Chapter 7：樹莓派 5 X IoT

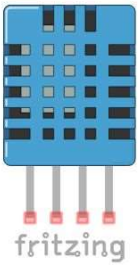
本章重點

介紹樹莓派的物聯網應用。

準備材料

可上網的電腦

樹莓派（含相機模組與外殼）



電子零件

學習目標

1. 監控樹莓派的硬體。
2. 樹莓派的 GPIO 輸出控制。
3. 數位輸出與數位輸入。
4. AIoT 專案 1：LLM 智慧燈號控制系統。
5. 物聯網實務三：使用 Node-RED。

7-1. 樹莓派 X 物聯網

物聯網的英文全名是：Internet of Things (IoT)，簡單的說，就是萬物連網，所有東西（物體）都可以上網，因為所有東西都連上網路，我們可以透過任何連網裝置來遠端控制這些連網的東西、就算遠在天涯海角也一樣可以進行監控，如下圖所示：

對於物聯網來說，每一個人都可以將真實東西連接上網，我們可以輕易在物聯網查詢這個東西的位置，並且對這些東西進行集中管理與控制。例如：

- 遙控家電用品、汽車遙控、行車路線追蹤和防盜監控。
- 建立更聰明的智慧家電、更安全的自動駕駛，和住家環境。

不僅如此，透過物聯網上大量裝置和感測器取得的資料，我們可以建立大數據 (Big Data) 來進行分析，並且從取得的數據分析結果來重新設計流程，改善我們的生活。例如：

- 減少車禍、災害預測、犯罪防治與流行病控制。

7-1-1. 監控樹莓派的硬體

基本系統指令

檢測 CPU 使用率：

```
~$ top -bn1
```

RAM 使用量：

```
~$ free -m
```

磁碟使用量：

```
~$ df -h /
```

CPU 溫度（Raspberry Pi 專用）：

```
~$ vcgencmd measure_temp
```

網路介面與流量：

```
~$ ip -o link show
```

讀取 RX (接收) / TX (傳送)：

```
~$ cat /proc/net/dev
```

使用 Streamlit 建立監控網頁

以下介紹 Streamlit 來建立樹莓派硬體的監控網頁：

建立虛擬環境時，加上 --system-site-packages 參數就會包含系統全域範圍安裝的 Python 套件，例如：建立一個 iot_311 的虛擬環境。

```
~$ python -m venv --system-site-packages ~/iot_311
```

啟用虛擬環境 iot_311：

```
~$ source ~/iot_311/bin/activate
```

安裝 Streamlit：

```
(iot_311) ~$ pip install streamlit
```

下載講師提供的範例程式 ch07.zip：

```
(iot_311) ~$ wget http://max543.com/debugger/class/python02/%E5%BD
%B1%E5%83%8F%E8%BE%A8%E8%AD%98%20X%20%E6%A8%B9%E8%8E%93%E6%B4%BE%205_202
5/example/ch07.zip
```

解壓縮：

```
(iot_311) ~$ unzip ch07.zip
```

進入資料夾 ch07：

```
(iot_311) ~$ cd ch07
```

先執行 monitor.py，觀看結果：

```
(iot_311) ~$ cpython run_monitor.py
```

monitor.py 程式碼如下：

monitor　樹莓派監控網頁。

```
1.  import streamlit as st
2.  import subprocess
```

```
import time

# ---------- 頁面設定 ----------
st.set_page_config(page_title="RPi 5 Monitor", layout="wide")  # 頁面標題與布局

# 自訂 CSS 樣式
st.markdown(
    """
<style>
.block-container { padding-top: 2rem; }  /* 頁面上方間距 */
div[data-testid="metric-container"] {
    background-color: #0e1117;
    border: 1px solid #1f2937;
    padding: 1rem;
    border-radius: 12px;
    color: white;
}
.temp-warn {
    background-color: #7f1d1d !important;
    border-color: #ef4444 !important;
}
</style>
""",
    unsafe_allow_html=True,
)

st.title("🍓 Raspberry Pi 5 • 系統監控")  # 標題

# ---------- 輔助函數 ----------
def run(cmd):
    """執行系統指令並返回結果（去除換行）"""
    return subprocess.check_output(cmd, shell=True,
                                   text=True).strip()

# ---------- CPU 使用率 ----------
def cpu_usage():
    """取得 CPU 使用率"""
    idle = run(
```

```
        "top -bn1 | grep 'Cpu(s)' | "
        "sed 's/,/ /g' | "
        "awk '{for(i=1;i<=NF;i++) if ($i==\"id\") print $(i-1)}'"
    )
    return 100 - float(idle)

# ---------- RAM 使用率 ----------
def ram_usage():
    """取得 RAM 已用量、總量、使用率"""
    used, total, percent = run(
        "free -m | awk 'NR==2{print $3,$2,$3/$2*100}'"
    ).split()
    return int(used), int(total), float(percent)

# ---------- Swap 使用率 ----------
def swap_usage():
    """取得 Swap 已用量、總量、使用率"""
    used, total, percent = run(
        "free -m | awk 'NR==3{print $3,$2,($3/$2)*100}'"
    ).split()
    return int(used), int(total), float(percent)

# ---------- 磁碟使用率 ----------
def disk_usage():
    """取得根目錄磁碟使用量"""
    used, total, percent = run(
        "df -h / | awk 'NR==2{print $3,$2,$5}'"
    ).split()
    return used, total, percent

# ---------- CPU 溫度 ----------
def cpu_temp():
    """取得 CPU 溫度"""
    temp = run("vcgencmd measure_temp | cut -d= -f2 | tr -d \"'C\"")
    return float(temp)

# ---------- 網路使用量 ----------
def net_usage():
    """取得主要網卡名稱與上下行流量 (MB)"""
```

```
    iface = run(
        "ip -o link show | awk -F': ' '{print $2}' | "
        "grep -E '^(eth|wlan|usb)' | head -n1"
    )
    rx, tx = run(
        f"cat /proc/net/dev | grep {iface} | "
        "awk '{print $2,$10}'"
    ).split()
    return iface, int(tx) // 1024**2, int(rx) // 1024**2

# ---------- 使用者介面 ----------
c1, c2, c3, c4, c5, c6 = st.columns(6)

# CPU
c1.metric("🧠 CPU", f"{cpu_usage():.0f} %")

# RAM
ram_used, ram_total, ram_pct = ram_usage()
c2.metric("💾 RAM", f"{ram_pct:.0f} %", f"{ram_used} / {ram_total} MB")

# Swap
swap_used, swap_total, swap_pct = swap_usage()
c3.metric("🌀 Swap", f"{swap_pct:.0f} %", f"{swap_used} / {swap_total} MB")

# 磁碟
disk_used, disk_total, disk_pct = disk_usage()
c4.metric("💿 Disk", disk_pct, f"{disk_used} / {disk_total}")

# CPU 溫度
temp = cpu_temp()
if temp >= 80:  # 超過 80°C 顯示警告
    html_temp_warn = f"""<div class="temp-warn" style="padding:1rem;border-radius:12px">
🌡️ <b>Temp</b><br>
<span style="font-size:2rem">{temp:.1f} °C</span><br>
⚠️ OVERHEAT
</div>"""
    c5.markdown(html_temp_warn, unsafe_allow_html=True)
else:
```

```
    c5.metric("🌡 Temp", f"{temp:.1f} °C")

# 網路流量
iface, tx, rx = net_usage()
c6.metric("🌐 Net", f"{iface} ↑ {tx} MB", f"↓ {rx} MB")

# ---------- 自動刷新 ----------
time.sleep(2)
st.rerun()
```

Streamlit 監控程式裡，使用了多個 Linux 系統指令來取得 CPU、記憶體、磁碟、溫度和網路資訊。下面整理出來，每個函式用到的指令：

1. cpu_usage()：

```
top -bn1 | grep 'Cpu(s)' | sed 's/,/ /g' | awk '{for(i=1;i<=NF;i++) if ($i=="id") print $(i-1)}'
```

- top -bn1：非互動模式執行一次 CPU/記憶體狀態檢查。
- grep 'Cpu(s)'：過濾出 CPU 使用行。
- sed 's/,/ /g'：將逗號替換成空格。
- awk ...：取得 CPU idle 百分比，然後用 100 - idle 得到使用率。

2. ram_usage()：

```
free -m | awk 'NR==2{print $3,$2,$3/$2*100}'
```

- free -m：以 MB 顯示記憶體資訊。
- awk 'NR==2{...}'：取得第二行 (RAM)，分別輸出已用、總量、百分比。

3. swap_usage()：

```
free -m | awk 'NR==3{print $3,$2,($3/$2)*100}'
```

- 與 RAM 類似，但取第三行 (Swap)。

4. disk_usage()：

```
df -h / | awk 'NR==2{print $3,$2,$5}'
```

- df -h /：檢查根目錄磁碟使用量。
- awk 'NR==2{...}'：取得已用、總量、百分比。
-

5. cpu_temp()：

```
vcgencmd measure_temp | cut -d= -f2 | tr -d "'C"
```

- vcgencmd measure_temp：取得 Raspberry Pi CPU 溫度。
- cut -d= -f2：取得等號右邊數值。
- tr -d "'C"：去掉文字和單位。

6. net_usage()：

```
ip -o link show | awk -F': ' '{print $2}' | grep -E '^(eth|wlan|usb)' |
head -n1
cat /proc/net/dev | grep {iface} | awk '{print $2,$10}'
```

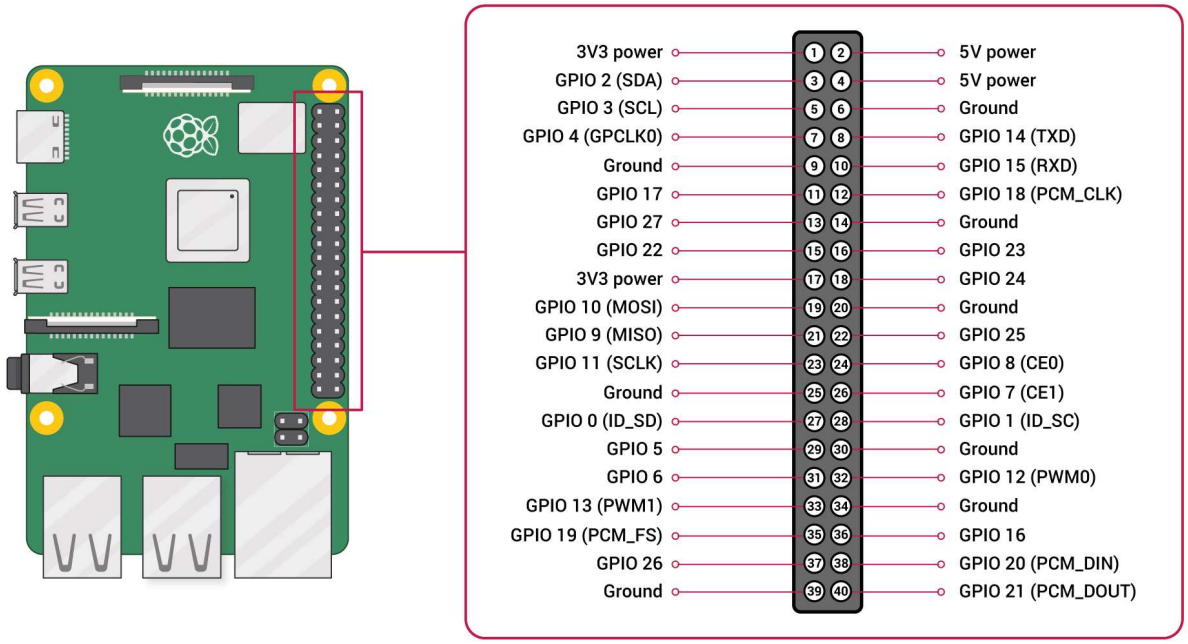
- ip -o link show：列出網路介面。
- awk -F': ' '{print $2}'：取介面名稱。
- grep -E '^(eth|wlan|usb)'：篩選有線、無線或 USB 網卡。
- head -n1：只取第一個符合的網卡。
- cat /proc/net/dev | grep {iface} | awk '{print $2,$10}'：取得該介面的接收 (rx) 和傳送 (tx) bytes。

7-2. 樹莓派的 GPIO 控制

7-2-1. 認識樹莓派的 GPIO

GPIO (general-purpose input/output) 通用輸入/輸出接腳，因為樹莓派提供了這些可程式控制的腳位，可以用來連接外部電子電路或感測器模組。

樹莓派有 40 個腳位可外接感測器或控制電路。注意單晶片的實際腳位與 PCB 板上的編號不同，寫程式時是控制實際腳位的，例如：當使用『 12 』，實際上是『 GPIO 18 』。



腳位特性

這些接腳的相關特性如下：

- 電源腳位有 5V (2 支)、3.3V (2 支)及 GND (8 支)。
- 當輸出腳位時，其電壓分為高電位 3.3V、低電位 0V。
- 當輸入腳位時，其電壓 < 0.8V 時判斷為低電位、 > 1.3V 時判斷為高電位。
- 除了 GPIO 2、GPIO 3 是固定內接上拉電阻，其他腳位可透過程式設定內接上拉或下拉電阻。
- 單一腳位輸出電流 3mA，電流輸出總和不超過 50mA。

以上為電路特性，比較需要注意的點如下：

1. 輸入腳位可忍受的電位為 3.3V，若會超過就需要用電源轉換模組或是採用電阻分壓方式，避免將樹莓派弄壞。
2. 當程式規劃該腳位為輸出時，不可連接輸入特性的裝置接腳，避免造成樹莓派損毀。例如：樹莓派輸出腳位，接上按鈕開關的訊號腳位，可能會導致短路或過電流，進而損壞 GPIO 或整個板子。

特定腳位功能

除了設定腳位為輸入或輸出模式外，單一接腳附加功能或多接腳組合技如下列：

- PWM (Pulse-Width Modulation)
  - 軟體模擬 PWM (Software)：適用所有控制腳位。缺點是無法確保輸出的脈寬穩定性。
  - 硬體 PWM (Hardware)：僅有兩組可用，第一組為 GPIO 12、GPIO 18 (此兩接腳輸出一致)，第二組為 GPIO 13、GPIO 19。
- UART (Universal Asynchronous Receiver-Transmitter) 通用非同步收發傳輸器
  - 使用接腳：TX (GPIO 14)、RX (GPIO 15)。
  - 於其他裝置連接時，將樹莓派的 TX，連結接其他裝置的 RX；樹莓派的 RX，連接其他裝置的 TX，便可以進行 UART 序列埠資料傳輸。
- I2C (Inter-Integrated Circuit)
  - 使用接腳：資料線 SDA (GPIO 2)、時脈線 SCL (GPIO 3)。

    - 另一組為 EEPROM 使用接腳。
- SPI (Serial Peripheral Interface)
    - SPI0 使用接腳：MOSI (GPIO 10)、MISO (GPIO 9)、SCLK (GPIO 11)、CE0 (GPIO 8)、CE1 (GPIO 7)。
    - SPI1 使用接腳：MOSI (GPIO 20)、MISO (GPIO 19)、SCLK (GPIO 21)、CE0 (GPIO 18)、CE1 (GPIO 17)、CE2 (GPIO 16)。

7-2-2. 使用 Python 的 GPIO Zero 模組

樹莓派的 GPIO 接腳的控制語言可以使用 Python、Java 和 C 語言等，在本講義是使用 Python 的 GPIO Zero 模組來控制 GPIO 接腳。

方法一：使用 --system-site-packages 建立虛擬環境

建立虛擬環境時，加上 --system-site-packages 參數就會包含系統全域範圍安裝的 Python 套件，例如：建立一個 iot_311 的虛擬環境。

```
~$ python -m venv --system-site-packages ~/iot_311
```

方法二：在虛擬環境中自行安裝 gpiozero 和 lgpio

如果虛擬環境沒有可使用 --system-site-packages 參數，可自行 pip install 來安裝。

安裝 gpiozero：

```
(虛擬環境名稱) ~$ pip install gpiozero
```

安裝 lgpio：

```
(虛擬環境名稱) ~$ pip install lgpio
```

7-2-2. 數位輸出與數位輸入

數位輸出 (Digital Output, DO) 是輸出數位訊號**高電位**或**低電位** 2 種狀態至連接 GPIO 接腳的感測器或電子元件，例如：控制 LED 燈的點亮或熄滅。數位輸入 (Digital Input, DI) 是指從連接的感測器或電子元件偵測到外界電壓訊號的改變後，可以轉變成對應的數位訊號的 2 種狀態，例如：按鈕開關，按下是 1，放開是 0 等。

數位輸出：閃爍 LED 燈

Python 程式可以使用 GPIO Zero 模組來控制 GPIP 接腳的數位輸出。

所需的電子材料

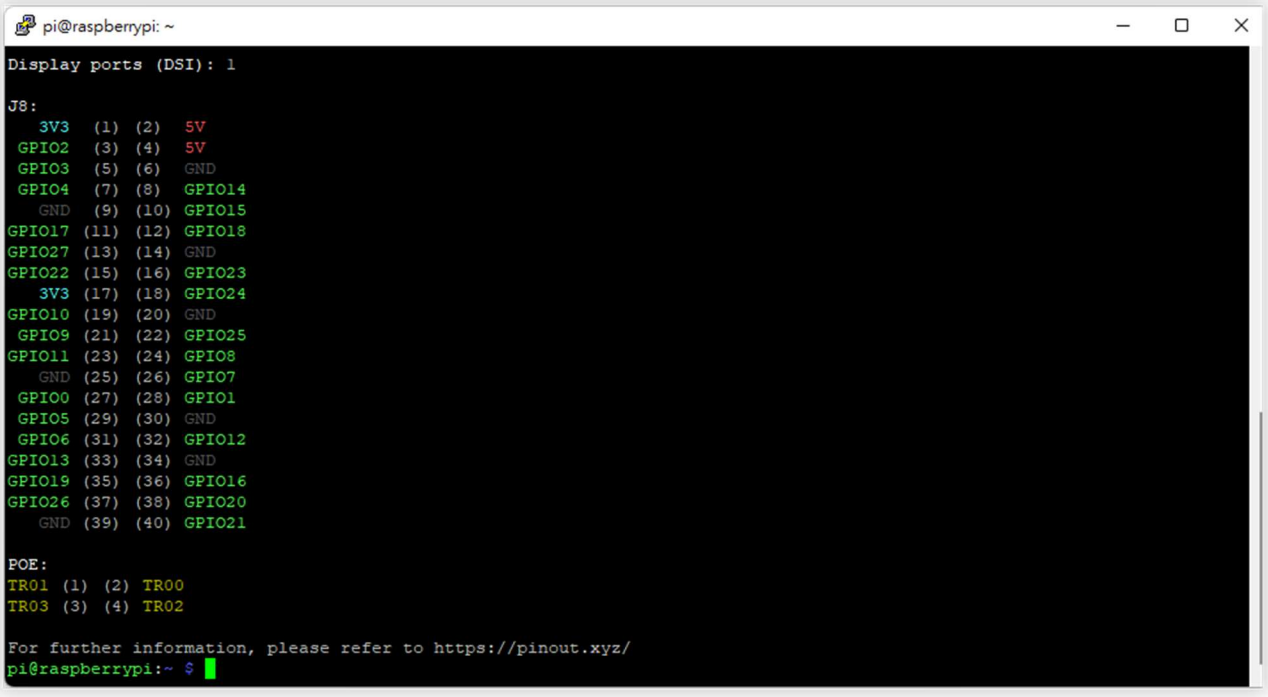
- 紅色 LED 燈 x 1
- 220Ω 電阻 x 1
- 麵包板 x 1
- 公-母杜邦線 x 3

實驗步驟

Step 1 確認樹莓派的 GPIO 腳位。

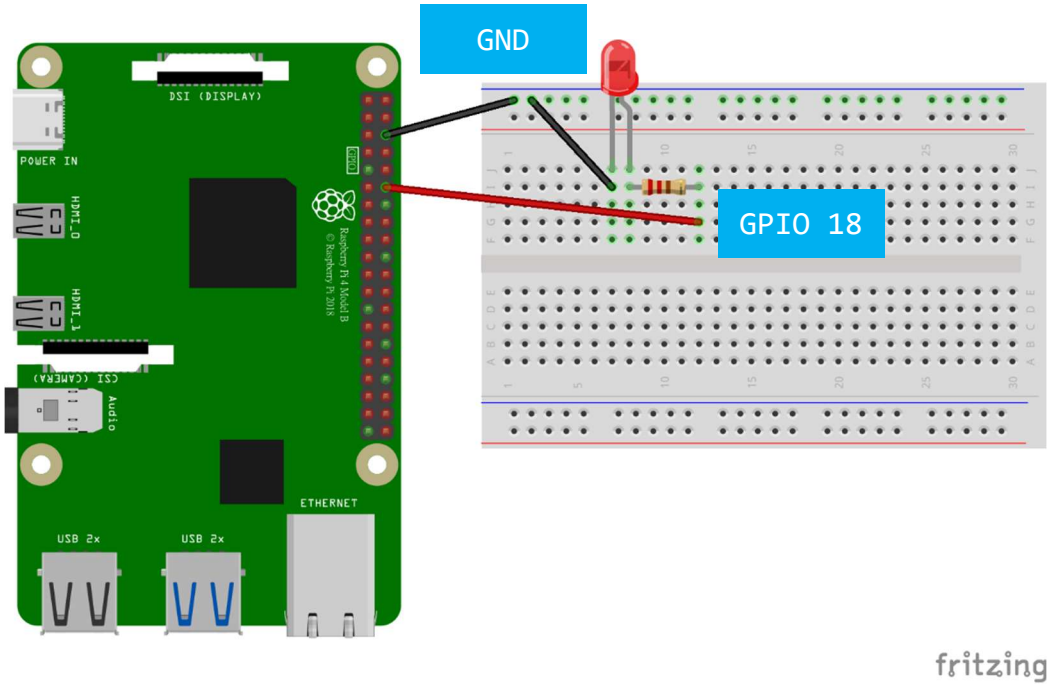
GPIO 可利用以下指令，在終端機中查詢腳位編號：

```
(iot_311) ~$ pinout
```



Step 2 一個 LED 的接線。

依據下圖連接建立電子電路後，紅色 LED 燈的長腳（正）連接一個 220Ω 電阻（電阻沒有方向），電阻用來降低電流以保護 LED，電阻另一頭連接 GPIO 18，LED 的短腳（負）接地。如下圖所示：



Step 3 撰寫程式。

7-1 不斷閃爍的 LED 燈。

```
1.  from gpiozero import LED
2.  from time import sleep
3.
4.  led = LED(18)
5.
6.  while True:
7.      led.on()
8.      sleep(1)
9.      led.off()
10.     sleep(1)
```

Web 介面的 GPIO 輸出控制

物聯網最基本的功能就是透過 Web 介面進行遠端控制。以樹莓派來說，主要就是使用 Web 介面來進行 GPIO 控制。我們準備使用 Streamlit 模組建立 Web 伺服器後，在網頁提供按鈕，來遠端點亮或熄滅一顆紅色，和一顆綠色的 LED 燈。

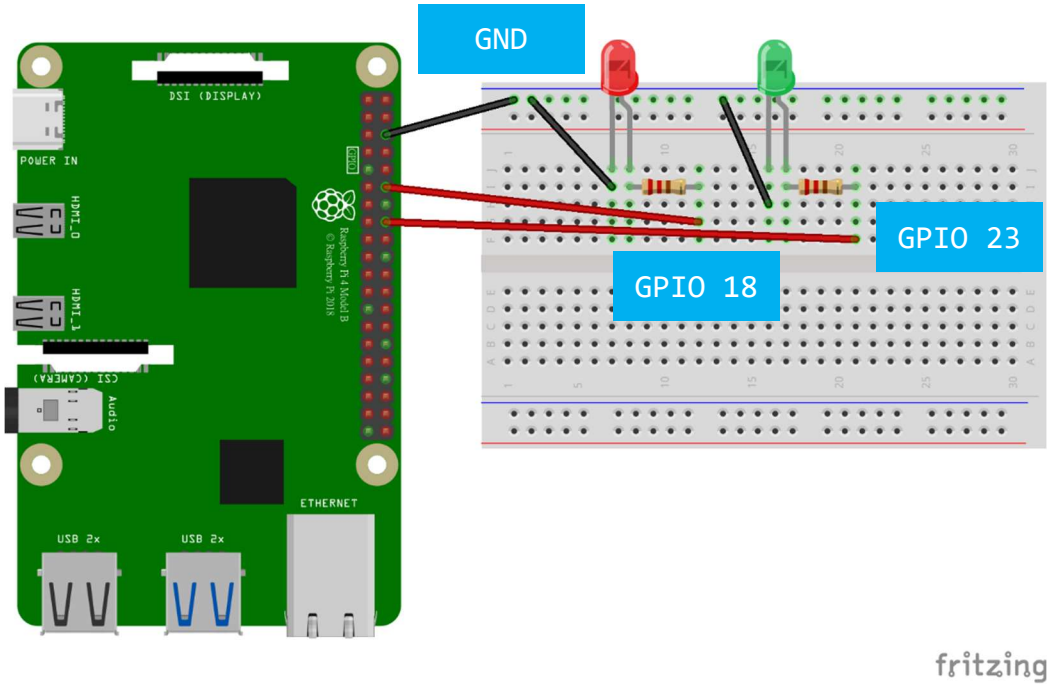
所需的電子材料

- 紅色 LED 燈 x 1
- 綠色 LED 燈 x 1
- 220Ω 電阻 x 2
- 麵包板 x 1
- 公-母杜邦線 x 5

實驗步驟

Step 1 2 顆 LED 的接線。

請依據下圖連接電路後，紅色 LED 燈的長腳（正）連接 GPIO 18，綠色 LED 燈的長腳（正）連接 GPIO 23，就完成本節實驗的電子電路設計。如下圖所示：



Step 2 撰寫程式。

7-2 利用 Streamlit 建立 Web 介面控制 GPIO 的輸出。

```
1.  import streamlit as st
2.  from gpiozero import LED
3.
4.  # ============================================================
5.  # GPIO 初始化 (只會執行一次)
6.  # ============================================================
```

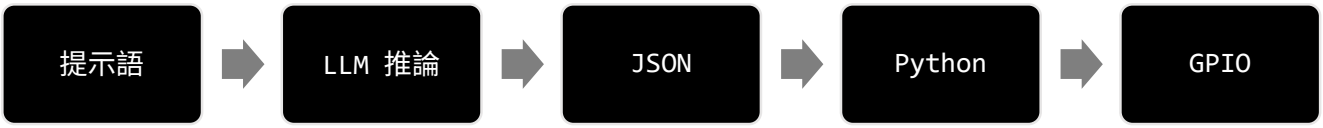
```
@st.cache_resource
def init_leds():
    red = LED(18)
    green = LED(23)
    return red, green

red_led, green_led = init_leds()

# ============================================================
# Streamlit UI (手動測試區)
# ============================================================
st.title("💡 燈號手動控制（測試模式）")
st.write("用於 GPIO / LED 硬體測試")

st.divider()
st.subheader("🔦 手動控制")

c1, c2 = st.columns(2)

with c1:
    if st.button("🔴 紅燈 ON"):
        red_led.on()
        st.success("紅燈已開啟")
    if st.button("🔴 紅燈 OFF"):
        red_led.off()
        st.success("紅燈已關閉")

with c2:
    if st.button("🟢 綠燈 ON"):
        green_led.on()
        st.success("綠燈已開啟")
    if st.button("🟢 綠燈 OFF"):
        green_led.off()
        st.success("綠燈已關閉")
```

結果如下：

7-2-3. AIoT 專案 1：LLM 智慧燈號控制系統

加入 LLM 控制燈號

1. 為了防止 LLM 亂回答，可以限制 LLM 的回答只能是 JSON 格式的回應，再進行 Python 程式控制 GPIO。
2. 避免 LLM + GPIO 每次 rerun 重啟，將 GPIO 初始化與 LLM 模型載入放入 Streamlit 的快取中。



7-3 LLM 模型，來進行文字的推論控制 GPIO 的輸出。

```
import streamlit as st
import ollama
import json
from gpiozero import LED
import os

# ============================================================
```

```
# 環境設定
# ============================================================
os.environ["OLLAMA_NUM_GPU"] = "1"

# ============================================================
# GPIO 初始化（只會執行一次）
# ============================================================
@st.cache_resource
def init_leds():
    red = LED(18)
    green = LED(23)
    return red, green

red_led, green_led = init_leds()

# ============================================================
# LLM 預熱（第一次呼叫較慢）
# ============================================================
@st.cache_resource
def warmup_llm():
    try:
        ollama.chat(
            model="llama3.2:3b",
            messages=[{"role": "user", "content": "hi"}],
            options={"num_predict": 1}
        )
    except:
        pass

warmup_llm()

# ============================================================
# LLM 語意分析（快速 JSON）
# ============================================================
def analyze_command(text):
    response = ollama.chat(
        model="llama3.2:3b",
        messages=[
            {
                "role": "system",
                "content": (
```

```
                    "只輸出 JSON，必須是以下之一："
                    '{"color":"red","action":"on"},'
                    '{"color":"red","action":"off"},'
                    '{"color":"green","action":"on"},'
                    '{"color":"green","action":"off"},'
                    '{"color":"all","action":"on"},'
                    '{"color":"all","action":"off"}'
                )
            },
            {"role": "user", "content": text}
        ],
        options={
            "temperature": 0,
            "num_predict": 20
        }
    )

    raw = response["message"]["content"].strip()
    try:
        return json.loads(raw)
    except json.JSONDecodeError:
        return None

# ============================================================
# Streamlit UI - LLM 控制
# ============================================================
st.title("💡 LLM 智慧燈號控制系統")
st.write("請輸入中文指令，例如：**請亮綠燈 / 把紅燈關掉 / 全部開 / 全部關**")

cmd = st.text_input("🗣️ 輸入指令")

if st.button("送出指令"):
    if not cmd.strip():
        st.warning("請先輸入指令")
        st.stop()

    result = analyze_command(cmd)

    if result is None:
```

```
88.         st.error("❌ LLM 回傳格式錯誤，無法解析")
89.         st.stop()
90.
91.     color = result.get("color")
92.     action = result.get("action")
93.
94.     st.write("🤖 LLM JSON 回傳：", result)
95.
96.     # ========================================================
97.     # 燈號控制（增加全部燈選項）
98.     # ========================================================
99.     targets = []
100.      if color == "red":
101.          targets.append(red_led)
102.      elif color == "green":
103.          targets.append(green_led)
104.      elif color == "all":
105.          targets = [red_led, green_led]
106.
107.      if not targets:
108.          st.error("❌ 無法識別的控制指令")
109.          st.stop()
110.
111.      for led in targets:
112.          if action == "on":
113.              led.on()
114.          elif action == "off":
115.              led.off()
116.
117.      if color == "all":
118.          st.success(f"💡 全部燈已 {'開啟' if action=='on' else '關閉
    '}")
119.      else:
120.          st.success(f"💡 {color}燈已 {'開啟' if action=='on' else '
                                                          關閉'}")
121.
122.  # ========================================================
123.  # 手動測試區（除錯用）
124.  # ========================================================
125.  st.divider()
```

```
st.subheader("🧨 手動控制（測試用）")

c1, c2 = st.columns(2)

with c1:
    if st.button("🔴 紅燈 ON"):
        red_led.on()
        st.success("紅燈已開啟")
    if st.button("🔴 紅燈 OFF"):
        red_led.off()
        st.success("紅燈已關閉")

with c2:
    if st.button("🟢 綠燈 ON"):
        green_led.on()
        st.success("綠燈已開啟")
    if st.button("🟢 綠燈 OFF"):
        green_led.off()
        st.success("綠燈已關閉")

st.write("💡 也可以在 LLM 指令中輸入 **全部開 / 全部關** 控制所有燈")
```

結果如下：

增加樹莓派硬體監控

主畫面是 LLM 智慧燈號控制，側邊欄顯示 Raspberry Pi 5 硬體即時監控。這樣畫面清楚、不互相干擾，也很像工業控制或 IoT Dashboard。UI (User Interface) 分工如下：

Main Page (主畫面)	Sidebar (側邊欄)
1. LLM 中文指令控制 LED 2. 手動燈號測試	1. 硬體監控 2. 自動刷新 (2 秒)

只要將 monitor.py 程式的硬體監控全部放進 st.sidebar，請自行執行 7-4_streamlit.py。原本 monitor.py 中的程式片段：

```
c1, c2, c3, c4, c5, c6 = st.columns(6)
```

全部放進 st.sidebar 中：

```
with st.sidebar:
    c1, c2 = st.columns(2)
    c3, c4 = st.columns(2)
    c5, c6 = st.columns(2)
```

結果如下：

實現工業儀表板：現場長期運轉版本

工業儀表板正解：放在主畫面最上方，用 CSS Grid 控制。目標：

1. 24×7 不當機。
2. LLM 卡住不影響 GPIO。
3. 溫度 / 資源異常可自救。
4. Streamlit 當 UI，不是核心控制。
5. 就算 UI 掛了，燈還能正常運作。

Streamlit 現在 (風險)：

```
├── LLM
└── GPIO
```

現場正確架構：

```
[ Streamlit UI + LLM ]
          │
          │  寫 JSON
          ▼
   /tmp/led_cmd.json   ← ★ 就是 CMD_FILE
          ▲
          │  讀 JSON
          │
[ LED Daemon (systemd) ]
          │
        GPIO
```

1. GPIO 不能依賴 UI 存活。
2. GPIO 核心守護程式 (必做)：建立『 燈號控制核心 』led_daemon.py。

GPIO 控制核心（長期運轉）

將硬體單獨運作，就算 Streamlit / LLM 掛掉，燈仍然正常。首先，建立一個專案資料夾 aiot_LED：

Step 1 撰寫 GPIO 控制核心，這個程式永遠跑，不碰 LLM、不碰 UI。

led_daemon GPIO 控制核心。

```python
# led_daemon.py
from gpiozero import LED
import json
import time
import os

RED = LED(18)
GREEN = LED(23)

CMD_FILE = "/tmp/led_cmd.json"

def apply(cmd):
    color = cmd.get("color")
    action = cmd.get("action")

    targets = []
    if color == "red":
        targets = [RED]
    elif color == "green":
        targets = [GREEN]
    elif color == "all":
        targets = [RED, GREEN]

    for led in targets:
        led.on() if action == "on" else led.off()

print("LED daemon started")

```

```
while True:
    if os.path.exists(CMD_FILE):
        try:
            with open(CMD_FILE) as f:
                cmd = json.load(f)
            apply(cmd)
            os.remove(CMD_FILE)
        except:
            pass
    time.sleep(0.2)
```

Step 2 讓 led_daemon 開機自動跑。

新增一個 led-daemon.service：

```
~$ sudo nano /etc/systemd/system/led-daemon.service
```

led-daemon.service 內容如下：

```
[Unit]
Description=LED Control Daemon
After=multi-user.target

[Service]
ExecStart=/usr/bin/python /home/你的帳戶名稱/led_daemon.py
Restart=always
RestartSec=1

[Install]
WantedBy=multi-user.target
```

永久啟動/立即啟用 led-daemon：

```
~$ sudo systemctl enable led-daemon    # 永久啟動
~$ sudo systemctl start led-daemon    # 立即啟用
```

查看服務目前狀態：

```
~$ sudo systemctl status led-daemon
```

永久停用/立即停用 led-daemon：

```
~$ sudo systemctl stop led-daemon    # 永久停用
~$ sudo systemctl disable led-daemon    # 立即停用
```

Step 3 撰寫 Streamlit 的 UI。

app GPIO 控制核心。

```
1.  # app.py
2.  from gpiozero import LED
3.  import streamlit as st
4.  import ollama
5.  import json
6.  import subprocess
7.  import time
8.  import os
9.
10. CMD_FILE = "/tmp/led_cmd.json"
11.
12. # ============================================================
13. # 基本設定
14. # ============================================================
15. st.set_page_config(
16.     page_title="工業儀表板 • LLM 燈號控制",
17.     layout="wide"
18. )
19.
20. os.environ["OLLAMA_NUM_GPU"] = "1"
21.
22. # ============================================================
23. # CSS（科技感工業風）
24. # ============================================================
25. st.markdown("""
26. <style>
27. body { background-color:#020617; }
28. .dashboard {
29.     display:grid;
30.     grid-template-columns:repeat(6,1fr);
31.     gap:14px;
```

```
    margin-bottom:1rem;
}
.card {
    background:linear-gradient(145deg,#020617,#020617);
    border:1px solid #1e293b;
    border-radius:14px;
    padding:14px 10px;
    text-align:center;
}
.card h4 {
    margin:0;
    font-size:0.9rem;
    color:#94a3b8;
}
.card .value {
    font-size:2.1rem;
    font-weight:800;
    color:#e5e7eb;
}
.card .sub {
    font-size:0.8rem;
    color:#64748b;
}
.warn {
    background:linear-gradient(145deg,#450a0a,#020617);
    border-color:#dc2626;
}
</style>
""", unsafe_allow_html=True)

# ============================================================
# 標題
# ============================================================
st.markdown(
    "<h2 style='color:#e5e7eb;margin-bottom:0.8rem'>🤗 工業儀表板 · LLM 燈號控制</h2>",
    unsafe_allow_html=True
)

# ============================================================
# 系統工具
```

```
72. # ============================================================
73. def run(cmd):
74.     return subprocess.check_output(cmd, shell=True,
    text=True).strip()
75.
76. def cpu_usage():
77.     idle = run(
78.         "top -bn1 | grep 'Cpu(s)' | sed 's/,/ /g' | "
79.         "awk '{for(i=1;i<=NF;i++) if ($i==\"id\") print $(i-1)}'"
80.     )
81.     return 100 - float(idle)
82.
83. def ram_usage():
84.     _, _, p = run("free -m | awk 'NR==2{print
    $3,$2,$3/$2*100}'").split()
85.     return float(p)
86.
87. def swap_usage():
88.     _, _, p = run("free -m | awk 'NR==3{print
    $3,$2,($3/$2)*100}'").split()
89.     return float(p)
90.
91. def disk_usage():
92.     _, _, p = run("df -h / | awk 'NR==2{print $3,$2,$5}'").split()
93.     return p
94.
95. def cpu_temp():
96.     return float(run("vcgencmd measure_temp | cut -d= -f2 | tr -d
    \"'C\""))
97.
98. def net_usage():
99.     iface = run(
100.         "ip -o link show | awk -F': ' '{print $2}' | "
101.         "grep -E '^(eth|wlan|usb)' | head -n1"
102.     )
103.     rx, tx = run(
104.         f"cat /proc/net/dev | grep {iface} | awk '{{print
    $2,$10}}'"
105.     ).split()
106.     return iface, int(tx)//1024**2, int(rx)//1024**2
107.
108. # ============================================================
```

```
# LLM + fallback（現場保命）
# ============================================================
def analyze(text):
    try:
        r = ollama.chat(
            model="llama3.2:3b",
            messages=[
                {
                    "role": "system",
                    "content": (
                        "只輸出 JSON："
                        '{"color":"red","action":"on"},'
                        '{"color":"red","action":"off"},'
                        '{"color":"green","action":"on"},'
                        '{"color":"green","action":"off"},'
                        '{"color":"all","action":"on"},'
                        '{"color":"all","action":"off"}'
                    )
                },
                {"role": "user", "content": text}
            ],
            options={"temperature": 0, "num_predict": 20}
        )
        return json.loads(r["message"]["content"])
    except:
        return fallback(text)

def fallback(text):
    if "紅" in text and "開" in text:
        return {"color":"red","action":"on"}
    if "紅" in text and "關" in text:
        return {"color":"red","action":"off"}
    if "綠" in text and "開" in text:
        return {"color":"green","action":"on"}
    if "綠" in text and "關" in text:
        return {"color":"green","action":"off"}
    if "全部" in text and "開" in text:
        return {"color":"all","action":"on"}
    if "全部" in text and "關" in text:
        return {"color":"all","action":"off"}
```

```
149.      return None
150.
151.  def send_cmd(cmd):
152.      tmp = CMD_FILE + ".tmp"
153.      with open(tmp, "w") as f:
154.          json.dump(cmd, f)
155.          f.flush()
156.          os.fsync(f.fileno())
157.      os.replace(tmp, CMD_FILE)
158.
159.  # ============================================================
160.  # 取得監控數據
161.  # ============================================================
162.  cpu  = cpu_usage()
163.  ram  = ram_usage()
164.  swap = swap_usage()
165.  disk = disk_usage()
166.  temp = cpu_temp()
167.  iface, tx, rx = net_usage()
168.
169.  # ============================================================
170.  # 儀表板顯示
171.  # ============================================================
172.  st.markdown(f"""
173.  <div class="dashboard">
174.    <div class="card"><h4>CPU</h4><div
    class="value">{cpu:.0f}%</div></div>
175.    <div class="card"><h4>RAM</h4><div
    class="value">{ram:.0f}%</div></div>
176.    <div class="card"><h4>SWAP</h4><div
    class="value">{swap:.0f}%</div></div>
177.    <div class="card"><h4>DISK</h4><div
    class="value">{disk}</div></div>
178.    <div class="card {'warn' if temp>=80 else ''}">
179.        <h4>TEMP</h4><div class="value">{temp:.1f}°C</div>
180.    </div>
181.    <div class="card">
182.        <h4>NET</h4>
183.        <div class="value">{iface}</div>
184.        <div class="sub">↑{tx} ↓{rx} MB</div>
```

```
  </div>
</div>
""", unsafe_allow_html=True)

# ============================================================
# 燈號控制
# ============================================================
st.subheader("💡 燈號控制")
cmd = st.text_input("中文指令（例：開紅燈 / 關綠燈 / 全部關）")

if st.button("送出指令"):
    result = analyze(cmd)
    if result:
        send_cmd(result)
        st.success(f"已送出：{result}")
    else:
        st.error("無法解析指令")

# ============================================================
# 過熱硬保護
# ============================================================
if temp >= 85:
    send_cmd({"color":"all","action":"off"})
    st.error("⚠ CPU 過熱，已強制關燈")

# ============================================================
# 自動刷新
# ============================================================
time.sleep(3)
st.rerun()
```

結果如下：

7-2-4. LLM 智慧音樂

數位輸出：蜂鳴器

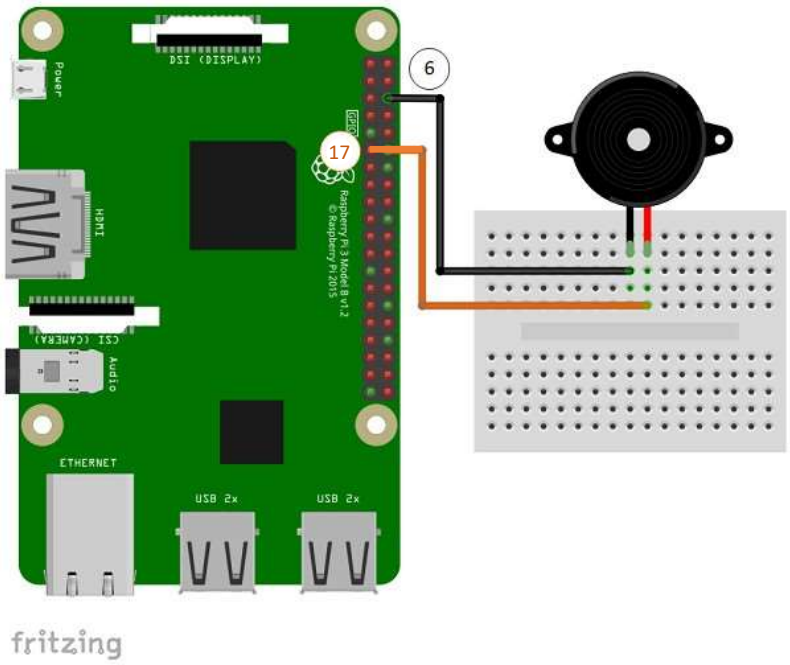
蜂鳴器 (PIEZO) 是一種壓電式喇叭 (Piezoelectric Speaker) 的電子元件，我們可以透過 0 和 1 來讓蜂鳴器發出音效。

所需的電子材料

- 無源蜂鳴器 x 1
- 麵包板 x 1
- 公-母杜邦線 x 2

實驗步驟

Step 1　連接樹莓派與蜂鳴器。



7-5 讓蜂鳴器發出音效。

```
1.  from gpiozero import PWMOutputDevice
2.  from time import sleep
3.
4.  buzzer = PWMOutputDevice(17)
5.
6.  while True:
7.      buzzer.frequency = 1000    # 頻率 (Hz，1000Hz 約為嗶聲)
8.      buzzer.value = 0.5    # 音量 (0 ~ 1)
9.      sleep(1)
10.
11.     buzzer.off()
12.     sleep(1)
```

GPIO 電子鋼琴

可利用網頁上的功能實現電子鋼琴的按鍵與音符。

7-6 讓蜂鳴器發出音效。

```
1.  import streamlit as st
2.  from gpiozero import PWMOutputDevice
3.  from time import sleep
4.
5.  st.set_page_config(page_title="GPIO 鋼琴", layout="centered")
```

```
6.
7.  # ==========================
8.  # GPIO 初始化（只做一次）
9.  # ==========================
10. @st.cache_resource
11. def init_buzzer():
12.     return PWMOutputDevice(17)
13.
14. buzzer = init_buzzer()
15.
16. # ==========================
17. # 音階設定（C 大調）
18. # ==========================
19. NOTES = {
20.     "C": 262,
21.     "D": 294,
22.     "E": 330,
23.     "F": 349,
24.     "G": 392,
25.     "A": 440,
26.     "B": 494,
27.     "C5": 523
28. }
29.
30. # ==========================
31. # 播放音符
32. # ==========================
33. def play_tone(freq, duration=0.3):
34.     buzzer.frequency = freq
35.     buzzer.value = 0.5   # 音`量（0~1）
36.     sleep(duration)
37.     buzzer.off()
38.
39. # ==========================
40. # UI
41. # ==========================
42. st.title("🎹 Raspberry Pi GPIO 鋼琴")
43. st.write("點擊按鈕播放音符（GPIO17 被動蜂鳴器）")
44.
45. cols = st.columns(len(NOTES))
```

```
46.
47. for col, (note, freq) in zip(cols, NOTES.items()):
48.     with col:
49.         if st.button(note, use_container_width=True):
50.             play_tone(freq)
51.
52. st.divider()
53.
54. # ==========================
55. # 安全停止（保護用）
56. # ==========================
57. if st.button("🔇 停止蜂鳴器"):
58.     buzzer.off()
```

結果如下：

7-2-5. AIoT 專案 2：LLM 智慧音樂盒

可利用網頁上的功能實現電子鋼琴的按鍵與音符。

7-7 LLM 智慧音樂盒。

```
import streamlit as st
import ollama
import json
from gpiozero import PWMOutputDevice
from time import sleep
import os

os.environ["OLLAMA_NUM_GPU"] = "1"

# ============================================================
# GPIO 初始化（只會執行一次）
# ============================================================
@st.cache_resource
def init_buzzer():
    return PWMOutputDevice(17)

buzzer = init_buzzer()

# ============================================================
# 音階表（白名單）
# ============================================================
NOTES = {
    "C": 262,
    "D": 294,
    "E": 330,
    "F": 349,
    "G": 392,
    "A": 440,
    "B": 494,
    "C5": 523
}

# ============================================================
# LLM 語意 → 旋律 JSON
# ============================================================
```

```
def analyze_music(text):
    prompt = f"""
你是一個「旋律生成器」。

請「只輸出 JSON」，不要解釋、不要多餘文字。
JSON 格式必須完全符合：

{{
  "melody": [
    {{"note":"C","duration":0.4}},
    {{"note":"E","duration":0.4}}
  ]
}}

規則：
- 只能使用音符：C D E F G A B C5
- duration 範圍：0.2 ~ 0.8
- 音符數量 8～16 個
- 不要輸出任何其他文字

使用者需求：
{text}
"""

    response = ollama.chat(
        model="llama3",
        messages=[
            {"role": "system", "content": "你只能輸出 JSON"},
            {"role": "user", "content": prompt}
        ]
    )

    raw = response["message"]["content"].strip()

    try:
        return json.loads(raw)
    except json.JSONDecodeError:
        return None

```

```
# ============================================================
# 播放旋律（安全執行）
# ============================================================
def play_melody(melody):
    for item in melody:
        note = item.get("note")
        duration = float(item.get("duration", 0.4))

        if note not in NOTES:
            continue

        buzzer.frequency = NOTES[note]
        buzzer.value = 0.5
        sleep(duration)
        buzzer.off()

# ============================================================
# Streamlit UI
# ============================================================
st.title("🎼 LLM 自動作曲 → GPIO 彈奏")
st.write("輸入中文描述，LLM 生成旋律並由蜂鳴器播放")

cmd = st.text_input(
    "🗣️ 輸入需求（例如：快樂的旋律 / 悲傷慢歌）",
    "請生成一段快樂、輕快的旋律"
)

if st.button("🤖 LLM 作曲並播放"):
    if not cmd.strip():
        st.warning("請輸入描述")
        st.stop()

    result = analyze_music(cmd)

    if result is None:
        st.error("❌ LLM 回傳格式錯誤")
        st.stop()

    melody = result.get("melody", [])

```

```
115.     st.subheader("🎵 LLM 生成旋律（JSON）")
116.     st.json(result)
117.
118.     play_melody(melody)
119.
120. # ============================================================
121. # 安全停止
122. # ============================================================
123. st.divider()
124.
125. if st.button("🔕 停止播放"):
126.     buzzer.off()
```

結果如下：