과제 #2

M3239.000300 데이터 사이언스를 위한 소프트웨어 플랫폼

Due: 2020년 4월 12일 23시 59분

1 문제 1: 캘린더

Google Calendar와 유사하게 일정을 추가, 확인 및 수정할 수 있는 캘린더를 python class Calendar로 구현하라. Calendar의 제약조건은 아래와 같다.

- Calendar에는 2021년 4월 5일 00시 00분 이후의 임의의 시간에 대한 일정을 저장할 수 있다.
- 한 일정은 **일정 이름, 시작 시간, 끝나는 시간**으로 구성된다. 시작 시간 및 끝나는 시간은 'YYYY-MM-DD HH:MM'꼴의 string으로 표현한다 (예: '2021-04-05 09:35', '2025-05-18 15:28'). 모든 시간은 분 단위까지만 표기한다.
- Calendar에 여러 개의 일정을 저장할 경우, 일정 간 시간이 겹치지 않아야 한다. 새로 입력하는 일정이 기존 일정과 충돌하는 경우 Exception을 발생시켜야 한다 (add_schedule method 참고)
- 같은 이름의 일정이 여러 개 있을 수 있다.
- 생성자는 별도의 입력을 받지 않는다.
- Calendar object를 print할 경우, **2021년 4월 5일 00시 00분 기준**으로 가장 가까운 일정 10개의 정보를 출력한다. 한 일정당 한 줄 씩 출력하며, 각 일정은 **일정 이름, 시작 시간, 끝나는 시간** 순서대로 출력한다. 각 항목(일정 이름, 시작 시간 등) 사이는 comma(,) 한 개와 띄어쓰기 한 개로 구분한다.

Calendar class는 add_schedule, remove_schedule, check_schedule method를 지원하며, 각각의 기능은 아래와 같다.

- **add_schedule(name, start, end)**

  start 시간에 시작하여 end 시간에 끝나는 일정 한 개를 추가한다. name, start, end는 string object이다. start와 end는 'YYYY-MM-DD HH:MM'포맷을 따른다 (예: '2021-04-05 15:35'). 일정은 start 시간에 시작하여 end 시간 직전에 끝난다고 가정한다(예: start가 '2021-04-05 15:35'이고 end가 '2021-04-05 15:40'인 경우, 2021년 4월 5일 15:35, 15:36, 15:37, 15:38, 15:39는 해당 일정에 포함되는 시간이며, 15:40은 포함되는 시간이 아님).

  모든 입력은 올바른 포맷으로 들어온다고 가정하며, 입력 포맷과 관련해서는 별도의 에러 처리 루틴을 구현하지 않아도 된다 (올바르지 않은 start, end 예시: 'April 5, 15:30', '2021-04-05 9:27 '). 또한 start가 end보다 뒷 시간으로 입력되는 경우도 없다고 가정한다.)

  입력한 일정이 이미 저장된 일정과 충돌할 경우, ValueError를 발생시킨다.

- **remove_schedule(time)**

  time 시간을 포함하는 일정이 있는지 확인하고, 일정이 있는 경우 해당 일정을 삭제한다. 일정이 없는 경우는 아무것도 하지 않는다.

- **check_schedule(time)**

  time 시간을 포함하는 일정이 있는지 확인하고, 일정이 있는 경우 해당 일정의 정보를 string으로 반환한다. 반환되는 string은 **일정 이름, 시작 시간, 끝나는 시간** 순서대로 구성된다. 각 항목(일정 이름, 시작 시간 등) 사이는 comma(,) 한 개와 띄어쓰기 한 개로 구분한다. 일정이 없는 경우 빈 string을 반환한다.

입출력 예시는 아래와 같다.

```
spdsta@login:~/2021-1/answer/HW2$ python3 -i prob1.py
>>> calendar = Calendar()
>>> calendar.add_schedule('Lab meeting', '2021-04-05 13:00', '2021-04-05 14:00')
>>> print(calendar)
Lab meeting, 2021-04-05 13:00, 2021-04-05 14:00
>>> calendar.add_schedule('Lunch', '2021-04-05 12:00', '2021-04-05 13:00')
>>> calendar.add_schedule('Lunch', '2021-04-06 12:00', '2021-04-06 13:00')
>>> ret1 = calendar.check_schedule('2021-04-06 12:30')
>>> ret2 = calendar.check_schedule('2021-04-05 12:00')
>>> ret3 = calendar.check_schedule('2021-04-05 13:00')
>>> ret4 = calendar.check_schedule('2021-04-05 14:00')
>>> calendar.remove_schedule('2021-04-05 12:30')
>>> ret5 = calendar.check_schedule('2021-04-05 12:00')
>>> print(ret1)
Lunch, 2021-04-06 12:00, 2021-04-06 13:00
>>> print(ret2)
Lunch, 2021-04-05 12:00, 2021-04-05 13:00
>>> print(ret3)
Lab meeting, 2021-04-05 13:00, 2021-04-05 14:00
>>> print(ret4)

>>> print(ret5)

>>> try:
...   calendar.add_schedule('Homework', '2021-04-06 11:00', '2021-04-06 13:00')
... except ValueError:
...   print("We have another schedule!")
...
We have another schedule!
>>>
```

2 문제 2: 반복되는 일정 처리

매일 반복되는 일정을 추가/삭제할 수 있는 class SmartCalendar를 구현하라. SmartCalendar는 Calendar를 상속한 class로 구현한다. 추가로 구현해야 할 method는 아래와 같다.

- **add_daily_schedule(name, start, end)**

  매일 반복되는 일정을 추가한다. 이 때 start, end의 경우 'HH:MM'포맷으로 시간만 표기하며(예: '11:45', 매일 같은 이름의 일정이 해당 시간에 추가된다. 이 때 add_schedule method와 마찬가지로, 기존에 추가된 일정, 혹은 기존에 추가된 반복 일정과 충돌이 있을 경우 ValueError exception을 raise하여야 한다.

기존 Calendar class의 method들 또한 add_daily_schedule의 동작을 고려하여 동작하여야 한다. 예를 들어 매일 12:00-12:30에 반복 일정이 잡힌 경우, add_schedule method로 4월 15일 12:15-12:45 일정을 추가하고자 하면 Exception이 발생되어야 한다.

또한 remove_schedule 명령으로 지정한 시각이 반복 일정에 해당하는 시간일 경우, 전체 반복 일정을 삭제한다 (아래 예시의 Lunch 참고).

입출력 예시는 아래와 같다.

```
spdsta@login:~/2021-1/answer/HW2$ python3 -i prob2.py
>>> calendar = SmartCalendar()
>>> calendar.add_schedule('Lab meeting', '2021-04-05 13:00', '2021-04-05 14:00')
>>> calendar.add_schedule('Play video game', '2021-04-10 15:00', '2021-04-10 16:00')
>>> calendar.add_daily_schedule('Breakfast', '08:00', '08:30')
>>> calendar.add_daily_schedule('Lunch', '12:00', '13:00')
>>> calendar.add_daily_schedule('Dinner', '18:00', '19:00')
>>> print(calendar)
Breakfast, 2021-04-05 08:00, 2021-04-05 08:30
Lunch, 2021-04-05 12:00, 2021-04-05 13:00
Lab meeting, 2021-04-05 13:00, 2021-04-05 14:00
Dinner, 2021-04-05 18:00, 2021-04-05 19:00
Breakfast, 2021-04-06 08:00, 2021-04-06 08:30
Lunch, 2021-04-06 12:00, 2021-04-06 13:00
Dinner, 2021-04-06 18:00, 2021-04-06 19:00
Breakfast, 2021-04-07 08:00, 2021-04-07 08:30
Lunch, 2021-04-07 12:00, 2021-04-07 13:00
Dinner, 2021-04-07 18:00, 2021-04-07 19:00
>>> calendar.remove_schedule('2021-04-10 12:30')
>>> print(calendar)
Breakfast, 2021-04-05 08:00, 2021-04-05 08:30
Lab meeting, 2021-04-05 13:00, 2021-04-05 14:00
Dinner, 2021-04-05 18:00, 2021-04-05 19:00
Breakfast, 2021-04-06 08:00, 2021-04-06 08:30
Dinner, 2021-04-06 18:00, 2021-04-06 19:00
Breakfast, 2021-04-07 08:00, 2021-04-07 08:30
Dinner, 2021-04-07 18:00, 2021-04-07 19:00
Breakfast, 2021-04-08 08:00, 2021-04-08 08:30
Dinner, 2021-04-08 18:00, 2021-04-08 19:00
Breakfast, 2021-04-09 08:00, 2021-04-09 08:30
>>>
```

```
>>> try:
...   calendar.add_schedule('Homework', '2021-04-06 18:30', '2021-04-06 18:45')
... except ValueError:
...   print("We have another schedule!")
...
We have another schedule!
>>> try:
...   calendar.add_daily_schedule('Coffe break', '18:30', '18:45')
... except ValueError:
...   print("We have another schedule!")
...
We have another schedule!
>>> try:
...   calendar.add_daily_schedule('Coffe break', '13:30', '13:45')
... except ValueError:
...   print("We have another schedule!")
...
We have another schedule!
>>> ret = calendar.check_schedule('2050-12-25 08:15')
>>> print(ret)
Breakfast, 2050-12-25 08:00, 2050-12-25 08:30
>>>
```

3 구현 및 제출

실습 서버의 실습 계정 홈 디렉토리의 HW2 디렉토리에 prob1.py, prob2.py를 작성하여 저장한다. 각각의 파일에는 아래와 같은 내용이 구현되어 있어야 한다.

파일	설명
prob1.py	Calendar class 구현
prob2.py	SmartCalendar class 구현

제출 시 아래와 같은 사항을 유의하라.

- 제출 파일에는 **반드시 class를 정의하는 코드만 포함되어야 하며, 테스트 코드 등이 포함되어서는 안 된다**. 과제 채점 시 실행 예시 스크린샷에서 보인 바와 유사한 방법으로 기계적인 채점을 수행할 예정이다. 이 때 제출한 코드에 불필요한 내용이 포함되어 채점이 방해될 경우 해당 과제가 0점 처리될 수 있다. 단, class에 추가로 method를 구현하여 내부 구현에서 사용하는 것은 괜찮다.
- SmartCalendar class는 반드시 Calendar class를 상속하도록 구현하여야 한다. 이를 만족하지 않을 경우 과제가 크게 감점될 수 있다. 상속을 위해 아래 예시와 같이 prob1.py 파일에서 Calendar class를 import 하여 사용할 수 있다.

```
fropm prob1 import Calendar

class SmartCalendar(Calendar):
    ...
```

- 과제 1번과 마찬가지로 기본 Python 문법에서 제공하는 기능만을 사용하여 과제를 구현하여야 하며, **prob1.py 파일 외에 타 모듈을 import하여 사용하는 것은 허용하지 않는다**.
- 과제 조건 중 **Exception 처리를 해야 하는 경우**와 **없다고 가정한 경우**를 잘 구분하여 구현하여야 한다. 과제 문서에 명시적으로 Exception 처리를 요구한 경우, 해당 case를 조교가 채점 시 사용할 수 있다. 없다고 가정한 경우들은 채점에서 사용하지 않는다. 과제 문서의 설명이 명확하지 않다고 판단되면 조교 메일 혹은 게시판에 질문할 것.
- **이번 과제부터는 별도의 언급이 없으면 과제를 ETL로 제출하지 않는다**. 실습 서버의 HW2 디렉토리에 파일을 저장하면, 과제 제출 시간에 자동으로 해당 파일을 복사해 간다. 파일명을 다른 이름으로 수정할 경우 (예: p1.py) 과제가 제출되지 않을 수 있으니 유의하여야 한다.
- 각 method의 입출력 정의를 반드시 이해하고 그대로 구현하도록 한다. 예를 들어 check_schedule method는 string을 반환하는데, 이 대신 method 내에서 화면에 해당 string을 출력하게 구현한 경우 점수가 크게 감점될 수 있다.
- 그레이스 데이를 사용할 경우 제출 당일 메일로 조교에게 이를 알려야 하며, 메일 제목은 [SPDS] HW2-Graceday 학번 이름의 포맷으로 작성하라 (예: [SPDS] HW2-Graceday 2020-20000 안규수).