Module submission header

Submission preparation instructions

Completion of this header is mandatory, subject to a 2-point deduction to the assignment. Only add plain text in the designated areas, i.e., replacing the relevant 'NA's. You must fill out all group member Names and Drexel email addresses in the below markdown list, under header **Module submission group**. It is required to fill out descriptive notes pertaining to any tutoring support received in the completion of this submission under the **Additional submission comments** section at the bottom of the header. If no tutoring support was received, leave NA in place. You may as well list other optional comments pertaining to the submission at bottom. *Any distruption of this header's formatting will make your group liable to the 2-point deduction.*

Module submission group

- Group member 1
  - Name: Kasonde Chewe
  - Email: kc3745@drexel.edu
- Group member 2
  - Name: Kholoud Hamed M Al Nazzawi
  - Email: ka974@drexel.edu
- Group member 3
  - Name: Ghanath V
  - Email: gv347@drexel.edu
- Group member 4
  - Name: NA
  - Email: NA

Additional submission comments

- Tutoring support received: NA
- Other (other): NA

Assignment Group 2

Module A *(25 points)*

Overall, our goal will be to take the site's base, comma-delimiated format for schedules in this format, to convert them into a functioning schedule, like in

[http://www3.septa.org/ccstations/ss/index.php] (this page).

A1. *(3 points)* First, complete the url construction in the function to get the real-time schedule for Suburban Station using SEPTA's API (as shown in the example in **Section 3.1.3** in the lecture notes, use the correct station code) and put this in a list. Make sure you access the right endpoint for csv response! Note: documentation may be found here, under the heading 'Center City Regional Rail Arrivals (csv)'.

In [1]:
```
# A1:Function(2/3)
# Test:len(get_current_schedule(station_code = "ss")[0])
# Test:len(get_current_schedule(station_code = "30th")[0])

import re, requests, csv
from pprint import pprint
from datetime import datetime as dt

def get_current_schedule(station_code = '30th'):


    #---your code starts here---

    schedule_url = "".join(["http://www3.septa.org/ccstations/", station_code, "/sc

    #---your code stops here---

    access_time = dt.now()
    response = requests.get(schedule_url)


    return list(csv.reader(response.text.strip().split("\n"))), access_time
```

Depending on the time of day and the day of the week, your output could be:

```
(datetime.datetime(2021, 9, 25, 16, 11, 30, 259963),
 [["EMG=' No Emg Message"],
  ['R4S=04:55',
   'Airport',
   '3B',
   ' 2 LATE',
   'LOCAL                    ',
   '3449  ',
   '<_NEXT_MSG>05:55',
   'Airport',
   '3B',
   'ON TIME',
   'LOCAL                    ',
   '453   ',
   '<_NEXT_MSG>06:55',
   'Airport',
   '3B',
   'ON TIME',
   'LOCAL                    ',
```

```
   '3457  ',
   '<_NEXT_MSG>07:55',
   'Airport',
   '3B',
   'ON TIME',
   'LOCAL                    ',
   '461   ',
   ''],
  ['R4N=05:35',
   'Warminster',
   '2A',
   'ON TIME',
   'LOCAL                    ',
   '450   ',
   '<_NEXT_MSG>07:35',
   'Warminster',
   '2A',
   'ON TIME',
   'LOCAL                    ',
   '458   ',
   '<_NEXT_MSG>09:35',
   'Warminster',
   '2A',
   'ON TIME',
   'LOCAL                    ',
   '464   ',
   '<_NEXT_MSG>10:35',
   'Warminster',
   '2A',
   'ON TIME',
   'LOCAL                    ',
   '468   ',
   '']],
 '...',
 [['SERVICE=Effective Sunday September 5 new schedules will be in
effect for most lines. See SEPTA.org for more information'],
  ['TIMESTAMP=09/25/2021 16:11:21 PM']])
```

In [2]:
```
# A1:SanityCheck

schedule, access_time = get_current_schedule(station_code = "ss")
access_time, schedule[0:3], "...", schedule[-2:]
```

Out[2]:
```
(datetime.datetime(2023, 5, 22, 20, 40, 43, 682239),
 [["EMG=' No Emg Message"],
  ['R4S=08:55',
   'Airport',
   '3B',
   ' 1 LATE',
   'LOCAL                    ',
   '8467  ',
   '<_NEXT_MSG>09:25',
   'Airport',
   '3B',
   'ON TIME',
   'LOCAL                    ',
   '469   ',
   '<_NEXT_MSG>09:55',
   'Airport',
   '3B',
   'ON TIME',
   'LOCAL                    ',
   '8471  ',
   '<_NEXT_MSG>10:25',
   'Airport',
   '3B',
   'ON TIME',
   'LOCAL                    ',
   '473   ',
   ''],
  ['R4N=09:05',
   'Warminster',
   '2A',
   'ON TIME',
   'LOCAL                    ',
   '464   ',
   '<_NEXT_MSG>10:05',
   'Warminster',
   '2A',
   'ON TIME',
   'LOCAL                    ',
   '468   ',
   '<_NEXT_MSG>11:05',
   'Warminster',
   '2A',
   'ON TIME',
   'LOCAL                    ',
   '472   ',
   '<_NEXT_MSG>12:05',
   'Temple U',
   '2A',
   'ON TIME',
   'LOCAL                    ',
   '9476  ',
   '']],
 '...',
 [['SERVICE=No Smoking in SEPTA Stations or Platforms'],
  ['TIMESTAMP=05/22/2023 20:40:41 PM']])
```

Now review the data, is there a single column devoted to *all* of each type of data, e.g., a single timestamps column?

In [3]:
```
# A1:Inline(1/3)

# Are all timestamps contained in a single column?
# Print "Yes" or "No"
print("No")
```

```
No
```

A2. *(8 points)* Next, your job is is to complete the function to pre-process data from **A1** into a three-column format, as a list (rows) of lists (columns).

In particular, extract three pieces of information for each train: its scheduled arrival time, destination, and its lateness/timeliness status. Store these in a list that looks like the following.

```
[[<scheduled time>, <destination>, <on-time status>],...
 [<scheduled time>, <destination>, <on-time status>]]
```

[**HINT:** Regular expressions can extract the times. Each train-line is on a separate newline, and a variable number of train information is reported on each line. Consider using the modulus operator (`%`), which provides the remainder when one number is divided by another: `remainder = numerator % denominator`. Each of the variable number of trains takes up a fixed number of columns.]

In [4]:
```
# A2:Function(6/8)
# Test:sum([len(z) for z in get_trains(schedule)])

def get_trains(schedule):
    trains = []

    #---your code starts here---

    for line in schedule:
        if len(line) > 1:
            for n, field in enumerate(line):
                if not n % 6 and re.search("\d+:\d+", field):
                    time = re.findall("\d+:\d+", field)[0]
                    dest = line[n + 1]
                    lateness = line[n +3]
                    train = [time, dest, lateness]
                    trains.append(train)

    #---your code stops here---

    return trains
```

Depending on what time's `schedule` you currently have stored in your active workspace, your output could be:

```
[['04:55', 'Airport', ' 2 LATE'],
 ['05:55', 'Airport', 'ON TIME'],
 ['06:55', 'Airport', 'ON TIME'],
 ['07:55', 'Airport', 'ON TIME'],
 ['05:35', 'Warminster', 'ON TIME'],
 ['07:35', 'Warminster', 'ON TIME'],
 ['09:35', 'Warminster', 'ON TIME'],
 ['10:35', 'Warminster', 'ON TIME'],
 ['05:35', 'Wilmington', 'ON TIME'],
 ['07:35', 'Wilmington', 'ON TIME']]
```

In [5]:
```
# A2:SanityCheck

trains = get_trains(schedule)
trains[:10]
```

Out[5]:
```
[['08:55', 'Airport', ' 1 LATE'],
 ['09:25', 'Airport', 'ON TIME'],
 ['09:55', 'Airport', 'ON TIME'],
 ['10:25', 'Airport', 'ON TIME'],
 ['09:05', 'Warminster', 'ON TIME'],
 ['10:05', 'Warminster', 'ON TIME'],
 ['11:05', 'Warminster', 'ON TIME'],
 ['12:05', 'Temple U', 'ON TIME'],
 ['10:35', 'Wilmington', 'ON TIME'],
 ['09:21', 'West Trenton', ' 4 LATE']]
```

Does the format use 12- or 24-hour time?

In [6]:
```
# A2:Inline(2/8)

# Does the format use 12- or 24-hour time?
# Print "12" or "24"

print("12")
```

```
12
```

A3. *(2 points)* Now complete the time parsing function which takes the `trains` output from **A2** and parses its timestamp column using the `dateutil.parser` module-function. The three values (now with timestamp parsed) should then be output as a new list, which is sorted according to arrival time.

In [7]:
```
# !pip install py-dateutil
```

In [8]:
```
# A3:Function(2/2)
# Test:parse_times(trains)[0][0]

import dateutil.parser

def parse_times(trains):

    datetime_parsed_trains = []
```

```
    #---your code starts here---
    for train in trains:
        timestamp = train[0]
        datetime_obj = dateutil.parser.parse(timestamp)
        datetime_parsed_trains.append([datetime_obj] + train[1:])

    #---your code stops here---

    return sorted(datetime_parsed_trains, key = lambda x: x[0])
```

For reference, your output could be:

```
[[datetime.datetime(2021, 9, 25, 4, 35), 'West Trenton', 'ON
TIME'],
 [datetime.datetime(2021, 9, 25, 4, 35), 'West Trenton', 'ON
TIME'],
 [datetime.datetime(2021, 9, 25, 4, 45), 'Lansdale', 'ON TIME'],
 [datetime.datetime(2021, 9, 25, 4, 45), 'Thorndale', ' 2 LATE'],
 [datetime.datetime(2021, 9, 25, 4, 45), 'Lansdale', 'ON TIME'],
 [datetime.datetime(2021, 9, 25, 4, 45), 'Thorndale', ' 2 LATE'],
 [datetime.datetime(2021, 9, 25, 4, 50), 'Elwyn', 'ON TIME'],
 [datetime.datetime(2021, 9, 25, 4, 50), 'Elwyn', 'ON TIME'],
 [datetime.datetime(2021, 9, 25, 4, 55), 'Airport', ' 2 LATE'],
 [datetime.datetime(2021, 9, 25, 4, 55), 'Airport', ' 2 LATE']]
```

In [9]:
```
# A3:SanityCheck

datetime_parsed_trains = parse_times(trains)
datetime_parsed_trains[:10]
```

Out[9]:
```
[[datetime.datetime(2023, 5, 22, 8, 40), 'Trenton', ' 4 LATE'],
 [datetime.datetime(2023, 5, 22, 8, 40), 'Trenton', ' 4 LATE'],
 [datetime.datetime(2023, 5, 22, 8, 48), 'Chestnut H West', ' 3 LATE'],
 [datetime.datetime(2023, 5, 22, 8, 48), 'Chestnut H West', ' 3 LATE'],
 [datetime.datetime(2023, 5, 22, 8, 50), 'Temple U', 'ON TIME'],
 [datetime.datetime(2023, 5, 22, 8, 50), 'Temple U', 'ON TIME'],
 [datetime.datetime(2023, 5, 22, 8, 53), 'Chestnut H East', 'ON TIME'],
 [datetime.datetime(2023, 5, 22, 8, 53), 'Chestnut H East', 'ON TIME'],
 [datetime.datetime(2023, 5, 22, 8, 55), 'Airport', ' 1 LATE'],
 [datetime.datetime(2023, 5, 22, 8, 55), 'Airport', ' 1 LATE']]
```

A4. *(7 points)* If you haven't noticed by now, there's a problem—the arrival times are lacking AM/PM information, even though the data are listed in 12-hour time. This leads `dateutils.parser` to treat the 12-hour format timestrings as 24-hour format timestrings.

To solve this problem, utilize tools from the `datetime` module to 'fix' the original timestamps, and complete the `fix_times` function to process the original list created in **A2** and using the `datetime` module to infer AM/PM information, and hence, the precise dates/times. The function then should output these new arrival times, the destination, and lateness information as usual in a sorted list.

[**HINT:** Use the current system time and the fact that the schedule information only contains trains arriving in the next few hours to fix the AM/PM problem.]

MH: The solution for this part is included on purpose. There's nothing you need to change. Just review it for your own understanding.

In [10]:
```
# A4:Function(7/7)
# Test:parse_times(fix_times(trains, access_time))[0][0]

from datetime import timedelta, datetime

def fix_times(trains, access_time):

    trains_24_hour = []
    access_hour = access_time.hour ## 'fix' zero times
    if access_hour >= 12: ## now is PM
        access_hour -= 12
        current_am_or_pm = ["PM", "AM"]
    else: ## now is AM
        current_am_or_pm = ["AM", "PM"]
    access_date = access_time.strftime("%m/%d/%Y")
    tomorrow = format(datetime.now() + timedelta(days=1), '%m/%d/%Y')
    for train in trains:

        #---your code starts here---
        hour = int(train[0][:2])
        if hour<access_hour and current_am_or_pm[0]=="PM":
            access_date = tomorrow
        if hour<access_hour:
            trains_24_hour.append([access_date + " " + train[0] + current_am_or_pm[
        else:
            trains_24_hour.append([access_date + " " + train[0] + current_am_or_pm[
        #---your code stops here---

    return trains_24_hour
```

For reference, your output could be:

```
[[datetime.datetime(2021, 9, 25, 16, 35), 'West Trenton', 'ON
TIME'],
 [datetime.datetime(2021, 9, 25, 16, 35), 'West Trenton', 'ON
TIME'],
 [datetime.datetime(2021, 9, 25, 16, 45), 'Lansdale', 'ON TIME'],
 [datetime.datetime(2021, 9, 25, 16, 45), 'Thorndale', ' 2 LATE'],
 [datetime.datetime(2021, 9, 25, 16, 45), 'Lansdale', 'ON TIME'],
 [datetime.datetime(2021, 9, 25, 16, 45), 'Thorndale', ' 2 LATE'],
 [datetime.datetime(2021, 9, 25, 16, 50), 'Elwyn', 'ON TIME'],
 [datetime.datetime(2021, 9, 25, 16, 50), 'Elwyn', 'ON TIME'],
 [datetime.datetime(2021, 9, 25, 16, 55), 'Airport', ' 2 LATE'],
 [datetime.datetime(2021, 9, 25, 16, 55), 'Airport', ' 2 LATE']]
```

In [11]:
```
# A4:SanityCheck
datetime_parsed_trains_24_hour = parse_times(fix_times(trains, access_time))
```

```
datetime_parsed_trains_24_hour[:10]
```

Out[11]:
```
[[datetime.datetime(2023, 5, 22, 12, 5), 'Temple U', 'ON TIME'],
 [datetime.datetime(2023, 5, 22, 20, 40), 'Trenton', ' 4 LATE'],
 [datetime.datetime(2023, 5, 22, 20, 40), 'Trenton', ' 4 LATE'],
 [datetime.datetime(2023, 5, 22, 20, 48), 'Chestnut H West', ' 3 LATE'],
 [datetime.datetime(2023, 5, 22, 20, 48), 'Chestnut H West', ' 3 LATE'],
 [datetime.datetime(2023, 5, 22, 20, 50), 'Temple U', 'ON TIME'],
 [datetime.datetime(2023, 5, 22, 20, 50), 'Temple U', 'ON TIME'],
 [datetime.datetime(2023, 5, 22, 20, 53), 'Chestnut H East', 'ON TIME'],
 [datetime.datetime(2023, 5, 22, 20, 53), 'Chestnut H East', 'ON TIME'],
 [datetime.datetime(2023, 5, 22, 20, 55), 'Airport', ' 1 LATE']]
```

A5. *(5 points)* Finally, complete the function to create hourly log files with train information in `"data/trains/%Y-%m%-d%-H%.txt"` named with the appropriate timestamp containing date and hour, so that when sorted by name, they are also sorted chronologically. The files should contain the 24-hour format arrival time, destination, and lateness for trains scheduled to arrive in that hour, with one train per line.

For example, some of the lines from a log file for 7 PM could look like this:

```
19:02, Trenton, ON TIME
19:09, Norristown, ON TIME
19:35, Warminster, ON TIME
19:35, Wilmington, ON TIME
```

In [12]:
```
# A5:Function(5/5)
# Test:save_schedule(datetime_parsed_trains_24_hour)

import os

def save_schedule(datetime_parsed_trains_24_hour):

    ## Note: this uses the os module to execute a command line
    ## but the (bash) command could be run just once from the command line
    os.system("mkdir -p data/trains/")

    for train in datetime_parsed_trains_24_hour:
        timestamp = train[0].strftime("%Y-%m-%d-%H")
        filename = f"data/trains/{timestamp}.txt"

        with open(filename, "a") as file:
            file.write(f"{train[0].strftime('%H:%M')}, {train[1]}, {train[2]}\n")

    sorted_files = sorted(os.listdir("data/trains/"))
    return sorted_files

    #---your code stops here---
```

For reference, your output could be:

```
['2021-09-25-23.txt',
 '2021-09-25-22.txt',
```

```
 '2021-09-25-20.txt',
 '2021-09-25-21.txt',
 '2021-09-25-19.txt',
 '2021-09-25-18.txt',
 '2021-09-25-16.txt',
 '2021-09-25-17.txt']
```

In [13]:
```
# A5:SanityCheck

save_schedule(datetime_parsed_trains_24_hour)
[x for x in os.listdir("data/trains/")
 if re.search(datetime_parsed_trains_24_hour[0][0].strftime("%Y-%m-%d-\d\d.txt"), x
```

Out[13]:
```
['2023-05-22-12.txt',
 '2023-05-22-20.txt',
 '2023-05-22-21.txt',
 '2023-05-22-22.txt',
 '2023-05-22-23.txt']
```

In [ ]:
```

```