Scheduler 排班系統

一、專案說明

本專案實作一個簡易但具可維護性的 **排班（Scheduling）系統**，目標是在滿足每位工作人員可上班日（Availability）的前提下，為每一天指派一位合適的工作人員。

此專案著重於：

- 演算法正確性（Constraint Satisfaction）
- 程式結構清楚、可重構
- 具備單元測試，確保行為穩定
- 使用 Python 型別提示提升可讀性與安全性

二、問題定義

輸入

- 多位 `Worker`
- 每位 Worker 對應一組可上班的 `Day`

範例：

```
availability = {
    "Alice": {"Mon", "Tue"},
    "Bob": {"Tue"}
}
```

輸出

- 若存在合法排班：

```
{
    "Mon": "Alice",
    "Tue": "Bob"
}
```

- 若不存在合法排班：

```
None
```

三、專案結構

```
project/
├── scheduler/
│   ├── __init__.py
│   ├── core.py           # 排班核心演算法（DFS）
│   ├── emergency.py.     # 突發狀況情境調整可上班對照表
│   └── types.py          # 型別定義
│
├── tests/
│   ├── __init__.py
│   ├── test_basic.py     # 單元測試（pytest）
│   ├── test_emergency.py # 單元測試（pytest）
│   └── test_no_result.py # 單元測試（pytest）
│
├── README.md
└── main.py
```

四、設計說明

1. 型別設計（types.py）

```
Day = str
Worker = str
Availability = Dict[Worker, Set[Day]]
Schedule = Dict[Day, Worker]
```

用途：

- 清楚表達資料結構語意
- 提升 IDE 補齊與重構安全性

---

2. 核心演算法（core.py）

- 使用 **DFS**
- 逐日指派 Worker
- 若遇到無解則回溯

五、時間與空間複雜度分析

時間複雜度

- 最壞情況：

```
O(W^D)
```

其中：

- `D`：天數（Day 數量）
- `W`：平均每一天可選的 Worker 數

空間複雜度

- 遞迴深度：`O(D)`
- 排班結果與可用性資料：`O(W * D)`

六、測試說明

測試工具

- 使用 `pytest`

測試目標

- 驗證基本可行排班
- 驗證無解情境
- 驗證最小輸入（單一 worker / day）

執行測試

```
pytest
```

使用方式

```
python main.py
```