

# 树莓派实验二

## 实验内容

按键控制呼吸灯的开启及关闭，红绿蓝交替呼吸灯效果

要求：按键按一下，交替呼吸灯开启

按键再按一下，关闭

如此循环

阅读参考书相关介绍，理解 `dweet_led.py` 并整理进实验报告

## 实验代码

```
from gpiozero import TrafficLights, Button
```

```
from time import sleep
```

```
lights = TrafficLights(2, 3, 4)
```

```
#lights.green.on()
```

```
button = Button(12, pull_up=False)
```

```
def breath():
```

```
    # while True:
```

```
        sleep(1)
```

```
        lights.green.off()
```

```
        lights.amber.on()
```

```
        sleep(1)
```

```
        lights.amber.off()
```

```
lights.red.on()

sleep(1)

lights.red.off()

lights.green.on()

# sleep(1)

# lights.green.off()

# lights.amber.on()

def stop():

    lights.green.off()

    lights.amber.off()

    lights.red.off()

# button.when_pressed = breath

# button.when_released = stop


# 定义按键事件处理函数

def button_pressed() :

    print("button is pressed")

    global running

    if running :

        running = False

        print("关灯")
```

```
    else :  
        running = True  
        print("开灯")  
  
running = True  
flag = True  
while True:  
    if flag :  
        print("默认开灯")  
        breath()  
        flag = False  
    if running :  
        print("主函数: breath")  
        breath()  
    else :  
        print("主函数: stop")  
        stop()  
button.when_activated = button_pressed
```

实验视频见附件

# 理解代码

dweet\_led.py

运行结果

```
{ "this": "succeeded", "by": "dweeting", "the": "dweet", "with": { "thing": "81ba4edc", "created": "2024-03-08T13:32:18.798Z", "content": { "state": "blink", "transaction": "9e9d9b9c-9ec1-48c1-aedb-595ae9b51283" } } }

{ "this": "succeeded", "by": "dweeting", "the": "dweet", "with": { "thing": "81ba4edc", "created": "2024-03-08T13:32:08.172Z", "content": { "state": "off", "transaction": "1015281d-2cdd-45e9-a41d-dd0d1ca179f6" } } }

{ "this": "succeeded", "by": "dweeting", "the": "dweet", "with": { "thing": "81ba4edc", "created": "2024-03-08T13:31:53.545Z", "content": { "state": "on", "transaction": "432fd801-ea97-4fe4-ae42-8875a3ffba2b" } } }
```

参考 [dweet.io](https://dweet.io) 和相关博客可知

该代码通过 [dweet.io](https://dweet.io) 来远程控制 LED 灯。

## 1. 依赖

```
import signal
import json
import os
import sys
import logging
from gpiozero import Device, LED
from gpiozero.pins.pigpio import PiGPIOFactory
from time import sleep
from uuid import uuid1
import requests
```

依赖 [gpiozero](https://gpiozero.readthedocs.io/en/stable/) 来操作 GPIO, [pigpio](https://pigpio.org/) 作为 GPIO 的底层驱动, [requests](https://requests.readthedocs.io/en/latest/) 用于 HTTP 通信。

## 2. GPIO 配置:

```
38 # Initialize GPIO
39 Device.pin_factory = PiGPIOFactory()
```

-使用`Device.pin\_factory = PiGPIOFactory()`来初始化 GPIO 工厂，这是针对 Raspberry Pi 的配置。

## 3. 日志设置:

```
32 # Initialize Logging
33 logging.basicConfig(level=logging.WARNING) # Global logging configuration
34 logger = logging.getLogger('main') # Logger for this module
35 logger.setLevel(logging.INFO) # Debugging for this file.
```

使用`logging`模块，有全局的日志设置和针对当前文件的调试级别设置。

## 4. 全局变量:

```
23 # Global Variables
24 LED_GPIO_PIN = 21 # GPIO Pin that LED is connected to
25 THING_NAME_FILE = 'thing_name.txt' # The name of our "thing" is persisted into this file
26 URL = 'https://dweet.io' # Dweet.io service API
27 last_led_state = None # Current state of LED ("on", "off", "blinking")
28 thing_name = None # Thing name (as persisted in THING_NAME_FILE)
29 led = None # GPIOZero LED instance
```

LED 连接的 GPIO 引脚号`LED\_GPIO\_PIN`(21)，LED 的初始状态为`off`，`thing\_name`变量用于存储“thing”的名称，`thing\_name.txt`用于记录“thing”名称。

## 5. 函数定义:

-`init\_led()`：初始化 LED 对象。

```
42 # Function Definitions
43 def init_led():
44     """Create and initialise an LED Object"""
45     global led
46     led = LED(LED_GPIO_PIN)
47     led.off()
```

-`resolve\_thing\_name(thing\_file)`：从文件中读取或创建一个新的“thing”名称。

如果不从文件中读取`thing\_name`，就生成一个`uuid1`对象

ps:

UUID (Universally Unique Identifier, 通用唯一识别码) 是一种用于标识信息的技术，它旨在创建一个唯一的标识符，这个标识符在整个宇宙中几乎是不可能重复的。UUID 有多种格式，包括 UUID1、UUID3、UUID4、UUID5 和 UUID6，但在 Python 中通常使用的是 UUID1 和 UUID4。

UUID1 是基于时间和节点 ID (MAC 地址) 的 UUID。它通常包含一个时间戳和一个节点 ID，因此它可以被认为具有一定的可追溯性。这意味着 UUID1 生成的标识符可能与特定的设备和时间相关联。

在许多应用场景中，UUID 被用来唯一标识数据库记录、网络设备、软件配置等。由于它们

的唯一性和不可预测性，UUID 在需要确保标识符全球唯一性的场合是非常有用的。

```
50 def resolve_thing_name(thing_file):
51     """Get existing, or create a new thing name"""
52     if os.path.exists(thing_file): # (3)
53         with open(thing_file, 'r') as file_handle:
54             name = file_handle.read()
55             logger.info('Thing name ' + name + ' loaded from ' + thing_file)
56             return name.strip()
57     else:
58         name = str(uuid1())[0:8] # UUID object to string. # (4)
59         logger.info('Created new thing name ' + name)
60
61         with open(thing_file, 'w') as f: # (5)
62             f.write(name)
63
64     return name
```

- `get_latest_dweet()`: 从 dweet.io 获取关于这个“thing”的 dweet\_content。

```
67 def get_latest_dweet():
68     """Get the last dweet made by our thing."""
69     resource = URL + '/get/latest/dweet/for/' + thing_name # (6)
70     logger.debug('Getting last dweet from url %s', resource)
71
72     r = requests.get(resource) # (7)
73
74     if r.status_code == 200: # (8)
75         dweet = r.json() # return a Python dict.
76         logger.debug('Last dweet for thing was %s', dweet)
77
78         dweet_content = None
79
80         if dweet['this'] == 'succeeded': # (9)
81             # We're just interested in the dweet content property.
82             dweet_content = dweet['with'][0]['content'] # (10)
83
84         return dweet_content
85
86     else:
87         logger.error('Getting last dweet failed with http status %s', r.status_code)
88         return {}
```

- `poll_dweets_forever(delay_secs=2)`: 不断地轮询（延时两秒）dweet.io 服务来检查关于 led 的新 dweet。

```
91 def poll_dweets_forever(delay_secs=2):
92     """Poll dweet.io for dweets about our thing."""
93     while True:
94         dweet = get_latest_dweet() # (11)
95         if dweet is not None:
96             process_dweet(dweet) # (12)
97
98             sleep(delay_secs) # (13)
99
```

- `stream_dweets_forever()`: 无限期地监听来自 dweet.io 关于 thing 的流式 dweet。

ps:

流式 dweet 的特点包括:

1. 实时性: 数据可以立即传输和接收, 适用于需要快速响应的应用。
2. 连续性: 数据可以持续地传输, 而不需要每次都重新建立连接。
3. 轻量级: 因为数据是逐个发送的, 所以可以在保持较低的网络负载的同时传输大量数据。
4. 可靠性: 通常会有机制来确保数据的完整性和顺序性。

```

101 def stream_dweets_forever():
102     """Listen for streaming for dweets"""
103     resource = URL + '/listen/for/dweets/from/' + thing_name
104     logger.info('Streaming dweets from url %s', resource)
105
106     session = requests.Session()
107     request = requests.Request("GET", resource).prepare()
108
109     while True: # while True to reconnect on any disconnections.
110         try:
111             response = session.send(request, stream=True, timeout=1000)
112
113             for line in response.iter_content(chunk_size=None):
114                 if line:
115                     try:
116                         json_str = line.splitlines()[1]
117                         json_str = json_str.decode('utf-8')
118                         dweet = json.loads(eval(json_str)) # json_str is a string in a string.
119                         logger.debug('Received a streamed dweet %s', dweet)
120
121                         dweet_content = dweet['content']
122                         process_dweet(dweet_content)
123                     except Exception as e:
124                         logger.error(e, exc_info=True)
125                         logger.error('Failed to process and parse dweet json string %s', json_str)
126
127         except requests.exceptions.RequestException as e:
128             # Lost connection. The While loop will reconnect.
129             #logger.error(e, exc_info=True)
130             pass
131
132     except Exception as e:
133         logger.error(e, exc_info=True)

```

- `process\_dweet(dweet)`：处理接收到的 dweet 内容，设置 LED 的状态。

led\_state 为"on","blink","off"分别对应触发 led.on(),led.blink(),led.off(),实现了 dweet 的远程控制

```

136 def process_dweet(dweet):
137     """Inspect the dweet and set LED state accordingly"""
138     global last_led_state
139
140     if not 'state' in dweet:
141         return
142
143     led_state = dweet['state']
144
145     if led_state == last_led_state: # (14)
146         return # LED is already in requested state.
147
148     if led_state == 'on': # (15)
149         led.on()
150     elif led_state == 'blink':
151         led.blink()
152     else: # Off, including any unhandled state.
153         led_state = 'off'
154         led.off()
155
156     if led_state != last_led_state: (variable) led_state: Literal['on', 'blink', 'off'] # (16)
157         last_led_state = led_state
158         logger.info('LED ' + led_state)

```

- `print\_instructions()`：打印如何控制 LED 的说明。



```

161 def print_instructions():
162     """Print instructions to terminal."""
163     print("LED Control URLs - Try them in your web browser:")
164     print("  On    : " + URL + "/dweet/for/" + thing_name + "?state=on")
165     print("  Off   : " + URL + "/dweet/for/" + thing_name + "?state=off")
166     print("  Blink : " + URL + "/dweet/for/" + thing_name + "?state=blink\n")

```

- `signal\_handler(sig, frame)`: 处理退出信号，如 CTRL+C，正确关闭 LED。

```

169 def signal_handler(sig, frame):
170     """Release resources and clean up as needed."""
171     print('You pressed Control+C')
172     led.off()
173     sys.exit(0)

```

## 6.主程序:

- 主程序首先设置了一个信号处理器来处理退出信号(CTRL+C)。当信号处理器被触发时，LED 会被关闭，程序退出。

```

181 # Main entry point
182 if __name__ == '__main__':
183     signal.signal(signal.SIGINT, signal_handler) # Capture CTRL + C
184     print_instructions() # (17)
185

```

- 接着初始化 LED 并根据最新的 dweet 调用 process\_dweet()设置 LED 状态。

```

186 # Initialise LED from last dweet.
187 last_dweet = get_latest_dweet() # (18)
188 if last_dweet:
189     process_dweet(last_dweet)
190

```

- 然后打印控制 LED 的说明并进入一个无限循环等待 dweet 的到来。循环中可以选择实时流式 dweet 或定期轮询 dweet (流式: stream\_dweets\_forever(), 轮询: poll\_dweets\_forever())。

```

191 print('Waiting for dweets. Press Control+C to exit.')
192 # Only use one of the following. See notes later in Chapter.
193 # stream_dweets_forever() # Stream dweets real-time.
194 poll_dweets_forever() # Get dweets by polling a URL on a schedule. # (19)
195

```