SOMAIYA VIDYAVIHAR UNIVERSITY
S K Somaiya College

Somaiya TRUST

MSc IT Sem-III

AIOT Lab Journal

Academic Year 2025-2026

Seat Number: 31031424017

Name: Priya Prakash Killedar

Course Faculty: Salma M Shaikh

INDEX

SrNo	Title	Date
1	Loading Raspbian & Windows IOT core on Raspberry Pi & running Python/Node.js program & understand its working	03/09/2025
2	Create a home automation system and control devices remotely	04/09/2025
3	Build Your Own IoT Platform Using Node-RED	10/09/2025
4	Implement Microservices on IoT Device	11/09/2025
5	Create a Blockchain on Raspberry Pi and Authenticate IoT Devices	17/09/2025
6	Create programs using Microsoft Cognitive APIs for IoT.	18/09/2025
7	Use IoT Device with AWS IoT Core	24/09/2025
8	Send Telemetry from Device to Azure IoT Hub and Read with Service Application	01/10/2025
9	Face detection using IOT devices	02/10/2025

PRACTICAL NO : 1

Loading Raspbian & Windows IOT core on Raspberry Pi & running Python/Node.js program & understand its working

Aim : To perform loading Raspbian & Windows IOT core on Raspberry Pi & running Python/Node.js program & understand its working

Apparatus required:

1. Raspberry Pi
2. Monitor or TV
3. HDMI cable
4. Ethernet cable
5. USB keyboard
6. USB mouse
7. Micro USB power supply
8. 8GB or larger microSD card
9. SD Card Reader
10. The Pi4J Project – Pin Numbering - Raspberry Pi 3 Model B

Theory:

1. Raspberry Pi 3 Model B (v1.2)

The Raspberry Pi 3 Model B (v1.2) is the third generation single-board computer from the Raspberry Pi Foundation. It was launched in February 2016 as an upgrade to the Raspberry Pi 2 Model B, offering a significant boost in processing power and built-in wireless connectivity, making it ideal for IOT, media centers, home automation, and educational use.

Key Specifications

Processor (CPU)
Broadcom BCM2837 SoC
1.2 GHz 64-bit Quad-Core ARM Cortex-A53 CPU
Built on ARMv8-A architecture (64-bit capable)
Delivers-50-60% performance improvement over Raspberry Pi 2 (ARM Cortex-A7, 32-bit)
Supports NEON SIMD extensions for faster multimedia and signal processing

Graphics (GPU)

Broadcom VideoCore IV
Capable of hardware-accelerated OpenGL ES 2.0
Supports 1080p30 H.264 video decode and encode

Dual-support output through HDMI and composite video
Capable of light gaming and multimedia playback

Memory
1 GB LPDDR2 SDRAM
Shared between CPU and GPU
Faster access compared to previous generations

Connectivity
Wireless
802.11n Wi-Fi (2.4 GHz)-onboard wireless LAN, no external dongle required Bluetooth 4.1+Bluetooth Low Energy (BLE)- for low-power IoT devices and wireless peripherals

Wired
10/100 Mbps Ethernet port for stable, wired networking

Input/Output (I/O)
40-pin GPIO Header
Fully backward-compatible with Raspberry Pi 1 Model B+ and Pi 2
Supports UART, SPI, 12C, PWM, and GPIO functionality

4x USB 2.0 Ports
Can connect keyboards, mice, USB storage, cameras, Wi-Fi dongles (if additional needed)

Full-size HDMI Port
Outputs up to 1080p resolution
Supports CEC (Consumer Electronics Control) for controlling with TV remotes

3.5mm Audio/Composite Video Jack
Provides analog stereo audio output
Composite video output for older displays

Camera Interface (CSI)
Compatible with Raspberry Pi Camera Module for high-resolution imaging projects

Display Interface (DSI)
For connecting official Raspberry Pi Touch Display or other DSI-compatible displays

MicroSD Card Slot
Used for operating system and data storage

Supports high-speed Class 10 UHS cards for better performance

Power
Micro-USB Power Input (5V/2.5A recommended)
Stable power supply is essential for reliable operation
Supports Power-over-USB (but not native PoE)

Power Consumption
Idle: -400-500 mA
Under load: up to 1.3A (with peripherals attached)

Software Support
Operating Systems:
Raspberry Pi OS (official), Ubuntu Server, LibreELEC, OSMC, Windows 10 IoT Core, and other Linux-based distributions

Programming Languages:
Python, C/C++, Java, Scratch, Node.js, Go, etc.

Community Support:
Extensive forums, tutorials, and open-source libraries available

2. Monitor or TV (Display Device)
Purpose: To view the Raspberry Pi's graphical interface and output.

Recommended Connection: HDMI
Most modern TVs and monitors come with an HDMI port, making setup quick and simple. Just connect the Raspberry Pi to the display using an HDMI-to-HDMI cable.
If Using Older Displays:
DVI Monitor: Use an HDMI-to-DVI cable or adapter (video only, no sound).
VGA Monitor: Requires an HDMI-to-VGA converter (not just a simple adapter), because the Raspberry Pi outputs digital video, and VGA is analog video.

3. HDMI-to-HDMI Cable
Purpose: To transmit both video and audio signals from the Raspberry Pi to the monitor/TV
4. Ethernet Cable (Optional but Recommended)

Purpose: To connect the Raspberry Pi to the internet using a wired network.
Benefits:
Faster and more stable than Wi-Fi.

Useful for headless setup (when you run the Pi without a monitor, accessing it remotely via SSII or VNC).

USB Keyboard and Mouse
Purpose: To interact with the Raspberry Pi, enter commands, and navigate the desktop interface.
Compatibility:
Any standard USB keyboard and mouse will work (plug-and-play).
No drivers needed-just plug them in and start using them.

Power Supply
Recommended: 5V, 2A (minimum) micro-USB power supply for Raspberry Pi 3.
Why It Matters:
Stable power is crucial. Low-quality adapters can cause random reboots or "under-voltage" warmings.
Prefer official Raspberry Pi power supplies for best results.

Micro SD Card
Purpose: Acts as the main storage for your Raspberry Pi (like a hard drive).
Requirements:
Minimum 8 GB capacity (Class 10 recommended).
Holds the Operating System (such as Raspberry Pi OS) and all your files and projects.

Procedure:
Installation
Now since you have all the required hardware, we will now learn how to get the operating system onto your microSD card so that you can start using software on your Raspberry Pi 1. Get

Raspbian OS on your microSD card

I. Raspbian comes preinstalled with plenty of software for education, programming and general use. It has Python, Scratch, Sonic Pi, Java, Mathematica and more.
II. To download Raspbian log onto raspberrypi.org and click on download, then click on the download, then click on Raspbian and lastly download the RASPBIAN JESSIE WITH DESKTOP file. You can choose either
the Torrent file or ZIP file.
III. The downloaded file will be in zip format. To unzip the file, you will require an unzip tool. You can use any unzipping tool viz. WINRAR, 7ZIP etc. After unzipping the file, you will find a disc image file in the unzipped folder.
IV. Now format the micro SD Card before writing the disc image file on the micro SD card. You can use the SD Formatter tool or any other tool. To write the image file of the

operating system on the SD card you will require a Disk Imager tool. For this you can use the Win32 Disk Imager tool.

V. Once the image is written on the SD Card, your untitled SD card will now have the name boot. Your SD Card will now hold the Raspbian Operating system required for the first-time setup

Plugging in your Raspberry Pi

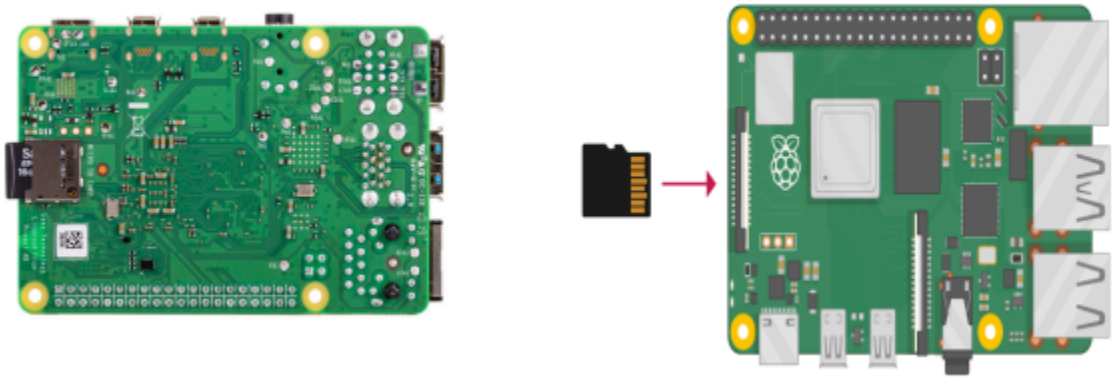
1. Begin by placing your SD card into the SD card slot on the Raspberry Pi. It will only fit one way.
2. Next, plug your keyboard and mouse into the USB ports on the Raspberry Pi.
3. Make sure that your monitor or TV is turned on, and that you have selected the right input (e.g. HDMI 1, DVI, etc). Connect your HDMI cable from your Raspberry Pi to your monitor or TV
4. If you intend to connect your Raspberry Pi to the internet, plug an Ethernet cable into the Ethernet port, or connect a WiFi dongle to one of the USB ports (unless you have a Raspberry Pi 3)
5. When you have plugged all the cables and SD card in correctly, connect the micro USB power supply. This action will turn on and boot your Raspberry Pi.

Code:

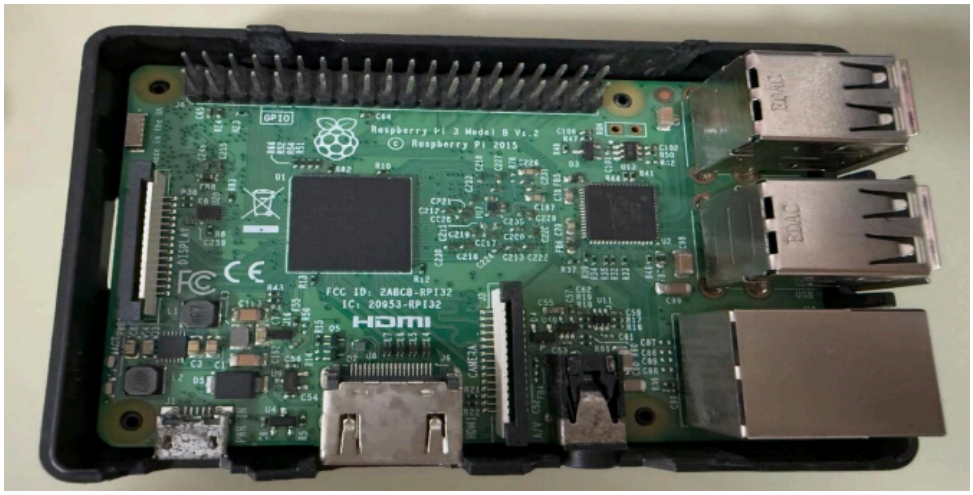
Create and run a simple Python script:

```
print("Hello, IoT World!")
```

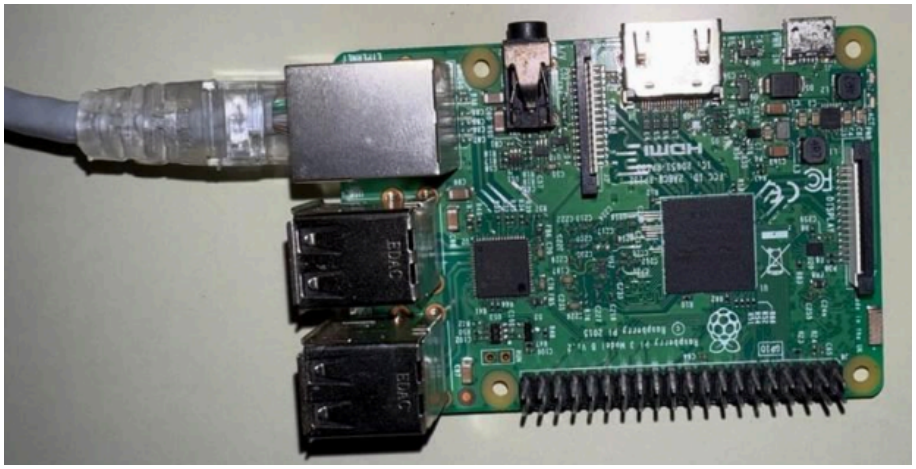
A. Insert bootable memory card into Raspberry pi.



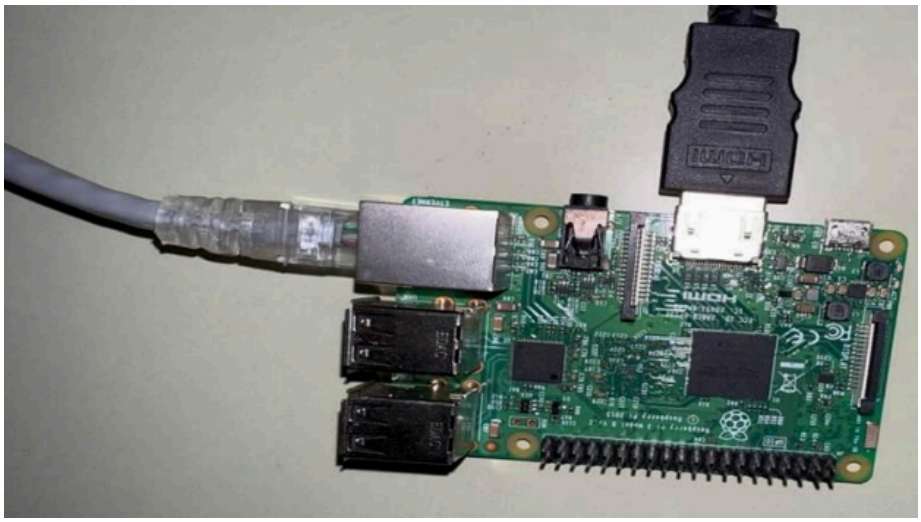
B. Connect all devices



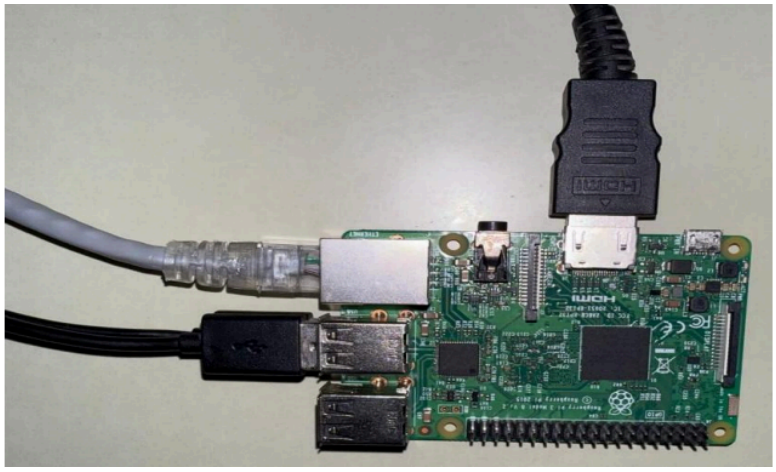
1. Ethernet Cable



2. HDMI Cable or HDMI to VGA port



3. Use keyboard and mouse.

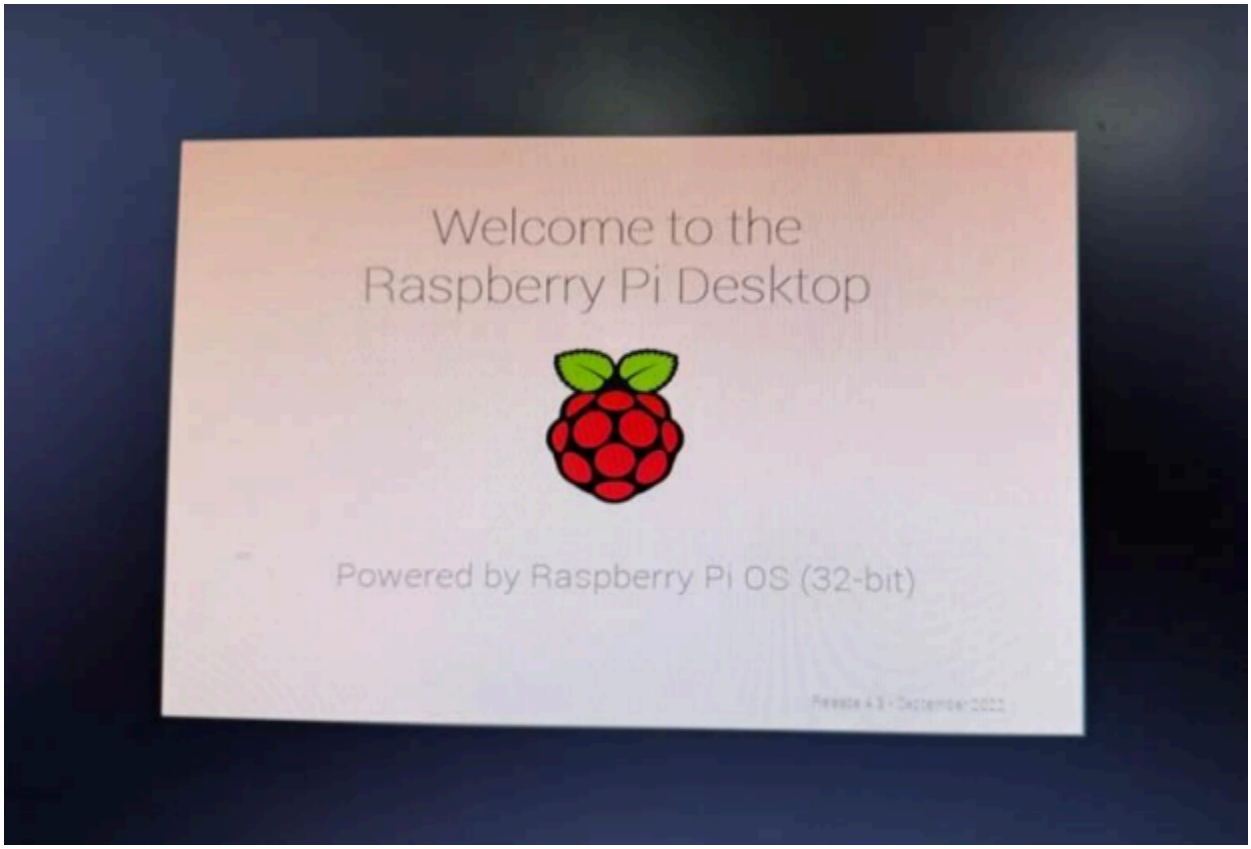


Mouse & Keyboard

4. Power Supply (Charger)

C. Power on Switch button.

Output :



Welcome screen

PRACTICAL NO : 2

Create a Home Automation System and Control Devices Remotely

Aim: To perform, create a home automation system and control devices remotely and understand its working.

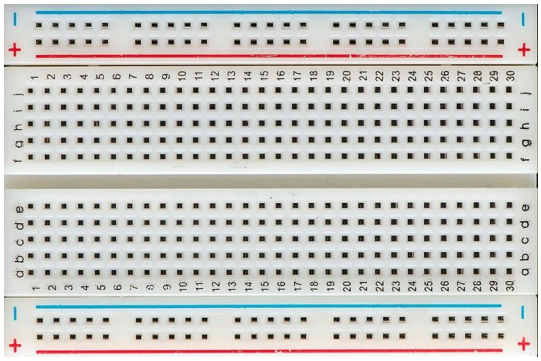
Apparatus Required:

1. Raspberry Pi
2. LED
3. A Breadboard
4. 330 ohm resistor
5. 2x Male to Female jumper wires

Theory: The Breadboard

The breadboard is a convenient way to connect electronic components to each other without having to solder them together.They are often used to test (prototype) a circuit design before making a final version of a project, whether that be soldering wires together or creating a Printed Circuit Board (PCB).

The holes on the breadboard are connected as follows:



- All holes in this row are connected.
- All holes in this row are connected.
- All holes in each column are connected.
- The centre breaks the column connections.
- All holes in each column are connected.

With the breadboard, the top row of holes are all connected together, markd with a red line. And so are the second row of holes, marked with a blue line. The same goes for the matching two blue/red rows of holes at the bottom of the breadboard.

In the middle, the columns of holes are connected together with a break in the middle (which comes in handy if you straddle a chip or similar component across the two sides, allowing you to connect to both sides of a chip with wires, without the chip's legs connecting together).

The LED

A Red LED When you pick up the LED, you will notice that one leg is longer than the other.

The longer leg (known as the anode), is always connected to the positive supply of the circuit. The shorter leg (known as the cathode) is connected to the negative side of the power supply, known as ground.

LED stands for Light Emitting Diode, and glows when electricity (current) is passed through it.

LEDs will only work if power is supplied the correct way round (i.e. if the polarity is correct).

The Resistor

A 330 Ohm Resistor, You must ALWAYS use resistors to connect LEDs up to the GPIO pins of the Raspberry Pi. The Raspberry Pi GPIO pins can only supply a small current (about 60\\text{mA}). The LEDs will want to draw more, and if allowed to they may damage your Raspberry Pi or the pins used. Therefore adding resistors to the circuit will ensure that only this small amount of current will flow.

Resistors are a way of limiting the amount of electricity going through a circuit; specifically, they limit the amount of current that is allowed to flow. The measure of resistance is called the Ohm (Ω), and the larger the resistance, the more it limits the current. The value of a resistor is marked with coloured bands along the length of the resistor body.

You will be using a 330 Ω.Omega resistor. You can identify the 330 Ω.Omega resistors by the colour bands along the body. The colour coding will depend on how many bands are on the resistors supplied:

If there are four colour bands, they will be Orange, Orange, Brown, and then Gold.

If there are five bands, then the colours will be Orange, Orange, Black, Black, Brown.

It does not matter which way round you connect the resistors. Current flows in both ways through them.

Jumper Wires

Jumper wires are used on breadboards to 'jump' from one connection to another. The ones you will be using in this circuit have different connectors on each end. The end with the 'pin' will go into the Breadboard. The end with the piece of plastic with a hole in it will go onto the Raspberry Pi's GPIO pins.

The Raspberry Pi GPIO Pins

GPIO stands for General Purpose Input Output. It's a way the Raspberry Pi can control and monitor the outside world by being connected to electronic circuits.

The Raspberry Pi is able to control LEDs, turning them on or off, or motors, or many other things using these pins (this is known as an 'output'). It is also able to detect whether things like a switch has been pressed, or temperature, light and more (this is know an an 'input').

See the GPIO pin diagram for the raspberry pi model you have for connection:

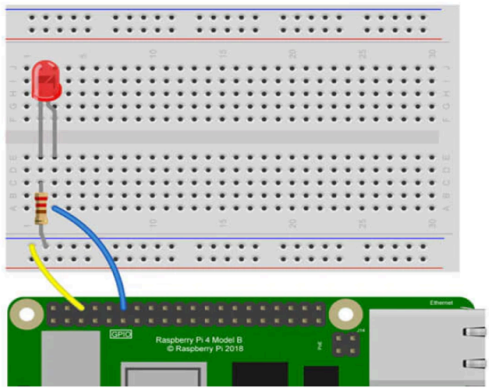
https://www.pi4j.com/1.2/pins/model-3b-rev1.html

Procedure

The circuit consists of a power supply (the Raspberry Pi), an LED that lights when the power is applied, and a resistor to limit the current that can flow through the circuit:

- You will be using one of the 'ground' (GND) pins to act like the 'negative' or 0 volt ends of a battery.
- The 'positive' end of the battery will be provided by a GPIO pin. Here we will be using pin GPIO18 (which is physical pin 12).
- When these pins are 'taken high', which means it outputs 3.3 volts, the LED will light.

Now take a look at the circuit diagram below:

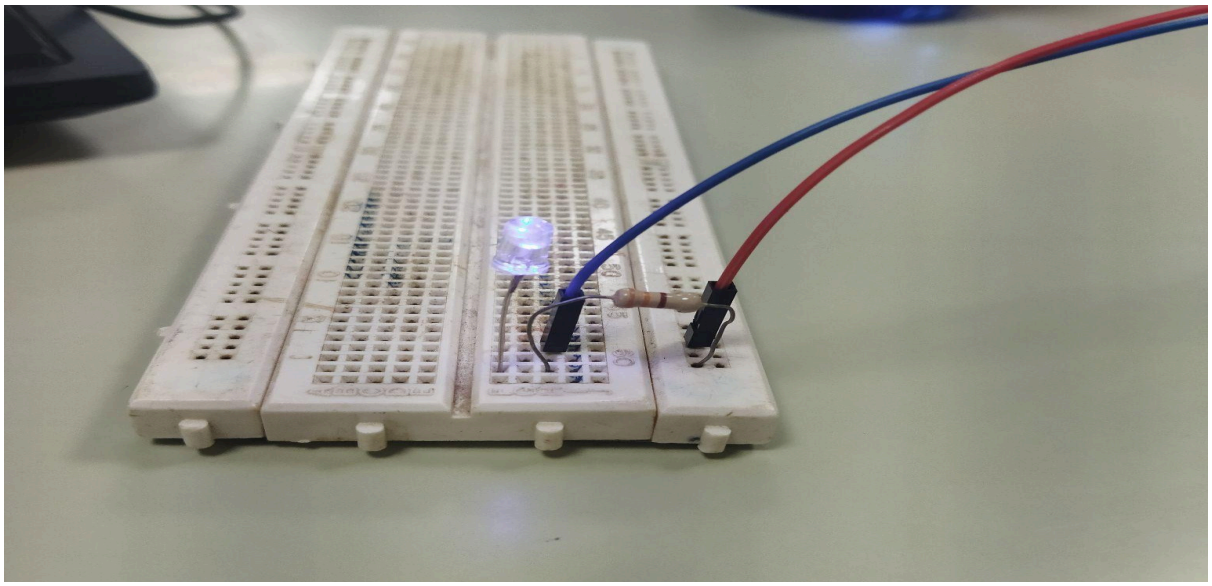


You should turn your Raspberry Pi off for the next bit, just in case you accidentally short something out.

- Use one of the jumper wires to connect a ground pin to the rail, marked with blue, on the breadboard. The female end goes on the Raspberry Pi's pin, and the male end goes into a hole on the breadboard.
- Then connect the resistor from the same row on the breadboard to a column on the breadboard, as shown above.
- Next, push the LED's legs into the breadboard, with the long leg (with the kink) on the right.
- Lastly, complete the circuit by connecting the right hand leg of the LED to GPIO18. This is shown here with the blue wire.

led.py

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(4,GPIO.OUT)
print("LED on")
GPIO.output(4,GPIO.HIGH)
time.sleep(2)
print("LED off")
GPIO.output(4,GPIO.LOW)
```



PRACTICAL NO : 3

Build your own IOT platform.

Aim: To perform build your own iot platform using node-red and understand its working.

Apparatus Required:

Raspberry Pi board, SD card (16/32GB), Power supply, HDMI cable, Internet connection, Laptop/PC, Sensor modules (DHT11/Relay/LED as needed),Node-RED.

Theory:

Node-Red

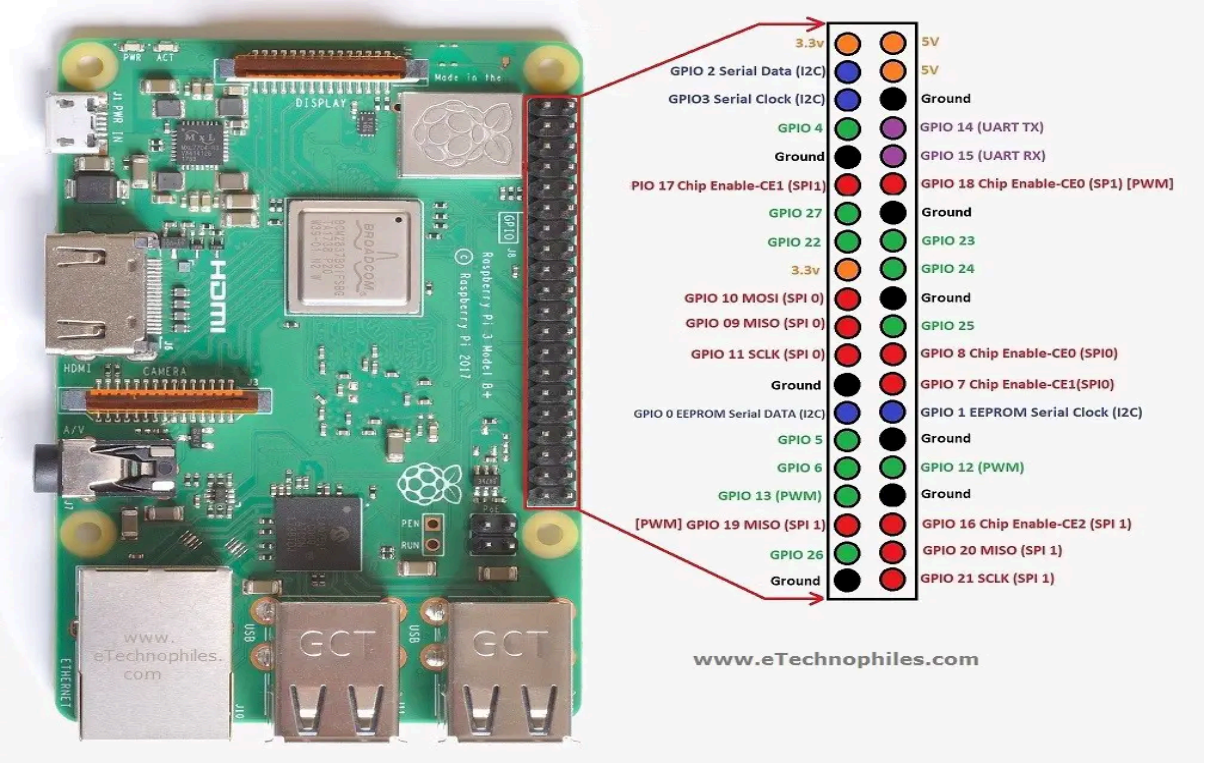
Node-RED is an open-source, low-code development tool that uses a visual, flow-based programming environment to connect hardware devices, APIs, and online services, making it easy to create applications for the Internet of Things (IoT). Users can drag and drop "nodes" into a browser-based editor to build "flows" that automate tasks by sending and processing data between different components. Built on Node.js, Node-RED simplifies the development of industrial control systems, home automation, and other data-driven applications, supporting protocols like MQTT and HTTP.

Procedure:

1. **Circuit Setup**
   The circuit consists of a power supply (the Raspberry Pi), an LED that lights up when power is applied, and a resistor to limit the current.

   You'll use 3 of the ground (GND) pins on the Raspberry Pi to act as the 'negative' or 0 volt end of a battery. Precisely the 3rd ground pin on right
   The 3 'positive' ends will be provided by a GPIO pin. In this case, we'll use pin GPIO4, GPIO5, GPIO6.
   When GPIO4 is "taken high," it outputs 3.3V, and the LED1 will light up, same logic for GPIO5 and GPIO6

Steps to build the circuit:



Turn your Raspberry Pi off before connecting anything.

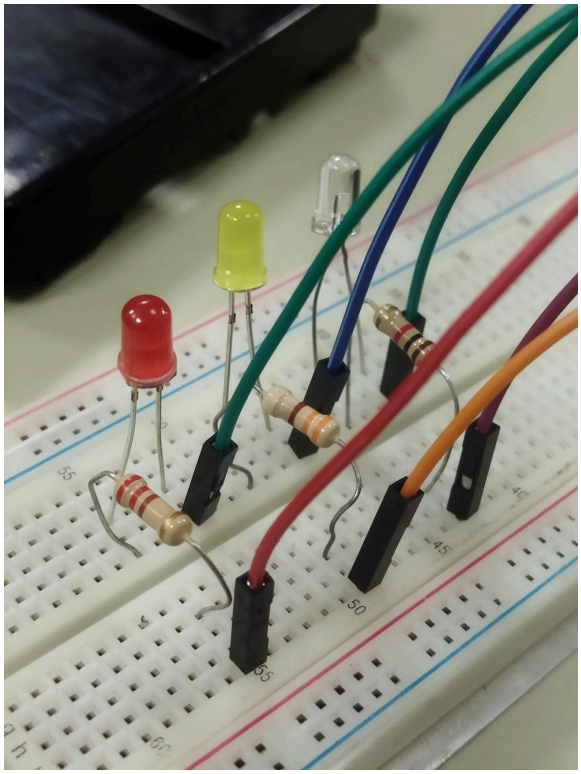
Use one of the jumper wires to connect a ground (GND) pin on the Raspberry Pi to the ground rail (marked with a blue line) on the breadboard. The female end goes on the Pi's pin, and the male end goes into a hole on the breadboard.

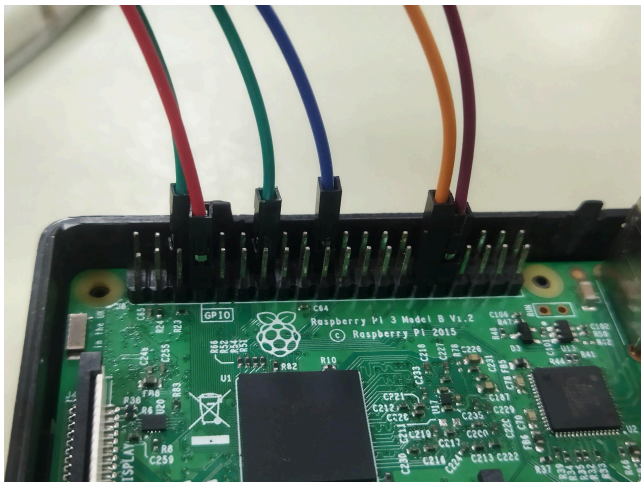
Connect the resistor from a ground rail column on the breadboard to a new column on the breadboard.

Push the LED1's legs into the breadboard. The long leg (anode, with the kink) should be in a column connected to the resistor. The short leg (cathode) should be in a different column.

Complete the circuit by connecting the long leg of the LED1 to GPIO4 using the second jumper wire.

Perform the same steps for LED2 and LED3 and connect them to GPIO5 and GPIO6 respectively, as shown in below images:

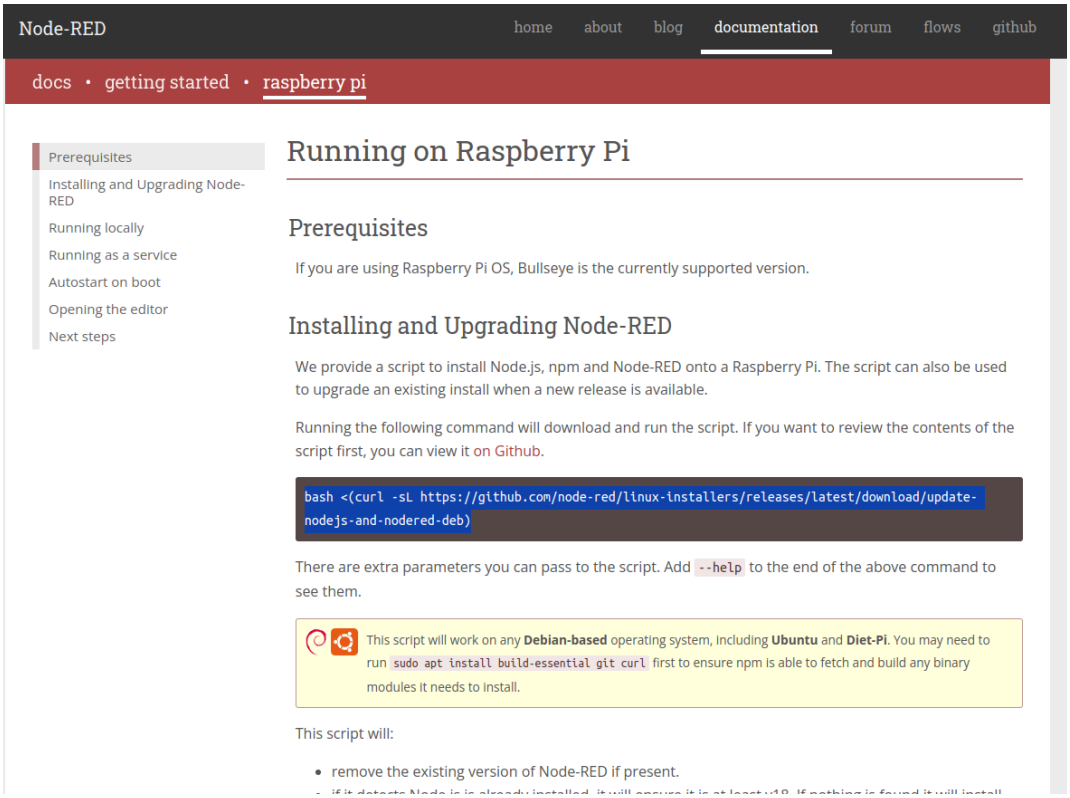




2. **Installing Node-RED**
   Open the Raspberry Pi's terminal.
   Copy and paste the following command to install Node-RED and its dependencies.

   ```
   bash <(curl -sL https://github.com/node-red/linux-installers/releases/latest/download/update-nodejs-and-nodered-deb)
   ```

   Press **Enter** to run the script. This may take several minutes.



3. **Starting Node-RED**
   Once the installation is complete, start Node-RED by typing the following command in the terminal:

   ```
   node-red-start
   ```

   Node-RED will start, and the terminal will display the IP address and port number (e.g., 192.168.1.5:1880).

   Open a web browser on your PC or laptop connected to the same network and navigate to the IP address shown to access the Node-RED web interface.

4. **Installing the Dashboard**

   Inside the Node-RED interface, click on the **three horizontal lines (menu)** in the top-right corner.

   Select **"Manage palette"** from the dropdown menu.

   Go to the **"Install"** tab.

   In the search bar, type node-red-dashboard.

Click the **"install"** button next to the Node-RED Dashboard package.

5. **Building the Flow**

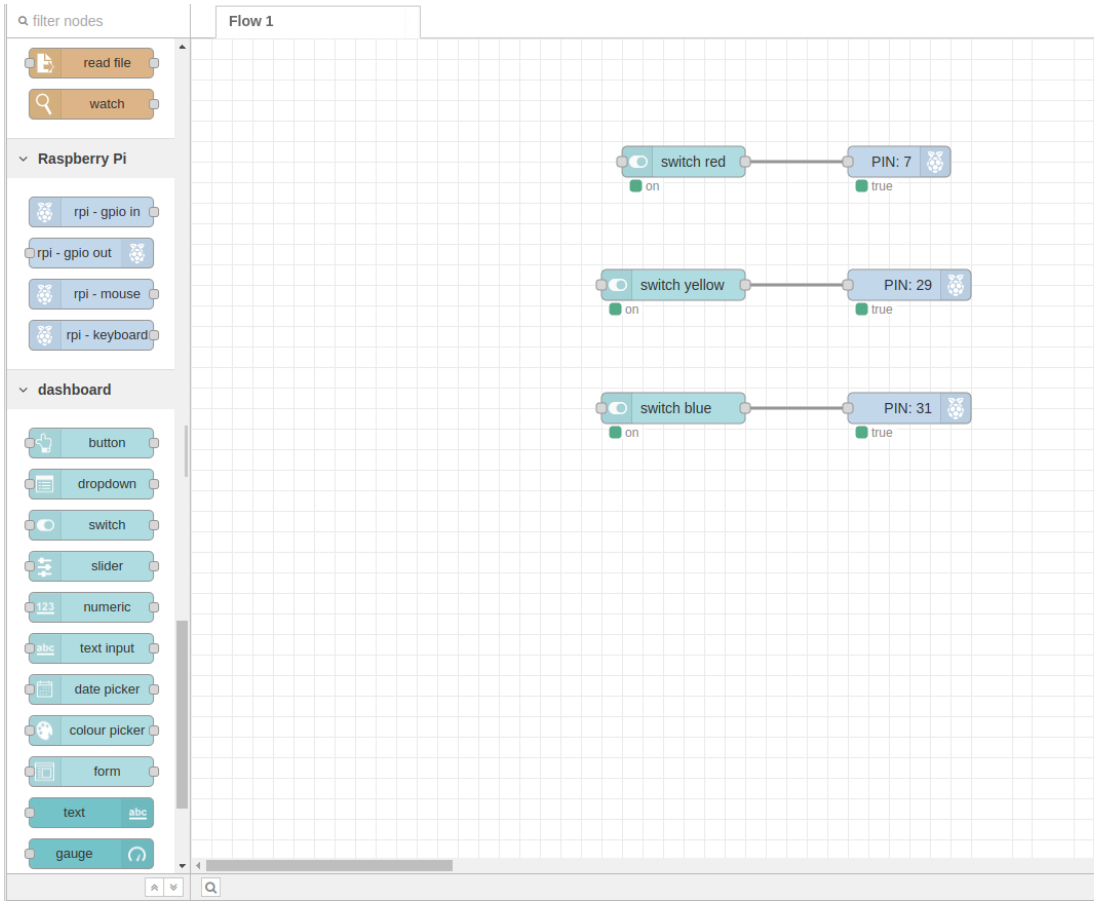
Drag a ui_switch node from the dashboard section on the left-hand side into the workspace.

Double-click the ui_switch node to configure it. Create a new group and give it a label, for example, "Led Controller." and then add a new Dashboard.

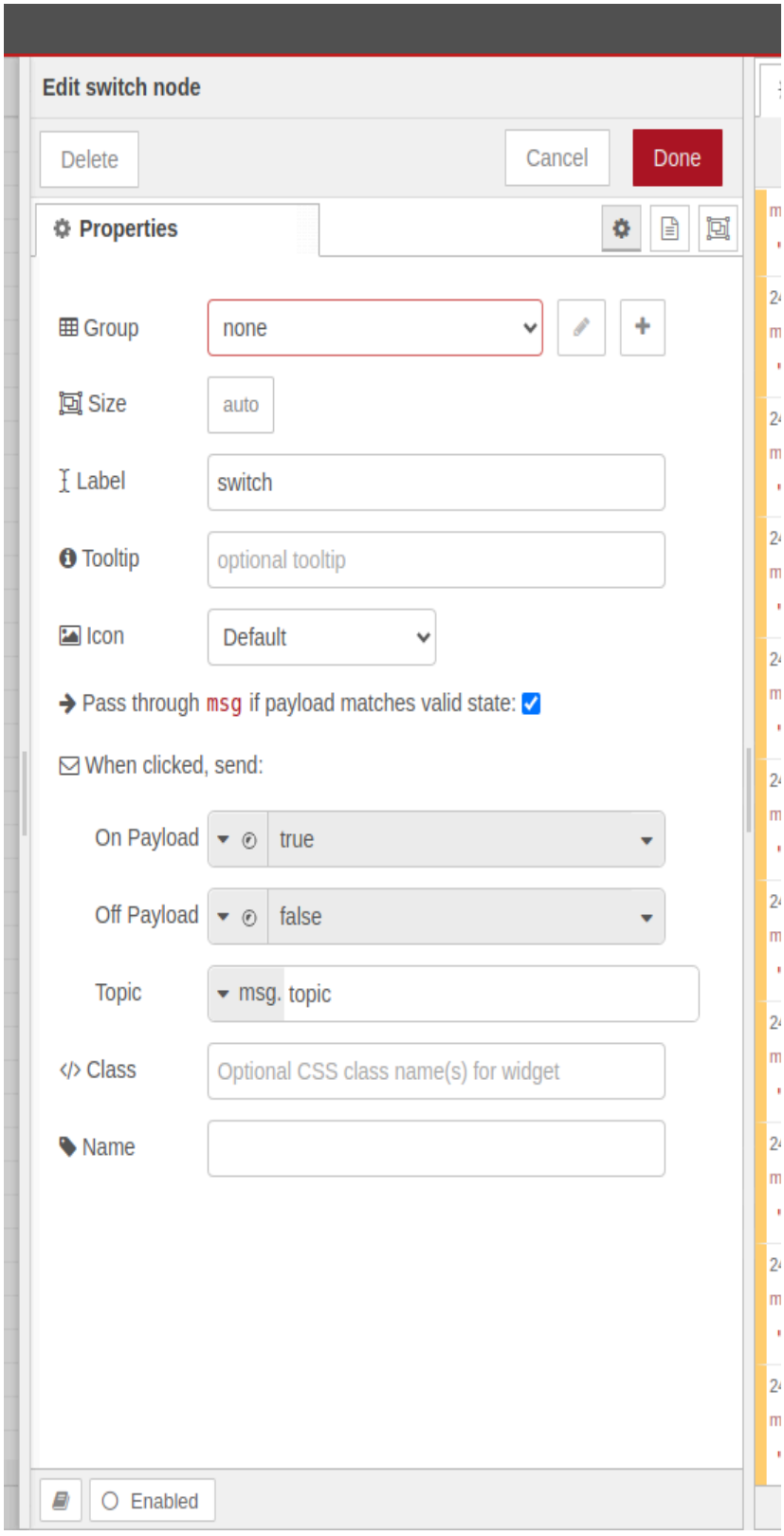
Drag a rpi gpio out node from the Raspberry Pi section into the workspace.

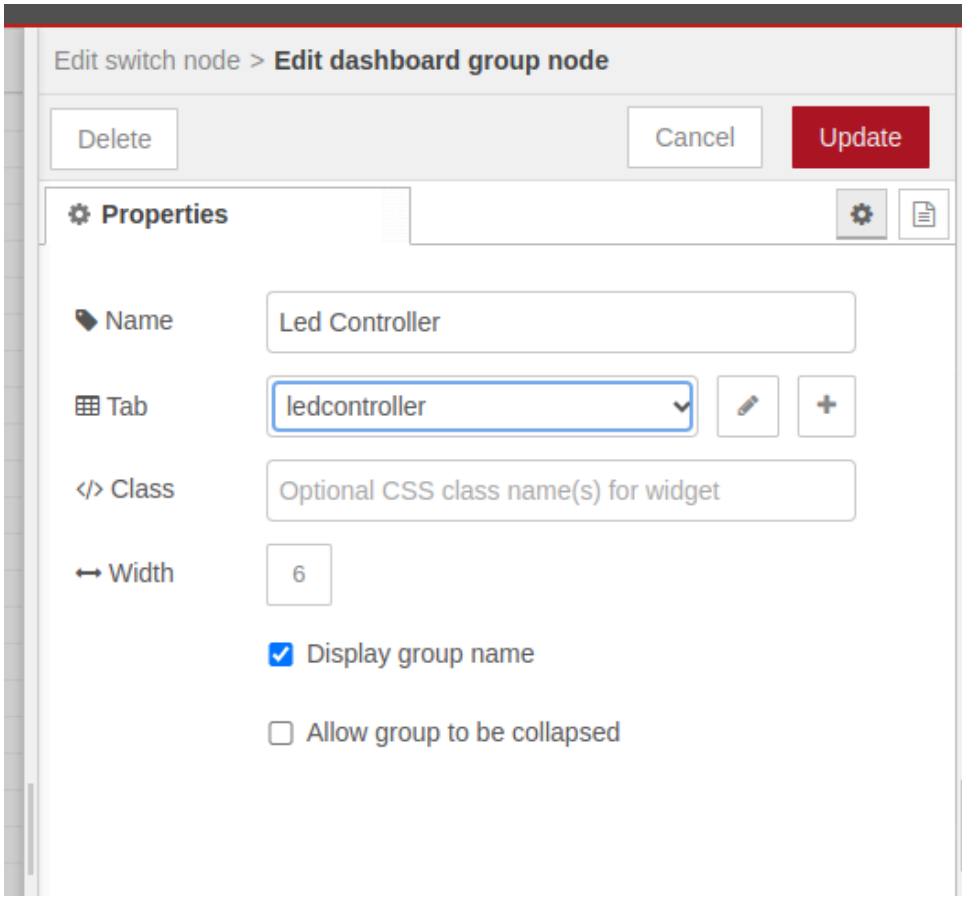
Double-click the rpi gpio out node. In the settings, select the GPIO4 pin that the LED1 is connected to (e.g., pin 18). Set the output type to digital output.

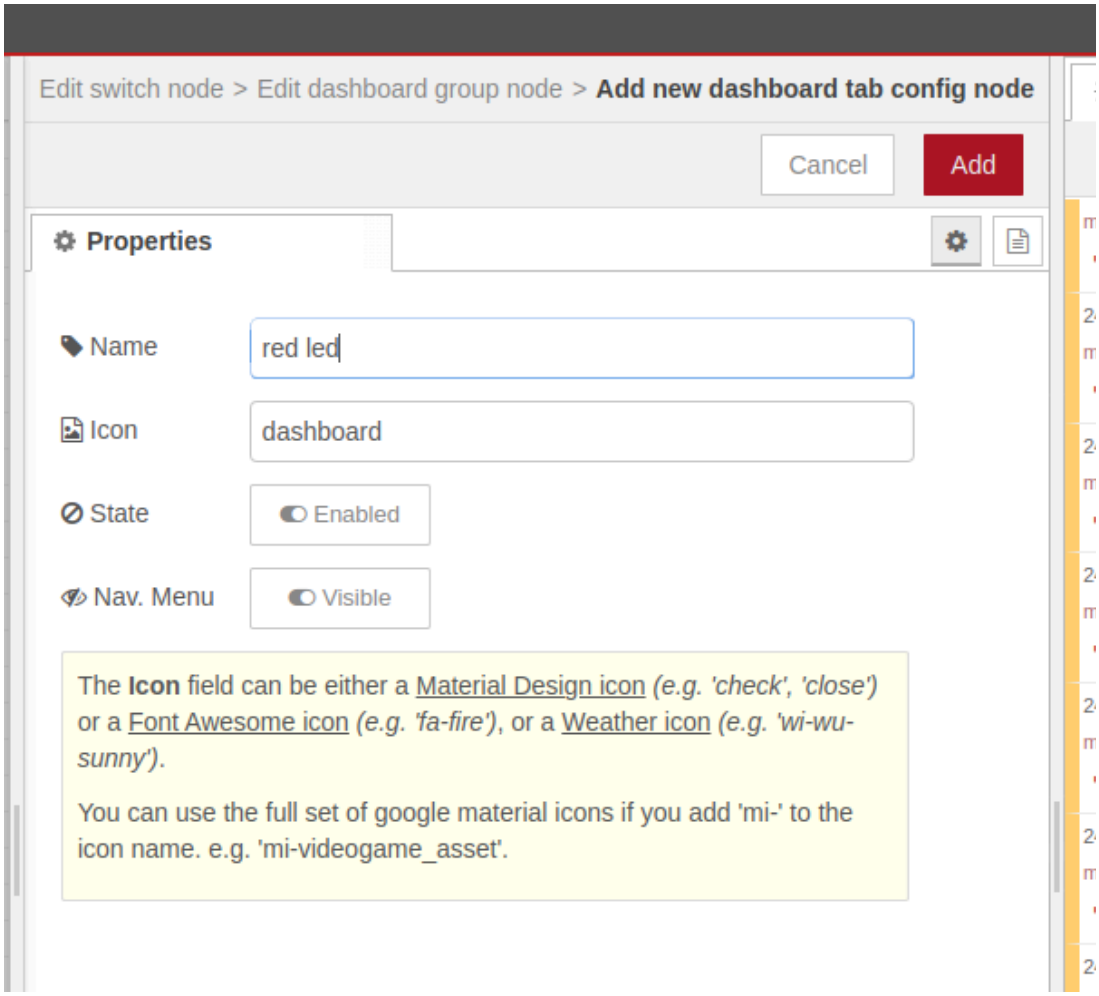
Connect the output of the ui_switch node to the input of the rpi gpio out node by dragging a line between them.

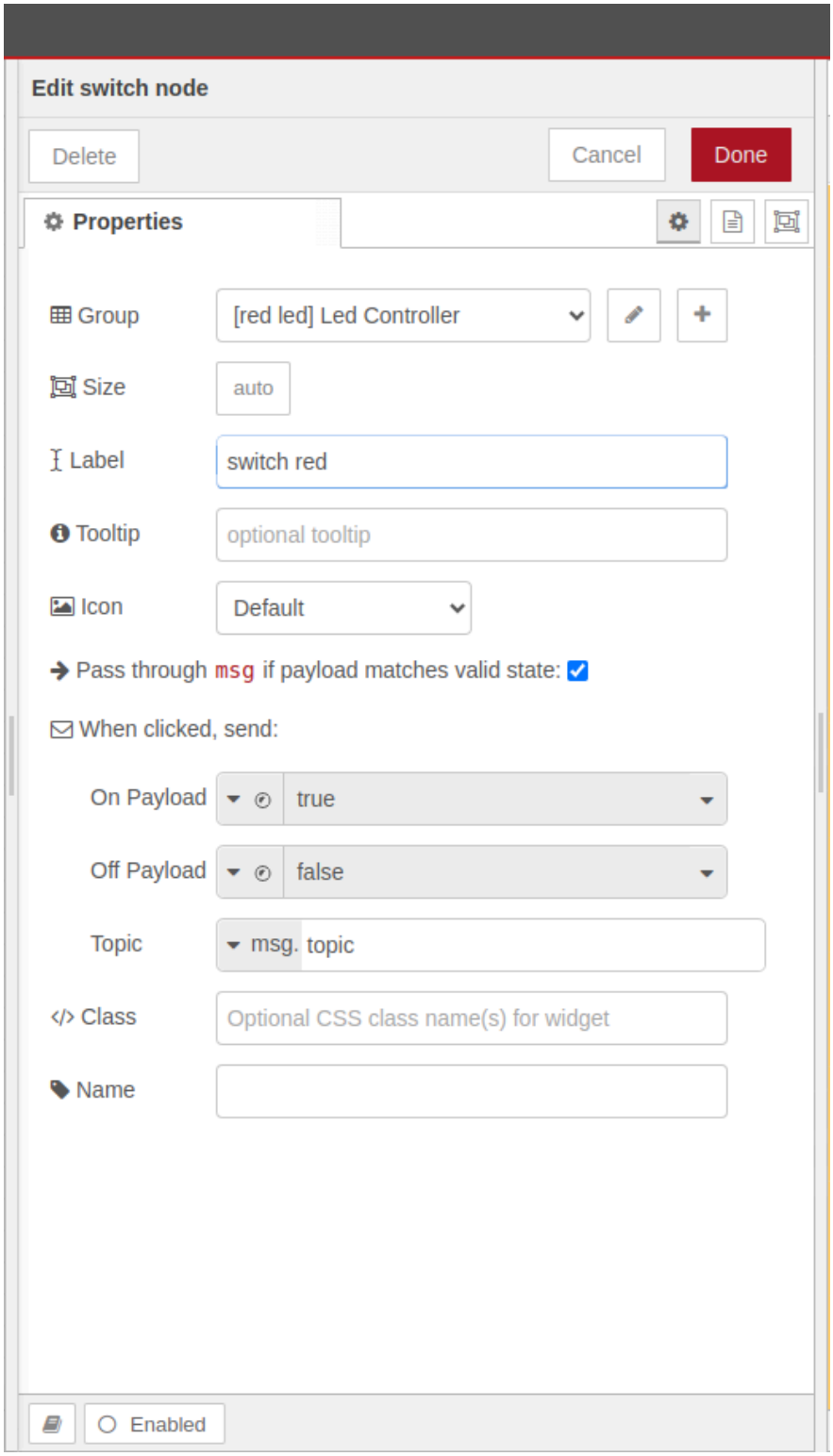



Edit rpi-gpio out node
Delete
Cancel
Done
Properties
Pin
3.3V Power - 1
2 - 5V Power
SDA1 - GPIO02 - 3
4 - 5V Power
SCL1 - GPIO03 - 5
6 - Ground
GPIO04 - 7
8 - GPIO14 - TxD
Ground - 9
10 - GPIO15 - RxD
GPIO17 - 11
12 - GPIO18
GPIO27 - 13
14 - Ground
GPIO22 - 15
16 - GPIO23
3.3V Power - 17
18 - GPIO24
MOSI - GPIO10 - 19
20 - Ground
MISO - GPIO09 - 21
22 - GPIO25
SCLK - GPIO11 - 23
24 - GPIO8 - CE0
Ground - 25
26 - GPIO7 - CE1
SD - 27
28 - SC
GPIO05 - 29
30 - Ground
GPIO06 - 31
32 - GPIO12
GPIO13 - 33
34 - Ground
GPIO19 - 35
36 - GPIO16
GPIO26 - 37
38 - GPIO20
Ground - 39
40 - GPIO21
BCM GPIO
4
Type
Digital output
Initialise pin state?
Name
Name
Pins in Use: 4,5,6
Enabled


Edit switch node
Delete
Cancel
Done
Properties
Group
none
Size
auto
Label
switch
Tooltip
optional tooltip
Icon
Default
Pass through msg if payload matches valid state:
When clicked, send:
On Payload
true
Off Payload
false
Topic
msg. topic
Class
Optional CSS class name(s) for widget
Name
Enabled


Edit switch node > Edit dashboard group node
Delete
Cancel
Update
Properties
Name
Led Controller
Tab
ledcontroller
Class
Optional CSS class name(s) for widget
Width
6
Display group name
Allow group to be collapsed


Edit switch node > Edit dashboard group node > Add new dashboard tab config node
Cancel
Add
Properties
Name
red led
Icon
dashboard
State
Enabled
Nav. Menu
Visible
The Icon field can be either a Material Design icon (e.g. 'check', 'close') or a Font Awesome icon (e.g. 'fa-fire'), or a Weather icon (e.g. 'wi-wu-sunny').
You can use the full set of google material icons if you add 'mi-' to the icon name. e.g. 'mi-videogame_asset'.


Edit switch node
Delete
Cancel
Done
Properties
Group
[red led] Led Controller
Size
auto
Label
switch red
Tooltip
optional tooltip
Icon
Default
Pass through msg if payload matches valid state:
When clicked, send:
On Payload
true
Off Payload
false
Topic
msg.
topic
Class
Optional CSS class name(s) for widget
Name
Enabled

Perform the same steps for LED2 and LED3.

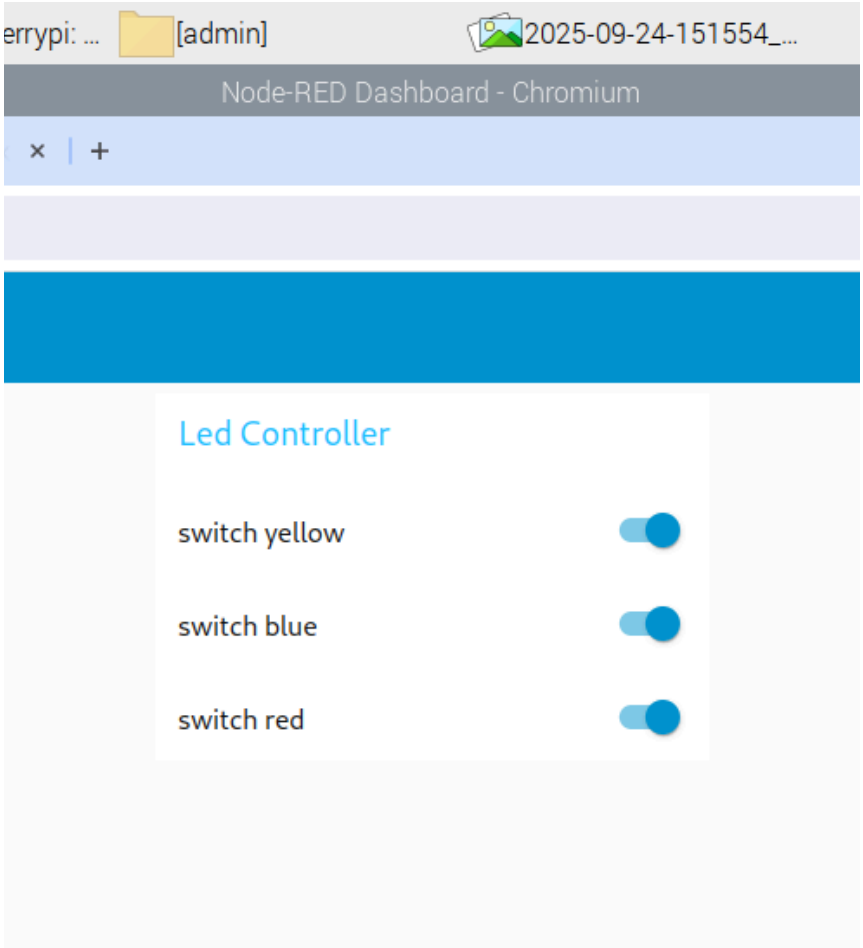
Click the red **"Deploy"** button in the top-right corner to save and activate your flow.

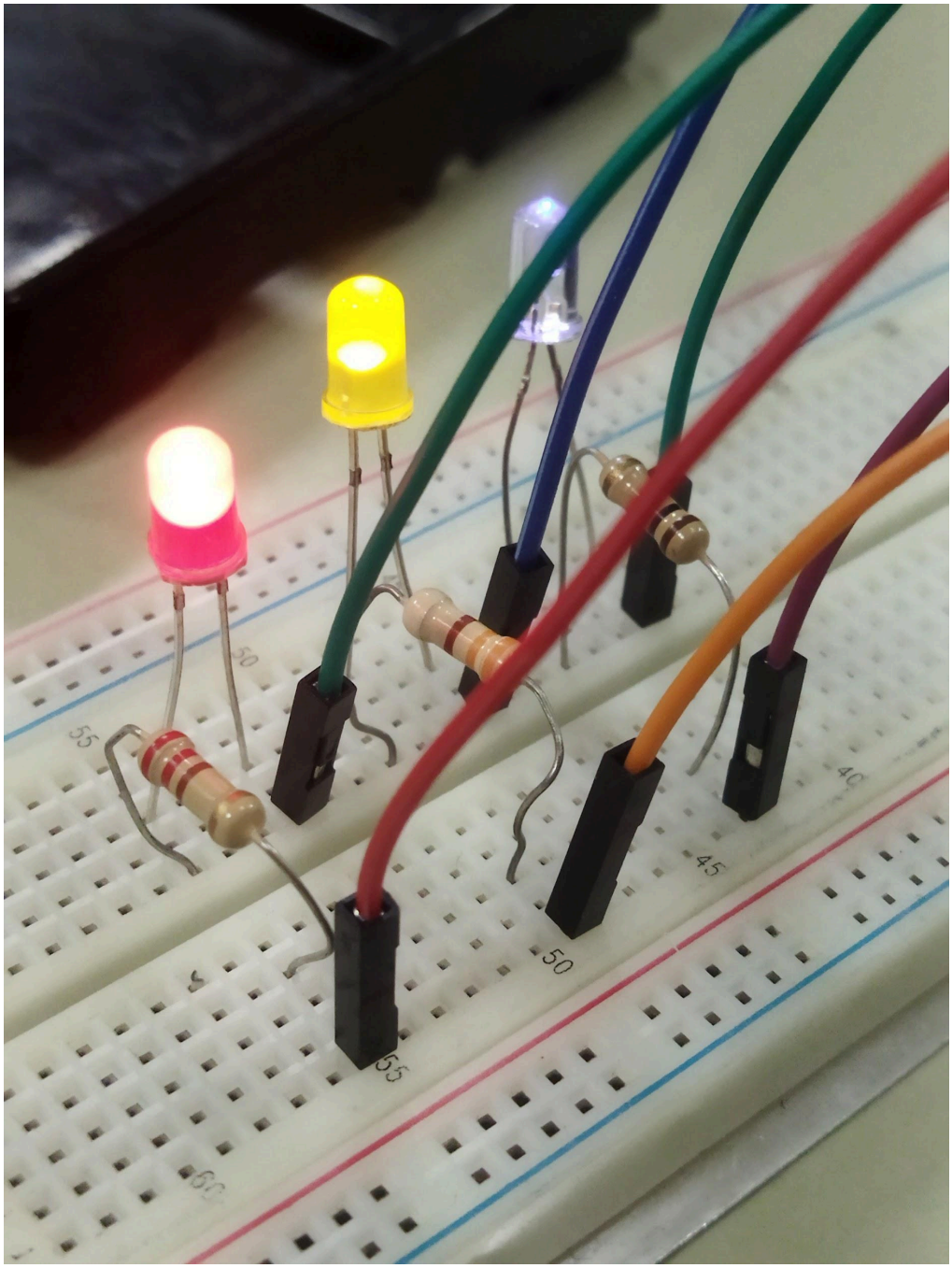
6. **Output**

After deploying the flow, open a new browser tab and navigate to the Node-RED dashboard URL, which is typically

http://<your-pi-ip-address>:1880/ui.

You will see a simple web interface with a switch. When you toggle the switch on the dashboard, the Node-RED flow will send a signal to the specified GPIO pin on the Raspberry Pi, turning the connected LED on or off. This demonstrates how a visual, flow-based programming environment like Node-RED can be used to create a simple yet powerful IoT platform for remote control of devices.




55
50
45
40
55
50

PRACTICAL NO : 4

Implement Microservices on IoT Device

Aim: To design and implement a simple microservices-based architecture on an IoT device (Raspberry Pi) for modular data processing and communication.

Apparatus Required

- **Hardware:**
  - Raspberry Pi (with Python 3.x installed)
  - Any IoT sensor (DHT11/DHT22) or simulated sensor data
- **Software:**
  - Python libraries: Flask, requests, json, time
  - Postman or browser to test APIs
  - Internet connection (if accessing APIs remotely)

Theory:

1. **Microservices Architecture**

   Microservices is an architectural style that structures an application as a collection of small, independent services. Each service:

   - **Performs a single function** (e.g., collect sensor data, process it, store it).
   - **Communicates with other services** over HTTP/REST APIs.
   - Can be **developed, deployed, and scaled independently**.

   This approach improves **scalability, maintainability, and fault tolerance** compared to a monolithic (single-application) structure, which is crucial for distributed IoT systems.

   Microservices in an IoT Context

   A typical IoT data pipeline can be broken down into the following microservices:

   a. **Sensor Microservice:** Collects raw data from physical sensors or generates simulated data.
   b. **Processing Microservice:** Performs transformations, filtering, or aggregation on the raw data.
   c. **Storage/Logging Microservice:** Saves processed data to a file or database (not implemented in this simplified practical).
   d. **Control Microservice:** Triggers actuators based on processed data (not implemented in this simplified practical).

2. **Flask:** A lightweight micro web framework used to quickly build the RESTful API endpoints for each microservice.
3. **Requests:** An HTTP library used by the Processing Microservice to make easy and intuitive HTTP calls to fetch data from the Sensor Microservice.
4. **JSON (JavaScript Object Notation):** A built-in Python package used for serializing and deserializing (encoding and decoding) data into the standard, human-readable format used for API communication.
5. **Time:** A standard Python module used to handle time-related operations and potentially introduce delays
6. **Random:** A standard Python module used to generate pseudo-random numbers

Procedure:

1. **Set Up Raspberry Pi**
    a. **Boot Raspberry Pi** and connect to the terminal.
    b. **Install Flask and requests** (if not already installed). Flask is a web framework for the API, and requests is for making HTTP calls between services, use the following command in terminal
        a. pip install Flask
        b. pip install requests

2. **Create Sensor Microservice (File: sensor_service.py)**
   This service simulates reading a temperature and humidity sensor and exposes the data via a RESTful API on port 5001.
    a. Create a file named sensor_service.py.
    b. Write the Python code as shown in the Code section below.
    c. The main function is get_sensor_data(), which generates random temperature and humidity values and returns them as JSON data.

   Code:

```
from flask import Flask, jsonify
import random

app = Flask(__name__)

@app.route('/get_sensor_data', methods=['GET'])
def get_sensor_data():
    temperature = round(random.uniform(20, 30), 2)
    humidity = round(random.uniform(40, 60), 2)
```

```
    data = {
        "temperature": temperature,
        "humidity": humidity
    }
    return jsonify(data)

if __name__ == '__main__':
    # Run on port 5001
    app.run(host='0.0.0.0', port=5001)
```

3. **Create Sensor Microservice (File: sensor_service.py)**

   This service simulates reading a temperature and humidity sensor and exposes the data via a RESTful API on port 5001.

   a. Create a file named **sensor_service.py.**
   b. Write the Python code as shown in the Code section below.
   c. The main function is get_sensor_data(), which generates random temperature and humidity values and returns them as JSON data.

   Code:

```
from flask import Flask, jsonify
import requests

app = Flask(__name__)

@app.route('/process_data', methods=['GET'])
def process_data():
    response = requests.get("http://localhost:5001/get_sensor_data")

    # Check if the request was successful
    if response.status_code != 200:
        return jsonify({"error": "Failed to retrieve sensor data"}), 500

    # Get the JSON data from the sensor service
    data = response.json()

    # Simple processing: Check if temperature is > 28
    if data['temperature'] > 28:
        data['alert'] = "High Temp"
    else:
```

```
        data['alert'] = "Normal"

    return jsonify(data)

if __name__ == '__main__':
    # Run on port 5002
    app.run(host='0.0.0.0', port=5002)
```

4. **Run Both Microservices**
    a. Open two separate terminal windows on the Raspberry Pi.
    b. In the **first terminal**, run the Sensor Microservice:
    python3 sensor_service.py
    c. In the **second terminal**, run the Processing Microservice:
    python3 process_service.py

5. **Test Using Browser or Postman**
    a. Open a web browser or Postman on a PC or the Raspberry Pi.
    b. Access the Processing Microservice API: http://localhost:5002/process_data (or use the Pi's actual IP address instead of localhost if testing remotely).
    c. The output will be the processed data, which includes the raw sensor values plus the calculated alert status.
6. **Observe**
    a. Each service runs **independently**.
    b. If you stop one, others can still run (showing decoupling of services).
    c. You can scale services (e.g., multiple sensor services) without touching others.

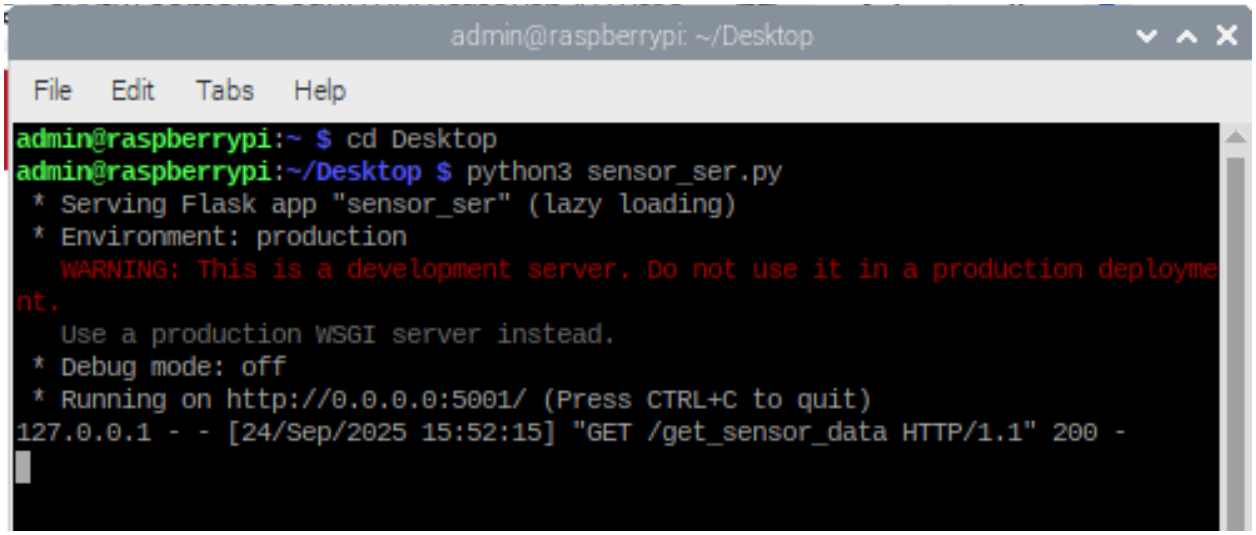
Sample Output
Sensor Microservice Response:

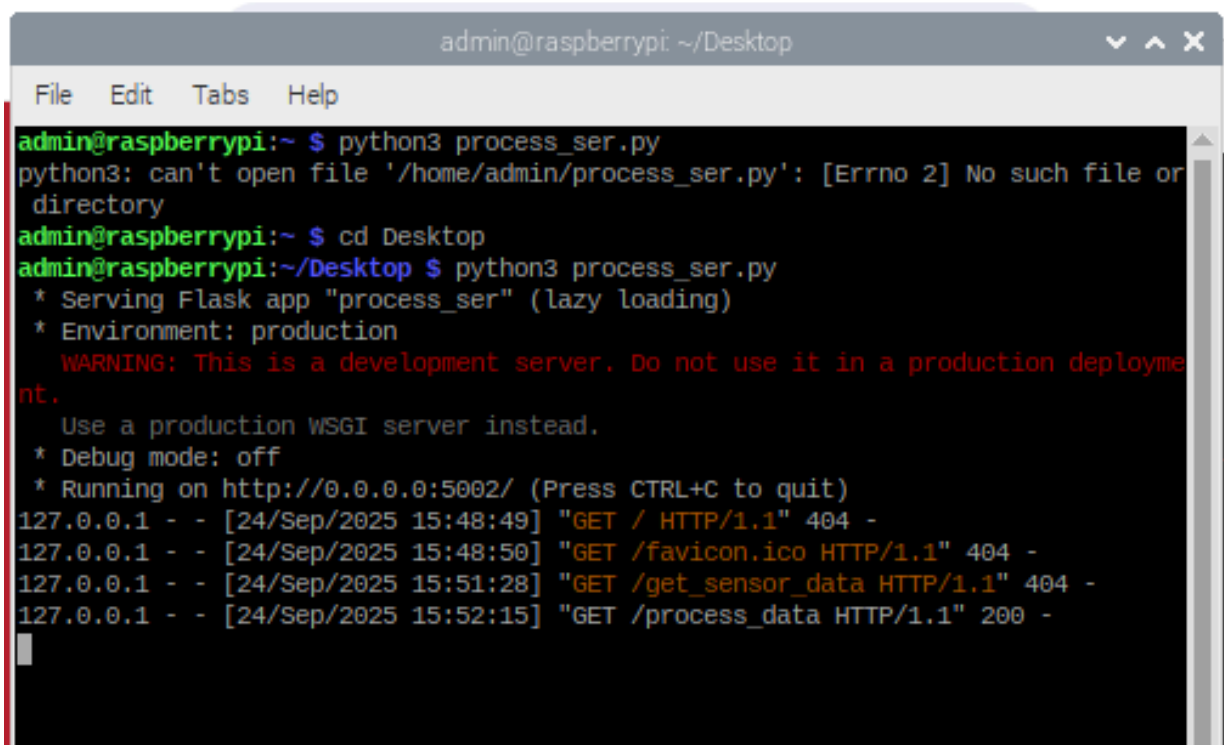
```
{
  "temperature": 27.53,
  "humidity": 44.8
}
```

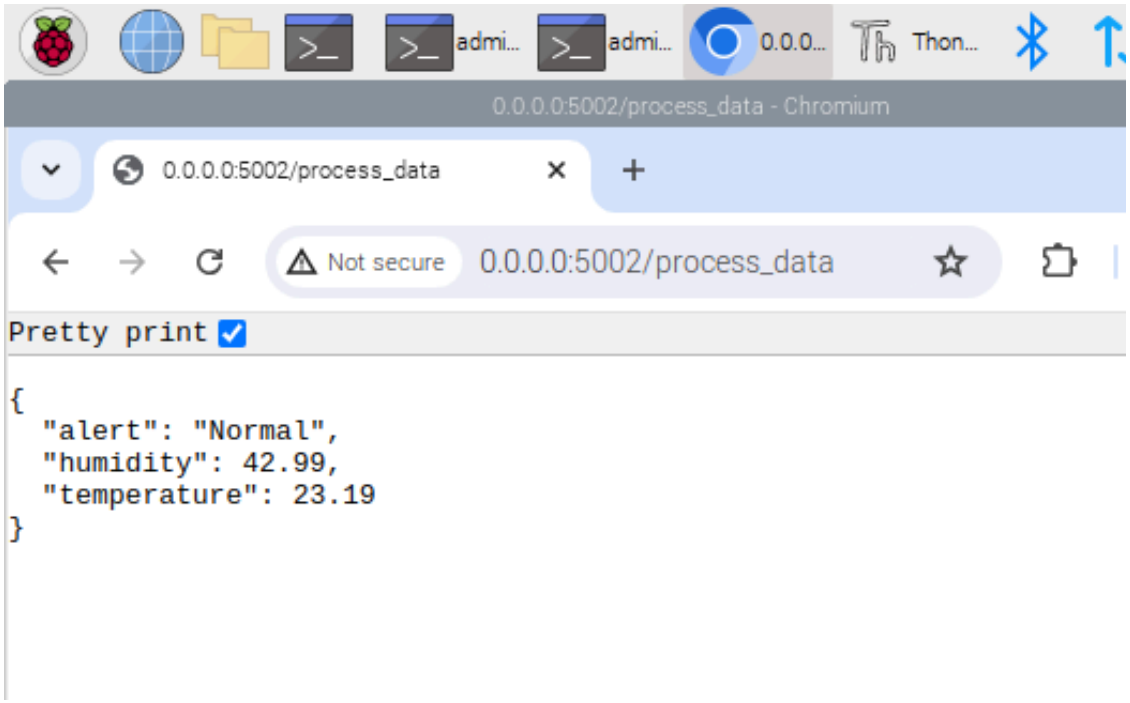
Processing Microservice Response:

```
{
  "temperature": 27.53,
  "humidity": 44.8,
  "alert": "Normal"
}
```

Output:


admin@raspberrypi: ~/Desktop
File Edit Tabs Help
admin@raspberrypi:~ $ cd Desktop
admin@raspberrypi:~/Desktop $ python3 sensor_ser.py
* Serving Flask app "sensor_ser" (lazy loading)
* Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Debug mode: off
* Running on http://0.0.0.0:5001/ (Press CTRL+C to quit)
127.0.0.1 - - [24/Sep/2025 15:52:15] "GET /get_sensor_data HTTP/1.1" 200 -


admin@raspberrypi: ~/Desktop
File Edit Tabs Help
admin@raspberrypi:~ $ python3 process_ser.py
python3: can't open file '/home/admin/process_ser.py': [Errno 2] No such file or directory
admin@raspberrypi:~ $ cd Desktop
admin@raspberrypi:~/Desktop $ python3 process_ser.py
* Serving Flask app "process_ser" (lazy loading)
* Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Debug mode: off
* Running on http://0.0.0.0:5002/ (Press CTRL+C to quit)
127.0.0.1 - - [24/Sep/2025 15:48:49] "GET / HTTP/1.1" 404 -
127.0.0.1 - - [24/Sep/2025 15:48:50] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [24/Sep/2025 15:51:28] "GET /get_sensor_data HTTP/1.1" 404 -
127.0.0.1 - - [24/Sep/2025 15:52:15] "GET /process_data HTTP/1.1" 200 -


admi...
admi...
0.0.0...
Thon...
0.0.0.0:5002/process_data - Chromium
0.0.0.0:5002/process_data
Not secure
0.0.0.0:5002/process_data
Pretty print
{
"alert": "Normal",
"humidity": 42.99,
"temperature": 23.19
}

PRACTICAL NO : 5

Create a Blockchain on Raspberry Pi and Authenticate IoT Devices

Aim: To create a simple blockchain on Raspberry Pi, implement it to store authorized IoT device IDs, and authenticate IoT devices before accepting their data.

Apparatus Required:

- **Hardware:**
  - Raspberry Pi (any model with Python 3.x)
  - Internet connection
  - (Optional) DHT11 sensor / simulated IoT device for data generation
- **Software:**
  - Python libraries: hashlib, json, time
  - Thonny / VS Code for coding
  - Terminal or SSH access to Raspberry Pi

Theory:

Blockchain is a distributed ledger technology where data is stored in blocks linked together using cryptographic hashes.

In IoT, blockchain can be used to:

- Register and authenticate devices to prevent unauthorized access.
- Ensure data integrity by linking each block with the previous one.
- Create tamper-proof logs of which device sent data and when.

In this experiment:

1. We build a simple blockchain on Raspberry Pi.
2. Each block contains a list of authorized device IDs.
3. When a device tries to send data, the blockchain is queried to check if it is authorized.
4. If authorized, the data is accepted; otherwise, it is rejected.

This approach simulates secure IoT authentication using blockchain.

Procedure:

1. **Setup Raspberry Pi**
   - Boot the Raspberry Pi and connect it to a monitor/SSH.
   - Open Thonny or VS Code.
   - Ensure Python 3.x is installed (python3 --version).
2. **Create a New Python File**
   - Name it iot_blockchain_auth.py.
3. **Write the Blockchain Code**
   - Define a IoTBlockchain class with:
     - create_block() to create blocks.

- proof_of_work() to generate proof.
- hash() to compute block hash.
- is_device_authorized() to check if a device is registered.

4. **Initialize Blockchain**
    - Create the Genesis block with pre-approved device IDs like ["device_01", "device_02"].
5. **Add Blocks**
    - Create additional blocks to simulate a growing chain.
6. **User Input for Device Authentication**
    - Ask the user to input a device ID.
    - If the device is present in the last block's list → Print "Authorized".
    - If not → Print "Rejected".
7. **Test the System**
    - Run the code using python3 iot_blockchain_auth.py.
    - Enter different device IDs (both valid and invalid).
    - Observe results on the terminal.
8. **Stop Execution**
    - Exit by typing exit in the input prompt or pressing Ctrl+C.

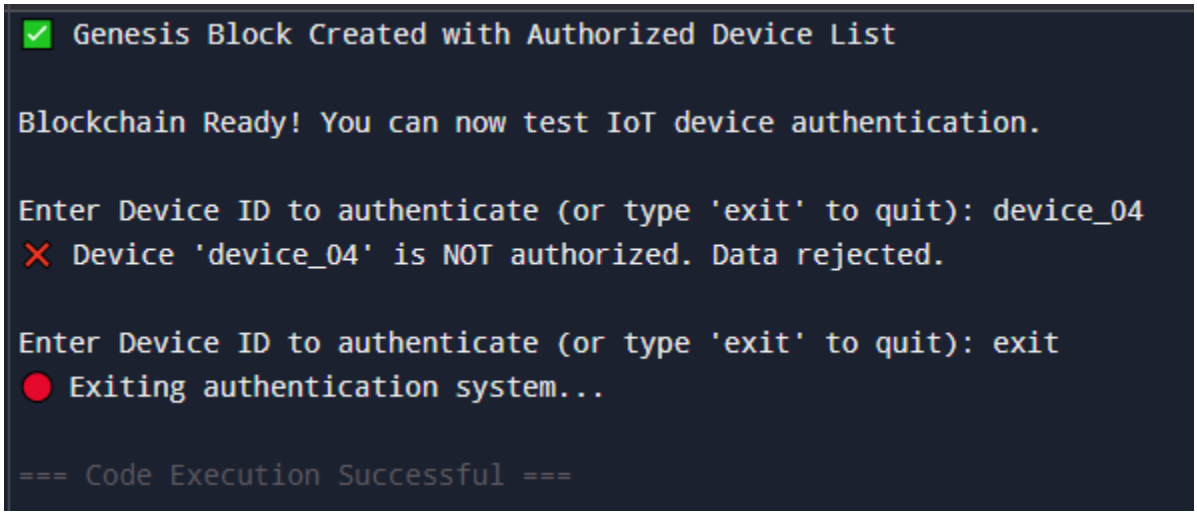
Code:

```
import hashlib, json, time

class IoTBlockchain:
    def __init__(self):
        self.chain = []
        self.authorized_devices = ["device_01", "device_02", "device_03"]  # pre-approved devices
        self.create_block(proof=1, previous_hash='0')

    def create_block(self, proof, previous_hash):
        block = {
            'index': len(self.chain) + 1,
            'timestamp': time.time(),
            'proof': proof,
            'previous_hash': previous_hash,
            'authorized_devices': self.authorized_devices.copy()
        }
        self.chain.append(block)
        return block

    def get_previous_block(self):
```

```
        return self.chain[-1]
    def proof_of_work(self, previous_proof):
        new_proof = 1
        while True:
            hash_operation = hashlib.sha256(str(new_proof**2 -
previous_proof**2).encode()).hexdigest()
            if hash_operation[:4] == '0000':
                return new_proof
            new_proof += 1

    def hash(self, block):
        return hashlib.sha256(json.dumps(block, sort_keys=True).encode()).hexdigest()

    def is_device_authorized(self, device_id):
        return device_id in self.get_previous_block()['authorized_devices']


# ------------------- MAIN PROGRAM -------------------
blockchain = IoTBlockchain()
print("✅ Genesis Block Created with Authorized Device List")

# Adding a few more blocks (optional)
for i in range(2):
    prev_block = blockchain.get_previous_block()
    proof = blockchain.proof_of_work(prev_block['proof'])
    prev_hash = blockchain.hash(prev_block)
    blockchain.create_block(proof, prev_hash)

print("\nBlockchain Ready! You can now test IoT device authentication.")
# --------------- User Input Loop -------------------
while True:
    user_input = input("\nEnter Device ID to authenticate (or type 'exit' to quit): ").strip()

    if user_input.lower() == "exit":
        print("🔴 Exiting authentication system...")
        break

    if blockchain.is_device_authorized(user_input):
        print(f"✅ Device '{user_input}' authenticated successfully! Data accepted.")
    else:
```

print(f"❌ Device '{user_input}' is NOT authorized. Data rejected.")

Output:



PRACTICAL NO : 6

Create programs using Microsoft Cognitive APIs for IoT.

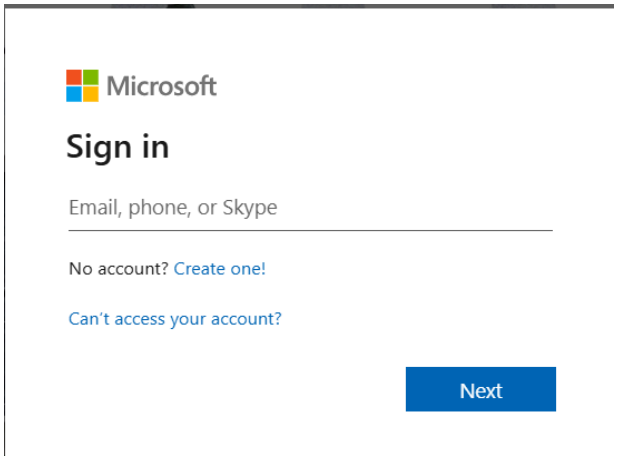
Aim: To perform create programs using Microsoft cognitive apis for iot and understand its working.

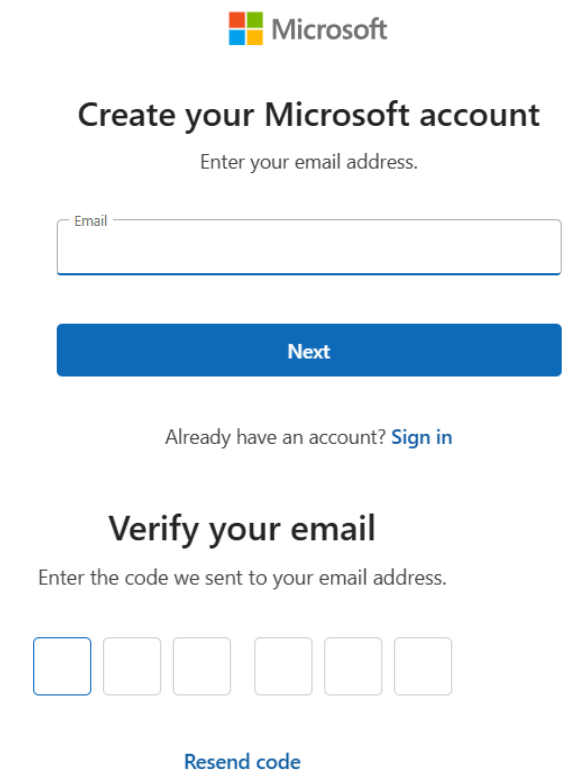
Apparatus Required:
Python
Azure accounts
Request Library

Theory:
Azure Account
An Azure account is a unique identifier with a login and password that grants access to Microsoft Azure, a cloud computing platform offering over 200 products and services like compute, storage, and AI. It serves as a top-level container for all your Azure cloud resources and acts as the entry point for managing and deploying these services. Within an account, you can have one or more subscriptions, which are used to group resources for business or technical purposes and handle billing.

Request Library
The Requests library is a popular and powerful Python library designed to simplify the process of making HTTP requests. It provides a user-friendly API for interacting with web services and consuming data from APIs or web pages.

Click on start free button


Microsoft
Sign in
Email, phone, or Skype
No account? Create one!
Can't access your account?
Next


Microsoft
Create your Microsoft account
Enter your email address.
Email
Next
Already have an account? Sign in

Verify your email
Enter the code we sent to your email address.
Resend code

Add some details

Add your country/region and birthdate to help us apply age-appropriate settings to your Microsoft account.

Country/Region
India

Month | Day | Year

If a child is using this device, create a child account by entering your child's date of birth.

A child account enables you to enforce parental controls and impose usage limits for this device for reasons of privacy and safety. You can manage these settings at microsoft.com/family

Next

Add your name

Add your name to your Microsoft account.

First name

Last name

I would like information, tips, and offers about Microsoft products and services.

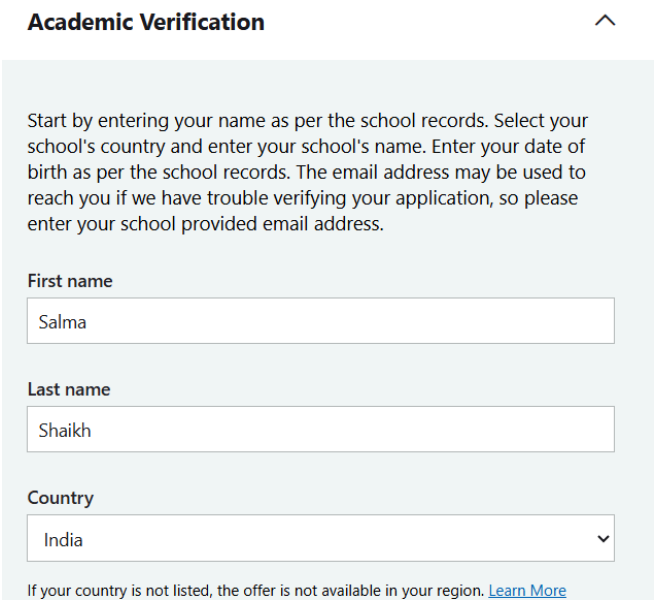
By selecting Next, I agree to the Microsoft Services Agreement and Privacy Statement.

Next

Let's prove you're human

Press and hold the button.

Press and hold



Procedure:
Getting Azure Cognitive Services Subscription Key

Step 1: Create a Free Azure Account
Go to https://azure.microsoft.com/free
Sign up with a college email ID (or personal email).
Microsoft gives $200 free credit for 30 days (enough for labs) + limited free tier services.

Step 2: Create Cognitive Services Resource
After logging in to Azure Portal → Search for Cognitive Services → Create.
Select:

- Subscription (your free subscription)
- Resource Group (create a new one, e.g., IoTLabGroup)
- Region (choose closest to you)
- Pricing Tier: Free (F0) if available.

Click Review + Create → then Create.

Step 3: Get Keys and Endpoint
After deployment → Go to the Cognitive Services resource.
In Keys and Endpoint section:

- Copy Key1 → this is your subscription_key
- Copy Endpoint URL → this is your endpoint

Step 4: Install Required Library

Open Command Prompt (CMD) and run:
pip install requests

Code:

```
import requests, json
subscription_key = '<YOUR_AZURE_KEY>'
endpoint = '<YOUR_ENDPOINT>'
image_url =
'https://learn.microsoft.com/azure/cognitive-services/computer-vision/media/quickstarts/presentation.png'
headers = {'Ocp-Apim-Subscription-Key': subscription_key, 'Content-Type': 'application/json'}
params = {'returnFaceAttributes': 'age,gender,emotion'}
response = requests.post(endpoint + '/face/v1.0/detect', headers=headers, params=params,
json={'url': image_url})
print(json.dumps(response.json(), indent=2))
```

PRACTICAL NO : 7

Use IoT Device with AWS IoT Core

Aim: To connect a Raspberry Pi IoT device with AWS IoT Core using the MQTT protocol and publish telemetry data securely to the cloud.

Apparatus Required:

- **Hardware:** Raspberry Pi, SD card (16 / 32 GB), power supply, internet connection
- **Software:**
    - Python 3.x
    - AWS IoT Core account
    - AWSIoTPythonSDK library (pip install AWSIoTPythonSDK)
    - Certificates: AmazonRootCA1.pem, private.pem.key, certificate.pem.crt

Theory:

AWS IoT Core enables devices to communicate securely with the cloud via the **MQTT** publish/subscribe model.
Each device (Thing) is authenticated with unique certificates and sends data over a secure (TLS 8883) connection.
Raspberry Pi acts as the **publisher**, and the AWS IoT Core **MQTT test client** acts as the **subscriber**.

Procedure:

1. **Open AWS IoT Core** → *Manage* → *Things* → *Create Thing* → *Create single thing*
2. **Download** certificates (AmazonRootCA1.pem, certificate.pem.crt, private.pem.key).
3. **Note** the AWS IoT endpoint from *Settings* → *Device data endpoint*.
4. On Raspberry Pi:
5. pip install AWSIoTPythonSDK
6. mkdir ~/aws_iot && cd ~/aws_iot

Copy all certificate files here.

7. **Create a Python file:** aws_iot_publish.py

Code Snippet

```
from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient
import time, json, random

# Create MQTT client
client = AWSIoTMQTTClient("RaspberryPiClient")
# Replace with your AWS IoT endpoint
```

```
client.configureEndpoint("a3k7odshaiipe8-ats.iot.us-east-1.amazonaws.com", 8883)
# Provide certificate paths
client.configureCredentials(
   "AmazonRootCA1.pem",
   "private.pem.key",
   "certificate.pem.crt"
)

client.connect()
print("Connected to AWS IoT Core")

# Publish random sensor data continuously
while True:
   payload = {"temperature": random.randint(20, 35)}
   client.publish("iot/topic", json.dumps(payload), 1)
   print("Published:", payload)
   time.sleep(5)
```

Run the Code

```
python3 aws_iot_publish.py
```

Verify on AWS

1. In AWS IoT Core → MQTT Test Client, subscribe to:
2. iot/topic
3. Observe live JSON messages arriving every 5 seconds.

Sample Output

Terminal (Raspberry Pi):

```
Connected to AWS IoT Core
Published: {'temperature': 29}
Published: {'temperature': 34}
```

AWS MQTT Test Client:

```
{"temperature": 34}
```

PRACTICAL NO : 8

Send Telemetry from Device to Azure IoT Hub and Read with Service Application

Aim: To send telemetry data from an IoT device (Raspberry Pi) to Azure IoT Hub and understand how the data can be received and processed by cloud applications.

Apparatus Required:

- **Hardware:**
  - Raspberry Pi board
  - SD card (16 / 32 GB)
  - Power supply, HDMI cable
  - Internet connection
  - Optional: DHT11 sensor or simulated data
- **Software:**
  - Python 3.x
  - Azure IoT Device SDK for Python (pip install azure-iot-device)
  - Azure account with IoT Hub service

Theory:

Azure IoT Hub is a cloud platform for connecting, monitoring, and managing IoT devices. It uses secure communication protocols (MQTT, AMQP, HTTPS) to exchange data between **devices** and **cloud services**.

- **Device-to-Cloud (D2C)** → Device sends telemetry (e.g., temperature, humidity) to IoT Hub.
- **Cloud-to-Device (C2D)** → Azure sends commands or configuration back to devices.

This practical demonstrates D2C communication:
the Raspberry Pi publishes telemetry to the IoT Hub, which can then be viewed using Azure tools or routed to storage/Event Hub.

Procedure:

1. **Create an IoT Hub**
   - Log in to the Azure Portal.
   - Click **Create a Resource → Internet of Things → IoT Hub**.
   - Choose subscription, resource group, and region → Click **Review + Create**.
2. **Register a Device**
   - Inside your IoT Hub → Go to **Devices → Add Device**.
   - Enter a name (e.g., RaspberryPi01) and click **Save**.
   - Copy the **Primary Connection String** (used in your Python script).
3. **Install the Azure IoT SDK**
4. **pip install azure-iot-device**

5. **Create a Python File**
    - File name: azure_iot_send.py
    - Paste the following code:

Code Snippet

```
from azure.iot.device import IoTHubDeviceClient, Message
import random, time

# Replace with your device connection string from Azure IoT Hub
CONNECTION_STRING = "<YOUR_DEVICE_CONNECTION_STRING>"

# Create IoT Hub client
client = IoTHubDeviceClient.create_from_connection_string(CONNECTION_STRING)

print("Connected to Azure IoT Hub... Sending telemetry data.")

while True:
    # Simulate sensor data
    temperature = round(random.uniform(20.0, 35.0), 2)
    humidity = round(random.uniform(40.0, 70.0), 2)

    # Create message payload
    msg = Message(f'{{"temperature": {temperature}, "humidity": {humidity}}}')

    # Send message
    client.send_message(msg)
    print(f"Sent message: {msg}")
    time.sleep(5)
```

6. **Run the Code**

On Raspberry Pi or laptop:

```
python3 azure_iot_send.py
```

7. **Verify in Azure Portal**
    1. Go to **IoT Hub → Devices → RaspberryPi01 → Message to Device**.
    2. Open **Monitoring → Built-in Endpoints → Events** or use **Azure CLI** to check telemetry:
    3. az iot hub monitor-events --hub-name <YourHubName>
    4. You should see real-time messages arriving.

Sample Output

Terminal (Raspberry Pi):

Connected to Azure IoT Hub... Sending telemetry data.
Sent message: {"temperature": 29.6, "humidity": 55.1}
Sent message: {"temperature": 31.2, "humidity": 48.9}

Azure Monitor (CLI):

{
  "temperature": 29.6,
  "humidity": 55.1
}

PRACTICAL NO : 9

Face Detection Using IoT Device

Aim: Detect human faces in real-time using a Raspberry Pi camera or USB webcam and display detection results.

Apparatus / Requirements:
Raspberry Pi 3/4 with Raspberry Pi OS
Pi Camera Module (or USB Webcam)
Monitor/Keyboard/Mouse (or SSH access)
Python 3 installed
Required Libraries:
opencv-python for image processing
numpy for arrays
picamera2 (if using Pi camera)

Theory:

Pi Camera Module

A Raspberry Pi Camera Module is a small, purpose-built camera designed to connect to a Raspberry Pi computer via its CSI (Camera Serial Interface) port using a ribbon cable, enabling users to capture high-definition photos and videos for projects like security systems, robotics, and creative applications. These modules feature a Sony IMX sensor, offer various resolutions, support video recording in formats like 1080p, and are supported by the Raspberry Pi operating system and software libraries, such as Picamera2, for easy integration and control.

Opencv-Python

OpenCV, or Open Source Computer Vision Library, is a powerful open-source library primarily used for computer vision, machine learning, and image processing tasks. While originally written in C++, it provides robust Python bindings, allowing developers to leverage its extensive functionalities within Python applications.

Numpy

NumPy is an open-source Python library for numerical and scientific computing, providing a powerful multidimensional array object and a collection of high-level mathematical functions for efficient data processing. It is a fundamental tool for data science and machine learning because it enables fast, efficient, and sophisticated operations on large datasets that are not possible with standard Python lists.

Picamera2

Picamera2 is a Python library that gives convenient access to the camera system of the Raspberry Pi. It is designed for cameras connected with the flat ribbon cable directly to the connector on the Raspberry Pi itself, and not for other types of camera, although there is some limited support for USB cameras.

Procedure:

A. Hardware Setup
Connect the Raspberry Pi Camera Module to the Pi's camera port (or plug in USB webcam).

Enable camera support:
sudo raspi-config
Go to Interface Options → Enable Camera → Reboot.

B. Install Required Libraries
Run:
sudo apt update
sudo apt install python3-opencv
pip install numpy

If using Pi camera module:
pip install picamera2

Code:

```
import cv2
from picamera2 import Picamera2

# Initialize PiCamera2
picam2 = Picamera2()
picam2.configure(picam2.create_preview_configuration(main={"format": 'XRGB8888', "size": (640, 480)}))
picam2.start()

# Load Haar Cascade
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')


while True:
```

```
    frame = picam2.capture_array()  # Capture frame from PiCamera
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Detect faces
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30,
30))

    # Draw bounding boxes
    for (x, y, w, h) in faces:
        cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)

    cv2.imshow('Face Detection', frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
    cv2.destroyAllWindows()
```

OR

```
  #!/usr/bin/python3

  import cv2
  import os
  import time
  from picamera2 import Picamera2



  # download the cascade file from the link below
  #https://github.com/opencv/opencv/blob/4.x/data/haarcascades/haarcascade_frontalface_defa
  ult.xml

  face_detector = cv2.CascadeClassifier("/home/pi/Face
  Recognition/haarcascade_frontalface_default.xml")
  cv2.startWindowThread()



  picam2 = Picamera2()
```

```
picam2.configure(picam2.create_preview_configuration(main={"format": 'XRGB8888',
"size": (640, 480)}))
picam2.start()

# Create a directory to store detected faces
output_directory = "detected_faces"
os.makedirs(output_directory, exist_ok=True)

while True:
  im = picam2.capture_array()

  grey = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
  faces = face_detector.detectMultiScale(grey, 1.1, 5)


  for (x, y, w, h) in faces:
     cv2.rectangle(im, (x, y), (x + w, y + h), (0, 255, 0))

  # Generate a unique filename using timestamp for every saved image
     timestamp = int(time.time())
     filename = os.path.join(output_directory, f"face_{timestamp}.jpg")
     cv2.imwrite(filename, im[y:y+h, x:x+w])  # Save only the detected face portion

  cv2.imshow("Camera", im)
  cv2.waitKey(1)
Or
 # Wait for 1ms & check if 'q' is pressed
    if cv2.waitKey(1) & 0xFF == ord('q'):
       print("Exiting Program...")
       break
```

TroubleShoot for first code:
Where the Cascade File Actually Is

When you install opencv-python using pip, OpenCV includes a folder containing multiple pre-trained Haar Cascade XML files (for face, eyes, smile, etc.).

On your Raspberry Pi or PC, the path is usually:
/usr/local/lib/python3.x/site-packages/cv2/data/

Inside this folder, you will find:
haarcascade_frontalface_default.xml
haarcascade_eye.xml
haarcascade_smile.xml
and many more.

The code:
cv2.data.haarcascades
returns this exact directory path so you don't have to hardcode it.

How to Verify It Exists
In Python, you can check like this:

```
import cv2
import os
print("Cascade Path:", cv2.data.haarcascades)
print("Exists?", os.path.exists(cv2.data.haarcascades + "haarcascade_frontalface_default.xml"))
```

If it prints True, then the file is available and you are good to go.

If File Is Missing
If for some reason your OpenCV installation doesn't have it:
Download it manually from OpenCV GitHub:
https://github.com/opencv/opencv/tree/master/data/haarcascades
Save it in your project folder and update code like this:

```
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
```

Run the program:

```
python3 face_detection.py
```

Face detection by image (without camera)

```
import cv2
import os

# Path to folder containing your images
folder_path = r"./images"   # <-- change this path

# Load Haar Cascade (pre-trained model)
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')

# Check if folder exists
if not os.path.exists(folder_path):
   print("Error: Folder not found. Check the path.")
else:
   # Loop through all files in folder
   for filename in os.listdir(folder_path):
      # Only process image files
      if filename.lower().endswith((".jpg", ".jpeg", ".png", ".bmp")):
         image_path = os.path.join(folder_path, filename)
         img = cv2.imread(image_path)

         if img is None:
            print(f"Could not read image: {filename}")
            continue

         # Convert to grayscale
         gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

         # Detect faces
         faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5,
minSize=(30, 30))

         # Draw rectangles
         for (x, y, w, h) in faces:
            cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)

         # Show image
         cv2.imshow(f"Detected Faces - {filename}", img)
```

```
        cv2.waitKey(0)
        cv2.destroyAllWindows()
print("✅ Face detection completed for all images!")
```

Output: