, x3

print("%d, %.4f, %.4f, %.4f" % (k, x1, x2, x3))

# If the change is smaller than epsilon, stop and declare convergence

if dx < epsilon:

converged = True

print('Converged!')

break

# Update old values of x1, x2, x3 for the next iteration

x\_old = x

# If not converged within 50 iterations, print a message

if not converged:

print('Not converge, increase the # of iterations')

**Output:**

Iteration results

k, x1, x2, x3

1, 1.7500, -1.0625, 1.5875

2, 2.7437, -0.3188, 2.9275

3, 2.9673, 0.4629, 3.8433

4, 3.0177, 1.0226, 4.4332

5, 3.0290, 1.3885, 4.8059

6, 3.0315, 1.6208, 5.0397

7, 3.0321, 1.7668, 5.1861

8, 3.0322, 1.8582, 5.2776

9, 3.0322, 1.9154, 5.3348

10, 3.0323, 1.9512, 5.3705

11, 3.0323, 1.9735, 5.3929

12, 3.0323, 1.9875, 5.4068

13, 3.0323, 1.9962, 5.4156

14, 3.0323, 2.0017, 5.4210

Converged!

**2. The package scipy and pyplot**

**a) Finding if two sets of data have the same mean value.**

**By using the == operator:**

We can check if two sets are equal or not by using the == operator. It will check if two sets are equal or not and return one boolean value. It returns True if the elements of the sets are equal. Else, it returns False. The order of the elements in the set is not considered for the comparison

**Program:**

First\_set = {'one', 'two', 'three'}

second\_set = {'one', 'two', 'three'}

print(First\_set == second\_set)

Output: True

**b) Plotting data read from a file**

**Program:**

import matplotlib.pyplot as plt

# Step 1: Read the data file

filename = input("Enter the filename (e.g., data.txt): ")

x = []

y = []

with open(filename, 'r') as f:

for line in f:

if line.strip() == "":

continue # skip empty lines

parts = line.strip().split()

if len(parts) >= 2:

x.append(float(parts[0]))

y.append(float(parts[1]))

# Step 2: Plot the data

plt.plot(x, y, marker='o', linestyle='-', color='blue', label='Data from file')

plt.title("Data Plot")

plt.xlabel("X-axis")

plt.ylabel("Y-axis")

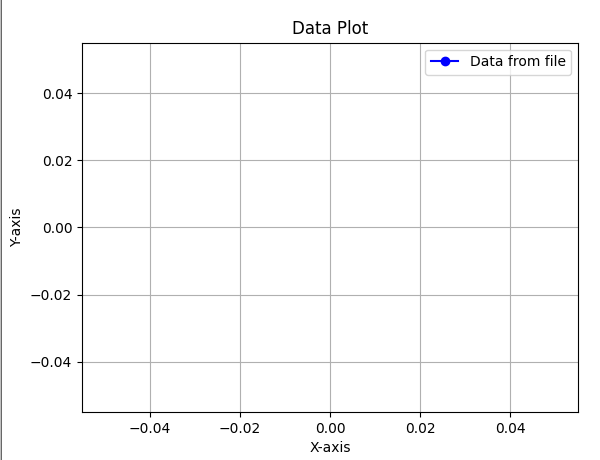
plt.legend()

plt.grid(True)

plt.show()

Output:

**Enter the filename (e.g., data.txt): word.txt**



**c) Fitting a function through a set a data points using polyfit function**

Let’s dive into a basic example of polynomial fitting with NumPy. Suppose we have some experimental data and we believe that it can be approximated by a quadratic polynomial. Here’s how you could do it:

x = np.linspace(0, 10, 10)

y = np.random.normal(loc=1, scale=2, size=10) + 1 \* x \*\* 2

c = np.polyfit(x, y, 2)

p = np.poly1d(c)

In the above code, we:

1. Created an array of x-values using linspace.

2. Generated corresponding y-values using a quadratic function with some added random noise.

3. Used polyfit to fit a second-degree polynomial to our x and y data, storing the coefficients in c.

4. Created a polynomial object p using poly1d that represents the fitted polynomial.

We can also visualize our results with the help of the matplotlib library. Here’s how:

**Program:**

import numpy as np

import matplotlib.pyplot as plt

# Generate x values

x = np.linspace(0, 10, 10)

# Generate noisy quadratic y values

y = np.random.normal(loc=1, scale=2, size=10) + 1 \* x\*\*2

# Fit a polynomial of degree 2

c = np.polyfit(x, y, 2)

p = np.poly1d(c)

# Plotting

plt.scatter(x, y, label='Data') # Original data points

x\_line = np.linspace(min(x), max(x), 100) # Smooth x values for curve

plt.plot(x\_line, p(x\_line), label='Fitted Polynomial', color='red') # Fitted curve

plt.xlabel("x")

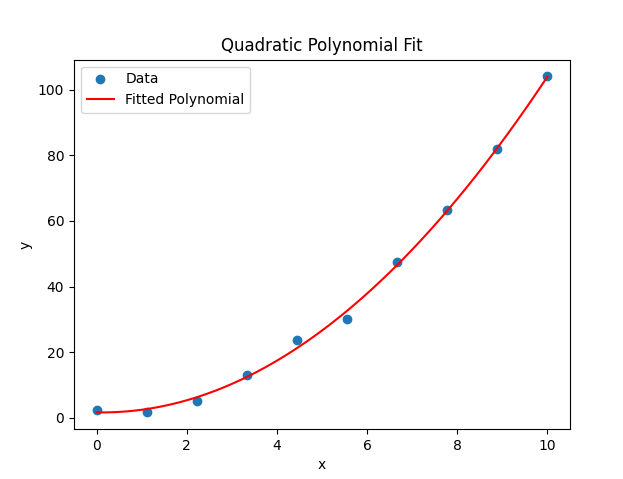
plt.ylabel("y")

plt.title("Quadratic Polynomial Fit")

plt.legend()

plt.show()

Output:



**d) Plotting a histogram of a given data**

**Program:**

import matplotlib.pyplot as plt

import numpy as np

# Generate random data for the histogram

data = np.random.randn(1000)

# Plotting a basic histogram

plt.hist(data, bins=30, color='skyblue', edgecolor='black')

# Adding labels and title

plt.xlabel('Values')

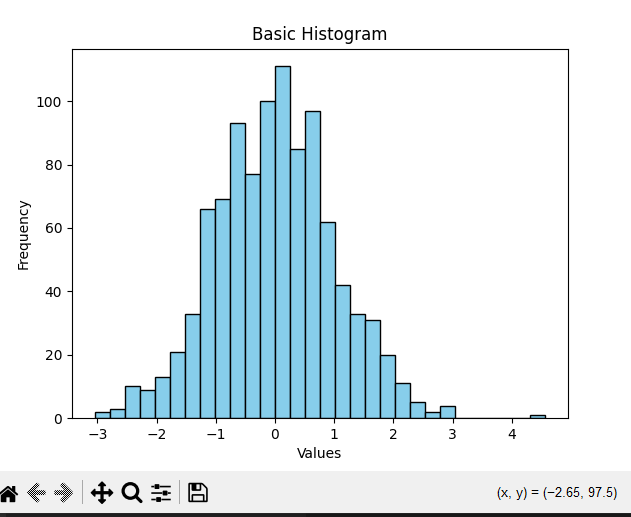
plt.ylabel('Frequency')

plt.title('Basic Histogram')

# Display the plot

plt.show()

Output:



**Week5**

**2. The strings package**

**A) Read text from a file and print the number of lines, words and characters**

**Program:**

# count\_text\_stats.py

filename = 'word.txt' # Make sure this file exists in the same folder

# Initialize counters

num\_lines = 0

num\_words = 0

num\_chars = 0

# Read and process the file

with open(filename, 'r') as file:

for line in file:

num\_lines += 1

num\_words += len(line.split())

num\_chars += len(line)

# Print results

print(f"Lines: {num\_lines}")

print(f"Words: {num\_words}")

print(f"Characters: {num\_chars}")

Output:

Lines: 3

Words: 15

Characters: 84

**B) Read text from a file and return a list of all n letter words beginning with a vowel**

**Program:**

def find\_n\_letter\_words\_starting\_with\_vowel(filename, n):

vowels = ('a', 'e', 'i', 'o', 'u') # Vowels to check

result = []

try:

with open(filename, 'r') as file:

for line in file:

words = line.split() # Split line into words

for word in words:

word = word.strip(",.!?;:()[]{}").lower() # Clean word of punctuation and lowercase it

if len(word) == n and word[0] in vowels:

result.append(word)

except FileNotFoundError:

print("File not found. Please check the filename and try again.")

return result

# Example usage:

filename = input("Enter the filename (e.g., file.txt): ")

n = int(input("Enter the number of letters for the words (n): "))

matching\_words = find\_n\_letter\_words\_starting\_with\_vowel(filename, n)

print(f"Words that are {n} letters long and start with a vowel: ")

print(matching\_words)

Output:

Enter the filename (e.g., file.txt): word.txt

Enter the number of letters for the words (n): 2

Words that are 2 letters long and start with a vowel:

['if', 'if', 'is']

**C) Finding a secret message hidden in a paragraph of text**

**Program:**

def extract\_secret\_message(paragraph, n):

secret\_message = []

# Loop through the text and pick every nth character

for i in range(n - 1, len(paragraph), n): # Start from n-1 (because indexing starts from 0)

secret\_message.append(paragraph[i])

# Join the characters to form the secret message

return ''.join(secret\_message)

# Example usage:

filename = input("Enter the filename (e.g., secret.txt): ")

try:

with open(filename, 'r') as file:

paragraph = file.read() # Read the entire file content

n = int(input("Enter the nth number to find the hidden message: "))

secret\_message = extract\_secret\_message(paragraph, n)

print("Secret message found:")

print(secret\_message)

except FileNotFoundError:

print("File not found. Please check the filename and try again.")

**Output:**

Enter the filename (e.g., secret.txt): word.txt

Enter the nth number to find the hidden message: 2

Secret message found:

edtefl.

pi h otn nowrs

Ceki ahwr trswt oe a ,i ,u n fislnt sn

Cletadrtr h acigwrs

**D) Plot a histogram of words according to their length from text read from a file.**

**Program:**

import matplotlib.pyplot as plt

def plot\_word\_length\_histogram(filename):

word\_lengths = []

try:

# Open and read the file

with open(filename, 'r') as file:

content = file.read()

# Split content into words

words = content.split()

# Calculate length of each word and store it

for word in words:

cleaned\_word = word.strip(",.!?;:()[]{}").lower() # Clean punctuation and convert to lowercase

word\_lengths.append(len(cleaned\_word))

# Plotting the histogram of word lengths

plt.hist(word\_lengths, bins=range(1, max(word\_lengths) + 1), edgecolor='black')

plt.title("Histogram of Word Lengths")

plt.xlabel("Word Length")

plt.ylabel("Frequency")

plt.grid(True)

plt.show()

except FileNotFoundError:

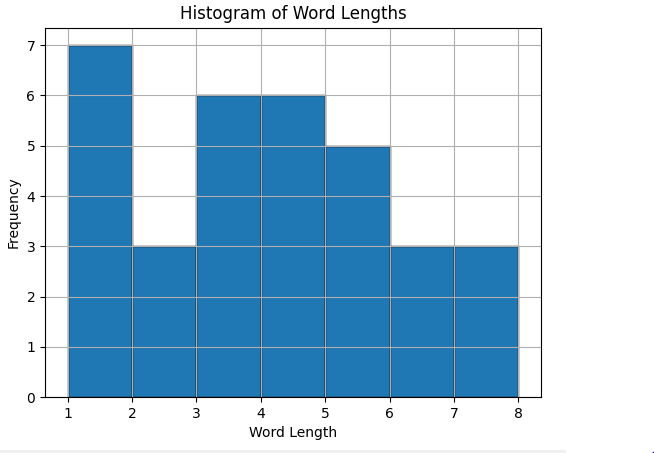
print("File not found. Please check the filename and try again.")

# Example usage:

filename = input("Enter the filename (e.g., textfile.txt): ")

plot\_word\_length\_histogram(filename)

**Output:**



**Week6**

**Installing OS on Raspberry Pi**

**a) Installation using Pi Imager**

**b) Installation using image file**

• Downloading an Image

• Writing the image to an Scard

• using Linux

• using Windows

• Booting up Follow the instructions given in the URL [https://www.raspberrypi.com/documentation/computers/getting started.html](https://www.raspberrypi.com/documentation/computers/getting%20started.html)

If you’re new to the Raspberry Pi world, installing (or reinstalling) the operating system can be a daunting challenge. Hopefully, the Raspberry Pi Foundation does its best to make it as easy as possible for you. This article will show you the easiest and safest method to do it on your own.

**Raspberry Pi OS can be installed on a new SD card from any computer, by using an application named “Raspberry Pi Imager”, created by the Raspberry Pi manufacturer. It includes all versions of Raspberry Pi OS, to flash it to the SD card in a few clicks.**

**Install Raspberry Pi Imager**

The only tool you need on your computer to install Raspberry Pi OS on your SD card is Raspberry Pi Imager. It has been developed by the Raspberry Pi Foundation to make this process easier. No manual downloads or random apps are required anymore.

Here is how to get it on your computer:

• Go to the official Raspberry Pi website.

• Visit the “Software” page.

• Download Raspberry Pi Imager for your system:

• It’s available for Windows, macOS and Ubuntu. It’s even possible to install it on an existing Raspberry Pi OS system if you already have one.

• It’s even included in some RPI OS editions, so you may already have it.

• Once downloaded, follow the installation steps you’d use for any application on your computer. On Windows, it’s a double click on the file, and the next, next, next.

**Week7**

Accessing GPIO pins using Python

**a)Installing GPIO Zero library.**

First, update your repositories list:

sudo apt update

Then install the package for Python 3:

GPIO Zero is installed by default in the Raspberry Pi OS desktop image, Raspberry Pi OS Lite image, and the Raspberry Pi Desktop image for PC/Mac, all available from raspberrypi.org. Follow these guides to installing on other operating systems, including for PCs using the remote GPIO feature.

1.1. Raspberry Pi3

GPIO Zero is packaged in the apt repositories of Raspberry Pi OS, Debian and Ubuntu. It is also available on PyPI

1.1. aptℑ

First, update your repositories list:

pi@raspberrypi:~$ sudo apt update

Then install the package for Python 3:

pi@raspberrypi:~$ sudo apt install python3-gpiozero

or Python 2:

pi@raspberrypi:~$ sudo apt install python-gpiozero

1.1.2. pipℑ

If you’re using another operating system on your Raspberry Pi, you may need to use pip to install GPIO Zero instead. Install pip using get-pip and then type:

pi@raspberrypi:~$ sudo pip3 install gpiozero

or for Python 2:

pi@raspberrypi:~$ sudo pip install gpiozero

To install GPIO Zero in a virtual environment, see the Development page.

1.2. PC/Macℑ

In order to use GPIO Zero’s remote GPIO feature from a PC or Mac, you’ll need to install GPIO Zero on that computer using pip. See the Configuring Remote GPIO page for more information.

**b)Blinking an LED connected to one of the GPIO pin**

With the circuit created we need to write the Python script to blink the LED. Before we start writing the software we first need to install the Raspberry Pi GPIO Python module. This is a library that allows us to access the GPIO port directly from Python.

To install the Python library open a terminal and execute the following

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

With the library installed now open your favorite Python IDE (I recommend Thonny Python IDE more information about using it here).

Our script needs to do the following:

• Initialize the GPIO ports

• Turn the LED on and off in 1 second intervals

To initialize the GPIO ports on the Raspberry Pi we need to first import the Python library, the initialize the library and setup pin 8 as an output pin.

import RPi.GPIO as GPIO # Import Raspberry Pi GPIO library

from time import sleep # Import the sleep function from the time module

GPIO.setwarnings(False) # Ignore warning for now

GPIO.setmode(GPIO.BOARD) # Use physical pin numbering

GPIO.setup(8, GPIO.OUT, initial=GPIO.LOW) # Set pin 8 to be an output pin and set initial

value to low (off)

Next we need to turn the LED on and off in 1 second intervals by setting the output pin to either

high (on) or low (off). We do this inside a infinite loop so our program keep executing until we manually stop it.

while True: # Run forever

GPIO.output(8, GPIO.HIGH) # Turn on

sleep(1) # Sleep for 1 second

GPIO.output(8, GPIO.LOW) # Turn off

sleep(1) # Sleep for 1 second

Combining the initialization and the blink code should give you the following full Python program:

import RPi.GPIO as GPIO # Import Raspberry Pi GPIO library

from time import sleep # Import the sleep function from the time module

GPIO.setwarnings(False) # Ignore warning for now

GPIO.setmode(GPIO.BOARD) # Use physical pin numbering

GPIO.setup(8, GPIO.OUT, initial=GPIO.LOW) # Set pin 8 to be an output pin and set initial

value to low (off)

while True: # Run forever

GPIO.output(8, GPIO.HIGH) # Turn on

sleep(1) # Sleep for 1 second

GPIO.output(8, GPIO.LOW) # Turn off

sleep(1) # Sleep for 1 second

With our program finished, save it as blinking\_led.py and run it either inside your IDE or in the console with:

$ python blinking\_led.py

**c)Adjusting the brightness of an LED**

**d)Adjust the brightness of an LED (0 to 100, where 100 means maximum brightness) using the in-built PWM wavelength.**

Do you want to adjust the brightness of your Light-Emitting Diode (LED)? If we break it down to the most basic, there are two ways to change the brightness of an LED:

• Changing the resistance value.

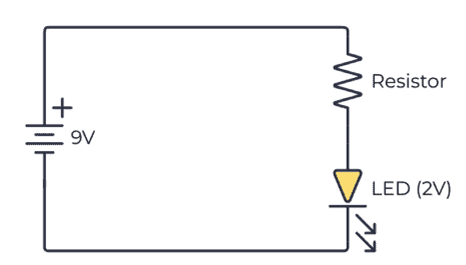
• Turning it on and off fast (Using PWM).

Below I’ll explain the two options and show you circuits you can build.

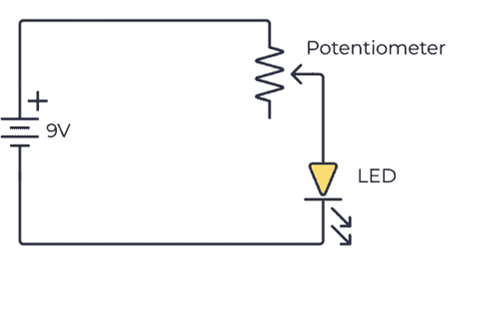


**Option 1: Change the resistor value**

The brightness of an LED depends on the current through it. A resistor in series with the LED sets the current. In the circuit below, you can see the basic setup for controlling an LED:



A potentiometer is a variable resistor that you can use to change the brightness of the LED. Just add the potentiometer in series with the LED. When you adjust the knob of the potentiometer, the brightness of the LED will change.



**Week8**

**Collecting Sensor Data**

**DHT Sensor interface**

◦ Connect the terminals of DHT GPIO pins of Raspberry Pi.

◦ Import the DHT library using import Adafruit\_DHT

◦ Read sensor data and display it on screen.

We're going to use a special library called adafruit\_blinka (named after Blinka, the CircuitPython mascot) to provide the layer that translates the CircuitPython hardware API to whatever library the Linux board provides.

Installing CircuitPython Libraries on Raspberry Pi or BeagleBone Black If you haven't set up your Raspberry Pi or BeagleBone Black for running CircuitPython libraries yet, follow our guide and come back to this page when you've completed the steps listed on the page and verified that your setup is working:

• Setup your Linux Board for using CircuitPython Libraries

**Installing the CircuitPython-DHT Library**

You'll also need to install a library to communicate with the DHT sensor. Since we're using Adafruit Blinka (CircuitPython), we can install CircuitPython libraries straight to our small linux board. In this case, we're going to install the CircuitPython\_DHT library. This library works with both the DHT22 and DHT11 sensors.

Run the following command to install the CircuitPython-DHT library:

pip3 install adafruit-circuitpython-dht

sudo apt-get install libgpiod2

**Testing the CircuitPython DHT Library**

To make sure you've installed everything correctly, we're going to test that we can read values from the DHT sensor connected to your device.

Create a new file called dht\_simpletest.py with nano or your favorite text editor and put the following in:

Download Project Bundle

Copy Code

# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries

# SPDX-License-Identifier: MIT

import time

import board

import adafruit\_dht

# Initial the dht device, with data pin connected to:

dhtDevice = adafruit\_dht.DHT22(board.D18)

# you can pass DHT22 use\_pulseio=False if you wouldn't like to use pulseio.

# This may be necessary on a Linux single board computer like the Raspberry Pi,

# but it will not work in CircuitPython.

# dhtDevice = adafruit\_dht.DHT22(board.D18, use\_pulseio=False)

while True:

try:

# Print the values to the serial port

temperature\_c = dhtDevice.temperature

temperature\_f = temperature\_c \* (9 / 5) + 32

humidity = dhtDevice.humidity

print(

"Temp: {:.1f} F / {:.1f} C Humidity: {}% ".format(

temperature\_f, temperature\_c, humidity

)

)

except RuntimeError as error:

# Errors happen fairly often, DHT's are hard to read, just keep going

print(error.args[0])

time.sleep(2.0)

continue

except Exception as error:

dhtDevice.exit()

raise error

time.sleep(2.0)