🧪 Practical No. 1: Display Single LED Blinking Pattern using Raspberry Pi

🧰 Requirements:

Hardware:

- Raspberry Pi board
- 1 × LED
- 1 × 330Ω resistor
- Breadboard
- Jumper wires

Software:

- Raspberry Pi OS
- Python 3 with RPi.GPIO library

⚡ Circuit Connection:

Raspberry Pi Pin	LED Connection
GPIO 11 (Pin 11)	LED anode (+) through 330Ω resistor
GND (Pin 6)	LED cathode (–)

💻 Code (Write this in exam notebook)

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BOARD)
GPIO.setup(11, GPIO.OUT)

for i in range(5):
    GPIO.output(11, True)
    print("LED ON")
    time.sleep(1)
```

```
    GPIO.output(11, False)
    print("LED OFF")
    time.sleep(1)

print("Done")
GPIO.cleanup()
```

▶ Procedure (Short Notes):

1. Connect LED to GPIO 11 with resistor and GND.
2. Write and save the program as `led_blink.py`.
3. Run using: `python3 led_blink.py`
4. Observe LED blinking ON and OFF every 1 second.

Practical No. 2: Display Three LED Blinking Pattern using Raspberry Pi

🧰 Requirements:

Hardware:

- Raspberry Pi board
- 3 × LEDs (Red, Yellow, Green)
- 3 × 330Ω resistors
- Breadboard
- Jumper wires

Software:

- Raspberry Pi OS
- Python 3 with `RPi.GPIO` library

⚡ Circuit Connection:

Raspberry Pi Pin	LED	Connection

GPIO 11 (Pin 11)	LED 1 (Red)	Through 330Ω resistor
GPIO 13 (Pin 13)	LED 2 (Yellow)	Through 330Ω resistor
GPIO 15 (Pin 15)	LED 3 (Green)	Through 330Ω resistor
GND (Pin 6)	All LEDs negative (–)	Common Ground

💻 Code (Write in exam):

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BOARD)
GPIO.setup(11, GPIO.OUT)
GPIO.setup(13, GPIO.OUT)
GPIO.setup(15, GPIO.OUT)

for i in range(5):
    GPIO.output(11, True)
    print("LED 1 ON")
    time.sleep(0.5)
    GPIO.output(11, False)
    time.sleep(0.5)

    GPIO.output(13, True)
    print("LED 2 ON")
    time.sleep(0.5)
    GPIO.output(13, False)
    time.sleep(0.5)

    GPIO.output(15, True)
    print("LED 3 ON")
    time.sleep(0.5)
    GPIO.output(15, False)
    time.sleep(0.5)

print("Done")
```

```
GPIO.cleanup()
```

▶ Procedure (Short):

1. Connect 3 LEDs to GPIO 11, 13, and 15.
2. Save program as `three_led_blink.py`.
3. Run: `python3 three_led_blink.py`
4. Observe LEDs blinking one after another.

Practical No. 3: Display Four LED Blinking Pattern using Raspberry Pi

🧰 Requirements:

Hardware:

- Raspberry Pi board
- 4 × LEDs (any colors)
- 4 × 330Ω resistors
- Breadboard
- Jumper wires

Software:

- Raspberry Pi OS
- Python 3 with `RPi.GPIO` library

⚡ Circuit Connection:

Raspberry Pi Pin	LED	Connection
GPIO 11 (Pin 11)	LED 1	Through 330Ω resistor
GPIO 13 (Pin 13)	LED 2	Through 330Ω resistor
GPIO 15 (Pin 15)	LED 3	Through 330Ω resistor
GPIO 16 (Pin 16)	LED 4	Through 330Ω resistor
GND (Pin 6)	All LEDs (–)	Common Ground

💻 Code (Write this in exam):

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BOARD)
GPIO.setup(11, GPIO.OUT)
GPIO.setup(13, GPIO.OUT)
GPIO.setup(15, GPIO.OUT)
GPIO.setup(16, GPIO.OUT)

for i in range(5):
    GPIO.output(11, True)
    print("LED 1 ON")
    time.sleep(0.5)
    GPIO.output(11, False)

    GPIO.output(13, True)
    print("LED 2 ON")
    time.sleep(0.5)
    GPIO.output(13, False)

    GPIO.output(15, True)
    print("LED 3 ON")
    time.sleep(0.5)
    GPIO.output(15, False)

    GPIO.output(16, True)
    print("LED 4 ON")
    time.sleep(0.5)
    GPIO.output(16, False)

print("Done")
GPIO.cleanup()
```

▶️ Procedure (Short):

1. Connect 4 LEDs to GPIO 11, 13, 15, and 16.
2. Save program as `four_led_blink.py`.
3. Run the program: `python3 four_led_blink.py`
4. Observe LEDs blinking in sequence.

🧪 Practical No. 4: Display Time over 4-Digit 7-Segment Display using Raspberry Pi

🧰 Requirements:

Hardware:

- Raspberry Pi board
- TM1637 4-digit 7-segment display module
- Jumper wires (Female-to-Female)
- Breadboard

Software:

- Raspberry Pi OS
- Python 3
- `tm1637` library

⚡ Circuit Connection (for TM1637 Module):

TM1637 Pin	Raspberry Pi Pin	Function
VCC	Pin 4 (5V)	Power supply
GND	Pin 14 (GND)	Ground
DIO	Pin 18 (GPIO 24)	Data line
CLK	Pin 16 (GPIO 23)	Clock line

💻 Code (Write in exam):

```
import time
import datetime
import tm1637

# CLK -> GPIO23 (Pin 16)
# DIO -> GPIO24 (Pin 18)
Display = tm1637.TM1637(clk=23, dio=24)

Display.Clear()
Display.SetBrightnes(1)

while True:
    now = datetime.datetime.now()
    hour = now.hour
    minute = now.minute
    second = now.second
    currenttime = [int(hour/10), hour%10, int(minute/10), minute%10]

    Display.Show(currenttime)
    Display.ShowDoublepoint(second % 2)
    time.sleep(1)
```

▶️ Procedure (Short):

1. Connect TM1637 display as per table above.
2. Install required library: `pip3 install tm1637`
3. Save program as `time_display.py`.
4. Run using:`python3 time_display.py`
5. Observe the current time displayed in HH:MM format with blinking colon.

Practical No. 5: Raspberry Pi based Oscilloscope

🎯 **Aim:** To replicate the signal visualization capabilities of an oscilloscope by using the **Raspberry Pi** and an **Analog-to-Digital Converter (ADC) module**.

🧰 **Requirements:**

Category	Item
Hardware	Raspberry Pi Model A/B/B+
	ADS1115 ADC (or similar, like the ADS1015 used in the code)
	Breadboard , Jumper Wires
Software	Raspbian Stretch OS
	Adafruit module for interfacing with the ADC chip
	Python Module **matplotlib** (used for data visualization)

⚡ **Circuit Connection (ADS1115 ADC to RPi GPIO Pins):**

The ADC chip communicates with the Raspberry Pi using the **I²C protocol**.

ADS1115 ADC Pin	Function	RPi Physical Pin (Board Mode)	RPi GPIO Function
VDD	Power	Pin 17	3.3v
GND	Ground	Pin 9	GND
SCL	I²C Clock	Pin 5	GPIO 3
SDA	I²C Data	Pin 3	GPIO 2

💻 **Code (Write this in exam notebook)**

This script uses the ads1015 library to read raw values and voltage from an analog channel (P0).

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT
import time
import board
import busio
import adafruit_ads1x15.ads1015 as ADS
from adafruit_ads1x15.analog_in import AnalogIn

# Create the I2C bus
i2c = busio.I2C(board.SCL, board.SDA)

# Create the ADC object using the I2C bus
# The ADS1015 is used in this simple test code
ads = ADS.ADS1015(i2c)

# Create single-ended input on channel 0
chan = AnalogIn(ads, ADS.P0)

# Create differential input between channel 0 and 1
# chan = AnalogIn(ads, ADS.P0, ADS.P1)

print("{:>5}\t{:>5}".format("raw", "v"))
while True:
 print("{:>5}\t{:>5.3f}".format(chan.value, chan.voltage)) # Prints
raw reading and calculated voltage [cite: 50]
 time.sleep(0.5)
``` [cite: 49]
```

▶️ **Procedure (Short Notes):**

1. Connect the **ADC module** to the Raspberry Pi's **I²C pins** (**SCL/SDA**) and **power pins** (**3.3V/GND**)[cite: 48].
2. Install the required Adafruit packages for the ADC and CircuitPython interface:
    * `$ sudo pip3 install adafruit-circuitpython-ads1x15` [cite: 48]
    * `$ sudo pip3 install adafruit-blinka` [cite: 48]
```

3. Save the Python script above (e.g., as `ads1x15_simpletest.py`).
4. Run the script from the terminal:
    * `$ python3 ads1x15_simpletest.py` [cite: 48]
5. The script will continuously print the **raw digital value** and the **calculated voltage** from Analog Channel 0 (P0)[cite: 49, 50].

This demonstrates the ADC's conversion functionality, which is the basis for the oscilloscope visualization[cite: 44].

Practical No. 6: Controlling Raspberry Pi with Telegram App

🎯 **Aim:** To control a device (e.g., an LED) connected to the Raspberry Pi remotely by sending commands via the **Telegram** messaging application.

🧰 **Requirements:**

Category	Item
Hardware	Raspberry Pi Model A/B/B+
	LED
	Breadboard, Jumper Wires
Software	Telegram App (in Mobile)
	Telegram Bot (on Raspberry Pi)

Python Script with `telepot` and `RPi.GPIO` libraries

⚡ **Circuit Connection:** The sample code controls a single LED connected to GPIO Pin 11 (Board mode).

Raspberry Pi Pin (Board Mode)	Connection
Pin 11 (GPIO 17)	LED anode (+) through a current-limiting resistor (e.g., 330Ω)
GND (Pin 6 or any GND)	LED cathode (–)

💻 **Code (Write this in exam notebook)**

This script controls an LED connected to **GPIO Pin 11. Note:** The token (5329...) must be replaced with your actual Telegram bot's **access token.**

```
Python
import time, datetime
import RPi.GPIO as GPIO
import telepot
import sys

def on(pin):
 GPIO.output(pin,GPIO.HIGH)
 return

def off(pin):
 GPIO.output(pin,GPIO.LOW)
 return

GPIO.setmode(GPIO.BOARD)
GPIO.setwarnings(False)
GPIO.setup(11,GPIO.OUT) # Pin 11 set as output

def handle(msg):
 chat_id=msg['chat']['id']
```

```
 command=msg['text']
 print('Got command:%s' % command)
 if command=='on':
  bot.sendMessage(chat_id,on (11)) # Send 'on' command to pin 11
 elif command=='off':
  bot.sendMessage(chat_id,off (11)) # Send 'off' command to pin 11

bot=telepot.Bot('5329577491:AAHhKMS0XTk-H2DMAVMwI69hoN9vWXVGKfk') # <-
-- REPLACE WITH YOUR BOT TOKEN
bot.message_loop(handle)
print("I am Listening....")

while 1:
 try:
  time.sleep(10)
 except keyboardInterrupt:
  print('/ program interrupt')
  GPIO.cleanup()
  exit()
 except:
  print('other error or exception occured')
  GPIO.cleanup()
```

▶️ **Procedure (Short Notes):**

1. Connect the LED to the Raspberry Pi's **GPIO Pin 11.**
2. Install the Telegram app on your mobile and follow the process to obtain an **access token** for your Telegram bot.
3. Install the required telepot library on the Raspberry Pi: $ sudo pip3 install telepot.
4. Write or modify the Python script (bot=telepot.Bot(...)) with your actual bot token and save it.
5. Run the Python script on the Raspberry Pi.
6. Send the commands 'on' or 'off' to your Telegram bot to control the LED.
7. The script will print **"I am Listening...."** and wait for commands.

Practical No. 10: Interfacing Raspberry Pi with RFID (by gemini)

🎯 **Aim:** To read information stored in an **RFID card or tag** using a connected RFID Reader, enabling the implementation of various authentication and control applications like **Access Control** or **Attendance Systems.**

🧰 **Requirements:**

Category	Item
Hardware	Raspberry Pi Model 3 B/B+
	RFID Reader (RC 522)
	RFID Tags or Cards
	Jumper wires (Female to Male), Breadboard
Software	Raspbian Stretch OS
	SPI Supporting Libraries
	RC522 Python Library

⚡ **Circuit Connection (RC 522 RFID Reader to RPi GPIO Pins):** The RC 522 communicates with the Raspberry Pi using the **SPI protocol.**

RFID Reader Pin	Function	RPi Physical Pin (Board Mode)	RPi GPIO Function

3.3V	Power	Pin 1	3.3V PWR
GND	Ground	Pin 6	GND
SCK	SPI Clock	Pin 23	GPIO11 (SPI0_CLK)
MOSI	Master Out Slave In	Pin 19	GPIO10 (SPI0_MOSI)
MISO	Master In Slave Out	Pin 21	GPIO9 (SPI0_MISO)
SDA (or CS)	Chip Select	Pin 24	GPIO8 (SPI_CE0_N)
RST	Reset	Pin 22	GPIO25 (GPIO_GEN6)
IRQ	Interrupt	UNUSED	UNUSED

💻 **Code (Write this in exam notebook)**

This practical requires two separate scripts: one to write data to the tag and one to read it back.

Script 1: Write Data (write.py)

Python

```
import RP1.GPIO as GPIO
from mfrc522 import SimpleMFRC522

reader = SimpleMFRC522()
#welcome message
print("Looking for cards")
print("Press Ctrl+c to STOP")
```

```
try:
 text=input('Enter New Data:')
 print("Now place your tag to write....")
 reader.write(text) # Writes data to the tag
 print("Data Written Successfully...")
finally:
 GPIO.cleanup()
``` [cite: 119]

Script 2: Read Data (`read.py`)

```python
import RPi.GPIO as GPIO
from mfrc522 import SimpleMFRC522

reader = SimpleMFRC522()
#welcome message
print ("Looking for cards")
print ("Press Ctrl+c to STOP")

try:
 id, text = reader.read() # Reads data from the tag
 print(id)
 print(text)
finally:
 GPIO.cleanup ()
``` [cite: 122]

***

▶️ **Procedure (Short Notes):**

1. Connect the **RC 522 RFID Reader** to the Raspberry Pi's **SPI GPIO Pins** according to the table above.
2. Enable the **SPI Interface** on the Raspberry Pi using the `sudo raspi-config` tool.
3. Install necessary packages, including the `spidev` and **MFRC522 libraries** using `pip`[cite: 111, 112].
4. Write and save the **`write.py`** script[cite: 119].

5. Run the script (e.g., `python3 write.py`). The terminal will ask you to enter data[cite: 113].
6. Place an RFID Tag on the reader to write the data[cite: 115, 116]. A successful write will display **"Data Written Successfully..."**[cite: 117].
7. Write and save the **`read.py`** script[cite: 122].
8. Run the script (e.g., `python3 read.py`) and place the same RFID Tag on the reader[cite: 121].
9. The script should immediately print the **Tag ID** and the **written data** (text)[cite: 122, 120].

You want to adapt **Practical 6** (which used Telegram) to use **WhatsApp** for controlling the Raspberry Pi.

Direct control of a Raspberry Pi via a standard, personal WhatsApp account is **not straightforward** due to WhatsApp's strict API policies, which are designed for **businesses** and not for personal IoT projects.

The official way to send and receive messages programmatically requires using the **WhatsApp Business Platform API**, which involves hosting a secure cloud environment and getting verified by Meta.

However, for a hobby project like controlling an LED, you can use a simpler, **indirect method** using a third-party service or webhooks.

Here is the concept for how you would implement it indirectly, similar to the Telegram practical:

Practical No. 2: Display Three LED Blinking Pattern using Raspberry Pi

🎯 **Aim:** To interface **three LEDs** with the Raspberry Pi and make them blink in a sequence or pattern.

🧰 **Requirements:**

Category	Item

Hardware	Raspberry Pi board
	3 × LED (e.g., Red, Yellow, Green)
	3 × 330Ω resistors
	Breadboard, Jumper wires
Software	Raspberry Pi OS, Python 3 with `RPi.GPIO` library

⚡ **Circuit Connection:** You will connect each LED anode (positive) through a 330Ω resistor to a different GPIO pin, and connect all LED cathodes (negative) to a common GND pin.

Raspberry Pi Pin (Board Mode)	Connection
Pin 11 (GPIO 17)	**Red LED** Anode (+) via Resistor
Pin 13 (GPIO 27)	**Yellow LED** Anode (+) via Resistor
Pin 15 (GPIO 22)	**Green LED** Anode (+) via Resistor
GND (Pin 6 or any GND)	All LED Cathodes (–)

💻 **Code for a Sequential Pattern (Traffic Light Style):**

This code will blink the three LEDs in a sequence: **Red** → **Yellow** → **Green** and then back.

```
Python
import RPi.GPIO as GPIO
import time

# --- Setup ---
GPIO.setmode(GPIO.BOARD)

# Define the pins for the three LEDs
RED_LED = 11
YELLOW_LED = 13
GREEN_LED = 15

# Set all three pins as output
GPIO.setup(RED_LED, GPIO.OUT)
GPIO.setup(YELLOW_LED, GPIO.OUT)
GPIO.setup(GREEN_LED, GPIO.OUT)
```

```
# --- Pattern Function ---
def blink_pattern():
    # 1. RED ON, others OFF
    GPIO.output(RED_LED, True)
    GPIO.output(YELLOW_LED, False)
    GPIO.output(GREEN_LED, False)
    print("RED ON")
    time.sleep(2) # Red stays on for 2 seconds

    # 2. RED OFF, YELLOW ON
    GPIO.output(RED_LED, False)
    GPIO.output(YELLOW_LED, True)
    GPIO.output(GREEN_LED, False)
    print("YELLOW ON")
    time.sleep(1) # Yellow stays on for 1 second

    # 3. YELLOW OFF, GREEN ON
    GPIO.output(RED_LED, False)
    GPIO.output(YELLOW_LED, False)
    GPIO.output(GREEN_LED, True)
    print("GREEN ON")
    time.sleep(2) # Green stays on for 2 seconds

    # 4. GREEN OFF, all OFF for a moment
    GPIO.output(GREEN_LED, False)
    time.sleep(0.5)

# --- Main Loop ---
print("Starting sequential pattern...")
try:
    for i in range(5):  # Run the sequence 5 times
        blink_pattern()
except KeyboardInterrupt:
    print("\nPattern stopped by user.")
finally:
    # Always clean up pins when done
    GPIO.cleanup()
```

```
    print("GPIO cleanup complete.")
```

Code for a Simple Alternating Pattern

To show a different pattern, here is a script to make the lights alternate, switching between two adjacent LEDs at once:

Python

```
# (Use the same setup block as above: import, GPIO.setmode, GPIO.setup)

# ... Setup Block ...
# Assuming the pins are already set up: RED_LED=11, YELLOW_LED=13, GREEN_LED=15

# --- Alternating Pattern Function ---
def alternate_pattern():
    # 1. RED and GREEN ON (outer LEDs)
    GPIO.output(RED_LED, True)
    GPIO.output(YELLOW_LED, False)
    GPIO.output(GREEN_LED, True)
    time.sleep(0.5)

    # 2. YELLOW ON (middle LED), others OFF
    GPIO.output(RED_LED, False)
    GPIO.output(YELLOW_LED, True)
    GPIO.output(GREEN_LED, False)
    time.sleep(0.5)

# --- Main Loop ---
print("Starting alternating pattern...")
try:
    while True:
        alternate_pattern()
except KeyboardInterrupt:
    GPIO.cleanup()
    print("Cleanup complete.")
```

THE RASPBERRYPI WITH DIFFERENT PATTERN LED

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BOARD)

# LED pins
led1 = 29
led2 = 31
led3 = 33
led4 = 35
led5 = 37

# Set pins as output
GPIO.setup(led1, GPIO.OUT)
GPIO.setup(led2, GPIO.OUT)
GPIO.setup(led3, GPIO.OUT)
GPIO.setup(led4, GPIO.OUT)
GPIO.setup(led5, GPIO.OUT)

try:
  while True:
    # Turn LEDs ON (False if active-low)
    GPIO.output(led1, False)
    print("LED1 ON")
    time.sleep(1)

    GPIO.output(led2, False)
    print("LED2 ON")
    time.sleep(1.5)

    GPIO.output(led3, False)
    print("LED3 ON")
```

```
        time.sleep(2)

        GPIO.output(led4, False)
        print("LED4 ON")
        time.sleep(2.5)

        GPIO.output(led5, False)
        print("LED5 ON")
        time.sleep(3)

        # Turn LEDs OFF
        GPIO.output(led1, True)
        print("LED1 OFF")
        time.sleep(3.5)

        GPIO.output(led2, True)
        print("LED2 OFF")
        time.sleep(4)

        GPIO.output(led3, True)
        print("LED3 OFF")
        time.sleep(4.5)

        GPIO.output(led4, True)
        print("LED4 OFF")
        time.sleep(5)

        GPIO.output(led5, True)
        print("LED5 OFF")
        time.sleep(5.5)

except KeyboardInterrupt:
    GPIO.cleanup()
    print("Program stopped and GPIO cleaned up")
```

Practical 9: Interface Raspberry Pi with Pi Camera

Aim:

To interface Raspberry Pi with the Pi Camera Module and capture images and videos.

Components Required:

Component	Quantity
Raspberry Pi (any model with camera interface)	1
Pi Camera Module (V2 or HQ)	1
Micro SD Card	1
Power Supply	1
Monitor, Keyboard, Mouse	1 each
Internet Connection	Optional (for remote access)

Software Requirements:

1. **Raspberry Pi OS** (with Desktop recommended)
2. **Python 3**
3. Python Libraries:

```
pip3 install picamera
```

   a. For PiCamera v1 or v2: picamera
   b. For Raspberry Pi OS Bullseye or later, picamera2 may be used:

```
sudo apt install -y python3-picamera2
```

Circuit/Connection Diagram:

1. Connect the **Pi Camera ribbon cable** to the **CSI port** of Raspberry Pi.
2. Ensure the **metal contacts face the HDMI port** (depends on Pi model).
3. Secure the cable properly in the CSI connector.

Enable Camera:

1. Open terminal and run:

```
sudo raspi-config
```

2. Navigate to **Interface Options → Camera → Enable.**
3. Reboot Raspberry Pi.

Python Code Example: Capture Image

```
from picamera import PiCamera
from time import sleep

camera = PiCamera()

# Set resolution (optional)
camera.resolution = (1024, 768)

# Start preview (optional)
camera.start_preview()
sleep(2)  # Wait for camera to adjust

# Capture image
camera.capture('/home/pi/Desktop/image1.jpg')
print("Image Captured")
```

```
# Stop preview
camera.stop_preview()
```

Python Code Example: Record Video

```
from picamera import PiCamera
from time import sleep

camera = PiCamera()

camera.resolution = (1280, 720)
camera.framerate = 30

camera.start_preview()
sleep(2)

# Record video for 10 seconds
camera.start_recording('/home/pi/Desktop/video1.h264')
sleep(10)
camera.stop_recording()
print("Video Recorded")

camera.stop_preview()
```

Note: Video is saved in .h264 format. Convert to .mp4 if needed:

```
MP4Box -add video1.h264 video1.mp4
```

Procedure:

1. Connect the Pi Camera to CSI port properly.
2. Enable camera interface using raspi-config.
3. Install necessary Python libraries.
4. Run Python code to capture images or record videos.
5. Check saved files on Desktop (or specified path).

Practical 10: Interfacing Raspberry Pi with RFID by (gpt)

Aim:

To interface an RFID module with Raspberry Pi and read RFID tags/cards for identification or authentication purposes.

Components Required:

Component	Quantity
Raspberry Pi (any model)	1
RFID Module (RC522)	1
RFID Cards / Tags	2-3
Jumper Wires	As required
Breadboard	Optional

Software Requirements:

1. Raspberry Pi OS
2. Python 3
3. Python Libraries:

```
pip3 install mfrc522
pip3 install RPi.GPIO
```

Circuit Connection (RC522 RFID Module to Raspberry Pi GPIO)

RC522 Pin	Raspberry Pi Pin	Notes
SDA	GPIO 8 (CE0)	SPI Chip Select
SCK	GPIO 11 (SCLK)	SPI Clock
MOSI	GPIO 10 (MOSI)	SPI Master Out

MISO	GPIO 9 (MISO)	SPI Master In
IRQ	Not connected	Optional
GND	GND	Ground
RST	GPIO 25	Reset
3.3V	3.3V	Power

Make sure **SPI interface is enabled**:

```
sudo raspi-config
# Interface Options → SPI → Enable
```

Python Code Example: Read RFID Tag

```
import RPi.GPIO as GPIO
from mfrc522 import SimpleMFRC522

reader = SimpleMFRC522()

try:
    print("Place your RFID tag/card near the reader")
    id, text = reader.read()
    print(f"RFID ID: {id}")
    print(f"Text: {text}")

finally:
    GPIO.cleanup()
```

Python Code Example: Write Data to RFID Tag

```
import RPi.GPIO as GPIO
from mfrc522 import SimpleMFRC522

reader = SimpleMFRC522()

try:
    text_to_write = input("Enter data to write on RFID: ")
```

```
    print("Place the tag near the reader")
    reader.write(text_to_write)
    print("Data written successfully!")

finally:
    GPIO.cleanup()
```

Procedure:

1. Connect RC522 module to Raspberry Pi using SPI pins as per table.
2. Enable **SPI interface** via raspi-config.
3. Install Python libraries mfrc522 and RPi.GPIO.
4. Run **read.py** script and place an RFID card/tag near the reader.
5. Note the **RFID ID** and optional stored text.
6. Run **write.py** to store custom data on a new RFID card/tag.
7. Verify by running the **read script** again.

Precautions:

- Always use **3.3V power**, not 5V (RC522 is 3.3V).
- Keep hands and metal objects away from the reader during scanning.
- Clean GPIO pins properly using GPIO.cleanup().

Practical 7: Fingerprint Sensor Interfacing with Raspberry Pi

Aim:

To interface a fingerprint sensor with Raspberry Pi and perform fingerprint enrollment and authentication.

Components Required:

Component	Quantity
Raspberry Pi (any model)	1
Fingerprint Sensor (e.g., R305, GT-521F32)	1
Jumper Wires	As required
Breadboard	Optional
Power Supply	1

Software Requirements:

1. Raspberry Pi OS
2. Python 3
3. Python Library:

```
pip3 install pyfingerprint
```

Circuit Connection (UART Version, e.g., R305)

Fingerprint Pin	Raspberry Pi Pin	Notes
VCC	5V	Power
GND	GND	Ground
TX	RX (GPIO 15)	Serial Receive
RX	TX (GPIO 14)	Serial Transmit

For USB fingerprint sensors, simply connect via USB port.

Python Code Example: Enroll Fingerprint

```
from pyfingerprint.pyfingerprint import PyFingerprint

try:
    f = PyFingerprint('/dev/serial0', 57600, 0xFFFFFFFF, 0x00000000)
    if not f.verifyPassword():
```

```
        raise ValueError('Fingerprint sensor password incorrect')
except Exception as e:
    print('Sensor initialization failed:', e)
    exit(1)

print('Fingerprint sensor ready.')

try:
    print('Place your finger on the sensor...')
    while not f.readImage():
        pass

    f.convertImage(0x01)
    result = f.searchTemplate()
    positionNumber = result[0]

    if positionNumber >= 0:
        print(f'Fingerprint already exists at position
{positionNumber}')
    else:
        print('Fingerprint not found. Enrolling new fingerprint...')
        f.createTemplate()
        positionNumber = f.storeTemplate()
        print(f'Fingerprint enrolled at position {positionNumber}')

except Exception as e:
    print('Operation failed:', e)
```

Python Code Example: Verify Fingerprint

```
from pyfingerprint.pyfingerprint import PyFingerprint

try:
    f = PyFingerprint('/dev/serial0', 57600, 0xFFFFFFFF, 0x00000000)
    if not f.verifyPassword():
        raise ValueError('Fingerprint sensor password incorrect')
except Exception as e:
```

```
    print('Sensor initialization failed:', e)
    exit(1)

print('Place your finger for verification...')
try:
    while not f.readImage():
        pass

    f.convertImage(0x01)
    result = f.searchTemplate()
    positionNumber = result[0]

    if positionNumber >= 0:
        print(f'Fingerprint matched! Position: {positionNumber}')
    else:
        print('Fingerprint not recognized.')

except Exception as e:
    print('Verification failed:', e)
```

Procedure:

1. Connect fingerprint sensor to Raspberry Pi (UART or USB).
2. Enable serial interface if using UART:

```
sudo raspi-config
# Interface Options → Serial → Enable
```

3. Install pyfingerprint library.
4. Run the **enroll script** to add fingerprints.
5. Run the **verify script** to test authentication.
6. Optionally, store multiple fingerprints for testing.

Precautions:

- Keep fingers clean for accurate scanning.
- Avoid excessive pressure on the sensor.
- Ensure correct UART wiring: TX → RX, RX → TX.
- Use GPIO.cleanup() in scripts if GPIO pins are used for LEDs/buzzers.

Practical 8: Interfacing GPS Module with Raspberry Pi

Aim:

To interface a GPS module with Raspberry Pi and read real-time location (latitude, longitude) and time data.

Components Required:

Component	Quantity
Raspberry Pi (any model)	1
GPS Module (e.g., NEO-6M)	1
Jumper Wires	As required
Breadboard	Optional
Power Supply	1

Software Requirements:

1. Raspberry Pi OS
2. Python 3
3. Python Libraries:

```
pip3 install gps
pip3 install pynmea2
```

Circuit Connection (GPS Module to Raspberry Pi UART)

GPS Pin	Raspberry Pi Pin	Notes
VCC	3.3V / 5V	Power
GND	GND	Ground
TX	RX (GPIO 15)	Serial Receive
RX	TX (GPIO 14)	Serial Transmit

Make sure to **enable UART** on Raspberry Pi:

```
sudo raspi-config
# Interface Options → Serial → Enable
```

Python Code Example: Reading GPS Data

```
import serial
import pynmea2

# Configure serial port for GPS
port = serial.Serial("/dev/serial0", baudrate=9600, timeout=1)

try:
    while True:
        data = port.readline().decode('ascii', errors='replace')
        if data.startswith('$GPGGA'):
            msg = pynmea2.parse(data)
            print(f"Latitude: {msg.latitude} {msg.lat_dir}")
            print(f"Longitude: {msg.longitude} {msg.lon_dir}")
            print(f"Altitude: {msg.altitude} {msg.altitude_units}")
            print(f"Timestamp: {msg.timestamp}")
            print("------------------------------")

except KeyboardInterrupt:
    port.close()
    print("Program stopped")
```

Explanation:

- The GPS module continuously sends **NMEA sentences** over serial.
- $GPGGA sentence contains **latitude, longitude, altitude, and timestamp**.
- pynmea2 library parses these sentences for easy reading.

Procedure:

1. Connect GPS module to Raspberry Pi UART pins.
2. Enable UART using raspi-config.
3. Install required Python libraries (gps, pynmea2).
4. Run the Python script.
5. Wait a few seconds for GPS module to get a **fix**.
6. Observe real-time **latitude, longitude, altitude, and time** in terminal.

Precautions:

- GPS modules need **clear sky view** for better accuracy.
- Use proper voltage (3.3V or 5V as per module).
- Avoid connecting multiple devices to UART while reading GPS.