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

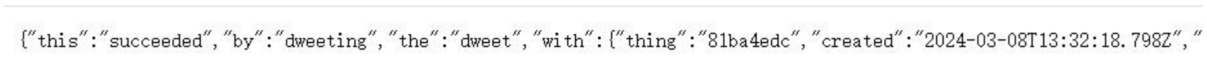
```
from gpiozero import TrafficLights,Button
from time import sleep
from gpiozero import PWMLED
from signal import pause

red_led = PWMLED(2)
blue_led = PWMLED(3)
green_led = PWMLED(4)

button = Button(12,pull_up=False)
def breath():
        sleep(2)
        green_led.off()
        blue_led.pulse()
        sleep(2)
        bule_led.off()
        red_led.pulse()
        sleep(2)
        red_led.off()
        green_led.pulse()

def stop:
        green_led.off()
        red_led.off()
        blue_led.off()

#  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_pressed = button_pressed
```

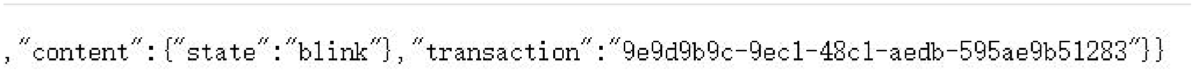
实验视频见附件

理解代码

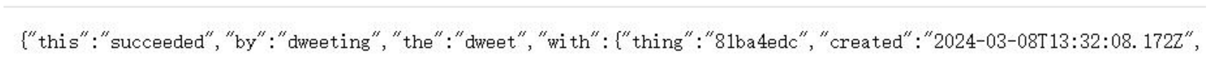
dweet_led.py
运行结果



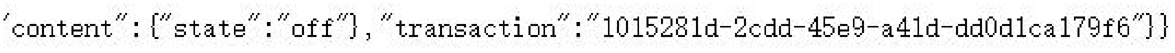
```
{"this":"succeeded","by":"dweeting","the":"dweet","with":{"thing":"81ba4edc","created":"2024-03-08T13:32:18.798Z","
```



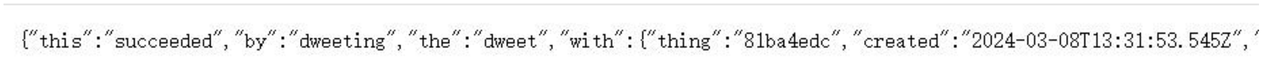
```
,"content":{"state":"blink"},"transaction":"9e9d9b9c-9ec1-48c1-aedb-595ae9b51283"}}
```



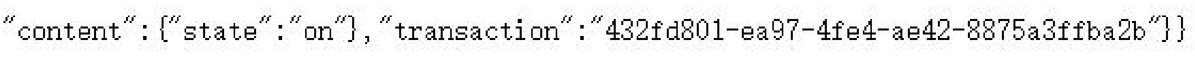
```
{″this″:″succeeded″,″by″:″dweeting″,″the″:″dweet″,″with″:{″thing″:″81ba4edc″,″created″:″2024-03-08T13:32:08.172Z″,
```



```
′content″:{″state″:″off″},″transaction″:″1015281d-2cdd-45e9-a41d-dd0d1ca179f6″}}
```

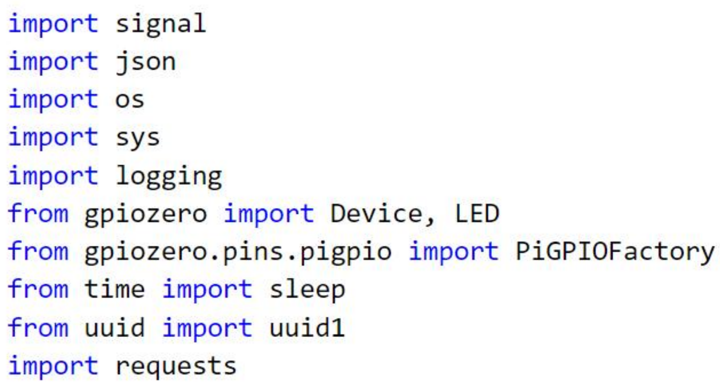


```
{″this″:″succeeded″,″by″:″dweeting″,″the″:″dweet″,″with″:{″thing″:″81ba4edc″,″created″:″2024-03-08T13:31:53.545Z″,′
```



```
″content″:{″state″:″on″},″transaction″:″432fd801-ea97-4fe4-ae42-8875a3ffba2b″}}
```

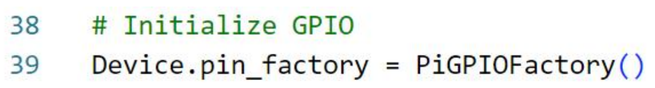
参考 dweet.io 和相关博客可知
该代码通过 dweet.io 来远程控制 LED 灯。
1. 依赖



```python
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 来操作 GPIO，pigpio 作为 GPIO 的底层驱动，requests 用于 HTTP 通信。

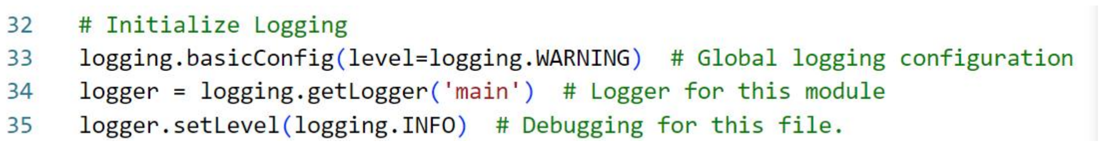
2. GPIO 配置：



```python
# Initialize GPIO
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。

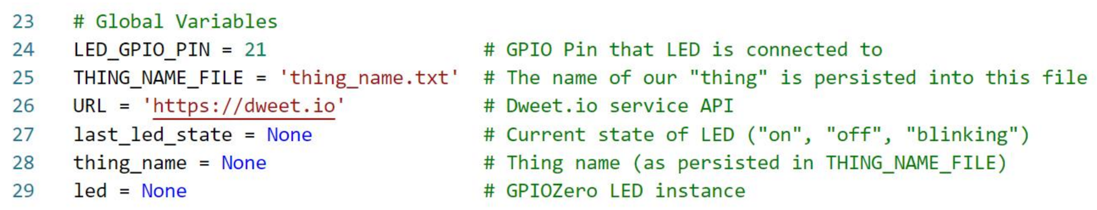
3. 日志设置：



```python
# Initialize Logging
logging.basicConfig(level=logging.WARNING)  # Global logging configuration
logger = logging.getLogger('main')  # Logger for this module
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：

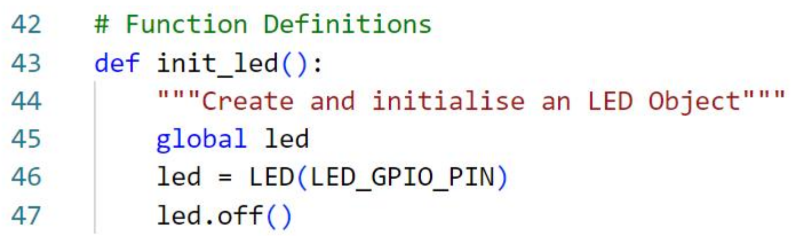


```python
# Global Variables
LED_GPIO_PIN = 21                   # GPIO Pin that LED is connected to
THING_NAME_FILE = 'thing_name.txt'  # The name of our "thing" is persisted into this file
URL = 'https://dweet.io'            # Dweet.io service API
last_led_state = None               # Current state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



```python
# Function Definitions
def init_led():
    """Create and initialise an LED Object"""
    global led
    led = LED(LED_GPIO_PIN)
    led.off()
```

- `resolve_thing_name(thing_file)`：从文件中读取或创建一个新的“thing”名称。

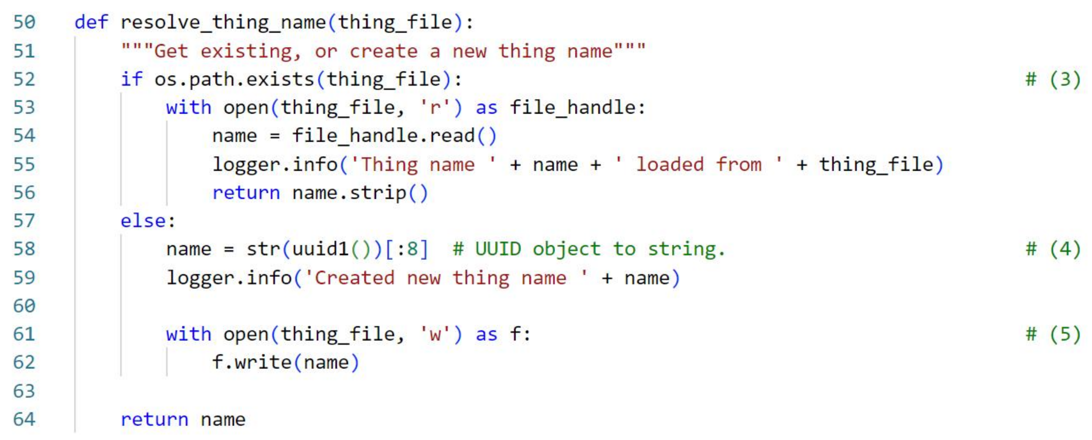
如果不从文件中读取 thing_name,就生成一个 uuid1 对象

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。

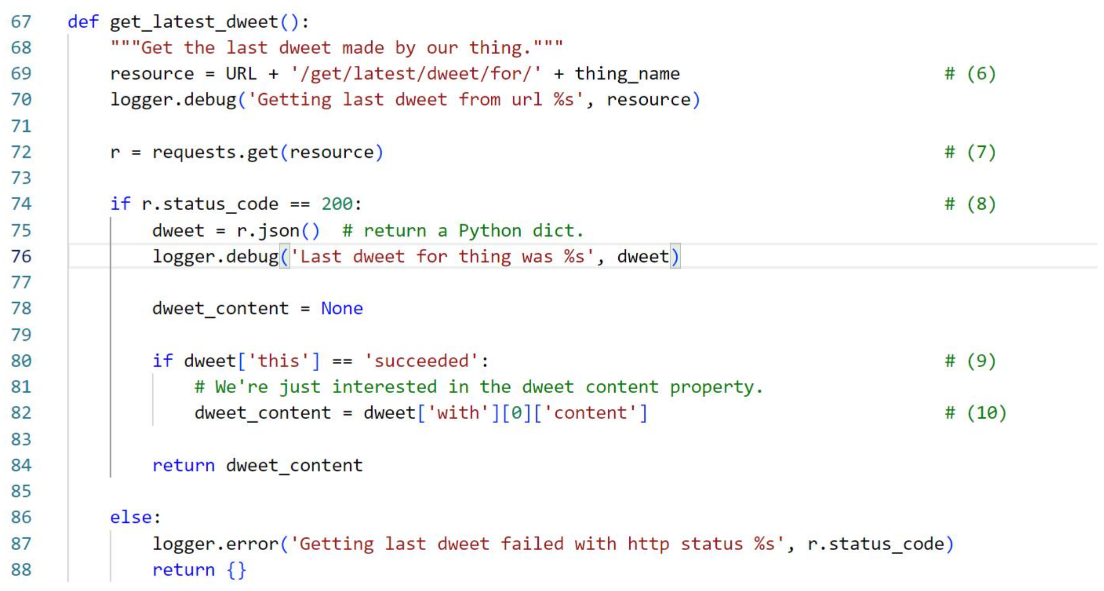
UUID1 是基于时间和节点 ID（MAC 地址）的 UUID。它通常包含一个时间戳和一个节点 ID，因此它可以被认为具有一定的可追溯性。这意味着 UUID1 生成的标识符可能与特定的设备和时间相关联。

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。



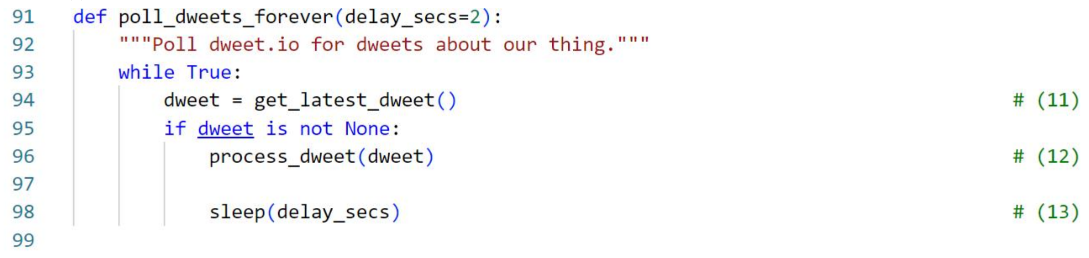
```python
def resolve_thing_name(thing_file):
    """Get existing, or create a new thing name"""
    if os.path.exists(thing_file):                                          # (3)
        with open(thing_file, 'r') as file_handle:
            name = file_handle.read()
            logger.info('Thing name ' + name + ' loaded from ' + thing_file)
            return name.strip()
    else:
        name = str(uuid1())[:8]  # UUID object to string.                   # (4)
        logger.info('Created new thing name ' + name)

        with open(thing_file, 'w') as f:                                    # (5)
            f.write(name)

    return name
```

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



```python
def get_latest_dweet():
    """Get the last dweet made by our thing."""
    resource = URL + '/get/latest/dweet/for/' + thing_name                # (6)
    logger.debug('Getting last dweet from url %s', resource)

    r = requests.get(resource)                                            # (7)

    if r.status_code == 200:                                              # (8)
        dweet = r.json()  # return a Python dict.
        logger.debug('Last dweet for thing was %s', dweet)

        dweet_content = None

        if dweet['this'] == 'succeeded':                                  # (9)
            # We're just interested in the dweet content property.
            dweet_content = dweet['with'][0]['content']                   # (10)

        return dweet_content

    else:
        logger.error('Getting last dweet failed with http status %s', r.status_code)
        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。



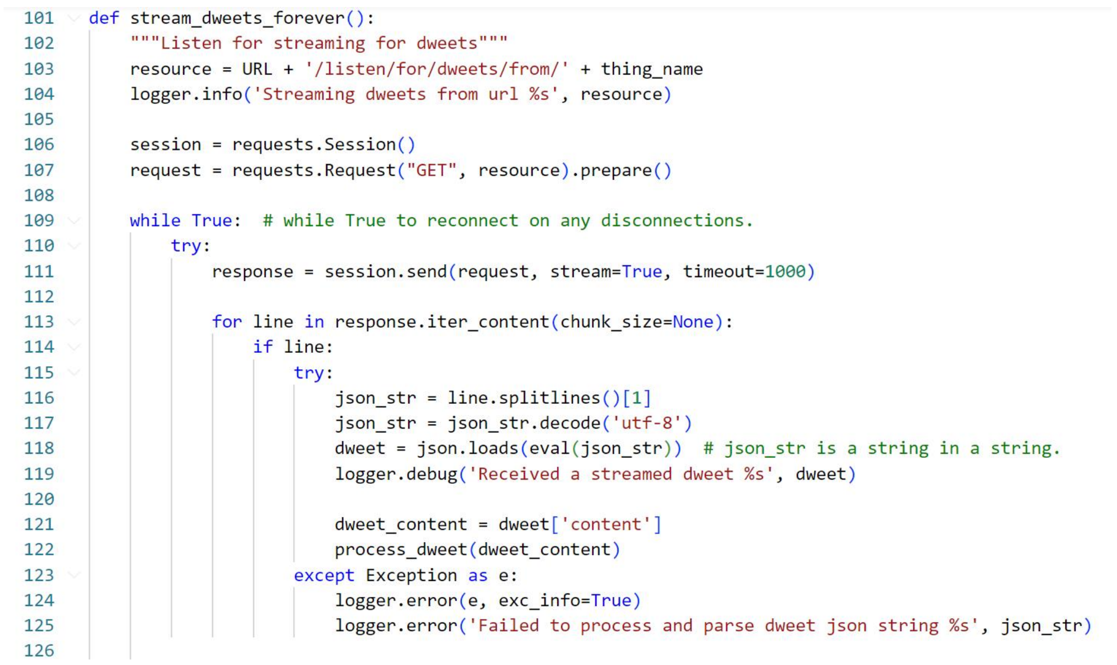
```python
def poll_dweets_forever(delay_secs=2):
    """Poll dweet.io for dweets about our thing."""
    while True:
        dweet = get_latest_dweet()                                        # (11)
        if dweet is not None:
            process_dweet(dweet)                                          # (12)

            sleep(delay_secs)                                             # (13)
```

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

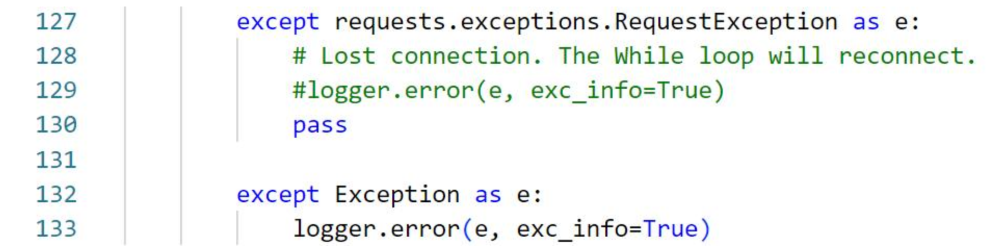
ps:

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。

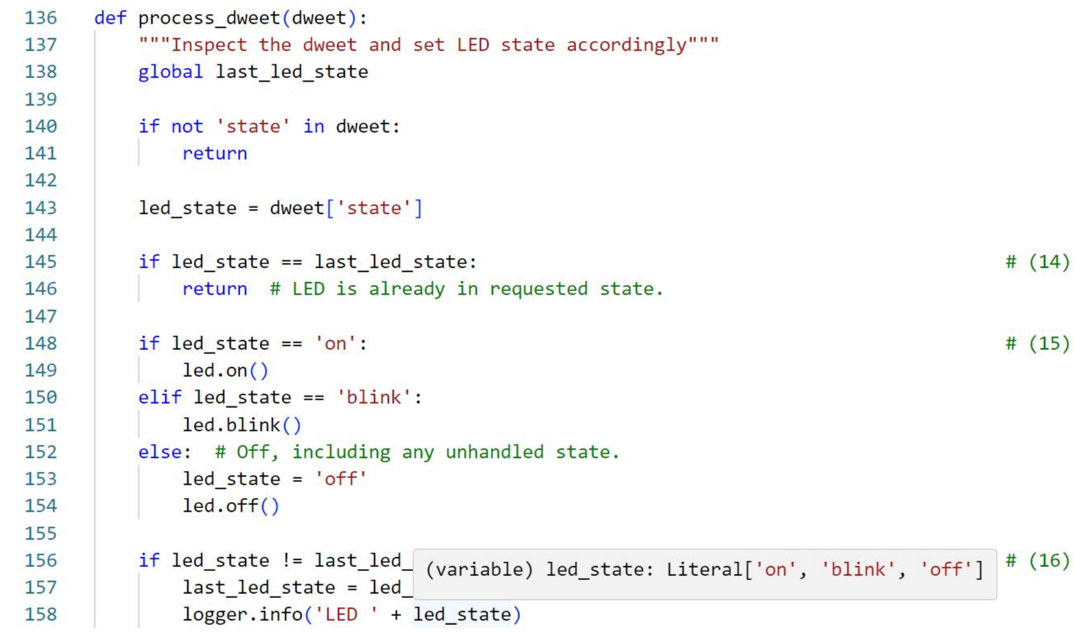


```python
def stream_dweets_forever():
    """Listen for streaming for dweets"""
    resource = URL + '/listen/for/dweets/from/' + thing_name
    logger.info('Streaming dweets from url %s', resource)

    session = requests.Session()
    request = requests.Request("GET", resource).prepare()

    while True:  # while True to reconnect on any disconnections.
        try:
            response = session.send(request, stream=True, timeout=1000)

            for line in response.iter_content(chunk_size=None):
                if line:
                    try:
                        json_str = line.splitlines()[1]
                        json_str = json_str.decode('utf-8')
                        dweet = json.loads(eval(json_str))  # json_str is a string in a string.
                        logger.debug('Received a streamed dweet %s', dweet)

                        dweet_content = dweet['content']
                        process_dweet(dweet_content)
                    except Exception as e:
                        logger.error(e, exc_info=True)
                        logger.error('Failed to process and parse dweet json string %s', json_str)

```



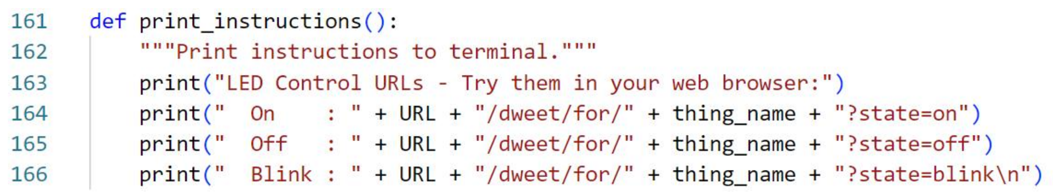
```python
        except requests.exceptions.RequestException as e:
            # Lost connection. The While loop will reconnect.
            #logger.error(e, exc_info=True)
            pass

        except Exception as e:
            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



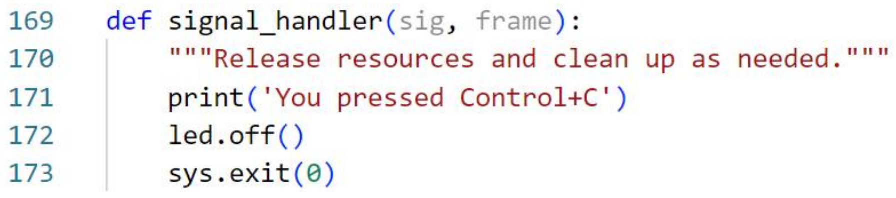
```python
def process_dweet(dweet):
    """Inspect the dweet and set LED state accordingly"""
    global last_led_state

    if not 'state' in dweet:
        return

    led_state = dweet['state']

    if led_state == last_led_state:                                         # (14)
        return  # LED is already in requested state.

    if led_state == 'on':                                                   # (15)
        led.on()
    elif led_state == 'blink':
        led.blink()
    else:  # Off, including any unhandled state.
        led_state = 'off'
        led.off()

    if led_state != last_led_ (variable) led_state: Literal['on', 'blink', 'off'] # (16)
        last_led_state = led_
        logger.info('LED ' + led_state)
```

- `print_instructions()`：打印如何控制 LED 的说明。



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

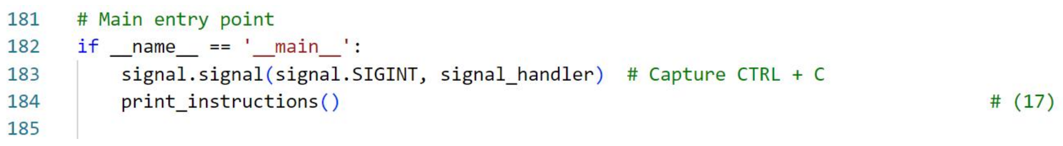
- `signal_handler(sig, frame)`：处理退出信号，如 CTRL+C，正确关闭 LED。



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

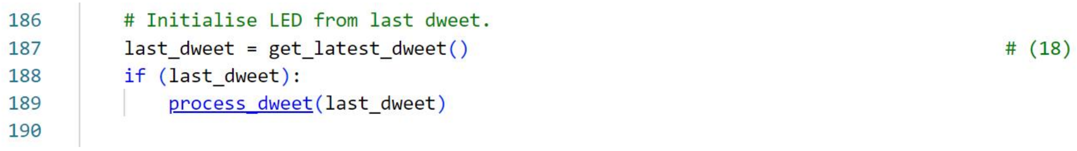
6.主程序：

- 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。



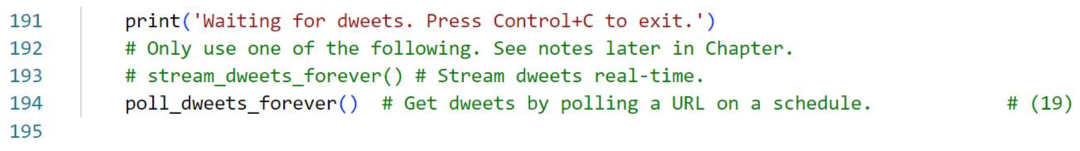
```python
# Main entry point
if __name__ == '__main__':
    signal.signal(signal.SIGINT, signal_handler)  # Capture CTRL + C
    print_instructions()                                                # (17)

```

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



```python
    # Initialise LED from last dweet.
    last_dweet = get_latest_dweet()                                     # (18)
    if (last_dweet):
        process_dweet(last_dweet)

```

- 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()）。



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

```