國立空中大學

「Python程式專題實作」

期末小組專題

「氣象預報系統」

開發文件

導師：賴素純

班級：ZZZ002

組別：第7組

組名：

組長：113122877 陳銘泓

組員：112222911 詹秉蒼

113126681 劉彥翎

113128305 傅柏凱

113122776 張心齡

1127075　羅若嘉

目 錄

一、　專題主題

氣象預報系統

二、　專題簡介

(一) 專題概述

本專題開發一個台灣氣象與地震資訊的視覺化查詢系統，透過使用者友好的圖形介面，為使用者提供即時、準確的氣象預報與地震資訊。系統結合了中央氣象署的開放資料API，讓使用者能夠便捷地獲取台灣各地區的天氣狀況與地震報告。

(二) 系統功能

氣象預報功能

地區選擇：用戶可選擇台灣任何城市及其行政區進行查詢
多天氣象預報：提供未來7天的詳細天氣預報資訊
全面氣象數據：包含溫度、體感溫度、降雨機率、相對濕度、紫外線指數等重要指標
天氣視覺化：使用直觀的圖標展示各種天氣現象，提升用戶體驗

地震資訊功能

即時地震報告：可查詢最新的地震資訊
地震分類查詢：區分「小區域有感地震」和「顯著有感地震」兩種類型
詳細地震數據：提供發生時間、震央位置、規模、深度、最大震度等完整資訊
資料視覺化呈現：表格化展示地震資料，易於閱讀與理解

（三）技術特點

1. 自適應介面設計：系統採用CustomTkinter框架，提供現代化的UI設計，支援深色/淺色模式切換
2. 資料可視化：結合圖標與表格清晰展示複雜的氣象與地震資訊
3. 動態資料整合：從API獲取資料後進行智能處理與整合，確保資訊的完整性與準確性
4. 跨平台相容性：支援Windows和iOS操作系統，擴大應用場景
5. 優化使用者體驗：提供簡潔直觀的操作流程，適合各年齡層用戶使用

（四）應用價值

1. 日常生活應用：幫助一般民眾規劃出行、穿著和活動安排
2. 防災科普教育：提升公眾對地震及極端天氣的認識與防範意識
3. 資訊教育示範：展示如何整合公開資料API，開發實用的應用程式
4. 開放源碼項目：可作為台灣氣象資訊應用開發的參考與基礎

此專題不僅展示了程式開發的技術能力，也體現了如何將公開資料轉化為實用服務的應用思維，為使用者提供有價值的氣象資訊服務。

三、小組分工

工作項目		人員
提案發想		陳銘泓
需求規格化		所有成員
專案計畫統籌		陳銘泓
需求分析		所有成員
程式撰寫	資料蒐集	詹秉蒼
	資料處理	張心齡
	資料呈現-氣象	劉彥翎
	資料呈現-氣象(地區選擇)	劉彥翎
	資料呈現-地震	傅柏凱
應用程式開發文件編纂		傅柏凱(主編) 所有成員
使用手冊製作		傅柏凱(文件編匯) 陳銘泓(Windows) 詹秉蒼(macOS)
執行檔編譯(Windows)		陳銘泓
執行檔編譯(MacOS)		詹秉蒼
整合測試		所有成員
報告投影片製作		羅若嘉
專題報告		陳銘泓

模組開發者	模組名稱
WeatherGUI.py	劉彥翎、傅柏凱
app.py	張心齡
RequestApi.py	詹秉蒼

四、開發環境

語言：Python 3.12.7

依賴套件：

- API 請求：requests
- GUI 介面：customtkinter、tkinter
- 圖片處理：Pillow（PIL）
- 資料處理：json、datetime、os
- 其他：functools.partial（用於事件綁定）

開發系統：Windows、macOS

流程圖：Draw.io

版本控制：Git、GitHub

IDE：Microsoft Visual Studio Code

資料來源：外部氣象 API（透過 RequestApi 模組）

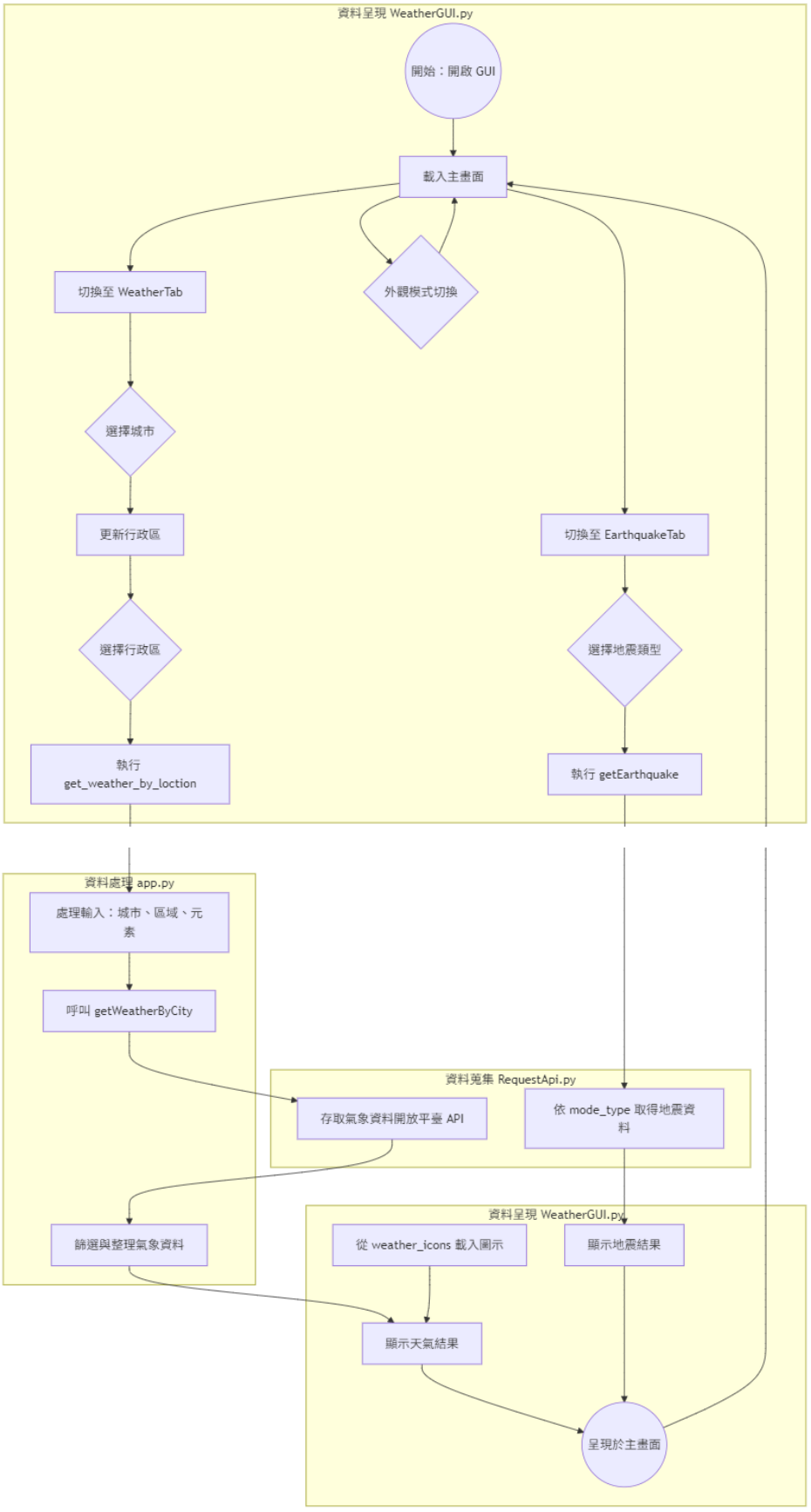
打包工具：PyInstaller（用於生成可執行檔）

五、系統開發流程

（一）需求分析

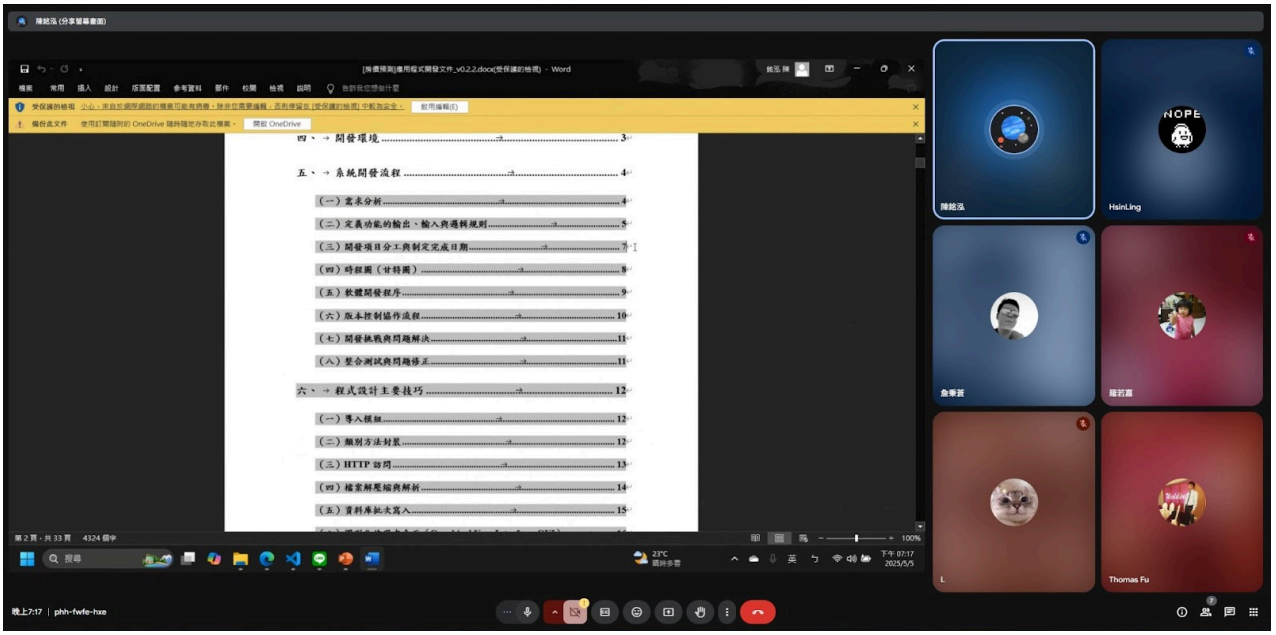
由全體組員進行討論，對於提案內容進行流程圖繪製、功能分解、使用套件、分配開發人員、制定函數、參數相關工作分派。

（二）功能的輸出、輸入與邏輯規則



（三）開發項目分工與制定完成日期

會議畫面截圖



於每次會議討論工作項目、分配工作並追蹤進度，會議紀錄如下表：

會議時間	討論內容
114年4月21日 19：00～20：00	1. 流程圖解說 2. 討論分工 3. 初步擬定作業時程
114年4月28日 19：00～20：00	1. 運行一遍程式的運作結果（由組長看完後合併到main分支，並由組長統一展示） 2. 各程式間的串接輸出與輸入的分析（確認所有屬性是否都存在、型別是否正常） 3. 指派程式串接事務，以及完成時間（1人，串接有問題，原開發者協助除錯與調整） 4. 著手進行文件、報告相關事宜（共同討論如何分工） 5. 作業時程規劃
114年5月5日 19：00～19：40	1. 未盡事宜工作分配（開發文件統整及使用手冊撰寫事宜） 2. 檢核程式完整性
113年5月12日 19：00～19：30	1. 共同檢視文件（開發文件、使用手冊）有無缺漏項目，並就不足部分分配撰寫、修改工作 2. 討論程式碼編譯及發佈事宜 3. 修改後報告投影片展示及說明

（四）時程圖(甘特圖)

時間 任務名稱	1	2	3	4	5	6
題目發想、提案報告撰寫、需求分析						
需求規格化、程式碼撰寫、開發						
測試（單元測試、整合測試、跨平台測試）						
報告投影片製作						
面授報告						
開發文件、 使用手冊製作						

註：1表示專案進行第一週（4/7～4/13）；2表示專案進行第二週（4/14～4/20）；以下以此類推。

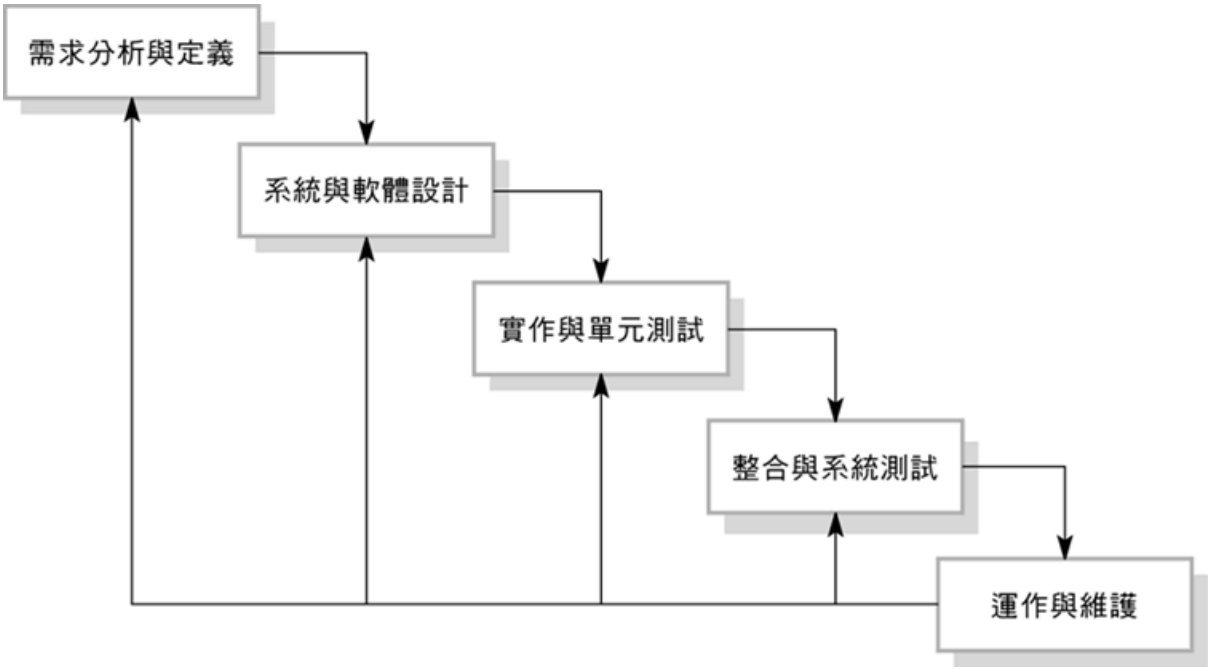
（五）軟體開發程序

軟體開發的程序有四種：瀑布式、漸進式、需求規格化、組合式的軟體開發程序。

本專案採傳統階梯式的軟體開發程序，亦即瀑布式的軟體開發程序，下圖為階梯式的軟體開發模型，也就是Waterfall model。

此軟體開發程序，講求軟體生命週期（Software life cycle），將軟體發展劃分為明確階段。此開發模式為標準的程序，為大多數人所接受，有利於軟體專案的管理。

由前揭甘特圖可見得，本專案開發程序中，有明確的階段。在經過系統需求分析後，進行需求規格化，接著進行系統開發實作，實作的過程中，亦同時進行單元測試，實作完畢後，進行整合測試，故屬瀑布式的軟體開發程序。

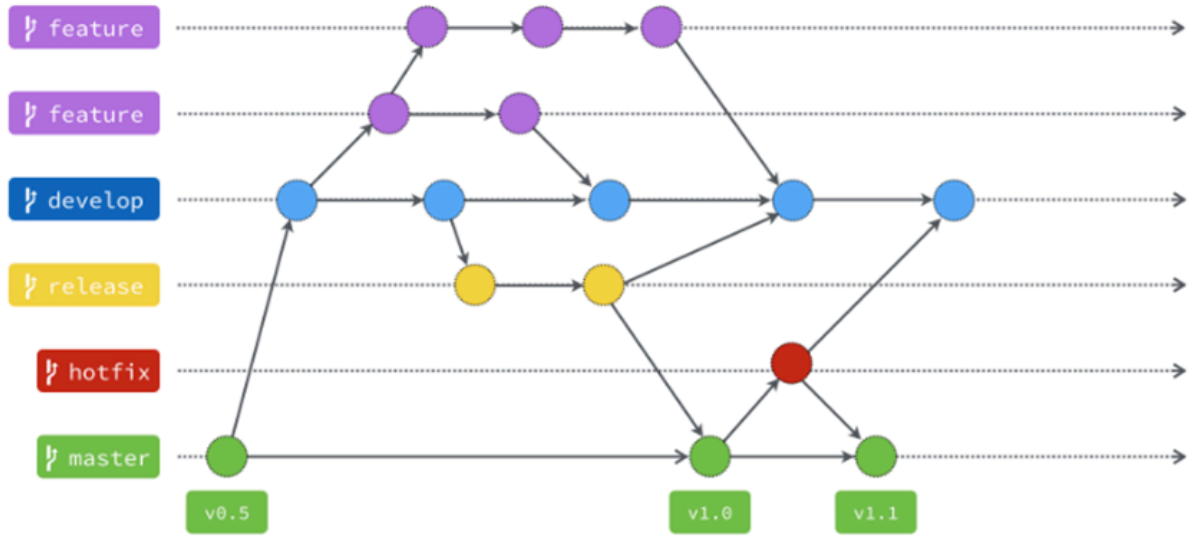


(圖片來源：https://wayne265265.pixnet.net/blog/post/113080214)

（六）版本控制協作流程

採用現今最流行的GitFlow流程，將Branch（分支）狀態系統化。

組員開發各自所負責之功能時，於各自之feature分支進行開發，功能開發完畢後，合併至develop分支，待功能測試完畢，整體程式可正常執行，即將develop分支合併至main分支（master分支），即成為穩定上線版本。



圖片為示意圖，非實際開發過程

（圖片來源：https://gitbook.tw/chapters/gitflow/why-need-git-flow）

下圖為本專案版本控制協作流程歷程節錄：

（七）開發挑戰與問題解決

問題一：不同城市包含不同數量的行政區，若無動態更新會導致選單與資料不一致。

解決方式：使用城市選單作為主鍵，點選後自動向 API 請求並更新行政區下拉選單，利用 `update_districts` 方法，從 API 動態取得各城市對應行政區，提升資料一致性與彈性。

問題二：顯示多行文字（如天氣描述、地震資訊）時，文字容易因寬度不足而被截斷或顯示不全，尤其在不同螢幕解析度或視窗大小改變時更為明顯。

解決方式：設定 `wraplength` 屬性，讓文字能自動換行。
測試不同解析度與主題顯示，確保中文字在所有模式下皆不會被裁切。

（八）整合測試與問題修正

由組員各自在本機上進行完整的測試，組員使用之作業系統包含Windows及macOS，故測試包含跨平台測試。

測試階段遇到問題反映給原作者，由原作者完成修正後，再進行測試。

六、原始碼重點解說

資料蒐集(RequestApi):

功能與目的：

以類別方式提供串接「氣象資料開放平台」RESTful API，並使用統一格式返回資料，包含HTTP代碼、串接是否成功、返回資料

1. 採用屬性方式將固定資料寫入，提高重複使用率

```
class RequestApi:
    # 要呼叫的API URL位置
    url = 'https://opendata.cwa.gov.tw'
    # API版本
    version = 'v1'
    # API路由
    path = f'api/{version}/rest/datastore'
    # token
    apiAuth ={'Authorization': 'CWA-5B306D7D-A5ED-4639-8532-D1C274899F48'}
    # 定義請求標頭
    headers = {
        'User-Agent': 'python work',
        'Accept': '*/*',
        'Connection': 'keep-alive'
    }
    '''
    城市對應的API代碼陣列中第一個位置是3天預報，第二個位置是1週預報
    '''
    cityApi = {
        '宜蘭縣': ['F-D0047-001', 'F-D0047-003'],
        '桃園市': ['F-D0047-005', 'F-D0047-007'],
        '新竹縣': ['F-D0047-009', 'F-D0047-011'],
        '苗栗縣': ['F-D0047-013', 'F-D0047-015'],
        '彰化縣': ['F-D0047-017', 'F-D0047-019'],
        '南投縣': ['F-D0047-021', 'F-D0047-023'],
        '雲林縣': ['F-D0047-025', 'F-D0047-027'],
        '嘉義縣': ['F-D0047-029', 'F-D0047-031'],
        '屏東縣': ['F-D0047-033', 'F-D0047-035'],
        '臺東縣': ['F-D0047-037', 'F-D0047-039'],
        '花蓮縣': ['F-D0047-041', 'F-D0047-043'],
        '澎湖縣': ['F-D0047-045', 'F-D0047-047'],
```

```
        '基隆市': ['F-D0047-049', 'F-D0047-051'],
        '新竹市': ['F-D0047-053', 'F-D0047-055'],
        '嘉義市': ['F-D0047-057', 'F-D0047-059'],
        '臺北市': ['F-D0047-061', 'F-D0047-063'],
        '高雄市': ['F-D0047-065', 'F-D0047-067'],
        '新北市': ['F-D0047-069', 'F-D0047-071'],
        '臺中市': ['F-D0047-073', 'F-D0047-075'],
        '臺南市': ['F-D0047-077', 'F-D0047-079'],
        '連江縣': ['F-D0047-081', 'F-D0047-083'],
        '金門縣': ['F-D0047-085', 'F-D0047-087']
    }
    '''
    地震API
    '''
    earthquakeApi = [
        'E-A0016-001', # 小區域有感地震
        'E-A0015-001', # 顯著有感地震
    ]

    # 要呼叫的API端點
    apis = [
        'O-A0003-001', # 現在天氣觀測
        'F-C0032-001', # 臺灣各縣市天氣預報資料及國際都市天氣預報
        'A-B0062-001', # 日出日落
        'E-A0014-001', # 海嘯
    ]
```

2. 提供可配置HTTP Request Header屬性，提高可擴充靈活度

```
def setHeader(self, params = {}):
        """
        配置request http header要傳遞的內容
        """
        self.headers = {**self.headers, **params}
        return None
```

3. 封裝發送請求函數，並返回統一的字典格式{status:”狀態”, code:”HTTP CODE”, message:”訊息”, data:”內容”}

```
def send(self, method, api, params):
        """
        proxy 代理方法，對應方法發調用request對應的方法
        """
        response = {"status": False, "code":0, "message":None, "data":
None}
        try:
            if(method.upper() == 'GET'):
                _r = requests.get(api, headers=self.headers,
params={**params, **self.apiAuth})
            elif(method.upper() == 'POST'):
                _r = requests.post(api, headers=self.headers, data=params,
params=self.apiAuth)
            else:
                response["message"] = f"錯誤的方法{method}"
                return response

            response["status"] = _r.status_code == 200
            response["code"] = _r.status_code
            response["message"] = "呼叫成功" if _r.status_code == 200 else "
呼叫失敗"
            response["data"] = _r.json()
            return response

        except requests.exceptions.HTTPError as errh:
            response["message"] = f"HTTP錯誤: {errh}"
            return response
        except requests.exceptions.ConnectionError as errc:
            response["message"] = f"連接錯誤: {errc}"
            return response
        except requests.exceptions.Timeout as errt:
            response["message"] = f"超時錯誤: {errt}"
            return response
        except requests.exceptions.RequestException as err:
            response["message"] = f"無法預期的錯誤: {err}"
            return response
```

4. 提供取得不同城市天氣的快速方法，方便其他程式調用

```
def getWeatherByCity(self, city, days = 3):
        """
        取得指定城市的天氣資料
        Args:
            self
```

```
            city (string): 要取得的程式名稱 ()
            days (int): 要取得的天數 (3或7)
        Returns:
            json
        """
        index = 0 if days == 3 else 1
        #取得縣市與API端點
        api = self.cityApi.get(city)
        if api is None:
            return {"status": False, "message": f"無法取得{city}的API端點"}
        #取得API端點
        if index >= len(api):
            return {"status": False, "message": f"無法取得{city}的{days}天預報
API端點"}
        api = api[index]
        #取得API資料
        return self.get(api)
```

5.提供取得地震資訊的快速方法，方便其他程式調用

```
def getEarthquake(self, modeType = 0):
        """
        取得地震資料
        Args:
            self
            type (int): 0:小區域有感地震 1:顯著有感地震
        """
        #取得API端點
        api = self.earthquakeApi[modeType]
        #取得API資料
        return self.get(api)
```

資料處理(app):

核心目的：

向氣象資料來源查詢某城市與行政區的天氣資訊，解析出指定氣象元素（如溫度、濕度、風速等），並分類為「白天／晚上」，同時可計算體感溫度。

1. **classify_period_by_end_time(start_time, end_time)**

   根據時間區段區分白天/晚上，並回傳日期與時段。

   白天為 6:00 ~18:00，晚上為18:00~隔日6:00。

```
if end_date.hour == 18:
    period = "白天"
elif end_date.hour == 6:
    period = "晚上"
else:
    period = "其他"

return date, period
```

2. **extract_element_value(element_value, allowed_element_type)**

   從單一氣象元素中抽出時間區段與值，整理為統一格式。透過allowed_element_type過濾不必要元素，留下目標氣象資訊。回傳每個目標元素的日期、時間與值。

```
回傳範例：
extracted = [
        {
            "date": 2025-05-18,
            "period": 白天,
            "value":  {'Temperature': '21' }
        }
    ]
"""
```

3. **parse_weather_elements(WeatherElement, allowed_element_type)**

解析氣象元素列表(WeatherElement)。取得這個氣象項目的名稱(ElementName)，並將extract_element_value回傳的資料合併，保留指定的氣象元素(平均溫度、濕度)，整理為標準格式一併回傳給get_weather_by_loction。

```
results = []
for element in WeatherElement:
    element_type = element.get("ElementName")
    element_values = extract_element_value(element, allowed_element_type)
    if element_values:
        results.append(
            {
                "elementType": element_type,
                "elementValue": element_values
            }
        )
return results
```

4. **get_weather_by_loction(city, district, target_elements)**

主功能，透過 **RequestApi** 取得天氣資料，並依需求項目過濾、解析、回傳

response = app.getWeatherByCity(city, 7)

取得目標城市並過濾行政區

```
location_list = locations[0].get("Location", [])
for loc in location_list:
    location_name = loc.get("LocationName")
    if location_name != district:
        continue
```

解析氣象元素，使用parse_weather_elements()來整理格式

```
weather_element = loc.get("WeatherElement", [])
weather_data = parse_weather_elements(weather_element, target_elements)
```

回傳指定城市、行政區以及目標元素值

```
return {
    "city": locations[0].get("LocationsName", []),
    "district": location_name,
    "weather": weather_data
}
```

5. **calc_water_vapor_pressure(RH, temp)**

   計算水氣壓(vapor press)，輸入參數為相對濕度(RH)、溫度(temp)

```
def calc_water_vapor_pressure(RH, temp):
    return RH*0.01*6.105*math.exp((17.27*temp)/(237.7+temp))
```

   公式：**e=RH × 0.01 × 6.105 × exp((17.27 × T) / (237.7 + T))**

   e：水氣壓(hPa)

   RH：相對溼度(%)

   exp ：指數函式

   T：溫度(℃)

6. **calc_apparent_temperature(temp, humd, wind_speed)**

   計算體感溫度，參數為溫度(temp)、濕度(humd)以及風速(wind_speed)，取值到小數點第一位。

```
def calc_apparent_temperature(temp, humd, wind_speed):
    """
    參數
    - temp：攝氏溫度
    - humd：相對溼度(%)
    - wind_speed：風速

    return 體感溫度
    """
    e = calc_water_vapor_pressure(humd, temp)
    return round(1.04 * temp + 0.2*e - 0.65 * wind_speed - 2.7, 1)
```

公式：**AT = 1.04 × T + 0.2 × e − 0.65 × V − 2.7**

AT：體感溫度(℃)

T：溫度(℃)

e：水氣壓(hPa)

V：風速(m/sec)

參考資料：交通部中央氣象署
https://www.cwa.gov.tw/Data/knowledge/announce/service12.pdf

資料呈現(WeatherGUI):

WeatherGUI.py 原始碼重點解說（by 劉彥翎、傅柏凱）

1. 主要類別結構

```
class WeatherApp:
    def __init__(self, root):
        self.root = root
        self.root.title("氣象預報系統")
        screen_width = self.root.winfo_screenwidth()
        screen_height = self.root.winfo_screenheight()
        self.root.geometry(f"{screen_width}x{screen_height}")
```

- 使用 WeatherApp 類別封裝所有功能
- 初始化時自動調整視窗至全螢幕大小

2. 天氣圖示載入機制

```
def load_weather_icons(self):
        """載入天氣圖示（PNG）"""
        icons = {}
        icon_size = (40, 40)
        icon_path = "weather_icons"
```

```
        # 定義天氣狀況和對應的圖示檔案名稱
        weather_icon_mapping = {
            "多雲": "多雲.png",
            "陰": "陰.png",
            "陰天": "陰天.png",
            "陰時多雲": "陰時多雲.png",
            "多雲時陰": "多雲時陰.png",
            "多雲午後短暫雷陣雨": "多雲午後短暫雷陣雨.png",
            "雨": "雨.png",
            "陣雨": "陣雨.png",
            "短暫雨": "短暫雨.png",
            "多雲短暫雨": "多雲短暫雨.png",
            "午後短暫雨": "午後短暫雨.png",
            "多雲午後短暫雨": "多雲午後短暫雨.png",
            "晴": "晴.png",
            "晴時多雲": "晴時多雲.png",
            "晴時陰": "晴時陰.png",
            "晴時多雲時陰": "晴時多雲時陰.png",
            "晴時多雲時陰短暫雨": "晴時多雲時陰短暫雨.png",
            "晴時多雲時陰陣雨": "晴時多雲時陰陣雨.png",
            "晴時多雲時陰陣雨短暫雨": "晴時多雲時陰陣雨短暫雨.png",
            "陰短暫陣雨或雷雨": "陰短暫陣雨或雷雨.png",
        }
```

- 使用字典映射天氣狀況和圖檔
- 自動載入並調整圖檔大小
- 支援多種天氣狀況的圖示顯示

3. 天氣資料處理

```
    def get_weather(self):
        city = self.city_var.get()
        district = self.district_var.get()

        if not district:
            messagebox.showerror("錯誤", "請選擇行政區")
            return

        # 請求所有需要的氣象資料
        target_elements = [
            "平均溫度",
            "體感溫度",
            "相對濕度",
            "天氣現象",
            "降雨機率",
            "風向",
            "蒲福風級",
            "紫外線指數",
            "天氣預報綜合描述"
        ]
```

- 使用正則表達式解析天氣描述
- 智能補全缺失資料
- 格式化數值顯示

4. 資料呈現設計

```
    def format_weather_data(self, data):
        for widget in self.weather_table_frame.winfo_children():
            widget.destroy()

        if not data:
            label = ctk.CTkLabel(self.weather_table_frame, text="無法獲
取天氣資料")
            label.pack()
            return

        # 定義星期對照表
        weekday_map = {
            0: "一", 1: "二", 2: "三", 3: "四", 4: "五", 5: "六", 6: "日
"
        }

        # 整理數據按日期分組
        daily_data = {}
        for weather in data['weather']:
            date = weather['time'].split(' ')[0]
            period = weather['time'].split(' ')[1]
            if date not in daily_data:
                daily_data[date] = {'白天': None, '晚上': None}
            daily_data[date][period] = weather
```

- 使用字典結構組織天氣資料
- 按日期和時段分類資料
- 自動處理日期格式轉換

5. 震度圖標設計

```
    def create_intensity_icon(self, parent, intensity_level):

        # 定義震度等級對應的顏色和文字樣式
        intensity_styles = {
            "0級": {"bg": "#CCCCCC", "fg": "black", "font_size": 16},
            "1級": {"bg": "#CCCCCC", "fg": "black", "font_size": 16},
            "2級": {"bg": "#B266FF", "fg": "black", "font_size": 16},
            "3級": {"bg": "#9999FF", "fg": "black", "font_size": 16},
            "4級": {"bg": "#3333CC", "fg": "white", "font_size": 16},
            "5弱": {"bg": "#CCFFCC", "fg": "white", "font_size": 16},
            "5強": {"bg": "#66CC66", "fg": "white", "font_size": 16},
            "6弱": {"bg": "#FFFF99", "fg": "black", "font_size": 16},
            "6強": {"bg": "#FFAA66", "fg": "white", "font_size": 16},
            "7級": {"bg": "#FF6666", "fg": "white", "font_size": 16},
        }
```

- 根據地震震度等級創建視覺化的圓形圖標：
- 使用不同顏色表示不同震度等級（從0級到7級）
- 圖標採用圓形設計，內部顯示震度等級文字

6. 地震資料處理

```
    def format_earthquake_data(self, data):
        # 清除現有表格
        for widget in self.earthquake_table_frame.winfo_children():
            widget.destroy()

        if not data or not data.get('records', {}).get('Earthquake'):
            label = ctk.CTkLabel(self.earthquake_table_frame, text="目
前查無地震資料")
            label.pack()
            return
```

- 建立標題行和數據行的表格框架
- 當沒有地震數據時，顯示「目前查無地震資料」的提示
- 從API返回的數據中提取相關的地震信息
- 根據窗口大小調整表格內容的換行寬度

7. 錯誤處理機制

```
    def get_earthquake(self):
        # 根據下拉選單獲取地震類型值
        earthquake_type_text = self.earthquake_type_var.get()
        mode_type = 0 if earthquake_type_text == "小區域有感地震" else 1

        try:
            result = self.api.getEarthquake(mode_type)
            if result['status']:
                self.format_earthquake_data(result['data'])
            else:
                messagebox.showerror("錯誤", result['message'])
        except TypeError as e:
            print(f"TypeError 發生: {str(e)}")
            messagebox.showerror("錯誤", f"處理資料時發生錯誤: {str(e)}")
        except Exception as e:
            print(f"錯誤發生: {str(e)}")
            messagebox.showerror("錯誤", f"發生未預期的錯誤: {str(e)}")
```

- 完整的異常處理流程
- 使用者友善的錯誤提示
- 錯誤日誌記錄

8. 主題切換功能

```
    def change_appearance_mode(self, new_mode):
        ctk.set_appearance_mode(new_mode)
        self.root.update_idletasks()
        self.root.update()
```

- 支援系統/淺色/深色主題
- 即時更新介面外觀
- 保持使用者設定

9. 打包設定

```
    # 匯出命令(無控制台)
    # pyinstaller --onefile --add-data "weather_icons;weather_icons"
--name "氣象預報系統 Windows/iOS v0.1.0" --noconsole <your_script.py>

    # 匯出命令(有控制台)
    # pyinstaller --onefile --add-data "weather_icons;weather_icons"
--name "氣象預報系統 Windows/iOS v0.1.0 debug" <your_script.py>
```

- 提供兩種打包模式
- 支援跨平台打包
- 可自訂版本號和檔名