

# 期末專案

## ——作業時間排程器——

台北大學 711483120 資工研一 簡瀚佑

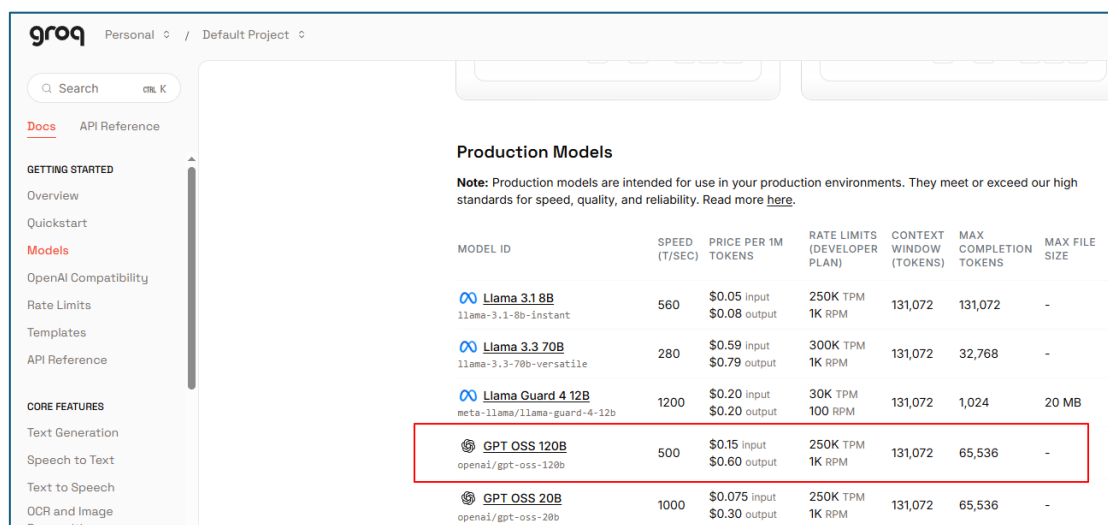
## === 檔案連結 ===

Youtube 影片連結：<https://youtu.be/VfkdVOY82lw>

GitHub 實作與程式碼連結：<https://github.com/JY-soldier/Task-Scheduler>

## === 專案使用模型 ===

本次專案使用 Groq 的免費模型 "openai/gpt-oss-120b"。



MODEL ID	SPEED (T/SEC)	PRICE PER 1M TOKENS	RATE LIMITS (DEVELOPER PLAN)	CONTEXT WINDOW (TOKENS)	MAX COMPLETION TOKENS	MAX FILE SIZE
<a href="#">Llama 3.1 8B</a> llama-3.1-8b-instant	560	\$0.05 input \$0.08 output	250K TPM 1K RPM	131,072	131,072	-
<a href="#">Llama 3.3 70B</a> llama-3.3-70b-versatile	280	\$0.59 input \$0.79 output	300K TPM 1K RPM	131,072	32,768	-
<a href="#">Llama Guard 4 12B</a> meta-llama/llama-guard-4-12b	1200	\$0.20 input \$0.20 output	30K TPM 100 RPM	131,072	1,024	20 MB
<a href="#">GPT OSS 120B</a> openai/gpt-oss-120b	500	\$0.15 input \$0.60 output	250K TPM 1K RPM	131,072	65,536	-
<a href="#">GPT OSS 20B</a> openai/gpt-oss-20b	1000	\$0.075 input \$0.30 output	250K TPM 1K RPM	131,072	65,536	-

```
# 想用哪個 Groq 模型改這行即可
#GROQ_MODEL_NAME = "llama-3.1-8b-instant" # Groq LLM 使用的模型名稱
GROQ_MODEL_NAME = "openai/gpt-oss-120b"

# 呼叫 Groq 的 chat.completions API 產生回應
completion = client.chat.completions.create(
    model=GROQ_MODEL_NAME, # 指定模型名稱
    messages=[
        {"role": "system", "content": system_prompt2}, # system 訊息，定義任務與格式
        {"role": "user", "content": raw_text}, # user 訊息，放使用者輸入的原始文字
    ],
    temperature=0.2, # 溫度設比較低，結果更穩定
)
```

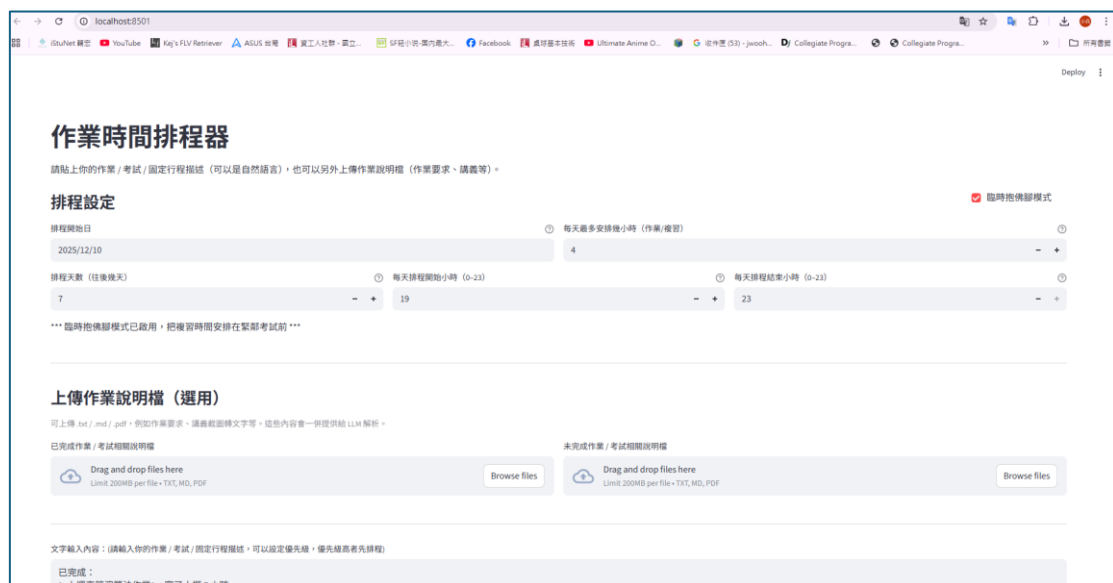
## === 專案目標 ===

製作一個可以幫助使用者安排作業/報告準備/考試複習等行程的排程器。

使用者可以輸入已完成、未完成(待安排)和固定(不能更動)的行程資訊，由 LLM 估算時間並透過演算法安排生成行事曆。

## === 排程器下載說明 ===

1. 前往 GitHub 下載整個 launcher 資料夾(也可以直接下載整個專案.zip)。
2. 前往 <https://console.groq.com/keys> 申請並記下 API key。
3. 打開 Windows PowerShell 將 API key 設定成環境變數。
  - 輸入：`setx GROQ_API_KEY "你的 Groq 金鑰"`
  - 重新開啟 Windows PowerShell 並輸入：`echo $Env:GROQ_API_KEY`若回傳內容是你的 key 值，表示設定成功。
4. 打開第一步下載的 launcher 資料夾並雙擊啟動 launcher.exe 打開網頁
5. 開始使用作業時間排程器 !!



## === 排程器使用說明 ===

排程設定：設定排程條件。



- 臨時抱佛腳模式：將考試複習行程安排在緊鄰考試的時間段複習。
- 排程開始日：從哪一天開始排 "未完成行程"。
- 每天最多安排幾小時：每天安排的 "未完成行程" 上限。
- 排程天數：從排程開始日開始計算。
- 每天排程開始/結束小時：每天安排的時間段。

說明檔上傳：可以上傳相關文字檔案給 LLM，協助估算所需時間。

上傳作業說明檔 (選用)

可上傳 .txt, .md, .pdf, 例如作業要求、讀書範圍轉文字等。這些內容會一併提供給 LLM 解析。

已完成作業 / 考試相關說明檔

Drag and drop files here  
Limit 200MB per file • TXT, MD, PDF

Browse files

數位影像處理hw1.pdf 1.4MB

×

高等演算法\_Exercise\_1.pdf 181.5KB

×

未完成作業 / 考試相關說明檔

Drag and drop files here  
Limit 200MB per file • TXT, MD, PDF

Browse files

高等演算法\_Exercise2\_.pdf 114.9KB

×

文字輸入：輸入已完成、未完成、固定行程描述，包含日期、耗時、優先級等等。

文字輸入內容：(請輸入你的作業 / 考試 / 固定行程描述，可以設定優先級，優先級高者先排程)

已完成：

1. 上週高等演算法作業1，寫了大概 7 小時。

2. 上週數位影像處理作業1，寫了大概 12 小時。

3. 上次線代小考前複習 3 小時。

4. 上次計算機圖學期中考複習 8 小時。

未完成：

1. 12/09 早上9點要考線性代數小考1。

2. 12/11 晚上11:59要交高等演算法作業2。

生成排程

- 生成排程：以上都設定好就可以按下按鈕開始排程。

逾期任務：截止時間已經超過當前時間，不計入排程。

逾期任務

以下任務的截止時間已經過去，因此不會被排入未來的時程中：

	標題	科目	截止時間	預估時間(分鐘)
0	線性代數小考 1	線性代數	2025-12-09 09:00	120

無法完全安排的任務：截止時間前無法排完的任務，"已排時間"大於 0 表示該行程仍有部分排入行事曆。

無法完全安排的任務

以下任務在目前設定的排程天數 / 每日作業上限 / 每天排程時段內，無法完全排完預估所需時間：

	標題	科目	截止時間	預估時間(分鐘)	已排時間(分鐘)	尚未排入(分鐘)
0	數位影像處理作業 2	數位影像處理	2025-12-10 23:59		720	600
1	高等網路期末考	高等網路	2025-12-18 13:00		180	120

排程列表：排入行事曆的行程，按照時間順序排列。

排程列表

	標題	開始	結束	種類
0	高等演算法作業 3	2025-12-10 12:00	2025-12-10 13:00	作業/複習
1	高等演算法作業 3	2025-12-10 13:00	2025-12-10 14:00	作業/複習
2	高等演算法作業 3	2025-12-10 14:00	2025-12-10 15:00	作業/複習
3	高等演算法作業 3	2025-12-10 15:00	2025-12-10 16:00	作業/複習
4	高等演算法作業 3	2025-12-10 16:00	2025-12-10 17:00	作業/複習
5	高等演算法作業 3	2025-12-10 17:00	2025-12-10 18:00	作業/複習
6	高等演算法作業 3	2025-12-10 18:00	2025-12-10 19:00	作業/複習

ICS 檔下載：下載可以匯入日曆的 ICS 檔。由於 ICS 檔不支援顏色設定，故提供固定行程和未完成行程分開的版本，供使用者分開匯入不同日曆產生不同顏色，方便查看。

匯入 Google Calendar

(或者其他支援.ics檔的日曆)

單一日曆下載

【全部行程】.ics

分成兩個日曆下載

方便分開匯入不同日曆，以顯示不同顏色

【固定行程】.ics

【作業/考試複習】.ics

我的日曆

☒ JY小兵

☒ Tasks

☒ 生日

其他日曆

+

☒ 台灣的節慶假日

Step 1

其他日曆

☒ 台灣的節慶假日

Step 2

訂閱日曆

建立新日曆

瀏覽感興趣的日曆

加入日曆網址

匯入

匯入 Step 3 選擇匯入檔案和日曆

fixed\_events.ics

新增至日曆

固定行程

你可以使用 iCalendar (ICS/VCS) 或 CSV 格式的檔案匯入活動資訊。

匯入 Step 4

- 單一日曆：

9 (二十)	<p>10 (廿一)</p> <ul style="list-style-type: none"> <li>下午12點 高等演算法作業 3</li> <li>下午1點 高等演算法作業 3</li> <li>下午2點 高等演算法作業 3</li> <li>下午3點 高等演算法作業 3</li> </ul> <p>還有 4 個</p>	<p>11 (廿二)</p> <ul style="list-style-type: none"> <li>下午12點 高等演算法作業 2</li> <li>下午1點 高等演算法作業 2</li> <li>下午2點 高等演算法作業 2</li> <li>下午3點 高等演算法作業 2</li> </ul> <p>還有 4 個</p>	<p>12 (廿三)</p> <ul style="list-style-type: none"> <li>下午12點 數位影像處理作業 2</li> <li>下午1點 數位影像處理作業 2</li> <li>下午2點 數位影像處理作業 2</li> <li>下午3點 數位影像處理作業 2</li> </ul> <p>還有 4 個</p>	<p>13 (廿四)</p> <ul style="list-style-type: none"> <li>下午6點 和家人吃飯</li> </ul>
<p>16 (廿七)</p> <ul style="list-style-type: none"> <li>下午7點 補習</li> <li>下午9點 線性代數小考 2</li> <li>下午10點 線性代數小考 2</li> </ul>	<p>17 (廿八)</p> <ul style="list-style-type: none"> <li>下午9點 高等網路期末考</li> <li>下午10點 高等網路期末考</li> </ul>	<p>18 (廿九)</p>	<p>19 (三十)</p> <ul style="list-style-type: none"> <li>下午8點 高等演算法期末考</li> <li>下午9點 高等演算法期末考</li> <li>下午10點 高等演算法期末考</li> </ul>	<p>20 (十一月)</p>

- 分開日曆：

9 (二十)	<b>10 (廿一)</b> <ul style="list-style-type: none"> <li>● 下午12點 高等演算法作業 3</li> <li>● 下午1點 高等演算法作業 3</li> <li>● 下午2點 高等演算法作業 3</li> <li>● 下午3點 高等演算法作業 3</li> </ul> 還有 4 個	<b>11 (廿二)</b> <ul style="list-style-type: none"> <li>● 下午12點 高等演算法作業 2</li> <li>● 下午1點 高等演算法作業 2</li> <li>● 下午2點 高等演算法作業 2</li> <li>● 下午3點 高等演算法作業 2</li> </ul> 還有 4 個	<b>12 (廿三)</b> <ul style="list-style-type: none"> <li>● 下午12點 數位影像處理作業 2</li> <li>● 下午1點 數位影像處理作業 2</li> <li>● 下午2點 數位影像處理作業 2</li> <li>● 下午3點 數位影像處理作業 2</li> </ul> 還有 4 個	<b>13 (廿四)</b> <ul style="list-style-type: none"> <li>● 下午6點 和家人吃飯</li> </ul>
<b>16 (廿七)</b> <ul style="list-style-type: none"> <li>● 下午7點 補習</li> <li>● 下午9點 線性代數小考 2</li> <li>● 下午10點 線性代數小考 2</li> </ul>	<b>17 (廿八)</b> <ul style="list-style-type: none"> <li>● 下午9點 高等網路期末考</li> <li>● 下午10點 高等網路期末考</li> </ul>	<b>18 (廿九)</b>	<b>19 (三十)</b> <ul style="list-style-type: none"> <li>● 下午8點 高等演算法期末考</li> <li>● 下午9點 高等演算法期末考</li> <li>● 下午10點 高等演算法期末考</li> </ul>	<b>20 (十一月)</b>

## === 排程器實作流程 ===

首先使用 Chat GPT 生成整個專案的基礎框架程式碼。

我想使用生成式AI製作一個作業時間排程應用，以下是條件:  
"1. 可以輸入的內容過去完成的作業(包含題目、結果、消耗時間)，也可以是複習考試內容)、尚未完成的作業(包含題目、截止日期，也可以是複習考試內容)、已經有安排不能更改的日期、時間(包含上課、聚餐、考試、回老家等等使用者認為已經有事的時段)"  
"2. 輸出一張日程安排表(內容必須包含所有輸入的未完成作業和育安排不能更改事務，以不同顏色標示)"  
"3. 生成式AI根據使用者輸入內容來製作時間安排表"  
"4. 輸入內容可以是條列或者文字輸入或二套智慧"  
請幫我構想一個實作的框架，輸入給生成式AI到輸出一段過程要能夠做調整(不一定是單一模型輸出結果)

已思考 15s

先直接給你一個「模組化 pipeline」的整體架構，從「輸入 → 生成式 AI → 排程結果」整個路都可以拆開調整，不會綁死在單一模型。

一、整體概念：把問題拆成 4 個階段

你要的東西可以拆成這四段（每一段都可以換模型 / 換演算法）：

1. 資料輸入與前處理

- 接收文字、檔案、行事曆等各種輸入。

2. 語意理解與結構化（LLM 模組 A）

- 把「作業描述、已排行程、轉成統一的結構化資料（JSON）」。

3. 排程規劃（演算法 + LLM 模組 B）

- 根據實際時程、截止日、可手調時間、做排程。

一個最小 demo 架構

那我們直接做一個「可以跑起來」的最小版本架構。全部用 Python + Streamlit + OpenAI API。  
下面我用「專業資訊與結構 + 條列式骨幹」的方式畫給你看看。

一、專案結構 { minimal demo }

```
test
```

複製程式碼

```
study-scheduler-demo/  
├─ app.py           # Streamlit 入口、UI + 可再修改處  
├─ llm_parser.py    # LLM：把文字輸入 → 結構化作業 & 固定行程  
├─ scheduler.py     # 簡單 greedy 排程  
├─ models.py        # Pydantic 資料結構定義  
└─ config.py       # API key 等設定
```

流程圖概念：

```
test
```

複製程式碼

```
graph TD  
    Input[使用輸入文字] --> Parse[app.py -> llm_parser.parse_input_with_llm()]  
    Parse --> Scheduler[scheduler.build_schedule()]  
    Scheduler --> Output[排程結果 JSON]
```

三、config.py：設定

```
python
```

複製程式碼

```
# config.py  
  
import os  
  
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY") # export 後跟提示  
DEFAULT_STUDY_START_HOUR = 19 # 每天 19:00-23:00 都可調整時間  
DEFAULT_STUDY_END_HOUR = 23  
BLOCK_MINUTES = 60 # 每格 1 小時
```

四、llm\_parser.py：用 LLM 把文字 → JSON

這裡示範一個最小 prompt，實際你可以再加強。

```
python
```

複製程式碼

```
# llm_parser.py  
  
from openai import OpenAI  
from datetime import datetime  
from models import ParsedInput  
from config import OPENAI_API_KEY  
  
client = OpenAI(api_key=OPENAI_API_KEY)  
  
  
SYSTEM_PROMPT = """  
你是一個能幫助學生行程的助理。  
使用者會給出自然語言描述的作業、考試和已經排好行程。  
請你輸出 JSON，格式為：  

```

接著讓 Chat GPT 調整並補上一些額外內容(說明檔上傳、輸出 ics 檔、把程式包裝成 exe 檔)。

<p>增加輸入檔案功能，包括已完成和未完成作業的說明檔(作業要求之緒論)</p> <p>已思考 1m 58s</p> <p>可以，加「上傳作業說明檔」很適合你這個應用 🧐</p> <p>我們只需要在 <code>app.py</code> 加上 <code>#file_uploader</code>，然後把檔案內容一起丟給 LLM 做解析就好。</p> <p>下面給你一份完整改好的 <code>app.py</code>，你可以直接整個檔案現在的 <code>app.py</code> 使用。</p> <p>(其它檔案像 <code>llm_parser.py</code>、<code>scheduler.py</code> 都不動)</p> <p>我有幫你做這些件事：</p> <ul style="list-style-type: none"><li>新增兩個上蓋：</li><li><input checked="" type="checkbox"/> 已完成作業相關說明檔 (與你作業要求、講義)</li><li><input checked="" type="checkbox"/> 未完成作業 / 考試相關說明檔</li><li>支援檔案格式：<code>.txt</code>、<code>.md</code>、<code>.pdf</code></li><li>提出內容後，用清楚的標記加在原本文字輸入後，一起送給 <code>parse_input_with_llm()</code></li></ul>	<p>能把排程結果放到 google calendar 上嗎?</p> <p>已思考 13s</p> <p>可以，而且用你現在的架構 (Streamlit + Python) 很好處理：</p> <p>把排程結果轉成 <code>ics</code> 檔 → 讓使用者下載 → 匯入 Google Calendar 就行了，不一定需要直接串 Google API (那會牽涉 OAuth，會比較麻煩)。</p> <p>我先給你「能直接用在你專案裡」的版本：</p> <ol style="list-style-type: none"><li>新增一個 <code>ics_export.py</code></li><li>修改 <code>app.py</code>，在排程後多一個下載 <code>ics</code> 的按鈕</li><li>最後教你怎麼在 Google Calendar 匯入</li></ol>	<p>我的意思是把所有程式包成一個檔案，例如 <code>exe</code> 檔直接點開，類似 <code>visual studio</code> 那種產生一個 <code>exe</code> 檔，我使用的是 <code>pycharm</code></p> <p>已思考 19s</p> <p>懂你的意思了，你是想要：</p> <ul style="list-style-type: none"><li>🧐 是 <code>Visual Studio</code> 那種匯出一個 <code>xxx.exe</code>，在 Windows 上雙擊就能把 <code>Streamlit</code> 網頁跑起來，而不是因為在 <code>PyCharm</code> 裡按 <code>run</code> 對吧。</li></ul> <p>因為你現在是 <code>Streamlit</code> 網頁，所以就該做成 <code>exe</code>，本質上還是：</p> <pre>exe 啟動 → 開啟一個本機的 Streamlit server → 自動在瀏覽器開 http://localhost:8501 ▶</pre> <p>下面給你兩種做法：</p>
<p>需要額外安裝 (用來講 PDF)</p> <pre>bash pip install PyPDF2</pre> <p>改好的 <code>app.py</code> (整份覆蓋即可)</p>	<p>1. 新增 <code>ics_export.py</code>：把排程轉成 <code>ics</code></p> <p>在你的專案資料夾 <code>HsScheduler/</code> 底下加一個檔案 <code>ics_export.py</code>：</p> <pre>python from datetime import datetime from typing import List from models import ScheduledBlock</pre>	<p>做法一 (推薦)：用 <code>PyInstaller</code> + <code>launcher.py</code></p> <p>讓原本的 <code>app.py</code> 完全不修改，只多一個「啟動器」，用 <code>PyInstaller</code> 打包。</p> <p>第 0 步：確認你現在的結構</p> <p>假如你的專案是這樣：</p> <pre>test ├── D:\ASUS\Desktop\Program\Python\HsScheduler\ │   ├── app.py          + 你目前的 Streamlit 主程式 │   ├── main.py         + view │   └── ...</pre> <p>第 1 步：在同一個資料夾建立 <code>launcher.py</code></p>

最後手動修改程式碼調整並添加額外功能，主要修改網頁 UI 設計、排程條件設定、Prompt 提示內容與流程、排程演算法。

```
# ---- 排程設定 ----
col1, col2, col3 = st.columns([0, 3, 1])

with col1:
    st.subheader("排程設定") # 小標題: 排程設定區塊

with col3: # 右欄: 把考試排班時間安排在考試前
    exam_or_not: bool = st.checkbox("臨時抱佛腳模式", value=True)

col_cfg1, col_cfg2 = st.columns(2) # 建立兩欄, 左邊設定開始日, 右邊設定每天最多小時

with col_cfg1: # 左欄: 排程開始日設定
    today = datetime.today().date() # 取得今天日期 (不含時間)
    start_date: date = st.date_input(
        "排程開始日", # 輸出標題
        value=today, # 預設值為今天
        min_value=today, # 不允許選今天以前
        help="從這一天開始往後排作業/複習時段 (不可早於今天)", # 提示文字
    )

with col_cfg2: # 右欄: 每天最多安排幾小時作業/複習
    max_hours_per_day: int = st.number_input(
        "每天最多安排幾小時 (作業/複習)", # 輸出標題
        min_value=1, # 最小值 1 小時
        max_value=10, # 最大值 10 小時
        value=4, # 預設值 4 小時
        step=1, # 每次調整步幅為 1
        help="只計算作業/複習時間, 不包含閒置時間", # 提示文字
    )
```

```
# 定義 GPT 的 system prompt, 根據說明檔判定難易度
system_prompt1 = """
你是一個幫忙估計學生作業、報告和考試難易程度的助理。
你主要為學生判斷作業、報告和考試的難易程度。
使用者會給出自然語言說明他的作業、報告和考試。
請根據"已完成作業 / 考試說明檔 (past_tasks)"和"未完成作業 / 考試說明檔 (todos)"輸出 JSON 格式:
"""

"past_tasks": [
    {
        "title": "string",
        "difficulty": 1
    },
    ...
]

"todos": [
    {
        "title": "string",
        "difficulty": 1
    },
    ...
]

重要規則:
1. "title" 根據說明檔的備名設定。
2. "difficulty" 必須是正整數, 難度越大則 difficulty 越大, 與完成作業所需時間成正比。
3. 第一個判斷說明檔的 "difficulty" 必須設定為 10, 其他說明檔的 difficulty 以第一個說明檔的難易度為基準來判斷。
4. 只輸出合法 JSON, 不要再加註解或多餘文字。
"""
```

```
# 這兒用排班時間給 GPT 說明排班邏輯
for slot_start in iter_study_slots(
    start_date, # 排班開始時間 (含時日) + 00:00
    days, # 排班幾天
    study_start_hour, # 每天排班開始小時
    study_end_hour, # 每天排班結束小時
):
    if slot_start >= todo.deadline:
        # 已經超過排班的截止時間, 不再安排排班
        break

    if remaining <= 0:
        break # 排班時間已用盡, 不再安排排班

    # 計算排班時間
    if exam_or_not == True and todo.exam_or_not == True:
        num_of_days = todo.deadline.day - slot_start.day - 1
        num_of_minutes = num_of_days * max_minutes_per_day
        num_of_minutes += max((study_end_hour - slot_start.hour), 0) * 60
        num_of_minutes += max((todo.deadline.hour - study_start_hour), 0) * 60
        for slot in iter_study_slots(slot_start, num_of_days+1, study_start_hour, study_end_hour, ):
            if not is_free(slot, slot.timestamp(minutes=BLOCK_MINUTES)):
                num_of_minutes -= BLOCK_MINUTES
            if num_of_minutes >= todo.estimated_time.minutes: continue

    slot_end = slot_start + timedelta(minutes=BLOCK_MINUTES) # 計算排班時間

    # 加上排班時間 (只算作業/複習, 不算閒置時間)
    if max_minutes_per_day is not None:
        day_key = slot_start.date() # 計算排班時間第一天
        used = day_to_minutes_per_day(day_key, 0) # 計算排班時間第一天
        if used + BLOCK_MINUTES > max_minutes_per_day:
            # 這一天的排班時間已經用盡, 換下一天
            continue # 不在排班時間, 直接換下一天時間

    if is_free(slot_start, slot_end): # 排班時間沒有被其他 block 占用
        blocks.append(
            ScheduleBlock(
```

=== 排程器運作邏輯 ===

使用者設定完排程條件並輸入資料後，對輸入資料進行以下處理：

1. 將輸入檔案整理成字串，和輸入文字分開輸入 LLM。

```
# ---- 文字輸入 ----
# 主要文字輸入框, 使用者輸入自然語言描述
raw_text = st.text_area("文字輸入內容: (請輸入你的作業 / 考試 / 制定排班時間, 可以設定優先級, 優先級越高先排)", value=default_example, height=200)

# 當按下「生成排程」按鈕時, 更新 session_state
if st.button("生成排程"): # 建立一個按鈕, 按下時執行以下區塊
    # 把檔案內容讀出來, 則清楚是什麼檔案
    extra_sections = [] # 用來累積上傳檔案文字內容的 List

    if done_files: # 若有上傳已完成作業說明檔
        for f in done_files: # 逐一處理每個檔案
            content = read_uploaded_file(f) # 讀取並轉為文字
            # 加上標題說明來源是複習檔案
            extra_sections.append(
                f"[已完成作業 / 考試說明檔: {f.name}]\n{content}"
            )

    if todo_files: # 若有上傳未完成作業 / 考試說明檔
        for f in todo_files: # 逐一處理每個檔案
            content = read_uploaded_file(f) # 讀取並轉為文字
            # 加上標題說明來源是複習檔案
            extra_sections.append(
                f"[未完成作業 / 考試說明檔: {f.name}]\n{content}"
            )

    # 合併成適合 LLM 的完整文字
    assignment_file = "" # 先從文字輸入內容開始
    if extra_sections: # 若有上傳上檔案
        assignment_file += ( # 在原文字後加上區隔和各個檔案內容
            "\n\n=== 以下為上傳的作業說明檔內容 ===\n\n"
            + "\n\n---\n\n".join(extra_sections)
        )

    # 呼叫 LLM 解析使用者輸入
    with st.spinner("解析輸入內容中 (呼叫 GPT LLM ...)"):
        parsed = parse_input_with_llm(raw_text, assignment_file, start_date, schedule_days) # 使用 llm_parser 將文字解析成結構化的物件
        if not parsed: st.error("A 發生錯誤: A\n\n可能原因如下:\n1. GPT_API_KEY 未設定, 導致 LLM 失敗\n2. 輸入檔案過大或文字過多, 超出 LLM 請求
        else:
            st.session_state["parsed"] = parsed # 存進 session_state 以便後續使用
```

2. 先將轉成字串的說明檔輸入 LLM，讓它判斷難易度，並按照 JSON 格式輸出。

如果 LLM 輸出要求以外的內容，則透過程式檢查並篩除。

最後將輸出結果和使用者輸入文字串接在一起。

```
# 定義 GPT 的 system prompt, 根據說明檔判定難易度
system_prompt1 = """
你是一個幫忙估計學生作業、報告和考試難易程度的助理。
你主要為學生判斷作業、報告和考試的難易程度。
使用者會給出自然語言說明他的作業、報告和考試。
請根據"已完成作業 / 考試說明檔 (past_tasks)"和"未完成作業 / 考試說明檔 (todos)"輸出 JSON 格式:
"""

"past_tasks": [
    {
        "title": "string",
        "difficulty": 1
    },
    ...
]

"todos": [
    {
        "title": "string",
        "difficulty": 1
    },
    ...
]

重要規則:
1. "title" 根據說明檔的備名設定。
2. "difficulty" 必須是正整數, 難度越大則 difficulty 越大, 與完成作業所需時間成正比。
3. 第一個判斷說明檔的 "difficulty" 必須設定為 10, 其他說明檔的 difficulty 以第一個說明檔的難易度為基準來判斷。
4. 只輸出合法 JSON, 不要再加註解或多餘文字。
"""
```

```
if assignment_file:
    # 呼叫 GPT 的 chat.completions API 產生回應
    completion = client.chat.completions.create(
        model=GPT_MODEL_NAME, # 指定模型名稱
        messages=[
            {
                "role": "system", "content": system_prompt1, # system 訊息: 定義任務與格式
            },
            {
                "role": "user", "content": assignment_file, # user 訊息: 使用者輸入的原始文字
            },
        ],
        temperature=0.2, # 溫度設比較低, 結果更穩定
    )

    # 取用 LLM 回應的主要文字內容
    content = completion.choices[0].message.content
    raw = content.strip() # 去掉前後空白

    # 如果有 '' json ... '' code block, 先去掉外層
    if raw.startswith('```'): # 檢查是否以 '```' 開頭 (包含 '```json')
        # 去掉開頭 '```' 或 '```json' 這一行
        raw = re.sub(pattern=r"```([a-zA-Z0-9]+)```", repl='', raw)
        # 去掉最後的 '```' 以及前面的空白換行
        raw = re.sub(pattern=r"```$", repl='', raw).strip()

    # 只取第一個 { 到最後一個 } 之間的內容
    first = raw.find('{') # 找到第一個 '{' 的位置
    last = raw.rfind('}') # 找到最後一個 '}' 的位置
    if first == -1 or last == -1: # 找不到大括號
        print("A GPT 回應內容找不到大括號, 原始內容:", content) # 印出原始內容
        return None # 因為 None, 讓上層改用 demo
```

```
json_str = raw[first: last + 1] # 取出完整 JSON 字串範圍
#print("=====")
#print(json_str)

# 丟給 Pydantic 解析
difficulty_result = DifficultyResult.model_validate_json(json_str) # 使用 Pydantic 的 JSON 解析功能
print("===== 難度解析結果 =====")
print(difficulty_result)
print("=====")

if difficulty_result.past_tasks or difficulty_result.todos:
    lines = []
    lines.append("以下是作業 / 考試難易度評估:")
    for t in difficulty_result.past_tasks:
        for t in difficulty_result.todos:
            lines.append(f"[未完成] {t.title} 難度={t.difficulty}")
    raw_text += "\n\n" + "\n".join(lines) + "\n\n"
```

## (1) LLM 輸入內容：

已完成作業 / 考試相關說明檔	未完成作業 / 考試相關說明檔
<div>Drag and drop files here Limit 200MB per file • TXT, MD, PDF</div> <div>Browse files</div>	<div>Drag and drop files here Limit 200MB per file • TXT, MD, PDF</div> <div>Browse files</div>
<div>數位影像處理hw1.pdf 1.4MB</div> <div>高等演算法_Exercise_1.pdf 181.5KB</div>	<div>高等演算法_Exercise2_.pdf 114.0KB</div>

## (2) LLM 輸出內容：

```
===== 難度解析結果 =====
past_tasks=[DifficultyPastTask(title='高等演算法_Exercise_1.pdf', difficulty=10), DifficultyPastTask(title='數位影像處理hw1.pdf', difficulty=9)] todos=[DifficultyTodo(title='高等演算法_Exercise2_.pdf', difficulty=12)]
=====
```

3. 將輸出結果和使用者輸入文字串接後再次輸入 LLM，讓它估算未完成行程的消耗時間。
- 輸出同樣按照 JSON 格式，並用程式檢查篩除要求以外的輸出。

```
# 定義 LLM 的 system prompt，說明要解析的 JSON 格式與欄位
system_prompt2 = """
你是一個能根據學生行程與時間，
從作業與考試 (schedule_start:YY-MM-DD 與 schedule_end:YY-MM-DD) 這段期間安排行程。
使用者會給出自定義說明檔的作業，你需給出建議的行程。
請輸出 JSON 格式為：
{
  "past_tasks": [
    {
      "title": "string",
      "subject": "string or null",
      "time_spent_minutes": 90,
      "difficulty": 1
    }
  ],
  "todos": [
    {
      "title": "string",
      "subject": "string or null",
      "deadline": "2025-12-07T23:59",
      "estimated_time_minutes": 120,
      "difficulty": 1,
      "priority": 1,
      "exam_or_not": false
    }
  ],
  "fixed_events": [
    {
      "title": "string",
      "start": "2025-12-07T18:00",
      "end": "2025-12-07T20:00"
    }
  ]
}
```

重要規則：

- "fixed\_events" 必須是隨機的「實際的每一次行程」。
- 如果文字中出現「每週三晚上 6 點到 9 點補習」這樣表述，你要在 {schedule\_start:YY-MM-DD} ~ {schedule\_end:YY-MM-DD} 這段期間內，找出所有符合「週三」的日期，並為每一天建立一個固定行程。
- 例如，如果行程時間包含 2025-12-10, 2025-12-17, 2025-12-24 這三個週三，則 fixed\_events 要包含三筆「補習」事件，時間分別為：  
2025-12-10 19:00-21:00  
2025-12-17 19:00-21:00  
2025-12-24 19:00-21:00
- 如果文字中只有「12/11 晚上 7 點到 8 點和家人吃飯」這種單次活動，就只建立一筆 fixed\_event。
- 所有日期時間一律使用 ISO 8601 格式，例如 "2025-12-07T23:59"。
- "difficulty" 的設定只能抄寫，根據輸入文字後半段的 JSON 內容抄寫，沒寫的不需自己猜。
- 如果文字中出現「作業 / 考試難易度評估」這樣表述，則根據文字後半段 JSON 格式的內容中 "title" (特別注意：「演算法作業1」和「演算法作業2」兩個是不同事件，不能共用或混淆彼此的 difficulty) 和「已完成(past\_tasks)/未開始(todos)」填寫對應 "difficulty"，如果輸入 JSON 格式中需要填寫 "difficulty"，則是輸入文字後半段 JSON 中並未寫明，則一律設定成 -1。
- "estimated\_time\_minutes" 絕對不能是 0 或負數。
- (1) 如果 past\_tasks 和 todos 中有相似的事件，則根據使用相似事件的 past\_tasks 時間和對應事件的 "difficulty"，自動推估出未開始行程的 estimated\_time\_minutes(difficulty 越大 estimated\_time\_minutes 越大)。
- (2) 如果 past\_tasks 和 todos 中相似事件的 difficulty 有點差(例如已完成演算法作業1的 difficulty=8，未完成演算法作業2的 difficulty=1)，則用 past\_tasks 和 todos 各項之間的難易度推算。(3) 推算不出來的則用科目和題型(例如線性代數) → 採用平均值或近似值。
- (4) 資訊嚴重不足時根據未開始行程自己的已知資訊和常識推算。
- 如果文字中有出現「必須優先處理」、「優先安排」、「a>b>c」等表述，則給對應事件設定 "priority"，優先級越高則 priority 越小，範圍為 0-100 之間的整數；其他未說明優先級的事件，priority 一律設定成 101。
- 未開始事件(todos)的 "title" 中如果出現「海試」、「小考」、「期中考」、「期末考」等文字，則將對應的 "exam\_or\_not" 設定成 true，沒有出現則設定成 false。
- 只輸出合法 JSON，不要加註解或多餘文字。

```
# 呼叫 Groq 的 chat.completions API 產生回應
completion = client.chat.completions.create(
    model=GROQ_MODEL_NAME, # 指定模型名稱
    messages=[
        {
            "role": "system", "content": system_prompt2, # system 訊息，定義行程與格式
            {
            "role": "user", "content": raw_text, # user 訊息，給使用者輸入的原始文字
        }
    ],
    temperature=0.2, # 溫度設定數值，結果更穩定
)

# 取得 LLM 回應的主體文字內容
content = completion.choices[0].message.content
raw = content.strip() # 去掉前後空白

# 如果 raw 里面有 ""json ..."" code block，先去掉外層
if raw.startswith('```'): # 檢查是否以 "" 開頭 (包含 ""json
    # 去掉開頭 "" 或 ""json 這一行
    raw = re.sub(pattern=r'^([""]|```json)', match='', repl='', raw)
    # 去掉最後的 ""，以及前面的空白換行
    raw = re.sub(pattern=r'[""]$', match='', repl='', raw).strip()
```

```
# 只抓第一個 { 到最後一個 } 之間的內容
first = raw.find('{') # 找到第一個 '{' 的位置
last = raw.rfind('}') # 找到最後一個 '}' 的位置
if first == -1 or last == -1: # 若找不到大括號
    print("A Groq 回應內容找不到大括號，原始內容:", content) # 印出原始內容
    return None # 回傳 None，讓上層改用 demo 資料

json_str = raw[first: last + 1] # 取出完整 JSON 字串範圍

# 丟給 Pydantic 解析
parsed = ParsedInput.model_validate_json(json_str) # 使用 Pydantic 的 JSON 解析功能
return parsed # 回傳解析後的 ParsedInput 實體

except Exception as e: # 若在上述任一步驟發生例外
    print("A Groq 解析失敗:", e) # 印出錯誤訊息
    return None # 回傳 None，呼叫端會改用 demo 資料
```

## (1) LLM 輸入內容：

文字輸入內容：(請輸入你的作業 / 考試 / 固定行程與時間，可以已寫成： 1. 上週高等演算法作業1，寫了大概7小時。 2. 上週數位影像處理作業1，寫了大概12小時。 3. 上次線代小考前複習3小時。 4. 上次計算機學期中考複習8小時。 未完成： 1. 12/09 早上10點要考線性代數小考1。 2. 12/11 晚上11:59要交高等演算法作業2。	3. 12/12 晚上11:59要交數位影像處理作業2。 4. 12/16 晚上11:59要交高等演算法作業3。 5. 12/17 早上9點要考線性代數小考2。 6. 12/22 早上9點要考計算機學期期末考。 7. 12/18 下午1點要考高等網路期末考。 8. 12/20 早上10點要考高等演算法期末考。	優先級：線性代數小考1 > 線性代數小考2 > 高等演算法作業3  固定行程： 1. 每週二晚上7點到9點補習。 2. 12/13 晚上6點到8點和家人吃飯。
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------

```
===== 難度解析結果 =====
past_tasks=[DifficultyPastTask(title='高等演算法_Exercise_1.pdf', difficulty=10), DifficultyPastTask(title='數位影像處理hw1.pdf', difficulty=9)] todos=[DifficultyTodo(title='高等演算法_Exercise2_.pdf', difficulty=12)]
=====
```



## (2) LLM 輸出內容：

```
LLM 解析結果 (debug 用)
{
  "past_tasks": [
    {
      "title": "高等演算法作業 1"
      "subject": "高等演算法"
      "time_spent_minutes": 420
      "difficulty": 10
    }
  ],
  "3": {
    "title": "數位影像處理作業 1"
    "subject": "數位影像處理"
    "time_spent_minutes": 720
    "difficulty": 9
  },
  "2": {
    "title": "線性代數小考 1"
    "subject": "線性代數"
    "time_spent_minutes": 180
    "difficulty": -1
  },
  "3": {
    "title": "計算機學期中考複習"
    "subject": "計算機學"
    "time_spent_minutes": 480
    "difficulty": -1
  }
}

"todos": [
  {
    "title": "線性代數小考 1"
    "subject": "線性代數"
    "deadline": "datetime.datetime(2025, 12, 9, 9, 0)"
    "estimated_time_minutes": 180
    "difficulty": -1
    "priority": 1
    "exam_or_not": true
  },
  {
    "title": "高等演算法作業 2"
    "subject": "高等演算法"
    "deadline": "datetime.datetime(2025, 12, 11, 23, 59)"
    "estimated_time_minutes": 504
    "difficulty": 12
    "priority": 101
    "exam_or_not": false
  },
  {
    "title": "數位影像處理作業 2"
    "subject": "數位影像處理"
    "deadline": "datetime.datetime(2025, 12, 12, 23, 59)"
    "estimated_time_minutes": 720
    "difficulty": -1
    "priority": 101
    "exam_or_not": false
  },
  {
    "title": "數位影像處理作業 2"
    "subject": "數位影像處理"
    "deadline": "datetime.datetime(2025, 12, 12, 23, 59)"
    "estimated_time_minutes": 720
    "difficulty": -1
    "priority": 101
    "exam_or_not": false
  }
]

"3": {
  "title": "高等演算法作業 3"
  "subject": "高等演算法"
  "deadline": "datetime.datetime(2025, 12, 16, 23, 59)"
  "estimated_time_minutes": 504
  "difficulty": -1
  "priority": 3
  "exam_or_not": false
}

"4": {
  "title": "線性代數小考 2"
  "subject": "線性代數"
  "deadline": "datetime.datetime(2025, 12, 17, 9, 0)"
  "estimated_time_minutes": 180
  "difficulty": -1
  "priority": 2
  "exam_or_not": true
}

"5": {
  "title": "計算機學期中考複習"
  "subject": "計算機學"
  "deadline": "datetime.datetime(2025, 12, 22, 9, 0)"
  "estimated_time_minutes": 480
  "difficulty": -1
  "priority": 101
  "exam_or_not": true
}

"6": {
  "title": "高等演算法期中考"
  "subject": "高等演算法"
  "deadline": "datetime.datetime(2025, 12, 18, 13, 0)"
  "estimated_time_minutes": 240
  "difficulty": -1
  "priority": 101
  "exam_or_not": true
}

"7": {
  "title": "高等演算法期中考"
  "subject": "高等演算法"
  "deadline": "datetime.datetime(2025, 12, 20, 10, 0)"
  "estimated_time_minutes": 504
  "difficulty": -1
  "priority": 101
  "exam_or_not": true
}

"fixed_events": [
  {
    "title": "補習"
    "start": "datetime.datetime(2025, 12, 16, 19, 0)"
    "end": "datetime.datetime(2025, 12, 16, 21, 0)"
  },
  {
    "title": "補習"
    "start": "datetime.datetime(2025, 12, 23, 19, 0)"
    "end": "datetime.datetime(2025, 12, 23, 21, 0)"
  },
  {
    "title": "和家人吃飯"
    "start": "datetime.datetime(2025, 12, 13, 18, 0)"
    "end": "datetime.datetime(2025, 12, 13, 20, 0)"
  }
]
```

## 4. 將 LLM 最終輸出結果和使用者設定的排程條件一起輸入排程演算法進行排程。

排程邏輯如下：

- (1) 排程按照使用者設定的每日限定時段、上限時間等條件將行程以 60 分鐘為單位插入使用者設定的日期範圍。
- (2) 超時行程會被篩除、固定行程時段不得更動。
- (3) 如果使用者有設定優先級，則按照優先級順序排程。
- (4) 未設定優先級則按照截止日期先後順序排程。
- (5) 如果使用者開啟 "臨時抱佛腳模式"，則考試複習行程安排在緊鄰考試時間前。

```
# 根據解析結果進行排程
with st.spinner("排程中..."):
    if parsed:
        schedule = build_schedule(
            parsed, # 解析後的結構化資料
            days=schedule_days, # 往後排幾天, 由使用者輸入
            start_date=start_date, # 排程開始日期
            max_hours_per_day=max_hours_per_day, # 每天最多作業/複習小時
            study_start_hour=int(study_start_hour), # 每天排程開始小時 (轉成 int)
            study_end_hour=int(study_end_hour), # 每天排程結束小時 (轉成 int)
            cram_or_not=cram_or_not, # 考試複習時間安排模式
        )
        st.session_state["schedule"] = schedule # 排好的結果存到 session_state

# 依 deadline 排序 todo (最近截止的先排)
todos = sorted(parsed.todos, key=lambda x: x.deadline)

# 依 priority 排序 todo (優先級最高的先排)
todos = sorted(todos, key=lambda x: x.priority)

def is_free(start: datetime, end: datetime) -> bool:
    """檢查這個時間區間有沒有被占用"""
    for b in blocks:
        # 結尾時間上有重疊 (不是完全在前或完全在後), 代表被占用
        if not (end <= b.start or start >= b.end):
            return False # 發現重疊直接回傳 False
    return True # 給沒有與任何 block 重疊則回傳 True

# 依序處理每一個 todo 行程
for todo in todos:
    remaining = todo.estimated_time_minutes # 剩餘需要安排的分鐘數
    # 給剩餘時間已低於 now 之前, 就不排 (之後在 UI 顯示為高亮)
    if todo.deadline <= now:
        continue # 直接跳過這個 todo
    # 在可用的時間區間中嘗試安排這個行程
    for slot_start in iter_study_slots(
        start_base, # 排程起始時間 (含排日期 + 00:00)
        days, # 排程天數
        study_start_hour, # 每天排程開始小時
        study_end_hour, # 每天排程結束小時
    ):
        if slot_start >= todo.deadline:
            # 已經超過最後作業的截止時間, 不再往後排
            break
        if remaining <= 0:
            break # 已經排滿時間, 不再需要排程

# 依 deadline 排序 todos
if cram_or_not == True and todo.exam_or_not == True:
    num_of_days = todo.deadline.day - slot_start.day - 1
    num_of_minutes = num_of_days * max_minutes_per_day
    num_of_minutes = max((study_end_hour - slot_start.hour) * 60, 0) + num_of_minutes
    num_of_minutes = min(todo.deadline.hour - study_start_hour, 0) * 60
    for slot in iter_study_slots(slot_start, num_of_days+1, study_start_hour, study_end_hour, ):
        if not is_free(slot, slot+timedelta(minutes=BLOCK_MINUTES)):
            num_of_minutes -= BLOCK_MINUTES
    if num_of_minutes > todo.estimated_time_minutes: continue
```

## 5. 在網頁上顯示排程結果並提供 ICS 檔案下載匯入第三方日曆程式。

```
# 顯示排程結果 + 排不完任務 + 下載 .ics
if st.session_state["schedule"] is not None:
    st.subheader("排程結果") # 小標題: 排程結果
    schedule = st.session_state["schedule"] # 取出排程結果列表
    parsed = st.session_state["parsed"] # 再取一次解析結果, 方便後續比對
```



```
# ---- 2. 正常排程列表 (表格) ----
st.markdown("### 排程列表") # 小標題: 排程列表
# 將所有排程區塊整理成表格資料
table_data = [
    {
        "標題": b.title, # 事件標題
        "開始": b.start.strftime("%Y-%m-%d %H:%M"), # 開始時間字串
        "結束": b.end.strftime("%Y-%m-%d %H:%M"), # 結束時間字串
        "種類": "作業/複習" if b.kind == "todo" else "固定行程", # 依 kind 顯示中文種類
    }
    for b in schedule
]
st.table(table_data) # 用表格顯示所有已排事件

# ---- 3. 下載 .ics ----

st.subheader("匯入 Google Calendar") # 小標題: Google Calendar 區塊
st.caption("( 或者其他支援 .ics 檔的日曆 )")

st.markdown("### 單一日曆下載") # 小標題: Google Calendar 區塊

# 3-1 全部一起版本
all_ics = schedule_to_ics(schedule) # 將完整排程轉成單一 .ics 字串
st.download_button(
    label=" [全部行程] .ics", # 按鈕文字
    data=all_ics, # .ics 檔案內容
    file_name="study_schedule_all.ics", # 下載檔名
    mime="text/calendar", # MIME 型別
)

# 3-2 Google Calendar 兩檔日曆版本
fixed_ics, todo_ics = split_schedule_to_ics_for_google(schedule) # 拆成兩份 .ics

st.markdown("### 分成兩個日曆下載") # 小標題: Google Calendar 區塊
st.caption("方便分開匯入不同日曆, 以顯示不同顏色")

# 下載固定行程 .ics
st.download_button(
    label=" [固定行程] .ics", # 按鈕文字
    data=fixed_ics, # 固定行程 .ics 內容
    file_name="fixed_events.ics", # 檔名
    mime="text/calendar", # MIME 型別
)

# 下載作業 / 複習 .ics
st.download_button(
    label=" [作業/考試複習] .ics", # 按鈕文字
    data=todo_ics, # 作業 / 複習 .ics 內容
    file_name="tasks_events.ics", # 檔名
    mime="text/calendar", # MIME 型別
)
```

6. 排程器輸出結果

排程結果

無法完全安排的任務

以下任務在目前設定的排程天數 / 每日作業上環 / 每天排程時段內，無法完全排完預估所需時間：

	標題	科目	截止時間	預估時間(分鐘)	已排時間(分鐘)	尚未排入(分鐘)	
0	高等演算法作業 2	高等演算法	2025-12-11 23:59		504	420	84
1	數位影像處理作業 2	數位影像處理	2025-12-12 23:59		720	480	240
2	高等演算法期末考	高等演算法	2025-12-20 10:00		504	480	24

排程列表

	標題	開始	結束	種類
0	高等演算法作業 3	2025-12-10 13:00	2025-12-10 14:00	作業/複習
1	高等演算法作業 3	2025-12-10 14:00	2025-12-10 15:00	作業/複習
2	高等演算法作業 3	2025-12-10 15:00	2025-12-10 16:00	作業/複習
3	高等演算法作業 3	2025-12-10 16:00	2025-12-10 17:00	作業/複習
4	高等演算法作業 3	2025-12-10 17:00	2025-12-10 18:00	作業/複習
5	高等演算法作業 3	2025-12-10 18:00	2025-12-10 19:00	作業/複習
6	高等演算法作業 3	2025-12-10 19:00	2025-12-10 20:00	作業/複習
7	高等演算法作業 3	2025-12-10 20:00	2025-12-10 21:00	作業/複習
8	高等演算法作業 3	2025-12-11 13:00	2025-12-11 14:00	作業/複習
9	高等演算法作業 2	2025-12-11 14:00	2025-12-11 15:00	作業/複習
10	高等演算法作業 2	2025-12-11 15:00	2025-12-11 16:00	作業/複習
11	高等演算法作業 2	2025-12-11 16:00	2025-12-11 17:00	作業/複習
12	高等演算法作業 2	2025-12-11 17:00	2025-12-11 18:00	作業/複習

匯入 Google Calendar

( 或者其他支援 .ics 檔的日曆 )

單一日曆下載

[全部行程] .ics

分成兩個日曆下載

方便分開匯入不同日曆，以顯示不同顏色

[固定行程] .ics

[作業/考試複習] .ics