51. Extend discover\_violations

You have now finished the final optional activity. However, if you run the application, you will discover that it still only counts weather and endorsement violations.

codio@mike-panther:~/workspace$ python3 auditor KITH-2017

125 violations found.

That is because you need to update discover\_violations to include inspection violations as well. This is a very simple change. Simply call list\_inspection\_violations and add this list to the other violations.

Make this change and run the test script to check your work. Since you made the change to the test script to include the inspections module, the test script will inform you if you did not do this correctly.

## 144 violations found. How many should be?

## list\_inspection\_violations(directory):

Returns the (annotated) list of flight lessons that violate inspection

    or repair requirements.

This function reads the data files in the given directory (the data files

    are all identified by the constants defined above in this module).

It loops through the list of flight lessons (in lessons.csv), identifying those takeoffs for which

(1) a plane has gone MORE than a year since its annual inspection,

(2) a plane has accrued OVER 100 hours of flight time since its last repair or inspection, and

(3) a plane is used for a lesson despite the repair logs claiming that it is in the shop for maintenance.

Note that a plane landing with exactly 100 hours used is not a violation.

Nor is a plane that has flown with 365 days since its last inspection.

This school likes to cut things close to safe money, but these are technically

not violations.

This function returns a list that contains a copy of each violating lesson,

    together with the violation appended to the lesson.

Violation of type (1) is annotated 'Annual'.

Violation of type (2) is annotated 'Inspection'.

Violations of type (3) is annotated 'Grounded'.

If more than one is violated, it should be annotated 'Maintenance'.

Example: Suppose that the lessons

S00898  811AX  I072  2017-01-27T13:00:00-05:00  2017-01-27T15:00:00-05:00  VFR  Pattern

S00681  684TM  I072  2017-02-26T14:00:00-05:00  2017-02-26T17:00:00-05:00  VFR  Practice Area

S01031  738GG  I010  2017-03-19T13:00:00-04:00  2017-03-19T15:00:00-04:00  VFR  Pattern

violate for reasons of 'Annual', 'Inspection', and 'Grounded', respectively

    (and are the only violations).  Then this function will return the 2d list

[['S00898', '811AX', 'I072', '2017-01-27T13:00:00-05:00', '2017-01-27T15:00:00-05:00', 'VFR', 'Pattern', 'Annual'],

['S00681', '684TM', 'I072', '2017-02-26T14:00:00-05:00', '2017-02-26T17:00:00-05:00', 'VFR', 'Practice Area', 'Inspection'],

['S01031', '738GG', 'I010', '2017-03-19T13:00:00-04:00', '2017-03-19T15:00:00-04:00', 'VFR', 'Pattern', 'Grounded']]

    Parameter directory: The directory of files to audit

Precondition: directory is the name of a directory containing the files

    'daycycle.json', 'fleet.csv', 'repairs.csv' and 'lessons.csv'

codio@mike-panther:~/workspace/auditor$ python3

...

>>> import tests

>>> tests.test\_inspections()

Testing module inspections

codio@mike-panther:~/workspace/auditor$ python3

...

>>> import os.path

>>> dataset = os.path.join('..','KITH-2018')

>>> import inspections

>>> inspections.list\_inspection\_violations(dataset)

The module inspections is designed to search for a third class of violations, which includes planes that have not been inspected or repaired properly. A plane must be inspected annually. In addition, planes used in a commercial setting (which includes a flight school) must be inspected every 100 hours that they have flown. This function in this module searches for cases in which a plane is flown past its required (annual or 100 hour) inspection. It also includes violations where a plane is claimed to be down for repairs, but a student flies the plane while it is still in maintenance.

This module will make use of the last file in the dataset: repairs.csv. This file is a list of when each plane is in maintenance and the reason for this maintenance (inspection or repairs).

A plane is assumed to be unavailable between midnight of the in date and midnight of the out date.

1/5/2017~1/7/2017 : 1/5/2017 12:00 AM, 1/7/2017 12:00 AM ? most of **1/7/2017 available!**

Technically, inspection violations are associated with flights in the file lessons.csv. A plane can go over a year between annual inspections so long as no one flies it while it is past due for inspection. Therefore, to find inspection violations, you will need to compare lessons.csv with the file repairs.csv.

However, these two files are still not enough to track inspection violations. Each of our datasets only covers a years’ worth of flights and inspections/repairs. How can we be sure that a plane has had its annual inspection in time? This is the purpose of the ANNUAL column in fleet.csv. This is the date of the last annual inspection of each plane *before* either lessons.csv or repairs.csv begin. For example, if fleet.csv claims the last annual inspection of Cessna 133CZ was 2016-04-15, then there is an inspection violation if repairs.csv does not contain an annual inspection for this plane by 2017-04-15.

Similarly, the column HOURS in fleet.csv counts the number of hours flown since the inspection *before* either lessons.csv or repairs.csv begin. For example, if fleet.csv claims that Cessna 133CZ starts with 88 hours, then there should only be 12 hours’ worth of lessons before the next inspection in repairs.csv.

**IMPORTANT**: This flight school will often try to push things to the limit, so you must interpret the rules exactly. If a plane has exactly 100 hours before its next inspection, this is still okay. Likewise, if a plane has 365 days between annual inspections, this is still okay.

One of the hardest parts of this function is searching for 100 hour violations. You need to keep a running total of the number of hours each plane has flown, but reset this value to 0 whenever the plane undergoes inspection (100 hour or annual). If you search from the beginning of the repair log for each individual flight your function will be incredibly slow – taking hours to run. You need to be smarter about how to approach it.

As a hint, we recommend creating a dictionary to track the hours flown for each plane. 🡨this is lessons, Then combine the lessons and repairs together and sort them by date. When you loop through the combined list, you increment the hours if the next element is a lesson flight and reset hours to 0 if it is a repair. If you can figure out how to do this, finding other types of inspection violations should be no problem.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 133sz | Lesson | 2017-01-02T09:00:00-05:00 | 2017-01-02T11:00:00-05:00 | 2hr |
| 133sz | Repair | 1/5/2017 12:00 | 1/7/2017 24:00 | 0hr |
|  | Lesson |  |  |  |
|  | Lesson |  |  |  |

Dictionary and list

{“133sz”:[[repair, 2017,2017 ], [lesson, 2017, 2018], [lesson,2017, 2017]],

“122XF”:[[repair, 2017,2017 ], [lesson, 2017, 2018], [lesson,2017, 2017]],

bad\_ifr(takeoff,student,instructor,plane):

Returns:

True if the student, instructor, or plane is not certified for IFR.

For an IFR flight to be valid, the plane must be outfitted for IFR.

If there is an instructor, that instructor must have a CFII.

If the student is alone, the student must have an instrument rating at the time of takeoff.

NOTE: The precondition for takeoff does not assume anything about the flight.

It may be a VFR flight not subject to IFR rules.

This function should still return False if that flight COULD have been a successful IFR flight.

    Parameter takeoff: The takeoff time of this flight

    Precondition: takeoff is a datetime object

    Parameter student: The student pilot

    Precondition: student is 10-element list of strings representing a pilot

    Parameter instructor: The flight instructor

    Precondition: instructor is a 6-element list of strings representing an instructor

    Parameter plane: The school airplane

    Precondition: plane is a 7-element list of strings representing an airplane

-----------

bad\_endorsement(takeoff,student,instructor,plane):

Returns True if the student or instructor did not have the right endorsement.

The endorsement depends on the plane type (advanced, multiengine).

is\_advanced(plane):, is\_multiengine(plane):

All instructors are certified for advanced planes, so a flight with an instructor is only a problem if the plane is multiengine and the instructor does not have an MEI.

If not teaches\_multiengine(instructor) and is\_multiengine(plane):

Return True

If there is no instructor, the student must be endorsed for this type of

    plane before the time of takeoff.

Parameter

takeoff: The takeoff time of this flight

   a datetime object

student: The student pilot

   10-element list of strings representing a pilot

instructor: The flight instructor

   a 6-element list of strings representing an instructor

plane: The school airplane

   7-element list of strings representing an airplane

Parameter

directory: The directory of files to audit

  Precondition: directory is the name of a directory containing the files 'daycycle.json','weather.json', 'minimums.csv', 'students.csv', 'teachers.csv', 'lessons.csv', 'fleet.csv', and 'repairs.csv'.

output: The CSV file to store the results

    Precondition: output is None or a string that is a valid file name

On its surface discover\_violations looks just like list\_weather\_violations.

1, discover\_violations prints out the number of violations found to the screen.

2 if the argument output is the name of a CSV file, it writes all of the violations found to that file.

3, if you choose to do any of the **optional activities**, discover\_violations will eventually include the violations found there as well.

Writing the CSV file is easy; there is a function for that in utils.

print statements:

When writing the code for the print statements, pay close attention to the specification. Remember the period.

Notice what to do if no violations are found.

remember to use the singular “violation” if only one violation is found.

This function will call list\_weather\_violations() to get the list of weather violations.

If list\_endorsment\_violations (optional) is completed, it will call that too, as well as list\_inspection\_violations.

It will concatenate all of these 2d lists into a single 2d list of violations (so a flight may be listed more than once for each of the three types of violations).

    If the parameter output is not None,

it will create the CSV file with name output and write the 2d list of violations to this file.  This CSV file should have the following header:

        STUDENT,AIRPLANE,INSTRUCTOR,TAKEOFF,LANDING,FILED,AREA,REASON

    Regardless of whether output is None, this function will print out the number of

    violations, as follows:

        '23 violations found.'

    If no violations are found, it will say

        'No violations found.'

codio@mike-panther:~/workspace/auditor$ python3

...

>>> import tests

>>> tests.test\_app()

Testing module app (this may take a while)

codio@mike-panther:~/workspace$ python3 auditor KITH-2017

93 violations found.

codio@mike-panther:~/workspace/auditor$ python3

...

>>> import os.path

>>> dataset = os.path.join('..','KITH-2018')

>>> import violations

>>> violations.list\_weather\_violations(dataset)

# 31. Implement list\_weather\_violations

This function will return a table of all lessons that violate weather minimums. Each row of the table is the lesson data plus one additional column. This column will contain the results of the function get\_weather\_violation (but only if it is not the empty string).

Returns :

the **(annotated) list** of flight reservations that violate weather minimums.

Parameter:

directory: The directory of files to audit

    Precondition: directory is the name of a directory containing the files 'daycycle.json',  'weather.json', 'minimums.csv', 'students.csv', and 'lessons.csv'

This function reads the data files in the given directory (the data files are all identified by the constants defined above in this module).  It loops through the list of flight lessons (in lessons.csv), identifying those takeoffs for which get\_weather\_violation() is **not the empty string**.

The function list\_weather\_violations analyzes a *complete* dataset to find weather violations. This function is simply given a directory, nothing more.

The function then reads the files daycycle.json, students.csv,minimums.csv, weather.json, and lessons.csv to create a list of weather violations.

Remember how all of the other functions assumed that these files were read somewhere else and converted to tables or dictionaries as appropriate? That happens in this function.

This function will use *almost every function* you have written so far. To help you organize your thoughts, here is some pseudocode to help you approach the function

|  |
| --- |
| def read\_csv(filename):  read\_json(filename):  For each of the lessons - lessons.csv : STUDENT,AIRPLANE,INSTRUCTOR,TAKEOFF,LANDING,FILED,AREA  Get the takeoff time - lessons  Lessons[studentid][3]  Get the pilot credentials students.csv  Pilots.get\_certification(takeoff,student):  Get the pilot minimums minimums.csv  Vfr from lessons, daytime from util.daytime  Pilots.get\_minimums(cert, area, instructed, vfr, daytime, minimums):  Returns the most advantageous minimums  Get the weather conditions - weather.json  get\_weather\_report(takeoff,weather):  Returns the most recent weather report  Check for a violation and add it to the list if so   get\_weather\_violation(weather,minimums):  if <> ‘’ : list\_violation.append( row) |

This function **returns** a list that contains a copy of each violating lesson, together with the violation appended to the lesson.

    Example: Suppose that the lessons

S00687  548QR  I061  2017-01-08T14:00:00-05:00  2017-01-08T16:00:00-05:00  VFR  Pattern

S00758  548QR  I072  2017-01-08T09:00:00-05:00  2017-01-08T11:00:00-05:00  VFR  Pattern

S00971  426JQ  I072  2017-01-12T13:00:00-05:00  2017-01-12T15:00:00-05:00  VFR  Pattern

violate for reasons of 'Winds', 'Visibility', and 'Ceiling', respectively (and are the

**only violations**).  Then this function will **return the 2d list**

[

[S00687, 548QR, I061, 2017-01-08T14:00:00-05:00, 2017-01-08T16:00:00-05:00, VFR, Pattern, Winds],

[S00758, 548QR, I072, 2017-01-08T09:00:00-05:00, 2017-01-08T11:00:00-05:00, VFR, Pattern, Visibility],

[S00971, 426JQ, I072, 2017-01-12T13:00:00-05:00, 2017-01-12T15:00:00-05:00, VFR, Pattern, Ceiling]

]

REMEMBER: VFR flights are subject to minimums with VMC in the row while IFR flights are subject to minimums with IMC in the row.  The examples above are all VFR flights.

If we changed the second lesson to

|  |
| --- |
| S00758, 548QR, I072, 2017-01-08T09:00:00-05:00, 2017-01-08T11:00:00-05:00, IFR, Pattern |

then it is possible it is no longer a visibility violation because it is subject to a different set of minimums.

**HINT**: One thing that we have seen confuse students is the parameter vfr in get\_minimums. Even if a pilot is instrument-rated, the pilot should use the minimums corresponding to the *lesson*. So an instrument-rated pilot on a VFR flight is subject to VMC minimums. Whether or not a lesson is VFR is indicated by the **FILED** column of lessons.csv.

29. Implement get\_weather\_violation

Returns : a string representing the type of weather violation (empty string if flight is ok)

Parameters:

weather:

The weather measure

Precondition: weather is **dictionary** containing a visibility, wind, and ceiling measurement, or None if no weather reading is available.

minimums:

The safety minimums for ceiling, visibility, wind, and crosswind

Precondition: minimums is a **list** of four floats

    The weather reading is a dictionary with the keys: 'visibility', 'wind', and 'sky'.

    These correspond to a visibility, wind, and ceiling measurement, respectively. It may have other keys as well, but these can be ignored. For example, this is a possible

    weather value:

        {

            "visibility": {

                "prevailing": 21120.0,

                "minimum": 1400.0,

                "maximum": 21120.0,

                "units": "FT"

            },

            "wind": {

                "speed": 12.0,

                "crosswind": 3.0,

                "gusts": 18.0,

                "units": "KT"

            },

            "temperature": {

                "value": -15.6,

                "units": "C"

            },

            "sky": [

                {

                    "cover": "clouds",

                    "type": "broken",

                    "height": 2100.0,

                    "units": "FT"

                }

            ],

            "weather": [

                "light snow"

            ]

        }

    The minimums is a list of the four minimums ceiling, visibility, and max windspeed, and max crosswind speed in that order.  Ceiling is in feet, visibility is in statute miles, max wind and cross wind speed are both in knots. For example, [3000.0,10.0,20.0,8.0] is a potential minimums list.

  This function uses bad\_visibility, bad\_winds, and bad\_ceiling as helpers. It returns

'Visibility' if the only problem is bad visibility,

'Winds' if the only problem is wind, and

'Ceiling' if the only problem is the ceiling.

'Weather' If there are multiple problems, it returns 'Weather',

'Unknown' It returns if no weather reading is available (e.g. weather is None).

'' : Finally, it returns '' (the empty string) if the weather is fine and there are no violations.

codio@mike-panther:~/workspace/auditor$ python3 tests

python3

...

>>> import tests

>>> tests.test\_violations()

A screenshot of a computer

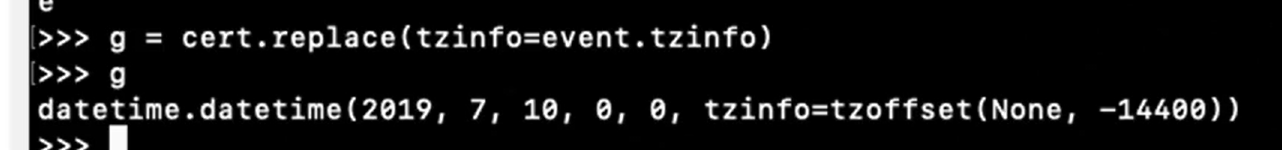
Description automatically generated with medium confidence

A screen shot of a computer

Description automatically generated with medium confidence

A screen shot of a computer

Description automatically generated with medium confidence



A screen shot of a computer

Description automatically generated with low confidence

A screenshot of a computer program

Description automatically generated with medium confidence

A screen shot of a computer code

Description automatically generated with low confidenceA screen shot of a computer

Description automatically generated with low confidence

A screen shot of a computer program

Description automatically generated with low confidence

A screen shot of a computer program

Description automatically generated with low confidence A screen shot of a computer

Description automatically generated with low confidence

A picture containing text, screenshot, font

Description automatically generated 0~mill

A screen shot of a computer code

Description automatically generated with low confidence

A computer code on a black background

Description automatically generated with low confidence

A picture containing text, screenshot, software, computer

Description automatically generated

A computer screen with white text

Description automatically generated with low confidence

|  |
| --- |
| import csv  def write\_csv(data,filename):      """      Writes the given data out as a CSV file filename.          file = open( filename, 'w')      wrapper = csv.writer(file)      for row in data :          wrapper.writerow(row)        file.close() |

A computer screen with white text

Description automatically generated with low confidence

A screen shot of a computer

Description automatically generated with medium confidence

A screen shot of a computer

Description automatically generated with medium confidence