2455012 김현식

1. 라이브러리 import

```
import gradio as gr
import pandas as pd
import altair as alt
import numpy as np
import sys
import os
```

먼저, 필요한 라이브러리를 불러옵니다.

- gradio: 웹 UI 구성
- pandas: 데이터 처리
- altair: 시각화를 위한 라이브러리
- numpy: 배열 및 수치 처리
- sys, os: 파일 경로 처리 등에 사용

2. OR-Tools 설치 여부 확인

```
try:
    from ortools.sat.python import cp_model
    ORTOOLS_AVAILABLE = True
except Exception:
    ORTOOLS_AVAILABLE = False
```

- OR-Tools는 구글의 최적화 라이브러리로, AI 시간표 스케줄링의 핵심 엔진입니다.
- 이 블록에서는 OR-Tools 설치 여부를 확인하고, 없으면 경고를 출력합니다.
- 앱이 실행은 되지만, AI 스케줄링 기능은 제한됩니다.

```
# ai_timetable_app_fixed.py
import gradio as gr
import pandas as pd
import altair as alt
import numpy as np
import sys
import os

# --- ortools 존재 확인 (앱 시작시 경고만 보여주기 위해 try/except) ---
try:
    from ortools.sat.python import cp_model
    ORTOOLS_AVAILABLE = True
    print("--- 'ortools' 라이브러리가 성공적으로 로드되었습니다. ---")
except Exception:
    ORTOOLS_AVAILABLE = False
    print("--- 경고: 'ortools' 라이브러리가 설치되어 있지 않습니다. ---")
    print("터미널에서 'pip install ortools'후 AI 배정을 사용하세요.")
```

3. 헬퍼 함수 정의

(1)컬럼명을 정리하는 함수

```
def clean_col_names_simple(df):
df.columns = df.columns.str.strip()
return df
```

CSV 파일에서 열 이름 앞뒤에 공백이 포함된 경우 오류를 방지하기 위해 컬럼명을 정리합니다.

```
# --- 헬퍼 함수들 ---
def clean_col_names_simple(df: pd.DataFrame) -> pd.DataFrame:
    """컬럼명 앞뒤 공백 제거"""
    df.columns = df.columns.str.strip()
    return df

def load_and_filter_data(csv_path, max_capacity=45):
    """
    CSV 파일 경로를 받아 필터링된 실습(실험실) 수업 데이터 반환.
    - 필수 컬럼 존재 여부 검사
    - '실습' 유형만 추출
    - 수강인원, 학점 컬럼 숫자 변환 및 결측 제거
    - duration(학점)>0 인 것만
    - max_capacity 이하인 것만 선별
    """
    try:
        df = pd.read_csv(csv_path)
        df = clean_col_names_simple(df)

        type_col = '강의유형구분'
        student_count_col = '수강인원'
        course_name_col = '교과목명'
        prof_col = '강좌담당교수'
        duration_col = '교과목학점'

        required_cols = [type_col, student_count_col, course_name_col, prof_col, duration_col]
        if not all(col in df.columns for col in required_cols):
            raise ValueError(f"Error: 필수 컬럼 누락. 필요한 컬럼: {required_cols}")
```

(2)데이터 로드 + 필터링 함수

def load_and_filter_data(csv_path, max_capacity=45):

이 함수는 AI 배정에 필요한실습 수업만 자동으로 걸러내는 핵심 전처리 함수입니다.

- CSV 파일 불러오기
- 필수 컬럼 존재 여부 확인
- 강의 유형이 "실습"인 것만 선택
- 수강인원과 학점(시수)을 숫자로 변환
- 결측값 제거
- 수강 인원이 최대 수용 인원 이하인 수업만 필터링

사용자가 업로드한 수업 중 실제로 배정 가능한 수업만 모아 최종적으로 반환합니다.

```
# '실습' 타입만
    df_lab = df[df[type_col] == '실습'].copy()

    # 숫자 변환
    df_lab[student_count_col] = pd.to_numeric(df_lab[student_count_col], errors='coerce')
    df_lab[duration_col] = pd.to_numeric(df_lab[duration_col], errors='coerce')

    # 결측 제거
    df_lab = df_lab.dropna(subset=[student_count_col, duration_col])

    # 정수형 학점, 양수 필터
    df_lab[duration_col] = df_lab[duration_col].astype(int)
    df_lab = df_lab[df_lab[duration_col] > 0]

    # 수용 인원 제약
    df_schedulable = df_lab[df_lab[student_count_col] <= max_capacity].reset_index(drop=True)

    log_message = f"총 {len(df_lab)}개 실습 수업 중 {len(df_schedulable)}개가 배정 대상입니다."
    print(log_message)

    col_names = [course_name_col, prof_col, duration_col, student_count_col]
    return df_schedulable, col_names, log_message

except FileNotFoundError:
    raise FileNotFoundError(f"Error: {csv_path} 파일을 찾을 수 없습니다.")
except Exception as e:
    raise Exception(f"데이터 로드 중 오류: {e}")
```

4. AI 스케줄러 (OR-Tools CP-SAT)

def run_ai_scheduler(df_demand, col_names):
이 부분이**AI**가 실제로 시간표를 배정하는 두뇌 역할입니다.단계별 설명

① 강의실 설정

rooms = {'1215': 45, '1216': 45, '1217': 45, '1418': 45, 'Rental_Room_1': 45}
AI가 배정할 수 있는 실습실 목록입니다.
수용 인원도 설정되어 있습니다.

② 시간 축 설정

num_days = 5 # 월~금num_periods = 9 # 1~9교시
총 5일 × 9교시 = 45개의 슬롯을 하나의 일렬 시간축으로 계산합니다.

③ OR-Tools 모델 초기화

model = cp_model.CpModel()
제약 조건을 정의하고 최적해를 찾는 모델입니다

```
# --- AI 스케줄러 ---
def run_ai_scheduler(df_demand: pd.DataFrame, col_names):
    """
    OR-Tools CP-SAT을 사용한 스케줄링.
    - rooms: 강의실 목록과 수용인원(현재 수용인원은 제약 필터링에서 이미 반영)
    - 하루 num_periods(1~9교시), 5일(월~금)
    - 단순한 시작슬롯 기반 스케줄링 (교시 단위)
    """
    if not ORTOOLS_AVAILABLE:
        raise ImportError("AI 배정을 실행하려면 'ortools'가 필요합니다. 터미널에서 'pip install ortools'를 설치하세요.")

    course_name_col, prof_col, duration_col, student_count_col = col_names

    rooms = {'1215': 45, '1216': 45, '1217': 45, '1418': 45, 'Rental_Room_1': 45}
    room_names = list(rooms.keys())
    num_rooms = len(room_names)
    num_days = 5
    num_periods = 9  # 교시 수 (1..9)
    horizon = num_days * num_periods
```

④ 각 수업에 대해 변수 생성
예:
언제 시작하는지(start)
언제 끝나는지(end)
어느 강의실에 배정되는지(room)
각 수업의 시간

```
starts[cid] = model.NewIntVar(0, horizon - dur, f'start_{cid}')
ends[cid] = model.NewIntVar(0, horizon, f'end_{cid}')
rooms_var[cid] = model.NewIntVar(0, num_rooms - 1, f'room_{cid}')
```

```
# 클래스 객체 리스트
classes = []
for i, row in df_demand.iterrows():
    classes.append({
        "id": int(i),
        "name": str(row[course_name_col]),
        "duration": int(row[duration_col]),
        "professor": str(row[prof_col]) if pd.notna(row[prof_col]) else "UNKNOWN",
        "students": int(row[student_count_col])
    })

model = cp_model.CpModel()
starts = {}
ends = {}
rooms_var = {}
intervals = {}
prof_intervals = {}

for c in classes:
    cid = c['id']
    dur = c['duration']
    starts[cid] = model.NewIntVar(0, horizon - dur, f'start_{cid}')
    ends[cid] = model.NewIntVar(0, horizon, f'end_{cid}')
    # end = start + dur
    model.Add(ends[cid] == starts[cid] + dur)
    rooms_var[cid] = model.NewIntVar(0, num_rooms - 1, f'room_{cid}')
    intervals[cid] = model.NewIntervalVar(starts[cid], dur, ends[cid], f'interval_{cid}')
```

⑤ 강의실 중첩 방지 제약

model.AddNoOverlap(room_intervals)

같은 강의실에서 두 수업이 겹치지 않도록 하는 제약 조건입니다.

⑥ 교수 중복 방지 제약

model.AddNoOverlap(intervals_list)

한 교수님이 같은 시간대에 여러 수업을 진행할 수 없도록 금지합니다.

```
    # 교수별 충돌 방지를 위한 맵
    prof = c['professor']
    prof_intervals.setdefault(prof, []).append(intervals[cid])

# 제약 1: 같은 강의실에서 겹치지 않도록 (각 강의실에 대해 OptionalInterval로 처리)
for r_idx in range(num_rooms):
    room_intervals = []
    for c in classes:
        cid = c['id']
        is_in_room = model.NewBoolVar(f'c{cid}_in_r{r_idx}')
        # rooms_var == r_idx <-> is_in_room
        model.Add(rooms_var[cid] == r_idx).OnlyEnforceIf(is_in_room)
        model.Add(rooms_var[cid] != r_idx).OnlyEnforceIf(is_in_room.Not())
        opt_interval = model.NewOptionalIntervalVar(starts[cid], c['duration'], ends[cid], is_in_room, f'opt_int_c{cid}_r{r_idx}'
)
        room_intervals.append(opt_interval)
    model.AddNoOverlap(room_intervals)

# 제약 2: 교수별 중복 금지
for prof, intervals_list in prof_intervals.items():
    if len(intervals_list) > 1:
        model.AddNoOverlap(intervals_list)
```

⑦ 하루를 넘기는 배정 금지
수업이 월요일 8~10교시 이런 식으로 하루를 넘어갈 수 없도록 제약을 걸어줍니다.

⑧ 최적화 목표model.Minimize(total_makespan)
전체 스케줄이 끝나는 시간을최대한 앞당기는최적화입니다.
즉, 시간표가 더 *압축*된 형태로 만들어집니다.

⑨ AI 스케줄링 실행status = solver.Solve(model)
해가 존재하면 DataFrame 형태로 배정 결과를 만들어 반환합니다.

```
# 제약 3: 수업이 하루를 넘지 않도록 (start_day == end_day)
for c in classes:
    cid = c['id']
    start_day = model.NewIntVar(0, num_days - 1, f'start_day_{cid}')
    end_day = model.NewIntVar(0, num_days - 1, f'end_day_{cid}')
    # division equality: day = start // num_periods
    model.AddDivisionEquality(start_day, starts[cid], num_periods)
    temp_end_time = model.NewIntVar(0, horizon + c['duration'], f'temp_end_time_{cid}')
    model.Add(temp_end_time == starts[cid] + c['duration'] - 1)
    model.AddDivisionEquality(end_day, temp_end_time, num_periods)
    model.Add(start_day == end_day)

# 목표: 전체 makespan 최소화 (수업이 끝나는 최대 슬롯 최소화)
total_makespan = model.NewIntVar(0, horizon, 'makespan')
model.AddMaxEquality(total_makespan, [ends[c['id']] for c in classes])
model.Minimize(total_makespan)

# Solve
solver = cp_model.CpSolver()
solver.parameters.max_time_in_seconds = 30.0
status = solver.Solve(model)

if status in (cp_model.OPTIMAL, cp_model.FEASIBLE):
    results = []
    day_map = {0: '월', 1: '화', 2: '수', 3: '목', 4: '금'}
    for c in classes:
        cid = c['id']
        room_index = int(solver.Value(rooms_var[cid]))
        start_slot = int(solver.Value(starts[cid]))
        day_val = start_slot // num_periods
        period_start = (start_slot % num_periods) + 1
        period_end = period_start + c['duration'] - 1
        results.append({
            "교과목명": c['name'],
            "담당교수": c['professor'],
            "수강인원": c['students'],
            "배정강의실": room_names[room_index],
            "배정요일": day_map.get(day_val, str(day_val)),
            "시작교시": period_start,
            "종료교시": period_end,
            "주당시수": c['duration']
        })
    results_df = pd.DataFrame(results).sort_values(by=['배정강의실', '배정요일', '시작교시']).reset_index(drop=True)
    status_str = solver.StatusName(status) if hasattr(solver, "StatusName") else str(status)
    return results_df, f"AI 스케줄링 성공! (상태: {status_str})"
else:
    status_str = solver.StatusName(status) if hasattr(solver, "StatusName") else str(status)
    raise Exception(f"AI 스케줄링 실패. 해를 찾지 못했습니다. (상태: {status_str})")
```

5. 공실 현황 분석 (히트맵)

def analyze_and_visualize(df_schedule):

AI가 배정한 시간표를 토대로

요일 × 강의실 × 교시공실 여부를 정리한 뒤 Altair로 히트맵을 만듭니다.
- 파란색: 사용 중
- 회색: 공실
- 요일별로 나누어 표시

강의실 운영 상황을 직관적으로 확인할 수 있는 중요한 시각화입니다.

```
# --- 공실 분석 (히트맵) ---
def analyze_and_visualize(df_schedule: pd.DataFrame):
    """스케줄 결과를 받아 강의실 × 요일 × 교시 히트맵 생성"""
    if df_schedule is None or df_schedule.empty:
        # 빈 차트 반환
        empty_df = pd.DataFrame({"교시": [], "배정강의실": [], "배정요일": [], "상태": []})
        return alt.Chart(empty_df).mark_text().encode()

    rooms = ['1215', '1216', '1217', '1418', 'Rental_Room_1']
    days = ['월', '화', '수', '목', '금']
    periods = list(range(1, 10))

    # grid 초기화
    master_grid = {(r, d, p): 'Available' for r in rooms for d in days for p in periods}

    # 사용중 표시
    for row in df_schedule.itertuples():
        for p in range(int(row.시작교시), int(row.종료교시) + 1):
            key = (row.배정강의실, row.배정요일, p)
            if key in master_grid:
                master_grid[key] = row.교과목명

    df_grid = pd.Series(master_grid).reset_index()
    df_grid.columns = ['배정강의실', '배정요일', '교시', '상태']
    df_grid['상태'] = df_grid['상태'].apply(lambda x: '공실' if x == 'Available' else '사용중')
    df_grid['배정요일'] = pd.Categorical(df_grid['배정요일'], categories=days, ordered=True)
    df_grid['교시'] = pd.Categorical(df_grid['교시'], categories=periods, ordered=True)

    heatmap = alt.Chart(df_grid).mark_rect().encode(
        x=alt.X('교시:O', axis=alt.Axis(title='교시', labelAngle=0)),
        y=alt.Y('배정강의실:N', axis=alt.Axis(title='강의실')),
        color=alt.Color('상태:N',
                        scale=alt.Scale(domain=['공실', '사용중'], range=['#F0F0F0', '#007BFF']),
                        legend=alt.Legend(title="상태")),
        tooltip=[
            alt.Tooltip('배정요일', title='요일'),
            alt.Tooltip('배정강의실', title='강의실'),
            alt.Tooltip('교시', title='교시'),
            alt.Tooltip('상태', title='상태')
        ]
    ).properties(
        title='강의실별 공실 현황 (AI 배정 결과)'
    ).facet(
        row=alt.Facet('배정요일:N', title='요일', header=alt.Header(titleOrient="right", labelOrient="right"))
    ).interactive()

    return heatmap
```

6. 공실 대여 가능 시간 추출

def analyze_rentable_slots(df_schedule):

- 2시간 이상 연속된 공실 시간만 찾아서 정리
- 강의실 운영팀이나 외부 대여 신청자가 참고할 수 있는 리스트 생성

```
# --- 대여 가능 슬롯 추출 ---
def analyze_rentable_slots(df_schedule: pd.DataFrame):
    """2시간(교시) 이상 연속 공실 슬롯 추출"""
    if df_schedule is None or df_schedule.empty:
        return pd.DataFrame(columns=['강의실', '요일', '대여시작', '대여종료', '연속시간'])

    rooms = ['1215', '1216', '1217', '1418', 'Rental_Room_1']
    days = ['월', '화', '수', '목', '금']
    periods = list(range(1, 10))
    master_grid = {(r, d, p): 'Available' for r in rooms for d in days for p in periods}

    for row in df_schedule.itertuples():
        for p in range(int(row.시작교시), int(row.종료교시) + 1):
            key = (row.배정강의실, row.배정요일, p)
            if key in master_grid:
                master_grid[key] = row. 교과목명

    rentable_slots = []
    min_rental_duration = 2

    for room in rooms:
        for day in days:
            consecutive_count = 0
            start_period = None
            for period in periods:
                if master_grid[(room, day, period)] == 'Available':
                    if consecutive_count == 0:
                        start_period = period
                    consecutive_count += 1
                else:
                    if consecutive_count >= min_rental_duration:
                        rentable_slots.append({
                            '강의실': room,
                            '요일': day,
                            '대여시작': f"{start_period}교시",
                            '대여종료': f"{period - 1}교시",
                            '연속시간': consecutive_count
                        })
                    consecutive_count = 0
                    start_period = None
            # 마지막까지 공실이 이어진 경우 체크
            if consecutive_count >= min_rental_duration:
                rentable_slots.append({
                    '강의실': room,
                    '요일': day,
                    '대여시작': f"{start_period}교시",
                    '대여종료': f"9교시",
                    '연속시간': consecutive_count
                })

    return pd.DataFrame(rentable_slots)
```

7. Gradio 통합 실행 함수

def run_all_goals(file_obj, max_capacity):

버튼 1번만 누르면:
1. 데이터 로드
2. AI 시간표 배정
3. 공실 시각화
4. 대여 가능 시간대 생성
을 모두 자동으로 실행합니다.

```
# --- Gradio 통합 실행 함수 ---
def run_all_goals(file_obj, max_capacity):
    """
    Gradio에서 호출되는 메인 함수.
    file_obj: gr.File로 업로드된 파일 오브젝트 (file_obj.name 사용)
    """
    if file_obj is None:
        raise gr.Error("먼저 'courses_data.csv' 파일을 업로드하세요.")

    try:
        # 1. 데이터 로드
        df_schedulable, col_names, log1 = load_and_filter_data(file_obj.name, int(max_capacity))

        # 2. Goal 1: AI 스케줄링 실행
        df_schedule, log2 = run_ai_scheduler(df_schedulable, col_names)
        schedule_file = "schedule_prototype.csv"
        df_schedule.to_csv(schedule_file, index=False, encoding='utf-8-sig')

        # 3. Goal 2: 공실 분석 및 시각화
        heatmap_chart = analyze_and_visualize(df_schedule)

        # 4. Goal 3: 대여 목록 추출
        df_rentable = analyze_rentable_slots(df_schedule)
        rentable_file = "rentable_slots.csv"
        df_rentable.to_csv(rentable_file, index=False, encoding='utf-8-sig')

        log_final = f"{log1}\n{log2}\nGoal 2(시각화), Goal 3(대여 목록) 생성 완료."

        # 반환 순서: schedule dataframe, schedule CSV경로, heatmap(Altair chart), rentable df, rentable CSV경로, 로
그
        return [df_schedule, schedule_file, heatmap_chart, df_rentable, rentable_file, log_final]

    except Exception as e:
        raise gr.Error(str(e))
```

8. Gradio UI 구성

with gr.Blocks() as demo:

- 파일 업로드 UI
- 실행 버튼
- 결과 탭 3개 (배정 결과, 공실 현황 시각화, 대여 가능 목록)
- CSV 다운로드 기능 포함

사용자는 별도의 환경설정 없이 웹에서 바로 AI 시간표 생성이 가능합니다.

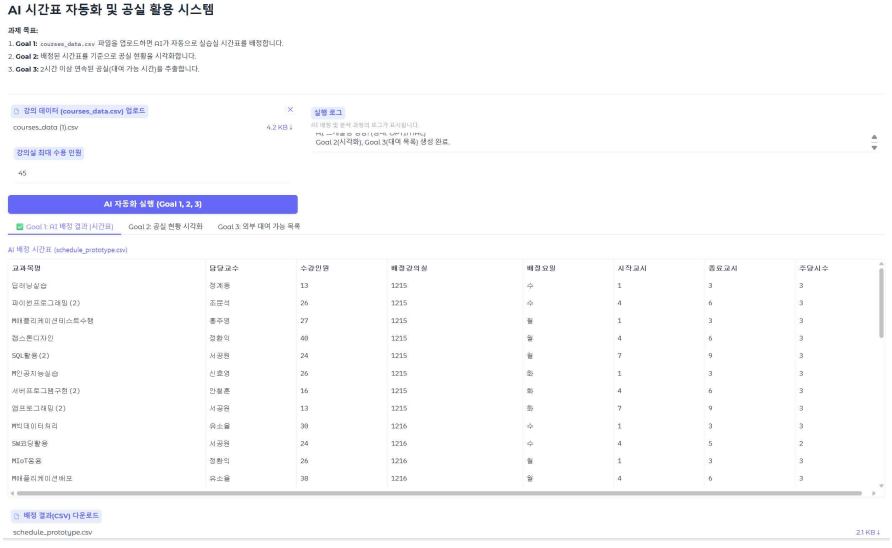
9. 마지막 — 앱 실행

```
if __name__ == "__main__":
    demo.launch(debug=True, share=True)
```

share=True 옵션으로
외부에서도 접속 가능한 URL이 생성됩니다.

```
# --- Gradio UI 구성 ---
with gr.Blocks(theme=gr.themes.Soft(), title="AI 시간표 자동화 시스템") as demo:
    gr.Markdown("""
# AI 시간표 자동화 및 공실 활용 시스템

**과제 목표:**
1. **Goal 1:** `courses_data.csv` 파일을 업로드하면 AI가 자동으로 실습실 시간표를 배정합니다.
2. **Goal 2:** 배정된 시간표를 기준으로 공실 현황을 시각화합니다.
3. **Goal 3:** 2시간 이상 연속된 공실(대여 가능 시간)을 추출합니다.
---
""")

    with gr.Row():
        with gr.Column(scale=1):
            input_file = gr.File(label="강의 데이터 (courses_data.csv) 업로드", file_types=[".csv"])
            max_capacity = gr.Number(label="강의실 최대 수용 인원", value=45)
            run_btn = gr.Button("AI 자동화 실행 (Goal 1, 2, 3)", variant="primary")
        with gr.Column(scale=2):
            run_log = gr.Textbox(label="실행 로그", info="AI 배정 및 분석 과정의 로그가 표시됩니다.", interactive=False)

    # 출력 탭
    with gr.Tabs():
        with gr.TabItem("✅ Goal 1: AI 배정 결과 (시간표)"):
            schedule_df_output = gr.DataFrame(label="AI 배정 시간표 (schedule_prototype.csv)", wrap=True)
            schedule_file_output = gr.File(label="배정 결과(CSV) 다운로드")
        with gr.TabItem("Goal 2: 공실 현황 시각화"):
            heatmap_output = gr.Plot(label="강의실별 공실 현황 히트맵 (요일별)")
        with gr.TabItem("Goal 3: 외부 대여 가능 목록"):
            rentable_df_output = gr.DataFrame(label="외부 대여 가능 공실 목록 (rentable_slots.csv)", wrap=True)
            rentable_file_output = gr.File(label="대여 목록(CSV) 다운로드")

    # 버튼 이벤트 연결
    run_btn.click(
        fn=run_all_goals,
        inputs=[input_file, max_capacity],
        outputs=[schedule_df_output, schedule_file_output, heatmap_output, rentable_df_output, rentable_file_output, run
_log]
    )

# --- 앱 실행 ---
if __name__ == "__main__":
    print("--- Gradio 앱을 실행합니다. ---")
    print("필요 패키지: gradio, ortools, pandas, altair, numpy")
    demo.launch(debug=True, share=True)
```

실행결과


AI 시간표 자동화 및 공실 활용 시스템
과제 목표:
1. Goal 1: courses_data.csv 파일을 업로드하면 AI가 자동으로 실습실 시간표를 배정합니다.
2. Goal 2: 배정된 시간표를 기준으로 공실 현황을 시각화합니다.
3. Goal 3: 2시간 이상 연속된 공실(대여 가능 시간)을 추출합니다.
강의 데이터 (courses_data.csv) 업로드
courses_data (1).csv
4.2 KB
실행 로그
AI 배정 및 분석 과정의 로그가 표시됩니다.
Goal 2(시각화), Goal 3(대여 목록) 생성 완료.
강의실 최대 수용 인원
45
AI 자동화 실행 (Goal 1, 2, 3)
Goal 1: AI 배정 결과 (시간표)
Goal 2: 공실 현황 시각화
Goal 3: 외부 대여 가능 목록
AI 배정 시간표 (schedule_prototype.csv)
교과목명
담당교수
수강인원
배정강의실
배정요일
시작교시
종료교시
주당시수
딥러닝실습
정계동
13
1215
수
1
3
3
파이썬프로그래밍 (2)
조문석
26
1215
수
4
6
3
캐릭터애니메이션스튜디오
홍주영
27
1215
월
1
3
3
캡스톤디자인
정환익
40
1215
월
4
6
3
SQL활용 (2)
서공원
24
1215
월
7
9
3
AI인공지능실습
신호영
26
1215
화
1
3
3
서버프로그램구현 (2)
안철훈
16
1215
화
4
6
3
앱프로그래밍 (2)
서공원
13
1215
화
7
9
3
빅데이터처리
유소율
38
1216
수
1
3
3
SW프로젝트
서공원
24
1216
수
4
5
2
IoT응용
정환익
26
1216
월
1
3
3
캐릭터애니메이션비보
유소율
38
1216
월
4
6
3
배정 결과(CSV) 다운로드
schedule_prototype.csv
2.1 KB

강의실별 공실 현황 히트맵 (요일별)

히트맵 결과

쉬는시간과 점심시간은 모두 배재하고,
가장 빠르고 효율적인 시간표 배정을 요구하여
위와 같은 결과가 도출되었습니다.

가상의 교실을 하나 더 추가하여 더 많은 학생들이
있을 경우의 시간표를 제작했습니다.